Ingeniería y Calidad

De Software

1 – Introducción a la Ingeniería del Software

Software ¿Qué es?

El software son programas de cómputo y documentación asociada. Los productos de software se desarrollan para un cliente en particular o para un mercado en general.

El buen software debe entregar al usuario la funcionalidad y el desempeño requeridos, y debe ser sustentable, confiable y utilizable.

Ingeniería de Software

La ingeniería de software es una disciplina de la ingeniería que se interesa por todos los aspectos de la producción del software, desde las primeras etapas de la especificación del sistema hasta el mantenimiento del sistema despues de que se pone en operación.

La ingeniería de sistemas se interesa por todos los aspectos del desarrollo de sistemas basados en computadoras, incluidos hardware, software e ingeniería de procesos. La ingeniería de software es parte de este proceso más general.

Antecedentes

El software era desarrollado principalmente para aplicaciones científicas y militares. Los programas eran relativamente simples, escritos por los mismos científicos que usaban las computadoras. A medida que las computadoras se volvieron más accesibles, se empezaron a usar en diversos sectores como los bancos, la aviación, la administración pública y la industria. Esto trajo una necesidad de sistemas más complejos. Las técnicas de programación eran rudimentarias, y no existían metodologías formales para gestionar proyectos grandes. Esto llevo a problemas de fiabilidad, mantenimiento y escalabilidad.

Crisis del Software

Es un término popularizado en 1968 en una conferencia de la OTAN, donde se describen el conjunto de principales problemas del software de aquella época.

- Proyectos que no se completaban o se entregaban tarde
- Costos que se disparaban más allá de lo presupuestado
- Software lleno de errores y difíciles de mantener
- Falta de personal capacitado y metodologías organizadas

Estado actual

La crisis dio origen a la Ingeniería de Software como disciplina formal. Hoy contamos con metodologías como Agile, DevOps, y modelos como Scrum y Kanban. Lo que no significa que no se presenten desafíos en la actualidad:

- Proyectos aún pueden exceder tiempos y presupuestos
- La complejidad sigue creciendo con la inteligencia artificial, la nube y los sistemas distribuidos
- La seguridad y la privacidad son ahora preocupaciones centrales

Disciplinas de la Ingeniería de Software

- **Disciplinas Técnicas**
  - ❖ Requerimientos
  - ❖ Análisis y Diseño
  - ❖ Construcción
  - ❖ Prueba
  - ❖ Despliegue
- **Disciplinas de Gestión**
  - ❖ Planificación de Proyecto
  - ❖ Monitore y Control de Proyectos
- **Disciplinas de Soporte**
  - ❖ Gestión de Configuración de Software
  - ❖ Aseguramiento de Calidad
  - ❖ Métricas

Proceso de Software

Serie de actividades, acciones y tareas relacionadas que la gente usa para conducir el desarrollo o mantenimiento de un producto de software y sus productos

asociados. Una **actividad** busca lograr un objetivo amplio y se desarrolla sin importar el dominio de la aplicación, tamaño del proyecto, complejidad del esfuerzo o grado de rigor con el que se usara la ingeniería de software. Una **acción** es un conjunto de tareas que producen un producto importante del trabajo. Una **tarea** se centra en un objetivo pequeño, pero bien definido que produce un resultado tangible.

El Proceso de desarrollo lo define la organización en función de los objetivos y de las características del software.

- **Proceso Definido**

  Inspirado en las lineas de producción. Asume que podemos repetir el mismo proceso una y otra vez, indefinidamente, y obtener los mismos resultados. La administración y control provienen de la predictibilidad del proceso definido. Esto es difícilmente aplicable al software, ya que no es tan definible. No podemos simplificar el software a una entrada de inputs que producen el mismo output, es decir, el output no es predecible en el software.

  Muchas corrientes y procesos confían en estas premisas para llevar a cabo un proyecto de software.

- **Proceso Empírico**

  Viene del empirismo, centra su foco en la experiencia del usuario, de negocio y la experiencia técnica. Asume procesos complicados con variables cambiantes, lo que implica la necesidad de adaptación para garantizar un software de calidad. Cuando se repite el proceso, se pueden llegar a obtener resultados diferentes. La administración y el control es a través de inspecciones frecuentes y adaptaciones. Son procesos que trabajan bien con procesos creativos y complejos.

  El proceso empírico no tiene un punto de mejora tan tangible como el definido. Este trabaja sobre distintos aspectos centrándose en las personas, las cuales capitalizan la experiencia. Esto lo hacen a través de un ciclo de retroalimentación que permite la adaptación y mejora, aprendiendo de la experiencia. Los procesos empíricos trabajan con un único tipo de ciclo de vida, el iterativo.

Proyecto de Software

Unidad de gestión del trabajo. Es la instancia especifica en la que se aplica el proceso para desarrollar un producto. Implica: Planificación, Gestión de riesgos,

Seguimiento de tareas, y Coordinación de equipo. Dirigidos a obtener resultados y ello se refleja a través de objetivos, los cuales no deben ser ambiguos debido a que guían a dicho proyecto. Las características de un proyecto son:

- Resultado Único
- Orientado a un objetivo
- Fecha de Inicio y Fin
- Elaboración gradual
- Tareas interrelacionadas

Producto

Software desarrollado como resultado del proyecto. Puede ser un sistema a medida o un producto genérico. Su calidad depende de la ejecución del proceso, la gestión del proyecto, y el entendimiento y cumplimiento de los requerimientos.

Relación Producto – Proyecto – Proceso en la Gestión de Proyecto

El **proceso** define el "cómo", se aplica en el proyecto para crear el producto. El **proyecto** ejecuta el "qué", usa el proceso para lograr el producto. El **producto** es el resultado, refleja la calidad del proceso y la gestión del proyecto.

- **El producto es intangible**

  El software es intangible, no se puede ver ni tocar. Los administradores de proyectos de software no pueden constatar el progreso con solo observar el artefacto que se construye. Mas bien, ellos se apoyan en otros para crear la prueba que pueden utilizar al revisar el progreso del trabajo.

- **Los grandes proyectos de Software con frecuencia son excepcionales**

  Incluso los administradores que cuentan con vasta experiencia pueden encontrar difícil anticiparse a los problemas. En conjunto con los vertiginosos cambios tecnológicos en computadoras y comunicaciones, pueden volver obsoleta la experiencia de un administrador. Las lecciones aprendidas de proyectos anteriores pueden no ser aplicables a nuevos proyectos.

- **Los procesos de software son variables y específicos de la organización**

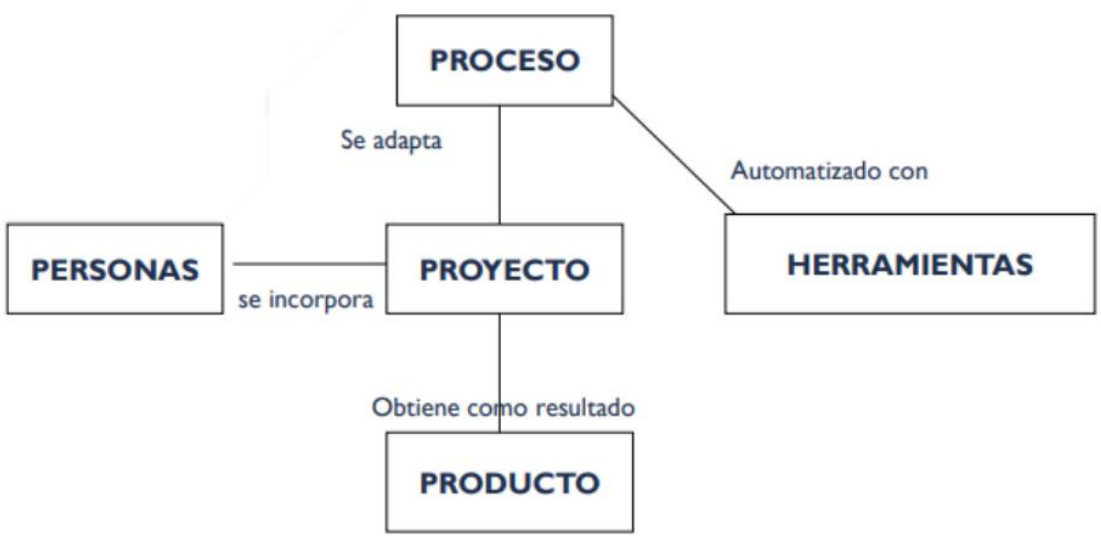
  Los procesos de software varían considerablemente de una organización a otra. No es posible predecir de manera confiable cuando un proceso de software particular conducirá a problemas de desarrollo. Esto es especialmente cierto si

el proyecto de software es parte de un proyecto de ingeniería de sistemas más amplio.

Componentes de un proyecto

Se suele decir que el proyecto es una instanciación del proceso en un ciclo de vida.

Un **proyecto** es llevado a cabo por **personas** que implementan **herramientas** para automatizar los **procesos** y que tiene como resultado un **producto**.



Administración de Proyectos

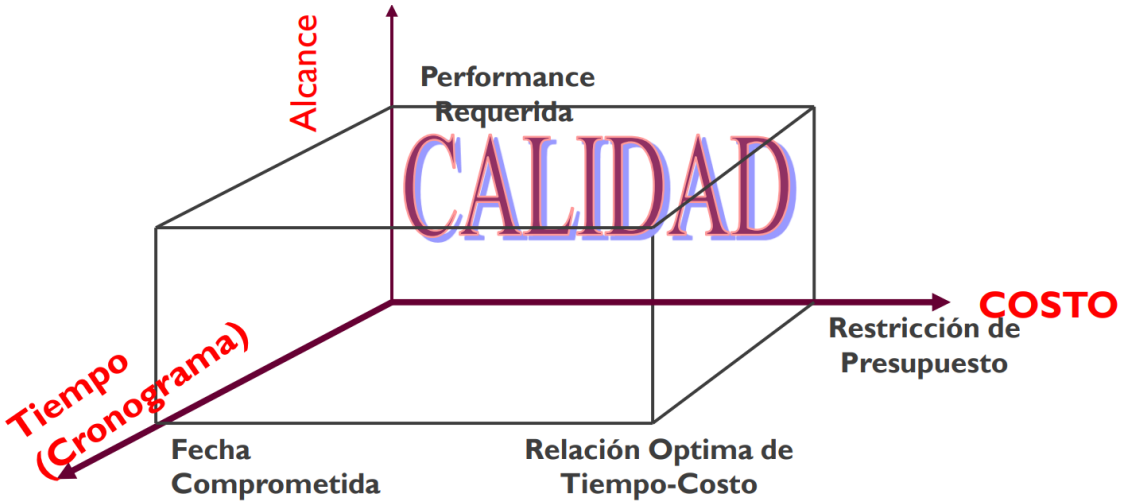
Aplicación de conocimientos, habilidades, herramientas y técnicas a las actividades del proyecto para satisfacer los requerimientos del proyecto y poder obtener como resultado el producto esperado.

La triple restricción

Es una de las bases de la gestión tradicional. Plantea que un proyecto va a tener una **restricción de tiempo**, una **restricción de presupuesto** y un **alcance**. Estas tres variables estan íntimamente relacionadas de manera tal que, si se modifica una, se deben modificar una de las otras o ambas. El balance de las mismas afecta la calidad del proyecto, y debemos tener en cuenta que, **la calidad del producto no se negocia**.

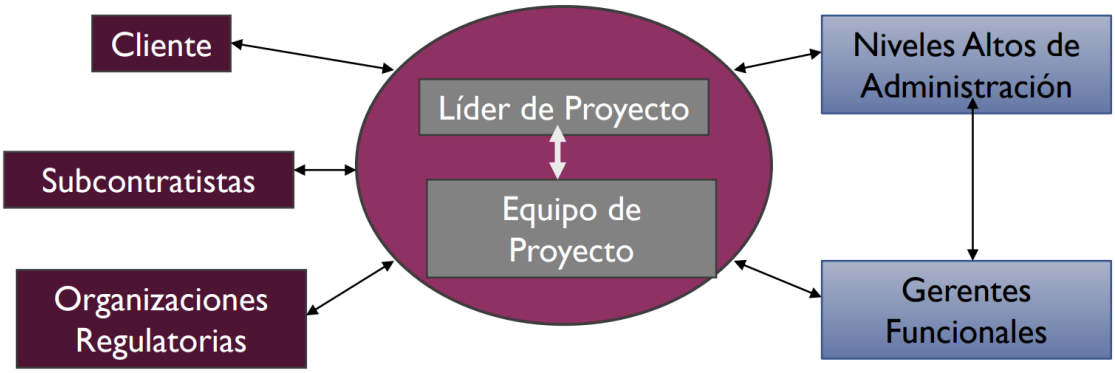
- **Alcance**
  Son los requerimientos que el cliente expresa, el objetivo, lo que quiero alcanzar.

- **Tiempo**
  Tiempo que debería llevar completar el proyecto.
- **Costo**
  Costo en recursos y dinero.



Roles dentro de un proyecto

En la gestión tradicional, nuestro equipo está formado por un **equipo de proyecto** y un **líder de proyecto** (PM).



En los procesos empíricos, el PM no es necesario ya que los proyectos se autogestionan. Es un rol propio de la gestión tradicional.

Plan de Proyecto

La idea es que sea algo vivo, que se vaya nutriendo y corrigiendo a medida que el proyecto se lleva a cabo. Debe estar en permanente actualización, ya que en el entorno de desarrollo del proyecto se encuentran con variaciones no estimadas al determinar el plan debido a la incertidumbre inicial.

En un plan de proyecto se deben definir las siguientes cosas:

- **Objetivo del proyecto**
  Define el ¿Qué? Debe ser claro y alcanzable.
- **Alcance del proyecto**
  Para definir el alcance del proyecto, primero se debe definir el alcance del producto.

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 en el objetivo. Se mide contra el plan del proyecto	Son todas las **características** que pueden incluirse en un producto o servicio. Se mide contra la especificación de requerimientos

- **Definir el proceso y el ciclo de vida**
  Cuando inicia el proyecto se debe definir qué proceso usar y con que ciclo de vida. Se define el ¿Cómo?

- **Definir el equipo de proyecto**
  Define el ¿Quién?

- **Estimación**
  - **Tamaño**
    Define el producto a construir. Casos de Uso x Complejidad.
  - **Esfuerzo**
    Horas Persona Lineal del proyecto.
  - **Tiempo**
    Tiempos del proyecto.
  - **Costos**
    Determinar un presupuesto necesario.
  - **Recursos críticos**
    Tanto humanos como físicos que van a hacer falta.

- **Calendarización – Programación**
  Determinar días y horas que se van a trabajar, además de cuantas personas.

- **Riesgos**
  Identificar los riesgos más probables o que más pueden impactar. Luego, se gestionan, generando acciones para disminuir el impacto o evitar la ocurrencia. Los que no se pueden evitar, se debe encontrar la manera de aliviarlos. A medida que el proyecto avanza, los riesgos pueden aumentar, disminuir, ser más probables o aparecer nuevos.

- **Métricas**
  Permiten identificar si un proyecto está en línea con lo planificado o si está desviado. Se pueden clasificar en:
  - **Métricas de Proceso**
    Permiten saber si como organización se trabaja bien o mal, y que pasa en términos organizacionales. Son las mismas métricas del proyecto, pero despersonalizadas de un proyecto en particular.
  - **Métricas de Proyecto**
    Permiten saber si un proyecto de software en ejecución se está cumpliendo de acuerdo con lo planificado o no.
  - **Métricas de Producto**
    Miden cuestiones que tiene una relación directa con el producto que se está construyendo.

  Métricas básicas:
  - **Tamaño**
  - **Esfuerzo**
  - **Tiempo**
  - **Defectos**

- **Seguimiento y Control**
  El PM se encarga de trabajar sobre lo definido en el plan del proyecto y determinar si se está cumpliendo o no. Para hacerlo se basa en el plan de proyecto y las métricas. Si se pueden corregir los desvíos de un proyecto en el tiempo adecuado, se está a tiempo de poder cumplir el objetivo.

2 – Ciclos de Vida

Serie de pasos a través de los cuales el producto o proyecto progresa. Define cuales son las etapas y el orden de cada una. No define que, quien, con que ni cómo hacer, sino las fases y en qué orden se ejecutan.

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

Ciclos de Vida de Proyecto de Software

Representación de un proceso. Grafica una descripción desde una perspectiva particular. Los modelos especifican las fases de proceso y el orden en el cual se llevan a cabo.

Ciclos de Vida de Producto de Software

El ciclo de vida de un proyecto termina cuando genera un producto, cuyo ciclo de vida va a necesitar mantenimiento una vez que "termino" de desarrollarse, y se va a mantener bajo este proceso de mantenimiento hasta que se deseche. Ahí podemos decir que el ciclo de vida del producto finaliza.

Tipos de Ciclos de vida

- **Secuencial**

  Se basan en ejecutar una etapa despues de otra sin retorno generalmente. Hay algunos que pueden llegar a devolver información (la cascada, por ejemplo).

- **Iterativo**

  Permiten la adaptación y mejora del proceso, a través de iteraciones. Además, permite el aprendizaje, mejorando la experiencia del equipo y realizando otra iteración para incrementar el producto.

- **Recursivo**

  Se utiliza para casos particulares como proyectos de alto riesgo. Se basa en tomar una característica especifica y en una iteración me centro en esa funcionalidad, en la siguiente en otra, y asi sucesivamente. Estos tipos de

ciclos de vida están en desuso debido a que se comprobó que no son útiles y no funcionan.

Criterios de elección

Para elegir un ciclo de vida, lo único importante no es la velocidad del mismo, ya que esta varía dependiendo del contexto. Para elegir el mejor ciclo de vida para un proyecto, se deben evaluar las necesidades del mismo y las características del producto. Las necesidades y características más importantes, las que más debemos tener en cuenta son:

Criterio	Cascada puro	Code-and-Fix	Espiral	Cascada modificado	Prototipado evolutivo
Requerimientos poco entendidos	Pobre	Pobre	Excelente	Regular a excelente	Excelente
Arquitectura poco entendida	Pobre	Pobre	Excelente	Regular a excelente	Regular
Alta confiabilidad del sistema	Excelente	Pobre	Excelente	Excelente	Regular
Gran capacidad de crecimiento	Excelente	Regular	Excelente	Excelente	Excelente
Gestión de riesgos	Pobre	Pobre	Excelente	Regular	Regular
Ajuste a cronograma	Regular	Pobre	Regular	Regular	Pobre
Carga administrativa	Pobre	Excelente	Regular	Excelente	Regular
Correcciones en el curso del proyecto	Pobre	Regular	Regular	Regular	Excelente
Visibilidad del progreso para el cliente	Pobre	Regular	Excelente	Regular	Excelente
Visibilidad del progreso para la gerencia	Regular	Pobre	Excelente	Regular a excelente	Regular
Baja sofisticación requerida	Regular	Excelente	Pobre	Pobre a regular	Pobre

3 – Filosofía Ágil

No es una metodología, ni un proceso, es una **ideología** que cuenta con un conjunto definido de principios que guían el desarrollo del producto. Busca encontrar un equilibrio entre procesos definidos y nada de procesos.

Se sustenta en los procesos empíricos que tienen como base la experiencia, por lo que es importante tener ciclos de realimentación cortos. Este enfoque es compatible únicamente con ciclos de vida iterativos y se basa en la idea de que la incertidumbre y los cambios son inevitables en el desarrollo de software, por lo que es necesario trabajar de manera flexible y adaptativa para satisfacer las necesidades del cliente.

Pilares del empirismo

- **Transparencia**

  Se refiere a la comunicación abierta y honesta de la información relevante a todas las partes interesadas. El proceso y el trabajo emergentes deben ser visibles tanto para quienes realizan el trabajo como para quienes lo reciben.

  La transparencia permite la inspección. La inspección sin transparencia es engañosa y derrochadora.

- **Inspección**

  Se refiere a la revisión regular y sistemática del progreso de trabajo y los productos resultantes para detectar problemas y desviaciones del plan.

  La inspección permite adaptación. La inspección sin adaptación se considera inútil.

- **Adaptación**

  Se refiere a la capacidad de adaptarse y ajustar el trabajo y el proceso para hacer frente a las desviaciones y problemas detectados durante la inspección. Si algún aspecto de un proceso se desvía fuera de los limites aceptables o si el producto resultante es inaceptable, el proceso que se aplica o los materiales que se producen deben ajustarse. El ajuste debe realizarse lo antes posible para minimizar una mayor desviación.

  La adaptación se vuelve más difícil cuando las personas involucradas no están empoderadas ni se autogestionan.

Valores Ágiles

- **Individuos e interacciones por 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.

- **Software funcionando por sobre documentación extensiva**

Enfatiza la importancia de proporcionar software funcional y de alta calidad en lugar de enfocarse en documentar cada detalle del proceso de desarrollo.

Se debe decidir qué aspectos del producto y/o proyecto van a quedar documentados. Además, se debe decidir que debe tener permanencia y disponibilidad infinita.

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

Enfatiza la importancia de trabajar en colaboración con el cliente y comprender sus necesidades y requisitos en lugar de simplemente negociar contratos y acuerdos formales.

Es fundamental que el cliente esté dispuesto a participar y sumarse a este enfoque, ya que, de lo contrario, la filosofía ágil no se puede aplicar. Por lo tanto, es importante formar al cliente, además del equipo, en el enfoque ágil.

- **Responder al cambio por sobre seguir un plan**

Enfatiza la importancia de ser flexible y adaptativo en la 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.

Plantea que debemos construir junto con el cliente, no definir tanto, empezar a trabajar una idea del producto y despues enfocarnos más a medida que avanzamos para darle la posibilidad al cliente de cambiar de opinión. Es mejor tener una actitud más ajustada con la realidad de que los requerimientos cambian.

Principios agiles

- **Satisfacción del cliente**

Nuestra mayor prioridad es **satisfacer al cliente** a través de la entrega temprana y continua de software con valor.

- **Adaptación al cambio**

Aceptamos que los requerimientos **cambien**, incluso en etapas tardías del desarrollo. Los procesos agiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.

- **Entregas frecuentes**

Entregamos software funcional con **frecuencia**, con preferencia a la documentación exhaustiva.

- **Trabajo en equipo**

Colaboramos con el cliente y los interesados durante todo el proyecto para asegurarnos de que el software entregado satisfaga las necesidades del negocio.

- **Personas motivadas**

Construimos proyectos en torno a **individuos motivados**. Les damos el entorno y el apoyo que necesitan y confiamos en que harán el trabajo.

- **Comunicación directa**

El método más eficiente y efectivo de comunicar información dentro de un equipo de desarrollo de software es la **conversación cara a cara**. La efectividad y la riqueza de la comunicación crece exponencialmente conforme estamos en un mismo espacio físico y compartiendo un pizarrón para construir juntos el producto que necesitamos.

- **Software funcionando**

El **software funcionando** es la principal **medida de éxito**.

- **Continuidad**

Los procesos agiles 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.

- **Excelencia técnica**

La **atención continua a la excelencia técnica y el buen diseño** mejoran la agilidad. La calidad del producto no es negociable, debemos ajustar cualquier aspecto del producto menos su calidad al entregar.

- **Simplicidad**

Al buscar la **simplicidad**, se puede maximizar la eficiencia y efectividad, al mismo tiempo que se reduce la complejidad y el riesgo de errores y problemas.

Esto se logra a través de la eliminación de tareas innecesarias y la concentración en los objetivos más importantes y críticos del proyecto. No agregar funcionalidades porque si, si el cliente no lo pidió.

- **Equipos auto organizados**

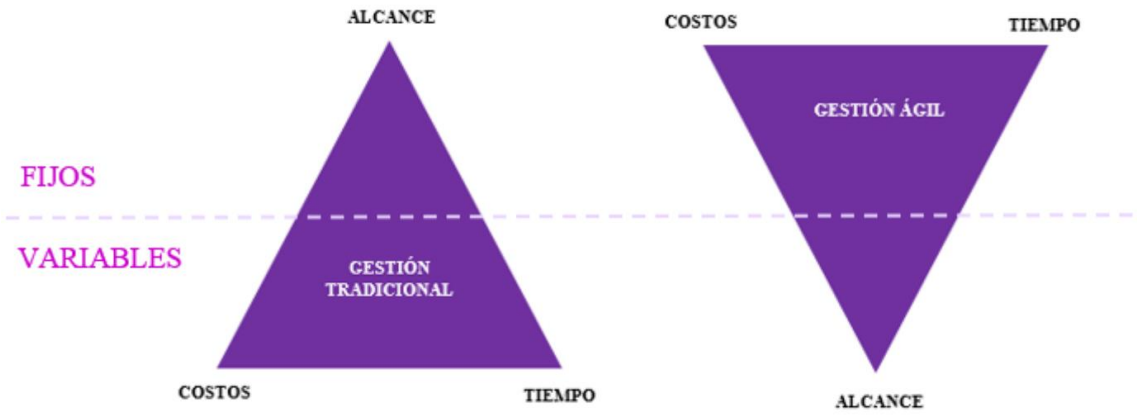
Las mejores arquitecturas, requisitos y diseños emergen de **equipos auto organizados** con libertad de tomar decisiones y diseñar soluciones de manera colaborativa. Los más aptos para definir características de los productos, son los que van a desarrollarlos.

- **Mejora continua**

El equipo reflexiona sobre como ser más efectivo para luego **ajustar y perfeccionar su comportamiento** en secuencia durante ciertos intervalos de tiempo.

La Triple Restricción

Al igual que la gestión tradicional, la filosofía ágil, también plantea la triple restricción en base a los tres elementos (alcance, costo, tiempo). Sin embargo, considera que son variables flexibles que pueden adaptarse y cambiarse a lo largo del proyecto. En lugar de establecer un plan fijo al inicio del proyecto, se utilizan iteraciones cortas y regulares para adaptar el proyecto según las necesidades y los requisitos cambiantes. Este modelo deja fijos el tiempo y el costo, manteniendo variable las funcionalidades o alcances. ***"Tengo estos recursos y este tiempo, con eso ¿Qué le puedo entregar al cliente?"***.



Requerimientos Agiles

El **valor de negocio** y la construcción del producto correcto son fundamentales. Se plantea que los requerimientos se irán descubriendo a medida que avanza el proyecto y se busca definir **"solo lo suficiente"** para comenzar a trabajar con lo

mínimo necesario y obtener retroalimentación lo antes posible para mejorar continuamente. En lugar de especificar todos los requerimientos desde el principio, el desarrollo de requerimientos está enfocado en trabajar en conjunto con el cliente.

Los requerimientos se expresan en forma de **historias de usuario**, las cuales se utilizan para **definir el alcance** de cada iteración del proyecto y **se priorizan en función del valor que aportan al producto**.

Una de las razones por las cuales necesitamos cambiar el enfoque, es decir, determinar **solo lo suficiente**, es porque todos los productos de software tienen un **desperdicio significativo**, esto quiere decir que, de todas las funcionalidades desarrolladas, es muy alto el porcentaje de aquellas que el cliente/usuario no usa. Por lo tanto, la gestión ágil de requerimientos intenta contrarrestar esta situación, priorizando solo aquellas que aportan valor al cliente.

Just in Time

Es un concepto que se usa fundamentalmente para **evitar desperdicios** (en este contexto seria invertir tiempo en especificar requerimientos que despues cambian). El producto no va a estar especificado 100% desde el principio, vamos a ir encontrando requerimientos y describiéndolos conforme haga falta.

Lo que nosotros tenemos que entregarle al cliente es valor de negocio, no características de software. El software es el medio por el cual nosotros le entregamos valor de negocio.

Tipos de requerimientos

- **De negocio**
  Son los de más alto nivel, nos referimos a cuáles son los requerimientos en términos de la visión de negocio.
- **De usuario**
  Tienen que ver concretamente con los requerimientos de usuario final.
- **Funcionales**
  Podríamos relacionarlo 1 a 1con los casos de uso en la gestión tradicional.
- **No funcionales**
  Suelen ser los más ocultos, que el cliente no tiene claro y debo especificarlos. Un requerimiento no funcional puede hacer que el software que estoy desarrollando no sirva.
- **De implementación**
  Se suele decir que están dentro de los no funcionales, pero tienen que ver con cuestiones de restricción o implementación específicos.

Se busca entender que le da valor al negocio, cuáles son los requerimientos de negocio y a raíz de estos construir una solución que pueda entregarse al cliente, priorizando lo que le da valor al mismo y a partir de entregas continuas.

Estimaciones de Software

Estimar es predecir en un momento donde no hay gran conocimiento, donde hay incertidumbre. Cuanto más al inicio de un proyecto se estima, mayor es la incertidumbre y mientras se avanza en el desarrollo del mismo, la incertidumbre va disminuyendo, por lo que se ajustan las estimaciones para que sean más precisas.

Estimar no es planear y no necesariamente el plan tiene que ser lo mismo que lo estimado. Si bien la estimación sirve como base para planificar, los planes no tienen que seguirla, las estimaciones no son compromisos. Se debe tener en cuenta que entre mayor diferencia haya entre lo estimado y lo planeado, mayor riesgo va a haber.

Existen distintas formas o técnicas de estimación que ayudan a disminuir la incertidumbre. Estas técnicas no eliminan como tal la incertidumbre, sin embargo, ayudan a estar en un contexto medianamente esperable. Cada técnica tiene ventajas y desventajas, por lo que es importante seleccionar la adecuada en cada proyecto.

- **Tamaño del Software**

  Que tan grande va a ser el trabajo que se tiene que hacer. Para esto se cuentan los siguientes aspectos:

  - ❖ Funcionalidades
  - ❖ Requerimientos
  - ❖ Cantidad de US
  - ❖ Cantidad de CU

- **Esfuerzo**

  Cantidad de horas lineales que se necesitan para construir el software, no se incluyen que personas lo hacen, ni calendario ni se especifican si las actividades son paralelas o secuenciales. Solo se estima cuantas horas de desarrollo se necesitarán.

- **Calendario**

  Calendarizar el esfuerzo y proponer una fecha de entrega.

Técnicas de Estimación

- **Métodos basados en la experiencia**

Se basa en la experiencia pasada en proyectos similares para determinar la estimación de esfuerzo y tiempo requerido para un nuevo proyecto.

- ❖ **Datos históricos**

  Implica comparar un proyecto nuevo con uno anterior que permita estimar el proyecto. Para las estimaciones, se necesitan como entrada el tamaño, esfuerzo, tiempo y defectos. Esta información debe estar completa y estructurada.

- ❖ **Juicio Experto**

  Técnica basada en la experiencia y conocimiento de expertos en la materia para determinar la estimación de esfuerzo y tiempo requerido. A estos se les detalla una descripción del proyecto y se les solicita que estimen basándose en su experiencia y conocimiento.

  - ➢ **Puro**

    Un experto estudia las especificaciones y hace su estimación. Se basa fundamentalmente en sus conocimientos. Si este desaparece, la empresa deja de estimar.

  - ➢ **Delphi**

    Un grupo de personas anónimas son informadas y tratan de adivinar lo que costara el desarrollo en esfuerzo y duración. Esto se envía a expertos para que las revisen y ajusten en rondas sucesivas. Requiere más tiempo y recursos.

- ❖ **Analogía**

  También se basa en datos históricos, pero se seleccionan específicamente aquellos proyectos que son similares al proyecto actual para realizar la comparación y estimación.

- **Métodos basados exclusivamente en recursos**

Se basa en la cantidad y el tipo de recursos necesarios para llevar a cabo un proyecto. La estimación se realiza en función de la disponibilidad y costo de los recursos necesarios.

- **Métodos basados exclusivamente en el mercado**

Se basa en el costo de proyectos similares en el mercado para determinar la estimación de esfuerzo y tiempo requerido para un nuevo proyecto. Se utiliza comúnmente en proyectos de desarrollo de software para clientes externos.

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

Se basa en la identificación y descomposición de los componentes de software y del proceso de desarrollo para estimar el esfuerzo y tiempo requerido para cada componente. Finalmente, se suman las estimaciones para obtener una estimación total del proyecto.

- **Métodos algorítmicos**

Se basa en modelos matemáticos y estadísticos para determinar la estimación de esfuerzo y tiempo requerido para un proyecto. Estos modelos pueden tener en cuenta diferentes variables como el tamaño del software, la complejidad, la productividad del equipo, etc.

Estimación Ágil

- Si las estimaciones se utilizan como compromisos son muy peligrosas y perjudiciales para cualquier organización.
- Lo más beneficioso en las estimaciones es el "proceso de hacerlas".
- La estimación podría servir como una gran respuesta temprana sobre si el trabajo planificado es factible o no.
- La estimación puede servir como una gran protección para el equipo.

En estimaciones agiles, se trabaja con ***features*** o ***stories***, las cuales son estimadas usando una medida de tamaño relativo como $\boldsymbol{Story\ Points}$ $(\boldsymbol{SP})$. Estos son una ponderación que se le da a la US cuyo peso es la combinación de su **incertidumbre**, **esfuerzo** y **complejidad**.

Se trata de **medidas o estimaciones relativas** ya que las define un equipo y no son comparables entre distintos equipos. Debido a esto, **no son absolutas** y, a su vez, tampoco están basadas en el tiempo. Al ser estimaciones relativas, significa que se realizan a través de la comparación, en este caso, se hace con una US canónica que se utilice como base para definir los SP.

Para poder estimar una US se debe estimar su **tamaño**, su **tiempo** y el **esfuerzo necesario**. El tamaño es una medida de la cantidad de trabajo necesario para

producirla o completarla, y a su vez, indica cuan compleja y grande es. Las estimaciones basadas en el tiempo son más propensas a errores debido a factores externos como las habilidades de las personas, el conocimiento del dominio, la experiencia, etc. También, no se debe confundir, tamaño no es esfuerzo, este tiene que ver con las horas lineales.

- **Velocidad / *Velocity***

  Medida del progreso de un equipo. Se calcula a partir de la suma de los SP asignados a cada US que el equipo completo al 100% durante la iteración que se está midiendo. Esta medida, sirve para corregir errores de estimaciones, gracias a su gran cantidad de información.

  A partir de esta métrica también podemos derivar la duración de un proyecto. Para ello se toma el número total de SP de las US del proyecto entero y se lo divide por la velocidad del equipo.

4 – Historias de Usuario

Técnica para capturar requerimientos. No son especificaciones detalladas de requerimientos, sino que expresan la intención de hacer algo. En general no tienen demasiado detalle al principio del proyecto y son elaboradas evitando especificaciones anticipadas. Las US tienen poco o nulo mantenimiento, y pueden descartarse despues de la implementación. Junto con el código, sirven de entrada a la documentación que se desarrolla incrementalmente despues.

No es tan importante lo que se escribe como historia, sino todo lo que se habla acerca de esa historia.

- **Conversación**

  Es la parte más importante que no queda escrita en ningún lugar. Es la parte **no visible**. Todo lo que se define en la historia ayuda a dar limites o a establecer algunos lineamientos a partir de esa conversación. Es el momento en el cual se dialoga con el PO, o quien conoce esta interfaz en el equipo de desarrollo, para captar las necesidades del problema.

  En esto se obtienen todos los detalles para que el equipo pueda trabajar esa historia.

- **Confirmación**

  Condiciones que se tienen que dar para dar por satisfecha la US. Esta dentro de la carta.

- **Carta o Tarjeta**

  Es la parte visible de la US. Es donde se escribe o expresa algo que indique cual es la historia de usuario, de que se trata y su valor de negocio.

Criterios de Aceptación

Definen límites a la US. Ayudan a que el PO responda los criterios que necesita para que la US provea valor al negocio y también a que el equipo tenga una visión compartida de la US.

Son las consideraciones que define nuestro usuario a la hora de hablarnos de como imagina él esa funcionalidad. No van detalles dentro de los criterios de aceptación.

Pruebas de Aceptación o de Usuario

Complementan la historia, y también expresan detalles de la conversación. Dan la confirmación de que, si todas las pruebas pasan, la US implementada hace lo que el PO espera.

Definición de Listo o *READY*

Permite definir que tiene que cumplir la US para asegurarnos de que esta lista para ser implementada. Es una definición acordada por el equipo, en base a la cual se arma un checklist a partir del cual se determina si una US cumple con los aspectos acordados. Si cumple con todos, puede ser incluida en un sprint, pero si no cumple con alguno de los criterios, tengo que trabajar más en esa US para poder incluirla.

Modelo *INVEST*

Nos ayuda a determinar cuándo una US está lista.

- ***Independiente***

  Que no dependa de otra US, por lo que puede ser implementada en cualquier orden. No afecta a otra US moverla dentro del Product Backlog.

- ***Negociable***

  Se negocia el ¿Qué? No el ¿Cómo?

- ***Valuable***

  Debe tener valor concreto para el cliente.

- ***Estimable***

  Tengo que poder definir cuanto esfuerzo me va a llevar hacer la US, para ayudar al cliente a armar un ranking basado en costos.

- ***Small***

  La US tiene que entrar en un sprint.

- ***Testeable***

  Tengo que poder demostrar que fueron implementadas.

Definición de Hecho o $DONE$

Es una definición propia del equipo, que se valida con checklist donde se especifican todas las características que debe tener una US para considerarse decentemente terminada y preséntasela al PO. Debe tener las pruebas unitarias y de aceptación en verde, tiene que haber pasado el ambiente de prueba.

Niveles de Abstracción

- **Épicas**

  Historias de usuario muy grandes, que en principio son asi de grandes porque están en un lugar de la pila donde todavia no fueron detalladas. Como todavia no se deben implementar, se dejan plasmadas como una gran US sabiendo que en algún momento se va a tener que disolver, detallarla y hacerla más pequeña.

- **Temas**

  Podrían ser más grandes y más abstracto que una épica. Es un conjunto de historias de usuario relacionadas que se definen en función de un título para recordar que todo lo que está incluido en ese tema, se va a tener que tratar en algún momento.

Spike

Tipo especial de US que sirve para quitar riesgo o incertidumbre de otro requerimiento, de otra US o de alguna faceta del proyecto que nosotros queremos investigar. Son creadas específicamente para esto.

- ***Spike Tecnica***

  Utilizadas para investigar enfoques técnicos en el dominio de la solución. Cualquier situación en la que el equipo necesite una comprensión más fiable sobre alguna tecnología a aplicar antes de comprometerse a desarrollar una nueva funcionalidad en un tiempo fijo.

- ***Spike Funcional***

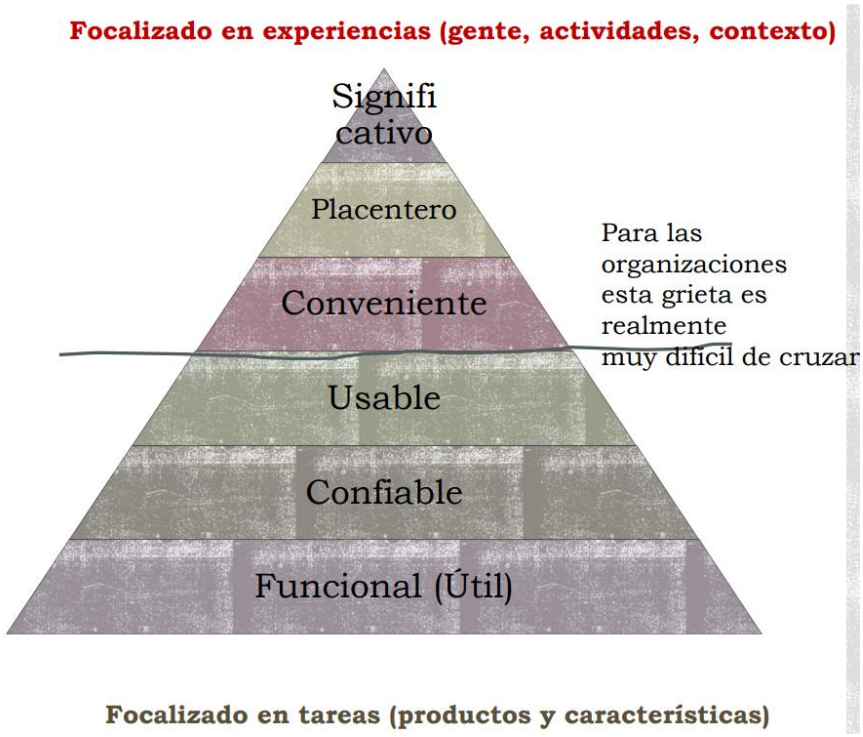
  Utilizadas cuando hay cierta incertidumbre respecto de cómo el usuario interactuara con el sistema. Usualmente es mejor evaluarlas con prototipos para obtener retroalimentación de los usuarios o involucrados.

Normalmente se trabajan siempre un sprint anterior al sprint donde se decide implementar la US que tiene incertidumbre. Los $spikes$ deben ser demostrables, estimables y aceptables, cumpliendo el criterio de $READY$. El $spike$ es una

excepción, no siempre se debe aplicar, es la ultima opción. Antes de implementarla, se gestiona todo dentro de la misma US.

5 – Gestión de Producto

Evolución de los productos de Software



En la **base**, se encuentran aquellas funcionalidades que deben existir para que el software funcione para el propósito esperado, que sean confiables y que de alguna manera tengan cierto lineamiento en experiencia de usuario que permita lograr la funcionalidad esperada. Estos tres aspectos, son lo esperado cuando se construye o se usa un producto de software.

En el **tope**, se encuentran aspectos relacionados con otras cuestiones de la visión del producto que son difíciles de alcanzar. Se refiere a un producto conveniente, placentero y significativo.

Normalmente, los productos se quedan en la base y hasta ahí llegan. El desafío se encuentra en cómo lograr construir un producto de software donde se desecha el desperdicio y se abarcan los aspectos perseguidos cuando se crea el producto de software.

Por todo esto, se busca desarrollar el mínimo producto o mínima característica para saber si el producto cubre las expectativas del usuario y del desarrollador. Para ello

se plantea una hipótesis con un mínimo desarrollo, un mínimo esfuerzo, y se válida para hacer que el desperdicio tienda a 0. La idea es no gastar esfuerzo en algo que no agrega valor al producto. Para esto, se utiliza la técnica **UVP** ($User\ Value\ Proposition$ o Propuesta de Valor para el Usuario), la cual se centra en comprender las necesidades, deseos y problemas de los usuarios y en crear soluciones que satisfagan esas necesidades.

MVP ($Minimun\ Viable\ Product$ – Producto Mínimo Viable)

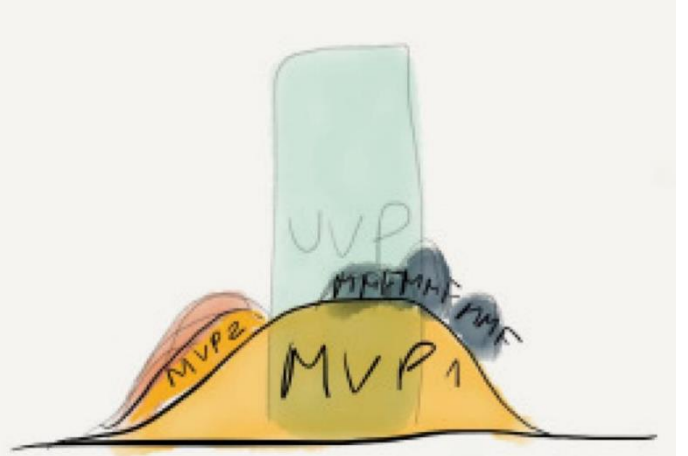
La idea es trabajar con lo mínimo del producto que se necesita. Para que los usuarios lo puedan evaluar y puedan decir si el producto les atrae o no. El objetivo es **comprobar que la hipótesis del producto a construir funciona**, o si se deben hacer cambios.

Si con cada cliente se recibe un feedback distinto, significa que no esta tan claro el mercado del producto y que se debe seguir trabajando sobre la visión del producto.

También, puede suceder que, en algún punto, la hipótesis planteada tenga algunas variaciones. En este punto, la hipótesis puede moverse o modificarse según la exigencia del mercado. Esto significa que el foco no era del todo erróneo, pero algunas características no estan tan bien. Debido a esto, se redefine el MVP en base al nuevo conocimiento, creando un MVP 2, el cual debe volver a hacer el circuito de retroalimentación.

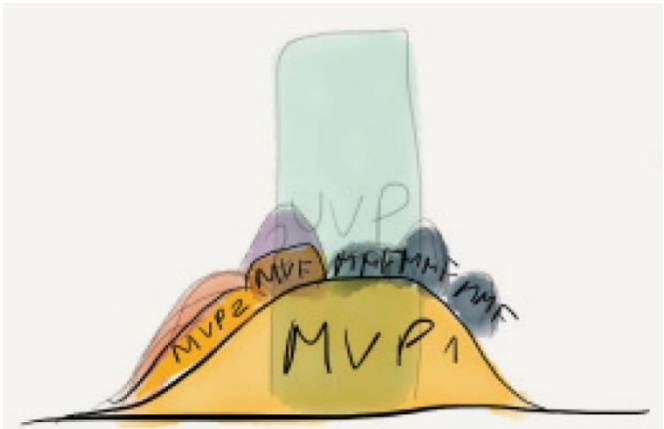
MMF ($Minimum\ Marketable\ Feature$ – Caracteristica Mínima Comerciable)

Se enfoca en definir los elementos mínimos que deben estar presentes en un producto o servicio para que sea comerciable. Se define cual es la pieza mínima que se tiene que construir y comercializar para que el producto sea rentable. El objetivo es asegurarse de que este primer MMF sirve o no. Con la diferencia de que es una $Feature$, que es mucho mas pequeña que el MVP. Es lo mínimo que se puede sacar a la venta.

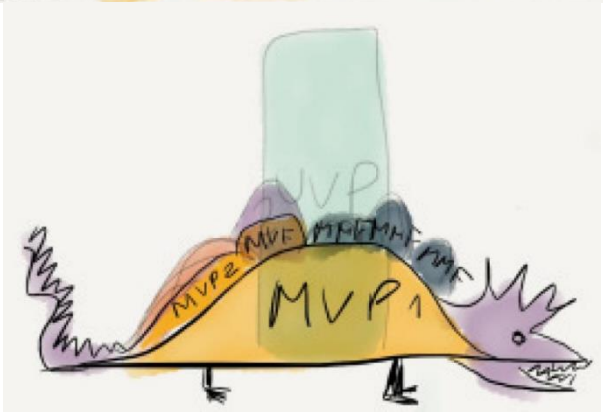


MVF ($\boldsymbol{Minimum\ Viable\ Feature}$ – Caracteristica Mínima Viable)

Se desea validar cual es la caracteristica que es la que hace la diferencia y que va a hacer en primera medida que compren el producto. Es una $Feature$ la cual se busca que de valor agregado por si misma, ahora la hipotesis se centra en una unica caracteristica. Si la MVF resulta exitosa, se pueden desarrollar mas MMF en esta área para tomar ventaja.



Si se sigue trabajando en el contexto de validar las hipotesis y ver como se trabaja con cada caracteristica, 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 que se esta enfocando.



MRF ($\boldsymbol{Minimum\ Realese\ Feature}$ – Característica Minima del Release)

Se trata del realese del producto que tiene el conjunto de caracteristicas mas pequeño posible. El Incremento mas pequeño que ofrece un valor nuevo a los usuarios y satisface sus necesidades actuales.

MMP = MMR1 (Pueden salir mas release despues, que mejoren el producto inicial). Al MRF se lo obtiene en varias iteraciones hasta que hay una cierta cantidad de caracteristicas a lanzar.

Relaciones Concretas

- **MVP**
  - ❖ Versión de un **nuevo producto** creado con el **menor esfuerzo posible**.
  - ❖ Dirigido a un **subconjunto de clientes potenciales**.
  - ❖ Utilizado para obtener **aprendizaje validado**.
  - ❖ Mas cercano a los **prototipos que a una version real funcionando de un producto**.
- **MMF**
  - ❖ Es la **pieza mas pequeña de funcionalidad** que puede ser liberada.

- ❖ Tiene valor tanto para la organización como para los usuarios.
- ❖ Es parte de un MMR o MMP

- **MMP**
  - ❖ Primer release de un MMR dirigido a **primeros usuarios**.
  - ❖ Focalizado en caracteristicas clave que satisfarán a este grupo clave.
- **MMR**
  - ❖ Release de un producto que tiene el conjunto de caracteristicas mas pequeño posible.
  - ❖ El incremento mas pequeño que ofrece un valor nuevo a los usuarios y satisface sus necesidades actuales.
  - ❖ **MMP = MMR1**

6 – SCRUM

Es un marco de trabajo liviano que ayuda alas personas, equipos y organizaciones a generar valor a traves de soluciones adaptativas para problemas complejos.

Valores

El éxito depende de que las personas se vuelvan mas competentes en vivir cinco valores: **Compromiso, Foco, Franqueza, Respeto** y **Coraje**.

El $Scrum\ Team$ se compromete a lograr sus objetivos y a apoyarse mutuamente. El $Scrum\ team$ y sus interesados son francos sobre el trabajo y los desafios. Los miembros se respetan entre si para ser personas capaces e independientes. Estos valores dan dirección con respecto al trabajo, acciones y comportamiento.

Scrum Team

Es un pequeño equipo de personas, considerado la unidad fundamental de Scrum. El equipo consta de un $Scrum\ Master$, un $Product\ Owner$ y $Developers$. Dentro del equipo no hay subequipos ni jerarquias. Es una unidad cohesionada de profesionales enfocados en un objetivo a la vez, el objetivo del producto.

Los equipos son multifuncionales y autogestionados. Es decir, tienen todas las habilidades necesarias para crear valor y deciden internamente quien hace que, cuando y como.

El equipo es lo suficientemente pequeño para ser agil y lo suficientemente grande para completar un trabajo significativo. Generalmente no mas de 10 personas.

El $Scrum\ Team$ es responsable de todas las actividades relacionadas con el producto, y de crear un $Increment$ valioso y util en cada Sprint.

- ***Developers***

  Las personas que se comprometen a crear cualquier aspecto de un $Increment$ utilizable en cada Sprint. Las habilidades especificas que necesitan suelen ser amplias y variarán según el ambito de trabajo. Sin embargo: siempre son responsables de:

  - ❖ Crear un plan para el sprint, el sprint Backlog
  - ❖ Inculcar calidad al adherirse a una definicion de terminado
  - ❖ Adaptar su plan cada dia hacia el objetivo del sprint
  - ❖ Responsabilisarse mutuamente como profesionales

- ***Product Owner***

  Responsable de maximizar el valor del producto resultante del trabajo del equipo. La forma de hacerlo puede variar entre organizaciones, equipos e individuos. Es responsable de la gestion efectiva del $Product\ Backlog$, lo que incluye:

  - ❖ Desarrollar y comunicar explicitamente el objetivo del producto
  - ❖ Crear y comunicar claramente los elementos del $Product\ Backlog$
  - ❖ Ordenar los elementos del $Product\ Backlog$
  - ❖ Asegurarse de que el $Product\ Backlog$ sea transparente, visible y se entienda

  El $Product\ Owner$ es una persona, no un comité. Puede representar las necesidades de muchos interesados.

- ***Scrum Master***

  Responsable de establecer $Scrum$ como se define en la guia. Lo hace ayudando a todos a comprender la teoria y la practica de scrum, tanto dentro del equipo como de la organización. Responsable de lograr la efectividad del $Scrum\ Team$. Apoyandolo en la mejora de sus practicas. Los $Scrum\ Master$ son lideres que sirven al equipo y a la organización en general.

  Sirven al $Scrum\ Team$ de varias maneras, que incluyen:

  - ❖ Guiar a los miembros del equipo en ser autogestionados y multifuncionales
  - ❖ Ayudar al equipo a enfocarse en crear $Increments$ de alto valor que cumplan con la definición de terminado

- ❖ Procurar la eliminacion de impedimentos para el progreso del equipo
- ❖ Asegurarse de que todos los eventos de $Scrum$ se lleven a cabo y sean positivos, productivos y se mantengan dentro de los limites de tiempo recomendados.

Sirve al $Product\ Owner$ de varias maneras:

- ❖ Ayudar a encontrar tecnicas para una definicion efectiva de objetivos del producto y la gestion del $Product\ Backlog$
- ❖ Ayudar al equipo a comprender las necesidades de tener elementos del $Product\ Backlog$ claros y concisos
- ❖ Ayudar a establecer una planificacion empirica de productos para un entorno complejo
- ❖ Facilitar la colaboracion de los interesados según se solicite o necesite

Sirve a la organización de varias maneras:

- ❖ Liderar, capacitar y guiar a la organización en su adopcion de $Scrum$
- ❖ Planificar y asesorar implementaciones de $Scrum$ dentro de la organización
- ❖ Ayudar a los empleados y los interesados a comprender y aplicar un enfoque empirico para el trabajo complejo
- ❖ Eliminar las barreras entre los interesados y los equipos

Eventos

- $\boldsymbol{Sprint}$

Son el corazon de $Scrum$, donde las ideas se convierten en valor. Son eventos de duracion fija de un mes o menos para crear consistencia. Todo el resto de eventos ocurren dentro de los $Sprints$.

- ❖ No se realizan cambios que pongan en peligro el Objetivo del $Sprint$
- ❖ La calidad no disminuye
- ❖ El $Product\ Backlog$ se refina según sea necesario
- ❖ El alcance se puede aclarar y renegociar con el PO a medida que se aprende mas

Existen varias practicas para pronosticar el progreso, como el trabajo pendiente $(burn-down)$, trabajo completado $(burn-up)$ o flujos acumulativos $(acumulative\ flows)$.

Un $Sprint$ podria cancelarse si el Objetivo del mismo se vuelve obsoleto. Solo el PO tiene la autoridad para hacerlo.

- ***Sprint Planning***

Inicia el *Sprint* al establecer el trabajo que se realizará para el mismo. El equipo crea este plan resultante mediante trabajo colaborativo. El PO se asegura que los asistentes esten preparados para discutir los elementos mas importantes del *Product Backlog* y como se relacionan con el Objetivo del producto.

En esta reunion se abordan los siguientes temas

- **¿Por qué es valioso este *Sprint*?**
  Se define un objetivo del sprint
- **¿Qué se puede hacer en este *Sprint*?**
  Se seleccionan elementos para incluirlos en el sprint
- **¿Cómo se realizará el trabajo elegido?**
  Por cada elemento seleccionado, se planifica el trabajo necesario para crear un *Increment*

El objetivo del sprint, los elementos del *Product Backlog* seleccionados, mas el plan para entregarlos se denominan juntos ***Sprint Backlog***. La *Sprint Planning* tiene un limite de tiempo de maximo ocho horas para un *Sprint* de un mes, son dos horas por semana que dure.

- ***Daily Scrum***

El proposito es inspeccionar el progreso hacia el objetivo del *Sprint* y adaptar el *Sprint Backlog* según sea necesario, ajustando el trabajo planificado entrante. Es un evento de 15 minutos para los *Developers*. Para reducir la complejidad, se lleva a cabo siempre a la misma hora y lugar, todos los dias habiles.

Las *Daily Scrums* mejoran la comunicación, identificacion de impedimentos, promueven la toma rapida de decisiones y eliminan la necesidad de otras reuniones.

- ***Sprint Review***

Su proposito es inspeccionar el resultado del *Sprint* y determinar futuras adaptaciones. El equipo presenta los resultados de su trabajo a los interesados clave y discute el progreso hacia el objetivo del producto. Se revisa lo que se logro, y lo que cambio en el entorno. Con base a esto, colboran sobre que hacer a continuacion. También se puede ajustar el *Product Backlog*.

Es el penultimo evento del *Sprint* y tiene un limite maximo de cuatro horas para un *Sprint* de un mes, es una hora por semana que dure.

- ***Sprint Retrospective***

Su proposito es planificar formas de aumentar la calidad y efectividad. Se inspecciona como fue el ultimo $Sprint$. Se identifican los supuestos que los llevaron por el mal camino y se exploran sus origenes. El equipo analiza que salio bien, que problemas encontro y como se resolvieron (o no).

Esto concluye el $Sprint$. Tiene un limite maximo de tiempo de tres horas para un $Sprint$ de un mes, son 45 minutos por semana que dure.

Artefactos

- ***Product Backlog***

Lista emergente y ordenada de lo que se necesita para mejorar el producto. Su compromiso es el Objetivo del producto.

- ***Sprint Backlog***

Se compone del objetivo del $Sprint$ (por qué), el conjunto de elementos del $Product\ Backlog$ seleccionados para el $Sprint$ (qué), asi como un plan de accion para entregar el $Increment$ (cómo). Su compromiso es el objetivo del $Sprint$.

- ***Increment***

Es un peldaño concreto hacia el objetivo del producto. Cada uno se suma a todos los $Increments$ anteriores y se verifica minuciosamente, lo que garantiza que todos funcionen juntos. Para proporcionar valor, debe ser utilizable. Su compromiso es la definicion de terminado.

7 – Software Configuration Management – SCM

Actividad de soporte transversal a todo el proyecto que aplica dirección y monitoreo administrativo y tecnico, y cuyo proposito es mantener la integridad del producto a lo largo de todo el ciclo de vida. Sirve de contención para poder construir un producto integro y de calidad.

El proposito de la SCM es también establecer la integridad de los productos, y esto involucra:

- Identificar y Documentar caracteristicas tecnicas y funcionales de **items de configuracion**.
- **Controlar los cambios** de esas caracteristicas identificadas y documentadas.
- **Registrar y reportar estos cambios**.
- Verificar **correspondencia con los requerimientos**.

Tiene aplicación en diferentes disciplinas como:

- Control de calidad de proceso (Gestión del Proyecto)
- Control de calidad de producto (Ingeniería del Producto)
- Prueba de software (Soporte)

Su proposito es resolver **problemas** de diferente indole, a traves del establecimiento y mantenimiento de la integridad del producto de software a lo largo de todo su ciclo de vida. Algunos de estos problemas son:

- Perdida de componentes
- Perdida de cambios
- Regresión de fallas
- Doble mantenimiento
- Supocision de cambios
- Cambios no validados

Integridad

Se dice que se mantiene la integridad de un producto de software cuando:

- **Satisface las necesidades de usuario**
  Hace lo que el usuario espera que haga.

- **Permite rastreabilidad durante todo su ciclo de vida**
  Existen vinculos o conexiones entre los items de configuracion, que permiten analizar en donde impactara un cambio en un item de configuracion. De esta forma, se puede determinar como impactará cada cambio de requerimientos a traves de la trazabilidad. A mayor cantidad de vinculos → mayor informacion → mayor trazabilidad → mayor costo.
- **Satisface criterios de performance y RNF**
- **Cumple con expectativas de costo**
  No solo el cliente y el equipo deben estar satisfechos, sino también el desarrollo del producto debe ser redituable. Mientras el producto existe hay actividad de gestión de configuración, que es responsabilidad de todo el equipo. Sin embargo, existen roles especificos como el rol del Gestor de configuración que tiene tareas adicionales como por ejemplo mantener la herramienta, marcar linea base, etc.

Ítem de Configuración

Artefactos que forman parte del producto o proyecto, y pueden ser almacenados en un repositorio, sin importar su extención/tipo. Pueden sufrir cambios, y se desea conocer su estado y evolución a lo largo del ciclo de vida. Es decir, cualquier aspecto asociado al producto de software que es necesario mantener. Son todos aquellos elementos que componen toda la información producida por parte del proceso de ingeniería de software. Los ítems, dependiendo de su naturaleza, pueden durar lo que dure el ciclo de vida del proyecto, producto o sprint.

Repositorio

Contenedor de ítems de configuración, se encarga de mantener la historia de cada ítem con sus atributos y relaciones, ademas es usado para hacer evaluaciones de impacto de cambios propuestos. Puede ser una o varias bases de datos. Tiene una estructura para mantener el orden y la integridad.

- **Repositorio Centralizado**

  Caracterizado porque un servidor contiene todos los archivos con sus versiones. Los administradores tienen un mayor control sobre el repositorio. Tiene como desventaja que si falla el servidor, todo lo positivo se cae.

- **Repositorio Descentralizado**

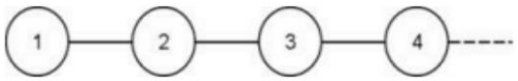
  Caracterizado porque cada cliente tiene una copia exactamente igual del repositorio completo. Tiene como ventaja que se resuelve el problema de los

servidores centralizados, debido a que si el servidor falla solo es cuestion de “copiar y pegar”, ademas posibilita otros workflows no disponibles en el modo centralizado. La desventaja que podemos mencionar es que es mas complicado de llevar un control sobre el mismo.

Versión

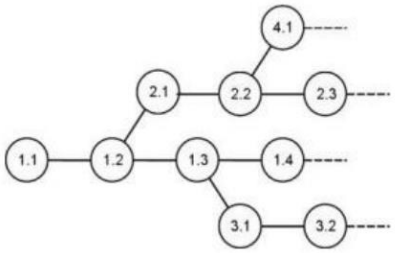
Instancia de un ítem de configuración que difiere de otras instancias de este ítem. Las versiones se identifican univocamente. Cada version de software es una colección de **elementos de configuración** (ECS). La version es un punto particular en el tiempo de ese ítem de configuración.

El **control de versiones** se refiere a la **evolución de un único ítem de configuración** (IC), o de cada IC por separado. La evolución puede representarse gráficamente en forma de grafo.



Variante

Es una versión de un ítem de configuración que evoluciona por separado. Las variantes representan configuraciones alternativas.



Configuración de Software

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

Línea Base

Es un conjunto de IC que han sido construidos y revisados formalmente, de manera que pueden ser tomados como referencia para demostrar que se ha alcanzado cierto nivel de madurez en ellos, y que sirve como base para desarrollos posteriores. Se acuerdan parámetros para determinar que se considera o que deben cumplir los IC para considerarse como línea base.

Si se desea modificar una línea base, existe un **protocolo formal de control de cambios**, que permite definir el procedimiento a seguir para manejar peticiones de cambios, y en caso de aceptarse, acordar nuevamente los parámetros y comunicar los cambios a todo el equipo, para que tomen la nueva línea base como modelo a seguir.

Fundamentalmente es para tener un punto de referencia, pero también sirve para hacer $Rollback$ o saber que se pone en producción. Permite saber cuál era la última situación estable en un momento de tiempo y como fue evolucionando. Permiten retroceder en el tiempo y reproducir el entorno de desarrollo en un momento dado del proyecto.

- **De especificación**

  Son las primeras línea base, dado que no cuentan con código. Podría contener el documento de especificación de requerimiento.

- **Operacionales**

  Contiene una versión del producto cuyo código es ejecutable, han pasado por un control de calidad definido previamente. La primera línea base operacional corresponde con la primera $Release$. Es la línea base de productos que han pasado por un control de calidad definido previamente.

Rama

Conjunto de ítems de configuración con sus correspondientes versiones, que permiten bifurcar el desarrollo de un software, por varios motivos, ya sea experimentación, resolución de errores en el desarrollo o para desarrollar un mismo software para distintas plataformas.

- **Integración de ramas o $merge$**

  Fusiona la configuración de los IC que forman dos ramas con sus correspondientes versiones en cada una de ellas.

Actividades Fundamentales de la Gestión de Configuración

- **Identificación de Ítems de Configuración**

  Implica una identificación univoca para cada ítem, donde en el equipo se definirán **políticas y reglas de nombrado y versionado** para todos ellos. También, se debe definir la **estructura del repositorio**, y la **ubicación de los ítems de configuración dentro de esta estructura**.

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

Los ítems se clasifican en:

- **Ítems de producto**
  Tienen el ciclo de vida más largo, y se mantienen mientras el producto exista.
- **Ítems de proyecto**
  El plan del proyecto, el listado de defectos encontrados, tienen un ciclo de vida de proyecto.
- **Ítems de iteración**
  Un plan de iteración, un $burn - down\ chart$, se mantienen durante una iteración.

Conocer el ciclo de vida de un ítem permite establecer la nomenclatura de este.

- **Control de Cambios**

Tienen su origen en un requerimiento de cambio a uno o varios IC que se encuentran en una línea base. Es un procedimiento formal que involucra diferentes actores y una evaluación del impacto del cambio.

- **Comité de Control de Cambios**

  Este proceso se lleva a cabo por el comité de control de cambios, el cual se reúne para autorizar la creación y cambios sobre la línea base. Forman parte de dicho comité los interesados en evaluar y en enterarse del cambio, decidiendo si lo aceptan o no.

Para realizar un cambio, se siguen ciertas etapas:

1. Se recibe una propuesta de cambio sobre una línea base determinada, no sobre un ítem.
2. El comité realiza un análisis de impacto del cambio para evaluar el esfuerzo técnico, el impacto en la gestión de recursos, los efectos secundarios y el impacto global sobre la funcionalidad y la arquitectura del producto.
3. En caso de que se autorice la propuesta, se genera una orden que define lo que se va a realizar, las restricciones a tener en cuenta y los criterios para revisar y auditar.
4. Luego de realizar el cambio, el comité vuelve a intervenir para aprobar la modificación de la línea base y marcarla como línea base nuevamente.
5. Finalmente se notifica a los interesados los cambios realizados.

- **Auditorias de Configuración**

Son controles autorizados a realizar por el equipo de desarrollo en un momento determinado, donde un auditor externo al equipo analiza las lineas base, las cuales permiten "congelar" y analizar en un momento determinado cual es el estado del software, y si se está cumpliendo todas las pautas que plantea esta disciplina. Esta es la única actividad que no se soporta por la filosofía ágil.

Es objetiva cuando hay **independencia de criterio**. Es independiente cuando no depende jerárquica, funcional ni salarialmente del equipo o elemento. La auditoría se hace mientras el producto se está construyendo y su objetivo es que el producto **tenga calidad e integridad**. Complementa a la revisión técnica y consiste en revisar si se están realizando las tareas como se planificaron y especificaron en el plan de gestión de configuración.

- **Auditoria Física de Configuración (PCA)**
  Asegura que lo que está indicado en cada IC en la línea base o en la actualización se ha alcanzado realmente.
- **Auditoria Funcional de Configuración (FCA)**
  Evaluación independiente de los productos de software, controlando que la funcionalidad y performance reales de cada IC sean consistentes con la especificación de requerimientos.

La auditoría sirve a dos procesos básicos:

- **Validación**

  Asegura que cada IC resuelva el problema apropiado, es decir, lo que el cliente necesita.

- **Verificación**

  Asegura que un producto cumple con los objetivos definidos en la documentación de lineas base. Todas **las funcionalidades son llevadas a cabo con éxito** y los test cases tengan status "ok" o bien consten como "problemas reportados" en la nota de reléase. Es decir, que se cumpla lo definido para cada IC en la documentación de una línea base.

- **Reportes e Informes de Estado**

Provee un mecanismo para mantener un registro de cómo evoluciona el sistema, y donde está ubicado el software, comparado con lo que está ubicado en la línea base. Esto permite mantener a todo el equipo informado sobre la última línea base, que se trabaje en base a su última versión, y no sobre una versión obsoleta.

Plan de Gestión de Configuración de Software

Este plan debe confeccionarse al inicio de un proyecto. Debe definir los documentos que se van a administrar y no debe quedar ningún producto del proceso sin administrarse.

- Reglas de nombrado de los IC
- Herramientas para utilizar en SCM
- Roles e integrantes del Comité de Control de Cambios
- Procedimientos formales de cambios
- Procesos de auditoria
- Estructura del repositorio
- SCM para software externo (opcional)
- Como se hará el control de cambios
- Registros que deben mantenerse
- Tipos de documentos

SCM en ambientes agiles

La gestión de configuración posibilita el seguimiento y la coordinación del desarrollo en lugar de controlar a los desarrolladores. Todos los procesos definidos establecidos en el enfoque tradicional son relativizados en ágil (exceptuando la auditoria que no es propia de un ambiente ágil). Por ejemplo, se elimina el comité de control de cambios.

Entre otras tareas:

- Hace seguimiento y coordina el desarrollo en lugar de controlar a los desarrolladores.
- Responde a los cambios en lugar de tratar de evitarlos.
- La automatización debe ser aplicada donde sea posible. Esto colabora con la entrega frecuente y rápida de software.
- Coordinación y automatización frecuente y rápida.
- Eliminar el desperdicio – No agregar nada más que valor.
- Provee documentación Lean y trazabilidad.
- Provee retroalimentación continua sobre calidad, estabilidad e integridad.
- Tareas de SCM embebidas en las demás tareas requeridas para alcanzar el objetivo del Sprint.