KND los chicos del K1

Ingeniería de Software

2021

Índice

Unidad 1: Componentes de proyecto de desarrollo de Software

El presente tema es una introducción a lo que sería la gestión de proyectos de software basados en procesos definidos, es decir, lo que normalmente referimos como ***gestión tradicional de proyectos***.

El proceso de Software

En términos generales, el proceso que nos interesa estudiar es el proceso que transforma ideas o necesidades en un producto de software específicamente. Esencialmente, se apunta a entregar un producto de software de calidad al cliente.

El proceso agrupa un conjunto de actividades que tienen un objetivo. En este caso, ese objetivo es entregar un producto de software. Además, utilizamos recursos y sumamos personas para que realicen este trabajo y finalmente poder obtener el resultado deseado.

Entonces, dicho de manera más formal, el proceso de software es un conjunto estructurado de actividades para desarrollar un sistema de software. Estas actividades varían dependiendo de la organización y el tipo de sistema que debe desarrollarse por lo que debe ser explícitamente modelado si va a ser administrado.

Definición de un proceso de software

Antes de comenzar dejemos explicita la diferencia entre dos conceptos:
Proceso: La secuencia de pasos ejecutados para un propósito dado (IEEE).
Proceso de Software: Un conjunto de actividades, métodos, prácticas, y transformaciones que la gente usa para desarrollar o mantener software y sus productos asociados (Sw-CMM).

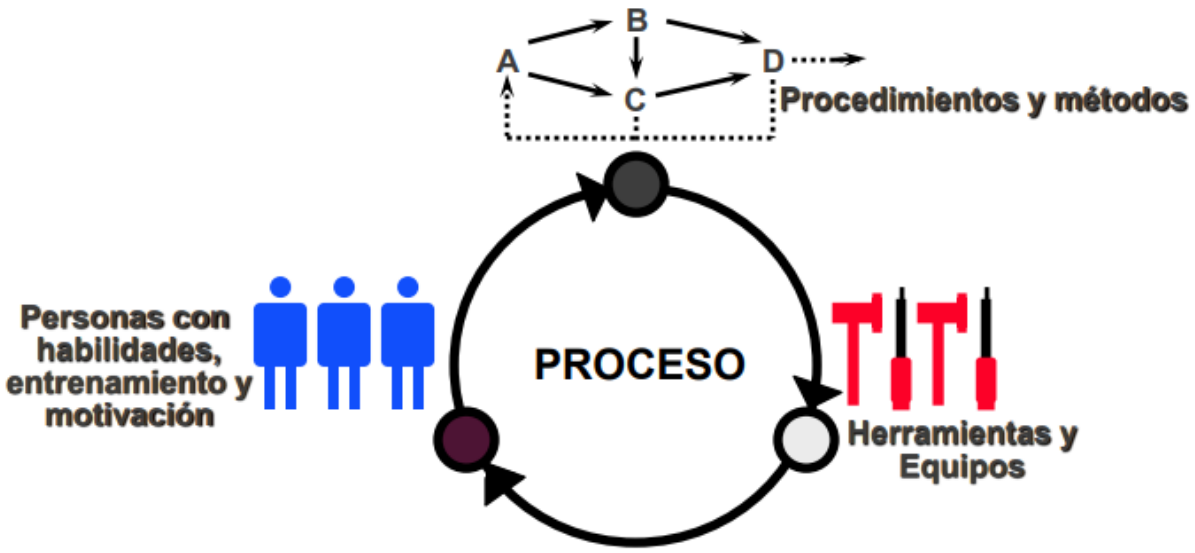
El ingeniero en Sistemas de Información trabaja en una industria que es ***humana intensiva***. Este término se refiere al hecho de que al software lo hacen personas y el aporte más importante para que el producto llegue a las manos de los usuarios interesados y los

clientes es el aporte que hacen las personas. Incluso lo más caro en términos de costos es pagarle a las personas que trabajan haciendo software (la mano de obra).
Es decir, que en definitiva, ***lo más importante son las personas***.

En la actualidad (año 2021) el 80% del costo de un producto de software es el costo del esfuerzo, es decir, el costo del trabajo de la gente que hace software. Años atrás, este costo era del 90%.

¿Por qué bajó el costo del recurso humano? Porque hoy tenemos una incidencia muy fuerte en el costo del producto sobre la conectividad. La conectividad ha sumado este 10% de costo en la determinación del costo del producto, es decir, el acceso a la nube, a procesamiento, a internet, toda esta clase de recursos han permitido que el costo del recurso humano se redujera considerablemente.
Entonces, el 80% del costo de un producto es el costo del esfuerzo (las personas que elaboran el producto), otro 10% del costo de un producto corresponde a conectividad y el 10% restante corresponde a costos menores tales como alquiler, luz, equipos electrónicos (computadoras),etc.

El foque “La mesa de tres patas”



Procedimientos y métodos: es la documentación del proceso, es decir, es tener en algún lado escrito lo que queremos hacer, cuáles son las tareas que vamos a llevar adelante y cómo vamos a medir las mismas.

Personas con habilidades, entrenamiento y motivación: las personas involucradas en el proceso deben estar capacitadas, estar formadas y poseer conocimiento técnico para realizar su trabajo. Estas personas además deben estar motivadas, tener ganas de trabajar.

Herramientas y equipos: son los materiales que se necesitan para llevar a cabo el proceso, es decir, son las herramientas de soporte que nos ayudan a automatizar lo máximo que se pueda el proceso de hacer software.

De estos tres ítems mencionados anteriormente, ***el más importante de todos son las personas***, aunque en realidad todos son importantes pero nos enfocamos en aquello que no puede ser reemplazado ni automatizado.
El enfoque de “La mesa de tres patas” enuncia que todo es importante para lograr el equilibrio necesario para que un proceso puesto a funcionar en una organización nos permita obtener un producto de software de calidad. Es decir, tanto las personas como las herramientas y los procedimientos son importantes.

Tipos de procesos

Dentro de la industria existen dos tipos de procesos: ***empíricos y definidos***.

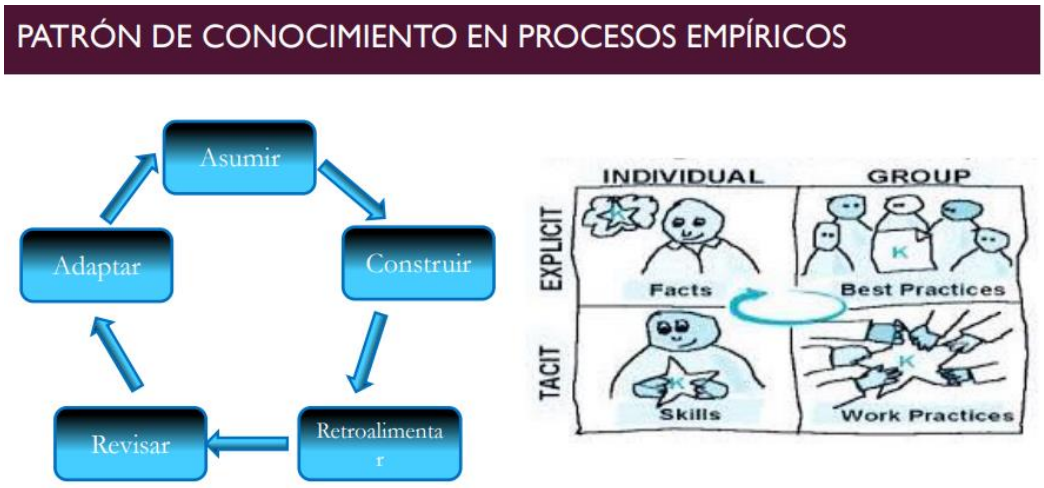
Procesos Empíricos

Son procesos que tienen un fuerte arraigo en la experiencia. Esto mismo, es complejo de explicar, ya que la experiencia se basa en algo anterior para hacer algo nuevo. La experiencia es algo que se adquiere con el paso del tiempo, no es algo que uno pueda comprar.
La experiencia se gana sobre la base de la propia realimentación del equipo.
Los procesos empíricos surgen en contraposición de los procesos definidos.

Ejemplo de proceso empírico o también llamado framework: scrum (framework de trabajo para la gestión de productos y proyectos de cualquier cosa)

Algunas características:

- Asume procesos complicados con variables cambiantes.
- Cuando se repite el proceso, se pueden llegar a obtener resultados diferentes.
- La administración y control se realiza través de inspecciones frecuentes y adaptaciones
- Son procesos que trabajan bien con procesos creativos y complejos.



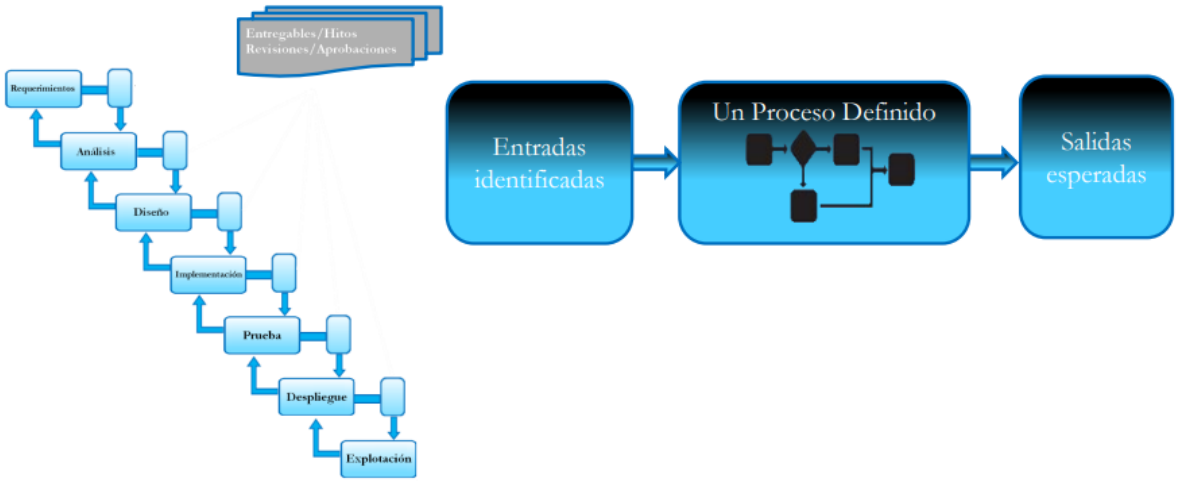
Procesos definidos

Estos procesos están inspirados en las líneas de producción.
Son procesos que plantean la necesidad de tener definidos ampliamente paso a paso que se va a realizar en cada momento. Así, se desagrega un conjunto de actividades y se define de manera explícita cuáles son las tareas, quién va a realizar cada tarea y en qué momento, cuáles son las entradas y salidas de cada tarea y qué artefactos son generados.

Este tipo de proceso asume que podemos **repetir** el mismo proceso una y otra vez, indefinidamente, y obtener los mismos resultados.
La administración y control provienen de la predictibilidad del proceso definido.

Un ejemplo de este tipo de proceso es el ***PUD*** (proceso unificado de desarrollo) junto con su evolución RUP (rational unified process).

En términos generales, para hacer software los pasos están bien definidos (requerimientos, análisis y diseño, programar, probar, hacer despliegue). La cuestión es cómo encaramos estas actividades y el nivel de definición que tenemos anticipado sobre las mismas.



¿Qué distingue esencialmente a los dos tipos de procesos?

La diferencia fundamental entre estos dos tipos de procesos ***es la intención de cada uno***.

Los ***procesos definidos*** buscan ser ***repetibles*** para lograr tener visibilidad. Es decir, al saber qué pasos realizar y que entra y sale en todo momento, las personas involucradas en el proceso saben qué es lo que pueden esperar a cada momento de tiempo y pueden predecir qué es lo que va a suceder.
Las personas involucradas en el proceso definido quieren tener la ilusión de control sobre el proceso. Claro está que usamos la palabra “ilusión” porque las cosas no siempre salen como uno esperaba, las personas pueden realizar determinada actividad de distintas formas y por ende el resultado o los problemas que se presenten a lo largo del proceso van a variar, pero al menos se busca tener una serie de pasos bien definidos que nos permitan saber cómo proseguir.
Además, los procesos definidos ***intentan ser procesos completos*** ya que describen la mayor cantidad de cosas posibles que se deben hacer para obtener un producto.

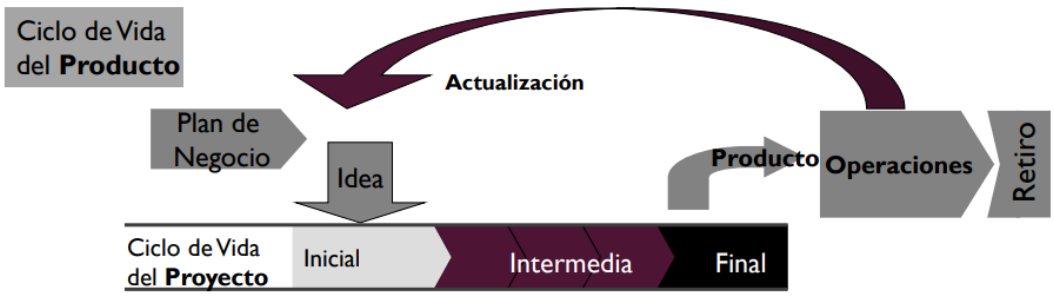
Los procesos definidos son ***aplicables a cualquier equipo*** en cualquier contexto, esto es a lo que nos referimos anteriormente cuando se hablaba de repetibilidad.

Por otro lado, los ***procesos empíricos no son repetibles ni son procesos completos***. En el mercado, existen muchos frameworks que son lineamientos, guías o heurísticas, es decir, son mejores prácticas que cubren algún aspecto en particular. El proceso empírico no cubre jamás la totalidad de tareas que se deben realizar, y es gracias a esta característica que cada equipo cuando inicia un trabajo (suponiendo que estamos en un contexto de proyecto) decide basado en la experiencia qué es lo que quieren hacer, cómo lo van a realizar y cuándo lo van a llevar a cabo.
Así, el ***empirismo se basa en ciclos de entrega cortos*** para poder generar realimentación que sirva como experiencia para evolucionar y seguir avanzando.
Estos procesos ***trabajan basados en una hipótesis***, asumen que se puede funcionar de cierta manera, se realiza la construcción (es decir, hacen algo) para obtener realimentación y revisar. Luego, si algo no salió bien, se adaptan y arrancan el ciclo nuevamente.
Además, estos procesos postulan que ***la experiencia es aplicable sólo a un equipo***, ya que la experiencia no se puede extrapolar a otros equipos en un contexto totalmente distinto.

Diferencia entre proceso y ciclo de vida

Un proceso tiene un ciclo de vida. Es decir, uno elige para un determinado proyecto un ciclo de vida para usar el proceso.
El ***proceso*** es el que define paso a paso lo que tenemos que hacer para obtener un objetivo.
El ***ciclo de vida*** para el desarrollo de un producto (es decir, el ciclo de vida del proyecto) define cómo se van a encarar las actividades que plantea el proceso, en qué orden, cuánto de cada actividad se va a realizar. El ciclo de vida es la serie de pasos a través de los cuales el producto o proyecto progresa. Ejemplos del ciclo de vida: en cascada (que consiste en hacer la etapa completa antes de pasar a la siguiente), otro ejemplo es el iterativo e incremental.

Relación: ciclo de vida del proyecto y del producto



Existen ***ciclos de vida para los productos*** y ***ciclos de vida para los proyectos.*** Los ciclos de vida iterativo e incremental o en cascada son ciclos de vida para los proyectos.
Un ciclo de vida de un producto inicia con la idea de crear un producto de software y termina cuando ese producto de software es retirado del mercado.

En un ciclo de vida de un producto puede haber n ciclos de vida de proyectos. Cada proyecto genera una versión (release) distinta del producto.

Un ***ciclo de vida de un proyecto software*** es una representación de un proceso. Gráfica una descripción del proceso desde una perspectiva particular. Los modelos especifican las fases de proceso y el orden en el cual se llevan a cabo.

Un software funcionando siempre va a requerir cambios y esa necesidad de cambios es el disparador para definir requerimientos para un nuevo proyecto.

Clasificación de los ciclos de vida

Hay tres tipos básicos de Ciclos de Vida para un proyecto de desarrollo de software:

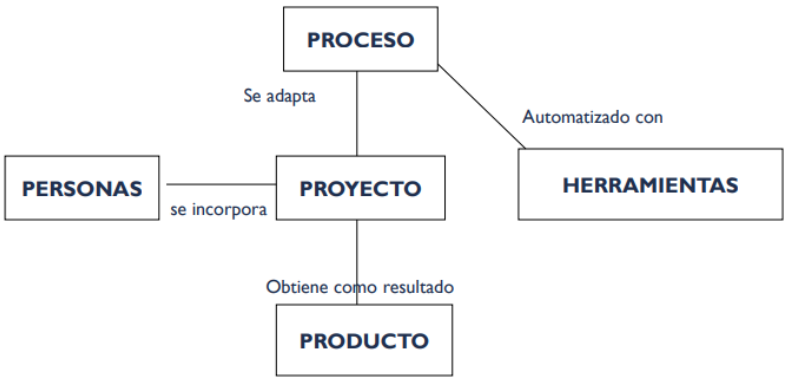
- ***Secuencial***: un ejemplo de este tipo es el de “en cascada”.
- ***Iterativo/Incremental***
- ***Recursivo:*** un ejemplo de este tipo es el de “espiral”.

En los ***procesos definidos*** podemos elegir cualquier ciclo de vida. Por otra parte, los ***procesos empíricos funcionan con el ciclo de vida iterativo e incremental***.

¿Por qué NO USAMOS el ciclo de vida recursivo en los procesos empíricos? Porque además de que fue un fracaso, este tipo de ciclo de vida no genera versiones intermedias del producto. Solo generamos una versión del producto que se puede poner en producción recién al final y por ende, perdemos la oportunidad de una realimentación real.
Jamás se debe mezclar scrum con cascada (secuencial).

Las 4 p

La disciplina de ingeniería de software se mueve básicamente en tres dimensiones (proceso, proyecto y producto) pero la profe agrega también a las personas (debido a que las personas también son muy importantes y un factor clave para el éxito cuando por ejemplo realizan la licitación de requerimientos).



Explicación de la imagen: las personas están involucradas en el proyecto, el proyecto es la forma de organizar el trabajo. El proceso es el conjunto de actividades que tienen una

entrada y una salida. Así, los requerimientos se convierten en productos, es decir, el resultado del proyecto es un producto.
Las actividades del proceso utilizan herramientas para poder ser automatizadas.
El proceso es el nivel más abstracto porque es donde está escrito en términos teóricos qué es lo que se debería hacer para hacer software. Ese proceso es el que, al momento de iniciar un proyecto se debe adaptar (en el caso que sea un proceso definido) o terminar de armar (en el caso que sea un proceso empírico) con la gente que va a estar involucrada. El proceso se define en términos de roles porque para cada proyecto las personas asumen uno o más roles.
El proceso es teórico y se instancia en el proyecto para tener un lineamiento de cada persona, que rol y qué responsabilidad tiene y qué tareas va a tener asignadas para ir obteniendo ese producto como resultado.
Cuando el proceso se adapta al proyecto es cuando se define el ciclo de vida.
Las tareas se ejecutan concretamente en el ámbito del proyecto.

La ingeniería de software se ocupa del producto como objetivo final, es decir, la motivación fundamental es entregar un producto de software de calidad a los clientes.
El proyecto es el medio por el cual se obtiene un producto de software, entonces, el proyecto es la unidad organizativa o la unidad de gestión de la disciplina.
El proyecto para poder existir necesita de una serie de elementos (personas, recursos y una indicación de cómo hacer el trabajo que hay que hacer para cumplir con el objetivo que ese proyecto se plantea). Es entonces donde entra el proceso, ya que es el proceso quien da una definición teórica de qué es lo que debería hacerse para hacer software y así, mediante ciertos criterios se escogen las tareas necesarias y esas tareas conforman lo que se llama ***"alcance del proyecto"*** .
El alcance del proyecto se define como el trabajo que hay que hacer para cumplir con el objetivo del proyecto.

En resumen, el proceso es un conjunto de actividades que toman como entradas requerimientos y obtienen como salida un producto o servicio (es este caso es el software de calidad). Ese conjunto de actividades están estructuradas y guiadas por un objetivo. El proceso es una teoría. Por otro lado, el proyecto es una unidad de gestión, un medio por el cual uno administra los recursos que necesita y las personas que van a estar integradas y trabajando son medios de gestión, de organización para obtener como resultado un producto o servicio (nuevamente, en nuestro caso es el software de calidad).
El proyecto empieza decidiendo y eligiendo la gente que va a sumarse a trabajar con los distintos roles en el proyecto. Por eso se dice que el proyecto "incorpora" personas.
Los roles de estas personas están definidos en el proceso.

En un proyecto que genera producto, cada tarea del proyecto tiene que tener un entregable porque si no, no sabemos si la tarea se hizo o no se hizo.

En la materia nos enfocamos en el ciclo de vida del proyecto.

Aclaración: una persona puede asumir más de un rol en un proyecto. Incluso puede haber un rol que sea asumido por muchas personas (ejemplo: ingeniero de componentes).

¿Qué es software? Es conocimiento empaquetado a distintos niveles de abstracción, no solamente un programa andando en una computadora. El producto de software que se le entrega al cliente incluye todo, no solo código funcionando, si no también la documentación, todos los diagramas, los requerimientos, los manuales para el usuario, etc. Todo lo que sale como resultado de la ejecución de las tareas de un proyecto es software.

Proyecto

Características

- ***Orientación a objetivos***:
    - ❖ Los proyectos están dirigidos a obtener resultados y ello se refleja a través de objetivos.
    - ❖ Los objetivos guían al proyecto
    - ❖ Los objetivos no deben ser ambiguos
    - ❖ Un objetivo debe ser ***claro*** y ***alcanzable***.
    - ❖ El proyecto tiene como resultado un producto.
    - ❖ El objetivo del proyecto que puede ser un producto o un servicio (o sea, aquello que quiero obtener) y es un objetivo único. Cada versión de un producto de software es única, porque son distintos proyectos.
- ***Duración limitada***:
    - ❖ Los proyectos son temporarios, cuando se alcanza el/los objetivo/s, el proyecto termina.
    - ❖ Una línea de producción no es un proyecto.
    - ❖ Los proyectos tienen una fecha de inicio y una fecha de fin.
- ***Tareas interrelacionadas basadas en esfuerzos y recursos***:
    - ❖ Complejidad sistémica de los problemas.
    - ❖ Se divide todo el trabajo en tareas más simples que se relacionan entre sí.
    - ❖ La interrelación hace referencia a que hay tareas que se pueden realizar en paralelo, otras no. O al hecho de que para empezar determinada tarea debo haber terminado otra.
    - ❖ La interrelación genera dependencias inevitables.
    - ❖ La definición de las tareas se obtiene del proceso.
- ***Son únicos***:
    - ❖ Todos los proyectos por similares que sean tienen características que los hacen únicos.

Existe objetivo del proyecto y objetivo del producto. El ***objetivo del proyecto*** define el trabajo que el proyecto tiene que hacer para dejar contento al cliente y a los demás involucrados.
Cuando un proyecto termina se liberan los recursos. *Ejemplo*: la persona que tenía el rol de analista en un proyecto, al finalizar ese proyecto se mete a otro proyecto con otro rol.

Un proyecto debe planificarse. “Lo importante de la planificación es el acto de planificar, no el resultado”.

Un proyecto debe estimarse, debe identificarse cuáles son los posibles riesgos, cuál es el objetivo y todo aquello que involucre planificar porque es gracias a este ejercicio de pensar estas cosas la razón por la cual aprendemos. Planificar no es precisamente documentar, en realidad, el hecho de documentar es algo que cada equipo de un proyecto debe decidir, es decir, se decide cuales son los aspectos más importantes que es necesario dejar escritos y explícitos para que todos tengan la misma idea acerca del proyecto.
Lo importante es lo que uno aprende en el proceso, no el resultado.

¿Qué es la administración de proyectos?

- "….tener el trabajo hecho…." en tiempo, con el presupuesto acordado y habiendo satisfecho las especificaciones o requerimientos.
- Más académicamente, la administración de proyectos es la aplicación de conocimientos, habilidades, herramientas y técnicas a las actividades del proyecto para satisfacer los requerimientos del proyecto.
- Administrar un proyecto incluye:
    - Identificar los requerimientos
    - Establecer objetivos claros y alcanzables
    - Adaptar las especificaciones, planes y el enfoque a los diferentes intereses de los involucrados (stakeholders).
- Como se quiere que el proyecto cumpla con el objetivo exitosamente, se debe administrar todos los recursos, organizar el trabajo de la gente que está involucrada y hacer un seguimiento de si las cosas se están dando como se esperaba o no.
- Todas estas actividades que nos permiten tener un seguimiento del proyecto son las que nos permiten administrar el proyecto.
- El ***líder de proyecto*** o jefe de proyecto es el que se encarga de realizar este trabajo.
- La ***administración de proyectos tiene dos grandes actividades incluidas*** dentro de la administración de proyectos:
    - ***planificación***
    - ***seguimiento y control*** de que las cosas sucedan como uno las planificó

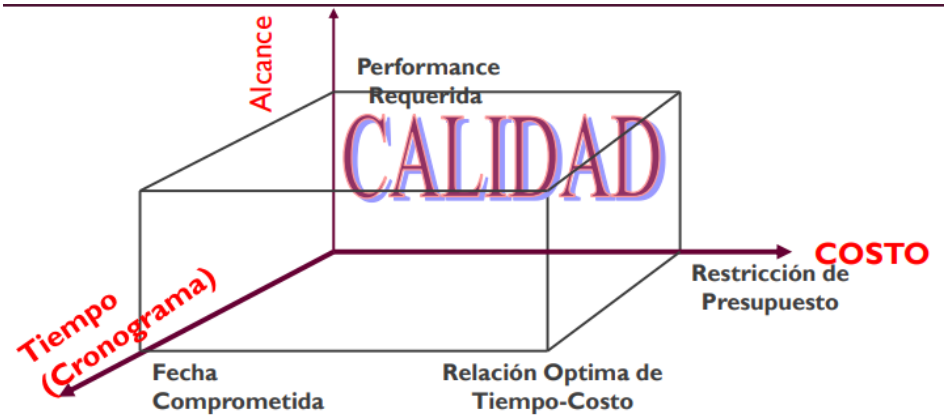
Aclaración: el foco de esta materia es la administración de proyectos ágiles.

La triple restricción (the triple constrain)

Se conoce también como ***triángulo de hierro.***

Objetivos de proyecto: ¿que está el proyecto tratando de alcanzar?
Tiempo: ¿cuánto tiempo debería llevar completarlo?
Costos: ¿cuánto debería costar?

El balance de estos tres factores afecta directamente la calidad del proyecto. "Proyectos de alta calidad entregan el producto requerido, el servicio o resultado, satisfaciendo los objetivos en el tiempo estipulado y con el presupuesto planificado."
Es responsabilidad del Líder de proyecto balancear estos tres objetivos, es decir, se debe lograr un equilibrio entre los objetivos (que a menudo compiten entre ellos).



Explicación de la imagen: la columna vertical que habla del alcance, se refiere al producto que se debe entregar. El alcance son los requerimientos que el cliente expresó y fueron entendidos de alguna manera. Para lograr entregar ese producto o servicio, se necesita tiempo, recursos y personas que terminan generando un costo. En el software, el costo más significativo de un proyecto es el costo del trabajo de la gente, recordemos que el 80% del costo de un proyecto de desarrollo son los honorarios de la gente que trabaja en ese proyecto.

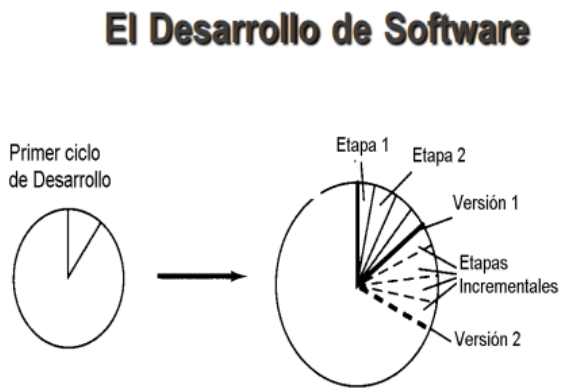
De las tres dimensiones, normalmente ***el alcance nunca depende de nosotros*** ya que es el cliente el que necesita el producto y el producto es del cliente. Es así que, en base al alcance podemos definir los costos y el tiempo y esas son cosas que sí podemos manejar desde el rol de líder de proyecto.

Si el cliente cambia el alcance, naturalmente los costos y el tiempo también van a cambiar. El problema surge cuando en la realidad, la toma de los requerimientos no es algo que se cobre entonces si el cliente decide no hacer el producto habría sido trabajado desperdiciado y encima no ganamos nada de plata por ese trabajo. Por otro lado, si el cliente decide hacer el producto pero cambia los requerimientos cuando ya se empezó a trabajar, lógicamente los costos y el tiempo establecido inicialmente también se modificarían y el problema surge cuando el cliente cree que cambiando los requerimientos, los costos y el tiempo se mantienen a como estaban inicialmente y entonces no quiere pagar la diferencia ni dar más tiempo para llevar a cabo el producto.

Como vemos, las tres dimensiones están relacionadas y si se realiza un enfoque de gestión basado en el alcance, el alcance determina el tiempo y el costo. Si cambia el alcance, los costos y el tiempo también van a cambiar.

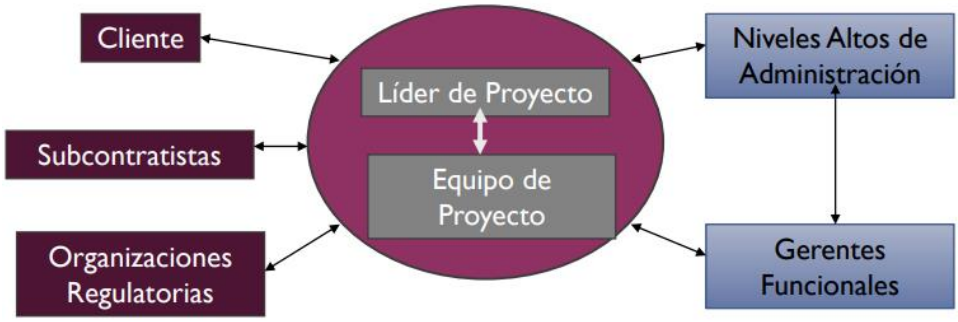
Si no logramos un equilibrio entre las tres dimensiones (costo-tiempo-alcance), se sacrifica la calidad del producto que se le entrega al cliente. Y debemos evitar a toda costa esta situación porque además de sacrificar la calidad del producto, el cliente va a quedar inconforme.

Para definir el tiempo y el costo debemos realizar estimaciones, pero como al comienzo del proyecto hay demasiada incertidumbre, establecer estas estimaciones en un inicio casi siempre sale mal, porque estimamos mal debido a esta incertidumbre.



Producto Software: Cada nueva versión es desarrollada **incrementalmente** en una serie de pasos

Rol de Líder de proyecto



En la gestión tradicional se toman decisiones anticipadas, decisiones al principio del proyecto que asumen que se tiene un nivel de información que en realidad no se tiene. La gestión tradicional tiene este concepto de equipo donde hay un líder.
El ***líder*** es el que se relaciona con los demás involucrados que son los que están alrededor (cliente, subcontratista, etc) y el líder es en general bastante conductista, es decir, le dice a la gente lo que tiene que hacer, cuando lo tiene que hacer, cuánto se tiene que demorar o para cuando eso tiene que estar terminado y el equipo recibe las asignaciones de trabajo y trabaja de la mejor manera posible.

El líder tiene la responsabilidad de:

- estimar
- planificar
- asignarle el trabajo a la gente
- hacer seguimiento de lo que está pasando con la gente en cada momento que va pasando el proyecto

El líder de proyecto no suele tener trabajo técnico como programar o diseñar o encargarse de los requerimientos. Si pasara el caso en el que tiene por ejemplo que programar, el líder de proyecto termina priorizando el programar antes que administrar el proyecto.

Equipo de proyecto

El ***equipo*** informa cuando ya está terminado el trabajo, informa cuánto se demoró, informa el porcentaje de avance y si hay una desviación interesante respecto a los planes debería el líder darse cuenta y ver qué acciones correctivas tomar como consecuencia de que se está teniendo un desajuste.
¿Qué es un equipo de Proyecto? Un grupo de personas comprometidas en alcanzar un conjunto de objetivos de los cuales se sienten mutuamente responsables.
Características de un equipo de proyecto:

- Diversos conocimientos y habilidades
- Posibilidad de trabajar juntos efectivamente /desarrollar sinergia
- Usualmente es un grupo pequeño
- Tienen sentido de responsabilidad como una unidad

¿Qué es el plan de proyecto?

En la gestión tradicional de proyectos, el líder de proyecto debe obtener un plan de proyectos y ese plan de proyecto es el que de alguna manera nos va a guiar a lo largo de todo el proyecto. El ***plan de proyecto*** define todo lo que nosotros queremos hacer, cómo lo vamos a hacer, cuándo lo vamos a hacer,etc todas esas preguntas se responden en el plan del proyecto.
El plan de proyecto es el artefacto resultante del proceso de planificación.

El plan de proyecto documenta:

- ***¿Qué es lo que hacemos?***: esto es el alcance del proyecto
- ***¿Cuándo lo hacemos?:*** es la calendarización
- ***¿Cómo lo hacemos?***: cómo se ven afectados los recursos y las decisiones de las tareas
- ***¿Quién lo va a hacer?***: hace referencia a la asignación de las personas.

¿Qué implica la planificación de proyectos de software? cuando tenemos gestión tradicional de proyectos basado en procesos definidos y se quiere hacer un plan de proyecto se debe establecer:

- Definición del Alcance del Proyecto
- Definición de Proceso y Ciclo de Vida
- Estimación
- Gestión de Riesgos
- Asignación de Recursos
- Programación de Proyectos (también conocido como calendarización)
- Definición de Controles
- Definición de Métricas: la métrica es un número, una definición cuantitativa que nos permite ver un proyecto desde un punto de vista más objetivo. Las métricas son objetivas, las métricas son la realidad, lo demás es percepción.

Alcance del producto y alcance del proyecto

El ***alcance del Producto*** son todas las características que pueden incluirse en un producto o servicio. Mientras que el ***alcance del Proyecto*** es todo el trabajo y solo el trabajo que debe hacerse para entregar el producto o servicio con todas las características y funciones especificadas.

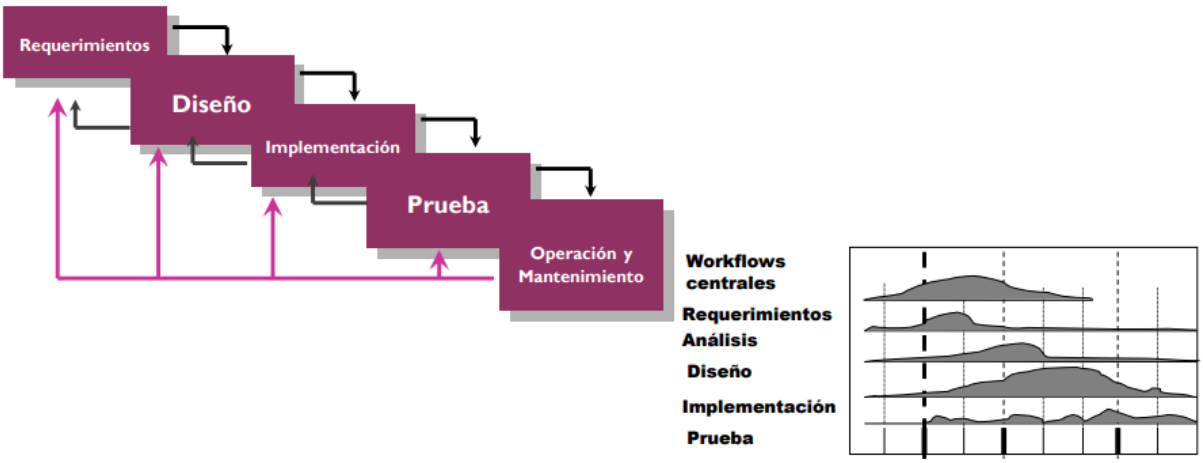
El ***alcance del producto***, que son todos los requerimientos funcionales y no funcionales, ***se guarda en la ERS.***
El ***alcance del proyecto*** son todas las tareas que se deben realizar para cumplir con el objetivo del proyecto. ***Se guarda en el plan de proyecto***.

Para definir el alcance del proyecto necesito tener definido antes el alcance del producto.

¿Cómo se mide el alcance?

- El cumplimiento del Alcance del Proyecto se mide contra el Plan de Proyecto (o Plan de Desarrollo de Software).
- El cumplimiento del Alcance del Producto se mide contra la Especificación de Requerimientos.

Definir un ciclo de vida



El proceso define las tareas que hay que hacer, mientras que el ciclo de vida te dice cómo vas a hacer esas tareas, cuánto de cada tarea vas a hacer cada vez.
Lo importante en el contexto de la planificación del proyecto es saber que cuando inicia el proyecto yo tengo que elegir qué proceso quiero usar y qué ciclo de vida quiero usar. Por ejemplo, el PUD recomienda el ciclo de vida iterativo e incremental.

Estimaciones de Software

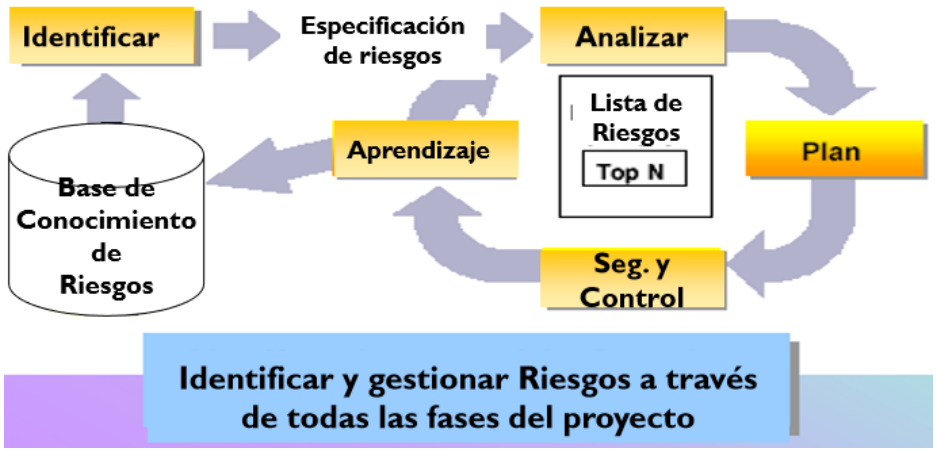
En la gestión tradicional se plantea una secuencia de estimaciones que tiene que tener el siguiente orden:

1) ***Tamaño*** del producto de software
2) ***Esfuerzo:*** son horas puras de trabajo, no se tiene en cuenta el tiempo para comer o ir al baño o cosas así.
3) ***Calendario***: se establecen qué días se trabajan para saber cuando va a estar listo el producto.
4) ***Costo***: se define al final porque debe ajustarse a la realidad
5) ***Recursos Críticos:*** son las cosas raras que nos hacen falta más allá de lo normal para hacer el producto. Se estima la necesidad, cuánto voy a necesitar, cuando los voy a tener que sumar a mi proyecto.

Riesgo

- Problema esperando para suceder
- Evento que podría comprometer el éxito del proyecto

Gestión de riesgos



Métricas de software

El dominio de las métricas del software se divide en:

- Métricas de proceso.
- Métricas de proyecto.
- Métricas de producto.

Las métricas del proyecto se consolidan para crear métricas de proceso que sean públicas para toda la organización del software.

Métricas básicas para un proyecto de software

- tamaño del producto
- esfuerzo
- tiempo (calendario)
- defectos

Tres factores para el éxito de un proyecto

- Monitoreo & Feedback
- Tener una misión/objetivo claro
- Comunicación

Causas de fracasos en proyectos

- Fallas al definir el problema
- Planificar basado en datos insuficientes
- La planificación la hizo el grupo de planificaciones
- No hay seguimiento del plan de proyecto
- Plan de proyecto pobre en detalles
- Planificación de recursos inadecuada
- Las estimaciones se basaron en “supuestos” sin consultar datos históricos
- Nadie estaba a cargo

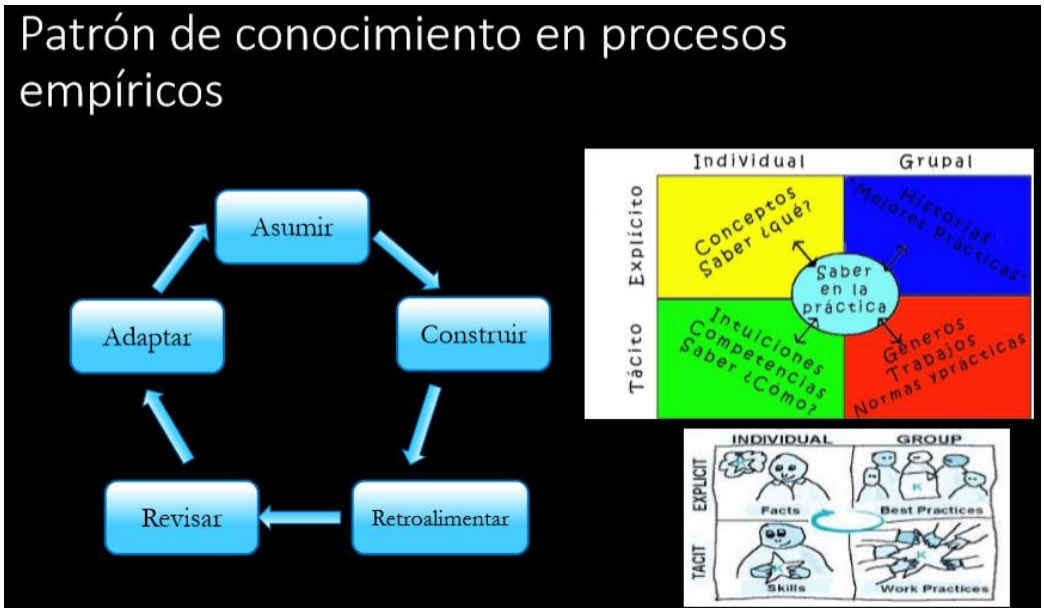
Unidad 2: Filosofía ágil - manifiesto ágil

El manifiesto ágil fue propuesto por desarrolladores y es un compromiso. El enfoque ágil nace buscando un equilibrio entre hacer software siguiendo rigurosamente un proceso donde a la larga termina siendo más importante cumplir con el proceso que entregar software de calidad y entre, yéndonos a otro extremo, hacer software a la ligera.

La filosofía ágil está materializada en el manifiesto ágil. El manifiesto ágil está basado en que la forma de trabajo va a ser sustentada en los conceptos de los procesos empíricos.

Desarrollo ágil de Software (Agile):es un compromiso útil entre nada de proceso y demasiado proceso (Fowler, 2001).

Recordemos que ***los procesos empíricos*** sólo aceptan ciclos de vida iterativo e incremental porque necesitan de una realimentación rápida para poder hacer ciclos cortos de inspección y adaptación.



Explicación de la imagen: siempre se comienza con algo, esta es la acción de ***asumir.*** Luego se ***construye*** y se muestra y el mostrarlo nos da la ***realimentación*** y la información para ***revisar*** lo que tengamos que revisar y si es necesario corregir cosas, se corrigen. Finalmente, el ciclo vuelve a empezar.

Pilares del empirismo:

- ***transparencia***
- ***inspección***
- ***adaptación***

El pilar de la transparencia es el que nos permite crecer como equipo y transformar el conocimiento implícito (es decir, el conocimiento que tenemos cada uno de nosotros) en conocimiento explícito (conocimiento que se pueda transmitir). El conocimiento explícito es conocimiento del equipo que luego es de la organización.
La transparencia se enfoca en "*no es mi código, es nuestro código*".

La transparencia ayuda a que los procesos empíricos puedan fluir, por eso es que es tan importante tener presentes estos pilares del empirismo.

¿Qué es ágil?

No es una metodología o proceso. Ágil es una ideología con un conjunto definido de principios que guían el desarrollo del producto. Ágil no es ser indisciplinado.

Valores de los equipos ágiles

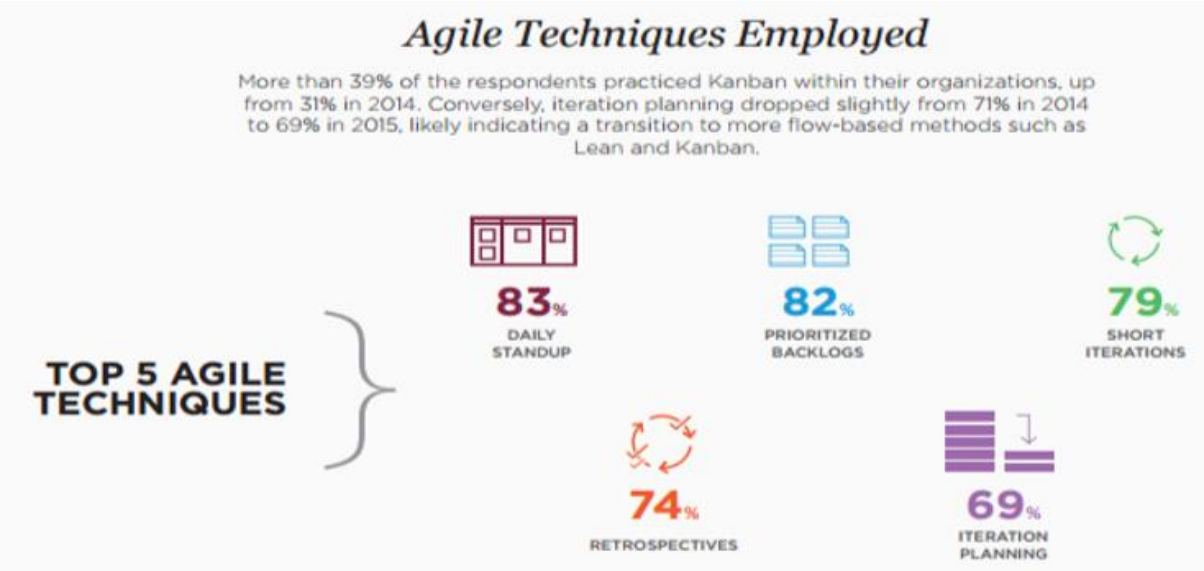
- Planificación continua, multi-nivel
- Facultados, auto-organizados, equipos completos
- Entregas frecuentes, iterativas y priorizadas
- Prácticas de ingeniería disciplinadas
- Integración continua
- Testing Concurrente

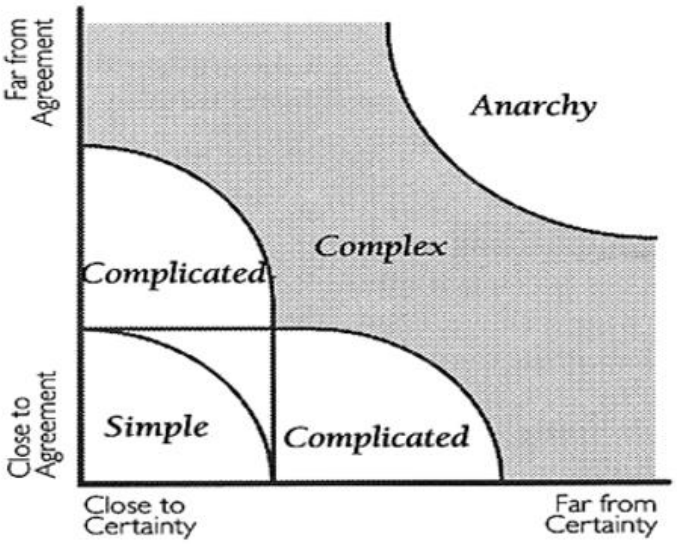
Ágil es entonces el balance entre ningún proceso y demasiado proceso. La diferencia inmediata es la exigencia de una menor cantidad de documentación, sin embargo no es eso lo más importante:

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

Algunos frameworks ágiles son: Scrum, XP, Crystal, FDD y ATDD.

Top 5 técnicas ágiles efectivas





¿Cuándo ágil es aplicable?

- Agile da mejores resultados cuando los problemas a ser resueltos caen dentro del espacio “Complex”.
- El desarrollo de nuevos productos y Knowledge Work tienden a estar en el espacio Complex.
- Investigación está dentro de Anarchy
- Mantenimiento cae en Simple

El manifiesto ágil

Posee **4 valores** y **12 principios.**

Valores ágiles

Es importante que los valores sean bien interpretados ya que el hecho de malinterpretarlos conduce a un software de mala calidad.

Explicación de la imagen:

- ***individuos e interacciones*** por sobre ***procesos y herramientas:*** las dos cosas son importantes, pero los individuos e interacciones son más importantes. Se valoran ambas cosas pero se valoran más los individuos e interacciones.
  Es decir, es más importante la comunicación entre los trabajadores y la comunicación con el cliente, con las organizaciones. Se hace énfasis en la interacción constante antes que el proceso en sí, que es rígido y estructurado y utiliza herramientas, modelos.
  Es más importante el vínculo y la comunicación que se establece entre la gente por sobre aferrarse a la herramienta.
- ***Software funcionando*** por sobre ***documentación extensiva***: es mucho más importante entregar software funcionando que tener una documentación extensiva con cada una de las características del software. Nos interesa el software en sí mismo más que el respaldo de qué hace, cómo lo hace, etc.
  Existe la necesidad de mantener información sobre el producto de software y sobre el proyecto que se realiza para obtener un producto y el ***enfoque ágil plantea que se debe generar la documentación cuando sea necesario. No plantea el "no generar documentación".***
  Ágil dice que los procesos de generación de conocimiento son lo más importante. La documentación es importante para evolucionar el producto hacia adelante, para que si otra gente viene a trabajar entienda las definiciones y las decisiones que se tomaron del producto, pero claro, hay que decidir qué es importante del producto mantener documentado. Ejemplo de algo que si o si debe estar documentado: la arquitectura.
  No toda la documentación que se genera es necesario que sea persistente eternamente sino que a veces es un medio para un fin, sirve mientras dura la iteración y cuando termina esa iteración ya no se necesita más esa documentación.
- ***Colaborar con el cliente*** por sobre ***negociación contractual***: estratégicamente es mejor generar un vínculo de confianza con el cliente antes de llegar a alguna negociación de contrato. Debemos hacer que el cliente forme parte del proyecto para generar el producto. El cliente no debe ser ajeno al proyecto.
  Es más importante dejar satisfecho al cliente por sobre las negociaciones contractuales.

¿En qué punto empieza a ser una fricción la situación de vínculo entre el equipo y los clientes/usuarios? La comunicación comienza a tener problemas cuando los clientes/usuarios piden cambios en los requerimientos. El problema de las negociaciones contractuales en muchos casos en las empresas empiezan cuando el cliente tiene ya una especie de acuerdo formal (contrato firmado) con el equipo por un monto y un plazo y una cantidad de características del producto de software que hay que entregar y el cliente viene y dice “me olvidé de esto” o “necesito esto” y ahí es donde empiezan los puntos de fricción por qué es donde comienzan las discusiones porque ya existe un contrato firmado donde se establecieron todas las pautas.
Esto también podemos relacionarlo con “la triple restricción”.
Por esto mismo es de fundamental importancia que el cliente participe y esté integrado en el proyecto, porque de esa forma se evitarían esta clase de conflictos.

Un equipo ágil va a poder sostenerse en el tiempo con sus prácticas si el contexto, es decir, la organización donde ese equipo funciona es ágil también.

- ***Responder al cambio*** por sobre ***seguir un plan***: este valor hace referencia a que cuando el cliente/usuario presenta cambios en los requerimientos, debemos adoptar una actitud de respuesta frente a esta situación y no de resistencia.
  Consiste en tener una actitud enfocada en el “construyamos juntos”. Por esto es que ágil comienza a trabajar con una idea del producto y a medida que transcurre el tiempo se va cerrando la idea y es esta característica lo que permite que se puedan introducir cambios en los requerimientos sin que se generen conflictos.
  La gente se puede equivocar y cambiar de opinión y por ello se debe adoptar una actitud más flexible, más ajustada con la realidad.
  Vamos con el proyecto “todos los días un poco” para lograr un resultado más exitoso en lugar de haber definido todo desde un comienzo y que si o si se deba seguir ese plan a rajatabla.

Los 12 principios del manifiesto ágil

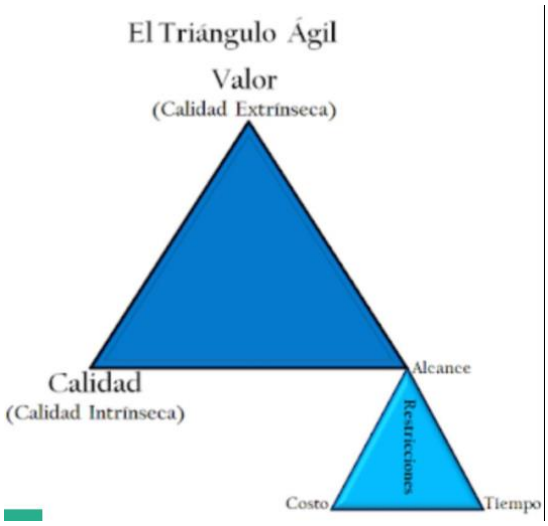
1) Nuestra mayor prioridad es ***satisfacer al cliente*** mediante la entrega temprana y continua de software con valor.
2) ***Aceptamos que los requisitos cambien***, incluso en etapas tardías del desarrollo. Los procesos Ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.
3) ***Entregamos software funcional frecuentemente***, entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto posible.
4) Los responsables de negocio y los desarrolladores ***trabajamos juntos*** de forma cotidiana durante todo el proyecto.
5) ***Los proyectos se desarrollan en torno a individuos motivados***. Hay que darles el entorno y el apoyo que necesitan, y confiarles la ejecución del trabajo.
6) El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la ***conversación cara a cara***
7) ***El software funcionando es la medida principal de progreso***.
8) ***Los procesos Ágiles promueven el desarrollo sostenible.*** Los promotores, desarrolladores y usuarios debemos ser capaces de mantener un ritmo constante de forma indefinida.Los frameworks ágiles apuntan a ciclos de vida iterativos de duración fija. La forma de lograr que un equipo tenga un desarrollo sostenible es lograr que el equipo tenga un ritmo de trabajo que asegure entregas en ciclos de por ejemplo 3 semanas o 4 semanas. Si el equipo puede crecer y generar la experiencia suficiente para obtener este desarrollo sostenible que no es rápido, entonces el equipo trabaja con un ritmo que puede garantizar al cliente una cantidad de software que se entrega en cada iteración.
9) ***La atención continua a la excelencia técnica y al buen diseño mejora la Agilidad.*** Significa que la calidad del producto no se negocia.
10) La simplicidad, o el arte de ***maximizar la cantidad de trabajo no realizado***, es esencial. La simplicidad se define como la maximización del trabajo no hecho. No

debemos agregar cosas extras al producto de software si el cliente no lo pidió por ejemplo, debemos obviar esos “y ya que estamos le agreguemos esto”.

11) ***Las mejores arquitecturas, requisitos y diseños emergen de equipos auto-organizados.*** Apunta a contrarrestar un poco el enfoque tradicional en el cual las decisiones de requerimientos de diseño y arquitectónica (que son las decisiones que más impacto tienen en el producto) suelen ser tomadas por consultores o por arquitectos o por gente que no está en el equipo. Ágil, por su parte, apunta a que la mejor gente para definir las características del producto y después para estimar y para decir cuánto tiempo necesita, etcétera, es la que va a hacer el trabajo. La calidad del producto emerge de equipos auto-organizados. Esto se relaciona con un principio “lean” que dice que hay que diferir las decisiones lo más que se pueda hasta el último momento responsable porque nos ayuda a tomar decisiones informadas.
    El equipo toma las decisiones en el momento que ellos creen adecuados.
12) ***A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo para a continuación ajustar y perfeccionar su comportamiento en consecuencia.***

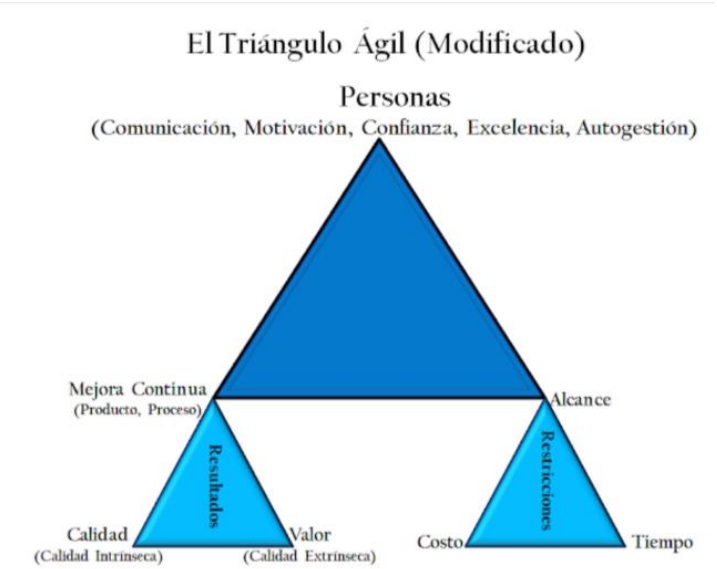
El triángulo ágil

De alguna manera, el triángulo ágil es una modificación de “la triple restricción”. La triple restricción hacía mucho foco en el proyecto, en los recursos del proyecto y en cumplir con los recursos del proyecto.
El ***triángulo ágil*** apunta a que lo que realmente tiene valor es el software funcionando para el cliente, por lo que se centra más en la calidad de producto, es decir, en la calidad intrínseca (las características de calidad que nosotros le ponemos al producto para que el producto pueda satisfacer al cliente) y la calidad extrínseca (el valor que el producto tiene para el cliente).



El ***triángulo ágil (modificado)*** apunta a un producto de software que pone a las personas (que tienen ciertas características como confianza, motivación, etc) en el lugar más importante ya que las personas son las que hacen el software. La intención de mejora continua que tienen las personas en el proyecto del producto y del proceso.

Finalmente se tienen en cuenta las variables que se usan para gestionar el trabajo (alcance-costo-tiempo).
Debemos hacer foco en entregarle al cliente un producto de calidad que se vaya mejorando y evolucionando en forma continua.



Relación de los valores ágiles con los principios ágiles

Este es el tp1 práctico que hicimos y nos sacamos 10, LEANLO PORQUE PUEDEN LLEGAR A TOMARLO: ISW_4K1_Grupo1_TP1.pdf

Requerimientos ágiles

El enfoque de requerimientos ágiles debe estar adherido al manifiesto ágil, es decir, a cumplir con los valores y con los principios que el manifiesto plantea.

Recordemos que la ingeniería de requerimientos tiene 2 procesos: por un lado ***el desarrollo de requerimientos*** y por otro lado ***la gestión de los requerimientos***. La gestión está más vinculada con mantener los requerimientos íntegros, bien identificados, bien trazados y que si cambian los requerimientos se actualicen todos los lugares donde se tengan que actualizar esa definición de requerimientos. Esto ocurre todo el tiempo mientras estamos trabajando con un producto de software. La parte de desarrollo de requerimientos, que es la parte que tiene que ver con identificar, especificar y validar los requerimientos.

Ahora, el enfoque ágil plantea cosas distintas para la gestión y el desarrollo de requerimientos.

Pilares de los requerimientos ágiles

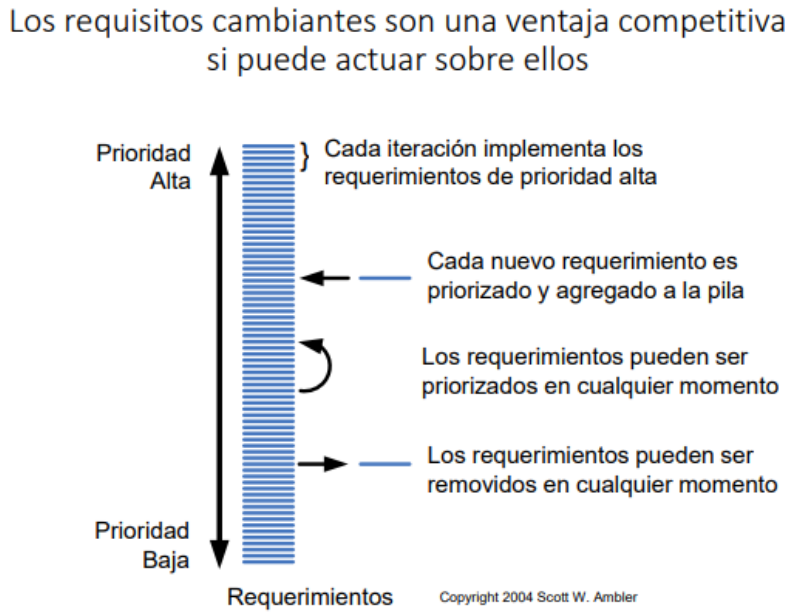
- ***Usar el "valor" para construir el producto correcto:*** Lo único importante es dejar contento al cliente con software funcionando que le sirva, por lo que, el foco de este enfoque de gestión ágil apunta a construir valor de negocio. Es decir, el software que creamos es para ayudar a alguien más a cumplir con sus objetivos y trabajo, por eso decimos que debemos enfocarnos en construir el producto correcto que le dé valor al negocio.
- ***Determinar que es "sólo lo suficiente"***: hace referencia a que los requerimientos deben ser encontrados de a poco, no buscar todos los requerimientos desde un inicio y quedarnos solo con eso. Este pilar postula que se tiene que empezar con lo mínimo necesario para generar realimentación y alimentar el empirismo y a partir de allí, se avanza continuamente.
- ***Usar historias y modelos para mostrar que construir:*** la parte de desarrollo de requerimientos está muy enfocada a la idea de construir junto con el cliente, respetando el principio ágil de "técnicos y no técnicos trabajando juntos". Se busca trabajar con un referente que esté del lado del negocio, que tenga el conocimiento del producto y que trabaje junto con el equipo.
  La etapa de capturar los requerimientos se transforma en una etapa de trabajo muy integrado entre el cliente y nosotros. Por eso el mejor medio de comunicación es la comunicación cara a cara porque la idea es armar equipos de trabajo en los que con técnicas de descubrimiento (las user stories) se pueda lograr crear la visión de qué producto se quiere construir y a partir de ahí, avanzar.

El costo del tradicional BRUF

Como podemos observar en el gráfico, los usuarios terminan usando muy poco de la totalidad de cosas que puede hacer el producto de software. Casi el 80% de las funcionalidades no se usan. Esto es algo muy real y un ejemplo de ello es que por ejemplo, de todo lo que hace excel solo lo usamos para funciones básicas.

Product backlog

La gestión ágil busca contrarrestar la situación planteada anteriormente donde gran parte del software no se termina utilizando a través del concepto de “product backlog”. El product backlog es definido por el PO (product owner) ya que es el product owner quien tiene claro cuáles son sus necesidades y su responsabilidad principal es priorizar requerimientos, es decir, es decidir qué es lo que él necesita, primero qué es lo que él considera que es de valor de negocio más importante como para que el equipo se ocupe primero de eso.



El product backlog es una lista priorizada organizada, donde los requerimientos más importantes se acomodan arriba en la lista y lo que va teniendo menos prioridad se va acomodando más abajo.

En contraposición con el enfoque tradicional, donde teníamos un usuario que no estaba muy dispuesto a comprometerse y al no comprometerse en priorizar dejaba la responsabilidad de priorizar al equipo que hace el software y en consecuencia se terminaba construyendo un software que no dejaba satisfecho al cliente, ahora con el enfoque ágil tenemos un cliente mucho más partícipe que se encarga de priorizar sus necesidades (requerimientos).

Entonces, para tener éxito usando el enfoque ágil se debe implantar una gestión ágil de requerimientos donde el cliente sea un referente que tenga ganas de trabajar de esta manera ya que debe estar disponible para el equipo para contestar todas las dudas de los mismos y además este cliente, debe ser alguien que tenga dentro de la organización conocimiento sobre el producto, es decir, que tenga la posibilidad de tomar decisiones sobre el producto.
Con esta situación se nos permite la compensación de no tener un documento de requerimiento formal escrito detallado con todos los puntos. Esta gestión ágil lo que ofrece

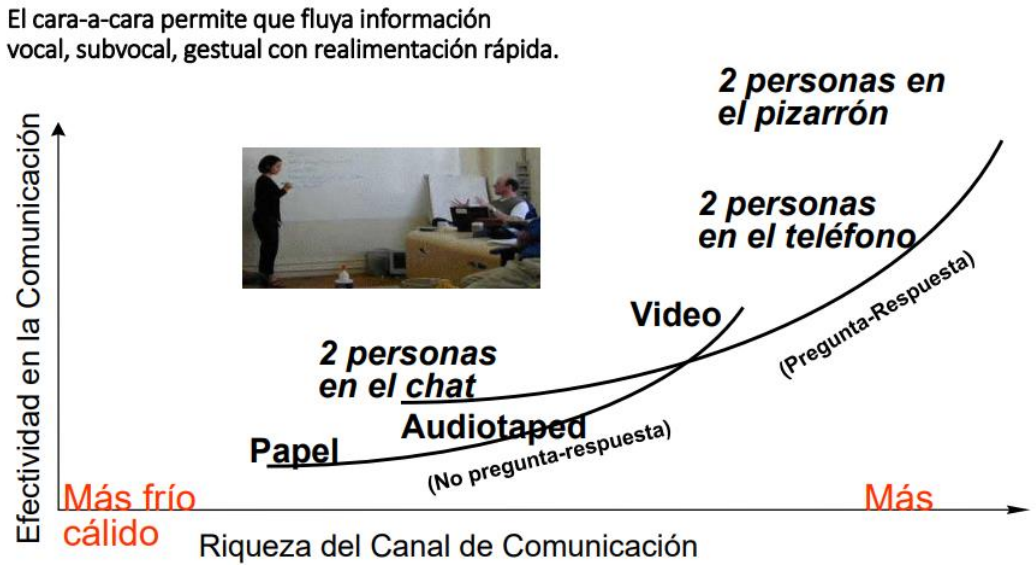
es una compensación al decir que “el mejor medio de comunicación es cara cara y el cliente está disponible para el equipo”.
La documentación formal se compensa con esta comunicación cara a cara y con esta disponibilidad del cliente.

Just in time

Es un concepto que se usa fundamentalmente para evitar desperdicios. Y con desperdicio se refiere a especificar requerimientos que después cambian y ya se ha invertido un montón de tiempo en especificarlos.
Por eso postula que se “analice cuando lo necesite, no antes”.

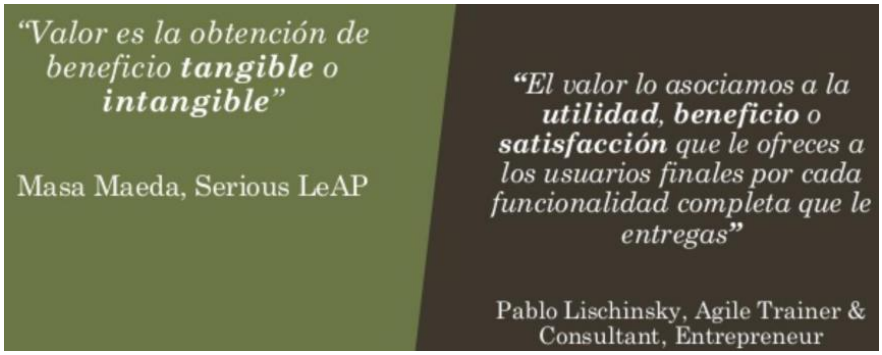
El producto no va a estar especificado al 100% desde el principio ni muchísimo menos, si no que se van a ir encontrando requerimientos y describiendo los requerimientos que hacen falta conforme haga falta. “ni antes, ni después”.

La comunicación cara a cara



La fundamentación del principio de la “comunicación cara a cara” es que la efectividad y la riqueza de la comunicación crece exponencialmente conforme todos puedan estar trabajando reunidos físicamente en un mismo lugar (co-locados) mientras se comparte un pizarrón para poder debatir y construir juntos el producto que se necesita.

El punto máximo se logra cuando logramos esta situación de dos personas en un mismo espacio físico con un pizarrón para compartir.



El software es un medio para nuestro cliente, es una herramienta de trabajo. Es decir, lo que debemos entregarle al cliente es el valor de negocio, no características de software. La característica de software es el medio por el cual les entregamos el valor de negocio.

Tradicional vs. Ágil

Explicación de la imagen: En el enfoque ágil donde estamos reforzando que el producto se va a ir definiendo conforme haga falta (just in time) y tomando decisiones en el último momento responsable, esto nos lleva a que en vez de dejar fijo los requerimientos (el alcance) y a partir de ahí derivar recursos y tiempo como plantea el enfoque tradicional, ahora, lo que hacemos es dejar fijo el tiempo con iteraciones de duración fija (en scrum es el sprint) y además dejamos fijos los recursos (es decir, tenemos un equipo de trabajo con una determinada capacidad que se estima con todos los recursos de infraestructura, comunicaciones y demás cosas que hagan falta, asignado a trabajar).
Así, manteniendo fijo los recursos y el tiempo, podemos establecer el alcance para esa determinada iteración en ese tiempo con ese equipo. Es decir, se acuerda cuánto del producto se puede construir con ese equipo en ese tiempo, o lo que es lo mismo, cuánto es lo que el equipo puede entregar de software funcionando en esa interacción. Y luego, esto se repite.
Qué es lo que se entrega en cada iteración se va definiendo al comienzo de cada iteración. Pero tenemos la ventaja de que como es el PO el que prioriza los requerimientos, los puede ir modificando sin problema gracias a esta forma de trabajo, conforme la realidad del negocio demande modificar los requerimientos y conforme el PO lo necesite.

Tipos de requerimientos

A nivel de la gestión ágil de requerimientos se trabaja con:

- requerimiento de negocio
- requerimiento de usuario

Las user stories son requerimientos de usuario alineados a un requerimiento de negocio.

Al usuario le interesa el requerimiento de negocio. El valor de negocio es el requerimiento de negocio, pero el requerimiento de usuario es el medio por el cual vamos a lograr ese valor de negocio.

En ASI y en DSI solo trabajamos con requerimientos funcionales, no funcionales y con algo de los requerimientos de implementación.

Las user stories no sirven para especificar requerimientos de software, sirven para identificar requerimientos de usuario.

El foco de la gestión ágil de requerimientos está puesto sobre el usuario y sobre sus necesidades y por eso en enfoques como scrum ***quién debe trabajar para identificar las user stories es el PO.*** Ya que es él quien se concentra en las necesidades de usuarios que van a satisfacer las necesidades de negocio.

Las user stories están ubicadas en “requerimientos de usuario” con un foco claro hacia el negocio.
Los casos de uso están ubicados en “requerimientos de software”.

“Entendiendo la necesidad y el negocio, descubriendo la solución de forma colaborativa junto a un equipo motivado y competente podemos entregar frecuentemente valor a los stakeholders.”
Lo que implica la gestión ágil de requerimientos es que tenemos que trabajar juntos (técnicos y no técnicos) entendiendo las necesidades y el negocio para construir un software que ayude al cliente. Luego, junto con los usuarios debemos descubrir cuál es la mejor forma de satisfacer esas necesidades (se incorpora el product backlog para especificar en una determinada iteración qué es lo que se le va a entregar al cliente).
Al entregarle al cliente la característica podemos obtener feedback y es ese feedback el que nos ayuda a refinar el product backlog.

¿Qué debemos asumir?

- Los cambios son la única constante. Todo el tiempo va a haber cambios en los requerimientos.
- Stakeholders: no son todos los que están. Debemos saber quienes son los involucrados y los involucrados son todos los que tiene algo que decir respecto del producto. El PO es el representante de todas estas personas y el que se comunica con todas ellas para luego ser él mismo el que se comunica con el equipo que hace el software.
- Siempre se cumple eso de que: “El usuario dice lo que quiere cuando recibe lo que pidió”.
- No hay técnicas ni herramientas que sirvan para todos los casos. Debemos detectar para cada situación cuál es la herramienta más adecuada.
- Lo importante no es entregar una salida, un requerimiento, lo importante es entregar, un resultado, una solución de “valor”.

Principios Ágiles relacionados a los Requerimientos Ágiles

Los más importantes son el 4 y el 6.

El PO es parte del negocio y es cliente, es quien hace el trabajo en el negocio y puede tomar decisiones respecto a cómo se hace el trabajo.

User stories

Se las llama “stories” porque se supone que el cliente cuenta una historia. Lo que se escribe en la tarjeta no es importante, lo que el cliente habla y pide si lo es.
La ***user storie*** es una descripción corta de una necesidad que tiene el usuario respecto del producto de software.
La parte más difícil de construir un sistema de software es decidir precisamente qué construir. Ninguna otra parte del trabajo conceptual es tan difícil como establecer los requerimientos técnicos detallados. Ninguna otra parte del trabajo afecta tanto el sistema resultante si se hace incorrectamente. Ninguna otra parte es tan difícil de rectificar más adelante como esa.
La user storie es una técnica de identificación de requerimientos.
Las user storie son multipropósito. Las historias son:

- Una ***necesidad del usuario***
- Una ***descripción del producto***
- Un ***ítem de planificación*** para saber lo que entra en una iteración
- ***Token para una conversación***
- ***Mecanismo para diferir una conversación***

¿Cuales son las partes de una user storie?

- conversación
- tarjeta
- confirmación

La más importante de las tres es la conversación. La conversación no queda guardada explícitamente en ningún lado, a lo sumo queda guardada en un pizarrón con un par de garabatos pero está centrada en este principio ágil de “la comunicación es cara a cara”.
La parte que queda de la conversación es la parte visible de la user storie (la ***tarjeta*** o card en inglés y la ***confirmación*** que son las pruebas de usuario).

Tarjeta

Posee una nomenclatura definida:

<Frase verbal>
Como **<nombre del rol>** yo puedo/quiero **<actividad>** de forma tal que **<valor de negocio que recibo>**

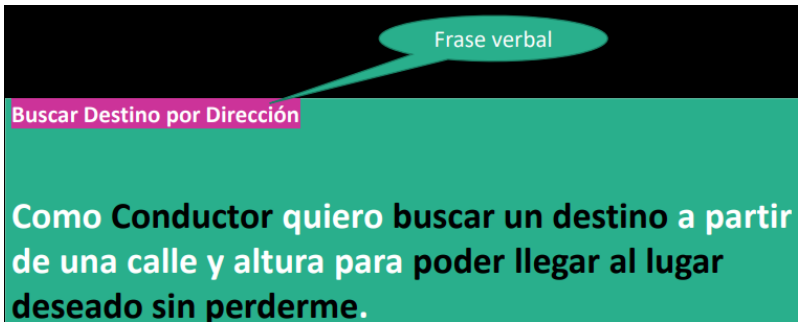
- **<Frase verbal>** no es obligatoria, es una forma corta para referenciar la user storie.
- **<nombre del rol>** Representa quién está realizando la acción o quién recibe el valor de la actividad. Debe ser especificado explícitamente el rol del usuario, no conviene usar el término genérico de "usuario" excepto, si y sólo si, estamos hablando de la user storie de *"loguear usuario"*.
- **<actividad>** Representa la acción que realizará el sistema.
- **<valor de negocio que recibo>** Comunica porque es necesaria la actividad.

Se busca contestar tres cosas:

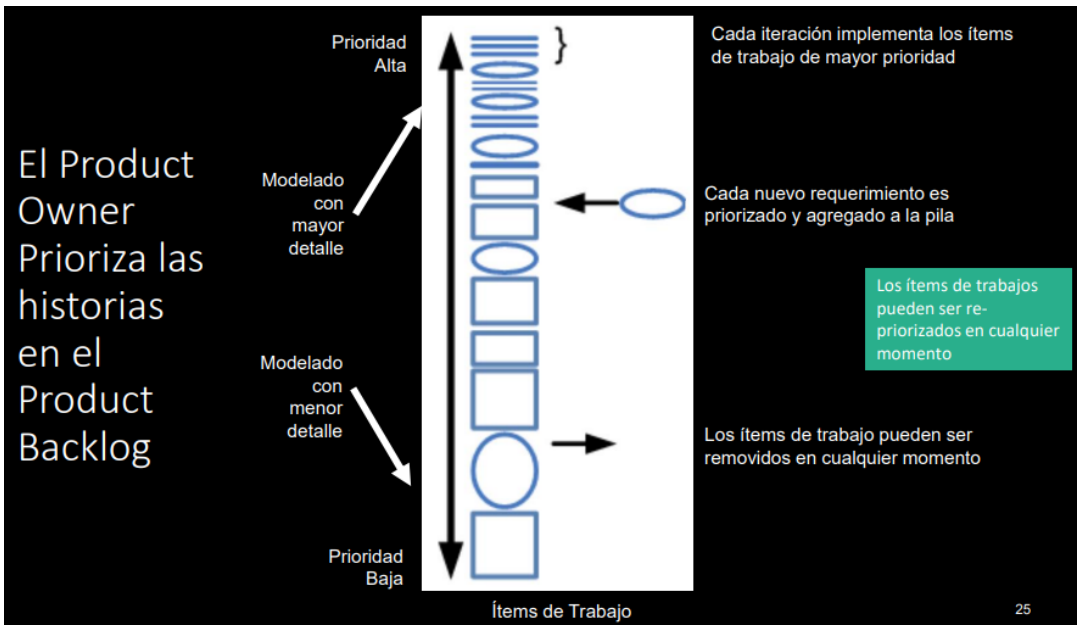
- who: quién necesita esto
- what: que se necesita
- why: por qué o para qué haciendo referencia al valor de negocio.

Es importante tener claro el valor de negocio para que el PO priorice el Product Backlog.

Ejemplo:



El Product Backlog y las user stories



Las user stories caen al product backlog en el orden de priorización que el PO defina y mientras tanto, conforme al PO se le van ocurriendo cosas, es él quien las agrega, las quita, la saca o les pone más prioridad, cambiandolas según lo que a él le parezca.
El dueño del Product Backlog es el PO.

Porciones verticales

Las ***user storie son porciones verticales***, es decir, para poder entregarle valor al cliente debemos tener un poco de interfaz, un poco de lógica de negocio y otro poco de base de datos.
Debemos diseñar el pedacito de cada cosa que nos hace falta para que una determinada característica funcione.

Modelado de roles: Tarjeta de Rol de Usuario

Es una tarjetita al estilo de la user storie donde se anotan las características en general que va a tener el usuario.
Ejemplo:

Rol de Usuario: Reclutador Interno
Nos es un experto en computadoras, pero bastante adepto a utilizar la Web. Utilizará el software con poca frecuencia pero muy intensamente. Leerá anuncios de otras compañías para averiguar cuál es la mejor palabra para sus anuncios. La facilidad de uso es importante, pero más importante es que lo que aprenda, lo pueda recordar meses después.

Describimos un perfil de usuario para entenderlo y conocerlo y para que quien diseña en la experiencia de usuario, la diseñe pensando en ese perfil de usuario con cosas concretas. Esta es una técnica genérica con la que se empieza pero después la idea es refinar esto con técnicas más específicas como por ejemplo una técnica que se llama "***personas***" que describe a una persona en particular. Claramente esta técnica "personas" no puede hacerse para todos los usuarios, por lo que se eligen tipos de usuario (o distintos perfiles) que representen a un conjunto de personas con las mismas características comunes.

¿Qué alternativa tenemos si el PO no tiene tiempo para nosotros?

Si el PO no tiene tiempo de responder nuestras preguntas y formar parte del proyecto activamente, entonces el PO debe elegir a alguien que también conozca el negocio tanto como él y que pueda sernos útil para comunicarnos los requerimientos y necesidades del negocio.
Los usuarios representantes (proxies) pueden ser:

- Gerentes de Usuarios
- Gerentes de Desarrollo
- Alguien del grupo de marketing
- Vendedores
- Expertos del Dominio
- Clientes
- Capacitadores y personal de soporte.

Si bien todas estas alternativas son perfiles de negocio, no son ideales como los usuarios verdaderos, así que siempre debemos tratar de trabajar con el PO.

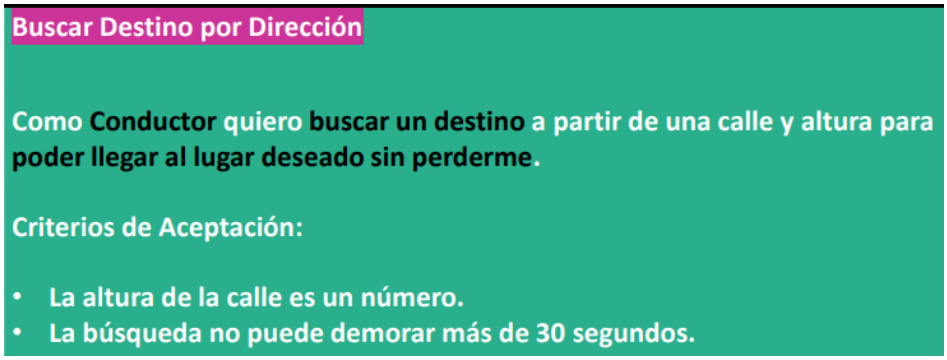
Confirmación: Criterios de aceptación de user stories

Los criterios de aceptación de la user storie son la base para que después se construyan los casos de prueba.
El criterio de aceptación es información concreta que tiene que servirnos a nosotros para saber si lo que implementamos es correcto o no. Debemos saber si el PO nos va a aceptar esta característica implementada o no, porque la va a aceptar si respeta los criterios de aceptación.

- Definen límites para una user story (US)
- Ayudan a que los PO respondan lo que necesitan para que la US provea valor (requerimientos funcionales mínimos)
- Ayudan a que el equipo tenga una visión compartida de la US
- Ayudan a desarrolladores y testers a derivar las pruebas.
- Ayudan a los desarrolladores a saber cuando parar de agregar funcionalidad en una US.

Los criterios de aceptación explícitamente escritos son un aporte que hace la cátedra a la plantilla. En la versión original aparecían como una notita opcional.

Ejemplo:



Los criterios de aceptación se escriben a nivel de usuario y tienen que ser objetivos y verificables.

¿Cuáles son los Criterios de Aceptación buenos?

- Definen una intención, no una solución. Ej.: El usuario debe elegir al menos una cuenta para operar.
- Son independientes de la implementación.
- Relativamente de alto nivel, no es necesario que se escriba cada detalle.

¿Y los detalles?¿Dónde van?

Detalles como:

- El encabezado de la columna se nombra “Saldo”
- El formato del saldo es 999.999.999,99
- Debería usarse una lista desplegable en lugar de un Check box.

Estos detalles que son el resultado de las conversaciones con el PO y el equipo puede capturarlos en dos lugares:

- Documentación interna de los equipos
- Pruebas de aceptación automatizadas

Cada equipo decide dónde guardar el detalle y cómo lo va a mantener.

Pruebas de aceptación de la User storie

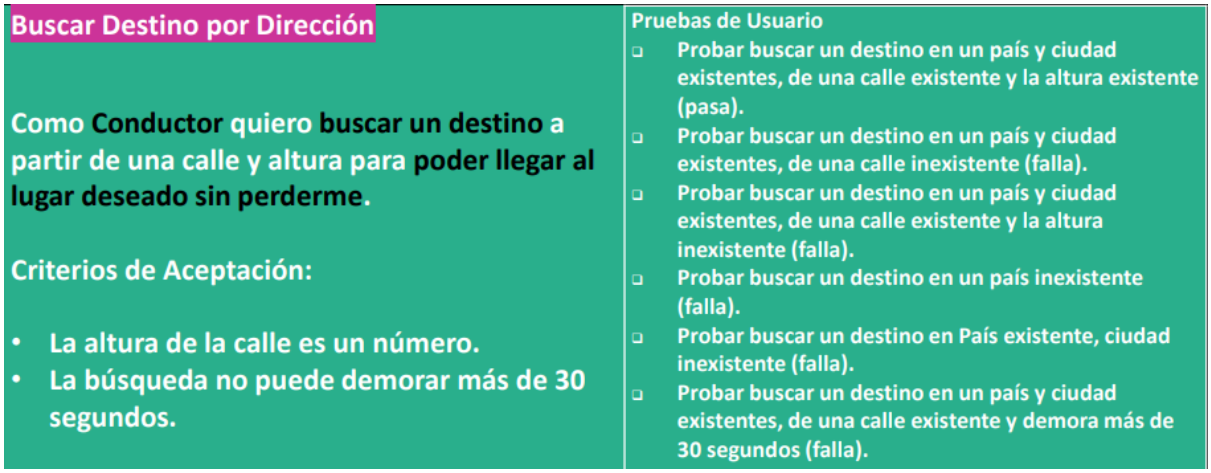
Expresan detalles resultantes de la conversación y complementan la User Story.

Proceso de dos pasos:

1) Identificarlas al dorso de la US.
2) Diseñar las pruebas completas

¿Cómo escribimos las pruebas de aceptación para la user storie?

Ejemplo:



Acá describimos lo que tenemos que probar.

En las pruebas de usuario tenemos que contemplar situaciones de éxito y de fracaso.

Pensar en las situaciones de fracaso nos permite pensar qué actitud vamos a tener frente a esa situación, eso sería ***el manejo de errores y excepciones.***

Las ***pruebas de usuario las define el usuario*** porque es el usuario quien nos cuenta la realidad del negocio, ahora bien, quien escribe y deja explícitas las pruebas de usuario no es el usuario en sí. Es un trabajo colaborativo entre el equipo y el PO.

Definition of ready (definición de listo)

Es una medida de calidad que construye el equipo para poder determinar que la user storie está en condiciones de entrar a una iteración de desarrollo. Para implementar la user, la misma debe cumplir con el INVEST Model.

INVEST model

Es el modelo de calidad de base y nos ayuda a saber si una user storie está en condiciones de incluirse en una iteración de desarrollo.

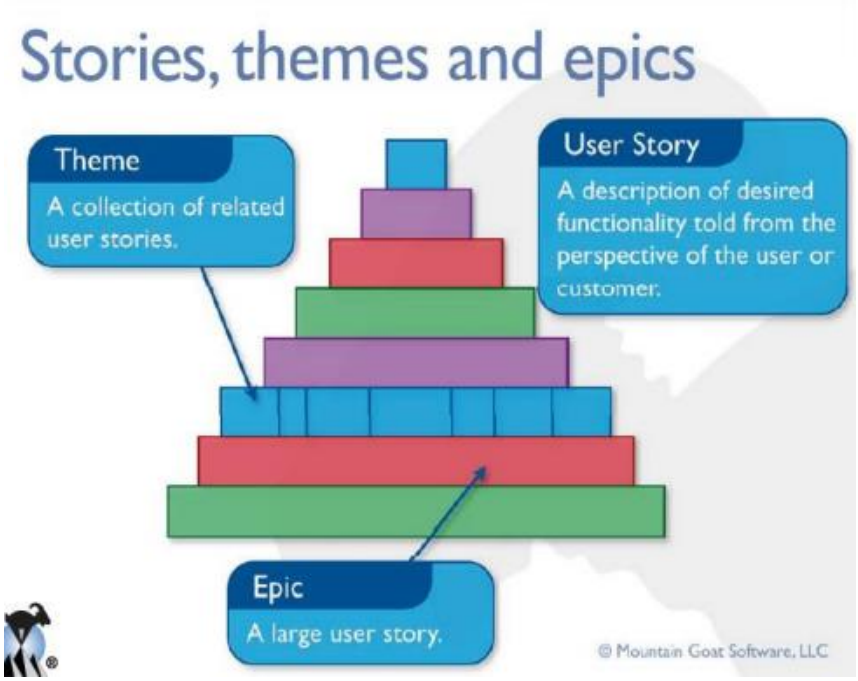
- ***Independent***: calendarizables e implementables en cualquier orden. Le damos la libertad al PO de que priorice las US para que se puedan desarrollar en cualquier orden.
- ***Negotiable***: el “qué” no el “cómo”. La US tiene que estar escrita en términos de qué necesita el usuario, no de cómo lo vamos a implementar.
- ***Valuable***: debe tener valor para el cliente. Debe estar el para qué, es decir, tiene que tener valor de negocio.
- ***Estimatable:*** para ayudar al cliente a armar un ranking basado en costos. Es asignarle un peso a la US para saber cuánto tiempo me va a llevar para terminar esa US completa.
- ***Small:*** deben ser “consumidas” en una iteración. Debemos empezar y terminar una característica en una iteración, no debe haber trabajo a medias. Esto depende mucho del equipo, de la experiencia del equipo, de que tanto el equipo ha definido su capacidad del trabajo y también depende de la duración de la iteración.

- ***Testable:*** demostrar que fueron implementadas. Está relacionado con las pruebas de aceptación, es poder demostrar que la US efectivamente se implementó cumpliendo los criterios de aceptación que se definieron.

Algo más sobre las US

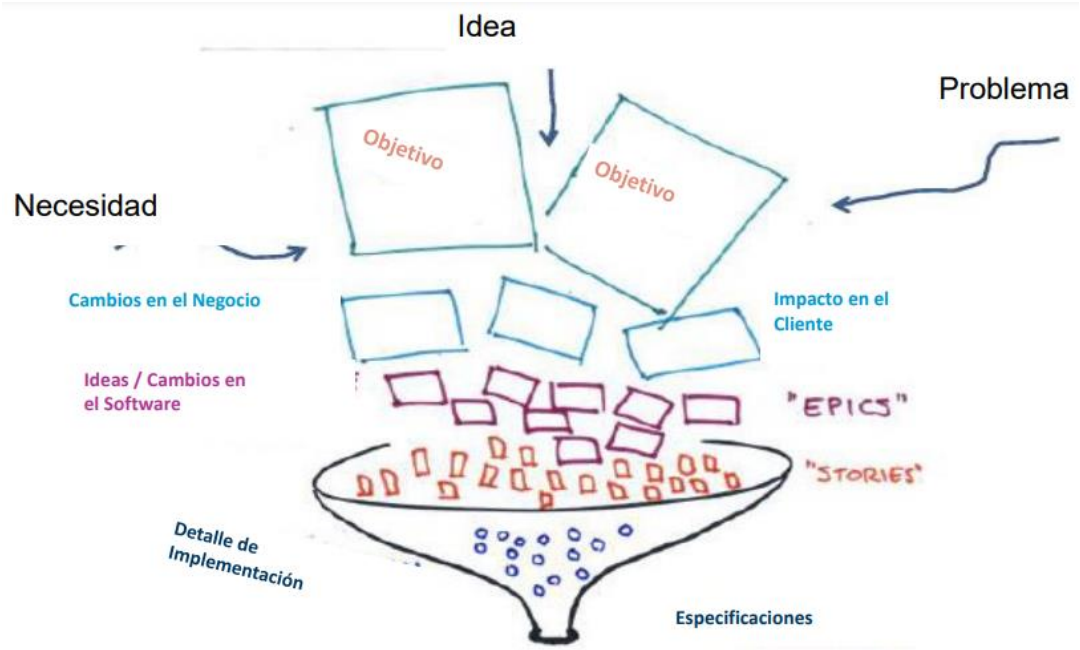
- No son especificaciones detalladas de requerimientos (como los casos de uso). Son más bien necesidades las que se transmiten, los deseos de funcionalidad de los usuarios.
- Son expresiones de intención, “es necesario que haga algo como esto…”
- No están detallados al principio del proyecto, elaborados 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.
- Una historia de usuario define una necesidad del usuario desde la perspectiva del usuario/cliente.
- Con las 3 C de la US ya estamos en condiciones de implementarla (recordemos que las 3 C son: conversación, confirmación y tarjeta).

Diferentes niveles de abstracción



Epic: la épica es una US grande.

¿Cómo distingo una Epic de una US? Cuando no se cumple INVEST o cuando no puedo ejecutar esa US en una iteración se trata de una Epic. Por ende, la Epic debe ser dividida en US más pequeñas.
Theme: agrupa las US que están relacionadas a un mismo tema.



Explicación de la imagen: los impactos y cambios en el software terminan transformándose en épicas y a medida que las épicas se van refinando se transforman en US.
El embudo representa una iteración que se va refinando hasta obtener las especificaciones.

Spikes

- Son un tipo especial de US.
- Tipo especial de historia, utilizado para quitar riesgo e incertidumbre de una User Story u otra faceta del proyecto.
- Se clasifican en : técnicas y funcionales.
- Pueden utilizarse para:
    - inversión básica para familiarizar al 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 prototipando para ganar confianza.
    - Frente a riesgos funcionales, donde no está claro como el sistema debe resolver la interacción con el usuario para alcanzar el beneficio esperado.

Spikes técnicas

- Utilizadas para investigar enfoques técnicos en el dominio de la solución.
    - Evaluar performance potencial
    - Decisión hacer o comprar
    - Evaluar la implementación de cierta tecnología.
- 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

- Utilizadas cuando hay cierta incertidumbre respecto de cómo el usuario interactuará con el sistema.
- Usualmente son mejor evaluadas con prototipos para obtener realimentación de los usuarios o involucrados.

Algunas User Stories requieren de ambos tipos de spikes.

Lineamientos para Spikes

- Estimables, demostrables, y aceptables
- La excepción, no la regla
    - Toda historia tiene incertidumbre y riesgos.
    - El objetivo del equipo es aprender a aceptar y resolver cierta incertidumbre en cada iteración.
    - Los spikes deben dejarse para incógnitas más críticas y grandes.
    - Utilizar spikes como última opción.
- Implementar la spike 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, spike e historia pueden incluirse en la misma iteración.

Algunas cosas para dejar en claro

Tips para que la US sea útil para el equipo

- Un paso a la vez (evitar la palabra “Y”)
- Usar palabras claras en los criterios de aceptación
- No olvides la parte invisible: la conversación
- Las user stories se escriben desde la perspectiva del usuario
- No forzar todo para escribirlo como user stories

Estimaciones de Software

El problema principal de las **estimaciones** es que tienen integrado el concepto de **probabilidad**, es decir, tienen un valor de probabilidad asociado o no, por lo que pueden o no salir bien. Informan un **valor cuantitativo asociado a una probabilidad** (por ejemplo, tengo el 70% de probabilidad de entregar este producto el 11 de noviembre).

Frente a la posibilidad de estimar mal y generar enojo por parte del cliente, tenemos dos reacciones posibles:
Como la estimación es un trabajo a riesgo, se subestima, o sea, simplificando sin considerar todo el trabajo que hay que hacer, por miedo a que si el valor es mas alto el cliente lo rechace.
Sobreestimar, cuando me preguntan por ejemplo cuantas horas me va a llevar un trabajo, y, sabiendo que me va a llevar 2 horas, digo 4 por las dudas pase algo en el medio, inflando muchísimo la complejidad del producto.
Las estimaciones se tienden a confundir con compromisos o planificación. Uno planifica basándose en la estimación, pero no son lo mismo.
No es una instancia única, ya que a medida que se avance el proyecto, más información vamos a tener y más eficiente van a ser nuestras estimaciones.

Teniendo en cuenta esto, hay que aceptar que la estimación no es fácil, de hecho, es lo más difícil luego de la elicitacion.

¿Para que estimamos?

El costo más grande es el del esfuerzo. Al inicio del proyecto no podemos asumir cosas inamovibles debido a que no tenemos mucha información. Es por esto por lo que se recomienda estimar entre 2 y 4 veces más de lo que se ha estimado originalmente, y se debe asumir que en ningún caso las estimaciones van a ser precisas.

Métodos para la estimación

La problemática en nuestra industria es que trabajamos con software, que es intangible, por lo que es difícil cuantificarlo.

- **Basados en la experiencia**
  - **Datos Históricos**: Se toma como base la experiencia de otras personas y proyectos. Esto no va muy acorde a la gestión ágil porque esta dice que la experiencia no se puede extrapolar a otros equipos.
  - **Juicio Experto**
    - **Puro**: Es el mas utilizado en la industria, implica que dentro de la organización hay un gurú dedicado a estimar en base a su propio criterio. Es un riesgo ya que todo el conocimiento recae en una sola persona, pero la industria del software tiende una rotación de empleados muy alta, por lo que es muy probable que este empleado cambie de empresa.
      Se puede utilizar el método de “optimista, pesimista y habitual”, en el que se le pregunta al experto sobre 3 valores distintos y se aplica una formula para calcular la estimación.
      También se puede hacer un checklist para asegurarse de que el experto no se ha olvidado de tener en cuenta ningún aspecto importante.
    - **Delphi**: Es grupal y se toma como base para el póker planning
  - **Analogía**: Se toman los datos históricos y se hace una abstracción de los que son parecidos o concuerdan con el proyecto en el que se esta trabajando
- Basados exclusivamente en los recursos
- Método basado exclusivamente en el mercado
- Basados en los componentes del producto o en el proceso de desarrollo
- Métodos algorítmicos

Podemos resumir que las estimaciones tradicionales tienen los siguientes problemas:

- Estimar demasiado pronto
- Resistencia a las reestimaciones
- Estimaciones que buscan precisión
- Estimaciones hechas por gente distinta a la que hace el trabajo
- Estimaciones que pretenden ser absolutas (inflexibles)

Desde esta problemática es que se aborda el tema de las estimaciones agiles.

Estimaciones en ambientes agiles

Hay una fuerte propuesta de que estime la misma persona que hace el trabajo, no algún gurú. Debemos tener la mente abierta para escuchar lo que el otro tiene que decir y no cerrarse a nada. Las estimaciones son un proceso y uno tiene que hacer foco en este proceso.

Lo ideal es empezar con una estimación que se pueda revisar todas las veces que sea necesario. Recordando el principio de que “la mejor métrica de progreso es el software funcionando”, tenemos que entender que todo lo definido en cuanto a estimaciones tiene que ver con el producto. Entonces en ágil, ahora se comienza a estimar el tamaño del producto, las características del sistema. Las medidas relativas no son absolutas (por ejemplo, M, L, XL, XXL, etc.). Story Points no es una medida basada en tiempo. La Story Point es el peso de la User Story y ese peso lo usamos por comparación (las estimaciones agiles, son comparaciones relativas, ya que son más fáciles las estimaciones relativas que las absolutas)

No podemos comprar directamente una user story con otra porque hay distintas dimensiones y ahí es donde entra el Story Point, que hace de unidad homogeneizadora que me permite comparar.

Para las estimaciones debemos tener en cuenta la **complejidad**, el **esfuerzo** (por eso es importante que estime el que hace el trabajo, ya que no es lo mismo el esfuerzo que le va a llevar a Juan que a Pedro) y la **duda** (incertidumbre, cuanta información falta para ser más adecuados en la estimación). Estas tres variables o dimensiones son las que conforman el story point, y así puedo comparar user stories.

Descripción del Trabajo vs Esfuerzo

Como el esfuerzo (horas de trabajo) es el factor mas significativo del costo de un proyecto, tenemos una gran tendencia a confundir tamaño con esfuerzo y esfuerzo con calendario. La complejidad es independiente de quien haga el proyecto, y el esfuerzo depende específicamente de la persona que lo va a llevar a cabo, de su capacitación, conlleva horas lineales (horas únicamente de trabajo, sin tener en cuenta almuerzo, baño, leer emails, etc. Para una jornada de trabajo de 8 horas, usualmente solo 5.5 o 6 son lineales). Por otro lado, no se lo debe confundir al esfuerzo con calendario, ya que yo puedo decir que puedo tener un trabajo para el viernes, pero en el medio pueden suceder millones de cosas que eviten que cumpla mi objetivo. Un trabajo puede requerir un esfuerzo de x horas, pero no se puede asegurar que para tal día va a estar terminado.
Hay diferentes tipos de formas de medir escalas (números del uno al 10, letras de las S a XXL, series 2^n, Fibonacci). Lo importante es que, una vez elegida la escala, no se cambia.

Velocidad

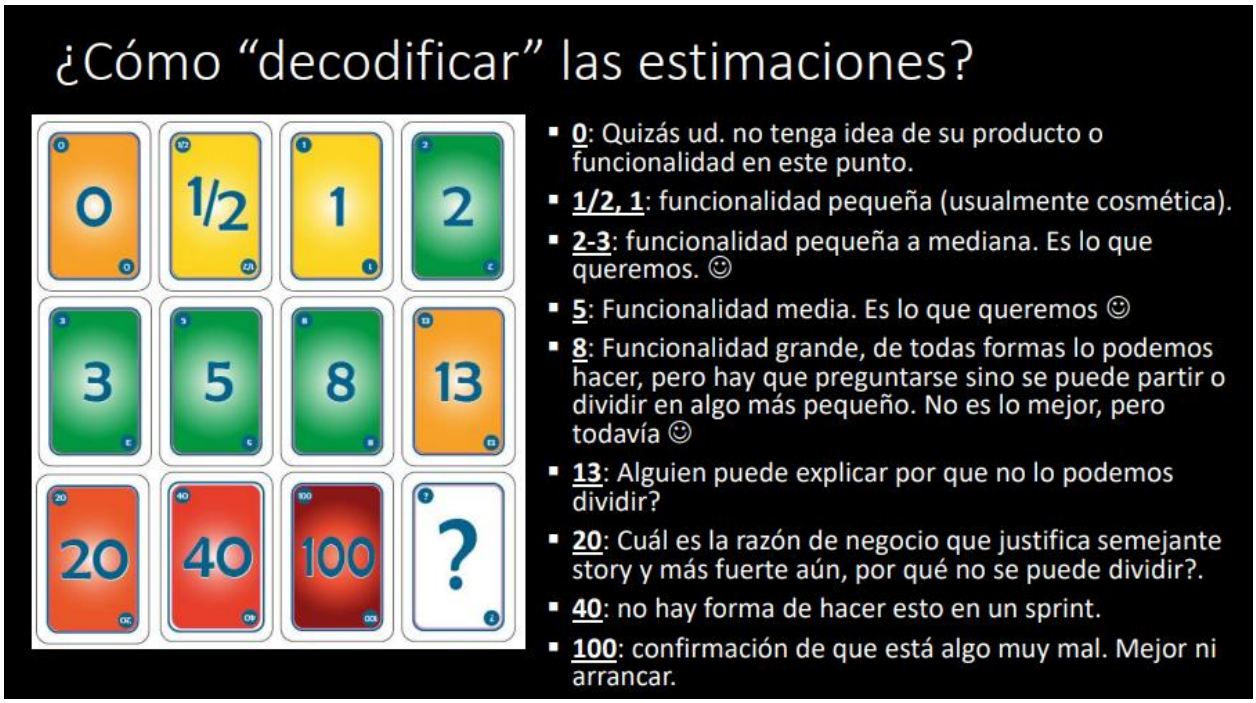
Es una medida (métrica) del progreso de un equipo. No se estima, se calcula el final de la iteración, sumando el número de story points (asignados a cada user story) que el equipo completa durante la iteración.
Se cuentan los story points de las user storys que están completas, sin tener en cuenta las que están parcialmente completas.

La velocidad corrige errores de estimación.

Nunca nos comprometemos a una velocidad especifica, sino a un rango.

La duración de un proyecto no se estima, se deriva tomando el numero total de story points y dividiéndolo por la velocidad de un equipo. Esta velocidad nos ayuda a determinar un horizonte de planificación apropiado.

La estimación en story points separa completamente la estimación de esfuerzo de la estimación de la duración.



SCRUM

Primero que nada, leerse la guía de scrum 2020
https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-Spanish-European.pdf

Artefactos de Scrum

Es importante recalcar que a partir de la guía de scrum 2020, se le asocia a cada artefacto de SCRUM un compromiso.

El producto backlog esta alineado con un objetivo (el objetivo del producto), y toda característica que agreguemos o quitemos tiene que alinearse con este objetivo.

También tenemos el Sprint Backlog, que se relaciona con el objetivo del sprint, siendo este último que nos permite tal variabilidad que podemos intercambiar algún ítem del producto backlog e insertarlo en el sprint backlog, siempre y cuando no impacte en este objetivo. Esto nos otorga flexibilidad y aceptación de la realidad de que a veces es necesario afectar algunas cuestiones.

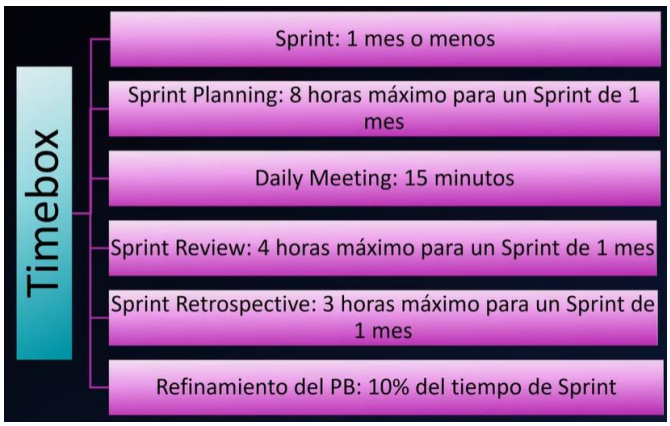
El incremento del producto está relacionado con el **Definition of Done** (DoD, se desarrollará más adelante)

Timebox en Scrum

El timebox existe primero, para que no se pierda más tiempo del necesario en las diversas tareas, y también para aprender a negociar en otros términos y que la variable de negociación no sea el tiempo. Si no llegamos al final de un sprint, en vez de extender el tiempo, se entrega menos, y de esta forma es que se cambia la variable de negociación.

Nos volvemos más confiables, con un ritmo de trabajo sostenible, que ayuda a generar confianza. Todas las ceremonias son Timebox. El refinamiento del producto backlog es una actividad continua, y es por eso que no tiene un tiempo definido, sino más bien esta expresado en términos porcentuales sobre la duración del sprint.

Esto se relaciona con la triple restricción porque se deja fijo el tiempo.



Definition of Ready (DoR) vs Definition of Done (DoD)

El DoR es un criterio que define cuando una historia está lo suficientemente bien formulada como para poderse implementar. Puedo decir, por ejemplo, que cada user story debe tener un prototipo, y el DoR de esa user story nos dirá “Se ha definido el prototipo para la user story”

El DoD es un criterio que nos dice cuando una historia está lo suficientemente bien implementada como para entrar a la sprint review, donde se lo mostrara al cliente, y ahí se determinara si entra a producción o no.

En el DoD se arma un checklist de todo lo que se debe cumplir para entrar en la review.

Definición de Hecho (DONE)	
❑	Diseño revisado
❑	Código Completo
❑	Código refactorizado
❑	Código con formato estándar
❑	Código Comentado
❑	Código en el repositorio
❑	Código Inspeccionado
❑	Documentación de Usuario actualizada
❑	Probado
❑	Prueba de unidad hecha
❑	Prueba de integración hecha
❑	Prueba de sistema hecha
❑	Cero defectos conocidos
❑	Prueba de Aceptación realizada
❑	En los servidores de producción

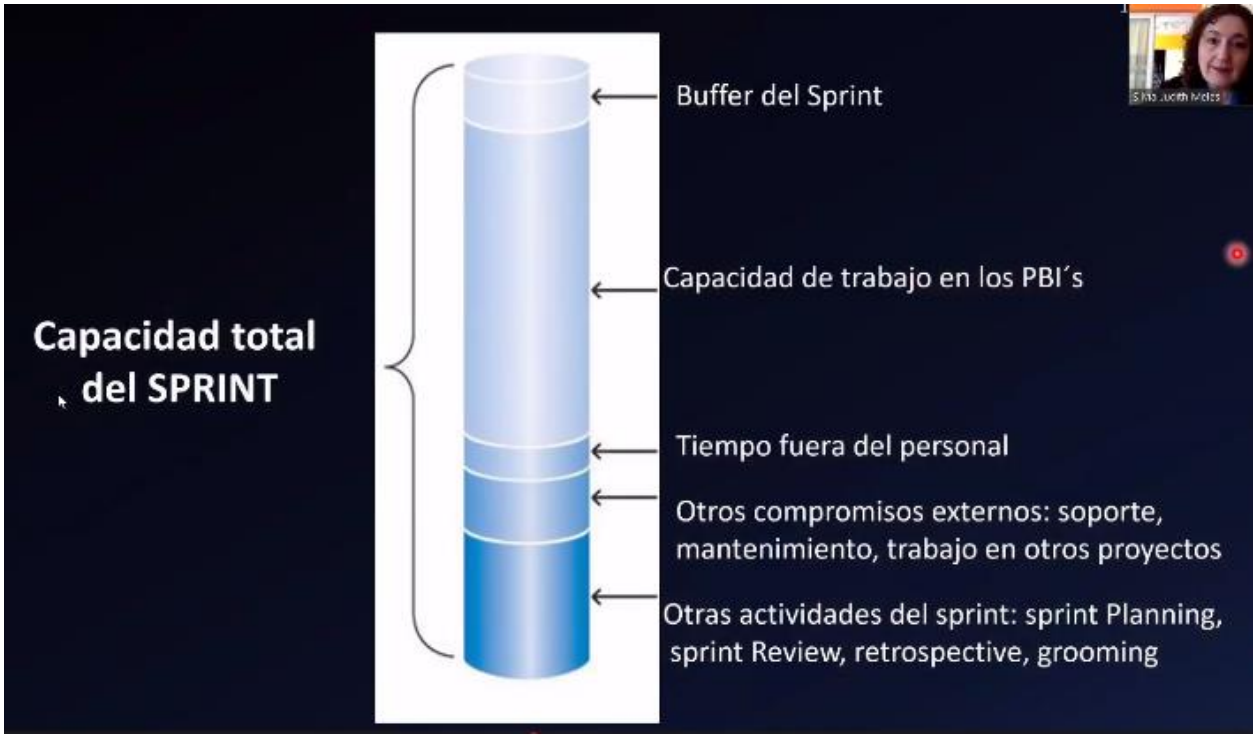
Capacidad del Equipo en un Sprint

La capacidad es una de las métricas que utiliza SCRUM, y la única que se estima y que se usa para ver cuanto compromiso de trabajo el equipo puede asumir en un determinado sprint.

Una de las cosas que se hacen en la Sprint Planning es determinar la capacidad del equipo, y entonces, teniendo en cuenta esa capacidad, se contrasta cuantas user storys del producto backlog pueden pasar al sprint backlog, y hasta donde nos podemos comprometer para un determinado Sprint.

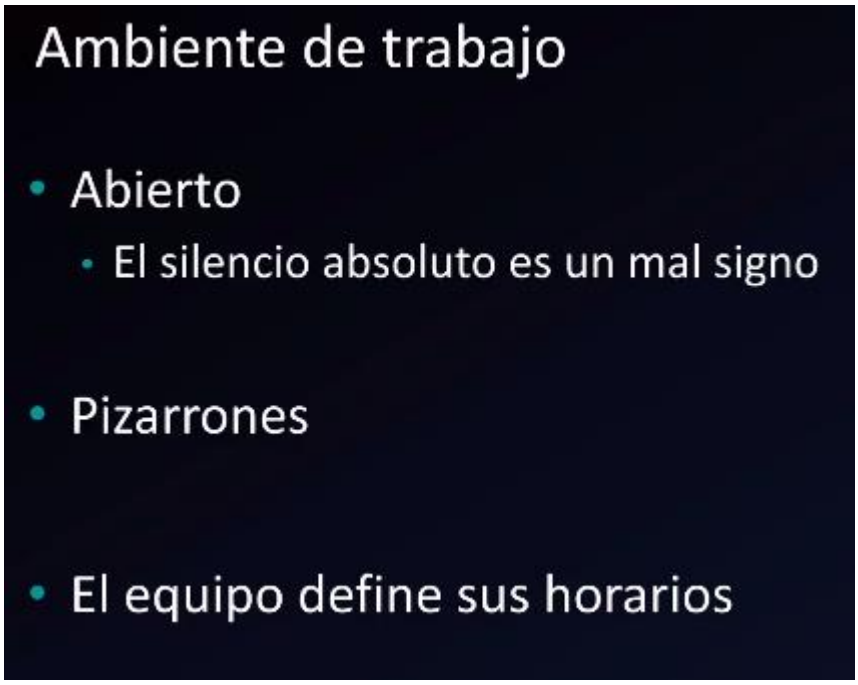
Se estima porque estamos empezando el sprint y estamos viendo cuantas horas (horas ideales/reales de trabajo) podremos dedicarle al trabajo en el sprint. La capacidad se puede estimar en horas ideales o Story Points.

Si un equipo es más maduro, entonces está capacitado para estimar en Story Points.



Persona	Días disponibles (sin tiempo personal)	Días para otras actividades Scrum	Horas por día	Horas de Esfuerzo disponibles
Jorge	10	2	4-7	32-56
Betty	8	2	5-6	30-36
Simón	8	2	4-6	24-36
Pedro	9	2	2-3	14-21
Raúl	10	2	5-6	40-48
Total				140-197

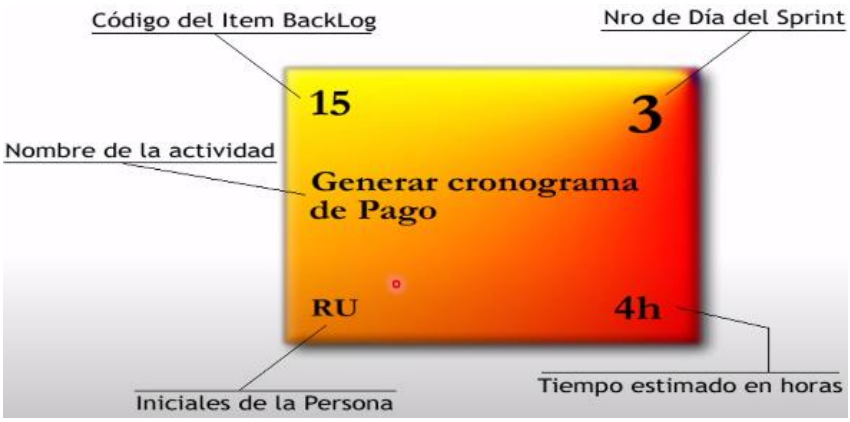
Entonces, decimos que la primera variable de definición es cuánto va a durar el sprint. La segunda es el cálculo de capacidad. Y finalmente se acuerda el objetivo y con esas 3 variables comenzamos a decidir qué cosas van a empezar a pasarse desde el producto backlog hasta el sprint backlog.



Herramientas

Taskboard

Hay una configuración de tablero básica que propone SCRUM, a partir de la cual se pueden hacer ajustes o variaciones según el equipo.



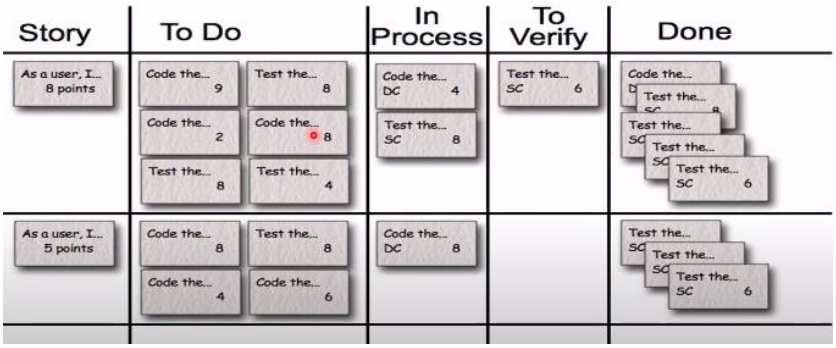
Lo mejor es tener personas multidisciplinares ya que caso contrario, una característica del sprint backlog tiene que pasar por muchas manos para poder terminarse y la configuración del tablero para poder tener visibilidad debe ser distinta a la básica recomendada por SCRUM.

Scrum recomienda estimar en Puntos de Historia ya que son estimaciones sobre el producto, y no sobre el trabajo (estimación en horas).

La configuración básica del tablero de SCRUM tiene solamente 3 columnas:

- Historias comprometidas
- In Process (o Doing, lo que se está haciendo)
- Done (lo terminado)

Cuando un empleado toma una tarea y la termina, la pasa a “Done”.



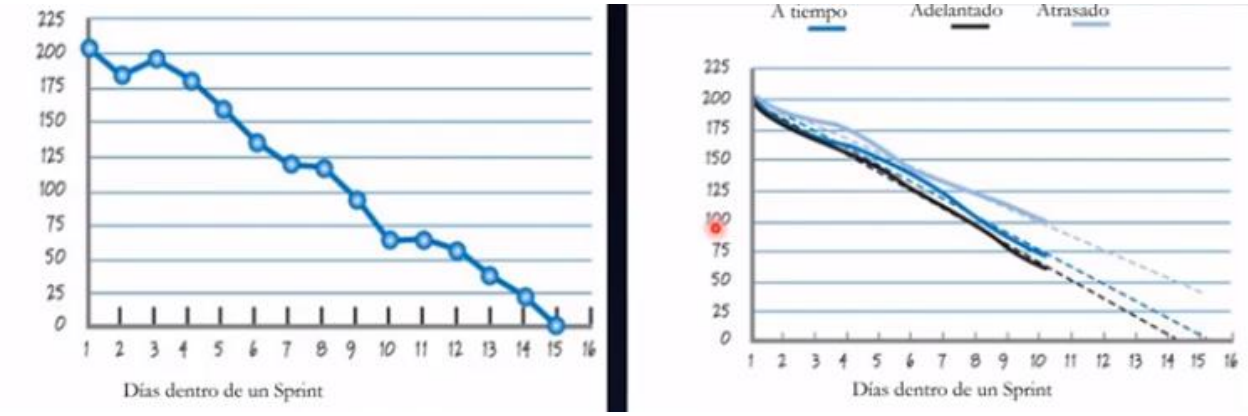
Cuando se tiene un tablero con pocas tareas, es porque son muy pesadas y vienen de una user story estimada con muchos puntos de historia.
Al trabajar con historias mas pequeñas, se tienen más historias, pero con granularidad mas fina, y una mayor posibilidad de poder maniobrar y asignarlas de forma que se puedan terminar antes del fin del sprint.

Sprint Burndown Charts

Son gráficos que nos sirven para la visualización del trabajo, en el eje “y” se muestran los puntos de historia restantes y en el eje “x” se colocan los días.

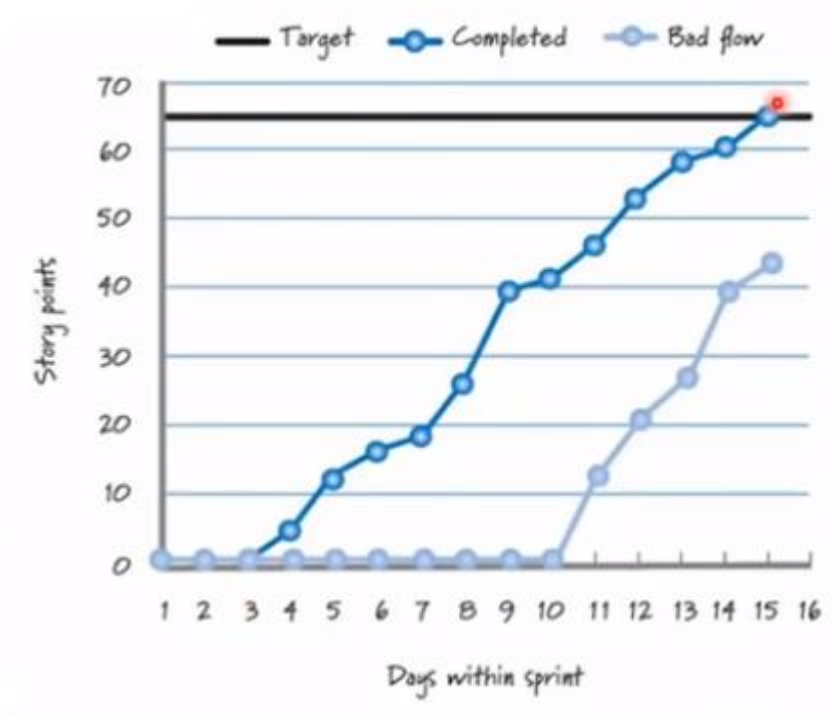
No conviene hacer esto con horas de trabajo (a pesar de que se puede hacer y se hace), ya que no es representativo del esfuerzo o logro obtenido (puedo pasar 8 horas tomando mate).

Los Burndown Charts sirven para calcular velocidad y se descartan al final del sprint.



Sprint Burnup Chart

Estos gráficos no se descartan al final del sprint, son permanentes ya que van haciendo un seguimiento de como se va trabajando en el producto a lo largo de los sprints.



Niveles de Planificación

No hay ningún producto de Software que se pueda terminar en un Sprint, es por ello por lo que es necesario que haya distintos niveles de planificación, donde el mas chiquito de los niveles es la daily, en el que nosotros sincronizamos y vamos viendo la situación de necesidad de hacer cambios, ajustes o adaptaciones en la diaria para poder cumplir con el compromiso.

El siguiente nivel que se conoce es el de iteración (que se corresponde con Sprint Planning). Cada espacio de tiempo que comienza termina y genera un incremento se llama iteración, que en Scrum se llama Sprint.

El que sigue es la planificación de Release (versión de producto que tiene un conjunto de características que se librean, o sea, se ponen en producción), que implica definir cuantos sprints voy a necesitar para poder entregar la cantidad de características que hemos definido que integran el reléase.

Lo que sigue es la planificación de producto, que es la sumatoria de los releases que voy a ir generando a lo largo del tiempo. Implica tomar decisiones funcionales y no funcionales e ir definiendo en que reléase se va a ir entregando cada característica.

Mas arriba, tenemos la planificación de portfolio, que se refiere a los diversos productos y cuales se van a ir priorizando, cuales van a ser descontinuados.

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 entero, 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 entero	Que aspectos entregar en el siguiente sprint	Objetivo del Sprint y Sprint Backlog
Día	Diaria	Scrum Master y Equipo de Desarrollo	Cómo completar lo comprometido	Inspección del progreso actual y adaptación a la mejor forma de organizar el siguiente día de trabajo

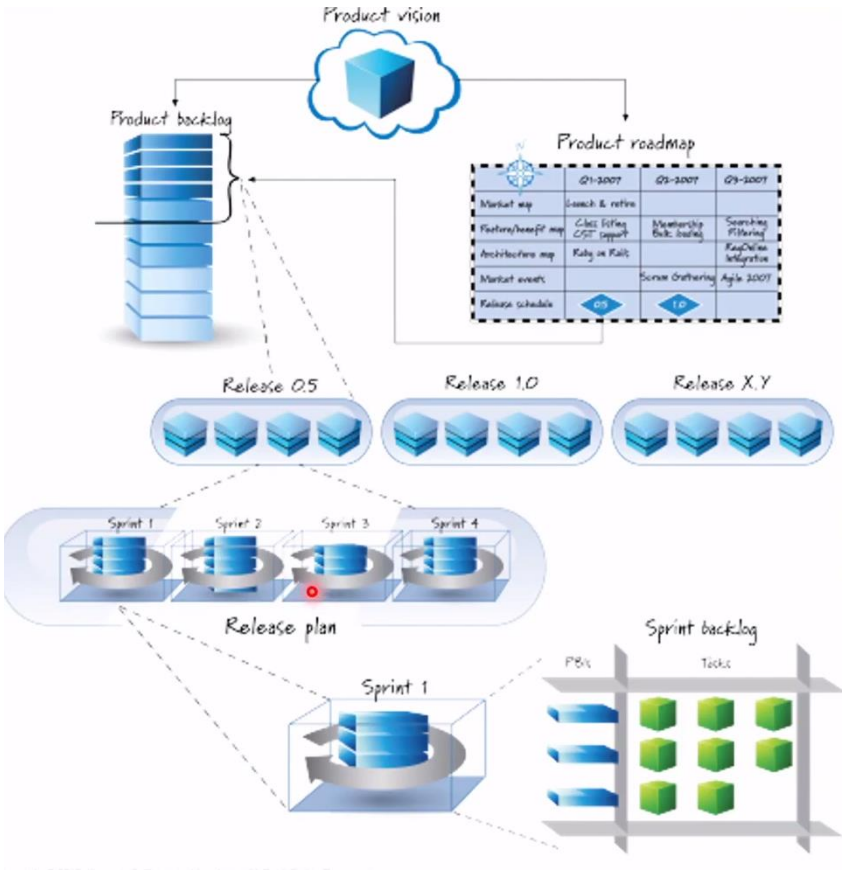
Qué y Cuando Estimar

Dependiendo del nivel de granularidad que estemos manejando, hay recomendaciones de que usar para estimar.

En el caso del Portfolio, se recomienda estimar en talles de remeras.

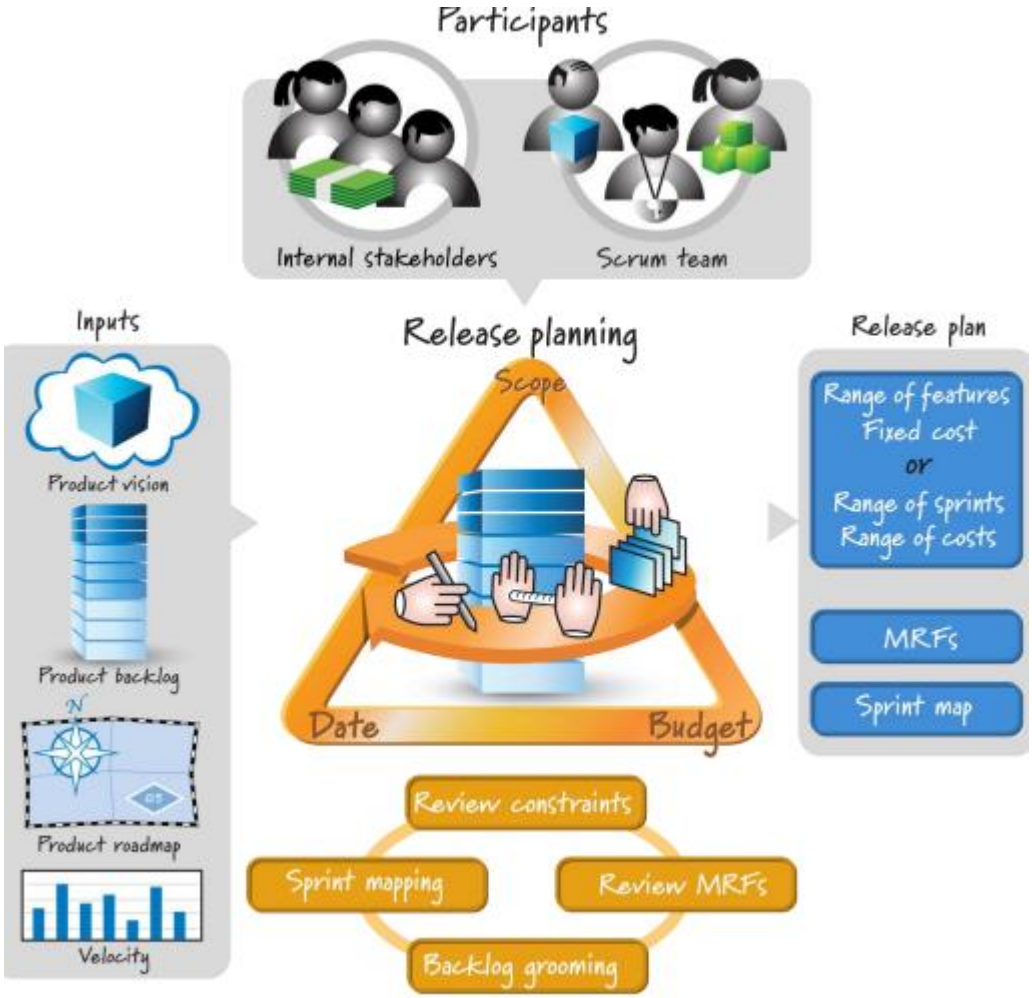
En Product Backlog hay una tendencia a estimar por Story Points, aunque antes se decía que podía no estar estimado o estar estimado en talles de remera. Mientras mejor estén priorizados los ítems del producto backlog, más fácil será la priorización.

En el Sprint Backlog se utilizan las historias con el DoR, estimadas en puntos de historia. Después el equipo decide si quiere dividir las historias en tareas o no. Si es así, las tareas se estiman en horas ideales.

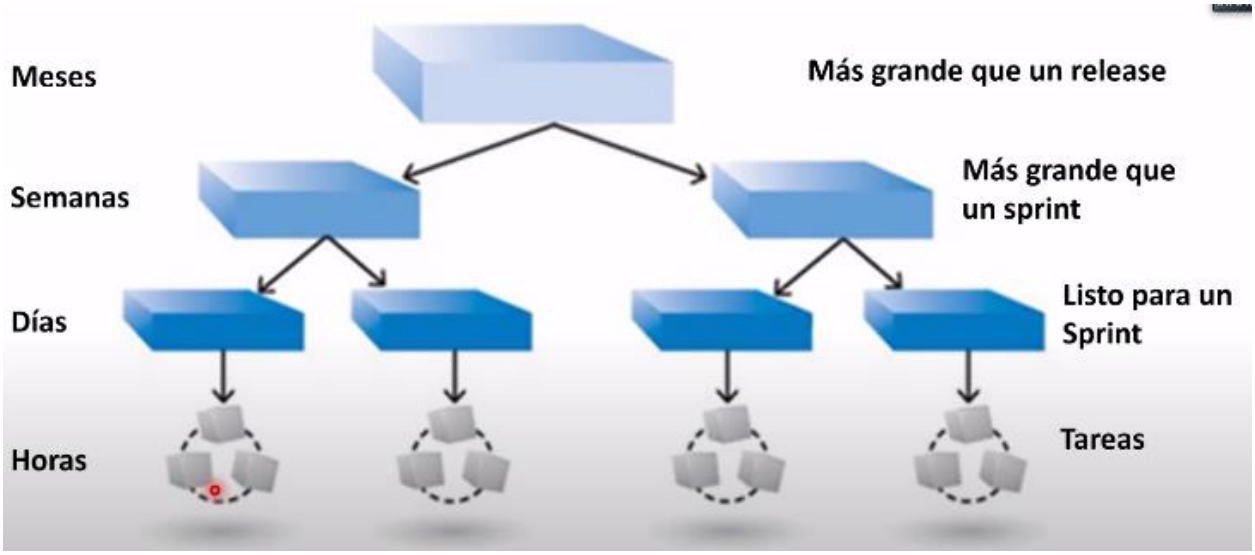


Planificación del Release

La salida de la planificación del reléase son las características que quiero desarrollar (en un rango) , cuantos sprints voy a necesitar para terminar el release, cuanto va a durar el sprint, la capacidad estimada del eq2uipo, los objetivos y características de cada sprint y en función de eso se hace una planificación. Si cambia algo en el camino, se tiene que replanificar.



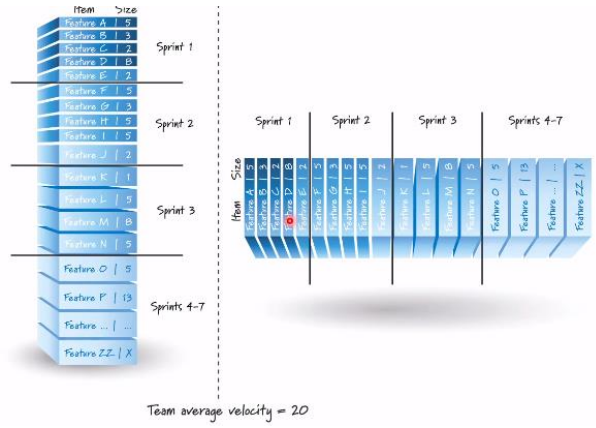
Distintos niveles de granularidad



Algo mas grande que una user story, no va a entrar en un sprint. Si son características muy grandes, es probable que ni siquiera entren en un reléase, y tengamos que repartirlo en varias releases.

Las tareas NO están en el producto backlog, sino en el sprint backlog.

En el producto backlog hay distintos niveles de granularidad.



Los primeros sprints tendrán las características mas prioritarias y si sé que mi equipo tiene una velocidad promedio de 20 puntos de historia, entonces sumo características hasta que sumo 20 puntos de historia (o menos, pero nunca más) y las asigno a un Sprint.

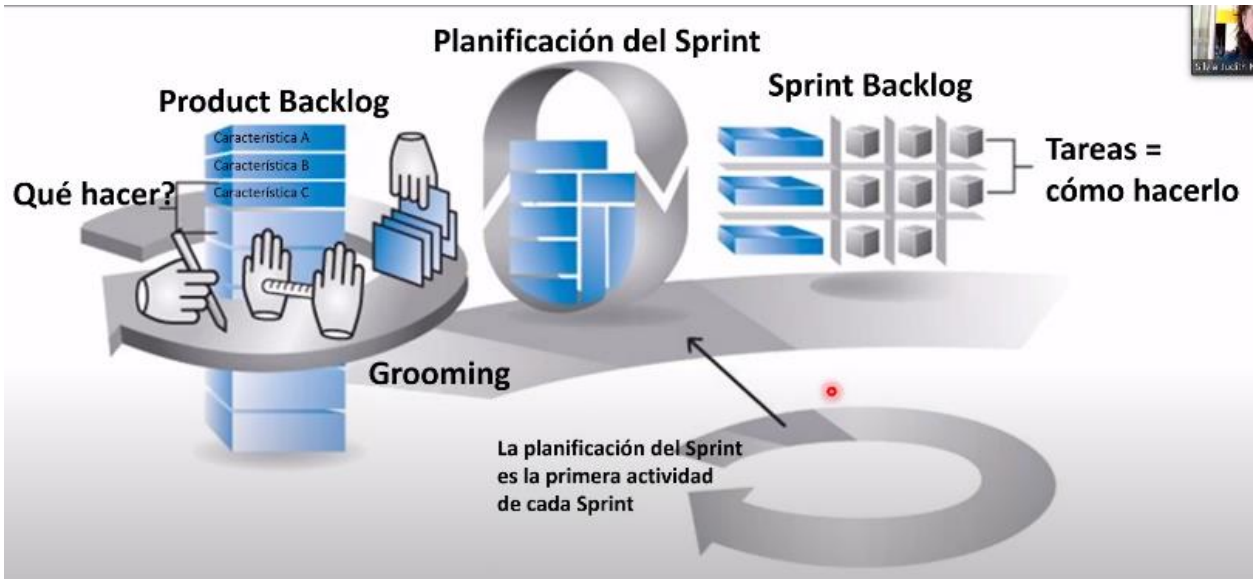
En los primeros sprints, no sabremos bien la velocidad del equipo, hasta que vayamos logrando experiencia en las sucesivas iteraciones. También iremos sabiendo si se tiene un desarrollo sostenible (cuando la velocidad del equipo es estable) o no

Cadencia de los Sprints

Hay varias formas de armar los releases, en algunas empresas se librean luego de terminar varios sprints, en otras hay una reléase por cada sprint, y en otras luego de cada feature

Planificación de Iteración: Sprint Planning

El proceso de Sprint planning debe tener como resultado la configuración del tablero, teniendo como entrada el objetivo del sprint, su duración, la capacidad estimada del equipo. El equipo dividirá en tareas cada user story, estimándolas en horas (si el equipo decide que así lo quiere). El proceso de planificación del sprint termina con el sprint backlog configurado.



Metricas utilizadas en Agile

- **Running Tested Features (RTF)**
    - **Se cuenta**
    - Cuenta cantidad de características (medidas en puntos de historia) funcionando que se han desarrollado en una iteración (sprint).
    - Se calcula solamente en puntos de Historia
    - Esta no se utiliza mucho ya que se reemplazó por velocidad.
- **Capacidad**
    - **Se estima**
    - Horas de trabajo disponibles por día * días disponibles de iteración = capacidad
    - También se puede calcular en RTF
- **Velocidad**
    - **Se cuenta**
    - Es una medida métrica del progreso de un equipo. Se calcula sumando el numero de Story Points (asignados a cada user story) que el equipo completa durante la iteración.
    - Se cuentan los Story Points de las iteraciones completas, no de las parcialmente completas
    - La velocidad corrige errores de estimación

Ejecutar practicas no significa que hayamos incorporado la filosofía y el pensamiento de Scrum. Somos agiles cuando incorporamos los valores agiles.

Unidad 3: Software Configuration Management (SCM)

¿Qué es la gestión de configuración de software?

Se define como una actividad paraguas porque se aplica en distintas partes del proyecto, no solamente en la gestión de las versiones. ***El propósito es mantener la integridad del producto de software.***
Es una disciplina que aplica dirección y monitoreo a administrativos para identificar y documentar, controlar cambios, registrar y reportar cambios.
No es una actividad, es una disciplina.
Específicamente, es una disciplina de soporte ya que ocurre transversalmente a lo largo de todo el proyecto pero más allá inclusive porque trasciende el proyecto para darle soporte al producto en todo su ciclo de vida. Desde el momento uno en el que nosotros estamos tratando de concebir al producto empezamos a generar productos de trabajo o artefactos o componentes que en la gestión de configuración tienen un nombre específico, llamado ***ítem de configuración***. Es por esta razón que se dice que es una "actividad paraguas".

Recordemos que software es cualquier producto de trabajo que sale de cualquier actividad dentro del ciclo de vida del proyecto y después dentro del ciclo de vida del producto.

Los ítems de configuración, que en general se gestan en el contexto de un proyecto, van conformando el producto.

Entonces, en una definición más formal decimos que la gestión de configuración de software 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 y verificar correspondencia con los requerimientos."* ***(ANSI/IEEE 828, 1990)***

¿Qué es la integridad del producto?
El producto para tener integridad debe:

- satisfacer la necesidad del usuario
- ser fácil y rastreable durante su ciclo de vida
- satisfacer los criterios de la performance
- cumplir con la expectativa de costo

Entonces, se dice que es transversal porque empieza desde el momento cero en el que esto empieza a gestarse y mientras el producto existe hay actividad de gestión de configuración, que es responsabilidad ***de todo el equipo***. Sin embargo, existen roles específicos como el rol del Gestor de configuración que tiene tareas adicionales como por ejemplo mantener la herramienta, marcar línea base, etc.

¿Cuál es el problema que se está intentando 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 tiene que tener su versión.

Cuando pensamos en software, ¿en qué pensamos?

Conjunto de:

- Programas
- Procedimientos
- Reglas
- Documentación
- Datos

La información tiene que ser:

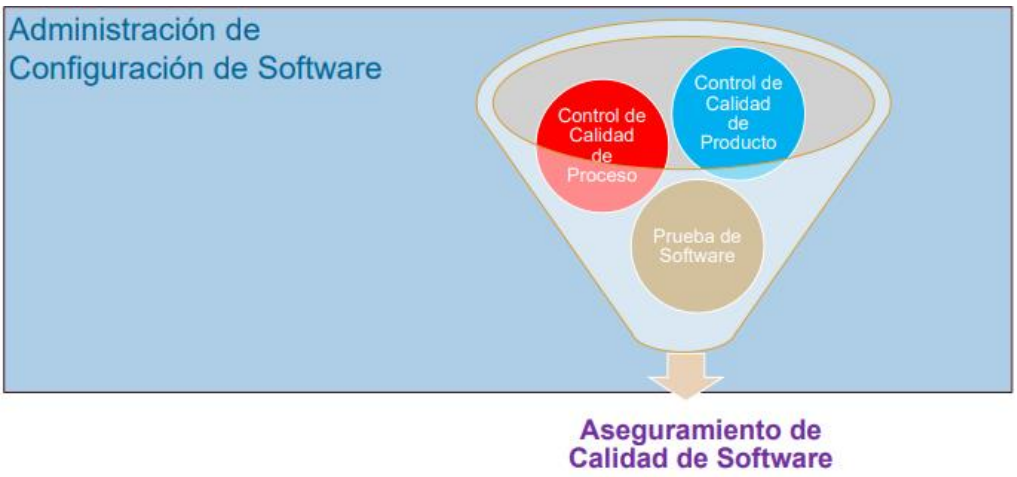
- estructurada con propiedades lógicas y funcionales.
- creada y mantenida en varias formas y representaciones.
- confeccionada para ser procesada por computadora en su estado más desarrollado

Cambios en el Software

Tienen su origen en:

- 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 mejora

Disciplinas de soporte del Software



La administración de configuración de software es un contenedor, es decir, es la disciplina que ayuda a que las otras disciplinas puedan funcionar y realizar su trabajo.

¿Por qué deberíamos gestionar la configuración?

Su propósito es establecer y mantener la integridad de los productos de software a lo largo de su ***ciclo de vida del producto***.
Involucra para la configuración:

- Identificarla en un momento dado
- Controlar sistemáticamente sus cambios
- Mantener su integridad y origen

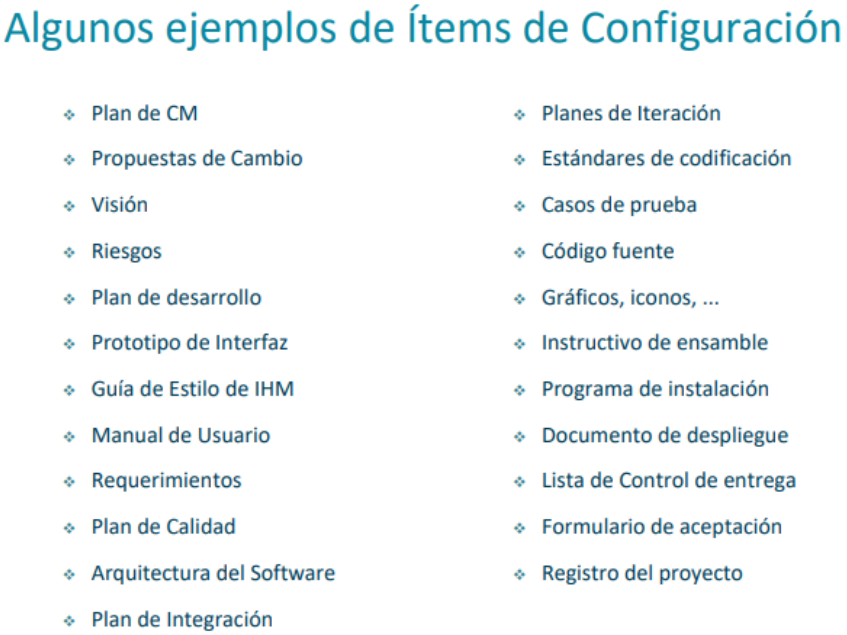
Problemas en el manejo de componentes

- 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

Algunos conceptos clave para la gestión de configuración de software

Ítem de configuración

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. Ej: un requerimiento en un archivo word, un excel, una base de datos, una US, una foto, código fuente, código ejecutable, documentos de diseño, etc.

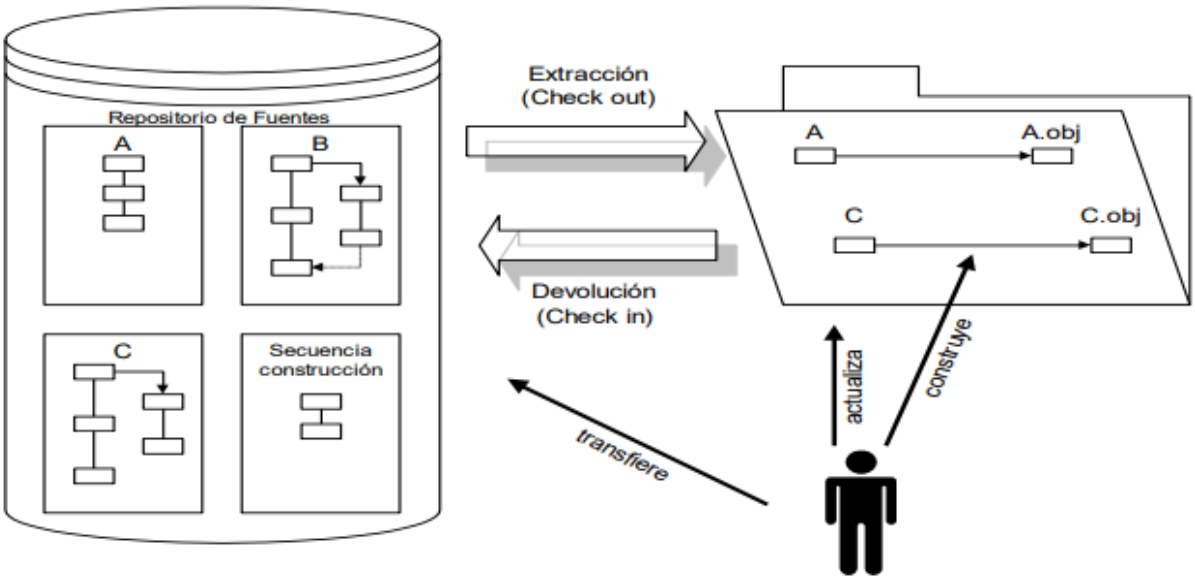


Repositorio

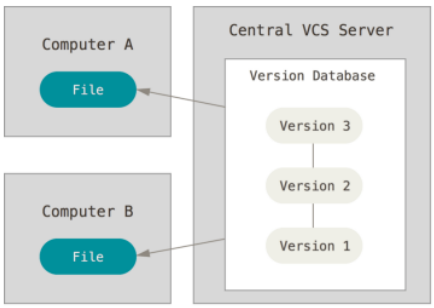
Es donde se mantiene la historia de cada ítem de configuración y el ítem. Es el contenedor de ítems de configuración. El repositorio tiene una estructura para mantener un orden y la integridad. El que tenga una estructura ayuda a la seguridad, los controles de acceso, las políticas de backup y todo aquello que se aplica sobre un repositorio. Además, el equipo comparte el repositorio por lo que no va a haber problemas en cuanto a situaciones donde no se pueda trabajar porque “solo fulanito accede a su carpeta y él tenía el trabajo”.
Lo ideal es tener un repositorio para un producto y tener todos los ítems de configuración que tiene ese producto que nos interesa controlar alojados en algún lugar de esa estructura del repositorio, con un esquema de configuración de permisos.
Entonces, un repositorio:

- Un repositorio de información conteniendo los ítems de configuración (ICs)
- Mantiene la historia de cada IC con sus atributos y relaciones.
- Usado para hacer evaluaciones de impacto de los cambios propuestos.
- Pueden ser una o varias bases de datos

Funcionamiento de un repositorio

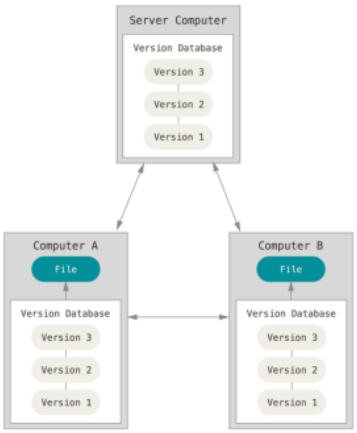


Repositorios centralizados



- Un servidor contiene todos los archivos con sus versiones.
- Los administradores tienen mayor control sobre el repositorio.
- Falla el servidor y "estamos al horno“.

Repositorios descentralizados



- Cada cliente tiene una copia exactamente igual del repositorio completo.
- Si un servidor falla sólo es cuestión de “copiar y pegar” .
- Posibilita otros workflows no disponibles en el modelo centralizado.

Línea base

Una línea base puede contener en un momento de tiempo un solo ítem de configuración o un conjunto de ítems.
La línea base está conformada por ítems de configuración que se toman como referencia, es decir, que esos IC han pasado todos los niveles de revisiones y aprobaciones que deberían haber pasado por lo que se los considera estables. Así, uno marca estos IC para identificar que esos IC forman parte de esa línea base.
La línea base debe tener una identificación unívoca, la línea base tiene que tener un nombre, una versión, una identificación única en un momento y tiene que indicar claramente cuáles son los IC con su estado que forman parte de esa determinada línea base.
No se debe confundir con la versión del producto.
Entonces, la línea base:

- Es una configuración que ha sido revisada formalmente y sobre la que se ha llegado a un acuerdo.
- Sirve como base para desarrollos posteriores y puede cambiarse sólo a través de un procedimiento formal de control de cambios.
- Permiten ir atrás en el tiempo y reproducir el entorno de desarrollo en un momento dado del proyecto.
- Aquello que no forme parte de la línea base lo podemos modificar con total libertad.

¿Para qué sirve una línea base? 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.

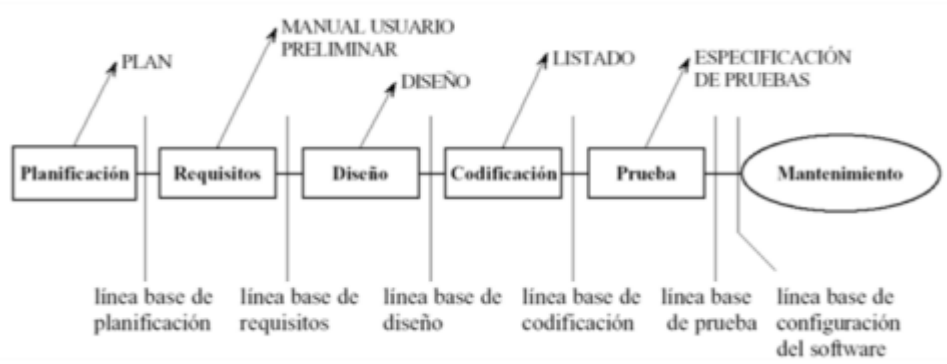
Las líneas bases pueden ser de dos tipos: de especificación u operacionales.

Línea base de especificación

Son de requerimientos, diseño. No poseen código, poseen información de la ingeniería del producto.

Línea base operacional

Son las líneas base que ya tienen código. Es la línea base de productos que han pasado por un control de calidad definido previamente.

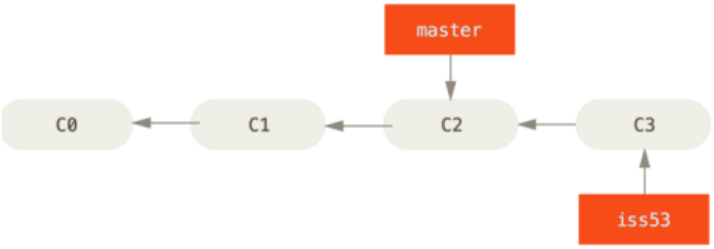


Ramas (branch)

Uno arma una bifurcación de la línea donde están los ítems de configuración que normalmente son la línea base y como no quiere ni se recomienda en ninguna buena práctica trabajar de accionar directamente sobre esos ítems que están en ***la rama principal*** que dependiendo de la herramienta de configuración algunos les llaman troncos.
La rama principal también es conocida como máster.
Podemos usar las bifurcaciones por ejemplo para tener una nueva versión del producto y una vez que terminemos esa nueva versión, recién ahí se integra a la rama principal modificando. La integración de las ramas bifurcadas con la rama principal se llama ***merge***.
Puede pasar la situación de que las ramas se descarten y nunca sean integradas con la rama principal.

Creación de ramas:

- Existe una rama principal (trunk, master)
- Sirven para bifurcar el desarrollo
- Pueden tener razones de creación con semántica
- Permiten la experimentación
- Pueden ser descartadas o integradas



Integración de ramas:

- La operación se llama ***merge***
- Lleva los cambios a la rama principal
- Pueden surgir conflictos (resolvemos con diff)
- Todas las ramas deberían eventualmente integrarse a la principal o ser descartadas

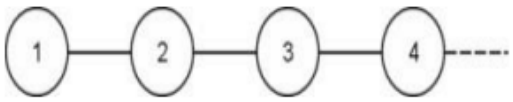
Configuración del software

Es un conjunto de ítems de configuración con su correspondiente versión en un momento determinado.

Análisis de impacto: es saber dónde me va a doler un determinado cambio y por consecuencia cuánto me va a llevar a hacerlo y cuánto nos va a costar en plata. (impacto del cambio)

Versión

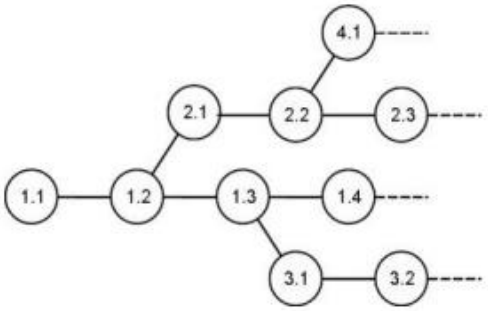
- La versión es un punto particular en el tiempo de ese ítem de configuración (es un estado).
- Una versión se define, desde el punto de vista de la evolución, como la forma particular de un artefacto en un instante o contexto dado.
- El control de versiones se refiere a la evolución de un único ítem de configuración (IC), o de cada IC por separado.
- La evolución puede representarse gráficamente en forma de grafo.



Evolución lineal de un ítem de configuración

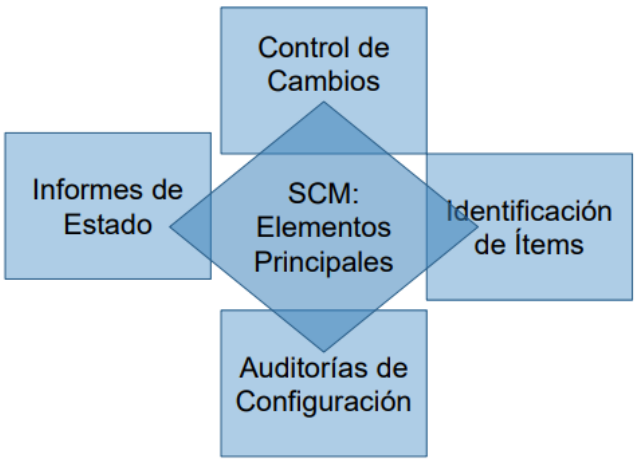
Variante

- 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.



Variante de un ítem de configuración

Actividades fundamentales de la administración de configuración de Software



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***.

Identificación de ítems

Corresponde a la identificación unívoca de los ítems, la estructura del repositorio, la ubicación de los ítems en el repositorio. Aquí se definen las reglas de nombrado (esquemas de nombrado) genéricos que suelen seguir ciertos estándares.
El rol del Gestor de Configuración es el rol que arma la estructura del repositorio y es el responsable de poner a disposición el repositorio.
Entonces, las actividades involucradas son:

- Identificación unívoca de cada ítem de configuración
- Convenciones y reglas de nombrado
- Definición de la Estructura del Repositorio
- Ubicación dentro de la estructura del repositorio

Tipos de ítems

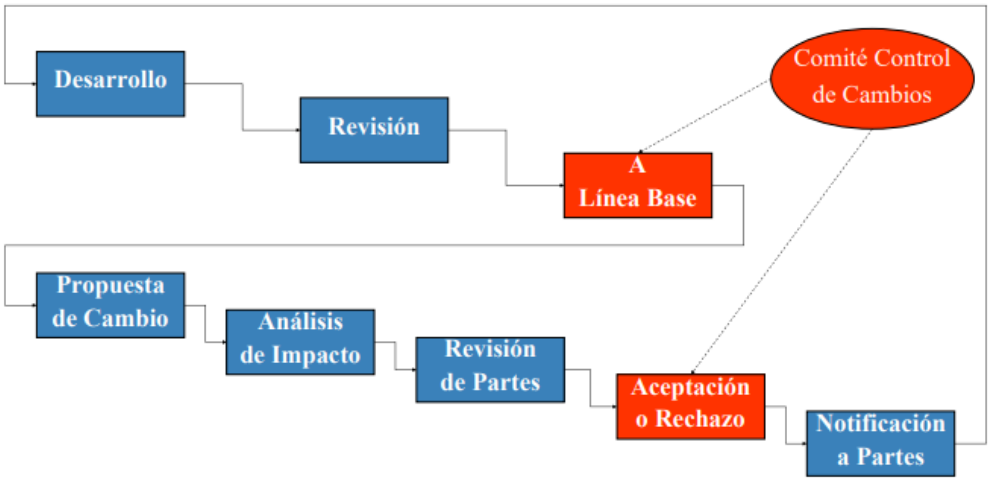
Cada tipo de ítem tiene un ciclo de vida distinto. Esto es lo que se hablaba al principio del resumen de ciclo de vida del producto y ciclo de vida del proyecto.
Entonces, cuando le asignamos a los ítems un lugar en el repositorio debemos tener en cuenta qué tipo de ítem es.
Por ejemplo: el documento de arquitectura no debe tener el nombre del proyecto porque el proyecto termina pero la arquitectura del producto trasciende.

El ciclo de vida más corto es el de el ítem de iteración.
Todo lo que tiene que ver con producto trasciende los proyectos donde se los crea.

Control de cambios

Está asociado a las líneas base. ***Busca mantener la integridad de las líneas base***.
Para cambiar la línea base, esta debe pasar por un proceso formal de control de cambios.
Es por esto que decimos que los ítems de configuración que ***no*** son línea base se pueden modificar sin tanto problema ni cháchara.

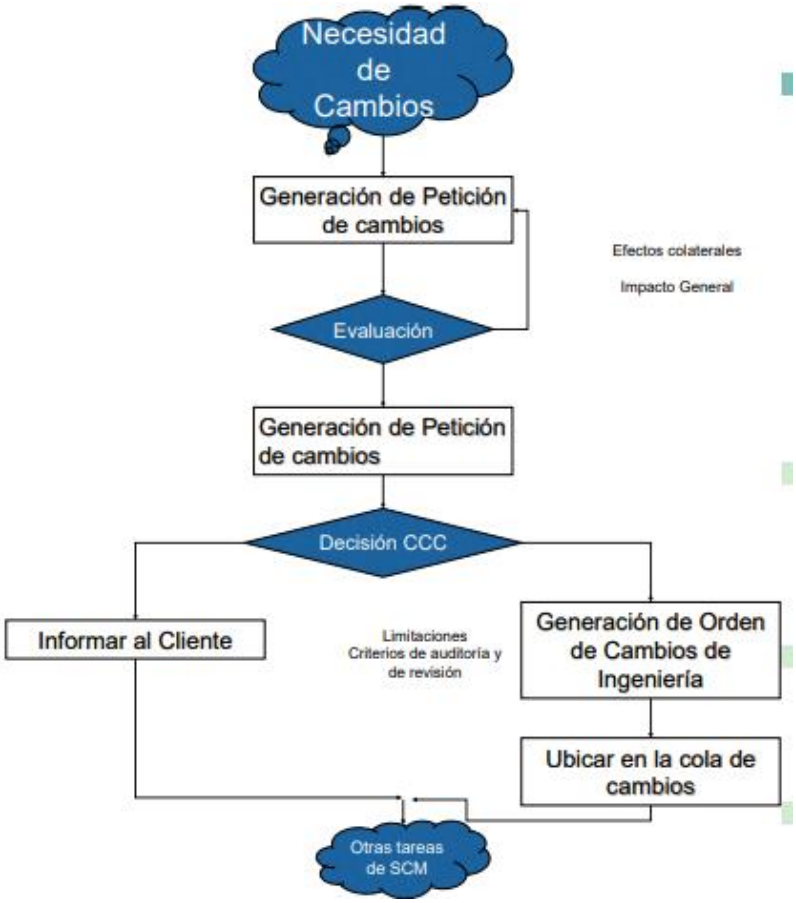
Proceso de control de cambios



El cambio se debe autorizar y una vez que se realiza el cambio se debe verificar que efectivamente el cambio sea el que se estableció previamente y se autorizó.
Acá se definen y se mantienen las líneas base a lo largo del tiempo.
Se realiza también un análisis de impacto o evaluación de impacto que determina si el comité del control de cambios va a autorizar un cambio o no .

El control de cambios:

- Tiene su origen en un Requerimiento de Cambio a uno o varios ítems de configuración que se encuentran en una línea base.
- Es un ***Procedimiento formal*** que involucra diferentes actores y una evaluación del impacto del cambio.



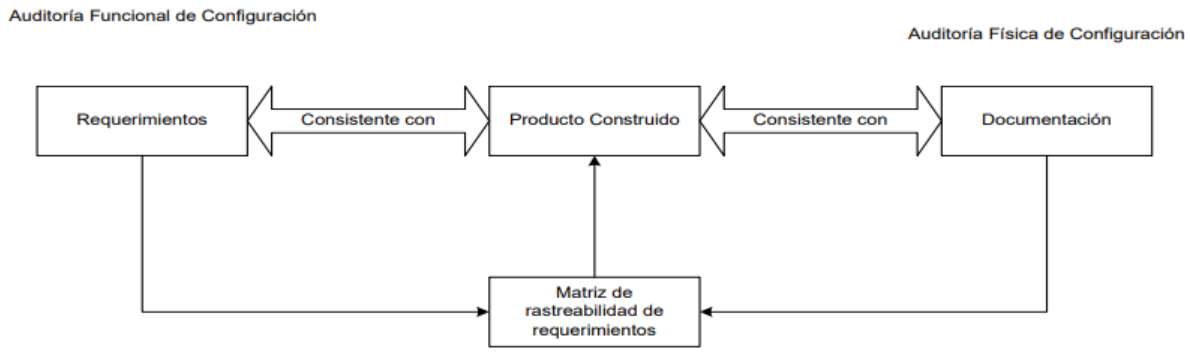
Comité de control de cambios

Está formado por representantes de todas las áreas involucradas en el desarrollo. Son referentes del equipo que está trabajando en el desarrollo del producto:

- Análisis, Diseño
- Implementación
- Testing
- Otros interesados

Auditorías de configuración de Software

Las auditorías necesitan un plan. Se realizan 2 auditorías: física y funcional.



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.
- Hace ***verificación***
- Se encarga de la integridad del repositorio
- Controla que el repositorio esté, que esté alojado en el lugar donde dijimos que iba a estar, que los IC respeten el esquema de nombrado, que estén guardados donde se dijo que iban a estar guardados.
- Lo que ***se audita es una línea base.***

Auditoría funcional de configuración (FCA)

- Hace ***validación***.
- 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.
- Se fija que el producto sea el producto correcto, es decir, que los IC tienen que responder a lo que se pidió en los requerimientos.
- La auditoría se hace por muestreo, es decir, se elige un requerimiento al azar y se analiza su IC.
- Antes de comenzar, pregunta si está disponible el reporte de la auditoría de configuración física. Ya que ***si la auditoría física no sale bien, la auditoría funcional no se hace***.

El auditor es alguien externo al equipo porque una auditoría es una revisión independiente y objetiva sobre el producto.
Primero se realiza la auditoría física y luego la funcional.
La auditoría toma un plan y controla una línea base y en función de ese proceso de control se obtiene un informe de auditoría que muestra todas las desviaciones que se detecten en el momento en el que se hace la auditoría. Después, el equipo tiene que hacer un plan de acción e informar cómo piensa resolver las cosas que el auditor le mostró que son desviaciones.

Sirve a dos procesos básicos: la validación y la verificación:

- ***Validación:*** el problema es resuelto de manera apropiada para 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.

Informes de estado

El propósito es generar reportes para mantener un registro de la evolución, ***para dar visibilidad para la toma de decisiones***. Sirve para que los involucrados se enteren de un estado de situación de la gestión de configuración.
El ***reporte básico es el inventario***, que lista todos los IC que tiene el repositorio con su estado en un momento de tiempo o listar cuántas solicitudes de cambios se atendieron en

un determinado periodo de tiempo o listar el contenido de una línea base o listar cuantas líneas base hay para determinados proyectos o saber quién autorizó un determinado cambio a una línea hace y cuando se implementó.
La mayoría de los reportes se obtienen de manera automática.

Registro e informe de estado:

- Se ocupa de mantener los registros de la evolución del sistema.
- Maneja mucha información y salidas por lo que se suele implementar dentro de procesos automáticos.
- Incluye reportes de rastreabilidad de todos los cambios realizados a las líneas base durante el ciclo de vida.

Algunas preguntas que podría responder:

- ¿Cuál es el estado del ítem?
- ¿Un requerimiento de cambio ha sido aprobado o rechazado por el CCB?
- ¿Qué versión de ítem implementa un requerimiento de cambio aprobado (saber cuál es el componente que contiene la mejora)?
- ¿Cuál es la diferencia entre una versión y otra dada?

Plan de gestión de configuración

El plan debe contener respuestas a las preguntas de cómo voy a hacer las cuatro actividades básicas de la gestión de configuración, cuanto a auditorías y de qué tipo y en qué momento voy a hacer, que informes de estado voy a sacar, la estructura del repositorio, cómo se conforma el comité de control de cambios, etc.

Existe un template de la IEEE para el plan de gestión de configuración que tiene un estándar para definir los planes .

¿Qué debería incluir el plan?:

- 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

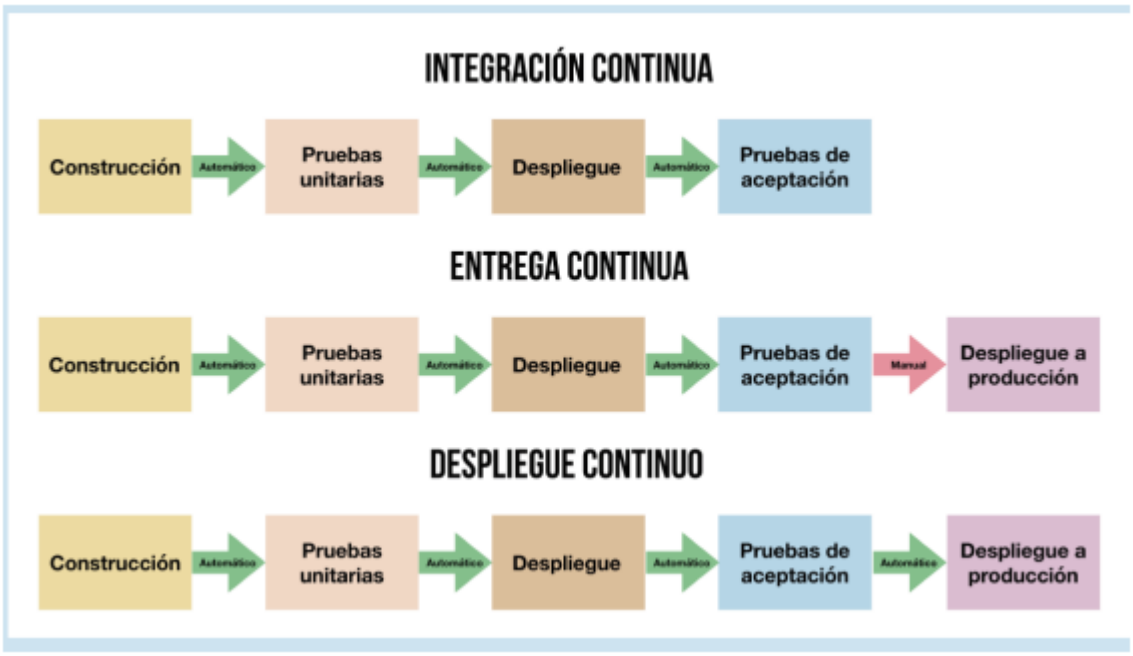
¿Por qué necesito el plan para poder auditar? Porque el plan es que contiene la estructura del repositorio y donde yo voy a guardar cada cosa, entonces, para auditar el repositorio tengo que tener algo que me diga cuál es “el deber ser” para contrastarlo con la realidad y entonces saber si funciona o no funciona. Cualquier proceso de auditoría en un proceso de control para poder comparar, entonces, se compara la línea base contra lo definido en el plan de gestión de configuración.

Evolución de la gestión de configuración de software

La gestión de configuración ha permitido que los equipos evolucionen en su forma de construir software y que traten de hacer un proceso que sea cada vez más productivo y

para lograr esa productividad tratan de automatizar la mayor parte que se pueda de los procesos. La automatización ayuda a evitar errores.
La evolución es lo que se conoce también como prácticas continuas.

Integración, entrega y despliegue (prácticas continuas)



Estas disciplinas son la evolución de la gestión de configuración de software, si no esta la base de gestión de configuración como primera capa ninguna de estas cosas es posible. Consiste en llevar la evolución de la gestión de configuración a un nivel de automatización que nos permite ir implementando cada una de esas prácticas(integración, entrega y despliegue), que en términos generales se les llama prácticas continuas.

Todas estas metodologías ágiles, a diferencia de lo que todo el mundo cree, formalizan mucho y son muy rigurosas respecto a la calidad y a la forma en la que se va a construir el código. Se le da muchísima importancia al testing y que esté lo más automatizado posible.

La ***integración continua***, que es lo primero que se hace, es que cada desarrollador en su entorno trabaja, realiza pruebas unitarias (en la medida de lo posible, automatizadas) desarrollando con algo que se llama ***TDD (desarrollo conducido por testing)*** y cuando terminó ese componente de código y lo probó y sabe que funciona, lo sube a un repositorio de integración. Hay pruebas de integración que se van corriendo automáticamente para tener una versión del producto que permanentemente funciona.

Así, la versión del producto está en condiciones de ir a las pruebas de aceptación de usuario sin problemas.

La ***entrega continua*** suma la automatización de las pruebas de aceptación y entonces el producto ya está listo para desplegarlo a producción.

Cuando el último paso también es automatizado, tenemos finalmente el ***despliegue continuo***, que consiste en poner en el ambiente de producción del usuario final el producto.

La diferencia entre estas tres prácticas es el nivel de automatización de las pruebas.

Gestión de configuración de software en ambientes ágiles

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.

El punto de discusión entonces, es cómo se va a realizar la gestión de configuración en el ambiente ágil.

¿Qué actividades de la gestión de configuración usamos en ágil?

- ***Auditorías de configuración*** es una actividad que ***podríamos no usar*** si el equipo ágil lo considera así, es decir, no necesitan opinión externa. Es opcional y depende del equipo usarla o no.
- Las otras 3 actividades van si o si, ni se discute.

SCM en Ágil

- Sirve para 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.