Unidad N°1: Ingeniería de Software en Contexto

Software: Conjunto de programas y su documentación.

Tipos de Software:

- **De Sistemas:** Se encarga de controlar y gestionar los recursos de hw de una computadora, así como brindar una interfaz para la interacción de los usuarios.
- **Utilitarios:** Referido al subconjunto de software del sistema que brinda utilidades/herramientas adicionales para mejorar la funcionalidad del SO y el rendimiento de la computadora (hw).
- **De aplicaciones:** Es utilizado para realizar tareas específicas en una computadora, por lo que está enfocado en satisfacer las necesidades de los usuarios → Enfoque de la materia.

Para crear software requerimos de un trabajo de raíz intelectual, por lo que depende de la personalidad e intelecto de los miembros del equipo que lo crean.

La **ingeniería en software** es la construcción de múltiples versiones de software realizadas por múltiples personas, se divide en tres tramas: Disciplinas **técnicas**, disciplinas de **gestión** y disciplinas de **soporte** (SCM, métricas, ect.)

Software ≠ Manufactura:

- El software es menos predecible.
- No hay producción en masa, casi ningún producto es **igual a otro**.
- No todas las fallas son errores.
- No se gasta.
- No está gobernado por la física.

Problemas comunes en el Desarrollo

- La versión final del producto no satisface las necesidades del cliente.
- No es fácil extenderlo y/o adaptarlo. Agregar más funcionalidad en otra versión es casi una misión imposible.
- Mala documentación.
- Mala calidad.
- Más tiempos y costos que los presupuestados.

Factores que influyen en el éxito de un producto:

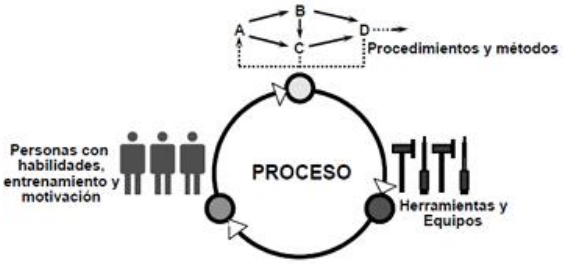
- Involucramiento del usuario – 15,9%.
- Apoyo de gerencia – 13%.
- Enunciado claro de requerimientos – 9,6%.
- Planeamiento adecuado – 8,2%.
- Expectativas realistas – 7,7%.
- Hitos intermedios – 7,7%.
- Personas involucradas competentes – 7,2%.

El proceso de Software

Es un conjunto estructurado de actividades para desarrollar un sistema de software. Las cuales varían dependiendo de la organización y el tipo de sistema a desarrollarse, por lo que deberá ser explícitamente modelado si va a ser administrado.

Definición de un Proceso de Software

- **Proceso:** Secuencia de pasos ejecutados para un propósito dado – *IEEE.*
- **Proceso de Software:** Conjunto de actividades, métodos, prácticas y transformaciones que la gente usa para desarrollar/mantener software y sus productos asociados (Sw-CMM).



Procedimientos y métodos: Definición de cómo se llevarán a cabo las actividades de desarrollo (Documentación).

Personas: Con habilidades, entrenamiento y motivación.

Herramientas y equipos: Materiales que se necesitan para llevar a cabo el proceso.

Las personas hacen uso de las herramientas, equipos, procedimientos y métodos, de forma que al partir de ellos se produzca un producto de software. Dentro del proceso se encuentran definidas las responsabilidades, actividades y herramientas a partir por las cuales las personas transforman los requerimientos en software.

Tipos de procesos: Dentro de la industria existen dos tipos de procesos: **Definidos y Empíricos.**

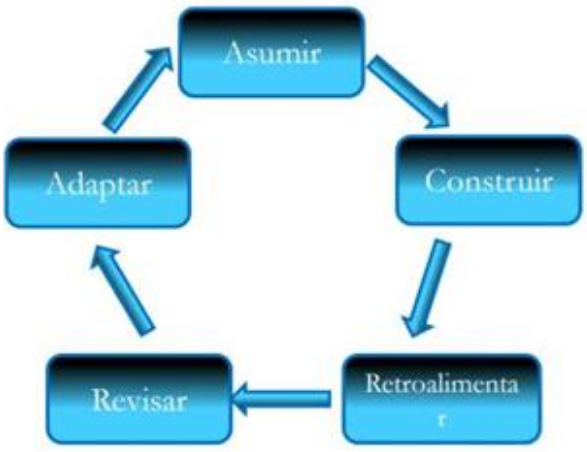
Procesos Definidos: Inspirados en las líneas de producción. Son procesos que plantean la necesidad de tener definidos ampliamente paso a paso que se realizará en cada momento. De esta forma, desagrega un conjunto de actividades y se define de forma explícita cuáles son las tareas, quien las realizará y en qué momento, a su vez, cuáles serán las entradas y salidas de cada tarea y los artefactos que se generarán.

- Asume que podemos **repetir** el mismo proceso una y otra vez, indefinidamente, y obtener los mismos resultados.
- Ejemplos: PUD (Proceso Unificado de Desarrollo), RUP (Rational Unified Process)

Procesos Empíricos: Son procesos que se basan enla experiencia, lo cual es complejo de explicar debido a que está se basa en algo anterior para hacer algo nuevo, es decir algo que puede ser adquirido con el paso del tiempo.

- Surgidos en contraposición de los procesos definidos.
- Asume procesos complicados con **variables cambiantes.**
- Al repetir el proceso, se pueden obtener resultados diferentes.
- La administración y control es realizado mediante inspecciones frecuentes y adaptaciones.
- Trabaja bien con procesos **creativos y complejos.**
- Ejemplo: Scrum.

Patrón de Conocimiento en Procesos Empíricos:

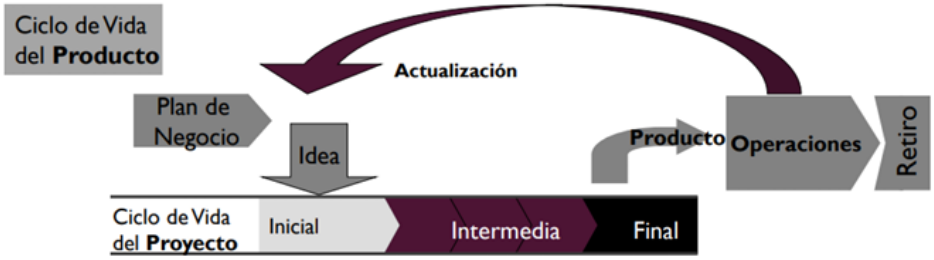


Se parte de una **Hipótesis** (**ASUMIR**), luego se comienza con la construcción (**CONSTRUIR**) y en base a algunos puntos que se puedan inspeccionar y analizar se toman mediciones, donde sacamos información y haciendo retrospectiva (**RETROALIMENTAR**) revisamos (**REVISAR**) el resultado de la construcción y contrastando la hipótesis inicial con el resultado del proceso, podemos ir adaptando el proceso y seguir construyéndolo (**ADAPTAR**).

Está basado en una concepción que necesita repetición, por lo que dentro de los procesos empíricos se da uso a un **ciclo de vida** específico (iterativo e incremental).

Ciclos de vida: Es la serie de pasos a través de los cuales un producto o un proceso progresa.

- Lo que indica que definirá las etapas (fases) y el orden de c/u de ellas. No indica que hacer, ni quien lo hará, ni las herramientas o procedimientos.



Ciclo de vida de un proyecto de software: Es una representación de un proceso. Grafica una descripción del proceso desde una perspectiva particular. Por lo que su ciclo de vida empieza y termina.

Ciclo de vida de un producto de software: El Ciclo de vida de un proyecto termina cuando genera un producto, cuyo ciclo de vida no va a terminar, sino que va a mantenerse mientras el **objeto exista.** Lo que quiere decir esto es que el producto necesitará mantenimiento una vez "termino" de desarrollarse, y se mantendrá bajo este proceso de mantenimiento hasta que sea desechado, donde podremos decir que su ciclo de vida llega al final.

Ciclos de vida básicos: Existen tres ciclos de vida básicos.

- **Secuencial:** Basados en la ejecución de una etapa tras etapa sin retorno. Algunas pueden llegar a devolver información (retroalimentación). Ejemplo: Ciclos de vida en Cascada.
- **Iterativo - Incremental:** Realiza iteraciones para ganar información, la cual se convertirá en un incremento que posteriormente será incorporado en el sistema.
- **Recursivo:** Utilizado en casos particulares (proyectos de alto riesgo). Basado en tomar una característica específica y en una iteración centrarse en esa funcionalidad, en la siguiente la otra, y así sucesivamente. Actualmente en desuso por ser inútiles y poco funcionales.
    - **Ejemplos:** Ciclo de vida espiral, ciclo de vida en B.

Dentro de los procesos definidos podemos encontrar cualquier ciclo de vida, mientras que en los procesos empíricos funcionan con el ciclo de vida iterativo e incremental, lo cual se da ya que les permite adaptar y mejorar el proceso, a su vez en cada iteración poder retroalimentarse y generando la experiencia que requieren para realizar la adaptación de esté.

*"El **proceso** es una implementación del **ciclo de vida** que tiene un objetivo que lo guía, que es la creación de un **producto.**"*

Las 4 P (Proceso – Proyecto – Producto - Personas): La disciplina de Ing. En software se mueve básicamente en tres dimensiones (proceso, proyecto y producto), pero debemos considerar a las personas como la cuarta dimensión, ya que son muy importantes y un factor clave para el éxito.

Personas: Son todas aquellas involucradas en el proyecto.

Proceso: Es el conjunto de actividades, es decir que tenemos que hacer, para hacer software (Requerimientos), dentro de este proceso encontramos los anteriormente mencionados como empíricos y definidos.

Proyecto: Es una unidad de trabajo/gestión del trabajo, siendo el medio por el cual se organiza el trabajo, recursos y personas. Utiliza los ciclos de vida, en donde se indicará como llevar a cabo el proceso elegido. Una vez finalizado el ciclo de vida del proyecto se obtiene como resultado el producto. Para ello deberá cumplir ciertas características.

Producto: Son versiones de software generadas para entregarle al cliente.

Unidad N°2: Gestión Lean-Ágil de Productos de Software

Características de un Proyecto

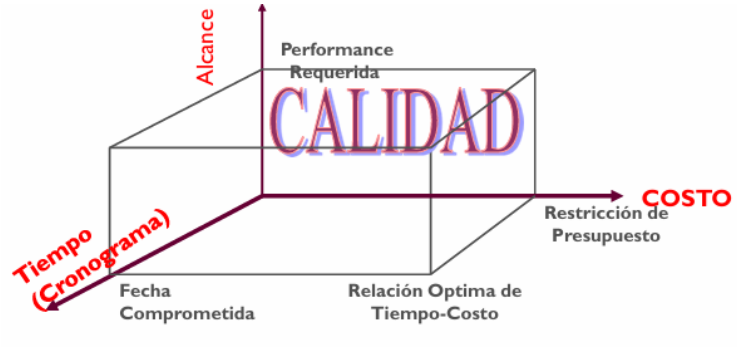
- **Orientación a objetivos:** Pueden ser un producto/servicio único. Cada versión de un producto de software es única, al ser diferentes proyectos.
  - Están dirigidos a obtener resultados y se refleja mediante objetivos.
  - Los objetivos guían al proyecto.
  - Los objetivos no deben ser ambiguos.
  - Un objetivo claro no alcanza, debe ser también alcanzable.
  - El proyecto tiene como resultado un producto.
- **Duración limitada:** Por lo cual tienen una fecha de inicio y una fecha de fin.
  - Los proyectos son temporarios, cuando se alcanzan el/los objetivo/s, el proyecto termina.
  - Una línea de producción **no** es un proyecto.
- **Tareas interrelacionadas basadas en esfuerzos y recursos:** Divide el trabajo complejo en tareas simples que se interrelacionan, lo cual genera dependencias inevitables.
  - Complejidad sistémica de los problemas.
  - El proceso surge por parte de la definición de las tareas.
- **Son únicos:**
  - Todos los proyectos por similares que sean tienen características que los hacen únicos.

Administración de Proyectos: Es la aplicación de conocimientos, habilidades, herramientas y técnicas a las actividades del proyecto para satisfacer los requerimientos de este y poder obtener como resultado el producto esperado.

- "… tener el trabajo hecho …" en tiempo, con el presupuesto acordado y habiendo satisfecho las especificaciones o requerimientos.
- 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).

La triple restricción – Triángulo de hierro: Es la base de la gestión tradicional de proyecto. Plantea que el mismo tendrá una **restricción de tiempo,** de **presupuesto** y un **alcance.**

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

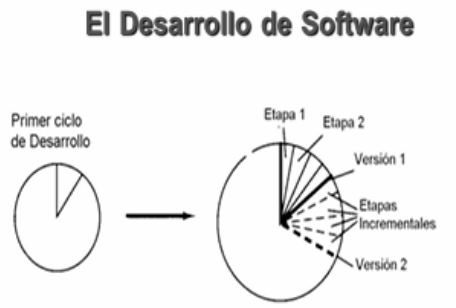


El balance de estos tres factores afecta directamente la calidad del proyecto.

"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 líder de proyecto (PM) balancear estos tres objetivos, ya que se suele competir entre ellos, donde se deberá decir, que la **calidad del producto no es negociable.**

- **Alcance:** Son los requerimientos que el cliente expresó, el objetivo, lo que quiere alcanzar.
  - Este nunca dependerá de nosotros ya que es el cliente quien necesita el producto. Por ende, en base al alcance podemos definir los costos y el tiempo, las cuales si se pueden manejar desde el rol de líder de proyecto.
- **Tiempo:** que debería llevar completar el proyecto.
- **Costo:** En recursos y en dinero.



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

El Rol del Líder de Proyecto/Equipo: En la gestión tradicional el equipo de proyecto está formado por un **equipo** y un **líder de proyecto (PM).** En procesos empíricos, no es necesario un PM ya que los proyectos se autogestionan.

Líder de proyecto – Project Manager (PM): Es el encargado de todas las tareas de gestión que hacen que el equipo de proyecto sea guiado a lograr el objetivo. Es el intermediario entre el ente que regula el avance del proyecto y demás involucrados con el equipo. A su vez también se encarga de administrar todos los recursos, organizar el trabajo de las personas

involucradas y hacer el seguimiento de la planificación verificando que todo vaya según lo planeado. Por lo que podemos decir que tiene la responsabilidad de:

- Estimar, planificar, organizar el trabajo y realizar el seguimiento del proyecto.

Equipo de Proyecto: Es un grupo de personas comprometidas en alcanzar un conjunto de objetivos de los cuales se sienten mutuamente responsables. Este equipo apunta a perseguir el mismo objetivo, el cual debe ser alcanzable y medible.

- Características de un equipo de proyecto
  - **Diversos conocimientos y habilidades.**
  - **Posibilidad de trabajar juntos efectivamente/desarrollar sinergia:** Siendo de gran diferencia a un trabajo individual.
  - **Usualmente es un grupo pequeño:** Fomenta de mejor forma la comunicación efectiva.
  - **Tiene sentido de responsabilidad como una unidad.**

Plan de Proyecto: Ante un proyecto este no estará completamente definido, pero se esbozará que se realizará durante el trayecto del proyecto, es decir: ¿Que se hará?, ¿cómo se hará?, ¿Que recursos vamos a utilizar?, ¿En qué tiempo se desarrollará? y ¿Quiénes son las personas involucradas? Brindando respuesta a todas esas preguntas logrando un plan detallado que servirá como guía del proyecto.

Mediante este plan de proyecto **documentamos:**

- **¿Qué es lo que hacemos?:** El alcance del proyecto.
- **¿Cuándo lo hacemos?:** Calendarizamos.
- **¿Cómo lo hacemos?:** Recursos y decisiones disponibles.
- **¿Quién lo va a hacer?:** Asignación de las tareas.

La idea primordial del plan de proyecto es que sea algo **vivo** a lo largo del proyecto, donde se retroalimente y se corrija a medida que el proyecto avanza. Siempre será útil si está en permanente actualización, ya que pueden surgir variaciones no estimadas a lo largo del proyecto.

Planificación de proyectos de software: En la gestión tradicional de proyectos basados en procesos definidos debemos contemplar/establecer lo siguiente siempre que se realice un plan de proyecto:

1. **Definición del alcance del proyecto:** Donde se realizará la diferenciación entre proyecto/producto de software.

Alcance del Proyecto	Alcance del producto
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 cumplimiento del alcance del proyecto se mide contra el **plan de proyecto.**	Son todas las características que pueden **incluirse** en un producto/servicio. El cumplimiento del alcance de producto se mide contra la **Especificación de Requerimientos (ERS)** que contiene los requerimientos funcionales y no funcionales.

   Relación: Si el producto a construir es grande, las tareas a definir en el proyecto van a ser más o van a requerir más tiempo y recursos, por lo que para **definir** el alcance del proyecto se debe primero **definir** el alcance del producto.
2. **Definición de proceso y ciclo de vida:** Lo importante es que al iniciar un proyecto se deberá definir que proceso quiero utilizar y que ciclo de vida voy a utilizar.
   El **proceso de desarrollo** es el conjunto de actividades que necesito ejecutar para construir el producto de software (Ej. PUD) y el **ciclo de vida** es de qué manera ejecuto esas actividades.
   En la gestión tradicional y los procesos definidos se puede elegir el proceso y cualquier ciclo de vida a utilizarse para llevar a cabo esté, mientras que en la **gestión ágil** el único ciclo de vida que se puede utilizar es el **iterativo.**

3. **Estimación:** Se debe estimar todo al momento de realizar un plan de proyecto, ya que no hay certezas de nada. Donde deberemos estimar todas las características necesarias para la planificación, lo cual se hará con un cierto nivel de incertidumbre, pero se tratan de estimar las siguientes características.
    - **Tamaño:** Definir el producto a construir.
    - **Esfuerzo:** Horas persona lineales (ideales) – No contempla tiempo para comer/ir al baño etc.
    - **Calendario:** Determinar qué días y horas se trabajará, y cuantas personas van a trabajar.
    - **Costo:** Determinar un presupuesto necesario – Usualmente definido al final ya que debe ajustarse a la realidad.
    - **Recursos críticos:** Tanto humanos como físicos que nos van a hacer falta.

    Al hablar de estimación se definen ciertos rangos. A medida que avanza el desarrollo de un proyecto, se minimiza la incertidumbre ya que se obtienen mayores certezas.
4. **Gestión de riegos:** El riego es algo que puede suceder o no, pero al suceder comprometerá el éxito del proyecto.

5. **Asignación de recursos.**
6. **Programación de proyectos.**
7. **Definición de controles:** El PM se encargará de trabajar sobre lo definido en el plan de proyecto y determinar si se está cumpliendo o no. Para ello se basará en esté y en las métricas.

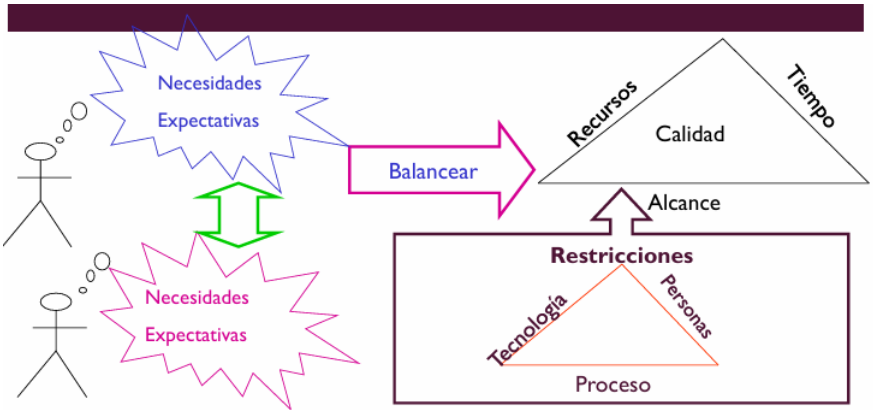
    "Un proyecto se atrasa de a un día a la vez" – Fred Brooks, Mythical Man Months.

    Razonando lo que dice Fred, si corregimos aquellos desvíos en el proyecto en el momento adecuado, estaremos a tiempo de cumplir el objetivo del mismo.
8. **Definición de métricas:** Las métricas de software se dividen en tres, donde nos permitirán darnos cuenta si nuestro proyecto está en línea con lo planificado. Esto se debe a que para planificar nos basamos en estimaciones, estas métricas permitirán saber si se cumple lo planificado o no.
    - **Métricas de proceso:** Permite saber independientemente de un proyecto en particular, si como organización estamos trabajando bien o mal. Permitiendo saber que pasa en términos **organizacionales**.
        - Son básicamente las mismas del proyecto, pero despersonalizadas de un proyecto en particular.
    - **Métricas de proyecto:** Se consolidan para crear métricas de procesos que sean públicas para toda la organización del software. Permitiendo saber si un proyecto de software en ejecución se está cumpliendo en base a lo planificado.
    - **Métricas de Producto:** Miden cuestiones/características que tienen una relación directa con el producto que estamos construyendo.

    Algunas métricas básicas para un proyecto de software son:
    - Tamaño del producto.

- Esfuerzo.
- Tiempo (Calendario).
- Defectos.

"*Si una métrica no tiene un fin particular, no tiene ningún sentido tomarla.*"



EL SUEÑO DEL PIBE…

Desarrollador

1. Esfuerzo
2. Esfuerzo y duración estimada y actual de una tarea.
3. % de cobertura por el unit test
4. Numero y tipo de defectos encontrados en el unit test.
5. Numero y tipo de defectos encontrados en revisión por pares.

Organización

1. Tiempo Calendario
2. Performance actual y planificada de esfuerzo.
3. Performance actual y planificada de presupuesto
4. Precisión de estimaciones en Schedule y esfuerzo
5. Defectos en Release

Equipo de Desarrollo

1. Tamaño del producto
2. Duración estimada y actual entre los hitos más importantes.
3. Niveles de staffing actuales y estimados.
4. Nro. de tareas planificadas y completadas.
5. Distribución del esfuerzo
6. Status de requerimientos.
7. Volatilidad de requerimientos.
8. Nro. de defectos encontrados en la integración y prueba de sistemas.
9. Nro. de defectos encontrados en peer reviews.
10. Status de distribución de defectos.
11. % de test ejecutados

Factores para el éxito	Causas de fracasos
1. **Monitoreo y feedback:** Tener monitoreo permanente y generar acciones correctivas cuando sea necesario para evitar una desviación mayor. 2. **Misión/objetivo claro:** Saber hacia dónde vamos. 3. **Comunicación:** En todos sus aspectos, con el líder del proyecto, con el equipo, con clientes y stakeholders.	• 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. • Mala comunicación.

Filosofía ágil: Movimiento surgido por desarrolladores, donde se acordaron un conjunto de enunciados a los que se llamaron **manifiesto ágil.**

El manifiesto es un compromiso que hacen todas las personas involucradas en un proyecto para trabajar de una determinada manera independientemente de las prácticas que realice c/u.

- La forma de trabajo será sustentada en los conceptos de los procesos empíricos.

Objetivo del movimiento ágil: Lograr un equilibrio entre seguir procesos rigurosos y formales en el desarrollo de software y trabajar de manera eficiente y efectiva para entregar un producto de calidad.

Ágil: Forma de llamar a está filosofía. No es una metodología ni un proceso, si no que es una **ideología** que cuenta con un conjunto definido de principios que guían el desarrollo de un producto. Buscando encontrar un equilibrio entre ningún proceso y demasiado proceso.

- Los métodos ágiles son adaptables en lugar de predictivos.
- Los métodos ágiles son orientados a la gente en lugar de orientados al proceso.
- Ejemplo de Frameworks ágiles: FDD, Cristal, XP, **SCRUM,** ATDD.

Como anteriormente se menciona el manifiesto ágil se sustenta en los procesos empíricos, los cuales centran sus bases en el **empirismo**, el cual tiene tres pilares:

1. **Transparencia:** Se refiere a la **comunicación abierta** y **honesta** de la información relevante a todas las partes interesadas.
    - Promueve la confianza y la colaboración entre los miembros del equipo y las partes interesadas, lo que a su vez conduce a una toma de decisiones más informada y eficaz.
2. **Adaptación:** Es la **capacidad de adaptarse** y **ajustar el trabajo** y el proceso para **hacer frente a las desviaciones** y **problemas** detectados durante la inspección.
    - Permite una respuesta rápida y efectiva a los cambios y permite mantener la entrega de valor al cliente.
3. **Inspección:** Se refiere a la **revisión regular** y **sistemática** del progreso del trabajo y los productos resultantes para **detectar problemas y desviaciones** del plan.
    - Permite una evaluación temprana y continua del progreso y ayuda a tomar decisiones informadas sobre los próximos pasos.

Manifiesto ágil: El manifiesto ágil posee cuatro valores y doce principios.

1. **Individuos e interacciones sobre procesos y herramientas:** Se enfatizan los vínculos, la comunicación efectiva y la colaboración entre los miembros del equipo y las partes interesadas, por sobre la adopción de la herramienta que tenemos que usar y el proceso a aplicar.
2. **Software funcionando sobre documentación extensiva:** Se enfatiza la importancia de brindar software funcional y de alta calidad en lugar de enfocarse en documentar cada detalle del proceso de desarrollo.
   Esto no quiere decir que **NO** se debe generar documentación, existe la necesidad de mantener la información del producto y del proyecto que estamos cursando para desarrollar le mismo.

El enfoque ágil plantea la idea de generar la información cuando haga falta. "Haciendo memoria lo más importante es el acto de planificar **no** el resultado".
Las decisiones tomadas respecto al producto de software deben quedar documentadas, ya que estas transcenderán más allá de la iteración en la cual se generó el incremento. De esta forma decidimos que aspectos del producto quedarán documentados. Respecto al proyecto, se definirá de igual forma que se documentará.

3. **Colaboración con el cliente sobre negociación contractual:** Enfatizado en la importancia de trabajar en colaboración con el cliente y comprender sus necesidades y requisitos en lugar de simplemente negociar contratos y acuerdos formales.
   Relación con la triple restricción: Se da ya que en las negociaciones contractuales el cliente firma un contrato con el equipo definiendo un alcance/costo/tiempo determinado y luego quiere realizar cambios.
   - Estos problemas se evitan al involucrar al cliente de manera más activa en el proyecto y se fomenta una actitud de colaboración e integración.
   - Siendo **fundamental** que el cliente esté dispuesto a participar y sumarse a este enfoque, en caso de que no, la metodología ágil no se podrá aplicar.
4. **Responder al cambio por sobre seguir un plan:** Enfatizado en la importancia de ser flexible y adaptativo en respuesta a cambios y desviaciones en el proceso de desarrollo en lugar de seguir un plan rígido y predefinido que puede no ser el adecuado para el contexto actual.
   Planteando que debemos construir en conjunto con el cliente, no definir tanto, empezar a trabajar con una idea del producto y después enfocarnos más a medidas que avanzando, dando la posibilidad al cliente cambiar de opinión. Por lo tanto, se tienen en cuenta las siguientes cuestiones:
   - En lugar de una ERS firmada por el cliente, es mejor tener una actitud más ajustada a la realidad de que los requerimientos cambian, la gente se equivoca, se olvida, se da cuenta de lo que quiere cuando lo ve, y no por un contrato.
   - En lugar de depender únicamente de un contrato, es mejor tener una comunicación frecuente y colaborativa con el cliente para adaptarse a los cambios y logar una entrega exitosa del producto.

Cuando hablamos de **requerimientos ágiles**, debemos ponerlos en contexto, ya que es una manera de trabajar con los requerimientos alineados con los doce principios del manifiesto ágil.

1. **Satisfacción del cliente:** Nuestra mayor prioridad es **satisfacer al cliente** mediante la entrega temprana y continua de software con valor.
2. **Adaptación al cambio:** Aceptamos que los requerimientos cambien, incluso en etapas tardías de desarrollo. Los procesos ágiles aprovechan el cambio para brindar ventaja competitiva al cliente.
3. **Entregas frecuentes:** Entregamos sw funcional con frecuencia, con preferencia a la documentación exhaustiva – Entre dos semanas/dos meses, con preferencia al periodo de tiempo más corto.
4. **Trabajo en equipo:** Colaboramos con el cliente e interesados durante todo el proyecto, asegurando así que el sw entregado satisfaga las necesidades del negocio.
5. **Personas motivadas:** Construimos proyectos en torno a individuos motivados. Brindando el entorno y el apoyo necesario y confiamos en que harán el trabajo.
6. **Comunicación directa:** El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación **cara a cara**.
7. **Software funcionando:** Es la medida principal de progreso.
8. **Continuidad:** Los procesos ágiles promueven el desarrollo sostenible, es decir, el equipo de trabajo debe ser capaz de mantener un ritmo constante y sostenible de trabajo a lo largo del tiempo.
    - **Framework agiles:** Apuntan a ciclos de vida iterativos y de duración fija, si no se llega con lo previsto en la iteración se entrega lo que esté listo.
    - La forma de llegar a que un equipo tenga un desarrollo sostenible es lograr que el equipo tenga un ritmo de trabajo que asegure entregar un ciclo de tiempo fijo.
9. **Excelencia técnica:** Tanto la excelencia técnica y el buen diseño mejoraran la agilidad. La calidad del producto no es negociable, debemos ajustar cualquier aspecto del producto menos su calidad al entregar.
10. **Simplicidad:** Buscando la simplicidad, se puede maximizar la eficiencia y la efectividad, al mismo tiempo que se reduce la complejidad y el riesgo de errores. Esto se loga eliminando tareas innecesarias y la concentración en los objetivos más importantes y críticos del proyecto. Por lo que también no **debemos** agregar funcionalidades porque si, si el cliente **no lo pidió.**
11. **Equipos auto-organizados:** Gracias a ellos emergen mejores arquitecturas, requisitos y diseños, ya que tienen libertad de tomar decisiones y diseñar soluciones de manera colaborativa. Brindarle al equipo de desarrollo cierto grado de autonomía y capacidad de decisión en el proceso de desarrollo es muy importante, a comparación de tener un enfoque jerárquico y rígido. El agilismo rompe por completo el enfoque tradicional y expresa que los más **aptos** para definir características del producto, son quienes lo están desarrollando.
12. **Mejora continua:** El equipo reflexiona sobre como ser más efectivo para luego ajustar y perfeccionar su comportamiento en secuencia durante ciertos intervalos de tiempo.

Triángulo de hierro vs Triángulo ágil: El triángulo de hierro es el modelo utilizado en la gestión tradicional de proyectos en donde se establece que los elementos fundamentales en cualquier proyecto son: el alcance, el tiempo y el costo. Según este modelo, cualquier cambio en uno de sus elementos afectaría a los otros dos. En la gestión tradicional se busca **establecer** un plan detallado y fijo en cuanto a estos tres elementos antes de comenzar el proyecto, y se trabaja para asegurar que se cumplan con el transcurso del proyecto.

Mientras que, en el enfoque **ágil**, se considera que el alcance, el tiempo y el costo son variables flexibles que pueden adaptarse y cambiarse a lo largo del proyecto. No se establece un plan fijo al inicio del proyecto, ya que se utilizan iteraciones cortas y regulares para adaptar el proyecto según las necesidades y los requisitos cambiantes.

Lo que nos quiere decir es que lo que tiene realmente valor es el **software funcionando** para el cliente, por lo que está centrado en la **calidad del producto** (intrínseca – Características

de calidad que le ponemos al producto para que satisfaga al cliente) y la calidad extrínseca (valor que tiene el producto para el cliente).

En la gestión ágil buscamos que los tres elementos fundamentales: Valor, Calidad y la Restricción del tiempo, maximicen el valor que se entrega al cliente, asegurando la calidad del producto y cumplir con los plazos establecidos, en lugar de centrarse en el alcance y el costo → Resumen.

Requerimientos ágiles: El enfoque ágil no solo plantea una gestión diferente del proyecto, sino que a su vez una definición de requerimientos distinta.

Dentro de la ingeniería de requerimientos, existen dos procesos: **El desarrollo de requerimientos** y **la gestión de los requerimientos.**

- La gestión esta vinculada en mantener los requerimientos íntegros, bien identificados y trazados y qué al momento del cambio, que todos aquellos lugares en donde se encuentre se actualicen. Esto ocurre recurrentemente al trabajar con un producto de software.
- El desarrollo de requerimiento es la parte que se centra en identificar, especificar y validar los requerimientos.

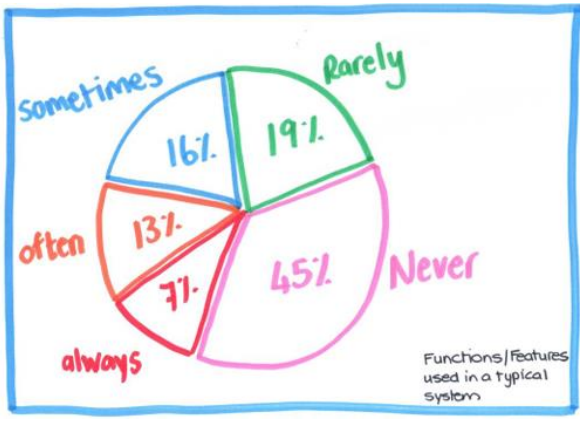
Pilares de los Requerimientos ágiles

Usar el valor para construir el producto correcto: Lo importante es dejar contento al cliente con sw funcionando que le sirva, por lo que el enfoque de gestión ágil apunta a construir valor de negocio.

Usar historias y modelos para mostrar que construir: Los requerimientos se expresan en forma de **historias de usuario**, que son breves descripciones de los requisitos del cliente que se describen en lenguaje natural y se enfocan en las necesidades del usuario. Las cuales son utilizadas para **definir el alcance** de cada iteración del proyecto y se **priorizan en función del valor que aportan al producto.** En este pilar hacemos foco en la idea de trabajar par a par con el cliente (1er y 3er valor del manifiesto ágil)

Determinar que es "solo lo suficiente": Los requerimientos se irán descubriendo a medida que avanza el proyecto, por lo que este pilar indica que debemos comenzar con lo mínimo necesario para generar retroalimentación para mejorar continuamente (empirismo) y a partir de allí, se avanza continuamente.

Todo esto difiere del enfoque tradicional que especifica los requerimientos antes de avanzar en el proyecto, lo cual es conocido como BRUF – Big Requirements Up Front, que básicamente es lo que nos indica que se debe cambiar el enfoque para **determinar solo lo suficiente**, esto se debe a que todos los productos de sw tienen un **desperdicio significativo** (no se utilizan todas las funcionalidades desarrolladas – Ejemplo: Excel).

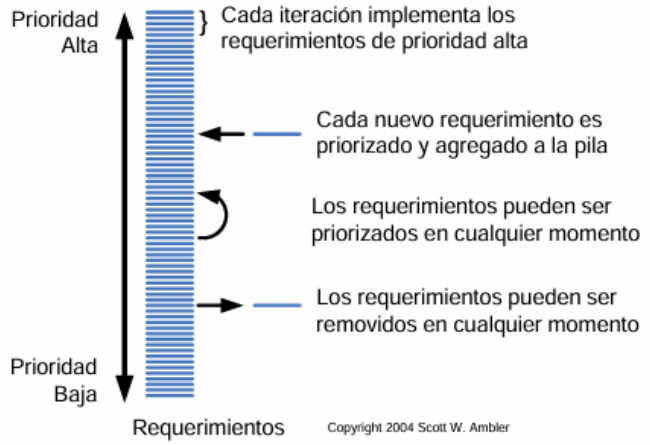


La imagen es sobre el BURF y los porcentajes asignados a las funcionalidas, como vemos, hay un gran desperdicio el 45% de las funcionalidades del producto de software no se utilizan y solo el 7% son utilizadas siempre. Para ello necesitamos de la…

Gestión ágil de requerimientos de software: Es quien busca la forma de contrarrestar la situación planteada anteriormente en el BURF, de forma que se prioricen **solo** aquellas funcionalidades que aporten **valor al cliente**.

Para lograr esto, la gestión ágil utiliza el concepto de dueño del producto (**Product Owner - PO**) como figura clave en el proceso de desarrollo. Es quien será responsable de identificar las necesidades y prioridades del negocio, para posteriormente priorizar los requerimientos del producto (que necesita primero y que da un valor significativo al negocio).

Product Backlog (PB): Es una lista priorizada y ordenada de requerimientos organizada por el PO. En donde los requerimientos más importantes se encuentran en la parte superior y los menos importantes en la parte inferior.



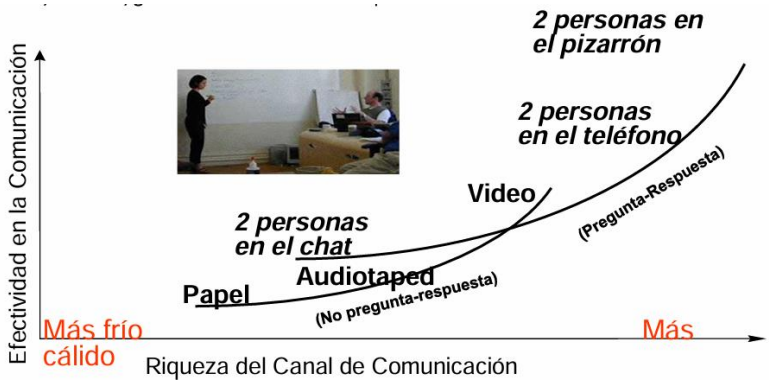
En la gestión ágil de los requerimientos se apunta a trabajar y a poner el esfuerzo en lo que nos da valor, y a trabajarlo en el momento adecuado. Aquí el cliente participa en la priorización de sus necesidades.

En la gestión tradicional, el usuario no estaba tan comprometido, de forma que la priorización se basaba en el equipo, como consecuencia, software que no satisfacía las necesidades del cliente.

Just In Time: Concepto fundamental utilizado para evitar desperdicios (especificar requerimientos que después cambian). El producto no estará 100% especificado desde el principio, se irán encontrando requerimientos y describiéndolos conforme haga falta.

"Analizo cuando lo necesito, no antes ni después (que no se haga tarde). Analizo con detalle un requerimiento cuando tenga el suficiente valor"

Lo que debemos entregarle al cliente es **valor de negocio**, no características de sw. El sw es el medio por el cuál nosotros le entregamos valor de negocio.



Comunicación cara a cara: Indica que la efectividad y riqueza de la comunicación crece exponencialmente conforme todos puedan estar trabajados físicamente en un mismo lugar. Donde el punto máximo se logra con dos personas en un mismo espacio físico con un pizarrón para compartir.

Tipos de requerimientos

- **De negocio:** Son los de más alto nivel, son aquellos en términos de la visión del negocio – Ejemplo: Disminuir X% de tiempo invertido en procesos manuales.
- **De usuario:** Tienen que ver concretamente con los requerimientos del usuario final – Ejemplo: Realizar consultas en línea del estado de cuenta de los clientes. → User Stories.
- **Funcionales:** Podríamos relacionarlo 1 a 1 con los CU en la gestión tradicional – Ejemplo: Notificaciones vía mail.
- **No funcionales:** Suelen ser los más ocultos, que el cliente no tiene en claro y se deben especificar. Un RNF puede hacer que el sw que se está desarrollando **no sirva** (importancia funcional) – Ejemplo: Formato de un PDF.
- **De implementación:** Se suele decir que están dentro de los no funcionales, pero tienen que ver con cuestiones de restricción o implementación específicos – Ejemplo: Servidores en la nube.

La **gestión ágil de requerimientos** implica trabajar junto a técnicos y no técnicos entendiendo las necesidades y el negocio para construir un sw que de **valor.** Y luego, junto a los usuarios descubrir **cuál es la mejor forma de satisfacer esas necesidades** (Product Backlog) al entregar al cliente la característica podemos obtener feedback el cual nos ayudará a refinar el PB.

- Las US (User Stories) no sirven para especificar requerimientos de sw, sirven para **identificar** requerimientos de usuario.
- En ASI y DSI se trabajan con requerimientos funcionales, no funcionales y un poco de requerimientos de implementación.

El foco de la gestión ágil de requerimientos está puesto sobre el usuario y sus necesidades.

User Stories → Están ubicadas dentro de los requerimientos de usuario con un **foco claro** hacia el negocio.

Casos de Uso → Están ubicados en requerimientos de software.

Ahora bien, con todo este contexto, tenemos en cuenta que vamos a tener una relación con los principios ágiles al utilizar la gestión ágil, por ende, estos son:

1. **Los cambios van a existir todo el tiempo de forma constante:** Todo el tiempo habrá cambio, por lo que se busca gestionar los requerimientos orientados a poder aceptar el cambio. Si el PO, el cliente o los stakeholders relacionados con el producto no plantean cambios es porque el producto no está siendo utilizado.
2. **Debemos tener una mirada de todos los que están involucrados:** Todos aquellos que tengan algo para decir del producto (Stakeholders: No son todos los que están.) Siendo el PO el representante de todos los involucrados.
3. **El usuario dice lo que quiere cuando recibe lo que pidió:** En base a esto las iteraciones son cortas y las entregas continuas son esenciales, permitiendo obtener retroalimentación y un enriquecimiento para futuras entregas y para una retroalimentación significativa.
4. **No hay técnicas ni herramientas que sirvan para todos los casos:** Se deberá identificar la herramienta adecuada para cada situación.

5. **Lo importante no es entregar una salida, un requerimiento, si no, un resultado, una SOLUCIÓN DE VALOR:** Con esto se hace referencia a que sea útil para el negocio o que el cliente minimice el desperdicio.

Principios ágiles relacionados a los requerimientos ágiles: En base a la imagen, debemos tener en cuenta que los más importantes son el principio 4 y 6. Ya que si recordamos lo expresado anteriormente podemos darnos cuenta qué son los que más se tienen en cuenta para esta etapa.

User Stories (US) – Historias de usuario: Es una herramienta/técnica para capturar requerimientos. Es utilizada de forma similar a contar una historia, es decir una descripción **corta** de una necesidad que tiene el usuario respecto del producto de software.

"La parte más difícil de construir un sistema de sw es decidir precisamente qué construir. Ninguna otra parte del trabajo conceptual es tan difícil como establecer los requerimientos técnicos detallados. Ninguna otra aparte del trabajo afecta tanto el sistema resultante si se hace incorrectamente. Ninguna otra parte es tan difícil de rectificar más adelante - Fred Brooks en No Silver Bullet."

Todo error cometido en la etapa de definición de requerimientos es más difícil de corregir ya que son identificados cuando el usuario tiene el sistema al frente, por lo cual este problema se arrastra a lo largo de toda la vida del proceso.

Podemos decir que las user stories son multipropósito debido a que 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.**

Partes de una user storie – Triple C:

- **Conversation - Conversación:** Es la parte más **IMPORTANTE** que no queda escrita en ningún lugar. En esta conversación, es donde se obtienen todos los **detalles** para que el equipo pueda trabajar esa historia – Es la parte no visible.
    - Es **importante** ya que nos permitirá dar límites o establecer lineamientos en base a está conversación con el PO o quien conozca el equipo de desarrollo.
- **Card – Tarjeta:** Es donde se escribe/expresa algo que os indique cual es la US, de que se trata y su **valor de negocio –** Parte visible de la US.
- **Confirmation – Confirmación:** Son aquellas condiciones que se tienen que dar para dar por satisfecha la US. Se encuentra dentro de la tarjeta – Son las pruebas de usuario.

Forma de expresar una US: Basados en la tarjeta, tenemos la siguiente nomenclatura definida:

<Frase verbal>

Como <Nombre del rol> yo quiero/puedo <actividad> de forma tal que <valor de negocio que recibo>.

- **<Frase verbal>:** No es obligatoria, pero es una forma cortada para referenciar la US.
- **<nombre del rol>:** Es quién está realizando la acción o quién recibe el valor de la actividad. Debiendo ser especificado explícitamente el rol del usuario.
- **<actividad>:** Representa la acción que realizará el sistema.
- **<Valor de negocio que recibo>:** Comunica porque es necesaria la actividad.

Esta forma de expresar una US busca contestar las siguientes preguntas:

- **WHO?:** Quién necesita esto.
- **WHAT?**: Que se necesita.
- **WHY?**: Por qué o para qué → Haciendo referencia al valor de negocio.

Ejemplo de US

*"Lo que más nos importa es el **valor de negocio** que recibo y la misma descripción explícitamente nos obliga a definir o identificar el valor de negocio que la actividad que voy a realizar."*

Para ello se deberá tener muy claro el valor del negocio, permitiendo que el PO priorice el Producto Backlog, ya que es el encargado de agregarlas, quitarlas, otorgarles mayor/menor prioridad, según lo que a él le parezca.

Se suele decir que las **US** son porciones verticales, donde para, poder **entregarle valor** al cliente debemos tener un poco de interfaz, lógica de negocio y base de datos. Se debe diseñar el pedacito de cada cosa que nos hace falta para que una determinada característica funcione.

Para que se asocie de mejor manera una US es como cortar un budín, no lo vas a cortar por capas, siempre se corta en rodajas (haciendo referencia a que tendrá todo lo de la imagen) – ¿Además, quien corta un budín en capas?

Modelado de roles: Se trata de modelar los roldes dentro del sistema para logar identificar aquellos que probablemente formen parte de las US, y en base a ellos, encontrar las US.

- **Tarjeta de rol de usuario:** Definimos un nombre específico y su explicación en la cual detallamos su función.
  - Es ambiguo poner como rol de usuario el nombre **usuario,** se debe evitar.
  - Se debe evitar el uso de **usuarios representantes (Proxies)** ya que no son ideales como los usuarios verdaderos como, por ejemplo: Gerente de usuarios, Gerentes de desarrollo, alguien del grupo de marketing,

vendedores, expertos del dominio, clientes, capacitados y personal de soporte.
Suelen ser utilizados en caso de que el PO no tenga tiempo para nuestras preguntas, pero solamente para cuestiones de negocio no para roles de usuario.

Criterios de aceptación (CA): Son la base para que luego se construyan los casos de prueba. Ya que son aquellas cuestiones que definen los límites de la US. Ayudan a que el PO responda los criterios que necesita para que la US provea **valor de negocio** y también a que el equipo tenga una visión compartida de la US.

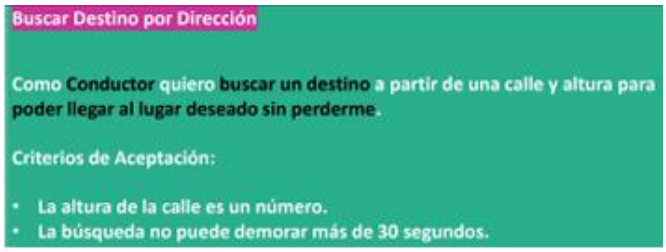
Características de los CA:

- Definen límites para una US.
- Ayudan a que los PO respondan lo que necesitan para que la US provea valor (Req. 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 funcionalidades a una US.
- **No van detalles dentro de los CA.**

Resumiendo, son aquellos que definen una **intención** y no una **solución**, que son independientes de la implementación y que son relativamente de alto nivel. Su redacción deberá ser clara y tienen que haber validaciones concretas.

- Puede → Opcional.
- Debe → Obligatorio.

Ejemplo:

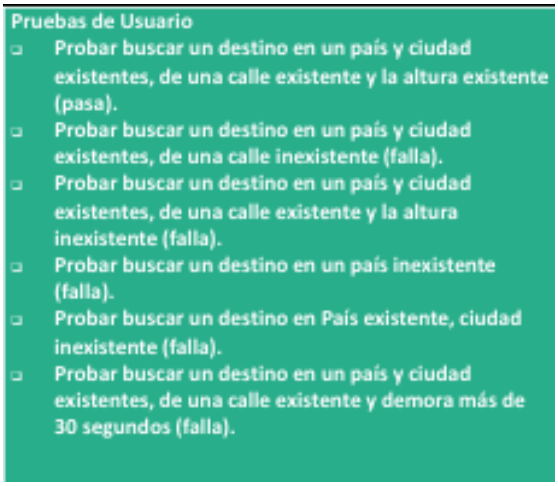


"Son las consideraciones que define nuestro usuario a la hora de hablarnos de cómo imagina él esa funcionalidad, no hay limitación en la cantidad de criterios."

"Se escriben a nivel de usuario, debiendo ser OBJETIVOS y VERIFICABLES"

Pruebas de aceptación (PA): Expresan detalles resultantes de la conversación, contemplando la US en situaciones de éxito y de fracaso. Otorgando la confirmación de que finalmente lo que estamos haciendo, va a hacer lo que el PO espera.

- **Manejo de errores y excepciones:** Son las situaciones de fracaso que nos permitirán pensar la actitud que tomaremos ante la misma.
- Son definidas por el usuario ya que es quien nos cuenta la realidad del negocio, pero quien las escribe no es el usuario en sí. Siendo un trabajo colaborativo entre el equipo y el PO.



Definition of Ready (DoR) – Definición de listo: Medida de calidad que construye el equipo para poder determinar que la US está en condiciones de entrar en una iteración de desarrollo. Para implementarla, la misma deberá cumplir con el **modelo INVEST.**

Modelo INVEST: Es una manera de trabajar sobre la DoR.

- **I**ndependent – **Independiente:** Calendarizables e implementables en cualquier orden. Otorgamos libertad al PO de que este priorice la US para que se puedan desarrollar en cualquier orden.
- **N**egotiable – **Negociable:** El "que" no el "como". La US debe estar escrita en términos de qué necesita el usuario, no de cómo se implementará.
- **V**aluable – **Valuable:** Debe tener **valor** para el cliente. Debe estar el para qué, es decir, debe tener valor de negocio.
- **E**stimatable – **Estimable:** Para ayudar al cliente a armar un ranking basado en costos. Se asigna un peso a la US indicando cuánto tiempo llevará terminar esa US completa.
- **S**mall**:** Debe ser "consumidas" en una iteración. Se comienza y termina una característica en una iteración, no se deja trabajo a medias. Depende mucho del equipo, la experiencia de este, la capacidad de trabajo y la duración de la iteración.
- **T**esteable: Demostrar que fueron implementadas. Relacionado con las PA, permitiendo demostrar que la US efectivamente fue implementada cumpliendo con los CA definidos.

Definition of Done (DoD) – Definición de Hecho: Criterio que nos dice cuando una US está lo suficientemente bien implementada como para entrar al sprint review, donde se la mostrará al cliente, y ahí se determinará si entra a producción o no.

- Se arma una checklist de todo lo que debe cumplir esta US para entrar a la review.

Algo más sobre las US:

- No son especificaciones detalladas de requerimientos (como los CU).
- Son expresiones de **intención**, "es necesario que haga algo como…"
- No están detallados al principio del proyecto, elaborados evitando especificaciones anticipadas, demoras en el desarrollo, inventario de req 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.
- Define una necesidad del **usuario** desde la perspectiva del **usuario/cliente**.
- Cumpliendo las tres C's estamos en condiciones de implementarla.

Diferentes Niveles de abstracción

- **Epic – Épica:** US muy grandes que no están en un lugar del Product Backlog o que todavía no deben ser implementadas
  - **Distinción → Épica vs US:** La épica no cumple el modelo INVEST o cuando no se puede ejecutar esa US en una iteración. La Epica debe ser dividida en US más pequeñas.
- **Theme - Tema:** Agrupa US relacionadas a un mismo tema. Su nivel de abstracción es más grande que el de una épica.
- **Spike:** Son un tipo especial de US, son utilizadas para quitar el riesgo e incertidumbre de una US u otra faceta del proyecto que nosotros queremos investigar. a su vez
  Utilidad:
  - Inversión básica para familiarizar al equipo con una nueva tecnología/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.

Las Spikes a su vez pueden dividirse entre: Técnicas y Funcionales

- **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 que comprometerse a una nueva funcionalidad en un tiempo fijo.
- **Funcionales:** Utilizadas cuando hay cierta incertidumbre respecto cómo el usuario **interactuará** con el sistema. Usualmente son mejor evaluadas con prototipos para obtener retroalimentación de los usuarios o involucrados.

"Hay US que requieren ambos tipos de spikes"

Lineamientos para Spikes:

- **Estimables, demostrables y aceptables** → Deben cumplir el Criterio de Ready.
- **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 probable de encontrar una solución rápida en cuyo caso, spike e US pueden incluirse en la misma iteración.
  - De lo contrario se realizan en un sprint anterior al que se va a implementar la US en la que se tiene incertidumbre.

Tips para que las US sean útiles para el equipo (*Se puede skipear esto*):

- Un paso a la vez (evitar la palabra "Y").
- No olvides la parte invisible: La conversación.
- No forzar todo para escribirlo como US.
- Usar palabras claras en los CA.
- Las US se escriben desde la **perspectiva del usuario.**

Estimaciones de Software: Estimar es predecir en un momento donde no tenemos gran conocimiento, donde hay incertidumbre.

La estimación no es precisa, corre alto riesgo de incertidumbre, por lo que situados en el inicio de un proyecto está será aun mayor. A medida que avanza el proyecto, la incertidumbre disminuye.

Problemática de las estimaciones: Toda estimación tiene integrado el concepto de **probabilidad**, por lo que por eso generan incertidumbre.

Frente a la posibilidad de estimar mal y generar enojo por parte del cliente tenemos dos reacciones posibles:

- **Subestimar:** Se simplifica sin considerar todo el trabajo que hay que hacer, por miedo a que si el valor es más alto el cliente lo rechace.
- **Sobreestimar:** Se infla de más la complejidad del producto, tendiendo a confundir con compromisos o planificación.

Estimar NO es planear y no necesariamente nuestro plan tiene que ser lo mismo que lo que estimado. Si bien está sirve para planificar, no tienen que seguirlas ya que no son compromisos.

¿Para qué estimamos?: Estimamos como se habla de predecir un momento donde no tenemos la certeza necesaria o hay un espectro de riego que puede hacer que lo que se está prediciendo no ocurra de la manera esperada.

En el momento donde realizamos la estimación es donde tenemos menos información de la que necesitamos, a medida que avanzamos, obtendremos la información que va a avalar o negar la estimación realizada, pero aún así, ante el riesgo del desarrollo del producto, necesitamos estimar.

Causas comunes de cometer errores de estimación:

- Si el proyecto no está organizado, el nivel de conocimiento sobre el producto es menor, provocando que las estimaciones sean imprecisas.
- Si no se tiene en claro los recursos con los que la empresa cuenta para desarrollar el producto ni las capacidades de estos, las estimaciones no suelen ser buenas.
- Las técnicas de estimaciones también pueden ser fuente de error, para solucionar esto se suele emplear más de una técnica y comparar sus resultados. Ante resultados muy distintos, es índice de que hay un error.

Tamaño del software: Es lo primero a estimar, nos referimos a que tan grande será el trabajo por realizar. Para ello contamos con la técnica **CONTAR.**

- Contar: Funcionalidades, Requerimientos, Cant. De US, Cant de CU.

Aquí entra en juego la información que tenemos para realizar la predicción/estimación y el nivel de incertidumbre/riesgo que vamos a tener. Para esto nos sirve utilizar **datos históricos** que nos serán útiles para comparar el tamaño posible de nuestro sw con un sw ya desarrollado con características similares.

Esfuerzo: Es la cantidad de horas lineales que se necesitarán para construir el sw, no se incluye a las personas, ni calendario, ni las actividades detalladas, solo **CUENTO**, en estimación, cuantas horas lineales de desarrollo necesitaré.

Calendario: Tras estimar el esfuerzo, se calendariza el esfuerzo y se propone una fecha de entrega.

Costos y presupuestos: Tras estimar, tamaño, esfuerzo y tiempo calendario se comenzara a trabajar con esto.

Técnicas de estimación: Son formas/técnicas que nos ayudarán a disminuir/acotar la incertidumbre. No van a eliminar la incertidumbre, pero nos situará en contexto.

- Cada técnica tiene sus ventajas y desventajas, por los cual, es importante seleccionar la adecuada para cada proyecto en función de sus requisitos y características.

Los métodos utilizados son los siguientes:

Métodos basados en la experiencia: Se basa en la experiencia pasada del equipo en proyectos similares. Para determinar la estimación de: Esfuerzo y tiempo requerido p/ un nuevo proyecto.

- **Datos históricos:** Tomo como base la experiencia de otras personas y proyectos. Lo cuál no es acorde a la gestión ágil ya que explicita que la experiencia **no se puede** extrapolar a otros equipos.

- **Juicio Experto:** Técnica basada en la experiencia y conocimiento de expertos en la materia para determinar la estimación. Sujeta a la subjetividad y experiencia de los expertos. Las estimaciones pueden variar dependiendo de las características del experto involucrado. Es el más utilizado en la industria.
    - **Puro:** Un experto estudia las especificaciones y hace su estimación. Basado fundamentalmente en sus conocimientos.
      Desventaja → Si desaparece el experto, la empresa deja de estimar.
    - **Delphi:** Se realiza de forma grupal donde indicarán lo que costará el desarrollo (esfuerzo y duración). Está estimación suele ser mejor que la individual. Es tomado como base para el **Poker Planning.**
      Desventaja → Requiere de más tiempo y recursos que el Puro, implicando múltiples rondas de estimaciones y análisis de datos hasta que la estimación converja de forma razonable.

  Criterios:
    - Estimar solo tareas con **granularidad aceptable** (no alta).
    - Usar el método **Optimista, Pesismista y habitual** que tiene que ver con estimar según 3 características y luego aplicar esto a una formula $F(x) = (O + 4H + P)/6$. No se sigue a rajatabla.
    - Utilizar un checklist y un criterio definido para asegurar cobertura.
- **Analogía:** Basada en datos históricos, pero se realiza una abstracción de aquellos parecidos o que concuerdan con el proyecto con el que se está trabajando.

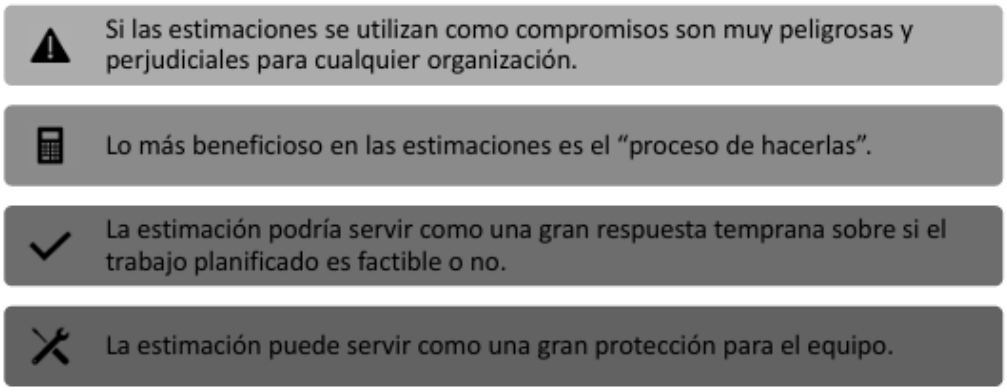
Métodos basados exclusivamente en recursos.

Métodos basados exclusivamente en el mercado.

Métodos basados en los componentes del producto o en el proceso de desarrollo.

Métodos algorítmicos.

Estimaciones en ambientes ágiles: Mantenemos los conceptos de estimación de software, pero debemos tener en cuenta lo siguiente:



Aquí, surge la propuesta de que estime la misma persona que hace el trabajo, no algún "experto". Pero debemos tener en cuenta las opiniones de los demás y no cerrarse a nada.

*"Lo ideal es empezar con una estimación que se pueda revisar todas las veces que sea necesario, ya que, como principio ágil, tenemos que la mejor métrica de progreso es el **software funcionando**"*

Las **features/stories** son estimadas usando una medida de tamaño relativa (Story Points - SP). Los cuales son una ponderación dada a la US cuyo peso es la combinación de: **Incertidumbre, Esfuerzo y Complejidad.**

- Este tipo de estimación separa por completo la estimación de esfuerzo de la estimación de duración del proyecto.
- La complejidad de una feature/US tiende a incrementarse exponencialmente.

Medidas o estimaciones relativas: Son aquellas que las define un equipo y no son comparables entre distintos equipos, por ende, decimos que **no son absolutas** y su vez tampoco es una medida basada en el tiempo.

Premisas de la estimación relativa: Usualmente las personas no saben estimar en términos absolutos, para ello utilizamos la comparación, ya que es generalmente más rápida para asociar. En este caso en la estimación ágil hacemos uso de las US Canónicas, obteniendo una mejor dinámica y acuerdo grupal, centrando el pensamiento y a su vez permitiendo emplear de mejor manera el tiempo de análisis de las US.

Al estimar una US se utiliza el: Tamaño, Tiempo y Esfuerzo necesario de esta.

- **Tamaño:** Medida de la cantidad de trabajo necesario para producirla/completarla, y a su vez índica que tan compleja y que tan grande es.
    - El **Tamaño NO ES esfuerzo:** El esfuerzo está asociado a las horas lineales, pero es bueno ya que nos ayuda a despegarnos de cuestiones que implican tiempos determinados (Calendarización – Suele tener más errores).
    - Existen diferentes formas de poner en escala el tamaño de una Feature/US.
        - **Por números:** 1 al 10.
        - **Talles de remeras:** S, M, L, XL, XXL.
        - **Serie 2^n**: $1, 2, 4, 8, 16, 32, 64, \dots, m$
        - **Fibonacci:** 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, Etc. → *La que usamos*.
- Las estimaciones basadas en el **tiempo** son más propensas a errores debido a factores externos como: Habilidades de las personas, conocimiento del dominio, experiencia, etc.

Velocity - Velocidad: Es una medida (métrica) del progreso de un equipo. Calculada mediante el número de SP asignados a cada US que el equipo **completo** al 100% durante la **iteración.**

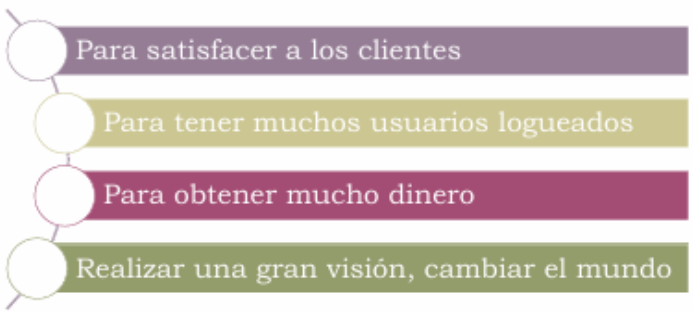
- No se contabilizan SP de US parcialmente completas.
- Útil para corregir errores de estimación debido a la información que brinda.
- En base a la está métrica podemos derivar la duración de un proyecto. Tomando la cantidad de SP de las US y dividiéndola por la velocidad del equipo.
- Nos ayuda a terminar un **horizonte** de planificación apropiado.
- No debemos comprometernos con una velocidad específica, sino a un rango.

Métodos de Estimación ágil:

- **Poker Estimation:** Combina la opinión de experto, analogía y desagregación. Está basado en la serie de Fibonacci, en base a esta, permite una estimación un poco más exacta sobre el planteo sobre el crecimiento exponencial de una US.
    - Si tenemos una SP muy grande, es probable que la US no se pueda ejecutar en un **solo sprint** → No cumple con el **DoR.**
- **Poker planning:** Los participantes son desarrolladores, siendo estos quienes estiman. Cada miembro del equipo de desarrollo selecciona una carta que representa un valor numérico (0 - 100) y la coloca boca abajo en la mesa. Tras la elección de todos los participantes, se dan vuelta simultáneamente permitiéndoles a todos verlas. Aquellos que hayan elegido cartas con valores más altos y los valores más bajos tienen que explicar el razonamiento de su elección. Luego se lleva a cabo una nueva ronda de estimación hasta llegar a un consenso en cuanto al valor de los SP para cada US.

*"Cuando hablamos de Agile hablamos de **empirismo**, por lo tanto, en las estimaciones tenemos que cumplir con las características y pilares del empirismo: Ejecutar, Inspeccionar y Adaptar. Y la retroalimentación"*

Gestión de Productos: Al hablar de esto, comenzamos a pensar en términos de visión para que construimos productos de software, en este contexto aparecen distintas razones.



¿Por qué creamos productos?

Creamos productos con una visión de cambiar el mundo o nuestra diaria (visión más ambiciosa que crear un producto que nos pide un cliente).

Usualmente sabemos que se invierte mucho tiempo en construir características del sw que no se usan o se usan muy rara vez. Lo cual implica un esfuerzo innecesario en algo que no se va a utiliza o vender. Por lo que es muy importantes, eliminar el **desperdicio** para poner nuestro esfuerzo en aquello que sea útil.

Evolución de los productos de software: Si tenemos en cuenta lo mencionado anteriormente, podríamos armar una especie de pirámide respecto a la evolución de los productos de sw, pensando en las características del sw.

En la **base** de esta pirámide, nos encontramos con aquellas, que si o si, deben existir para que el software funcione para el propósito esperado, que sean confiables y que de alguna manera estén alineadas con la experiencia de usuario, permitiendo lograr lo que esperamos con esa funcionalidad.

Usable, Confiable y Funcional son todos aquellos aspectos que esperamos cuando construimos/usamos un producto de sw.

Mientras que, en el **tope**, aparecen aspectos relacionados a cuestiones sobre la visión del producto las cuales son difíciles de alcanzar. Hablamos de un producto **Conveniente, Placentero y significativo.**

Normalmente los productos que se construyen en las organizaciones se quedan en la base de la pirámide y hasta ahí llegan.

Desafio: Surge ya que debemos ver **cómo hacer para construir un producto de sw donde desechamos el desperdicio y logamos abarcar aspectos que perseguimos al momento de crear el producto de sw** (Sea satisfacer a un cliente o la venta del producto, etc).

Por lo que al centrarse en el desarrollo del mínimo producto/mínima característica para saber si el producto que construiremos cubrirá las expectativas del usuario o las propias al desarrollarlo → Surge el planteo de la hipótesis: Mínimo desarrollo y mínimo esfuerzo para luego validarla haciendo que su desperdicio tienda a 0, evitando malgastar esfuerzo/energía en algo que no agregara valor/beneficio al producto.

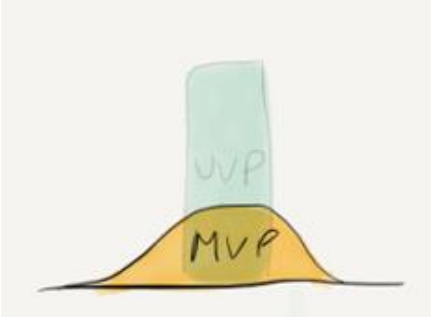
Para ello se utiliza la **técnica UVP (User Value Proposition – Propuesta de Valor para el usuario) o (Unique Value Proposition – Propuesta única de valor)** la cual está centrada en comprender las necesidades, deseos y problemas de los usuarios y en crear soluciones que satisfagan las necesidades de manera efectiva.

Al principio nos imaginamos una **hipótesis** o un valor que tiene el producto/servicio a desarrollar → Visión: Resolver esa hipótesis única.

- Si en base a está desarrollamos un producto completo con muchas funcionalidades y recursos utilizados y lo comercializamos, estamos ante el riesgo de que este producto no sea algo que el mercado/usuarios estén **demandando** o que no **cumpla las expectativas** del usuario exigente.

El **MVP (Minimum Viable Product – Producto Mínimo Viable)** surge en base a minimizar el esfuerzo para construir un producto que nuestro usuario quiere si es que la hipótesis es algo realmente demandada en el mercado.

- El MVP es un concepto de Lean Startup enfatizado en el impacto del aprendizaje en el desarrollo de nuevos productos.



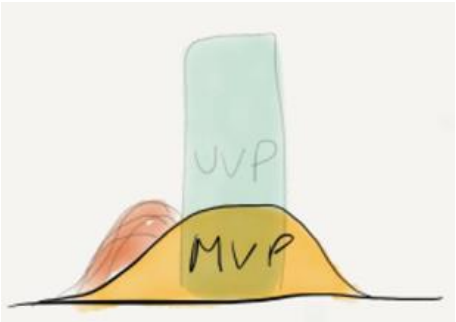
Idea del MVP: Trabajar con lo mínimo del producto que necesito obtener. Es decir, lo mínimo que debería tener el producto para que los (posibles) usuarios de este lo puedan evaluar y pueda indicar si el producto los atrae o no.

Objetivo: Comprobar que la **hipótesis** del producto a construir funciona o si hay que hacer cambios.

De esta forma la idea sería iniciar un circuito de aprendizaje, en donde, mediante la participación de los usuarios y la retroalimentación sobre si este cubre expectativas, sirve o si hay que hacer cambios en las características incluidas. "*No es vender el MVP, es validar la hipótesis*"

Puede que nuestra hipótesis sirva o que cada usuario que compruebe nuestro MVP solicite funcionalidades diferentes.

Si por cada usuario que prueba el MVP se obtiene un feedback diferente, quiere decir que no está tan claro el mercado en el cual el producto funcionará (Por eso las desviaciones) y significa que deberemos seguir trabajando sobre la visión del producto para encontrarlo.

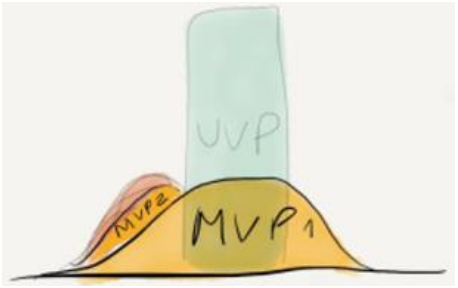


Pero si, al contrario, la retroalimentación de los usuarios indica una variación, pero todos apuntan a la misma. Existe sentido alguno de revisar la hipótesis ya que no es errónea, pero algunas características no están tan bien.

En base a el escenario actual, debemos redefinir el MVP en base al conocimiento adquirido, de esta forma creamos el nuevo MVP2.

Básicamente, pivoteamos hasta encontrar cual es el producto que nosotros queremos llegar a construir.

En base al circuito de retroalimentación, se vuelve a hacer, pero con el MVP2.



Una vez encontrado el MVP exitoso (feedback e investigación exitosas), surge el concepto de:

MMF (Minimun Marketable Feature – Característica mínima comercializable): Está enfocado en definir elementos mínimos que deben estar presentes en un producto o servicio para que sea comercializable.

Es la validación del producto en términos de la comercialización.

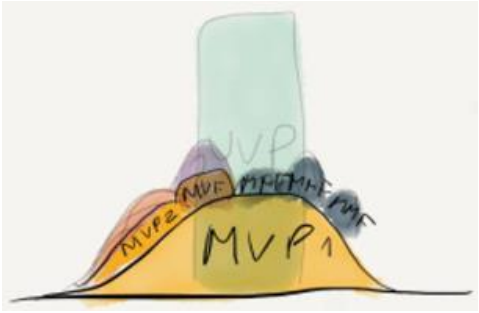
Ya no estamos parados en la hipótesis validando si el producto tiene mercado o no, ahora estamos definiendo cual es la pieza mínima que voy a tener que construir y comercializar para que el producto sea rentable.

Se lo construye en base a cuestiones mínimas para sacarlo rápidamente a la venta cuando el producto es atractivo y no está en el mercado, ya que cuanto más tiempo pase más riesgo se corre de que sea implementada antes. Todavía mantenemos la idea de aplicar el menor esfuerzo posible y no esfuerzos innecesarios antes de saber si el producto es o no comercializable.

Objetivo: Asegurarnos que este primer MMF sirve o no. Diferenciando en que ahora hablamos de una Feature, lo cual es más pequeño que el MVP, siendo lo mínimo que puedo sacar a la venta.

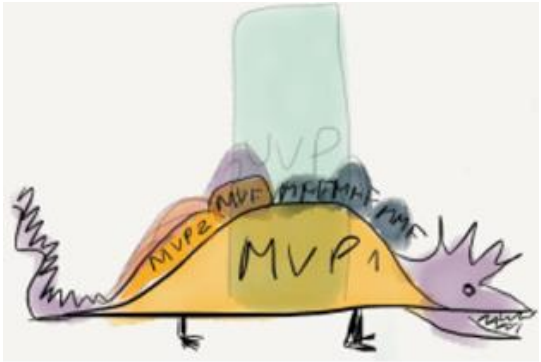
MVF (Minimun Viable Feature – Característica mínima viable): Surge al combinar el MVP con la MMF, se trata de la característica mínima que se puede construir e implementar rápidamente, utilizando recursos mínimos para probar la utilidad de esta.

Permite validar cual es la característica que hace la diferencia y la que será por la cual compren el producto.



Objetivo: Buscamos que esta Feature ofrezca un valor agregado, en donde la hipótesis se centra en una característica en vez de un producto.

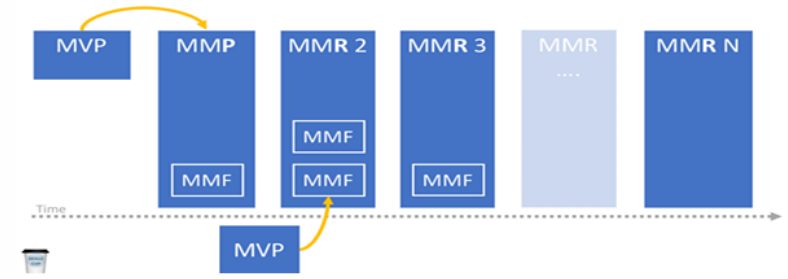
Si es exitosa se pueden desarrollar más MMF en esta área para tomar ventaja (ya que está comprobada), siendo una versión mini del MVP.



Modelo completo: Al seguir trabajando en el contexto de validar las hipótesis y ver como trabajaremos con cada característica, cuando se logra ajustar el producto en el mercado, se combinan MMF y MVP según el nivel de incertidumbre del negocio o las áreas en las cuales se está enfocando.

Dinosaurio Carpaccio: Obtenido al cortar cada una de esas piezas pequeñas en porciones destinadas a reducir el riesgo de ejecución/tecnología (Denominadas US). En donde estas porciones más pequeñas pueden tener un valor comercial tangible o no.

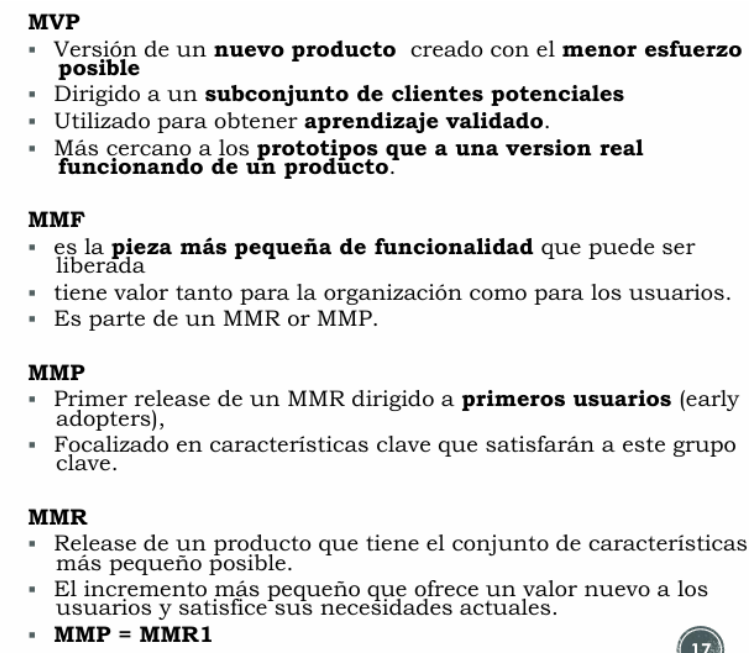
MRF (Minimun Release Feature – Características mínimas del release): Trata del reléase del producto que tiene el conjunto de características **más pequeño** posible. En donde el incremento más pequeño, ofrece un valor nuevo a los usuarios y satisface sus necesidades actuales.



MMP = MMR1 → Pueden salir más releases después, que mejoren el producto/reléase inicial.

El **MRF** se obtiene en varias iteraciones hasta que tengo una cierta cantidad de características a lanzar.

Relaciones Concretas:

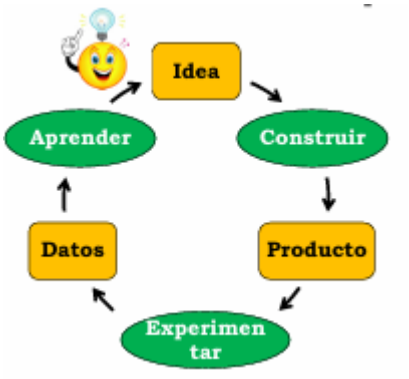


Errores comunes MVP vs MMF o MMP:

Valor vs Desperdicio: ¿Cuáles de nuestros esfuerzos crean valor y cuáles son desperdicio? Lean Thinking define la creación de valor como proveer beneficios a los clientes, lo demás es desperdicio.

- La productividad de un **Startup** no puede **medirse** en términos de cuanto se construye cada día, por lo que se debe medir en términos de averiguar la cosa correcta a construir cada día.

Ciclo de Feedback o Aprendizaje: El éxito no es entregar un producto, el éxito se trata de entregar un producto (o característica de producto) que el cliente usará.



La forma de hacerlo, es alinear los esfuerzos continuamente hacia las necesidades reales de los clientes.

The Build-Experiment-Learn Feedback Loop permite descubrir las necesidades del cliente y alinearlas metodológicamente.

La fase construir del MVP: Es importante saber que cuando construimos el MVP, su complejidad puede variar en complejidad, desde anuncios, explicaciones simples (pruebas de humo – Smoke Test), hasta prototipos tempranos. La idea es tener rápidamente y con el menor esfuerzo el MVP para poder validarlo y retroalimentar la hipótesis para entrar en el ciclo de aprendizaje.

Al hablar de MVP, surgen problemáticas sobre cómo hacemos para minimizar el desperdicio y el esfuerzo al construirlo. Para ello debemos:

- En caso de duda, simplificar.
- Evitar la construcción y promesa excesiva.
- Saber que cualquier trabajo adicional más allá de lo que necesita para comenzar el ciclo podría ser un desperdicio.

El punto central del **Loop de Feedbck** entonces, tiene que ver con **Experimentar**, siendo clave para trabajar en la construcción del MVP.

- Acá comienzan a surgir distintos planteos al momento no solo deconstruir un MVP sino de construir en si un nuevo producto.

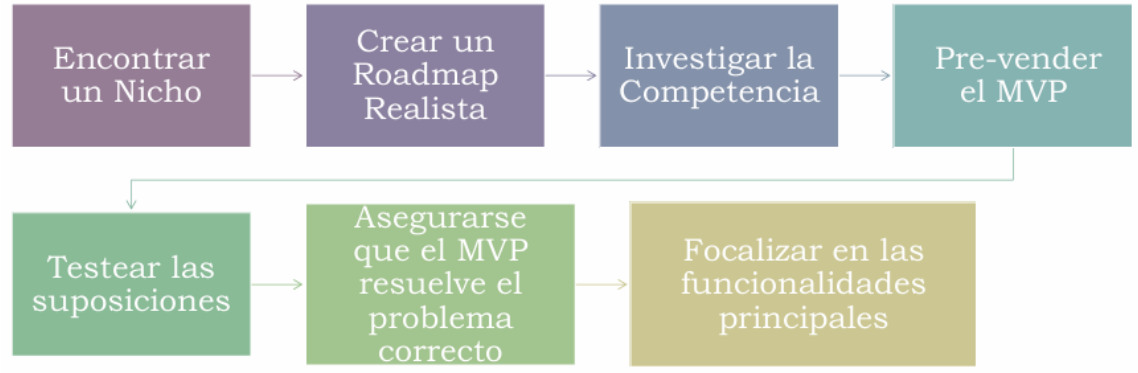
Audacia de cero – Incentivo: Al posponerse la experimentación con el MVP, surgirán resultados como una gran cantidad de trabajo desperdiciado, perdida de información esencial y el riesgo de construir algo muy poco significativo.

Por lo que debemos utilizar el MVP para **experimentar** con los primeros usuarios en el mercado verificando mi hipótesis probando todos los elementos disponibles comenzando por los más riesgosos.

Supuestos de "Saltos de Fe": Los elementos más riesgosos del plan / concepto de nuevo producto o startup se denominan supuestos de salto de fe. La mayoría de las personas no conocen una determinada solución (o incluso un problema); pero una vez que experimentan la solución, ¡No pueden imaginar cómo vivirían sin ella!

- **Hipótesis de valor:** Prueba si el producto realmente está entregando valor a los clientes después de que comienzan a usarlo → Métrica de prueba: Tasa de retención.
- **Hipótesis de crecimiento:** Prueba cómo nuevos clientes descubrirán el producto → Métrica de prueba: Tasa de referencia o Net Promoter Score (NPS).

Preparara un MVP: Para ello se sigue el siguiente RoadMap:



Scrum: Es un marco ligero que ayuda a las personas, equipos y organizaciones a generar valor a través de soluciones adaptables para problemas complejas.

- Es un framework de gestión ágil → Manifiesto ágil, basado en el empirismo (respeta sus tres pilares: Transparencia, Inspección y Adaptación) y el pensamiento Lean.
    - El empirismo afirma que el conocimiento proviene de la experiencia y la toma de decisiones basadas en lo que se observa.
    - El pensamiento Lean reduce los desperdicios y se centra en lo esencial.
- Emplea un enfoque iterativo e incremental para optimizar la previsibilidad y controlar el riesgo.

¿Por qué decimos que Scrum se apoya en los pilares del empirismo?

Transparencia: Tanto el proceso como el trabajo emergente deberá ser visible para cualquier involucrado. En Scrum, las decisiones importantes se basan en el estado percibido de sus tres artefactos formales → Sin transparencia conducirán a decisiones que disminuyen el valor y aumentan el riesgo.

"La transparencia permite la inspección, ya que la inspección sin transparencia genera engaños y desperdicios."

Inspección: Los artefactos y el progreso hacia objetivos acordados deben ser inspeccionados con frecuencia y diligentemente para detectar varianzas/problemas potencialmente indeseables → Scrum plantea eventos para llevar la cadencia.

"'La inspección permite la Adaptación, ya que la inspección sin adaptación se considera inútil. Los eventos de Scrum están diseñados para provocar cambios."

Adaptación: Ante cualquier desviación en un aspecto del proceso o si el mismo es inaceptable, el proceso que se está aplicando o los materiales que se producen deben ajustarse. El ajuste se realizará lo antes posible para minimizar la desviación adicional.

- Se vuelve **difícil** cuando las personas involucradas no están empoderadas o no poseen capacidad para autogestionarse. El equipo de Scrum deberá adaptarse al momento que aprenda algo nuevo mediante la inspección.

El equipo Scrum – Scrum Team: Es un pequeño equipo de personas, que consta de lo siguiente:

- **Scrum Master:** Es el responsable de establecer Scrum tal como se define en la Guía de Scrum, ayuda a todos a comprender la teoría y la práctica de Scrum, tanto dentro del equipo como en toda la organización. Es el responsable de la efectividad del Scrum Team, permitiéndole a todo el equipo que mejore sus prácticas dentro de este marco – Son verdaderos lideres que sirven al equipo Scrum y a toda la organización.
- **Product Owner – Dueño del producto:** Es el responsable de maximizar el valor del producto resultante del trabajo del equipo de Scrum. También es responsable de la gestión eficaz del Product Backlog – Representa al equipo.
- **Developers – Desarrolladores:** Personas del equipo Scrum, comprometidas a crear cualquier aspecto de un incremento útil (funcional) en cada Sprint.

Es un equipo pequeño de profesionales **multifuncionales** que está enfocado en un objetivo a la vez (Producto). En donde c/u de los miembros tienen todas las habilidades necesarias para **crear valor** en cada Sprint a su vez son **autogestionados** por lo que internamente deciden quién hace qué, cuándo y cómo.

- También es demasiado grande como para completar un trabajo significativo dentro de un Sprint (10 o menos personas).
- Es el responsable de todas las actividades relacionadas con los productos, desde la colaboración, verificación, mantenimiento, operación, experimentación, investigación y desarrollo, y cualquier otra cosa que pueda ser necesaria.

Eventos de Scrum: Son cuatro eventos formales para la inspección y adaptación dentro de un evento contenedor → Sprint.

- **Sprint:** Son eventos de longitud fija de un mes o menos para crear consistencia. Un nuevo Sprint comienza inmediatamente luego de la conclusión del anterior.

Dentro de está ocurre todo el trabajo necesario para alcanzar el objetivo del producto: Sprint Planning, Daily Scrum, Sprint Review y Sprint Retrospective.

Durante el Sprint:

- No se hacen cambios que pongan en peligro el **Objetivo Sprint.**
- La calidad no disminuye.
- El trabajo pendiente del producto se refina según sea necesario.
- El alcance se puede clarificar y renegociar con el Propietario del Producto a medida que se aprende más.

Permiten la previsibilidad al garantizar la inspección y adaptación del progreso hacia un objetivo del producto.

Formas de Pronosticar el progreso: Gráficos de Burn-Downs, Burn-Ups o flujos acumulativos.

Un Sprint podría ser cancelado si el objetivo de este se vuelve obsoleto. Solo el PO tiene la autoridad para cancelarlo.

- **Sprint Planning – Planificación de Sprint:** Inicia el sprint estableciendo el trabajo que se realizará para el mismo. Este plan resultante es creado por el trabajo colaborativo de todo el equipo de Scrum.
  El PO se asegura que los asistentes estén preparados para discutir los elementos de trabajo pendiente de producto más importante y cómo se asignan al objetivo del producto. Abarca temas como: ¿Por qué este Sprint es valioso?, ¿Qué se puede hacer este Sprint?, ¿Cómo se realizará el trabajo elegido?
- **Daily Scrumt – Scrum diario:** Su propósito es inspeccionar el progreso hacia el **objetivo sprint** y adaptar el Sprint Backlog según sea necesario, ajustando el próximo trabajo planeado. Suele durar 15 minutos como máximo para los desarrolladores del equipo Scrum.
  Estos mejoran la comunicación, identifican impedimentos, promueven una rápida para la toma de decisiones, y en consecuencia, eliminan la necesidad de otras reuniones**.**
- **Sprint Review – Revisión del Sprint:** Su propósito es inspeccionar el resultado del Sprint y determinar futuras adaptaciones. Es donde el equipo presenta los resultados de su trabajo a las partes interesadas clave y se discute el progreso hacia el objetivo del producto.
  Aquí es donde el equipo y las partes interesadas revisan lo logrado en la Sprint y lo que ha cambiado en su entorno. En base a esto, se decidirá que hacer a continuación. Todo trabajo pendiente del producto se puede ajustar para satisfacer nuevas oportunidades.
- **Sprint Retrospective – Retrospectiva del Sprint:** Su propósito es planificar formas de aumentar la calidad y la eficacia. En donde el equipo inspeccionará como fue el último Sprint respecto a individuos, interacciones, procesos, herramientas y su DoD. También las suposiciones que los desviaron son identificadas y se exploran sus orígenes. El equipo analiza qué fue bien durante el Sprint, qué problemas encontró y cómo esos problemas fueron (o no) resueltos.
  Aquí el equipo identifica los cambios más útiles para mejorar su eficacia. Las mejoras más impactantes se abordan lo antes posible. Incluso se pueden agregar al Sprint Backlog para el próximo sprint.
  Este evento concluye el Sprint, tiene una duración de hasta máximo de 3 horas para un Sprint de un mes.

Artefactos de Scrum: Representan trabajo o valor. Están diseñados para maximizar la transparencia de la información clave. Cada artefacto contiene un compromiso para garantizar que proporciona información que mejora la transparencia y el enfoque con el que se puede medir el progreso:

- **Objetivo del producto** → Para el trabajo pendiente del producto.
- **Sprint Goal** → Para el Sprint Backlog.

- **Definition of Done** → Para el Incremento.

Estos compromisos existen para reforzar el empirismo y los valores de Scrum tanto para el equipo como para los involucrados.

Pila del Producto – Product Backlog: Lista emergente y ordenada de lo que se necesita para mejorar el producto → Es el trabajo pendiente.

- Está lista puede ser hecha por el equipo dentro de un Sprint y se consideran listos para su selección en una **Planning**. → Para ello deben cumplir con el Definition of Ready. Adquieren el grado de transparencia luego de la refinación.
- **Refinación de backlog:** Se descomponen y definen aún más los elementos de trabajo pendiente del producto en artículos más pequeños y precisos.
- Dentro de un PB nos encontramos cualquier forma de **productos, NO tareas,** si no características, que el producto deberá tener, es decir: US, Epicas, Temas, Spikes, Restricciones, Tech Stories, Deuda Tecnica y Defectos (Post Iteraciones) → Meles.

Objetivo del Producto – Product Goal: Describe un estado futuro del producto que puede servir como objetivo para el equipo contra el cual planificar. Forma parte del PB, en donde el resto de trabajo pendiente del producto surge para definir **"qué"** cumplirá el objetivo del producto.

Pila del Sprint – Sprint Backlog: El trabajo pendiente de Sprint se compone del **objetivo sprint** (Por qué), el conjunto de elementos de trabajo pendiente de producto seleccionado para el sprint (qué), así como un plan accionable para entregar el incremento (cómo).

- El trabajo pendiente de Sprint es un plan por y para desarrolladores. Siendo una imagen visible y en tiempo real donde estos deberán realizar durante el Sprint para cumplir el **Objetivo Sprint**.
- El **Sprint Backlog (SB):** Es actualizado a lo largo del Sprint a medida que se aprende más. Debiendo tener los suficientes detalles como para poder inspeccionar su progreso en la **Daily Scrum.**
- **Objetivo Sprint – Sprint Goal:** Es el único objetivo para la Sprint. Es más, un compromiso de los desarrolladores, proporcionando flexibilidad en términos de trabajo exacto necesario para logarlo. Esté, crea coherencia y enfoque, animando a el trabajo en conjunto.
    - Se crea durante la Planning, y es agregado al trabajo pendiente de Sprint.

Incremento – Increment: Cada incremento es aditivo a todos los Incrementos anteriores y verificado a fondo, asegurando que todos funcionen juntos, de esta forma proporciona el valor, el incremento debe ser utilizable.

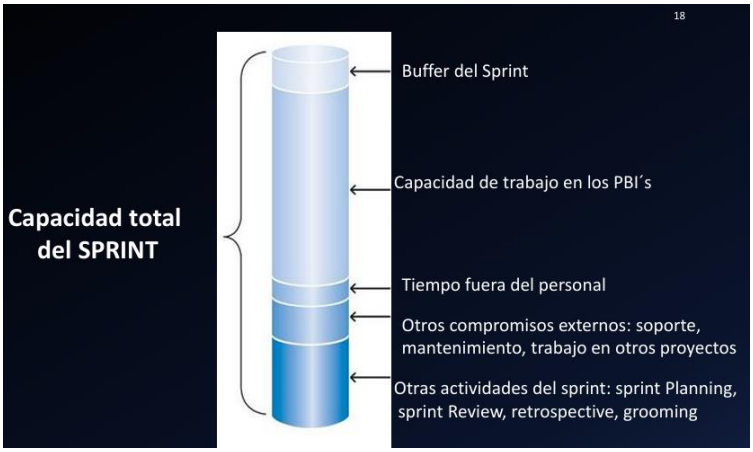
- Se pueden crear varios incrementos dentro de una Sprint, en donde la suma de estos es presentada en la **Revisión Sprint** apoyándose en el empirismo.
- El trabajo no puede ser considerado parte de un incremento a menos que cumpla con el DoD (Definition of Done).

Definition of Done – DoD – Definición de Hecho: Es una descripción formal del estado del Incremento cuando cumple con las medidas de calidad requeridas para el producto.

Capacidad del Equipo en un Sprint: Métrica utilizada por Scrum, siendo la única estimable y utilizada para ver cuanto compromiso de trabajo puede asumir el equipo en un determinado sprint.

Lo vemos reflejado en la Planning, donde determinamos la capacidad del equipo, teniendo en cuenta está, es contrastada en base a las US del PB que podrán pasar por el Sprint Backlog y hasta donde nos podemos comprometer en un determinado Sprint.

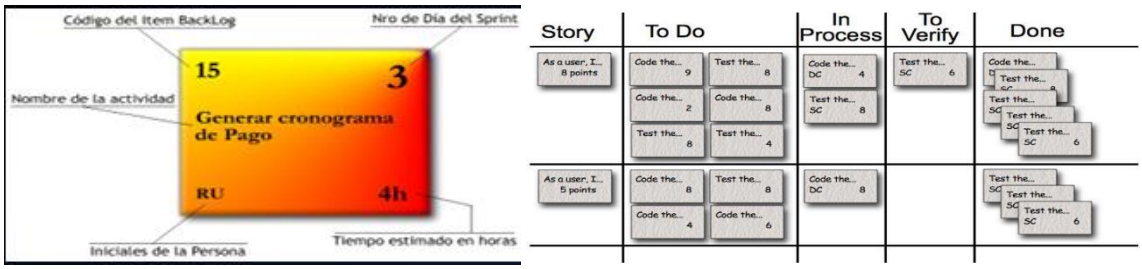
Se estima porque estamos comenzando el sprint y observamos cuantas horas reales podremos dedicarle al trabajo en el Sprint. La capacidad se estima en horas ideales/Story Points.



Por lo que la primera variable de definición es **cuánto** va a durar el sprint, la segunda será el **cálculo de capacidad.** Finalmente se acordará el objetivo en base a esas tres variables se indica qué cosas van a empezar a pasarse desde el Product Backlog hasta el Sprint Backlog.

Herramientas de Scrum

Taskboard: Hay una configuración de tablero básica que propone Scrum, donde de ahí se pueden realizar ajustes/variaciones según el equipo.



Scrum recomienda la estimación mediante Story Points, ya que son estimaciones **sobre** el producto y no sobre el trabajo.

La configuración básica de un tablero de Scrum tiene 3 columnas:

- US Comprometidas.
- In process – O doing → Lo que se está haciendo.
- Done → Lo terminado.

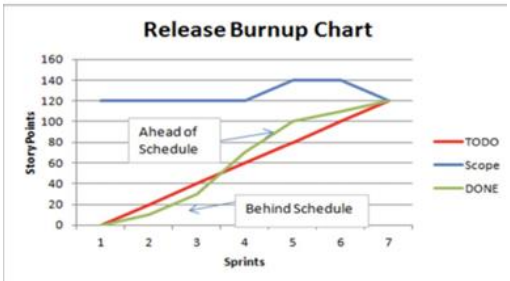
Si un tablero tiene pocas tareas, proviene de una US estimada con muchos SP, por lo que es muy pesada.

Con granularidad fina tendremos más US y una mayor posibilidad de poder maniobrar y asignarlas de forma que se puedan terminar antes del fin de una Sprint.

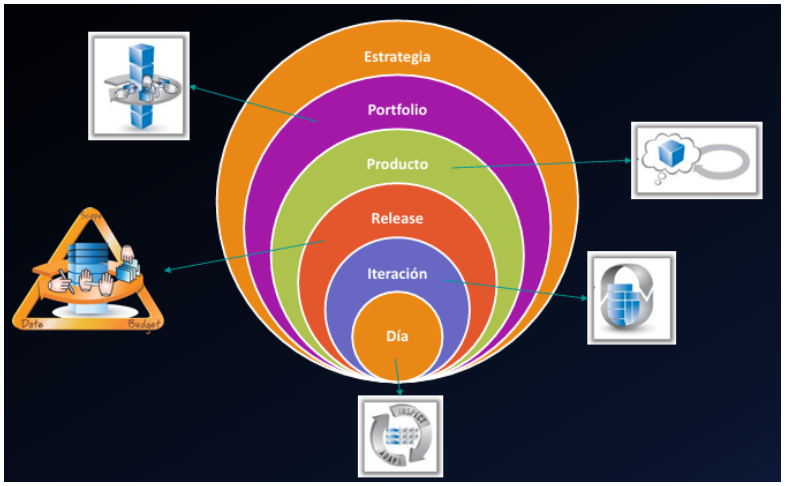
Sprint Burndown Charts: Son gráficos que permiten la visualización del trabajo. En el eje "y" se muestran los puntos de historia restante y en el eje "x" se colocan los días.

- No conviene realizarlo con las horas de trabajo, ya que no es **representativo** del esfuerzo o logro obtenido.
- Sirven para calcular **velocidad** y se descartan al final del Sprint.

Sprint Burnup Chart: Son gráficos que no se descartan al final de un sprint, permiten realizar un seguimiento de como se va trabajando en el producto a lo largo de los sprints.



Niveles de planificación: No existe producto de sw que pueda ser terminado en un Sprint, para ello existen diferentes niveles de planificación.



El más pequeño es la daily, en donde sincronizamos y vemos la situación de realizar cambios, ajustes o adaptaciones en la diaria para podre cumplir con el compromiso.

En la iteración (Corresponde al Planning). Cada espacio de tiempo que comiena termina y genera un incremento (Iteración) en scrum se llama Sprint.

En la release (vesión del producto que tiene un conj. de características) implicando definir cuántos sprint voy a necesitar para entregar la cantidad de características definidas previamente.

En la planificación de producto, es la sumatoria de releases, que se irán generando a lo largo del tiempo. Implicando la toma de decisiones funcionales y no funcionales e ir definiendo en que release se va a ir entregando cada característica.

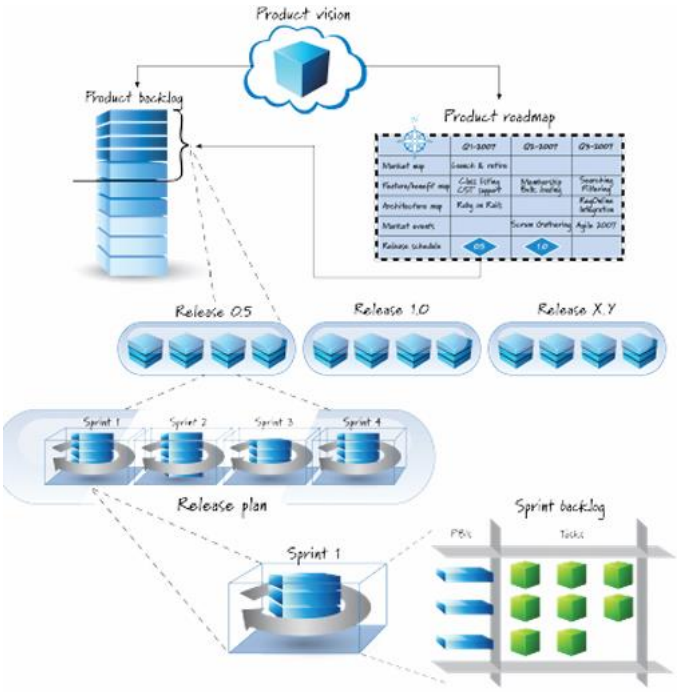
El portfolio se refiere a los diversos productos, los cuales clasificaran en priorizados o descontinuados.

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, Stakeholders	Balancear continuamente el valor de cliente y la calidad global con las restricciones de alcance, cronograma y presupuesto	Plan de Release
Iteración	Cada iteración (de 1 semana a 1 mes)	Equipo Scrum	Que aspectos entregar en el siguiente sprint	Objetivo del Sprint y Sprint Backlog
Día	Diaria	Equipo Scrum (al menos los que trabajan en IPB)	Cómo completar lo comprometido	Inspección del progreso actual y adaptación a la mejor forma de organizar el siguiente día de trabajo

¿Qué y cuándo estimar?

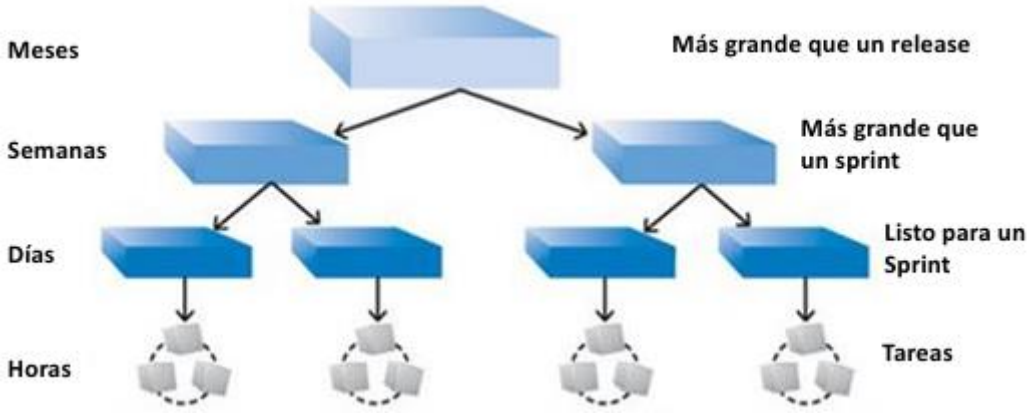
Depende puramente del nivel de granularidad que se esta manejando. Pero existen ciertas recomendaciones para estimar.

- **Portfilio:** Estimación en talles de remeras.
- **Product Backlog:** Estimación mediante Story Points, también se puede con talles de remera. Pero mientras mejor priorizados estén los ítems del PB, más fácil será su priorización.
- **Sprint Backlog:** Se utilizan historias con DoR, estimadas en Story Points. Luego el equipo es quien decide si dividirá las historias en tareas o no. Si lo hacen, las tareas se estiman en **horas ideales.**

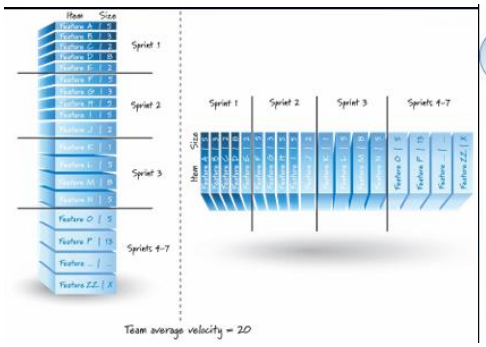


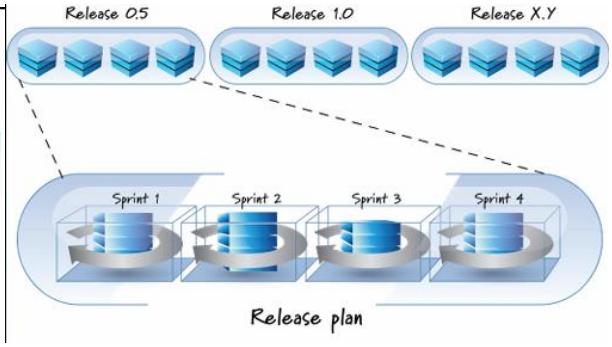
Planificación del Release: La salida de la planificación del release son las características que quiero desarrollar (en un rango), cuantos sprints serán necesarios para terminar el reléase, la capacidad estimada del equipo, objetivos y características de cada sprint. En función de eso se realiza una planificación, al ocurrir una variación, se deberá replanificar.

Distintos niveles de granularidad: Algo más grande que una US, no entrará en una Sprint. Si son características muy grandes, no entrará siquiera en un Release, se deberá repartir en varias releases.



Las tareas **NO** están en el Product Backlog, si no el Sprint Backlog. Dentro del Product Backlog tenemos diferentes niveles de granularidad en donde las releases se separarán en Sprints.

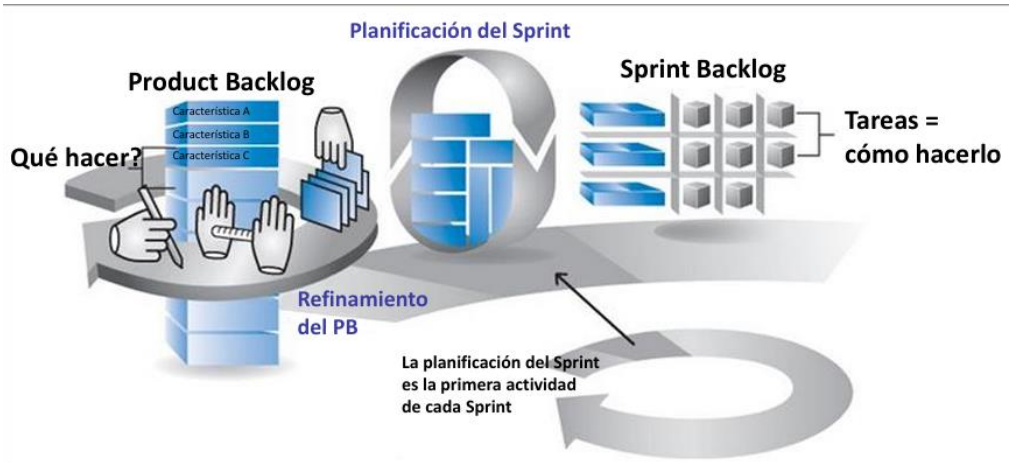




En la imagen observamos que la velocidad promedio es de 20 Story Points, por lo que se sumarán características hasta alcanzar ese valor y se le asigna un Sprint. En los primeros sprint, no sabremos bien la velocidad del equipo, hasta que vayamos logrando experiencia en las sucesivas iteraciones.

Cadencia de los Sprints: Existen varias formas de armar las releases, algunas empresas se liberan luego de terminar varios sprint, en otras hay una reléase por sprint y en otras luego de cada feature.

Planificación de Iteración – Sprint planning: Este proceso debe tener como resultado la configuración del tablero, teniendo como entrada el objetivo del sprint, su duración, la capacidad estimada del equipo. En donde el equipo dividirá en tareas cada US, estimándolas en horas. El proceso de planificación del sprint termina con el Sprint Backlog configurado.



Métricas en ambientes ágiles: Existe una regla de Oro ágil sobre métricas, y es:

"La medición es una salida, no una actividad"

Para la elección de métricas nos centramos en dos principios ágiles:

- Nuestra mayor prioridad es satisfacer al cliente por medio de entregas tempranas y continuas de sw **valioso**.
- El sw **trabajando** es la principal medida de **progreso.**

Running Tested Features – RTF: Se cuenta: Cantidad de características funcionando que se han desarrollado en un sprint. Es solamente calculada en Story Points, no se utiliza mucho ya que fue reemplazada por la velocidad.

Capacidad: Se Estima en: Horas de trabajo disponibles por día * días disponibles de iteración.

Velocidad: Se **cuenta,** es una medida métrica del progreso de un equipo, es calculada mediante el numero de Story Points que el equipo completa durante una iteración. Se cuentan los SP de las iteraciones **completas**. Está métrica permite la corrección de errores de estimación.

Unidad N°3: Gestión de Software como Producto

Al pensar en software pensamos en un conjunto de: Programas, Procedimientos, Reglas, Documentación y Datos. En donde 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: Estos deben su origen a:

- 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.

Gestión de Configuración de Software - SCM: Es una actividad paraguas ya que se aplica en distintas partes del proyecto, no solamente en la gestión de versiones. Tiene como **propósito** mantener la **integridad** del producto de sw.

Es considerada una disciplina de **soporte** ya que ocurre transversamente a lo largo de todo el proyecto, pero trasciende para dar soporte al producto en todo su ciclo de vida. Pero también tiene aplicación en diferentes disciplinas como:

- Control de calidad de proceso.
- Control de calidad de producto.
- Prueba de software.

Definición forma:

"Una disciplina que aplica dirección y monitoreo administrativo 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"

Items de Configuración (IC): Son todos aquellos productos de trabajos, artefactos o

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

- 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.

Problemas en el manejo de componentes:

- Perdida 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.

Mediante la Gestión de Configuración de Software se busca que el producto de sw sea integro y ante cambios, mantener un control de estos. Por lo que se le asocia el concepto de **versión,** ya que cada IC debe tener su versión.

Conceptos Clave para SCM: Aquí tendremos conceptos muy importantes los cuales son:

- **Ítem de Configuración (IC):** Son todos y cada uno de los artefactos que forman parte del producto o del proyecto, que puedan sufrir cambios o necesitan ser compartidos entre los miembros del equipo y sobre los cuales necesitamos conocer su estado y evolución.
  Ejemplos:

- Plan de CM
- Propuestas de Cambio
- Visión
- Riesgos
- Plan de desarrollo
- Prototipo de Interfaz
- Guía de Estilo de IHM
- Manual de Usuario
- Requerimientos
- Plan de Calidad
- Arquitectura del Software
- Plan de Integración
- Planes de Iteración
- Estándares de codificación
- Casos de prueba
- Código fuente
- Gráficos, iconos, ...
- Instructivo de ensamble
- Programa de instalación
- Documento de despliegue
- Lista de Control de entrega
- Formulario de aceptación
- Registro del proyecto

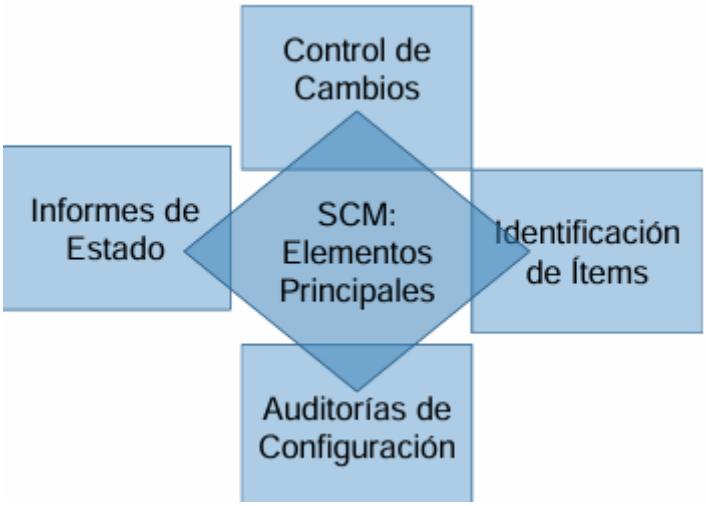
- **Versión:** Es la forma particular de un artefacto en un instante o contexto dado. El control de versiones se refiere a la evolución de un único IC, o de cada IC por separado.
- **Repositorio:** Aquí se mantiene la historia de cada IC. Es el contenedor de IC, tiene una estructura para mantener un orden y la integridad, a su vez, esta lo beneficia otorgando seguridad, control de acceso, políticas de backup's y todo aquello que se aplica sobre un repositorio.
  El equipo comparte el repositorio por lo que no existirán problemas sobre situaciones en donde no se pueda trabajar porque uno solo tenga acceso a cierta carpeta.
  - Puede ser una o varias bases de datos. Su uso esta dado para evaluaciones de impacto de los cambios propuestos.

  Tipos de repositorios:
  - **Centralizados:** Un servidor contiene todos los archivos con sus versiones. Los Admins tienen mayor control sobre el repositorio → Ante una falla de servidor estamos al horno.
  - **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 centralizaod.
- **Línea base (LB):** Puede contener en un momento de tiempo un solo IC o un conjunto de ítems. Está conformada por IC que se toman como referencia, los cuales han sido considerados estables en base a los niveles de revisión y aprobación.
  La LB debe tener una identificación **unívoca**, debe 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 LB. Por lo que:
  - Sirve como base para desarrollos posteriores y puede cambiarse sólo a través de un procedimiento formal de control de cambios.
  - Permite ir atrás en el tiempo y reproducir el entorno de desarrollo en un momento dado del proyecto.
  - Aquello que no forme parte de está, podrá ser modificado con total libertad.

  Utilidad: Funciona como un punto de referencia, útil para realizar rollbacks o saber que se pone en producción. Permite saber cuál era la última situación estable en un momento de tiempo y cómo se fue evolucionando.
  Tipos:
  - **De especificación:** Son de requerimientos, diseño. No poseen código, poseen información de la ingeniería del producto.

    - **Operacionales:** LB que ya tienen código, han pasado por un control de calidad definido previamente.
- **Ramas – Branch:** Al realizar la bifurcación de la línea donde están los IC, normalmente esta es la LB, pero no es considerada buena práctica trabajar de accionar sobre esos IC situados en la **rama principal o Master**. Se utilizan bifurcaciones para no trabajar sobre la Master, estas pueden ser descartadas o integradas y son utilizadas para experimentación entre otras. Al aceptar una bifurcación con la rama Master, estamos en presencia de un **Merge,** ante un conflicto se resuelve mediante **Diff**. Todas las ramas deberían eventualmente integrarse a la principal o ser descartadas.

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.



Indicación de ítems: Corresponde la identificación univoca 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.

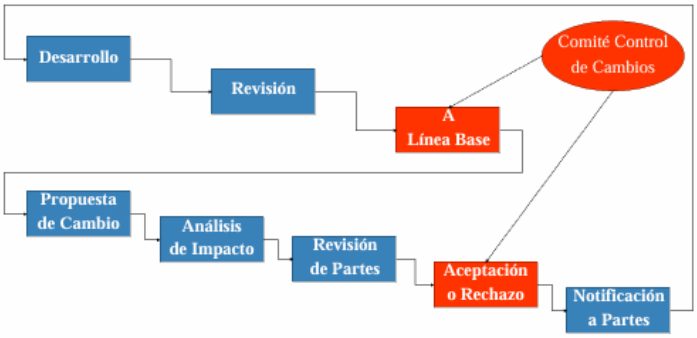
El Gestor de configuración es el rol que arma la estructura del repositorio y es el responsable de poner a disposición el repositorio. Las actividades involucradas son:

- Identificación univoca 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 ítem tiene un ciclo de vida distinto. Esto es lo que se habla al principio del resumen de ciclo de vida del producto y ciclo de vida del proyecto. Entonces, al asignarle un lugar en el repositorio a estos ítems deberemos tener en cuenta qué tipo de ítems es.

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

Control de cambios: Asociado a las LB. **Busca mantener la integridad de las LB**. Para cambiar la LB, esta debe pasar por un proceso formal de control de cambios. En base a esto decimos que los IC que **NO** son LB se pueden modificar sin tanto problema.



Proceso de control de cambios: El cambio debe ser autorizado, una vez se autoriza se deberá verificar que el cambio sea el establecido previamente y se autorizó.

Acá se definen y se mantienen las LB a lo largo del tiempo.

Se realiza también un análisis de impacto/evaluación de este, que determina si el comité del control de cambios va a autorizar un cambio o no.

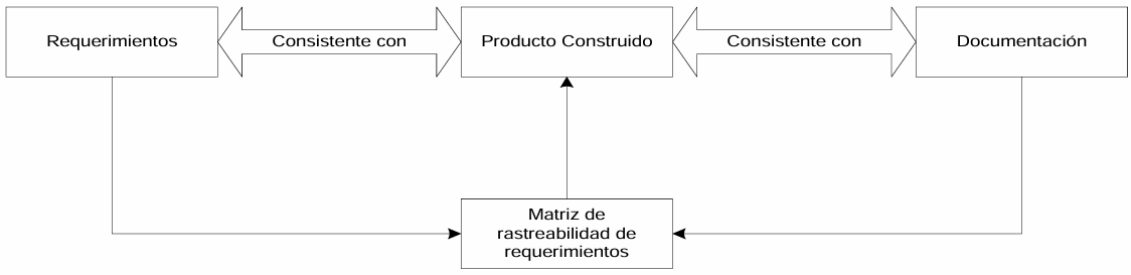
El control de cambios:

- Tiene su origen en un requerimiento de cambio a uno o varios IC que se encuentran en una LB.
- 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: Estas requieren un plan, en donde se realizan 2 auditorías: física y funcional.



Auditoría física de configuración (PCA): Asegura que lo que está indicado para cada ICS en la LB o en la actualización se ha alcanzado realmente.

- Hace **verificación.**
- Lo que **se audita es un LB.**

Auditoría funcional de configuración (FCA): Evaluación independiente de los productos de sw, controlado que la funcionalidad y performance reales de cada IC sean consistentes con la especificación de requerimientos.

- Hace **validación.**
- Ya qué **si la auditoría física no sale bien, la funcional no se hace.**

Validación: El problema es resulto de manera apropiada que el usuario obtenga el producto correcto.

Verificación: Asegura que un producto cumple con los objetivos preestablecidos, definidos en la documentación de LB. Todas las funciones son llevadas a cabo con éxito y los test cases tengan status “ok” o bien consten como “problemas reportados” en la nota de reléase.

Informes de estado: Su 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 situaciones de la gestión de configuración.

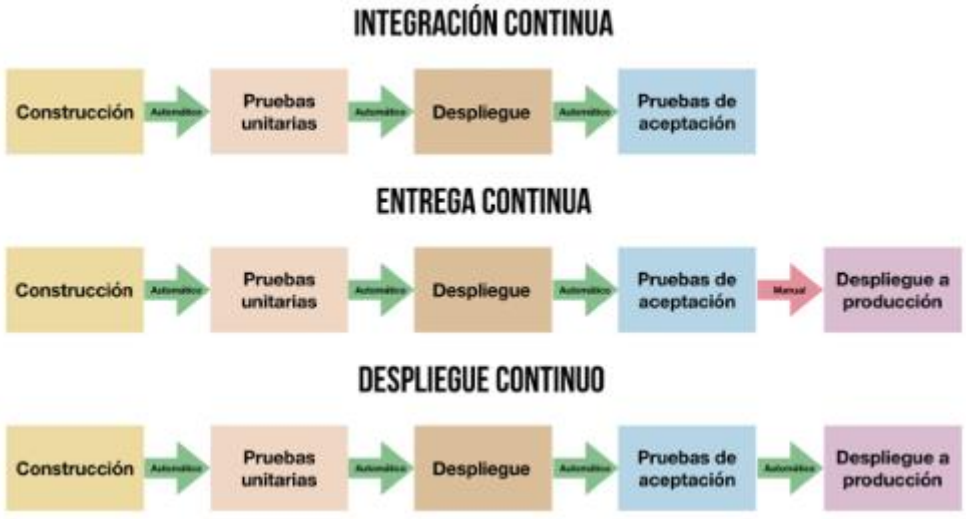
El **reporte básico** es el **inventario**, que lista todos los IC que tiene el repositorio con su estado en un momento de tiempo o listar cuántas solicitudes de cambios se atendieron en un determinado periodo de tiempo. **La mayoría de los reportes se obtienen de manera automática.**

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 LB durante el ciclo de vida.

Plan de gestión de configuración: El plan debe contener respuestas a las preguntas de cómo voy a hacer las 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 de repositorio, cómo se conforma el comité de control de cambios.

Evolución de la gestión de configuración de software: El SCM permitió que los equipos evolución en su forma de construir sw y que traten de hacer un proceso que sea cada vez más productivo y para logarlo se tratará de automatizar la mayor parte que se puedan de los procesos.



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.

Las 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 brinda mucha importancia al testing y que esté lo más automatizado posible.

La **integración continua,** que es lo primero que se hacer, es que cada desarrollados en su entorno trabaja, realiza pruebas unitarias, desarrollando con algo que se llama **TDD (Desarrollo conducido por testing)** al terminar ese componente de código y lo probó y sabe que funciona, se sube a un repositorio de integración.

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 sw en ambientes ágiles: Va de la mano con el manifiesto ágil porque recordemos que se buscaba estar siempre preparados para el cambio y justamente, la gestión configuración busca responder a los cambiar 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 sw 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.

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

- **Auditorías de configuración** es una actividad que **podrías no usar** si el equipo ágil lo considera así, es decir, no necesita opinión externa. Es opcional y dependa del equipo usarla o no.

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.

Tips para SCM en ágil:

- 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.