RESUMEN INGENIERÍA DE SOFTWARE

Software:

Concepto:

No es solamente el código, es también la documentación que lo acompaña, el conjunto de ambas es lo que define el software. *"Es un set de programas y la documentación que la acompaña"*.

Tipos:

- Sistemas de Software
- Utilitarios
- Software de Aplicación (lo que se ve en ISW)

Software vs Manufactura:

Manufactura: es un proceso de fabricación hecho con las manos (o en líneas de producción en una fábrica).

No es comparable porque:

- El SW ***no se produce en masa***, uno no produce un SW igual a otro en ningún contexto, aun cuando pareciera que son similares.

- El SW ***no es tangible***

- El SW ***es menos predecible***

- El SW ***es difícil de materializar en términos concretos***, sobre todo cuando se quiere comparar con un proceso de construcción de un producto final.

- El SW ***no esta gobernado por las leyes de la física*** (simplemente esta hecho con la materia gris de las personas).

- El SW ***no se gasta***, es decir de que el SW con el tiempo siempre funciona igual, lo que si cambia con el tiempo son los requerimientos o el funcionamiento del negocio o etc, pero parados en un mismo contexto a lo largo del tiempo el SW siempre va a funcionar igual.

Problemas con el desarrollo de SW:

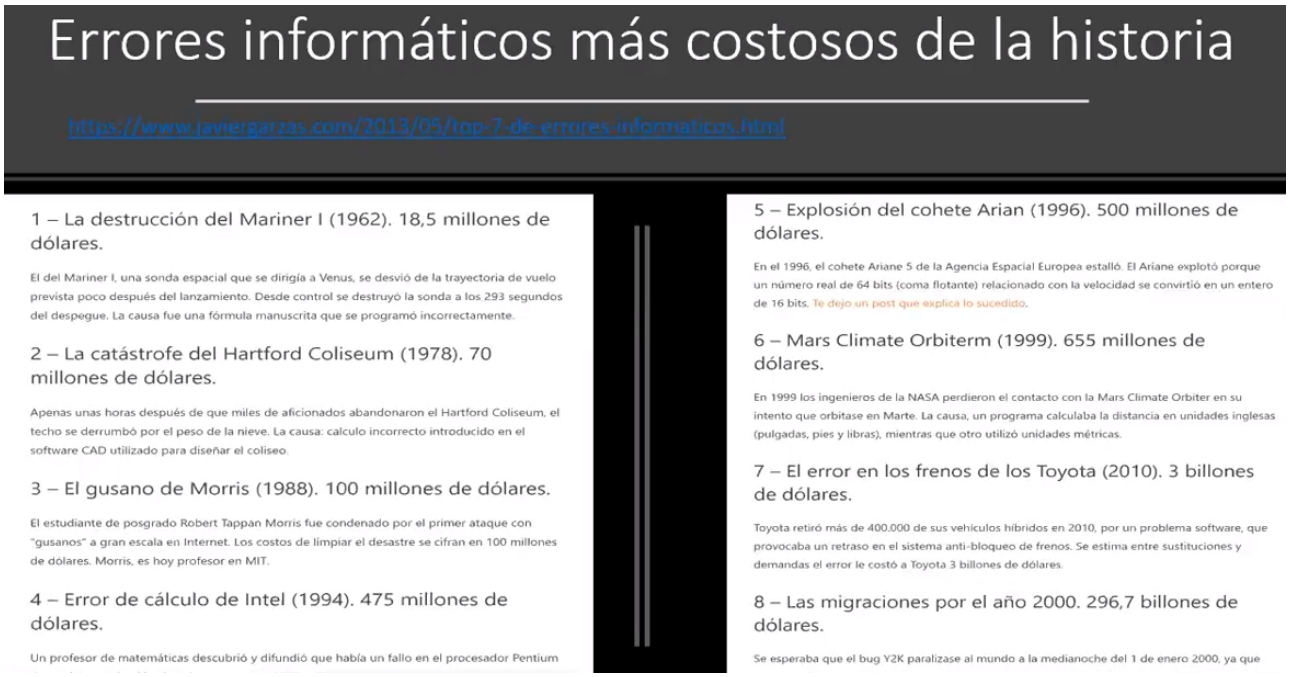
- La versión final del producto no satisface las necesidades del cliente, es decir, es difícil construir un producto que cumpla con todos los requerimientos o con las necesidades del cliente. Las metodologías agiles tratan de minimizar eso.

- No es fácil extenderlo o adaptarlo. Sobre todo cuando el producto o su arquitectura no está pensado para ser escalable sino para ser robusto en el tiempo, por lo que al querer agregar más funcionalidad en otra versión es casi imposible.

- Mala documentación

- Mala calidad

- Mas tiempos y costos presupuestados.



Motivos por los cuales nos va mal cuando construimos SW:

- Requerimientos Incompletos
- Falta de involucramiento del usuario
- Falta de recursos
- Expectativas poco realistas
- Falta de apoyo de la Gerencia
- Requerimientos Cambiantes

Motivos por los cuales nos va bien cuando construimos SW:

- Involucramiento del usuario
- Apoyo de la Gerencia
- Enunciado claro de los requerimientos
- Planeamiento adecuado
- Expectativas Realistas
- Hitos Intermedios
- Personas Involucradas Competentes

Origen del Concepto de Ingeniería de SW:

En 1987, Parmas definió a la ingeniería en SW como *"multi-person construction of multi-version software"*. Justamente porque construir SW no es solamente saber programar, sino tratar de abarcar todos los problemas mencionados anteriormente.

Para eso, la **IEEE** creo un cuerpo de conocimiento denominado ***Cuerpo de Conocimientos de la Ingeniería de SW (SEBOK)***, donde esta conformado por 15 areas de conocimiento.

Dentro de las 15 areas de conocimiento las que vamos a ver son:

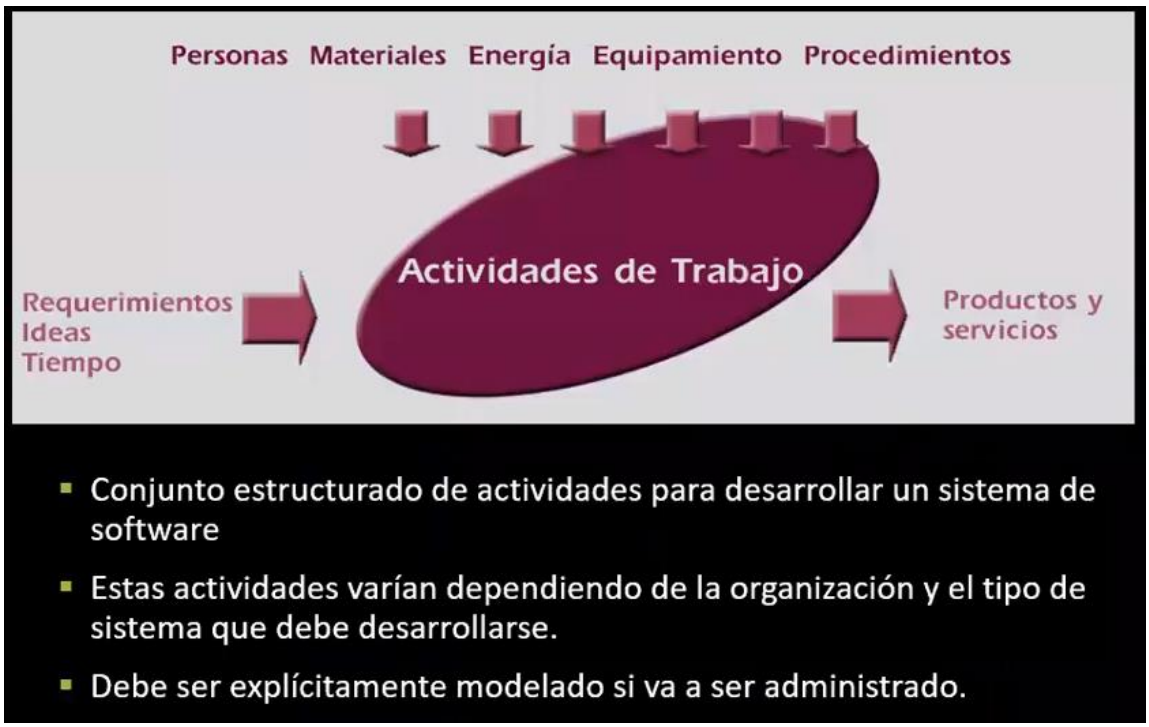
- **Disciplinas Técnicas:**
  - Testing

- **Disciplinas de Gestión:**
  - Planificación de Proyecto
  - Monitoreo y Control de Proyectos

- **Disciplinas de Soporte:**
  - Gestión de Configuración de Software
  - Aseguramiento de Calidad
  - Métricas

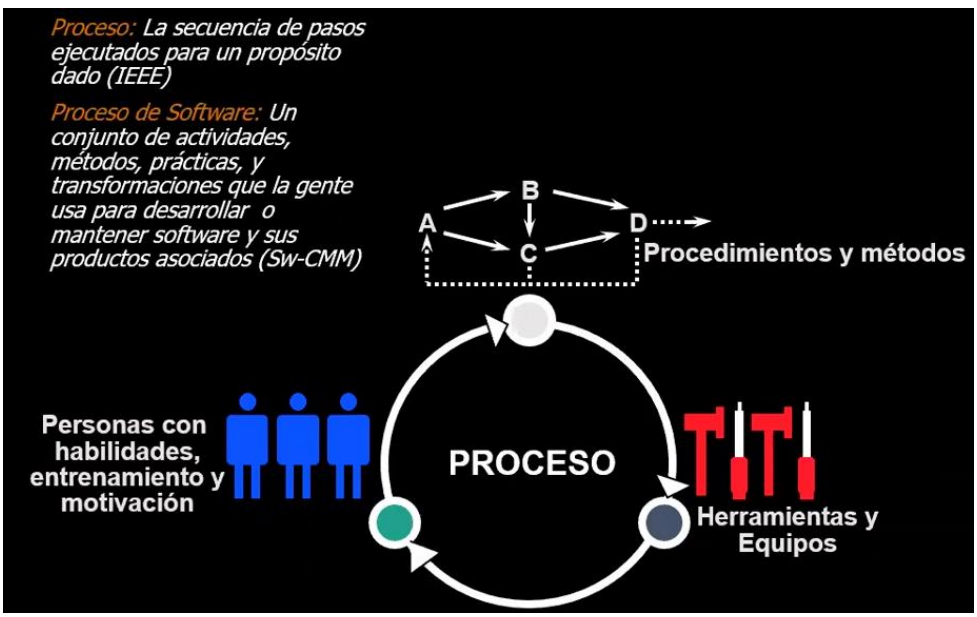
Proceso de Software:

Concepto: es un conjunto de actividades donde nosotros nos organizamos en dichas actividades para ejecutarlas con personas y con herramientas para finalmente obtener como salida productos y servicios que básicamente, en nuestro caso, es el Software.



Definición de un Proceso de Software:

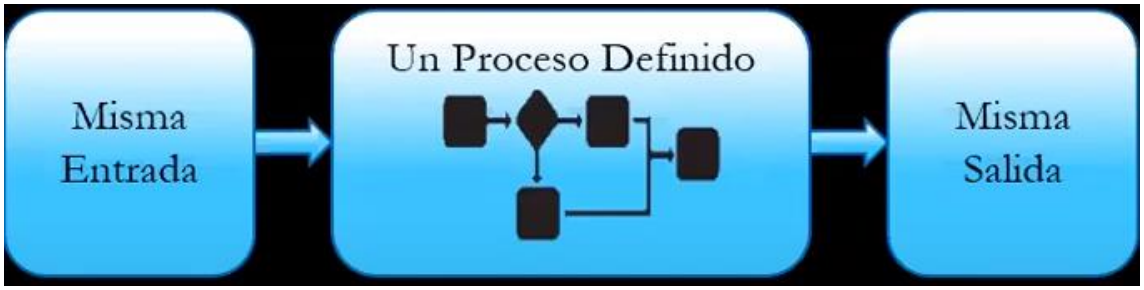
- **Proceso:** es una secuencia de pasos ejecutados para un propósito determinado.
- **Proceso de Software:** es un conjunto de actividades que se ejecutan utilizando personas (personas con habilidades, entrenadas y motivadas es una característica fundamental en el proceso de desarrollo de SW) y dichas personas teniendo distintas herramientas ejecutan actividades para producir un producto que sería el SW.



Procesos Definidos:

Están basados en las líneas de producción, donde si repetimos el mismo proceso indefinidamente, los resultados obtenidos son los mismos. Están basados en la administración y el control donde se reduce la incertidumbre.

Es decir que con una misma entrada, ejecutando siempre la misma serie de actividades, voy a obtener la misma salida.

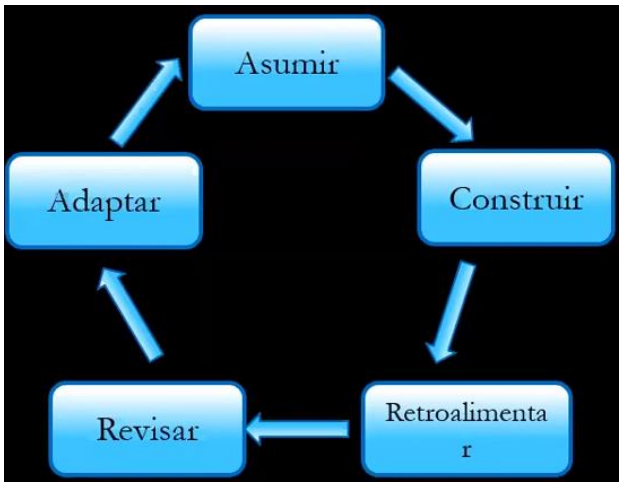


Procesos Empíricos:

Los procesos empíricos apuntan a que cuando los procesos son muy complicados, o estamos en una situación donde hay muchas variables cambiantes, por mas que se repita el proceso siempre de la misma manera, el resultado puede llegar a ser diferente. Por ende, bajo este contexto un proceso definido no podría aplicar, porque el resultado no podría ser el mismo.

La administración y el control en dichos procesos es a través de inspecciones frecuentes y adaptaciones.

En los procesos empíricos, el patrón tiene que ver con el aprendizaje, es decir con una retroalimentación de lo que va pasando, y a partir de ahí poder hacer adaptaciones de lo que vamos haciendo en términos de aprendizaje, para poder construir a través de eso que fuimos aprendiendo con la retroalimentación. Se va logrando una adaptación a medida que voy ejecutando ese proceso para ir aprendiendo de lo que paso en la ejecución anterior.

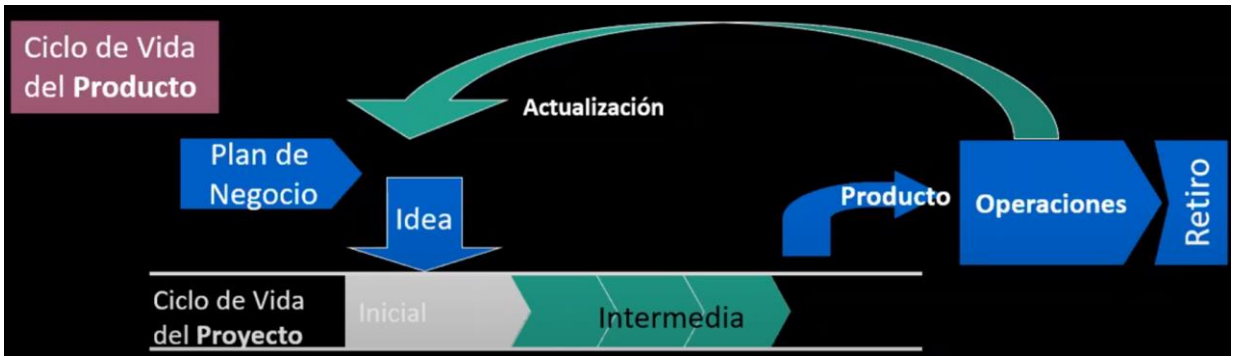


Ciclos de Vida:

Es la serie de pasos que hace que el producto o el proyecto progrese. Tanto los productos como los proyectos tienen ciclo de vida.

¿Que sobrevive primero, el ciclo de vida del proyecto o del producto?

La del producto, ya que el proyecto termina pero el producto sigue a lo largo del tiempo, y para un mismo producto se pueden tener *n* proyectos.



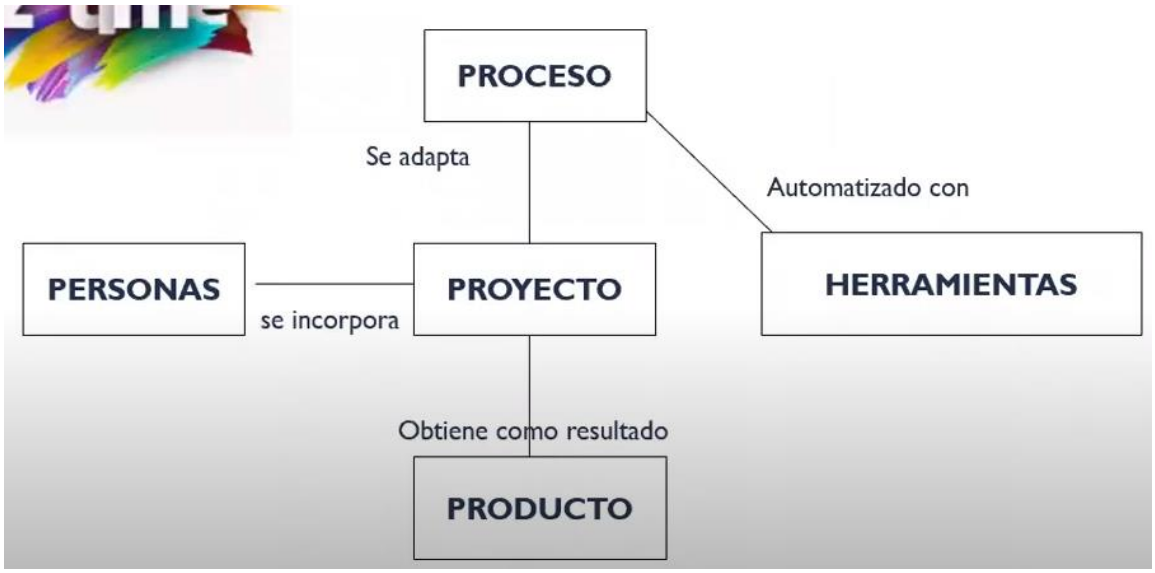
Ciclo de vida de Proyectos de Software:

El ciclo de vida de un proyecto de SW es una representación de un proceso. Grafica una descripción del proceso desde una perspectiva particular.

Los modelos de ciclo de vida especifican:

- Las fases del proceso y en que orden o de que manera esas fases se van a llevar a cabo.
    - Ejemplos de ciclo de vida: iterativo e incremental, en cascada (secuencial), recursivo, etc..

Las 4 "P":



El **proceso** esta definido, y puede haber tantos **proyectos** basados en ese proceso como se quiera. La diferencia entre proyecto y proceso es que el proyecto es una instancia del proceso y se hace una adaptación para una ejecución especifica.

Las **personas** son las que participan en el proyecto

El objetivo del proyecto es la obtención de un **producto**, donde la vida del producto se mantiene a lo largo del tiempo, y en cambio la vida del proyecto puede terminar antes que la vida del producto, donde pudo haber varios ciclos de vida de proyectos dentro de un mismo ciclo de vida de un producto.

El proceso esta automatizado con **herramientas**, como por ejemplo un compilador para que no se tenga que compilar código uno mismo, un entorno de desarrollo o herramientas de gestión de configuración, etc.

Proyecto:

Es un conjunto de personas que, con sus herramientas, realizan diferentes actividades para alcanzar un objetivo común.

Se deben tener objetivos claros, pueden ser uno o varios, pero el proyecto tiene que estar ***dirigido a obtener resultados***, y esos resultados se reflejan a través de esos objetivos, lo cual tienen que ser medibles.

- Los proyectos están dirigidos a obtener resultados y ello se refleja a través de objetivos.
- Los objetivos guían al proyecto.
- Los objetivos no deben ser ambiguos
- Un objetivo claro no alcanza… debe ser también alcanzable.

Los proyectos ***siempre son temporarios***, es decir que empiezan y terminan, una vez que se alcanza el objetivo el proyecto termina. Si no tiene principio y fin no es un proyecto. Por ejemplo: una línea de producción o un sistema en operación no es un proyecto, ya que no tienen principio ni fin.

*Entonces, las dos **características clave** de los proyectos son:*

I. Son orientados a objetivos, teniendo objetivos claros y alcanzables y que a su vez sean medibles, es decir, poder medir si se alcanzaron o no.

II. Son de duración limitada o temporarios, tienen un principio y un fin.

La ***tercer característica*** es que, debido a la **complejidad** y a las distintas variables que se presentan en las tareas a realizar en los proyectos, hace que un proyecto tenga características distintas de los demás proyectos, por más que en términos de objetivos y del producto a obtener sean similares.

La ***cuarta*** y última ***característica*** es que los proyectos **son únicos**, todos tienen características que los hace únicos, no puede repetirse.

Administración de Proyectos:

Es trabajar para que podamos cumplir con los objetivos que se plantearon, pudiendo medir el cumplimiento de dichos objetivos, en el contexto de el presupuesto que tenemos acordado y en el contexto de los requerimientos que tenemos que cumplir.

Administrar un proyecto implica en un principio identificar los requerimientos, y a partir de los mismos establecer cuales son los objetivos concretos, alcanzables y medibles que vamos a perseguir con la ejecución de ese proyecto.

Una vez que se identifican los requerimientos, y se definen los objetivos que queremos alcanzar y como los vamos a medir, el paso siguiente es trabajar en un plan de proyecto y en una gestión de ese plan de proyecto para asegurarnos de que a medida que el tiempo transcurre podemos cumplir con la meta que nos fijamos.

- Mas académicamente… administración de proyectos es la aplicación de conocimientos, habilidades, herramientas y técnicas a las actividades del proyecto para satisfacer los requerimientos del proyecto.

- Administrar un proyecto incluye:
  - Identificar los requerimientos
  - Establecer objetivos claros y alcanzables
  - Adaptar las especificaciones, planes y el enfoque a los diferentes intereses de los involucrados (stakeholders).

La triple restricción (The Triple Constrain):

Cuando tenemos que gestionar un proyecto, tenemos que balancear 3 aspectos que de alguna manera u otra se comprometen entre sí.

Los 3 aspectos son:

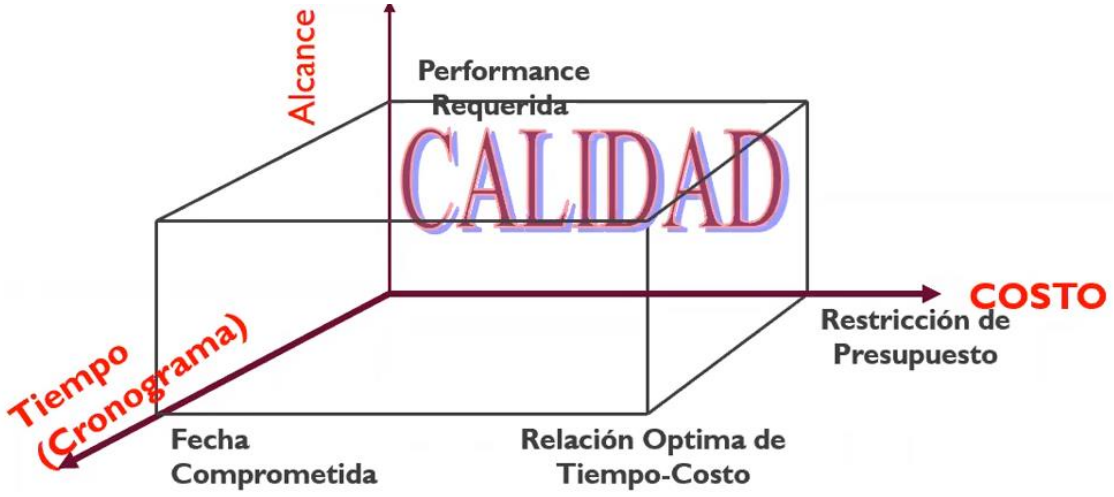
- **Objetivos del proyecto:** cuales son los objetivos del proyecto (o alcance)? Que esta tratando de alcanzar el proyecto?
- **Tiempo:** cuanto tiempo debería llevar completarlo?
- **Costo:** cuanta plata me va a salir llevar a cabo el proyecto?

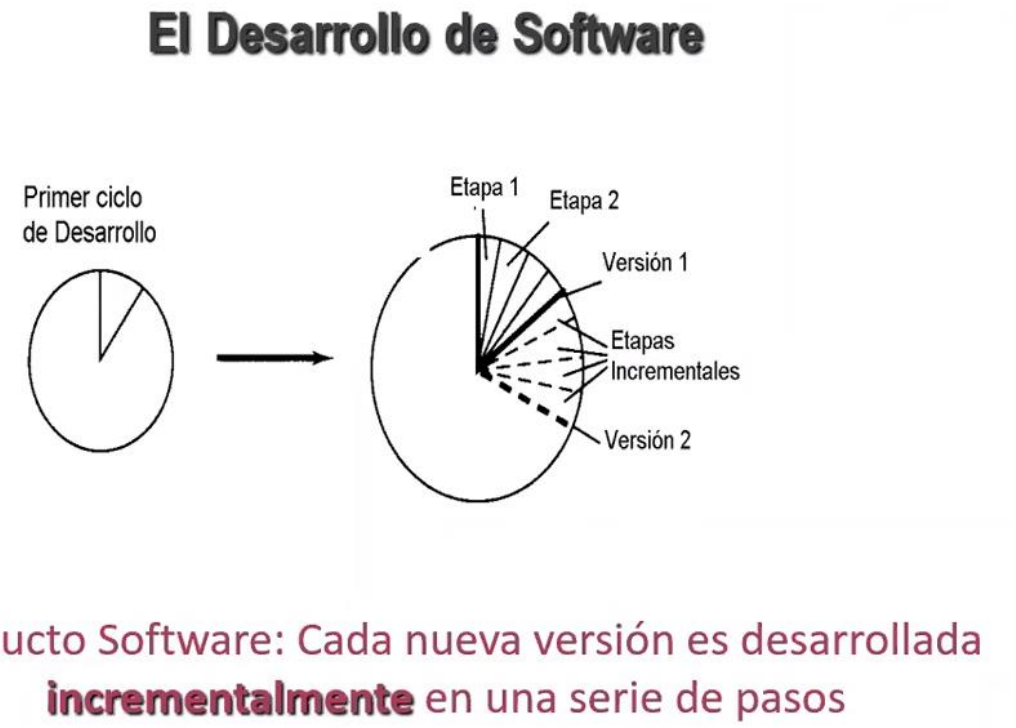
Se llama triple restricción porque no podemos tocar una de las 3 variables sin comprometer lo que ocurre con las otras 2. Frente al cambio de una de las variables probablemente tenga que acomodar las otras 2 variables restantes para poder cumplir con ese cambio.

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

  "proyectos de alta calidad entregan el producto requerido, el servicio o resultado, satisfaciendo los objetivos en el tiempo estipulado y con el presupuesto planificado."

Es responsabilidad del ***líder de proyecto*** balancear estos tres objetivos.

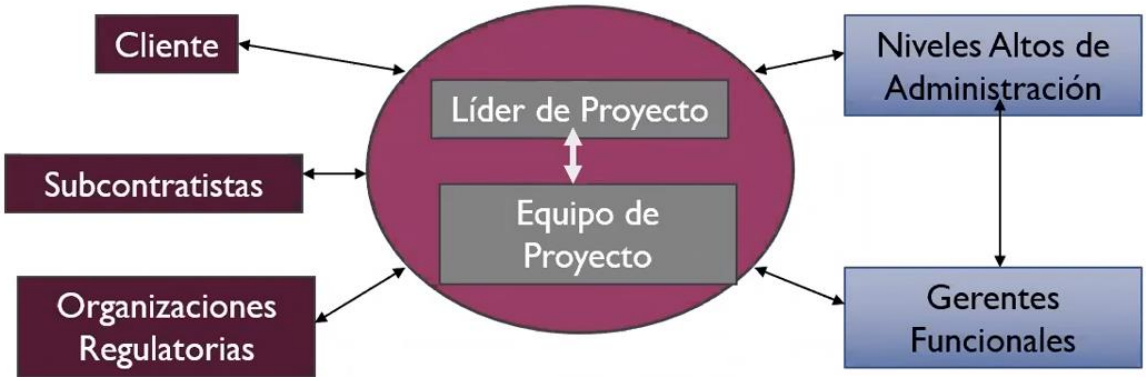




Cada una de esas etapas del producto de SW puede ser un proyecto distinto.

Rol del Líder de Proyecto / Equipo:

Es el mediador o el interlocutor o el que guía al equipo de proyecto para que cumpla con sus objetivos y para poder lograr la relación con los roles que están por fuera del equipo de proyecto. Por fuera están los niveles gerenciales, los proveedores externos, los entes reguladores, y nuestro cliente.



Equipo de Proyecto:

Es un grupo de personas que están comprometidas a alcanzar un objetivo común.

- Normalmente son grupos pequeños, para poder gestionarlos mas fácilmente.
- Las personas que lo conforman poseen diversos conocimientos y habilidades.
- Que sea un grupo pequeño posibilita que puedan trabajar juntos coordinada y efectivamente, y permite desarrollar sinergia.
- Tienen sentido de responsabilidad como una unidad.

Plan de Proyecto:

Sirve para guiar al equipo y como hacer para lograr los objetivos que se definieron inicialmente. *"Sería como la hoja de ruta de un viaje"*, en este caso seria todo lo que se tiene que hacer para que el proyecto se ejecute guiado por la triple restricción para cumplir con los objetivos pautados.

El plan de proyecto documenta:

- ¿Qué es lo que hacemos?
- ¿Cuándo lo hacemos?
- ¿Cómo lo hacemos?
- ¿Quién lo va a hacer?

¿Qué implica la planificación de proyectos de software?

- **Definición del Alcance del Proyecto:**

No es lo mismo el alcance del producto que del proyecto, porque el alcance del proyecto es definir que trabajo vamos a hacer para entregar el producto o servicio con las características y funciones especificadas, es decir con el alcance del producto que especifiquemos que vamos a cubrir en el contexto del proyecto.

Entonces:

Alcance del Producto:

Son todas las características que pueden incluirse en un producto o servicio.

El cumplimiento del alcance del producto ***se mide*** contra la *"Especificación de Requerimientos"*.

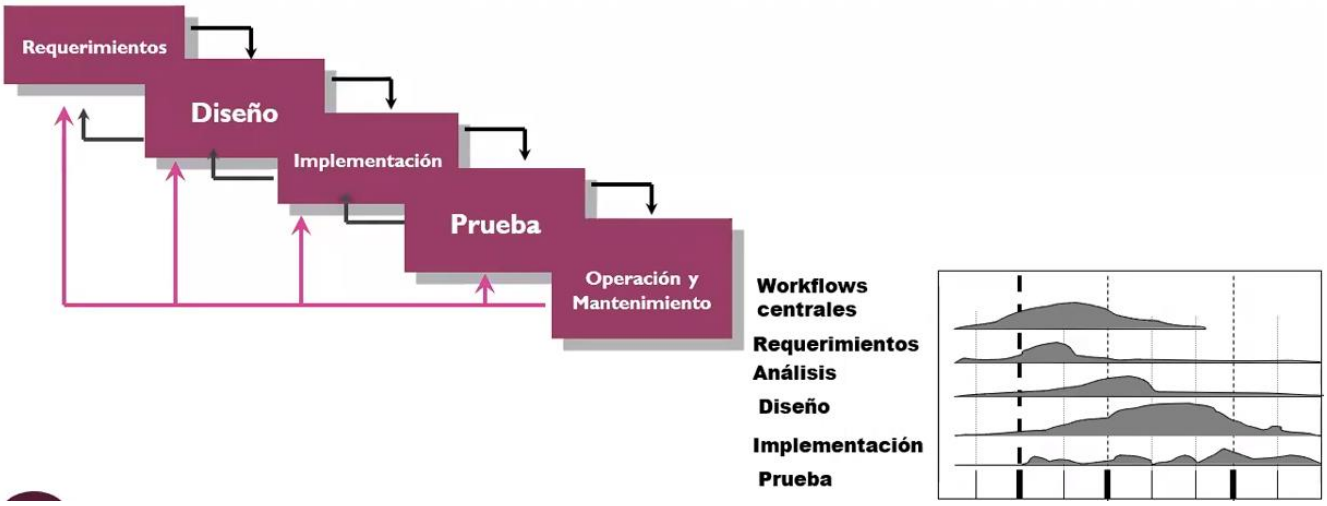
Alcance del Proyecto:

Es **todo el trabajo** y **solo el trabajo** que debe hacerse para entregar el producto o servicio con todas las características y funciones especificadas.

El cumplimiento del alcance del proyecto ***se mide*** contra el *"Plan de Proyecto"*, que nos va a indicar que es lo que tenemos que hacer y cuál es el objetivo por alcanzar.

- **Definición de Proceso y Ciclo de Vida:**

Se define un ciclo de vida para el proyecto, como por ejemplo el del Proceso Unificado de Desarrollo.



- **Estimación:**

Las estimaciones de SW tienen varios componentes, y se hacen dichas estimaciones para armar el Plan de Proyecto, se hacen en un determinado orden debido a que es más fácil lograr hacer dichas estimaciones:

I. ***Tamaño:*** del producto de SW a producir, y se pueden utilizar diferentes unidades de medida. (ej. Cantidad de funcionalidad a implementar, etc.).

II. ***Esfuerzo:*** se mide en horas hombre lineales, y seria cuanto le va a tomar al equipo hacer el software estimado previamente.

III. ***Calendario:*** se llevan las horas lineales del esfuerzo a un calendario, dependiendo de como se secuencian o paralelizan las actividades, la jornada laboral de cada persona, la cantidad de personas disponibles, las vacaciones, etc. Por lo que finalmente se determina una fecha de inicio y una fecha de fin.

IV. ***Costos:*** se estiman costos por persona, en base a los gastos fijos, gastos en licencias, costos administrativos, todo en base al calendario.

V. ***Recursos Críticos:*** tiene que ver con aquellas cuestiones que, a lo largo de la vida del proyecto, pueden generar algún problema, como pueden ser personas, momentos del tiempo, etc.

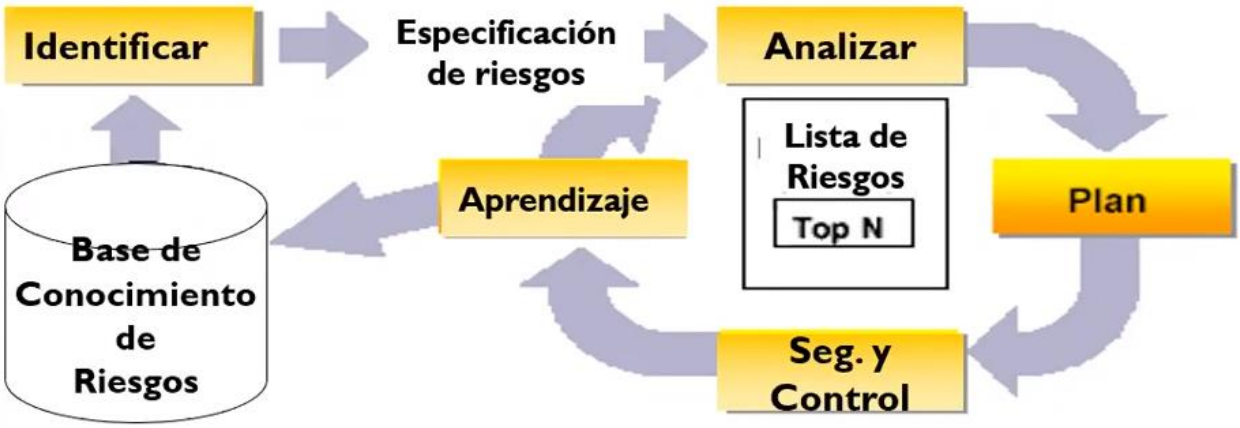
- **Gestión de Riesgos:**

Los riesgos son cosas que podrían ocurrir y que pueden comprometer el éxito del proyecto. Son problemas esperando para suceder. Para tratar de minimizar el impacto de los riesgos, se los gestiona.

Entonces lo que se hace es identificarlos, lo cual no van a ser siempre los mismos porque pueden ir cambiando, luego se los analiza, se los lista y se saca un Top 10 (o Top 5, dependiendo de la envergadura del proyecto) por así decir, ya que los riesgos pueden ser muchos.

Para cada riesgo se determina el *impacto* y la *probabilidad de ocurrencia*. Una vez determinados ambos, se les asigna una numeración y se los multiplica obteniendo lo que se denomina ***exposición al riesgo***, que es lo que permite hacer el Top de la lista de riesgos. Se eligen unos cuantos y se arma un plan para mitigar los riesgos seleccionados para minimizar su impacto y su probabilidad de ocurrencia.

Por ultimo se hace un seguimiento y control de los riesgos para ver que va pasando con los riesgos, lo cual reduce la *probabilidad de ocurrencia*.



Los planes de contingencia son los que tratan de reducir el *impacto* de los riesgos cuando ocurren.

- Asignación de Recursos
- Programación de Proyectos
- Definición de Controles
- **Definición de Métricas:**

Las métricas sirven para saber que tan bien se viene ejecutando el proyecto en función de lo que tiene definido el plan de proyecto.

Las métricas se dividen en tres dominios:

1. ***Métricas de Proceso:*** ej. Un Porcentaje de Desvió de los proyectos, que se realiza en base a otros proyectos realizados en el pasado.

2. ***Métricas de Proyecto:*** ej. La diferencia entre el avance real del proyecto con el avance estimado en el calendario.

3. ***Métricas de Producto:*** ej. Densidad de defectos del producto, medidas de calidad del producto, etc.

Las métricas básicas para un proyecto de SW son:

- ***Tamaño del Producto***
- ***Esfuerzo***
- ***Tiempo (calendario)***
- ***Defectos (normalmente tienen que ver con el producto)***

Factores para el éxito del Proyecto:

- Monitoreo & Feedback
- Tener una misión u objetivo claro
- Comunicación (importante)

Factores para el fracaso del Proyecto:

- 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

Métodos de Desarrollo Ágil:

Objetivo del enfoque Ágil:

El objetivo primordial es ***construir software de forma rápida*** para aprovechar las actuales oportunidades y responder ante la amenaza competitiva, convirtiendo la velocidad de entrega en el requerimiento fundamental de los sistemas de software: "software de entregas rápidas en un entorno cambiante".

Los desarrollos ágiles se utilizan para entornos con gran variabilidad de requerimientos, ya que los clientes encuentran imposible predecir como un sistema afectará sus prácticas operacionales o que cambios habrá en el entorno (mercado o políticas de negocio) que pueden dejar el sistema completamente obsoleto.

Los procesos de desarrollo de software rápido se diseñan para producir rápidamente un software útil, el cual no se desarrolla como una sola unidad, sino como una serie de incrementos, y cada uno de ellos incluye una nueva funcionalidad del sistema.

- Por lo general se crean nuevas versiones (incrementos) del sistema cada 2 o 3 semanas y se pone a disposición del cliente.

- Involucran a los clientes en el proceso de desarrollo para conseguir una rápida retroalimentación sobre los requerimientos cambiantes y minimizan la cantidad de documentación con el uso de comunicaciones informales en lugar de reuniones formales con documentos escritos.

- **Fowler:** “Un compromiso útil entre nada de proceso y demasiado proceso”

Valores del manifiesto Ágil:

- ***Individuos e interacciones*** *por sobre* ***procesos y herramientas****:*

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

Me va a sumar mucho más en calidad al producto la elección de las personas, como van a interactuar y cómo se encuentran (si motivadas o no), que las herramientas que utilice para realizarlo. La mejor forma de interactuar es cara a cara.

- ***Software funcionando*** *por sobre* ***documentación detallada****:*

Las metodologías ágiles entregan software funcionando todo el tiempo. Se documenta todo aquello que agregue valor al producto y este centrado en mi cliente (valor agregado). Si la documentación lo hace, tiene que ir creciendo en cada iteración.

La medida del progreso es cuanto SW tengo funcionando. No me sirve una ERS super completa sin SW. (Necesito algo de documentación, pero no tanto como en las metodologías tradicionales)

Es más importante ir construyendo el software e ir aprendiendo sobre la marcha. Es más barato que estar meses escribiendo requerimientos sobre papel.

- ***Colaboración*** *por sobre* ***negociación con el cliente****:*

Hay que estar dispuesto al cambio y cerca del cliente para predecirlo. Hay ciertos requerimientos que surgen de la colaboración con el cliente, donde van apareciendo alternativas que generan cambios en el producto. La relación con el cliente puede o no ser 1 a 1 (Ej. en Lyn thinking no).

Los dos queremos el éxito, tanto el cliente como quienes lo tenemos que hacer. Si no nos entendemos es un problema de los 2. No necesitamos tener pruebas de lo que dijimos, porque queremos lo mismo. Sin embargo en algún lado tiene que quedar plasmado lo mínimo de los requerimientos, no nos vamos a acordar todo.

Si logramos que el cliente sea parte del equipo, la conversación cambia rotundamente. Si logramos conversar bien, a los dos nos va a salir más barato. El cliente va a gastar menos y el desarrollador también porque no va a tener que cambiar mucho.

- ***Responder a cambios*** *por sobre* ***seguir un plan:***

Los nuevos requerimientos pueden tener un mayor valor que los iniciales. Se reciben cambios de requerimientos, y estos son vistos en su mayoría de buena forma por el equipo.

La idea es estar preparado para los cambios. No hay que estar siguiendo un plan ESTRICTO. Sino un plan que se puede ir cambiando, flexibilizando, adaptando. Teniendo en cuenta errores. Un error conceptual es pensar que no va a haber más documentación! o pensar que no va a haber más planning. Mentira, si va a haber! Si planificamos todo, justamente todo el tiempo, continuamente. Todos los días.

Los cambios son bienvenidos. Debemos tener en claro que somos capaces de hacer, si incluimos una nueva funcionalidad posiblemente podemos sacar otra (Tenemos que ir negociando para llegar a tiempo con el proyecto funcionando) (Hay un plan de los que se desarrolla que es un plan dinámico, vivo)

Los 12 Principios del Manifiesto:

1. ***La prioridad es satisfacer al cliente a través de releases tempranos y frecuentes:***

Nuestra mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software con valor es decir de un producto funcional

“Que el producto sea lo que realmente el cliente necesita” (Muchas veces el cliente no sabe bien lo que quiere, así que tenemos que ayudarlo a descubrir lo que quiere, cuál es su necesidad)

Fundamental ENTENDER lo que el cliente quiere, pide, necesita.

2. ***Recibir cambios de requerimientos, aun en etapas finales:***

Los procesos ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente

El cambio es bienvenido. No seguir el plan a rajatabla. El cliente puede ir dando un Feedback que implica que se puede modificar algo.

3. ***Releases frecuentes (2 semanas a un mes):***

Entrega de producto frecuentemente, se entrega SW funcionando cada dos o tres semanas, con preferencia al periodo de tiempo más corto posible.

Hagamos entregas chiquititas pero seguido. Entonces le vamos a mostrar al cliente avances de verdad.

4. ***Técnicos y no técnicos trabajando juntos TODO el proyecto***

Colaboración diaria, es decir que los responsables del negocio y los desarrolladores trabajan juntos en forma cotidiana durante todo el proyecto.

Clientes y el equipo de desarrollo. Colaboremos entre nosotros porque todos queremos que salga con éxitos porque así a TODOS nos va a salir más barato.

5. ***Hacer proyectos con individuos motivados***

Colaboradores motivados, los proyectos se desarrollan en torno a individuos motivados, hay que darles el entorno y el apoyo que necesitan y además confiarle la ejecución del trabajo. Tener personas motivadas agrega mucha más calidad al producto.

Si tenemos un equipo de gente motivada con excelencia técnica. no importa que modelo estemos usando seguro el proyecto va a ser un éxito. Los líderes de los equipos tienen que asegurar que el equipo trabaje de manera cómoda y trabaje feliz. Al equipo también le hace bien ver el sw funcionando.

Parte de la motivación del equipo es ver el SW funcionando.

6. ***El medio de comunicación por excelencia es cara a cara***

Esto quiere decir que el método más eficiente y efectivo de comunicar Info entre el equipo de desarrollo y los clientes es la conversación cara a cara. Siempre conviene equipos “Colocados” que estén en un lugar físico cercano, que se puedan ver y cuando tengan que discutir algo, se hace más fácil. La pandemia complicó un poco esto.

7. ***La mejor métrica de progreso es la cantidad de software funcionando***

Medición del avance por trabajo completado, esto quiere decir que el SW funcionando es la medida principal de progreso

¿Cuántas features tengo?

Hay un montón de métricas diferentes

8. ***El ritmo de desarrollo es sostenible en el tiempo***

Promover un ritmo sostenido, esto quiere decir que los procesos ágiles promueven el desarrollo sostenible y que los promotores, desarrolladores y usuarios deben ser capaces de mantener un ritmo constante de forma indefinida.

La idea es que se mantenga el ritmo de construcción de desarrollo tiene que ser continuo. Tiene que ver que ya no estoy hablando meses de construcción de documentación. Sino que estoy construyendo software TODO el tiempo, y ahí entran las métricas de velocidad.

9. ***Atención continua a la excelencia técnica***

Atención a la excelencia, la atención continua a la excelencia técnica y al buen diseño, mejora la agilidad

Constantemente, SIEMPRE, se tiene que tener en cuenta.

SW mantenible, performable, con los requerimientos funcionales claros, desde el principio hay que generar SW de calidad.

10. ***Simplicidad - Maximización del trabajo no hecho***

La simplicidad es esencial, maximizar el trabajo no hecho es decir no hago cosas de más innecesarias que el cliente no me pidió. No hago ningún tipo de actividad innecesaria, que no me piden

Ej: relevar requerimientos que voy a tener que hacer de acá a tres meses.

11. ***Las mejores arquitecturas, diseños y requerimientos emergen de equipos auto organizados***

La autoorganización del equipo, la posibilidad de que el equipo tome decisiones, y defina su propio proceso empírico. Definir cuales tareas tenemos sabiendo en que somos mejores cada uno del equipo. El cliente es como si fuera parte del equipo.

12. ***A intervalos regulares, el equipo evalúa su desempeño y ajusta la manera de trabajar***

A intervalos regulares hay que hacer una evaluación de lo que vengo haciendo, evaluar y ajustar lo que estoy haciendo (Procesos de mejora continua, Ej; retrospectiva: el equipo propone mejoras, y elige cuales son las más importantes)

¿Qué es Ágil?:

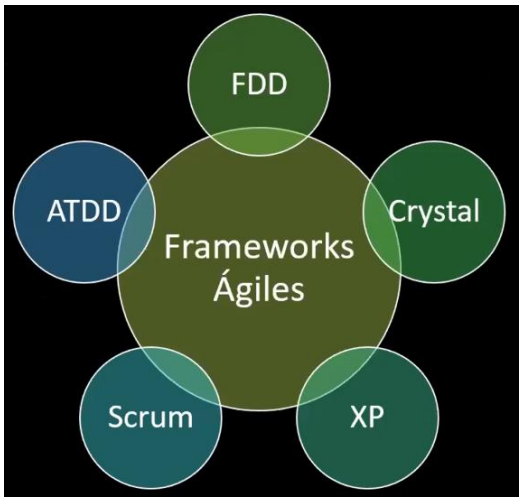
No es una metodología ni un proceso, es una ideología, filosofía o cultura que define un conjunto de principios que guían el desarrollo del producto.

Valores de equipos agiles:

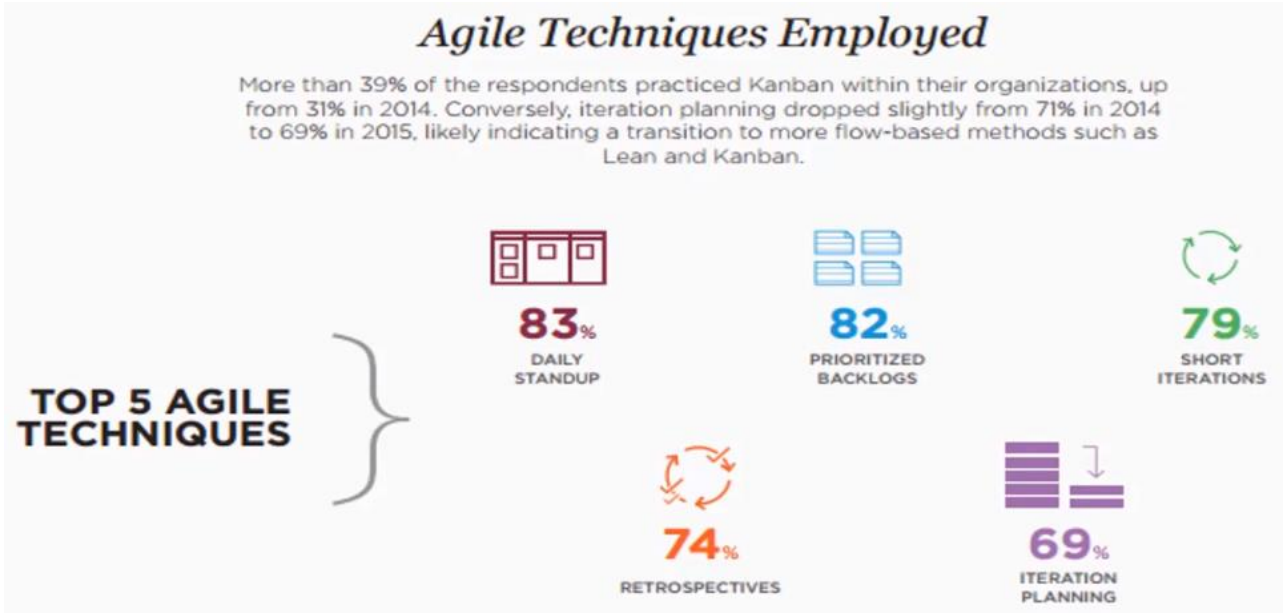
- Planificación continua, multinivel
- Facultados, autoorganizados, equipos comprometidos
- Entregas frecuentes, iterativas y priorizadas
- Prácticas de ingeniería disciplinadas
- Integración continua
- Testing Concurrente

Entonces, el agilismo es encontrar un equilibrio el tener nada de proceso y demasiado proceso, demasiada burocracia y demasiada documentación. El equipo se adapta al contexto, a los productos que tienen que construir y en función de eso definen los procesos mas apropiados (la adaptabilidad es clave). Otro concepto clave es el foco en las personas, es decir, individuos con excelencia técnica, capacitados, motivados, con capacidad de autoorganización. Las personas son el factor fundamental para obtener el éxito.

Algunos métodos agiles:

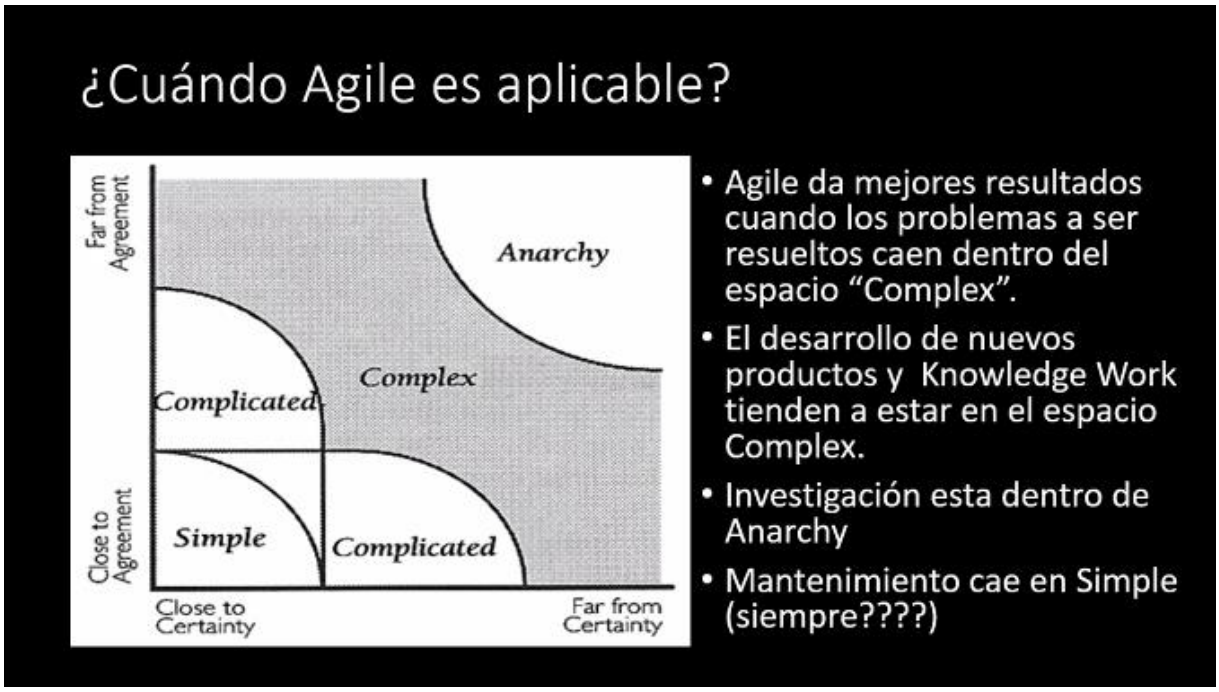


Técnicas Efectivas:



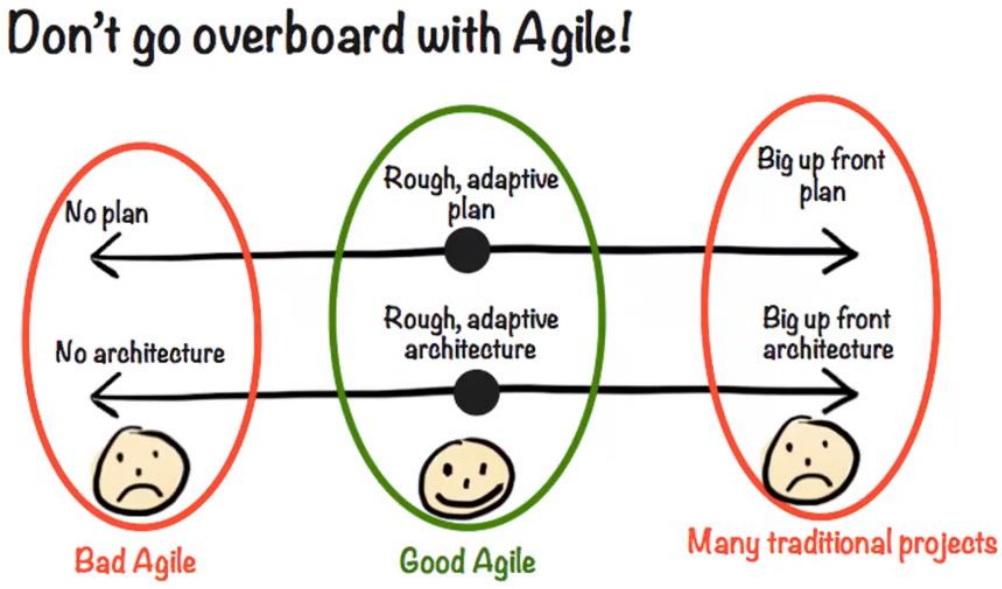
- **Daily Standup (Reuniones Diarias):** o reuniones de sincronización, son reuniones de 15 minutos donde el equipo conoce que es lo que esta haciendo, el resto que es lo que ya hizo y que es lo que está por hacer hoy, si hubo algún problema, etc. Sobre todo para sincronizarse y no pisar el trabajo y de esa manera perder tiempo y todos se deben poner de acuerdo.
- **Priorizar Requerimientos:** ordenar que es lo que hace falta primero y que hace falta después.
- **Iteraciones cortas y regulares:** que siempre duren lo mismo las iteraciones.
- **Planificación de Sprint:** planificar las iteraciones cortas
- **Reuniones de Retrospectiva:** ciclos de mejora para evaluar el trabajo del equipo, proponer acciones de mejora, aplicarlas, etc.

¿Cuándo es aplicable Agile?

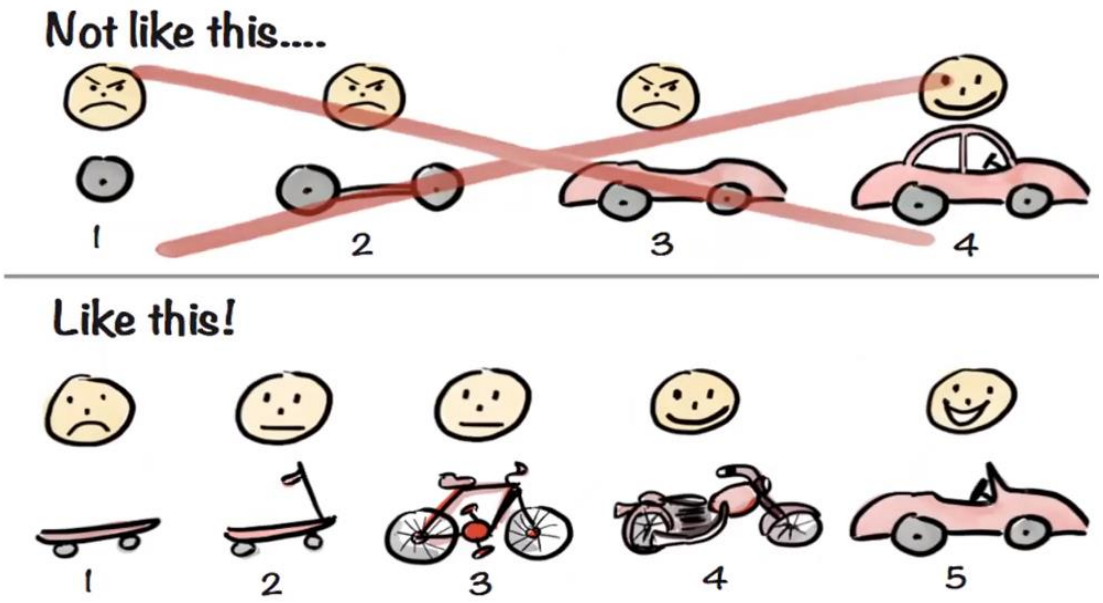


En la parte sombreada. Puedo hacer una investigación para averiguar qué es lo que quiere el cliente y lograr entrar en la zona gris. En los procesos de mantenimiento conviene usar agiles pero dependiendo, por ahí puedo usar otra metodología que no sea Scrum.

Ser ágil no es ser indisciplinado:



No es que ser ágil significa no tener documentación ni planes, justamente en lo que planificación es mucho más riguroso, en lo que es que el plan es adaptativo también. Ser ágil no es ser indisciplinado

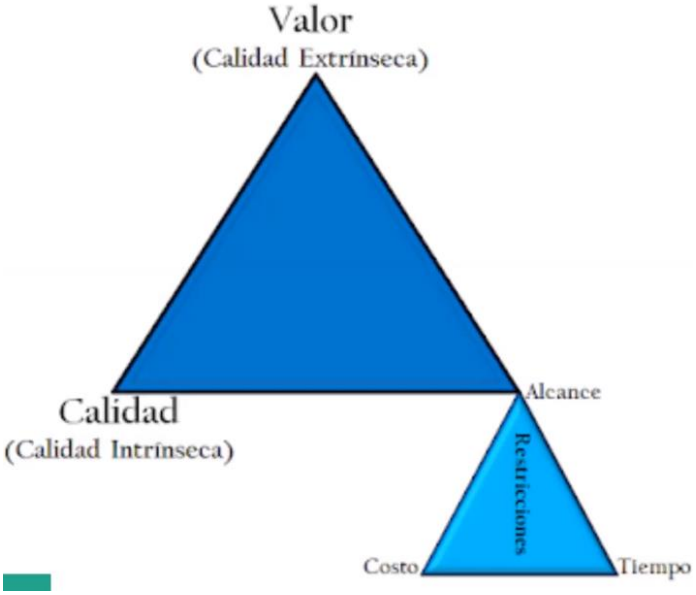


Cómo hacemos para ir mostrando SW funcionando??? Como la versión de la bici, ***ir haciendo pequeñas cosas que sean funcionales***.

Triangulo de Triple Constrain:

Tres variables que tiene todo proyecto, donde se crucen esas tres variables será la calidad de mi producto, Estas variables están en cambio, y hay que tratar de balancearlas.

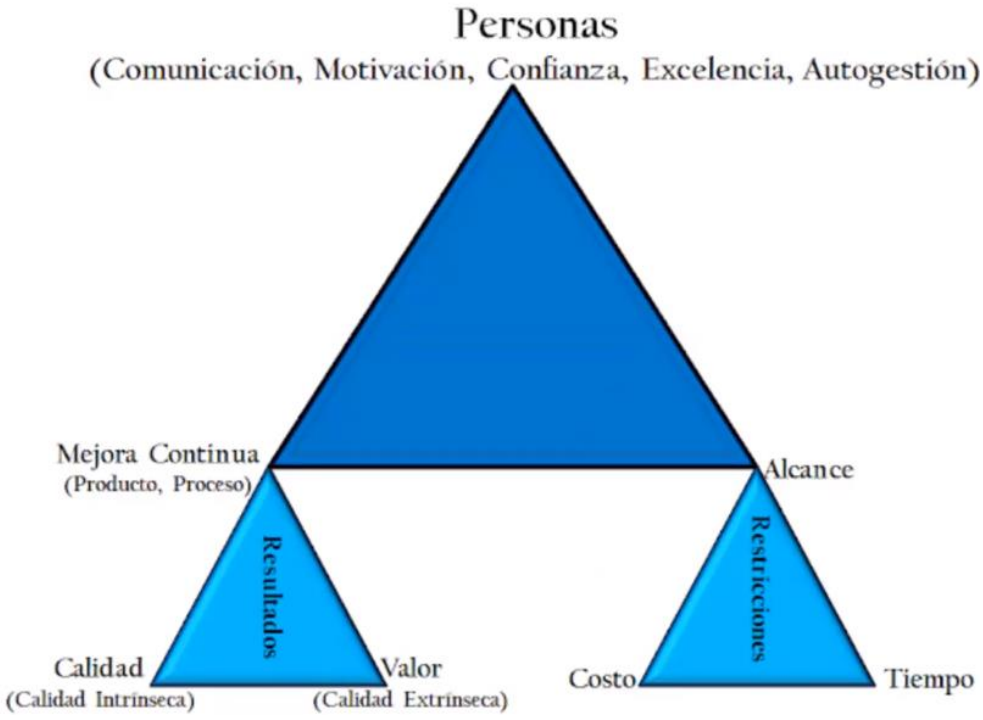
El triángulo Ágil:



El triángulo se modificó donde se tiene en cuenta la calidad intrínseca, que tiene que ver con la excelencia técnica, es decir que el software haga lo que tiene que hacer y funcione en la calidad que está por debajo que el cliente no ve.

Y la calidad extrínseca es cuál es el valor del negocio que le estoy dando al cliente. No es lo que dice que quiere, sino lo que necesita el cliente.

Triangulo Ágil Modificado:



Requerimientos Agiles:

<u>Requerimientos Agiles vs User Stories:</u>

Requerimientos ágiles tiene que ver con la forma de plantear lo que esta relacionado con la captura de requerimientos del SW en el contexto de lo que se deriva del manifiesto Ágil.

Las ***user story*** es una técnica para trabajar con requerimientos agiles, ***una técnica***, no es la única. En cambio los requerimientos agiles son la manera de trabajar con requerimientos en el contexto de los principios y valores que están planteados en el manifiesto ágil, y como hacemos para darle forma a los requerimientos alineándolos a lo que plantea el manifiesto.

NO SON SINONIMOS.

Dentro de los 12 principios del manifiesto ágil, los que se relacionan directamente con los requerimientos son:

- **“Cambios de requerimientos son bienvenidos”**
- **“Satisfacer al cliente con entregas frecuentes y tempranas”**
- **“Simplicidad”** (no directamente pero está relacionado)
- **“Arquitecturas, diseños y requerimientos emergentes”**
- **“Medio de comunicación: cara a cara”:** *por excelencia es uno de los que mas afectan o mas diferencian la forma en la que se gestionan los requerimientos.*
- **“Técnicos y no Técnicos trabajando juntos todo el proyecto”**

Premisas Básicas de los Requerimientos dentro de los Framework Agiles:

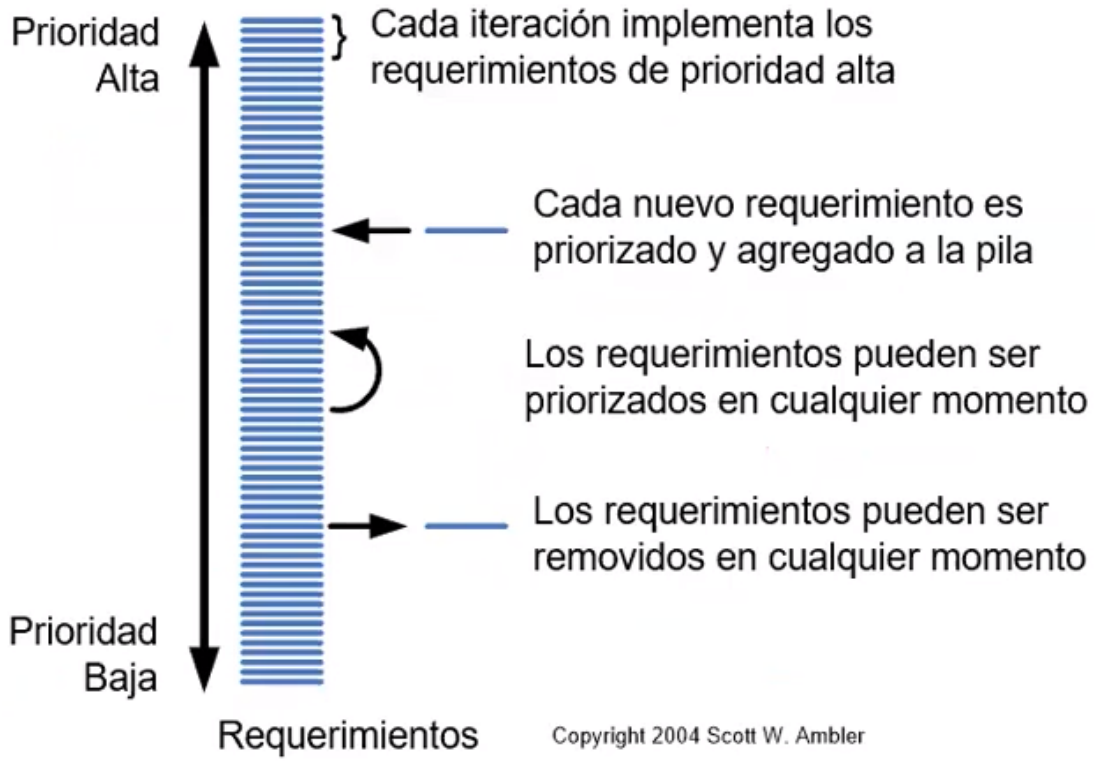
- Usar el “valor” para construir el producto correcto, es decir, a partir de lo que le de *valor* (en términos cuantitativos) al negocio. *Construir un producto que sea rentable para el negocio.*
- Usar historias y modelos para mostrar que construir.
- Solamente empezar a trabajar en el producto en lo que sea estrictamente necesario para darle al producto el valor que estamos persiguiendo, *determinar que es “solo lo suficiente”*. Esta fuertemente relacionado con el tipo de ciclo de vida que vamos a trabajar con este tipo de procesos empíricos.

  Esto pasa porque la mayoría de los productos, inclusive los *“exitosos”* tienen funcionalidad que prácticamente no se utiliza nunca o raramente

se utiliza, y aquella funcionalidad que de verdad se utiliza y es importante representa solo un 20% de la funcionalidad total. Por eso debemos enfocarnos en solo aquella parte que sea fundamental para el negocio y que le de valor.

Gestión Ágil de Requerimientos de Software:

Los requisitos cambiantes son una ventaja competitiva si puede actuar sobre ellos.



Lo primero que aparece es el *backlog* del producto, que va a contener los features del producto, donde en este contenedor se tendrían listados todos aquellos requerimientos que debería tener nuestro producto, ordenados según su prioridad (el que determina dicha prioridad es el ***product owner***, porque se prioriza en base a lo que de valor al negocio, y el que sabe eso es el *product owner*). Recordando de que el cliente (product owner) es como si fuera uno más del equipo.

Los *requerimientos que están al principio* de la pila son los que se va a trabajar con mayor nivel de detalle y a medida que se va yendo al fondo de la pila puede aparecer algún requerimiento con mayor granularidad que no están descriptos tan detalladamente porque no se va a perder tiempo describiendo requerimientos que hasta podrían ser removidos.

Normalmente los requerimientos que aparecen en el *tope de la pila* tienen característica de ***user story***, y los que están mas debajo son una *Épica*, o un *tema* (ya que no tienen la suficiente granularidad o detalle).

Just in Time:

Minimizar los costos de logística de tal manera de tener disponible el producto o materia prima en *el momento exacto en el que lo necesito*. Con respecto a los Requerimientos Agiles, vamos a analizar los requerimientos en el momento que haya que hacerlo, es decir, cuando tenga la prioridad suficiente para que sea un tema para que nosotros lo vayamos a tratar, ***no antes***. Si lo hacemos antes corremos el riesgo de trabajar en algo que va a cambiar, o que no tiene el suficiente valor para el negocio.

Cuadro Comparativo de "Tradicional" vs "Ágil":

	Tradicional	**Ágil**
Prioridad	Cumplir el plan	Entregar valor
Enfoque	Ejecución ¿Cómo?	Estrategia ¿Por qué? ¿Para qué?
Definición	Detallados y cerrados. Descubrimientos al inicio.	Esbozados y Evolutivos -
Participación	Sponsor, stakeholder de mayor poder e interés	Colaborativo con stakeholders (clientes, usuarios finales)
Equipo	Analista de Negocios, Project Manager y Areas de Proceso	Equipo multidisciplinario
Herramientas	Entrevistas, observación y formularios	Principalmente prototipado. Técnicas de facilitación para descubrir.
Documentación	Alto nivel de detalle – Matriz de Rastreabilidad para los Requerimientos	Historias de Usuario Mapeo de Historias (Story Mapping)
Productos	Definidos en Alcance	Identificados progresivamente
Procesos	Estables, adversos al cambio	Incertidumbre, abierto al cambio.

Tipos de Requerimientos:

- **Requerimiento de Negocio:** es un requerimiento en términos de objetivos estratégicos, es decir, por ejemplo, *"disminuir un X% de tiempo invertido en procesos manuales relacionados a la atención al cliente"*.

- **Requerimientos de Usuario:** imaginar como se puede resolver el problema planteado en el requerimiento de negocio, por ejemplo, *"automatizar las consultas que realizan los clientes"*. Es una feature escrita de manera no técnica de cara al cliente, esta planteado de manera de lo que el *usuario quiere hacer*, y no lo que el sistema debería hacer.

- **Requerimiento Funcional:** concretamente que es lo que se implementaría para resolver lo planteado en los requerimientos de usuario, por ejemplo, *"implementar un chatbot"*.

- **Requerimiento No Funcional**

- **Requerimiento de Implementación:** por ejemplo, *"Servidores en la nube"*.

El ***product owner*** participa tanto de los requerimientos de Negocio como de Usuario

User Stories:

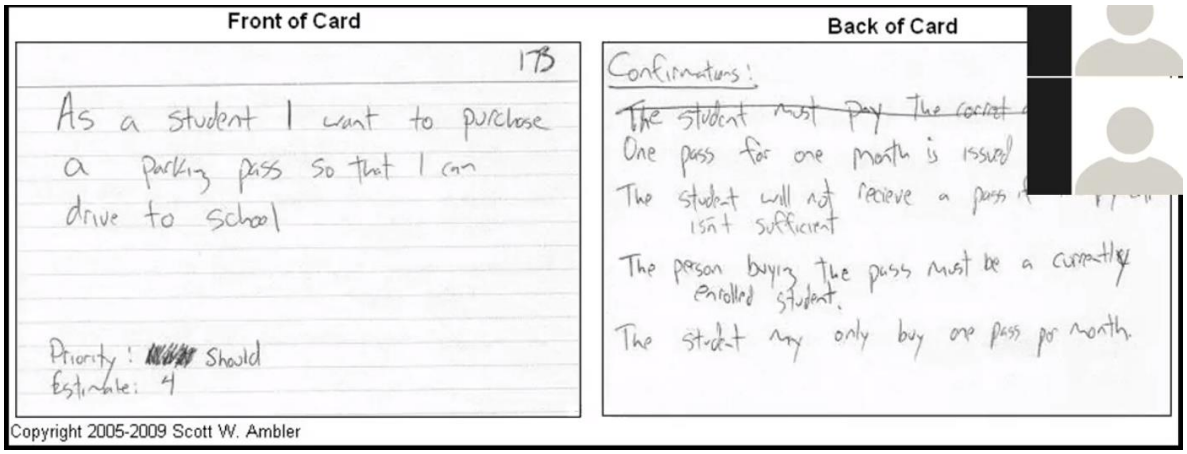
Se llama así porque justamente la idea es que se cuenta una historia, y lo mas importante es la conversación que llevamos adelante cuando contamos esa historia.

Según Fred Brooks: *"La parte más difícil de construir un sistema es definir los requerimientos, es decir, definir precisamente que construir"*

Partes de una User Story:

- **Tarjeta:**

Es lo ***visible*** de la User, es el lugar donde yo voy a escribir la parte de la user que queda plasmada. El contenido de la User es acotado, que contiene una descripción corta de una funcionalidad que incluya el valor para el usuario o cliente.



Forma de expresar la user story (en la parte delantera):

Como <nombre del rol>, yo puedo <actividad> de forma tal que <valor de negocio que recibo>.

Nombre de rol: representa a la persona que esta realizando la acción, o la persona a la que le interesa que se lleve a cabo cierta acción.

Actividad: representa la acción que realizara el sistema

Valor de negocio que recibo: comunica por que es necesaria la actividad, es fundamental porque va a determinar cómo priorizar los requerimientos.

*En un principio, el **product owner** es el que debe escribir las User Stories.*

- **Conversación (la más importante):** porque lo que plantea el agilismo es que el mejor medio de comunicación es cara a cara, y para reducir el *gap* entre lo que el usuario quiere y lo que vamos a construir realmente, lo mejor es charlar cara a cara sobre una US en particular.

- **Confirmación:** la parte de atrás de la tarjeta de la US.

Las User Story son Multipropósito:

Las historias son:

- Una necesidad del usuario
- Una descripción del producto
- Un ítem de planificación
- Token para una conversación
- Mecanismo para diferir una conversación

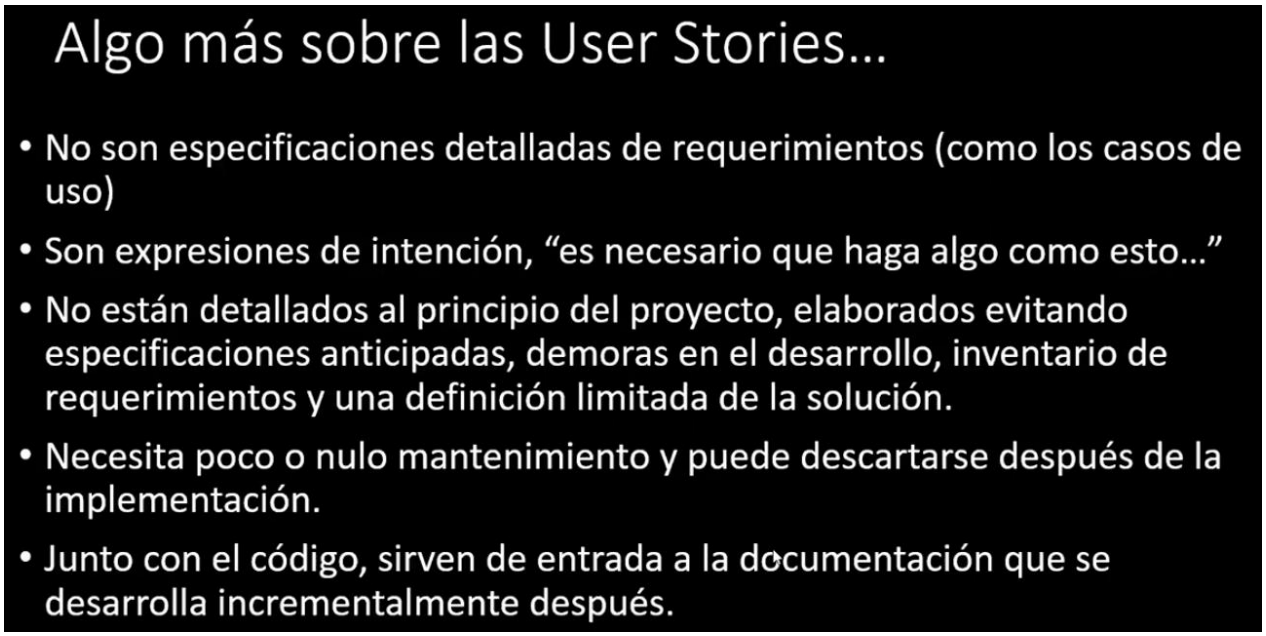
Las user son porciones verticales

Definition of Done – Definición de Hecho:

Es lo que define el equipo de desarrollo para establecer que la US esta efectivamente terminada, es decir, que está lista para ser presentada al Product Owner, como por ejemplo que todos los casos de prueba de la tarjeta se cumplan.

Definition of Ready – Definición de Listo:

La historia esta lista para ser empezada a trabajar en la iteración o sprint siguiente. Se agarran las historias que están mas arriba en el backlog y se tienen que cumplir ciertas condiciones, sobre todo que no tengan ambigüedades sino que la descripción sea bien clara. Es todo lo que tiene que tener la US para identificarla como *“lista”*.



INVEST Model

Modelo para definir que es lo que tiene cumplir una US para entrar en un sprint.

- **Independent:** las historias tienen que ser calendarizables e implementables en cualquier orden. Es decir que las US tienen que ser independientes entre sí.

- **Negotiable:** el *"que"* no el *"como"*. Se habla con el Product Owner sobre que es lo que tiene que hacer la funcionalidad de la US, y se negocia que se hace en cada sprint.

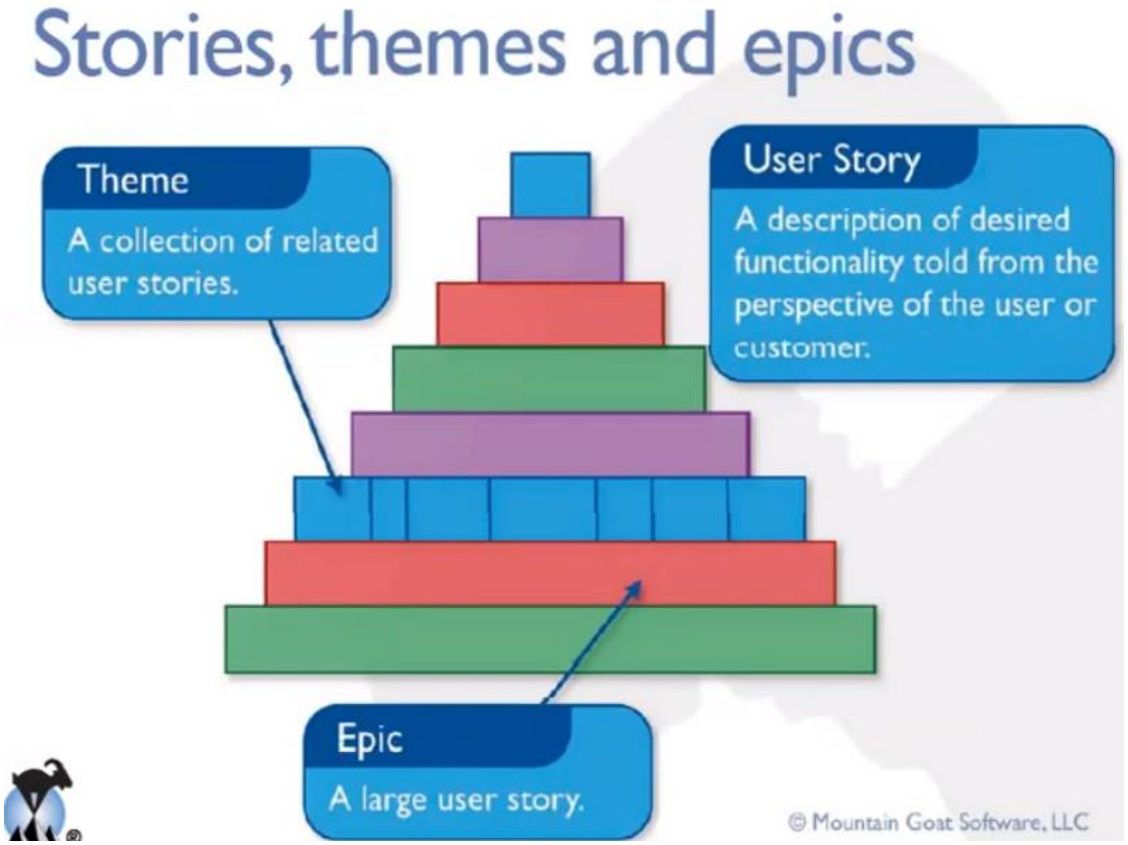
- **Valuable:** debe tener valor para el cliente.

- **Estimable:** para ayudar al cliente a armar un ranking basado en costos, tiene que poder ser estimable.

- **Small:** deben ser *"consumidas"* en una iteración, con Testing y todo.

- **Testable:** demostrar que fueron implementadas.

Diferentes Niveles de Abstracción:



Spykes:

Es un tipo especial de historia, utilizado para quitar riesgo e incertidumbre de una User Story. Va en el product backlog y en realidad se la utiliza para realizar una investigación sobre un aspecto que no nos quede claro dentro de otra US y que necesitemos investigar. Se materializa fundamentalmente cuando se tiene que estimar la US.

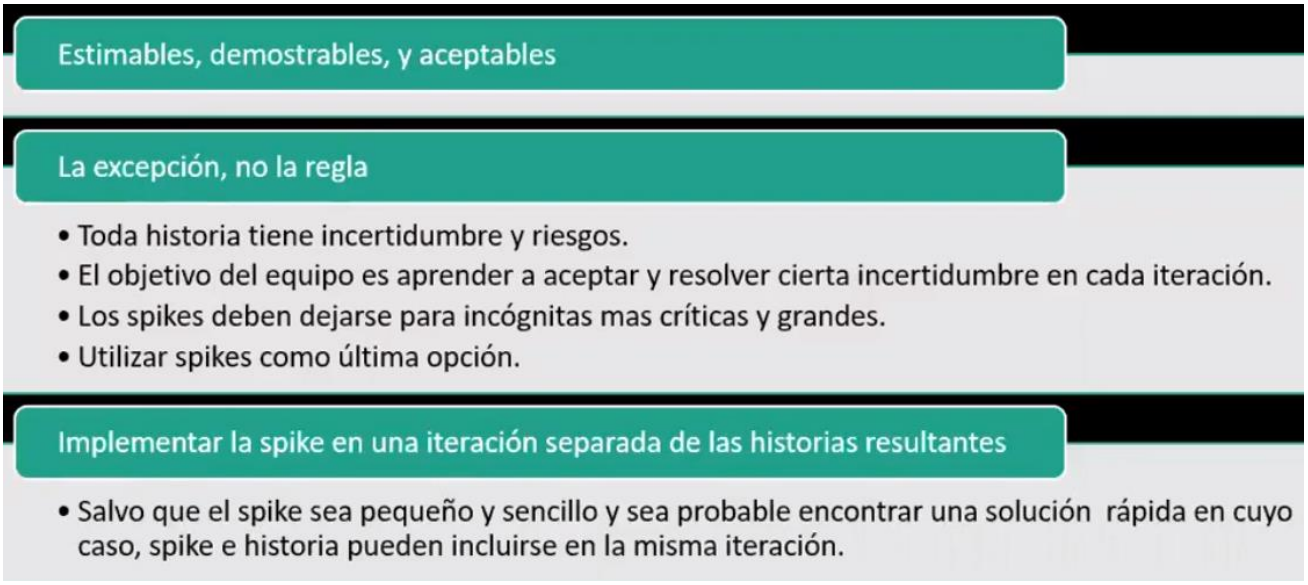
Se clasifican en: ***Técnicas*** *y* ***Funcionales***.

Pueden utilizarse para:

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

Siempre que se tenga que hacer una estimación de una user, siempre va a haber incertidumbre, por ende no siempre se va a aplicar una Spyke para cada incertidumbre que haya.

Lineamientos para las Spykes:



ALGUNAS COSAS PARA DEJAR EN CLARO:

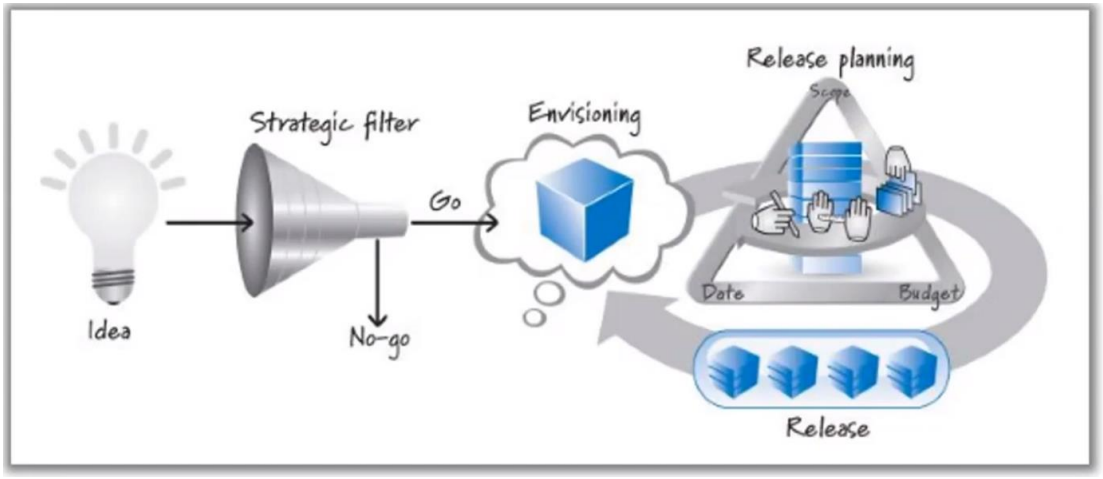
Gestión de Producto:

¿Por qué creamos productos?

- Para satisfacer a los clientes
- Para tener mucho dinero
- Para tener muchos usuarios logueados
- Cambiar el mundo, etc.

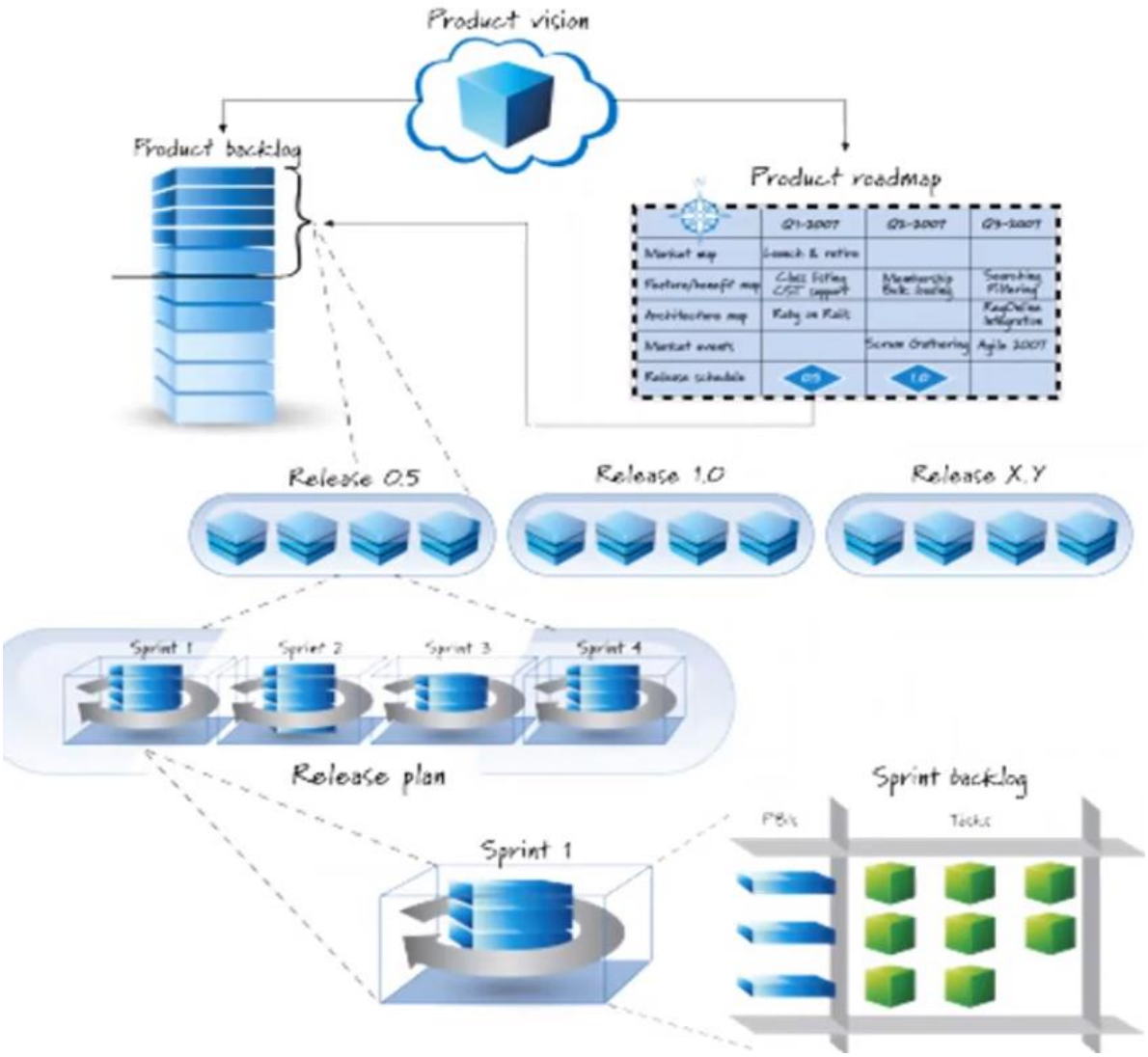
Evolución de los Productos de Software:

Para la creación de productos, hay que enfocarse en ese 20% de las funcionalidades que de verdad se utilizan, en el core funcional básicamente, y eso podría garantizar que nuestro producto sea exitoso. El ciclo de vida iterativo incremental es el mas utilizado en estos casos.



Si se plantea una primera versión del producto, se deberían considerar las funcionalidades esenciales de la idea que se planteó, para lograr el valor principal el cliente.

Planificación de Productos en ambientes lean-agile:



El ***release*** que tengo que hacer, cuando se hace la planificación tengo que decir cuantos sprints o iteraciones voy a necesitar para construir ese release o esa primera versión del producto. El plan de release seria cuantas y cuales serian las iteraciones o sprints necesarias para llevar a cabo la construcción de ese release.

El ***product roadmap*** es una herramienta donde lo que hace el Product Owner es describir en función de la visión en términos generales y en términos de las características que se imagina del producto, en qué momento del tiempo va a contar con esa característica.

Todas las tareas que estén incluidas dentro del ***sprint backlog*** tendrían que haber pasado el *Definition of Ready.*

Minimal Viable Product (Producto Mínimo Viable):

Tiene como objetivo que en esa primera versión del producto yo incluya las ***características necesarias*** que van a hacer que yo pueda ***validar*** que ese producto que estoy pensando ***cumpla con las expectativas de los usuarios***. Por ejemplo, en el mercado ver si a alguien le interesaría comprarlo, o si a alguien le interesa lo que se está construyendo.

El **objetivo** no es vender el producto, sino ***tener retroalimentación*** para poder construir finalmente mi producto pensando en esa retroalimentación del cliente. La premisa seria que en lugar de gastar tanto dinero recopilando información, lo mejor seria concentrarse en mostrar las características del producto, ya sea en video o en forma de prototipo.

Estimaciones Agiles:

Se estiman las features/stories usando una medida de tamaño y la unidad de medida es una ***Story Point***, lo cual es una unidad de medida abstracta.

Son ***medias relativas***, no absolutas, quiere decir que se comparan con ellos mismos, y no con un valor absoluto.

Los story points ***no están basadas en tiempo***. La diferencia entre tiempo y esfuerzo es que el esfuerzo se mide en horas hombre.

Es una ***estimación relativa*** debido a que compara con otra cosa, y siempre se toma de base una historia de usuario.

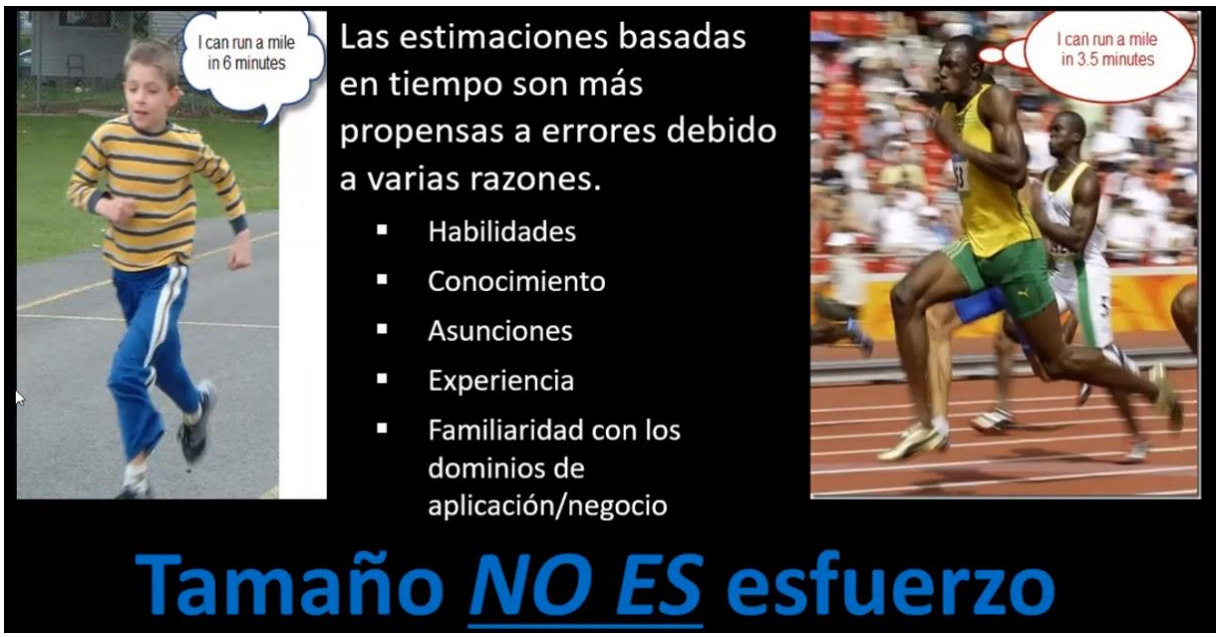
- Las personas no saben estimar en términos absolutos.
- Somos buenos comparando cosas
- Comparar es generalmente más rápido.
- Se obtiene una mejor dinámica grupal y pensamiento de equipo más individual.
- Se emplea mejor el tiempo de análisis de las stories.

El **tamaño** esta directamente relacionado con la ***complejidad***, a eso se refiere con el tamaño de una user story, es decir cuanto esfuerzo requiere esa US.

El tamaño indica:

- Cuan compleja es una feature/story
- Cuanto trabajo es requerido para hacer o completar una feature/story
- Cuan grande es una feature/story

Tamaño vs Esfuerzo:



Luego se utiliza alguna unidad de medida y se elige una **escala**, una vez que se elige ***no se cambia***.

La ***serie de Fibonacci*** es la escala para las *Story Points*, es decir, 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ….

Story Point:

Es una unidad de medida especifica de complejidad, riesgo y esfuerzo, es lo que indica el peso de la user story y decide cuan grande es. La complejidad de una feature/story tiende a crecer exponencialmente

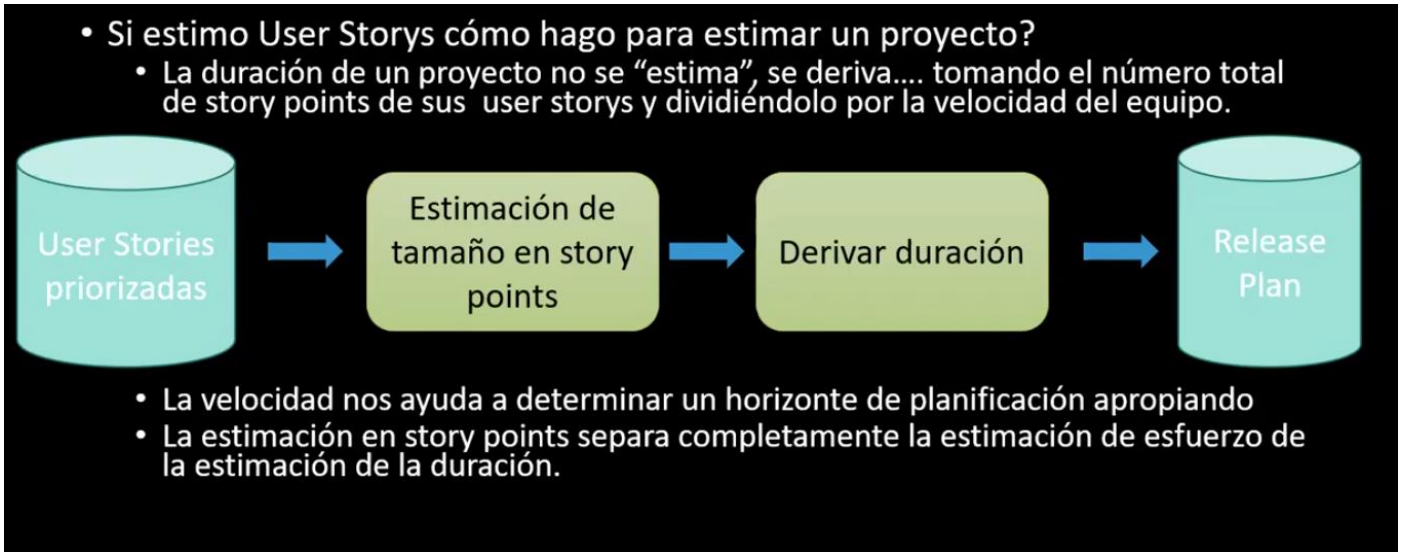
Se utilizan los siguientes parámetros:

- **Complejidad**
- **Esfuerzo**
- **Incertidumbre**

Velocidad (Velocity)

Es una medida (métrica) del progreso de un equipo. Se mide en *Story Points* y es la cantidad de puntos de historia que puede terminar un equipo en un sprint, y que el *product owner* me acepto esas historias. Siempre se cuentan historias ***completas***.

La velocidad corrige los errores de estimación, porque la velocidad es útil para poder estimar y para poder saber cuanto va a poder hacer el equipo en el siguiente sprint.



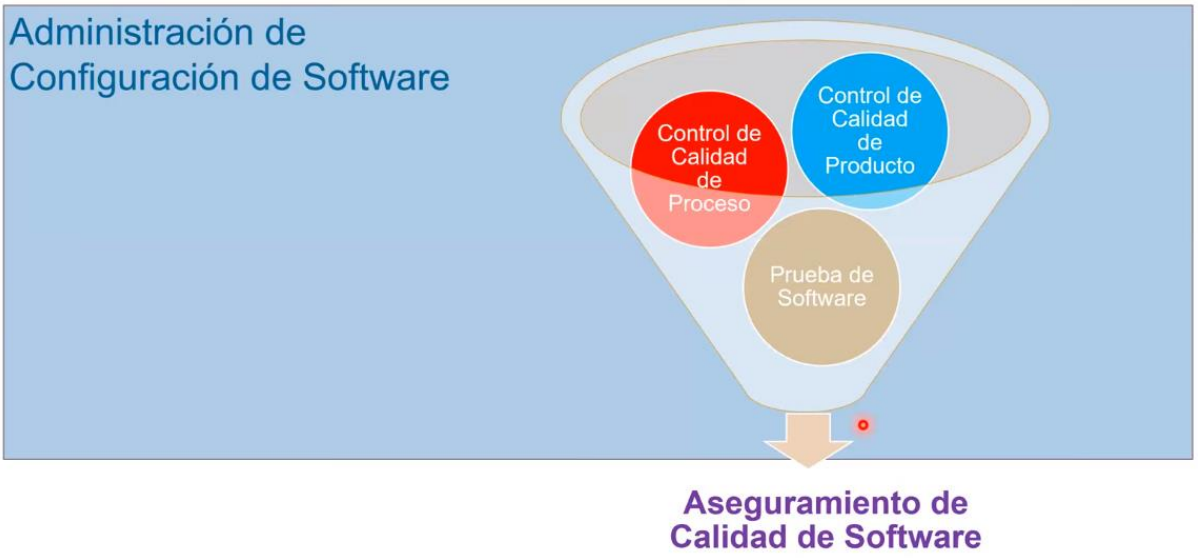
Gestión de Configuración de Software (SCM):

Es una ***disciplina*** (de soporte), protectora, que integra la Ingeniería de Software.

Se la realiza para mantener la integridad del software como producto, y es importante porque el software es maleable y muy fácilmente modificable, por lo que hay que estar preparados para afrontar los cambios de la mejor manera posible, y por ende es necesario mantener la integridad del producto.

Lo que tiene un ***producto para tener integridad*** es que tenga valor para el cliente, satisface las necesidades del mismo en términos de requerimientos funcionales y no funcionales, tiene que ser convenientes en términos de costos y presupuesto, y se tiene que mantener esa situación a lo largo de toda la vida del producto.

Es la base para poder hacer actividades de aseguramiento de calidad.



Hacer Gestión de Configuración de SW implica:

Establecer y mantener la integridad de los productos de software a lo largo de su ciclo de vida.

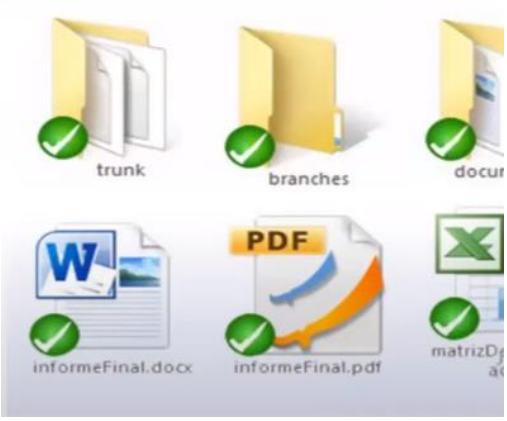
Involucra para la configuración:

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

Conceptos Clave para la SCM:

I. **Ítem de configuración:**

Son todos y cada uno de los artefactos que forman parte del producto o proyecto, es decir, cualquier cosa que puede ponerse en un file System que puede guardarse en una computadora. Pueden sufrir cambios o necesitan ser compartidos entre los miembros del equipo y sobre los cuales necesitamos conocer su estado y evolución.



La ***versión*** es el estado del ítem en un momento determinado y el ***control de versiones*** se refiere a la evolución de un único ítem de configuración.

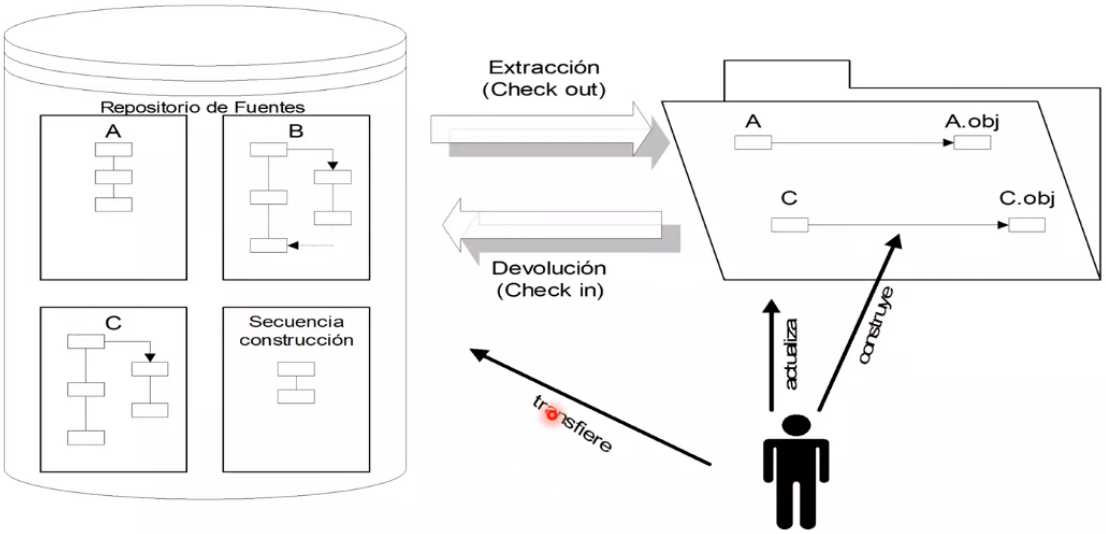
Configuración del Software:

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

II. **Repositorio:**

Es el contenedor de los ítems de configuración. El mismo debería tener una estructura, para tener un orden y para saber su ubicación.

Mantiene la historia de cada ítem de Configuración, junto con sus atributos y relaciones.



III. **Línea Base:**

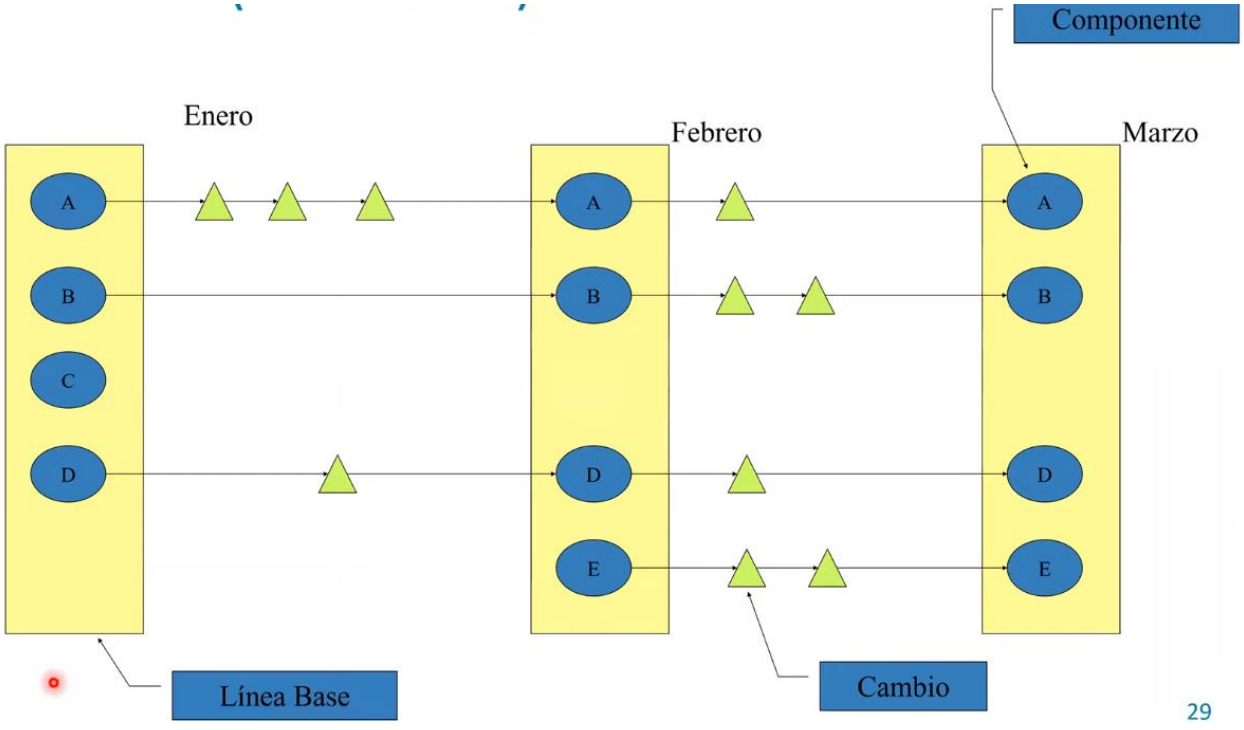
"Es una especificación o producto que se ha revisado formalmente y sobre los que se ha llegado a un acuerdo, y que de ahí en adelante, sirve como base para un desarrollo posterior y que puede cambiarse solamente a través de procedimientos formales de control de cambio".

Una línea base puede contener en un solo momento de tiempo un solo ítem de configuración, o un conjunto de ítems *"estables"*, y debe tener una identificación univoca y una versión.

Tiene que indicar cuáles son sus IC con su estado que forman parte de esa línea base.

Se la utiliza para saber que testear, para hacer rollback, para saber que poner en producción, que desplegar, pero fundamentalmente con tener un punto de referencia.

La idea de que algo llegue a ser Línea base es que los IC que están allí han sido revisados, probados y se ha determinado que están en un estado tal que uno los puede tomar como base para avanzar en el desarrollo.



Los triángulos indican incrementos.

Hay dos tipos de líneas base:

- **De especificación** (Requerimientos, Diseño): no tienen código, sino que tienen información de Ingeniería del producto.
- **Operacionales** o **de productos** que han pasado por un control de calidad definido previamente. Estas si poseen código que es código que ya es operativo.

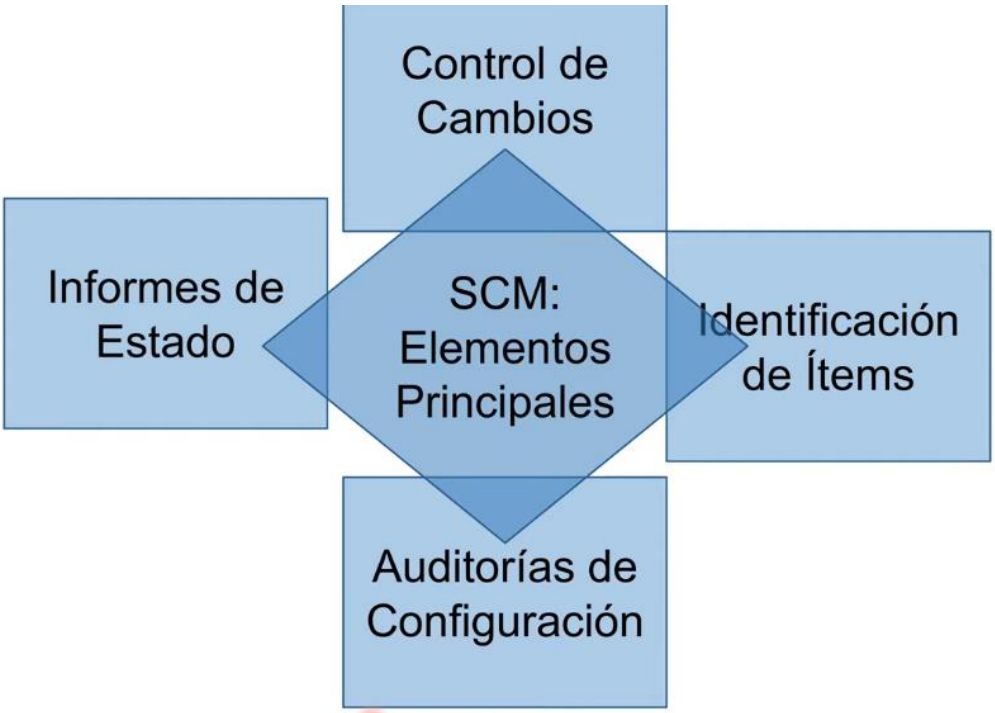
IV. Ramas (Branch):

Básicamente uno arma una bifurcación cuando uno no quiere trabajar sobre la rama principal, y cuando se termina de trabajar sobre la misma, se integra a la rama principal (*merge)*.

- Existe a una rama principal (Trunk, master).
- Las ramas sirven para bifurcar el desarrollo
- Pueden tener razones de creación con semántica.
- Permiten la experimentación.
- Pueden ser descartadas (porque no fue aceptado, o no funcionó) o integradas.

V. Configuración del Software:

Actividades fundamentales de la configuración de Software:



- ***Identificación de Ítems:***

Implica la identificación univoca de los ítems de configuración, se determina la estructura del repositorio, se define la ubicación de los ítems, etc. Lo que se hace es definir reglas de nombrado y esquemas genéricos previamente a el comienzo del proyecto o desarrollo.

Hay tres tipos de ítems de configuración:

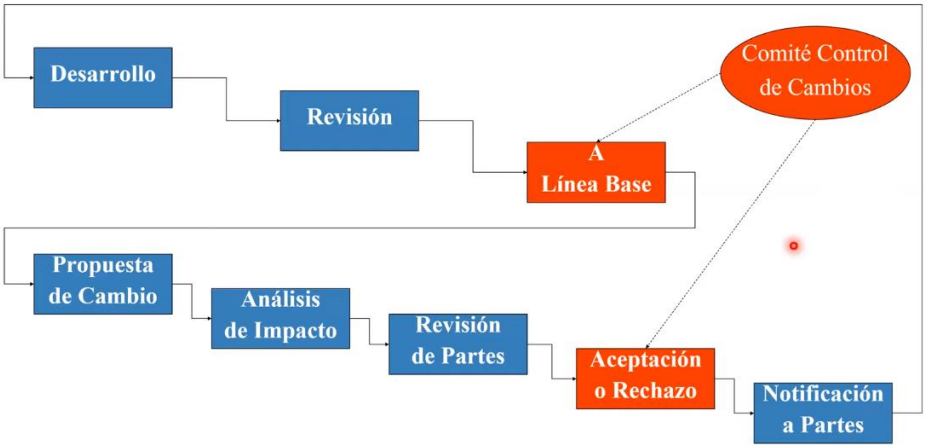
1) **Producto:** tienen el ciclo de vida más largo, y se mantienen mientras el producto exista. Ejemplo: una ERS, los casos de prueba, la base de datos, el manual de usuario.
2) **Proyecto:** los ítems de configuración a nivel de proyecto. El plan de proyecto, el listado de defectos encontrados, tienen un ciclo de vida de proyecto. Un plan de iteración, un burdown chart, se mantiene durante una iteración. La duración impacta también en el esquema de nombrado.
3) **Iteración:** como por ejemplo plan de iteración o reportes de defectos.

Todos los tipos de ítems de configuración ***tienen ciclos de vida distintos***, por ejemplo el ciclo de vida del producto sobrevive al ciclo de vida del proyecto, es decir que a lo largo de un ciclo de vida de un producto va a haber muchos proyectos.

Cuando se les asigna un lugar en el repositorio se tiene que tener en cuenta el tipo de ítem, sobre todo en relación con su ciclo de vida. El que tiene la menor duración o menor ciclo de vida es el de iteración, lo que significa no es que se borran, sino que ya no se gestionan más.

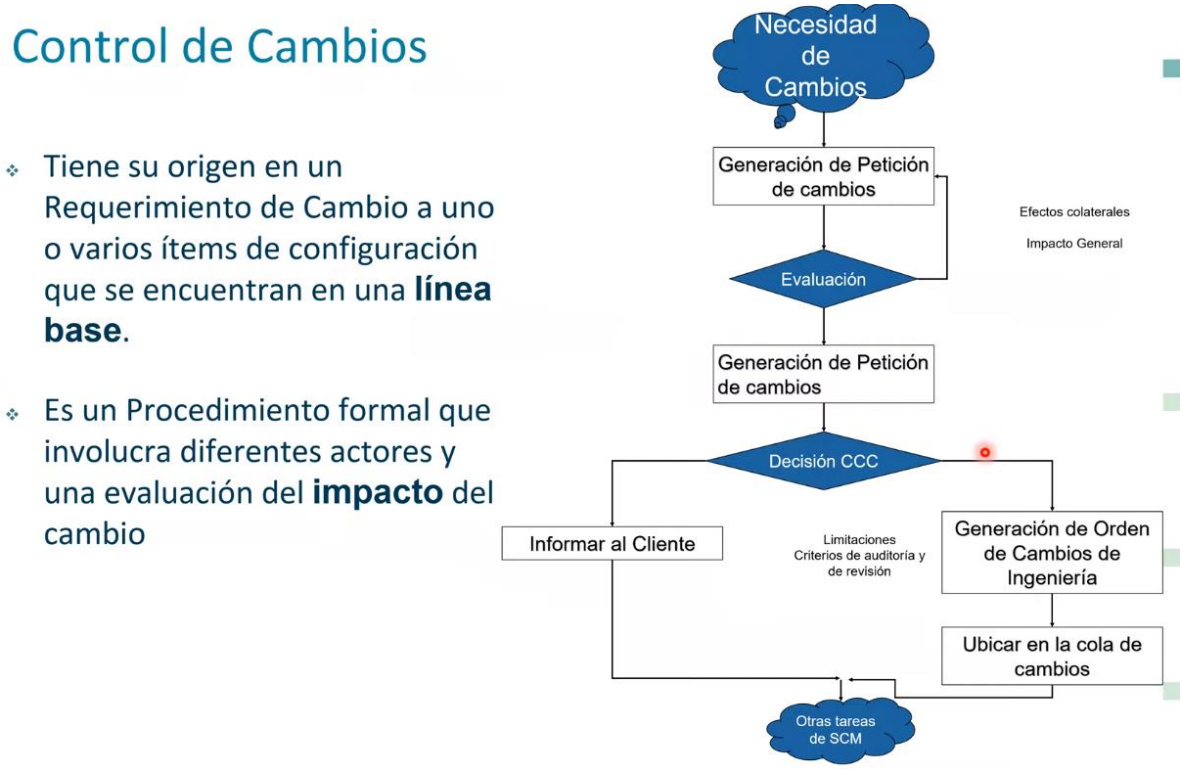
- ***Control de Cambios:***

Tiene que ver con mantener integridad en las líneas base, ***para cambiar una línea base*** tiene que pasar por un proceso formal de control de cambios.



Una vez que la línea base se conformó, no es posible cambiarla sin pasar por un proceso formal, llevado a cabo por lo que se conoce como *comité de control de cambios.*

La formalidad del proceso está dada por el hecho de que todos los involucrados se anoticien. Esta autoridad de cambio, al recibir una ***"propuesta de cambio"***, lleva adelante un ***análisis de impacto*** del cambio, donde se evalúa el esfuerzo técnico, efectos secundarios, impacto global sobre otras funciones y sobre otros objetos, se le asigna una prioridad para que posteriormente se realice lo que se conoce como ***revisión de partes***, en base a todo este análisis el comité ***acepta o rechaza el cambio*** y ***notifica a todas las partes involucradas***.



Comité de Control de Cambios:

Esta integrada por gente de todas las areas involucradas en el desarrollo, *son referentes del equipo que esta trabajando en el desarrollo del producto*, y se define a quien le interesa pertenecer a ese comité.

Roles:

- Arquitecto
- Alguien que vele por los requerimientos funcionales
- Los desarrolladores
- Líder de Proyecto que va a velar por los recursos, presupuesto, tiempo, etc.

- ***Auditorias de Configuración:***

La auditoría de configuración tiene como objetivo asegurar que lo que está indicado para cada Ítem de Configuración de Software en la línea base o actualización se ha alcanzado realmente y que el software y la documentación son internamente consistentes para entregarlos al cliente.

Debe ser objetiva e independiente

El que controla no debe estar involucrado con lo controlado, debe ser externo del proyecto.

Funciones:

- Determinar la semejanza entre el estado actual del sistema y el establecido como línea base.
- Provee el mecanismo para establecer una línea base.
- Transición desde:
  - línea base a establecer (en etapas formativas),
  - línea base sancionada

Procesos a los que sirve:

- **Validación:**

Construir el producto correcto

Asegurar que el problema se ha resuelto de la manera apropiada de tal manera de permitir que el usuario obtenga el producto correcto.

- **Verificación:**

Construir el producto correctamente

Asegura que un producto cumple con los objetivos definidos en la documentación de líneas base. Todas la funciones son llevadas a cabo con éxito y los test cases tengan status “ok” o bien consten como “problemas reportados” en la nota de release.

<u>Tipos de auditoria:</u>

- **Auditoria de Configuración física:**

Asegura que lo que está indicado para cada Ítem de Configuración de Software en la línea base o actualización se ha alcanzado realmente y que el software y la documentación son internamente consistentes para entregarlos al cliente.

Vela por la **integridad del repositorio**, que este, que este alojado donde debería estar, que los IC respeten el esquema de nombrado, que estén guardados donde se dijo que iban a estar nombrados. ***Lo que se audita es una línea base.***

El **auditor *tiene que ser alguien externo*** al proyecto.

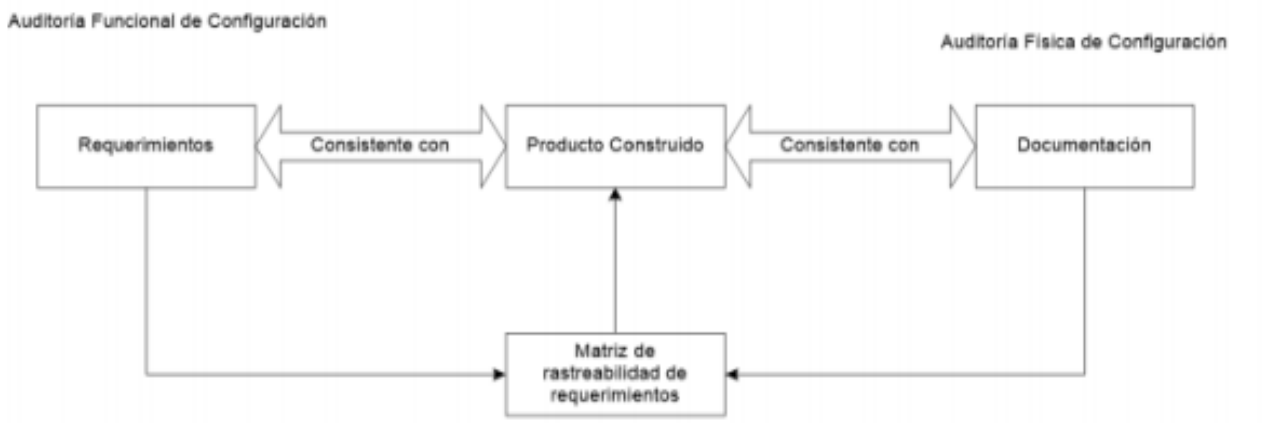
- El objetivo es **verificar**
- Se verifica consistencia contra la documentación
- La practica indica que ***primero se hace la auditoria física y luego la funcional.***

- **Auditoria de configuración funcional:**

Es la evaluación independiente de los productos de software, verificando que la funcionalidad y rendimiento reales de cada ítem sean consistentes con la especificación del requerimiento.

Ve si el producto es el producto correcto, es decir si los IC hacen lo que los requerimientos dicen que tienen que hacer.

- El objetivo es **validar**
- Se busca asegurar que el producto es lo que realmente el cliente pidió, y se compara contra la especificación de requerimientos.
- Matriz de rastreabilidad para saber qué caso de prueba corresponde a cada requerimiento



- ***Informes de estado:***

Se generan reportes, y la idea de generar informes es la de dar visibilidad para poder tomar decisiones.

Los informes nos dicen el estado actual de la configuración del software, el informe más conocido es el inventario, que contiene una copia del contenido del repositorio.

Objetivos:

- Mantener los registros de la evolución del sistema. Incluye reportes de rastreabilidad de todos los cambios realizados a las líneas base durante el ciclo de vida, es decir nos dice cuáles son las líneas base, cuando se modificó, quien modifico cada cosa.

- Manejan mucha información y salidas por lo que se suele implementar dentro de procesos automáticos.

- Este informe es útil desde el punto de vista administrativo para realizar auditoria. CCB (change control board – comité de control de cambios).

Plan de Gestión de Configuración

Debería tener respuestas a las preguntas de como se van a hacer las actividades de la gestión de configuración.

- Reglas de nombrado de los CI
- Herramientas a utilizar para SCM
- Roles e integrantes del Comité
- Procedimiento formal de cambios
- Plantillas de formularios
- Procesos de Auditoría