# Resumen Ingeniería y Calidad de Software

## Contenido

# Definición de software

El software no debemos comprenderlo únicamente como código. Es un set de programa y la documentación que lo acompaña.

“Software es conocimiento, información empaquetado a diferentes niveles de abstracción”

## Tipos de software

| | |
|---|---|
| System | Se encarga de controlar y gestionar los recursos del hrd de una compu, así como de proporcionar una interfaz para que los usuarios interactúen |
| Utilitarios | Proporciona utilidades o herramientas adicionales para mejorar la funcionalidad del sop y el rendimiento de una cpu.<br>Principal función es mejorar y optimizar el rendimiento del sop y hrd de una compu |
| Aplicación | Utiliza para realizar tareas específicas en una cpu. Como, procesar texto, crear presentaciones o navegar por internet. Enfocado en satisfacer las necesidades del usuario. |

Con este último trabajamos

## No debemos comparar al software con manufactura:

Hay distintas razones por las cuales es incorrecto decir que software y manufactura son similares. Acá 5 razones:

- **El software no es predecible** como los productos. No necesariamente si siempre hacemos lo mismo obtenemos lo mismo
- **Software es único**. Ninguno se parece al otro, debido a que las necesidades o requerimientos del cliente son diferentes
- **No se gasta**. No se gasta por el tiempo, si debe adaptarse a los nuevos contextos y necesidades pero no se desgastará.
- **No todas las fallas en softw son errores**. El tratamiento de errores no es el mismo en software que en la producción
- **No está gobernado por las leyes de la física**

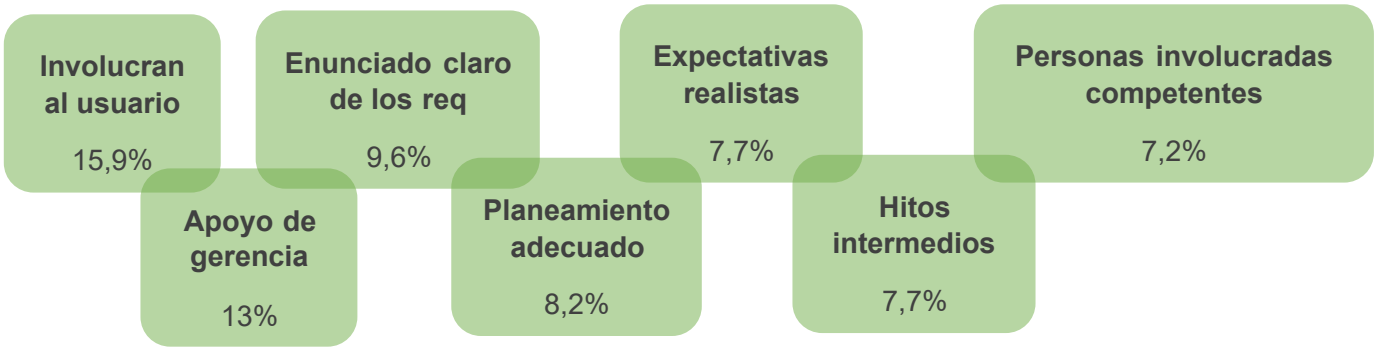
Claramente hay una relación entre estas características las cuales son que se trata la intangibilidad del software.

## Problemas al desarrollar software

- ⇒ Más tiempos y costos que los presupuestados
- ⇒ Que la versión final no satisfaga a las necesidades del cliente
- ⇒ No es fácil escalarlo y adaptarlo. Agregar nueva funcionalidad casi imposible
- ⇒ Mala documentación. Desactualizado e inconsistente
- ⇒ Mala calidad de software. Debemos de dejar de entender a la calidad con la cant de testing

## Claves del éxito

Algunos aspectos que se creen clave se tomarán como premisas para las nuevas metodologías.




Uno de los aspectos claves está relacionado con el hecho de dejar claros los requerimientos e incentivar a que el usuario tenga participación para esclarecerlos. Esto está relacionado con la idea de que los requerimientos van madurando a medida que avanzamos.

La retroalimentación es lo fundamental para poder corregir errores.

## El software no exitoso

El software no exitoso tendrá otras claves que influyen:

- Requerimientos Incompletos 13.1%
- Falta de involucramiento del usuario 12.4%
- Falta de recursos 10.6%
- Expectativas poco realistas 9.3%
- Falta de apoyo de la gerencia 8.7%
- Requerimientos cambiantes 8.1%

El software como conclusión podremos decir que es mucho más amplio que solo código, y su éxito o fracaso no descansa solamente en el hecho de que funcione adecuadamente.

# Proceso de software

El proceso de software se define como:

*"Conjunto de actividades que, a raíz de un conjunto de entradas, produce una salida"*

También lo entendemos como un conjunto estructurado de activades para desarrollar un sistema de software.

Cada actividad variará dependiendo de la organización y el producto final. Entonces son estos los que lo definirán según lo que busquen.

El proceso debe ser explícitamente modelado si se quiere administrar y llevar a cabo


En el concepto de proceso de software, debemos entender que no solo vamos a tener las entradas que nos definirán el alcance del producto.

Sino también los recursos y personas que nos permitirán lograr el producto.

Nuestro objetivo, aunque las definiciones varíen, siempre será obtener un producto de software que satisfaga las necesidades a partir de los inputs.

Como definición más formal nos centraremos en la de CMM

*"Un conjunto de actividades, métodos, prácticas y transformaciones que la gente usa para desarrollar o mantener software y sus productos asociados"*

Acá la clave es la es la estructura. Dentro del proceso tenemos claramente definida las actividades, roles y responsabilidades y herramientas y en base a eso construimos software.

De está definición sacaremos entonces:


Las personas harán uso de las herramientas, procedimientos y métodos y crearán un producto de software.

Dentro del proceso se encontrarán definidas las responsabilidad, actividades y herramientas a utilizar.

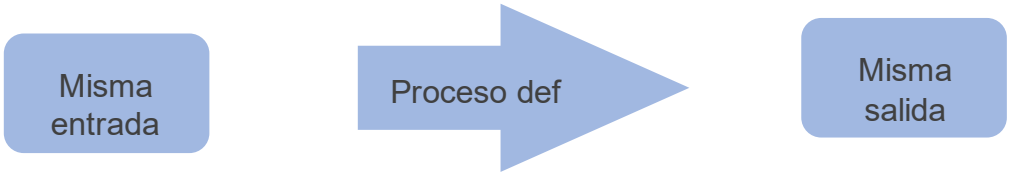
## Empirismo y Definidos

### Definidos

Basado en el modelo industrial. Define que bajo la mismas entradas y realizando las mismas actividades definidas obtendremos siempre el mismo producto final.

*Es difícilmente aplicable al software. Ya que el mismo no es tan definible. No podremos nunca encapsular al software en una premisa que dice que bajo las mismas entradas obtenemos las misma salidas.* ***El soft no es predecible***

Muchas corrientes y procesos confían en estas premisas para llevar a cabo un proyecto. Como el PUD.




### Empíricos

Proviene de la idea del empirismo. Una corriente donde centra el foco en la experiencia. No solo en la técnica, sino en varias de las mismas.

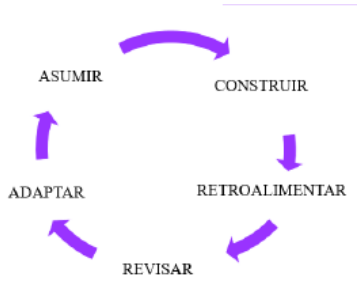
La idea es capitalizar la experiencia y así lograr la mayor calidad.

Vamos a confiar en que las variables no pueden ser definidas o controladas en su totalidad, nos encontramos en un ambiente complejo y cambiante en donde tenemos la necesidad de la adaptación para obtener producto de calidad.

Se basa en la experiencia y en el aprendizaje y tiene una característica particular que la diferencia con los definidos: **Pensar en como yo puedo mejorar el proceso.**

Otra diferencia que está relacionado con el definido es sobre el punto de mejora. Mientras que en el definido iremos directamente a optimizar los procesos ya definidos. En el empírico nos encontramos con varias aristas y puntos para atacar y así mejorarlo, todos estos teniendo como base y punto central el hecho de que la persona es quien capitalizará la experiencia.

En esto es importante el hecho de que exista un ciclo de retroalimentación para mejorarlo.




Empezamos con hipótesis (asumir) y construiremos y en base a algunos puntos medibles tomaremos retroalimentación y obtenemos información. Revisaremos el resultado del proceso y podemos ir adaptando el proceso y seguir construyendo (adaptar)

# Ciclos de vida

El concepto de ciclo de vida debemos de entenderlo como una abstracción. Donde no me dirá lo que tengo que hacer o como hacerlo, pero si unas etapas y el orden.

Tenemos entonces 2 elementos importantes en los ciclos de vida

1. Fases – etapas
2. Orden del 1

Los ciclos de vida son las serie de paso por la cual un proyecto o producto progresa.

## CV del proyecto

El CV de un proyecto se lo entiendo como una representación de un proceso. Grafica una descripción del proceso desde una perspectiva particular.

El cv de un proyecto nos define las fases del proceso y el orden en el cual se llevan a cabo.

## CV de un producto

El cv de un proyecto termina cuando genera un producto. Pero este perdura porque requiere de mantenimiento hasta que se deseche. Además se pueden plantear nuevos proyectos para un mismo producto.

## Tipos de ciclos de vida

### Secuencial

Se basan en ejecutar una etapa tras otra. Sin retorno por lo general. Hay algunas que pueden llegar a devolver información

### Iterativo/Incremental

Los procesos empíricos utilizan este tipo, debido a que estos nos podrán dar la características de adaptación y mejora del proceso (por la retroalimentación)

Permite que en cada iteración apliquemos lo aprendido, mejoremos la experiencia y hagamos otra iteración con lo ganado.

### Recursivo

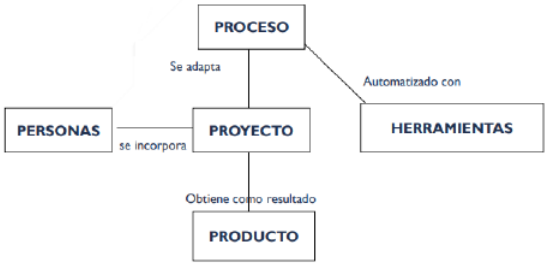
Utilizan para casos particulares, como proyecto de alto riesgo. Basado en la investigación de riesgos y ciclo de vida en B que permite hacer una vuelta en particular de cada etapa haciendo una funcionalidad e ir avanzando en los incrementos.

### Relacionando producto-proceso-cv

Según las características del producto y proyecto elegiremos el ciclo de vida.

El proceso es una implementación del CV que tiene como objetivo la creación de un producto.

# Componentes de proyecto de desarrollo de software




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

## Proceso

Es una plantilla, definición abstracta que se materializará en los proyectos. Donde se adapta las necesidades concretas del mismo.

Se expresará en forma teórico lo que se tiene que hacer para crear software y deberé adaptarlo (definidos) o terminar de definirlos (empíricos)

Es un conjunto estructurado y guiado por un objetivo que es lo que quiero hacer con el proceso. Además, toma como entrada requerimientos -> acts -> producto o servicio de software

## Proyecto

Es la unidad de gestión. Medio por el cual administraré los recursos que necesito y las personas involucradas para obtener un producto o servicio.

Empezará eligiendo la gente y asignando roles.

El proceso dice teóricamente lo que hay que hacer pero es el proyecto el que define que actividades hacer.

El proyecto es el *medio por el cual nosotros obtenemos el producto*. Es una unidad organizativa. El mismo para poder existir necesita de algunas cosas como personas y un método una forma o indicación de cómo hacerse el trabajo para lograr el objetivo que se plantea. Allí entra el proceso porque el mismo da una definición teórica que se debe hacer.

Un proyecto cumple con ciertas características:

- Planificarlo
    - Un proyecto debe ser planificado para poder llegar al éxito. Los proyectos planificados pueden llegar a fracasar pero si no planificamos va a fracasar.
- Objetivo
    - Los proyectos se encuentran orientados a un objetivo o resultado. Guiando el trabajo a realizar para lograr satisfacerlo.
    - Hay que definirlo de manera correcta, sin ambigüedades, claro y alcanzable

        - Importante que sea alcanzable porque nadie querrá estar en un proyecto que se sabe que fracasará
    - El objetivo debe ser único -> cada proyecto tendrá resultado único
- Únicos:
    - No existe dos proyectos iguales. Pueden ser similares pero seguirán siendo únicos.
- Tiempo limitado
    - Tiene un inicio y un fin
    - Se termina cuando alcanzamos objetivo.
- Tareas interrelacionadas basadas en esfuerzos y recursos
    - Dividiremos todo el trabajo en tareas que se encuentran relacionadas entre sí.
    - Además, cada una de estás tareas tiene asociado o asignado recursos.
    - La definición de las tareas la obtenemos del proceso

## Gestión tradicional de un proyecto

Utilizaremos un proyecto con proceso definidos.

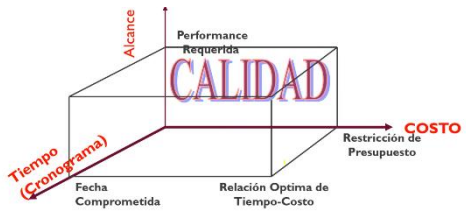
Se basa en ejecutar las actividades definidas para lograr el objetivo.

Lo que buscamos es que el **proyecto cumpla con el objetivo exitosamente**, entonces administraremos todos los recursos, organizaremos el trabajo de la gente afectada y se hará un seguimiento para controlar que las cosas se estén cumpliendo

Cuando hablamos de gestión debemos tener en mente:

- Planificación
- Seguimiento y control

### Triple restricción




**Alcance**: Los requerimientos del cliente, el objetivo, lo que quiero alcanzar.

**Tiempo:** Tiempo que debería llevar completarlo.

**Costo:** Afectado por la cantidad de recursos y personas.

La triple restricción es una de las bases de la gestión tradicional de proyectos. Plantea que un proyecto siempre nos encontraremos con estas 3 restricciones. Debemos de lograr balancear estas tres restricciones que están en constante competencia.

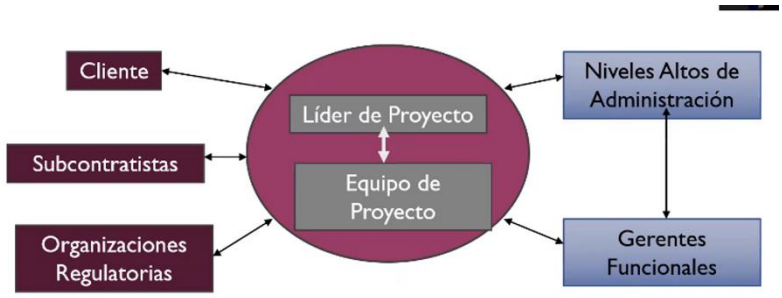
El balanceo de las mismas afecta directamente a la calidad del proyecto. Porque:

*"Proyectos de alta calidad entregan el producto requerido satisfaciendo los objetivos en el tiempo estipulado y con el presupuesto planificado"*

Es importante tener en cuenta que la calidad del producto NUNCA puede ser negociable.

La responsabilidad de balancear dichas restricciones es del **Líder de proyecto**

## Líder de proyecto




Partiendo de los roles que podemos encontrarnos en una organización. Vemos que tenemos un equipo que debe estar relacionado con el líder.

El líder será esta persona que se ocupará de todas las tareas de gestión que hacen que el equipo de proyecto sea guiado hacia el objetivo.

El líder se relacionará con todos los actores presentes en la imagen.

Es el encargado de administrar todos los recursos, organizar el trabajo y hacer el seguimiento de la planificación verificando que todo vaya según lo planeado.

No debería de tener trabajo técnico.

Desde un principio el PM debe saber en términos de negocio o producto de software que debemos de construir y que es lo más importante dentro del proyecto.

Tiene la responsabilidad de:

- Estimar
- Planificar
- Asignarle trabajo
- Hacer seguimiento.

## Equipo de proyecto

Grupo de personas comprometidas en alcanzar el objetivo del proyecto. Se sienten mutuamente responsables por lograrlo. Todos apuntan hacia el objetivo.

Debe haber una complementación en el equipo de proyecto. En el sentido de contar con diversos conocimientos y habilidades, la posibilidad de que trabajen en grupo, que sea pequeño y que tengan sentido de responsabilidad.

## Plan de proyecto

¿Como haremos para lograr el proyecto? Con un plan.

El plan es como una hoja de ruta de viaje que nos define lo que queremos hacer, cuando y como lo haremos.

Desde un momento cero nos encontraremos con dicho plan. Por más que en un principio no se encuentre completamente definido. Se esbozará que hará el proyecto, su alcance, objetivo, recursos a tener, tiempo, roles, funciones y que personas.

Entonces entendemos al plan de proyecto como lo primero que haremos. Realizando un plan detallado que nos va a servir como guía para hacerle frente al proyecto.

Documentamos:

- **Que es lo que hacemos** = Alcance
- **Cuando lo haremos** = Calendario
- **Como lo haremos** = Recursos y decisiones disponibles
- **Quién lo hará** = Asignación de tareas

La idea es que el proyecto sea algo vivo. Que se vaya nutriendo y corrigiendo a medida que el proyecto se lleva a cabo. No podemos plantearlo completamente desde un principio porque siempre existirán variaciones que no se tuvieron en cuenta.

## Planificación de proyectos de software

La gestión de proyecto son actividades de soporte no de construcción.

Incluiremos:

- **Definición del alcance:**
  - Del Proyecto: Todo el trabajo y solo el trabajo que debe hacerse para entregar el producto o servicio de sft con todas las características y funciones.
    - El cumplimiento se mide contra el Plan de proyecto
  - Del Producto: Son todas las características y funciones que pueden incluirse en un producto o servicio.
    - Se mide contra la especificación de requerimientos.
    - Es la sumatoria de todos los req func y no func.

  Entenderemos que la relación entre ambos es que si el alcance de producto crece, el de proyecto también. Por lo que primero se define el producto luego proyecto.

- **Definición de procesos y ciclos de vida:**
  - Cuando inicia el proyecto debe definir que proceso quiero usar, y ciclo de vida utilizaré
  - El proceso de desarrollo será el conjunto de actividades necesarias para construir el producto de soft.
  - Ciclo de vida de que manera ejecutaré esas actividades.
- **Estimación**
  - Cuando vamos a planificar debemos de definir lo que me pide. Esto lo realizaremos mediante estimaciones.
  - Buscamos estimar las características necesarias para la planificación. Siempre lo haremos bajo una línea de incertidumbre.
  - Estimaremos en orden:
    - Tamaño: Definir el producto a construir
    - Esfuerzo: Horas persona lineales
    - Calendario: Determinar que días y que horas se trabajará.
    - Costo: Determinar un presupuesto
    - Recursos críticos: Tanto humanos como físicos necesarios.
  - Daremos rangos en las estimaciones. A medida que avanzo la incertidumbre se achicará y tendremos más certezas.
- **Gestión de riesgos:**
  - Implica poner plata, esfuerzo y horas. Debemos destinar recursos a lo que puede pasar pero no desperdiciar.
  - Algo que podría suceder o no pero que si sucede compromete directamente al éxito del proyecto.

    - La idea es listar aquellos riesgos que tengan mayor probabilidad de ocurrir o tengan un mayor impacto en el sistema.
    - Multiplicaremos dos variables: probabilidad de ocurrencia e impacto y obtendremos **exposición de riesgo**
    - Luego deberemos gestionar. Generar acciones para disminuir impacto o evitar/litigar la ocurrencia.
- **Asignación de recursos**
- **Programación de proyectos**
- **Definición de métricas**
    - Una métrica de software nos permite saber si nuestro proyecto está en línea o se está desviando
    - Medirá la realidad, lo que está pasando y en función de eso sabremos como estamos.
    - Tenemos:
        - Proceso: Para saber en términos de organización como estamos trabajando.
            - Nos permite saber independientemente de un proyecto saber si estamos trabajando bien o mal.
            - Despersonalización de las métricas de proyecto.
        - Proyecto: Permite saber si un proyecto de software en ejecución se está cumpliendo de acuerdo con lo planificado
        - Producto: Miden cuestiones que tienen una relación directa con el producto que estamos construyendo.
    - Algunas métricas básicas
        - Tamaño
        - Esfuerzo
        - Tiempo
        - Defectos
    - Tomare métricas en base a las decisiones sobre el tiempo que va a tardar el proyecto, debo minimizar el desperdicio de las métricas. Hacer foco.
- **Monitoreo y control**
    - Es el líder quien se encarga de trabajar sobre lo definido en el plan y determinar que se cumple. Basará en el plan y en las métricas.
    - *“Un proyecto se atrasa un día a la vez”* Quiere decir que si puedo corregir los errores en el momento adecuado, nada cambia.

## Factores y causas del éxito y fracaso

| Factores para el éxito | Causas del fracaso |
|---|---|
| • Monitoreo y feedback: Generando acciones correctivas<br>• Misión/objetivo claro<br>• Comunicación: En todos sus aspectos con el líder de proyecto y todos los involucrados | • Falta de comunicación<br>• No existe monitoreo<br>• Falta de planificación o pocos datos<br>• No hay detalles en el plan<br>• Mala planif de recursos<br>• Estimaciones sin sustentos en datos históricos<br>• Nadie a cargo<br>• Mala comunicación |

# Filosofía Ágil

El agilismo o filosofía ágil es un movimiento que nace desde los desarrolladores hace 20 años. Cuando un grupo de personas se junto en un hotel en Utah y firmaron un acuerdo. *Manifiesto ágil.*

El manifiesto es un compromiso para trabajar de una determinada manera independientemente de las practicas propias.

Es una evolución cultural, en donde toman parte las experiencias propias

La filosofía define formas de manejarse ante situaciones relacionados con el desarrollo de software. Indep de la practica.

## Historia

Comienza con los principios desarrollos de software, donde todavía no podíamos hablar de una profesión. Cuando la de demanda de más software más complejo crece se empieza a ver la necesidad de la formalización.

Allí aparecen los militares los cuales van a definir estándares formales y muy rigurosos, donde tenían mucha exigencia y mucha documentación. La idea era encuadrar al desarrollo en una especie de ingeniería dura. Pero el problema es que el software no es tangible. En estos casos llega un punto donde cumplir con el proceso era más importante que la calidad del software

Paralelamente se seguía desarrollando software con poco profesionalismo, de manera "hippie" o más bien artesanal.

## Concepto Ágil

Entonces nace ágil con una idea clara:

*"Lograr un equilibrio entre procesos rigurosos y formales y trabajar de manera eficiente y efectiva para un producto de calidad"*

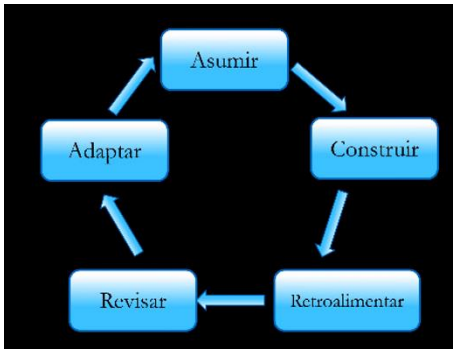
Lo podemos entender como un compromiso útil entre nada de proceso y mucho proceso.

No es una metodología ni un proceso, es una IDEOLOGIA. La cual cuenta con un conjunto definido de principios y valores que nos guían en el desarrollo.

## Manifiesto Ágil

El manifiesto ágil se sustenta en el uso de procesos empíricos[1].

Su sustento en el empirismo hace que esta filosofía solamente funcione en el ciclo de vida *iterativo e incremental* en donde tendremos una retroalimentación con ciclos cortos. Dicho ciclo de aprendizaje (visto en los procesos empiricos) hace que seamos capaces de afrontar los cambios.




La idea es asumir una hipótesis, trabajar en base a ella con la construcción de un MVP para poder presentárselo al cliente y obtener una retroalimentación del mismo. A partir de allí se revisa el feedback y adaptaremos el producto a lo que se pide.

*Nos vamos a basar en la idea de que la incertidumbre y los cambios son inevitables en el desarrollo de software. Por lo que tenemos que ser flexibles y adaptables para satisfacer al cliente.*

### Pilares del empirismo

El empirismo centra todo su desarrollo en 3 pilares:

- Transparencia: Es la comunicación abierta y honesta de la información que es relevante para todo el equipo.
    - Nos permite transformar el conocimiento propio en un conocimiento explicito disponible para el equipo y la organización.
- Adaptación: Capacidad de adaptarse y ajustar el trabajo para hacer frente a desviaciones y problemas detectado en la inspección
- Inspección: Revisión regular y sistemática del progreso del trabajo y los productos con la idea de detectar problemas y desviaciones.

[1] Procesos Empiricos en donde la experiencia del equipo es la base. De allí nace la importancia de tener ciclos cortos de retroalimentación

### Valores del manifiesto

Cuenta con 4 valores. La sintaxis es que los dos son importantes para la izquierda es más importante

#### Individuos e interacciones por sobre procesos y herramientas

Debemos de darle importancia al vinculo, colaboración y comunicación que se establece con las partes interesadas por sobre la adopción de herramientas y procesos a aplicar.

#### Software funcionando por sobre documentación extensiva

Es más importante un software funcionando por sobre una documentación que explique las características del mismo.

- No debemos de entender que no se hace documentación. Hay necesidad siempre de documentación.

Este enfoque plantea que generemos información cuando la necesitemos (NO dice que no lo hagamos) -> Idea relacionada con “lo más importante es el acto de planificar, no el resultado” y con que los procesos de generación de info son imp

Hay que decidir que información debe ser documentada teniendo en cuenta que el conocimiento del producto debe ser transparente. Además, la información podrá perdurar más allá de una iteración.

La idea es que si nosotros no tenemos software funcionando para el cliente toda la documentación no sirve, no le ve valor.

#### Colaboración con el cliente por sobre negociación contractual

Existe una importancia en trabajar con el cliente, entendiéndolo y comprendiendo sus necesidades y requisitos por sobre negociar contratos y acuerdos formales.

Hay un problema que es donde nosotros debemos aplicar el valor que es cuando el cliente desea cambiar algún requerimiento y ya se firmo un contrato. Esto genera muchos problemas en el desarrollo (cambios en la triple restricción).

Debemos de integrar al cliente como parte del equipo, haciendo que colabore lo más posible para poder ir aclarando los requerimientos en el desarrollo.

El problema es: El cliente quiere trabajar con nosotros? *Si -> agile* | *No -> tradicional*

#### Responder al cambio por sobre seguir un plan

Es más valioso la capacidad de respuesta y adaptación a cambios a la de seguir y asegurar el cumplimiento de planes en un ambiente que conocemos es volátil.

La idea es no empezar con algo tan definido. Le dejemos espacio al cliente para que cambie los requerimientos, allí la importancia del valor anterior. Esbocemos una idea y a partir de ella trabajamos juntos en definirla.

**12 principios del manifiesto**

**1. Satisfacción del cliente**

Nuestra prioridad es satisfacer al cliente con la entrega temprana y continua de software con valor

**2. Adaptación al cambio**

Debemos de aceptar que un ambiente volátil los requerimientos cambian. Aceptándolos los usaremos como ventaja competitiva

**3. Entregas frecuentes**

Entreguemos software funcional frecuente, con preferencia a la doc. Muy relacionado con el primer valor.

**4. Trabajo en equipo**

Colaborar con el cliente y los interesados durante todo el software para asegurarnos del valor.

**5. Personas motivadas**

Construimos proyectos entorno a individuos motivados. Hay que darles el entorno y el apoyo y la confianza que necesitan

**6. Comunicación directa**

Método más eficiente es la comunicación cara a cara. La riqueza y entendimiento crece muchísimo si nos encontramos en un mismo espacio físico.

**7. Software funcionando**

Software funcionando es la medida principal de progreso. Si no se entrega al cliente el equipo no progresa

**8. Continuidad**

Se promueve el desarrollo sostenible. Que el equipo sea capaz de mantener un ritmo constante y sostenible de trabajo a lo largo del tiempo. Para tenerlo hay que asegurarnos que el equipo tenga un ritmo de trabajo que asegure entregas en ciclos de tiempos fijos

**9. Excelencia técnica**

La excelencia técnica y buen diseño mejoran la agilidad. La calidad no se negocia, el resto si.

**10. Simplicidad**

El arte de maximizar la cantidad de trabajo no realizado. Buscando simplicidad logramos maximizar la eficiencia y efectividad y reducir los riesgos de errores y problemas.

**11. Equipos auto-organizados**

Las mejores arquitecturas, requisitos y diseños emergen de equipos autoorganizados. Con la libertad y confianza suficiente para diseñar soluciones y tomar decisiones

**12. Mejora continua**

El equipo reflexiona sobre como ser más eficaz para a continuación, ajustar y perfeccionar su comportamiento en los próximos intervalos.

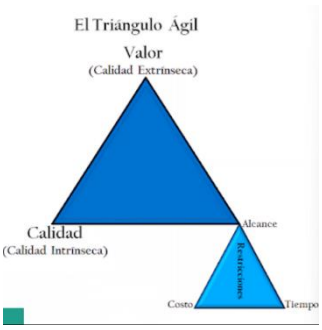
## Triángulo Ágil vs Tradicional

En el triangulo tradicional nosotros vemos el triangulo de hierro. Conocido como tener 3 elementos fundamentales en cualquier proyecto: Tiempo, Alcance y Costo. Este triangulo nos establece o nos dice que los 3 están relacionados entre si y si muevo uno afecto a los otros dos.

Hay un claro enfoque en el alcance, puesto es este el valor que dejaremos fijos, y variamos los costos y tiempos.

En las metodologías ágil nosotros no haremos foco en el alcance, dejaremos fijo el tiempo y los costos puesto a que tendremos iteraciones con duración de tiempo fija. Nos guiamos por la premisa que dado a este tiempo y recursos que le puedo entregar al cliente que tenga valor.

Allí nace la idea de darle más importancia al valor por sobre el resto. Le daremos importancia al triangulo de hierro, si, pero centremonos en el valor que tiene intrisicamente el producto y a las caracteristicas de calidad.

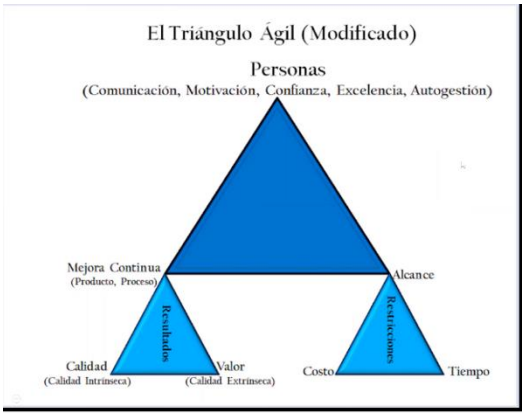



Llegando a nuestro proximo triangulo donde nos centramos en las personas porque son estás las más importante de nuestro proyecto. Según el empirismo son ellas las que van a capitalizar la experiencia que tienen y la del equipo para lograr una mayor calidad en el producto de software.

Las personas son las más importantes porque serán las que resuelvan el éxito o fracaso del producto.

Buscamos la mejora continua de todo producto y proceso y tengamos en cuenta el alcance y las variables asociadas.

Tenemos que hacer foco en las personas para nosotros poder entregar un producto de calidad.



# Requerimientos ágiles

El enfoque ágil no solo plantea una gestión diferente del proyecto, sino también una definición de requerimientos distinta.

Nos encontraremos con 3 enfoques:

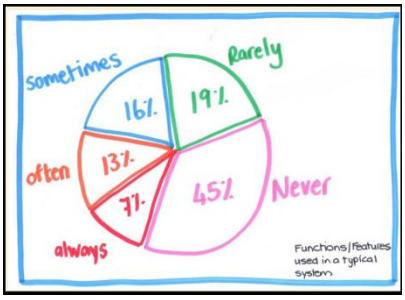
Valor: Usaremos valor para construir el producto correcto. Es decir, nos preguntaremos a quien queremos o vamos a ayudar con la creación de nuestro producto. La idea central de la construcción de software está centrada en el hecho de construir valor de negocio.

Sólo lo suficiente: *Just in time.* La idea es saber que no tendremos nunca todos los requerimientos definidos en un principio. Dado al contexto de software el cual es volátil. Definamos los requerimientos y características a medida que avanzamos en el software.

En un principio encontremos lo mínimo y necesario y trabajemos a partir de allí para empezar a crear retroalimentación.

Usar historias y modelos para mostrar que construir: Los requerimientos serán expresados en forma de historias de usuarios, que son breves descripciones de los requisitos del cliente. Definiremos el alcance con estas historias de cada iteración del producto y se priorizan. La etapa de licitación de requerimientos se transforma en una etapa de trabajo continua e integra con el negocio (PO)

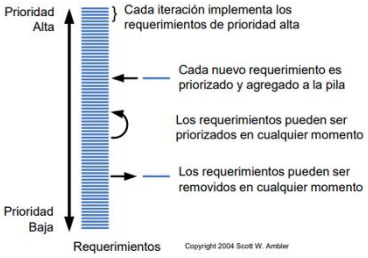
La necesidad de cambiar el enfoque a solamente lo suficiente surge del problema que solamente una pequeña porción 7% de las funcionalidades que construimos se utilizan siempre y una gran porción del 45% no se utilizan nunca lo que genera un desperdicio.




Con la gestión ágil se busca contrarrestar esto, priorizando solo aquellas funcionalidades que aportan valor al cliente.

Se define un nuevo rol **Product Owner** el cual es el representante del negocio dentro del equipo. Entonces será este el que conozco al negocio en su totalidad, pueda identificar las necesidades y prioridades del negocio y por ende será él el encargado de la priorización de US

Crearemos entonces un **producto backlog ,**una lista priorizada y ordenada de requerimientos. La administra y le pertenece al PO.




El hecho de que sea un lista o cola nos obliga a tener que acomodar las US y priorizarlas.

Es una diferencia grande con lo tradicional puesto a que no se prioriza, y mucho menos se deja una parte del negocio participar en esta priorización.

La gestión ágil de requerimientos nos apunta a trabajar y a poner el esfuerzo en lo que nos da valor, y a trabajarlo en el momento adecuado.

Debemos trabajar primero con los que se encuentran arriba de todo en la lista. Son los que generarán mayor valor al cliente. Es importante denotar que lo cambiante se aplica acá también, se puede removar req, repriorizarlos o agregar nuevos.

Siempre trabajamos con los primeros en su mayor nivel de detalle, los que se encuentran al fondo tiene la posibilidad de cambiarse sin que a nosotros nos afecte. Cuando lleguemos a ese nivel de prioridad trabajaremos con ellos.

## Diferencia entre ágil y tradicional

| | 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. Descubrimiento progresivo |
| **Participación** | Sponsors, stakeholder de mayor poder e interés. | Colaborativo con stakeholders de mayor interés (clientes, usuarios finales) |
| **Equipo** | Analista de negocios, Project Manager y Áreas de Proceso. | Equipo multidisciplinario. |
| **Herramientas** | Entrevistas, observación y formularios | Principalmente prototipado. Técnicas de facilitación para descubrir. |
| **Documentación** | Alto nivel de detalle.<br>Matriz de Rastreabilidad para los requerimientos. | Historias de Usuario.<br>Mapeo de historias (Story Mapping) |
| **Productos** | Definidos en alcance | Identificados progresivamente |
| **Procesos** | Estables, adversos al cambio. | Incertidumbre, abierto al cambio |

## Tipos de requerimientos

| Tipos | |
|---|---|
| De negocio | Son los de más alto nivel, en términos de visión del negocio |
| De usuario | Tiene que ver con los requerimientos de usuario final<br>Las US son requerimientos de usuario alineado a uno de negocio. |
| Funcionales | Relacionarlos 1 a 1 con los CU en gestión tradicional |
| No funcionales | Más ocultos, el cliente no los tiene claro pero puede hacer que el software no sirva |
| De implementación | Cuestiones de restricción o implementación específicos |

Se busca entender que le da valor al negocio, cuales son los requerimientos de negocio y construir una solución que pueda entregarse al cliente. Priorizando lo que le da valor al mismo y con entregas continuas.

La gestión de requerimientos agiles implica trabajar junto a técnicos y no técnicos. Buscando entender las necesidades y el negocio para construir un software que de valor y trabajar con los usuarios para encontrar la mejor forma de lograrlo (donde entra el producto backlog).

Entregamos las características, obtenemos feedback, con ello mejoramos el backlog.

Algunos conceptos para cerrar ideas:

- Los cambios van a existir todo el tiempo constantemente.
- La comunicación cara a cara es la mejor forma
- Lo importante es entregar una solución de valor
    - Nos centramos siempre en entregar un software. Pero el cliente ve a este como una herramienta.
    - Tenemos que entregar valor de negocio NO caract de software. Si lo segundo es herramienta para lo primero
- Debemos tener una mirada de todos los involucrados
- El usuario dice lo que quiere cuando recibe lo que pidió
- No hay técnicas ni herramientas que sirvan para todo.
- Diferenciamos el dominio del problema con el de la solución
    - Requerimiento de negocio y usuarios -> Dom Problema
        - US es dominio de problema
    - Requerimientos de software -> Dom de la solución.

## Principios ágiles relacionados

Se enuncian lo más relacionados. Pero todos los principios se encuentran presentes.

1. La prioridad está en satisfacer al cliente a través de entrega de producto temprana y continua
2. Recibir cambios hasta en las etapas finales

4. Técnicos y no técnicos trabajando conjuntamente

6. Medio de comunicación por excelencia es el cara a cara

11. Las mejores arquitecturas y diseños emergen del equipo auto organizados.

# USER STORIES

Técnica para capturar requerimiento de usuarios. El nombre proviene a que esta centrada en la creación historias. Además tienen un foco claro al negocio

La parte más difícil de construir software está dada por encontrar los requerimientos. A menudo estos son pocos claros y solamente se descubren los errores después de trabajarlos y mostrárselo al cliente.

La US no son especificación detalladas de requerimientos. Vamos a expresar la intención de hacer algo. No tiene detalles al principio y son elaboradas evitando especificaciones anticipadas.

*"Es una descripción corta sobre una necesidad que tiene un usuario respecto del producto de software"*

## Las 3 "C" de la US

### Conversación

La conversación es la parte más importante de la user. No quedará escrita en ningún lugar, pero todo lo que surja de la conversación nos ayudará a delimitar la US. En la misma obtendremos todos los detalles necesarios para construir la US

### Confirmación

La confirmación la conocemos también como pruebas de aceptación. Que serán las condiciones para poder dar como satisfecha esa US.

### Card

Es el medio físico donde se presenta tanto la comunicación, todo lo que obtuvimos de ella, como la confirmación, las pruebas de aceptación. Es la parte visible de la US.

## Nomenclatura de las US

---

**Como** <ROL> **yo quiero** <ACTIVIDAD> **para** <VALOR DE NEGOCIO>

---

La idea es contestar estás 3 preguntas: WHO, WHAT, FOR WHAT.

- **Rol**: Representa quien está realizando la acción o quien recibe valor
- **Actividad**: Acción a realizar en el sistema
- **Valor de negocio**: Nos dice el porque es necesaria la actividad.

El valor de negocio es lo que utilizará el PO para priorizar el backlog.

Se puede identificar una user con una frase verbal que identifique de que se trata y que va a tener la tarjeta

Las US son multipropósito porque modelan o representan distintas cosas, me plantea:

- Describe una necesidad del usuario
- Describe el producto
- Item de planificación: Vamos a definirlas y priorizarlas para planificarlo.
- Token para una conversación: Es un recordatorio de que tenemos que hablar con el negocio para especificar características.
- Mecanismo para diferir una conversación: Sirve para definir otras conversaciones a futuro.

## Producto Backlog

El PO priorizará las US en el producto backlog. Esta lista de requerimientos que tiene que cumplir mi proyecto para lograr el objetivo.

## La verticalidad


Las US se encuentran planteada como unas porciones verticales. No describiremos solamente funcionalidad, o haremos solamente implementación de la lógica o haremos solamente BD. Se trata de una unión de todas, un poco de todas.

Lo corto de tal manera que yo pueda ir entregando valor.

## Modelado de roles

Se intenta describir los roles para lograr identificar los que tengamos en una US.

- Tarjeta de rol de usuario: Definimos un perfil de usuario para entender y comprenderlo
- Personas: Descripción particular de una persona con nombre, hobbies, características y gustos. Así entendemos que el software se usa por personas
- Personajes extremos

## Que pasa si no hay PO

Debemos de elegir otro tipo de usuario que pertenezca al negocio para poder seguir trabajando de manera ágil.

## La Confirmación

Para realizar la confirmación y describir las pruebas de aceptación primero debemos de hablar de los criterios

### Criterios de aceptación

Nos definen la US, es información concreta que tiene que servirnos a nosotros para saber si lo que implementamos es correcto o no. Nos ayuda a determinar lo que se necesita para que se provea de valor.

Nos ayuda a que el equipo tenga una visión compartida de la US. No contienen detalles de implementación se definen en términos de negocio.

Ayudan a saber cuando parar de agregar funcionalidad y derivar a las pruebas.

Es importante que sean escritos en términos de negocio, definan una intención y que sean independientes de la implementación.

Se encuentran en la conversación.

### Pruebas de aceptación

Se encuentran al dorso de la tarjeta, complementan la tarjeta.

Se encuentran directamente relacionado con los CA. Deben encontrarse detalladas porque de lo contrario no sirve.

Se deben contemplar en ellas pruebas de fracaso y de éxito.

Se definen de manera colaborativa junto con el equipo y el PO.

Nos da la confirmación de si lo que nosotros estamos haciendo es lo que se espera.

## DoR: Definition of ready

Es una medida de calidad que nos establece que es lo que tiene que tener la US para poder pasar a implementación.

Se define en cada equipo.

Se arma un checklist con las características. Si cumple con todos los aspectos puedo incluirla en una sprint, pero si no cumple, debo seguir trabajando en ella. Que sea incluida en sprint implica que puedo sentarme a implementarla.

Si bien se define en cada equipo hay un acuerdo mínimo que todos las US deben de cumplir el cual es el INVEST.

**Independiente**: Cada US no debe depender de ninguna otra US. Esto es importante porque de esta manera le doy la libertad al PO para que las priorice y se puedan implementar en cualquier orden

Negociable: Tiene que estar escrita en términos de que necesita el usuario, no de como lo vamos a hacer.

Valuable: Debe tener un valor de negocio

Estimable: Tengo que poder definir cuanto esfuerzo me va a llevar realizarla. Tengo que poder asignarle un numero que me permita compararlas con otras users. De otra manera tengo una SPIKE

Small: Debe ser consumidas en una iteración. No hay trabajos a media. Esto depende mucho de la experiencia del trabajo y su duración de iteración

Testeable: Tengo que poder demostrar que la US fue implementada en los términos que el PO quiere.

## DoD definition of done

Además, nos encontramos con esto que es la definición de hecho que lo utilizaremos solamente para determinar si la historia está terminada para presentársela al cliente o PO. Tiene misma forma que DoR.

## Niveles de abstracción

Los requerimientos tengan diferentes niveles de abstracción. Podemos encontrarnos con diferentes abstracciones:

Épicas: Las cuales son US muy grandes que están así porque se encuentran en un lugar de la pila donde todavía no fueron especificadas. Así que la reconozco pero solamente cuando las implemente las detallo.

Temas: Son un conjunto de US que incluso podrían ser hasta más grande que la épica. Están relacionadas entre si bajo un mismo tema.

## SPIKES

Son un tipo especial de US creadas para quitar riesgo e incertidumbre de otro requerimiento, US o alguna faceta del proyecto que queramos investigar.

Son dos tipos: Técnicas o funcionales.

# Estimaciones

El concepto de estimar es el de predecir en un momento donde hay incertidumbre. Y si estamos en el inicio de un proyecto aún más todavía.

La estimación no son precisas. Son una valoración cuantitativa de lo que yo quiero estimar.

No es necesario esperar a tener toda la info para estimar, se puede ir estimando con poca info e ir ajustando a medida que avance.

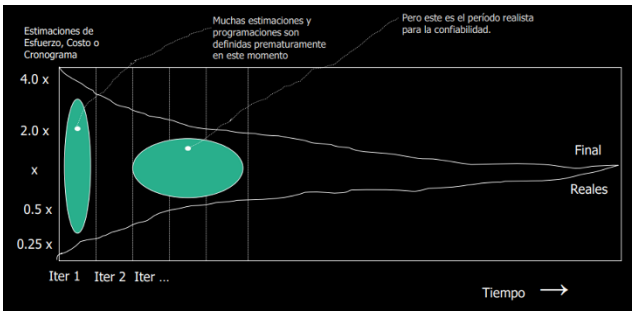
No son compromisos.

Estimar no es planear y no necesariamente nuestro plan tiene que ser lo mismo que lo estimado. Mientras mayor diferencia exista entre planes y estimado mas riesgo habrá

No podemos esperar a tener mucha precisión para estimarla. Debemos trabajarlas desde un principio sabiendo que la información es poca e ir ajustándolas a medida que avanzamos.

La realidad es que en un proyecto se puede llegar a tardar hasta 4 veces más de lo que estimamos. Esto es porque muchas veces se nos obliga a dar estimaciones tempranas donde no tenemos la información necesaria.

Acá viene un poco la idea que plantea LEAN cuando nos difiramos la toma de decisiones hasta el último momento posible.




A medida que voy progresando mi incertidumbre se disminuye y puedo estimar con una mayor precisión.

La foto representa lo que conocemos como conoce de la incertidumbre.

## Que se estima

Los siguientes ítems que se deben de estimar se realizan en orden-

1. Tamaño: **EL QUE** Que tan grande será la el trabajo que debemos de hacer. Utilizamos la técnica de contar:
    a. Requerimientos
    b. Lineas
    c. Cantidad US
    d. Cantidad CU
    e. Funcionalidades.

    En este caso hay un problema por el nivel de incertidumbre. Se pueden utilizar datos históricos de productos similares.

2. Esfuerzo: **EL COMO** – Son la cantidad de horas lineales que voy a necesitar para construir el software. 1 sola persona, sin acoplamiento y de principio a fin.
    a. No incluyo quien lo hará, que días se hará, ni si hay tareas paralelas.
3. Calendario: **EL CUANDO** – Calendarizamos el esfuerzo y propones una fecha de entrega
4. Costo: Una vez terminado todo lo anterior recién ahí puedo estimar los costos, en base al tiempo que me va a llevar, las características del producto, etc.

## Técnicas de estimación

Cada técnica nos ayudará reducir pero no eliminar la incertidumbre, cada una de ellas contará con sus ventajas y desventajas.

Es importante elegir siempre la más adecuada al proyecto y hacer un seguimiento a medida que este avance.

### Métodos basados en la experiencia

Se basan en la experiencia de los equipo de proyecto para determinar la estimación de esfuerzo y tiempo requerido. Tenemos 3 técnicas

- **Datos históricos**: Parto de la idea que la experiencia de otros equipos en proyectos a mi me puede servir. Comparamos un proyecto nuevo con uno viejo.
    - Es importante denotar que datos vamos a tener en cuenta puesto a que notodos se pueden considerar. Elegimos esfuerzo, tamaño, tiempo y defectos.
    - Problema es que los dominios pueden ser muy diferentes.
- **Juicio Experto**: Es el más utilizados de todos. Se basa en la experiencia y conocimiento de expertos en la materia para estimar.

    - Se les pide realizar estimaciones en base a la experiencia propia de cada uno de ellos.
        - Debemos de elegir bien en que áreas son expertos.
    - Es muy subjetivo al experto.
    - Se estructura para darle un numero
        - Usarlo solo en tareas con granularidad aceptable
        - Usar formula (O + 4H + P ) en term de tiempos.
        - Usar una checklist
    - Depender solamente del juicio experto no es bueno porque primero no podemos asegurar que el equipo tenga el mismo nivel de experiencia que el juicio y además hay un índice rotacional grande. No debemos de depender de nadie.
    - Se plantea entonces el WIDEBAND DELPHI
        - En donde un grupo de personas son las expertas y tratan de adivinar lo que costará el desarrollo.
        - Son personas con diferentes expertise
        - De acá nace *pokerestimation*
- **Analogía**: También se basa en datos históricos pero nos centraremos específicamente en aquellos similares y nos fijaremos que estén acorde al proyecto.

**Basados exclusivamente en recursos**

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

**Basados exclusivamente en el mercado**

Se basa en los costos de proyectos similares para determinar el propio

**Basados en los componentes del producto o en el proceso de desarrollo**

Se basa en la identificación y descomposición de los componentes. Estimar para cada uno tiempos y costos y luego sumar todos.

**Métodos algorítmicos**

Basa en modelos matemáticos y estadísticos para determinar la estimación

## Errores que se cometen estimando

Siempre hay errores puesto a que trabajamos en un campo de incertidumbre. En base a esto tenemos algunas más comunes como:

- Actividades omitidas
    - Debemos de estimar tiempo de prog, reuniones, testing, diseño, arquitectura.
- Las técnicas de estimación tienen un margen de error
- Falta de información
    - Asumir cosas cuando en realidad no lo sabemos.
- Cuando no tenemos claro el proceso que vamos a utilizar para trabajar.

# Estimando en ágil

Vamos a seguir una misma línea que lo anterior pero tendremos en cuenta lo siguiente:

- Las estimaciones como compromisos son peligrosas
- Lo más beneficio de estimar es hacerlo
- La estimación nos puede servir como una respuesta para saber si el trabajo es factible o no
- Puede servir como protección para el equipo

En ágil estimaremos con stories.

Buscamos corregir la mirada en contra de re-estimar y los problemas de estimar pronto, estimar buscando precisión, estimaciones de gente fuera del equipo, estimaciones absolutas.

## #NO ESTIMATE

Surge entonces esta necesidad de no estimar en el sentido de que para que lo vamos a hacer si ya sabemos que en el 90% de lo casos va a estar mal.

Sigue la idea de eliminemos el desperdicio y plantea la estimación como uno.

## Foco de estimaciones agiles

El foco lo tiene las personas, pero no cualquiera, las personas que se trabajando en el proyecto. Además, debemos de estimar en términos de grupos.

Bajo la idea de que lo que buscamos es aprender de los procesos, entendemos también a las estimaciones como procesos y a partir de ellas aprendemos. Por eso la importancia del proceso de estimar.

En ágil el foco de estimación lo va a tener el producto. Estimamos solo y únicamente producto. Y lo haremos en base a comparaciones, tenemos estimaciones relativas!

Aplica también el just in time en concordancia con el cono de la incertidumbre.

## Story Point

Son una medida de tamaño relativo. Son una ponderación que se le da a la US cuyo numero proviene del análisis de 3 factores importantes:

- Complejidad
- Esfuerzo: Siempre atado al tamaño
- Incertidumbre

Son estimaciones propias del equipo. Solo les sirven a ellos realizarlas.

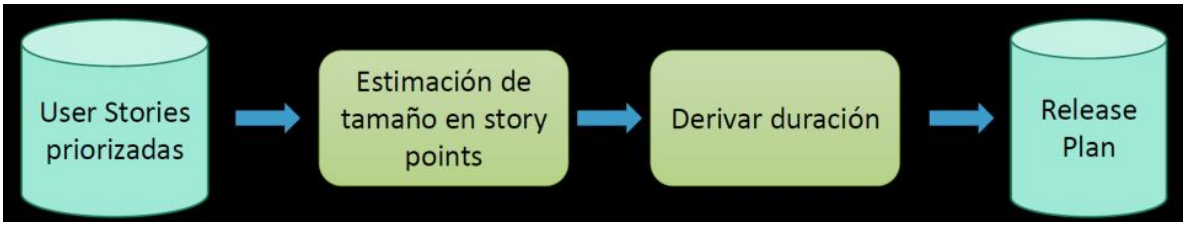
Para poder hablar de relatividad en los story points definiremos una US que será canónica. Es nuestro punto de comparación para el resto de las US. No necesariamente tiene que ser la más pequeña.

## Velocidad

Es la métrica más importante en ágil porque mide el progreso de un equipo en termino de cuanto producto le entrego yo al cliente. Decimos que es la más importante porque ágil propone que lo más importante es el software funcional y esta es una métrica que lo mide.

Se calcula, no se estima, sumando la cantidad de storypoints de las US que se entregaron al PO y cumplieron con el DoR.

En base a esta métrica podremos ver si se está cumpliendo el principio de desarrollo sostenible. Y logro tener previsibilidad en cuanto a que puedo esperar de un equipo en que tiempo.




## Método de estimación: Poker Estimation

Combina el juicio experto Delphi con analogía.

Se basa en la serie Fibonacci o números que tengan un crecimiento exponencial puesto a que la dificultad del software es exponencial.

Si tenemos SP grandes -> US no cumple con el DoR.

La canónica por lo general es 1. 0 significa que hay incertidumbre y el limite es 8.

Este método debe cumplir con el ciclo de retroalimentación del empirismo. Asumir, construyo, retroalimento, inspecciono, adapto.

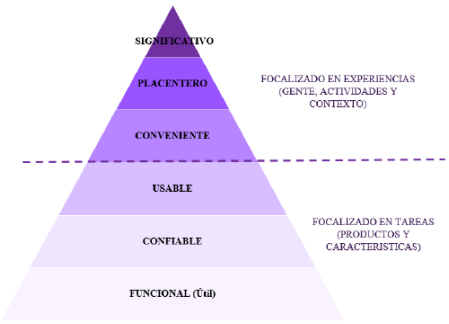
# Gestión de productos

La gestión es como hacemos nosotros con respecto al producto de soft que queremos construir y comercializar.

No es lo mismo que gestionar un proyecto. Tiene que ver con cual es nuestra visión o para que construimos software.

Existen diferentes razones que no son solo dinero o satisfacer clientes. A veces buscamos una masividad del producto o lograr un cambio significativo en el mundo.

Independientemente de nuestra visión hay algo que es transversal en todos los productos y es que solamente el 20% de la funcionalidades programadas se usan de manera regular mientras que el 80% casi nunca. Buscaremos eliminar el desperdicio de esto.




En esta pirámide podemos ver la evolución de los productos de software pensando en las características.

En la base tenemos lo importante que aparezca, que el software sirva para lo que fue creado, sea confiable y tenga algún lineamiento con ux

En el tope vamos a ver aspectos difíciles de obtener. Mas relacionado a la visión del producto. Hablamos de un producto conveniente, placentero y significativo.

El desafío entonces se encuentra en ver cómo hacemos para construir un producto de software donde evitemos el desperdicio y podamos abarcar los aspectos que queremos que tenga el producto.

Para lograrlo vamos a entender el concepto de MVP y todos los mínimos.

## Tecnica UVP

Esta técnica Propuesta de valor para el usuario es una técnica de desarrollo de productos que se centre en comprender al usuario y crear soluciones que satisfagan esas necesidades

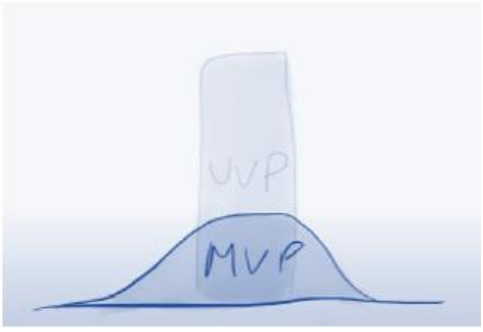
Vamos a iniciar un circuito de aprendizaje, poniendo a prueba la hipótesis y adaptándonos.

Primero debemos de plantear una hipótesis. Un valor que tiene el producto o servicio que quiero desarrollar. Mi visión del producto va a ser poder resolverla.

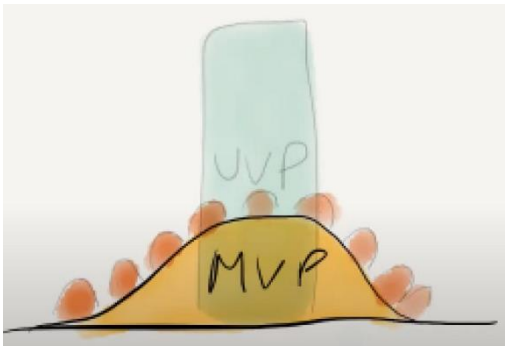
Queremos validarla para que nosotros no corramos el riesgo de armar un producto que no tiene un mercado. De esta manera buscamos minimizar el esfuerzo y el desperdicio. Construyendo un producto que el usuario quiere y demanda.

**MVP**

Mínimo producto viable. La idea es trabajar con lo mínimo del producto que necesito obtener. Es lo mínimo que necesita el producto para poder validar la hipótesis, es decir, que los usuarios lo puedan probar y decir si les gusta o no.



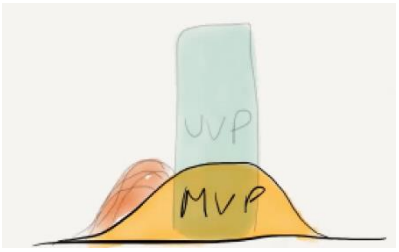

Objetivo es comprobar que la hipótesis del producto a construir funciona, o si necesitamos cambios.



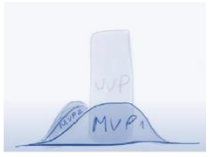

Nos puede suceder que la hipotesis como está definida sirva o que cada cliente potencial pida ciertas funcionalidades diferentes.

Si yo recibo muchas funcionalidades podemos ver que el mercado no está tan bien definido. Hay que seguir trabajando sobre la vision del producto.

Otra cosa que nos puede pasar es que la hipotesis planteada tenga algunas variaciones. Entonces la hipotesis va a poder moverse o modificarse según la exigencia. Mi hipotesis se desvia siempre hacia donde está el foco de los clientes.




En este escenario es necesario que redefinamos el mvp. Construiremos uno nuevo pivoteado hacia la nueva hipotesis. Tratando de encontrar cual es el producto que realmente estan buscando. Este circuito puede ocurrir varias veces hasta obetener un MVP exitoso.




Entra entonces alli el concepto de MMF

**MMF**

Mínima característica marketineable.

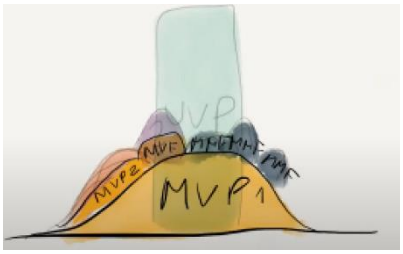
Dejamos de lado la hipótesis, ya fue validada, nos centraremos en definir elementos que deben estar presentes en un producto para que sea comercializable. validación del producto en términos de comercialización.

Busco la pieza mínima para que el producto sea rentable. Nos permite sacarlo al mercado minimizando el esfuerzo necesario. Aparte es importante sacarlo cuanto antes para no perder mercado.

**MVF**

Mínima caracteristica viable

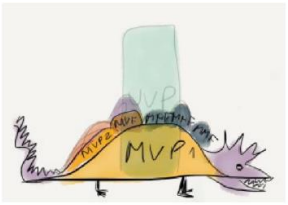
Se trata de una caracteristica mínima que se puede construir e implementar rápidamente. Quiero saber cual es la caracteristica mínima que hace la diferencia en mi producto.




Es una feature que se busca que por si misma tenga valor de negocio.

De alguna manera la MVF es una versión o parte del MVP

Si seguimos trabajando en el contexto de validar la hipotesis y ver como vamos a trabajar con las caracteristica. Obtenemos el dinosauiro.




Ademas tambien tenemos la MRF que es la caracteristica minima del release que se trata del release del producto tenga ciertas caracteristicas Es el incremento más pequeño que ofrece valor.

**MVP**
- Versión de un **nuevo producto** creado con el **menor esfuerzo posible**
- Dirigido a un **subconjunto de clientes potenciales**
- Utilizado para obtener **aprendizaje validado**.
- Más cercano a los **prototipos que a una version real funcionando de un producto**.

**MMF**
- es la **pieza más pequeña de funcionalidad** que puede ser liberada
- tiene valor tanto para la organización como para los usuarios.
- Es parte de un MMR or MMP.

**MMP**
- Primer release de un MMR dirigido a **primeros usuarios** (early adopters),
- Focalizado en características clave que satisfarán a este grupo clave.

**MMR**
- Release de un producto que tiene el conjunto de características más pequeño posible.
- El incremento más pequeño que ofrece un valor nuevo a los usuarios y satisfice sus necesidades actuales.
- **MMP = MMR1**

## Errores más comunes:

- Confundir un MVP con MMF o con MMP. El MVP se enfoca en el aprendizaje, los otros estamos hablando de comercializar y por ende ganar.

# Gestión de configuración de software - SCM

Los sistemas siempren cambian durante su desarrollo y posterior uso, conforme a esto, se irán creando diferentes versiones de software. Entonces entenderemos a los mismos como un conjunto de versiones de sistemas donde cada una de ellas debe mantenerse y gestionarse. – Necesario para mantener la trazabilidad –

> Trazabilidad: capacidad de rastrear y documentar de manera sistemática la relación entre diferentes artefactos y elementos en el ciclo de vida del desarrollo de software

Los cambios en el software se dan por cambios en el negocio y en el contexto donde se desarrollan. Ante estos cambios debemos ser capaces de mantener la integridad.

El **SCM** es una disciplina de soporte. Que ocurre transversalmente a lo largo de todo el proyecto y lo trasciende para darle soporte al producto.

El **objetivo** es mantener la integridad del producto a lo largo de todo el ciclo de vida. Para hablar de que un producto tenga integridad este debe tener:

- Satisfacer necesidades del usuario
- Fácil y rastreable durante su ciclo de vida: Existen vínculos entre los IC que permiten analizar en donde impacta un cambio.
- Satisfacer criterios de performance
- Cumplir con las expectativas de costo.

La idea va a ser resolver problemas de diferentes indoles pero siempre teniendo en cuenta que debemos mantener la integridad. Implica identificar la configuración en un momento, controlar sistemáticamente sus cambios y mantener su integridad y origen.

## Conceptos generales

### ítem de configuración (IC)

*Son todos y cada uno de los artefactos que forman parte del producto o proyecto, que pueden sufrir cambios o necesitan ser compartidos y sobre los cuales necesitamos conocer su estado y evolución. Además, se puedan guardar en un repositorio.*

Un ítem es software. Deben estar acompañados con su versión y un nombre univoco.

Además, podremos identificar ítems por cada ciclo de vida. Y duraran lo que dure ese ciclo de vida.

### Repositorio

Es el contenedor de los ítems de configuración. Se encarga de mantener la historia y relaciones de los IC.

Tiene que tener una estructura definida, que nos permite determinar que ítems pondremos en que lugares. Ayudando a la seguridad, control de acceso, políticas de backup

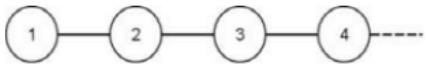
Dos tipos centralizados y descentralizados.

### Versión

Es una instancia de un ítem que difiere del resto

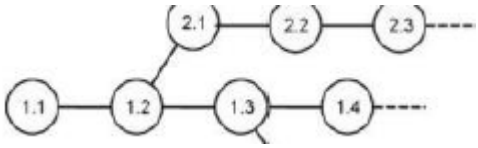
Es un estado del IC en un momento determinado. Se define desde el contexto de la evolución como la forma particular de un IC en momento determinado.

El control de versiones: Evolución de un único IC, o de cada IC por separado. La evolución puede representarse gráficamente.




### Variante

Es una versión de un IC que evoluciona por separado. Representando configuraciones alternativas.




### Configuración de software

Conjunto o sumatoria de todos los IC con su versión especifica en un momento determinado. Equivale a una foto de todos los ítems con su versión en un momento.

Es lo que vamos a gestionar.

### Línea Base

Es un IC o un conjunto de IC que han sido construidos, revisados formalmente y se los considera estable. Nos sirva como referencia para avanzar con el desarrollo.

Es una configuración de software, revisada y a la cual se la considera estable. Las cuales van a estar etiquetados para que sea posible la distinción.

Lo definiré para tenerlo como punto de referencia, para hacer rollback si lo necesito o saber que poner en producción. Nos permite saber cuál era la última situación estable en un momento de tiempo y cómo se fue evolucionando.

Se puede gestionar de varias maneras. Manteniendo una ultima LB o manteniendo una por fase. Además hay 2 tipos

- Especificación: No contienen código, solo información de ing de soft

- Operacionales: Contiene una versión de producto cuyo código es ejecutable. Pasaron por un control de calidad definido previamente.

### Rama – Branch

Conjunto de ítems de configuración con sus correspondientes versiones que nos permiten bifurcar el desarrollo. Se hace por varios motivos: Experimentación, resolución de errores, etc.

El merge es la acción por la cual uniremos la rama con el tronco. En caso de unirse las remas deben de descartarse.

## Extra SCM

La SCM tiene un rol determinado que es una persona que llevara a cabo ciertas tareas mas especificas con el SCM pero no significa que el resto del equipo se desentienda.

El SCM es más bien colaborativo. Todos los integrantes del equipo deben estar dispuestos a cumplir con lo planteado, ya sea manera de escribir los IC o donde se ubicarán, etc.

Además, debemos de entender que el software es muy fácil de cambiarlo. Pero cambiarlo en el sentido de eliminar trabajo de 6 meses con un click. Entonces buscamos tener control sobre los cambios para que esto no pase.

## Actividades relacionadas al SCM

Son las secciones que contendrá un plan de SCM.

Todas las actividades se realizan en ambas metodologías, menos la auditoria porque no va con los principios del ágil puro.

### Identificación de IC

Identificación univoca para cada IC, donde se definirán políticas y reglas de nombrado y versionado para todos ellos.

También definiremos la estructura del repositorio y la ubicación de los IC.

Proveen un camino que une a todas las etapas del CV del software lo que nos ayudará a controlar y velar por la integridad.

Se identifican IC según el CV

- Producto: CV más largo y se mantiene mientras el producto exista
- Proyecto: CV de un proyecto, impactando en el esquema de nombrado.
- Iteración: CV más corto.

### Control de cambios

Tiene su origen en la necesidad de realizar cambios en la LB

Es un procedimiento formal, que involucra un comité de cambios y una evaluación del impacto. Tratamos de mantener la integridad.

El control se hace sobre los ítems de configuración que pertenecen a la LB. Por eso decimos que necesitamos una revisión completa porque es un punto de referencia para todos.

#### Comité de control de cambios

Formarán parte del comité aquellas partes que se encuentren directamente afectada por los cambios. Los interesados en evaluar y enterarse de los cambios.

Las etapas son simples, se recibe un cambio, se realiza una evaluación de impacto, en caso de autorizarse se genera una orden de cambio, luego debe volver a intervenir para aprobar la modificación y finalmente se los notif.

### Auditorias de configuración

La auditoria es una revisión o control independiente y objetivo que se realiza sobre un producto o proceso. En este caso sobre una LB.

Es importante que el auditor sea una persona independiente para obtener lo objetividad que buscamos.

Se realiza mientras el producto se está construyendo. Logrando mantener la integridad y calidad.

Tenemos dos tipos:

1. Físicas (PCA): Realiza una verificación de que el repositorio sea integro.
    a. Buscamos que cumpla con lo planteado
    b. Que los IC puestos estén para cumplir con los requerimientos.
2. Funcional (FCA): Controla que la funcionalidad y performance reales de cada IC sea consistente con los requerimientos. Realiza una validación de que lo construí sea lo correcto y definido en los requerimientos.

| Validación | Verificación |
|---|---|
| Se encarga de asegurar que IC resuelva el problema apropiado.<br>Utilizaremos la matriz de trazabilidad para llegar desde una implementación hasta la definición de requerimientos. | Asegurar que un producto cumple con lo definido en la documentación de LB<br>Todas las funcionalidades son llevadas a cabo con éxito. |

### Informes de estado

Provee un mecanismo para mantener un registro de cómo evoluciona el sistema. Permite que exista la transparencia en todo el proyecto.

De esta manera tomamos decisiones con menor incertidumbre.

La idea principal es que se entere el que se tenga que entrar de lo que se tiene que enterar.

### Plan de SCM

Es un plan que debe confeccionarse al inicio del proyecto. Debe incluirse los puntos anteriormente definidos y debe hacerse tempranamente, definir los documentos que serán administrados y no debe quedar ningún producto del proceso sin administrarse.

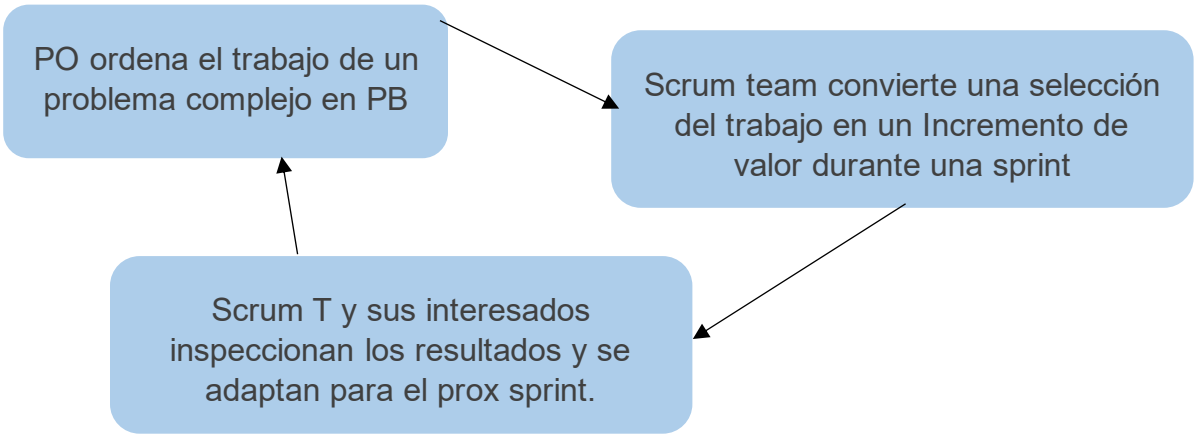
# SCRUM

Es un marco de trabajo liviano que ayuda a personas equipo y organizaciones a generar valor a través de soluciones adaptativas para problemas complejos.

Es un framework ágil de gestión de proyecto. Soluciones adaptativas a problemas complejos.

Nos sirve para desarrollar un producto en un contexto complejo e incierto.

En pocas palabras su funcionamiento es:




Es incompleto de manera intencional, solo define las partes necesarias para implementar la teoría de SCRUM.

Se basa en el empirismo (conocimiento proviene de las experiencias) y en el pensamiento Lean de donde toma el foco para eliminar el desperdicio y centrarse solamente en lo esencial.

Tiene un enfoque Iterativo e Incremental para optimizar la previsibilidad y controlar riesgo. Trabaja con empirismo por lo que tener un enfoque iterativo hace que existan mayores momentos de aprendizaje.

## Empirismo en SCRUM

Scrum combina cuatro eventos formales para inspección y adaptación en un evento contenedor denominado SPRINT. Los mismo funcionan porque aplican los 4 pilares del empirismo

### Transparencia

El proceso y trabajo deben ser visible para tanto quienes realizan el trabajo como para quien lo recibe y en última instancia la organización entera.

SCRUM toma decisiones en base a sus 3 artefactos (PB, SB, Inc) los artefactos con poca transparencia pueden llevar a decisiones que disminuyan el valor.

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

### Inspección

Los artefactos de SCRUM y el progreso hacia los objetivos acordados deben inspeccionarse con frecuencia. Así detectamos desviaciones.

SCRUM proporciona cadencia (sus cinco eventos). La cadencia es algo que se repite con ritmo.

La inspección permite la adaptación y sin ella se considera inútil. Los eventos de scrum están diseñados para provocar cambios.

### Adaptación

Si tenemos que algún aspecto del proceso se desvía fuera de los límites entonces debemos de ajustar el proceso.

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

Se espera que el ST se adapte en el momento que aprenda algo nuevo a través de la inspección.

## Valores

Scrum depende que las personas tengan 5 valores específicos:

***Compromiso, Foco, Franqueza, Respeto y Coraje***

El ST debe **comprometerse** a lograr sus objetivos y ayudarse. Su **Foco** principal está en el trabajo del Sprint para lograr progreso hacia los objetivos. El ST y los interesados son **Francos** respecto a su trabajo y los desafíos. Además, existe **Respeto** entre los miembros para ser personas capaces e independientes. Por último, tiene el **Coraje** de hacer lo correcto para trabajar en problemas difíciles.

Cuando estos valores se aplican los pilares del empirismo cobran vida y generan confianza.

## Roles o Categorías de responsabilidades

### Scrum Team (ST)

- Es la unidad fundamental.
- Consta de: Scrum Master (SM), Product Owner (PO), Developers (Dev)
- No existen las jerarquías
- Son multifuncionales, los miembros tienen todas las habilidades necesarias para crear valor en cada sprint.
- Se autogestionan (que no es lo mismo que autoorganizan) decidiendo intencionalmente quien hace que cuando y como.
- Son pequeños si son muy grande se dividen en varios ST con el mismo objetivo, PB y PO.
- Tiene la responsabilidad de crear un incremento valioso y útil cada sprint.

### Developers o teams members

- Personas del ST que se comprometen a crear cualquier aspecto de un incremento utilizable en cada sprint.
- Responsable de crear el SB y definir su granularidad.

### Product Owner

- Una persona responsable de maximizar el valor del producto resultante del trabajo del ST
- Es responsable de la gestión del PB
- Las decisiones del PO son visibles en el contenido y orden del PB y a través del Incremento inspeccionable en la Sprint Review.

**Scrum Master (SM)**

- Responsable de establecer Scrum como se define en la guía.
- También debe lograr la efectividad del ST mejorando las prácticas dentro de SCRUM.
- LIDER
- Asegura que todos los eventos del scrum se lleven a cabo y estén en los límites de tiempo.
- Al PO lo ayuda en la gestión del PB y en la comunicación del mismo al team.
- Ayuda al equipo a enfocarse en crear incrementos de valor que cumplan con el DoD.

## Eventos

Cada evento en Scrum es una oportunidad formal para inspeccionar y adaptar los artefactos de Scrum.

Diseñados para habilitar la transparencia.

Son oportunidades para inspección y adaptarse.

---

**Time-Box**

Es un concepto importante porque nos ayuda a generar un foco y tratar de llegar con el alcance. De lo contrario entregaremos menos pero nunca movemos el tiempo.

Es una restricción que condiciona la manera de trabajo

Cuando lo relacionamos con la triple restricción es la razón por la cual dejamos fijo al tiempo.

Cada evento tiene tiempos determinados por el equipo y deben ser cumplidos.

El refinamiento se plantea como un porcentaje porque es una actividad continua. Decimos que nos lleva un 10% del tiempo de sprint.

---

### Sprint

Donde las ideas se convierten en valor.

Son eventos de duración fija o menos para crear consistencia.

Es el evento contenedor del resto de los eventos.

Consta de todo el trabajo necesario para lograr el Objetivo del sprint.

- No se realizan cambios que pongan en peligro el Objetivo
- La calidad no disminuye
- PB se refina según sea necesario
- El alcance se puede aclarar y renegociar.

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

Podemos hacer sprints más cortos y generar más ciclos de aprendizaje.

Un sprint podrá cancelarse solamente si el objetivo del sprint se vuelve obsoleto y será decisión del PO para hacerlo.

### Sprint Planning

Inicia al Sprint al establecer el trabajo que se realizará para el sprint.

Aborda varios temas:

1. ¿Por qué es valioso este Sprint?
    a. El PO propone cómo el producto podría Incrementar su valor y utilidad en el Sprint actual
    b. Se define un Objetivo del Sprint que comunica porque es valioso
2. ¿Qué se puede hacer en este sprint?
    a. Se seleccionan elementos del PB para incluirnos en el sprint. Se pueden refinar los elementos durante este proceso
    b. Seleccionar cuanto es un problema, pero lo hacemos en base al desempeño pasado, capacidad actual y la DoD.
3. ¿Cómo se realizará el trabajo?
    a. Planifica el trabajo necesario para crear un incremento que cumpla con el DoD.
    b. Se descomponen los elementos del PB en trabajo más pequeño de un día o menos.

Con todo lo anterior tenemos Objetivo de sprint, elementos del PB seleccionados y el plan para entregarlos. Todo ello conforma el Sprint Backlog.

Limite de 8hs para 1 mes

### Daily Scrum

Inspeccionar el progreso hacia el Objetivo del Sprint y adaptar el SB según sea necesario

Evento de 15 minutos para los developers del ST.

Siempre a la misma hora y lugar.

Si el PO o SM están trabajando activamente en elementos del SB participan como Dev

Se debe centrar en el progreso hacia el objetivo del Sprint. Produciendo un plan viable.

Mejoran comunicación, identifican impedimentos, promueven la toma rápida de decisiones y eliminan la necesidad de otras reuniones.

### Sprint Review

Se inspecciona el resultado del Sprint y determinan futuras adaptaciones.

ST presenta los resultados de su trabajo al PO y se discute el progreso hacia el objetivo del producto

Se revisa lo que se logró en el Sprint y lo que ha cambiado en su entorno. Con esto se colabora para definir que hacer a continuación.

El PB también puede ajustarse para satisfacer nuevas oportunidades.

4 Hs para 1 mes

### Sprint Restrospective

Planificar formas de aumentar la calidad y la efectividad

El ST inspecciona cómo fue el último Sprint con respecto a las personas, interacciones, procesos, herramientas y el DoD.

Se analiza lo que se hizo bien, que problemas se encontró y como se resolvieron (o no).

Identifica los cambios más útiles para mejorar su efectividad.

Concluye el sprint y tiene un tiempo de 3 Hs para 1 mes.

## Artefactos

Representan trabajo o valor. Se diseñaron para maximizar la transparencia de la información clave.

Cada artefacto tiene un compromiso para garantizar que proporcione información que mejora la transparencia y ayuda al enfoque. Nos ayudan además a reforzar el empirismo y los valores de Scrum.

### Product Backlog

Lista emergente y ordenada de lo que necesitamos para mejorar el producto.

Es la única fuente de trabajo realizado por el ST.

Suele refinarse para adquirir un grado de transparencia.

Refinamiento: Acto de dividir y definir aún más los elementos del PB en elementos más pequeños y precisos. Actividad continua para agregar detalles.

*Compromiso: Objetivo del producto*

El objetivo del producto describe un estado futuro del producto que puede servir como un objetivo para que el equipo planifique. Este se encuentra en el PB el resto del PB emerge para definir "qué" cumplirá con el objetivo del producto.

### Sprint Backlog

Compone del objetivo del sprint (por qué), conjunto de elementos del producto backlog seleccionados (qué) y el plan de acción para entregar el incremento (cómo).

El SB es un plan realizado por y para los Dev. Imagen muy visible y en tiempo real del trabajo que los Dev planean realizar durante el Sprint para lograr el objetivo.

Se mantiene actualizado a lo largo del sprint y debe tener suficiente detalles para poder inspeccionar su progreso en la daily.

*Compromiso: Objetivo del sprint.*

Este objetivo es el único propósito del Sprint. Este es un compromiso de los Dev pero proporciona flexibilidad en términos de como hacerlo. Además crea coherencia y enfoque lo que alienta al trabajo en equipo.

### Incremento

Parte concreta hacia el objetivo del producto. C/Increm se suma a todos los anteriores y se verifica minuciosamente. Garantizando que todos funcionen juntos.

El mismo debe ser utilizable para generar valor

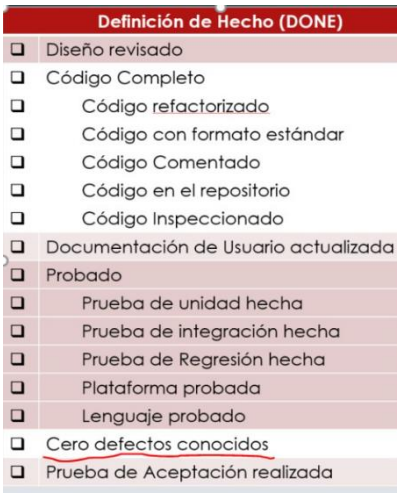
Un sprint puede generar varios incrementos. La suma de los increments se presenta en la Sprint Review. No se considera nunca el trabajo a no ser que cumpla con el DoD

*Compromiso: Definición de Terminado*

DoD es una descripción formal del estado del Increment cuando cumple las medidas de calidad requeridas para el producto.

Un elemento del PB que cumple con el DoD es un Increment y se puede presentar en la review.

Crea transparencia al brindar a todos un entendimiento compartido de qué se completó como parte del Increment.




Tanto el DoD y el DoR están directamente relacionado con la calidad esperada de los productos y la exigencia de estos según su propósito.

Deben ser concretos

Es un momento particular que resalta el compromiso y la cultura del respeto porque todos nos comprometimos a cumplir todos los check de las definiciones entonces no podemos ninguna afuera sino el producto no se encuentra listo.

## Métricas

### Capacidad

Es la única que se estima. Se trata de ver cuanto compromiso de trabajo un equipo puede asumir en un determinado sprint.

Contra esto yo constato cuantas US me comprometo a terminar en un sprint.

Se estima porque se calcula al comienzo de un sprint.

Se mide en Horas ideales y si los equipos son maduros pueden usar SP.

No se busca ser absolutos.

### Running Tested Features

No se utiliza, cuenta la cantidad de características funcionando desarrolladas en un sprint.

### Velocidad

Se calcula contando la cantidad SP que el PO me acepto al final de una sprint.

## Herramientas

### Taskboard

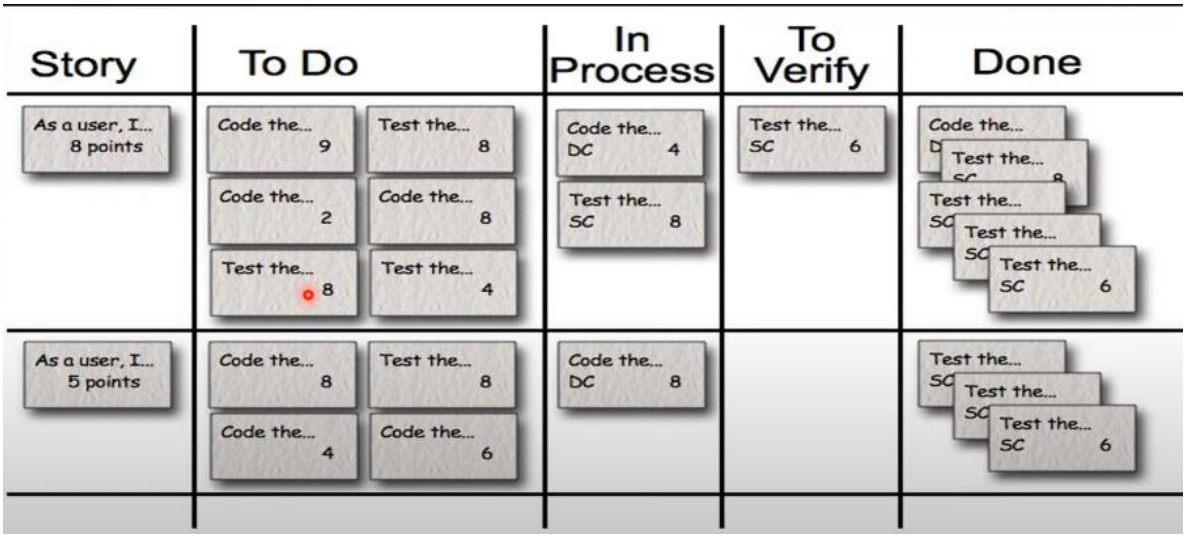
SCRUM utiliza una pizarra para hacer visible al sprint backlog a todos los interesados y el ST.

En ella encontraremos columnas que variarán según el equipo. Puesto a que si bien existe una configuración básica puede ser modificado para adaptarse a las exigencias del equipo.

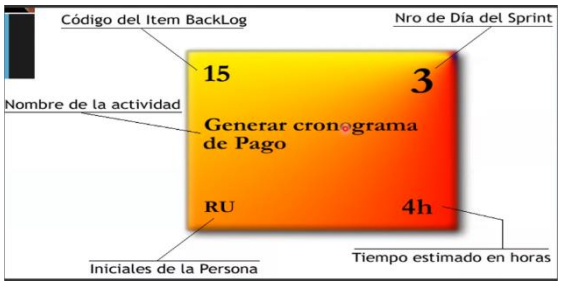
Configuración Básica

| TO DO | IN PROCESS | DONE |
|---|---|---|
| Características que salieron del PB y entran al SB | Lo que toma una persona para trabajar | Cuando cumple con el DoD |

Si el equipo desagrega las historias en tareas tenemos:




Las tarjetas representan US o tareas y se ven así:



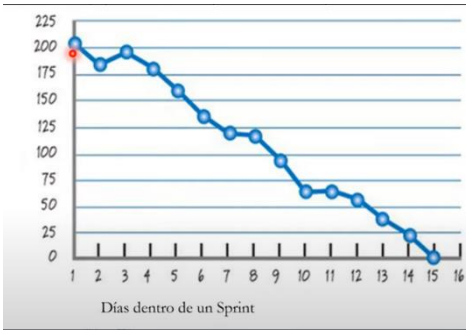

En donde lo importante es que la estimación puede ser en Hs Ideales o SP (scrum recomienda SP)

Las tareas se miden en horas ideales porque no son producto. Buscamos una granularidad más bien fina para mejor manejo y poder empezarlas y terminarlas en un sprint.

Las tareas se hacen o no. Gestion binaria.

## Gráficos

### Sprint Burdown Charts




Es un gráfico que nos muestra el trabajo terminado o el que nos falta para cumplir con el compromiso de nuestro sprint.

No es una métrica es un grafico

Se utiliza para poder ir viendo el progreso del sprint. Visualización del trabajo

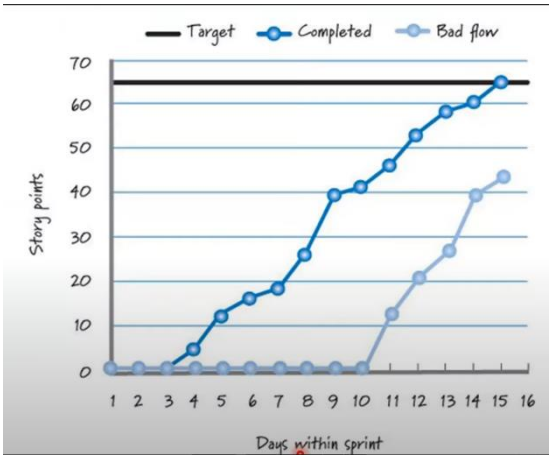
En la daily se actualiza el grafico en función del avance reportado.

Puede bajar y luego subir porque hay reestimaciones. Porque cuando estimamos se subestimo una US.

En base a la misma podemos definir una curva ideal y compararla con la real. Es decir que podemos tener un control de cómo vamos con el trabajo en nuestra sprint. Así podremos predecir si cumpliremos o no con el compromiso del sprint.

Se borran al final de cada sprint.

### Sprint burnup chart

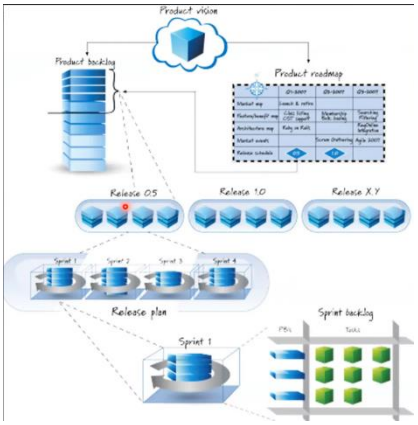



Este es permanente porque hace seguimiento de como voy trabajando en un producto a lo largo de todos los sprints.

Curva que sube porque va mostrando lo que voy haciendo para llegar al objetivo

## Niveles de planificación

Entendemos que ningún producto de software puede terminarse en un sprint.




La planificación siempre esta sujeta a cambios

El nivel más chico es la **daily** donde sincronizamos y vamos viendo la necesidad de hacer ajustes.

La **Sprint** le sigue. Obtenemos como resultado el sprint backlog. Nos preguntamos el porque, que y como.

**Planificación del reléase** entendiendo como reléase a la versión de producto que tiene ciertas características que se libera y se pone en producción.

Definimos cuantos sprints voy a necesitar para poder entregar esta cantidad de características del producto.

Configuramos el PB y a partir de allí veremos cuantas sprints me llevará realizarlo.

**Planificación del producto** entendiéndolo como la sumatoria de releases. Implica tomar decisiones sobre los req funcionales y no funcionales y definir en que reléase entrego cada característica.

**Planificación de portfolio** es una línea de productos y planificamos que producto le damos más recursos y cuales descontinuamos.

# Nexus

Nexus es un grupo de entre 3 y 9 Scrums Teams que trabajan juntos para entregar un único producto.

Tiene un solo: Product Owner , Producto Backlog

Elaborado sobre las bases de SCRUM. Tomara este marco para facilitar que múltiples equipos trabajen en un único PB. Así ellos formarán un Integrated Increment.

Es un marco para escalar SCRUM.

Nos elimina desafíos comunes como la dependencia entre equipos, preservar la autogestión y transparencia de los equipos y asegurar su responsabilidad.

## Dependencias

Entendemos a las dependencias como las situaciones donde una actividad o trabajo de un equipo depende de otro equipo para completarse.

Nexus nos ayuda a hacerlas transparentes y su origen provienen de:

- Estructura del producto: Grado al cual diferentes asuntos están separados independientemente en el producto.
- Estructura de la comunicación: Forma en que las personas se comunican dentro y entre los equipos.

## Escalando SCRUM

Nexus lo realiza de la siguiente manera:

- **Responsabilidades:** Nexus se asegura que se entregue un incremento útil y de valor en cada sprint. El Nexus Integration Team (ntt) está formado por PO, SM y miembros.
- **Eventos:** Se agregan, colocan alrededor o reemplazan a los eventos de SCRUM.
- **Artefactos:** Todos utilizan el mismo PB y tenemos un Nexus Sprint Backlog. El integrated Increment representa la suma de todo el trabajo integrado.

### Responsabilidades

Nexus Integration Team

Responsable de asegurar que se produzca el IntInc al menos una vez por cada sprint.

Proporciona un foco de integración para el Nexus.

Conformado por:

- **Product Owner:** Nexus funciona con un solo PB. El PO es responsable de maximizar el valor del producto y el trabajo realizado e integrado por los ST. Gestión eficaz del PB
- **SCRUM Master:** Responsable de asegurar que el marco de trabajo Nexus se entienda y se promulgue como se describe en la guía.
- **Uno o más miembros del NIT:** Formado por miembros de los ST que ayudan a los ST a adoptar herramientas y practicas que contribuyen a mejorar la capacidad de ST

### Eventos

Estos eventos también están con time-box

Son atendidos por los miembros del Nexus que sean necesarios para lograr el objetivo de ese evento.

- **Sprint:** Igual que scrum. Produce un único incremento integrado
- **Refinamiento entre equipos:** Reduce o elimina las dependencias entre equipos. Se descompone el PB para poder ver las dependencias.
    - Ayuda a los ST a prever que equipo entregará que elemento
    - Identifica dependencias entre los equipos

  Es continuo y puede seguir en cada ST

- **Nexus Sprint Planning:** Coordinar las actividades de todos los ST dentro de un Nexus para un solo Sprint. Los representantes de cada ST y el PO se unen para planificar el sprint. Obtenemos:
    - Objetivo del Sprint alineado con el del producto
    - Un objetivo de Sprint para cada ST alineado con el objetivo del Sprint Nexus
    - Un solo sprint nexus backlog.
    - Un SB para cada ST.
- **Nexus Daily Scrum:** Identificar cualquier problema de integración e inspeccionar el progreso hacia el objetivo del Sprint Nexus. Asisten representantes de los ST, inspeccionan el estado actual del IntInc e identifican problemas de integración y las dependencias.
    - **Daily Scrum:** De cada ST se complementa con la de nexus.
- **Nexus Sprint Review:** Final de cada sprint para brindar retroalimentación de los interesados.
    - Reemplaza a la sprint review individuales.
- **Nexus Sprint Restrospective:** Planificar formas de mejorar la calidad y la eficacia en todo el Nexus. Se complementa con las retrospectivas de cada ST.

### Artefactos

- **Product Backlog:** Uno solo que contiene una lista de lo que hay que hacer para mejorar el producto. Debemos de poder detectar dependencias.
    - **Compromiso es el objetivo del producto.**
- **Nexus Sprint Backlog:** Compuesto del objetivo del sprint nexus y elementos del PB de los SB de cada ST. Se resalta dependencias y flujo de trabajo
    - **Compromiso:** Objetivo del Sprint Nexus.
        - Este objetivo es el único para nexus. Suma de todo el trabajo y los objetivos de Sprint de cada ST. Se crea en el evento de planning.
- **Integrated Increment:** Representa la suma actual de todo el trabajo integrado completado por un nexus. Se inspecciona en la Nx Sprint Review pero puede entregarse antes. Debe cumplir con el DoD
    - **Compromiso es el DoD.** Definiendo el estado del trabajo integrado cuando este cumpla con la calidad y las medidas requeridas para el producto.

- Se termina solo cuando esta integrado, es de valor y utilizable.
- El DoD es universal para todos los ST. Todos deben cumplirla.

# Lean

Principios que lo sustentan:

### Eliminar desperdicios

Intentar que los procesos sean más eficientes y productivos haciendo que nuestro proceso permanentemente genere valor. Si no lo hacen es desperdicio

Algunos conocidos son: Caracts extras, trabajo a medias, procesos extra, movimiento, Defectos, esperas, cambio de tareas y talento no utilizado.

### Amplificar Aprendizaje

Transparencia en el empirismo. Somos un equipo y el trabajo que realizamos. El conocimiento individual convertirlo en conocimiento del equipo y de la organización.

Generar una cultura de mejora continua y un enfoque de proceso de calidad.

### Embeber la integridad conceptual

Relacionado a la calidad no se negocia. Todo nuestro trabajo tiene que estar enfocado en crear un producto de calidad. Ese aseguramiento lo tenemos que construir paso a paso.

Hay dos integridades:

- Percibida: el producto tiene un balance en func, uso, confiab y economía que gusta al usuario
- Conceptual: Todos los componentes funcionan de manera coherente en conjunto.

### Diferir compromisos hasta último momento responsable

La calidad de la toma de decisiones está relacionada con la cantidad de información que disponemos para realizarlo. El último momento responsable es porque no podemos nunca tomar la decisión.

### Empoderar al equipo

Tenemos que darle el poder al equipo para que sean capaces de tomar decisiones y realizar el trabajo, autoorganizarse y gestionarse. Son personas capacitadas, responsables y motivadas.

### Ver el todo

Necesitamos tener una visión integral del producto y el proceso. Entender dónde estamos hacia donde vamos y que hicimos.

### Entregar lo antes posible

Entrega frecuente de software funcionando. Direct relacionado con empirismo y la idea de poder hacer inspecciones en periodos cortos de tiempo y mejorar.

# Kanban

Kanban es un método para gestionar todos tipo de servicios profesionales como el de conocimiento.

Debemos de aclarar que no es ni una **metodología** ni un **marco de trabajo**.

Metodología contiene flujos de trabajos y procesos definidos y prescriptos. Con roles y responsabilidad que son específicos a un dominio

Framework es una metodología incompleta: Conjunto de estructuras que están destinadas a tener una aplicación más amplia, pero requiere adaptarlas.

Entonces es un *método de gestión que debe aplicarse a un proceso o forma de trabajo existente.* No debemos de preguntarnos si elegir Kanban o un FW, son compatibles ambas. Tenemos que añadir Kanban a la manera de trabajar actual.

Lo entendemos como el medio para mejorar el qué y el cómo se realizan las cosas.

Es bastante abstracto y tiene un amplio abanico de posibilidades.

## Valores

Kanban está motivado por la creencia de que es necesario respetar a todos los individuos que contribuyen en una organización. Los valores se resumen todos en uno sólo, que a su vez es la base de todos, el respeto. Tenemos que entender que el método necesita de todos los valores para ser aplicado fielmente.

Son 9 valores:

**Transparencia**

Compartir información abiertamente mejora el flujo de valor de negocio

**Equilibrio**

Entendimiento de que los diferentes aspectos, puntos de vista y capacidades deben ser equilibrados para conseguir efectividad.

**Colaboración**

Kanban fue creado para mejorar el trabajo en conjunto. La colaboración está en su corazón.

**Foco en el cliente**

Los clientes y el valor que reciben que fue creado por nosotros es el foco.

**Flujo**

La realización de ese trabajo es el flujo de valor, tanto si es continuo como puntual

**Liderazgo**

Habilidad de inspirar a otros a la acción a través del ejemplo, palabras y reflexión. En todos los niveles es necesario para alcanzar la entrega de valor y la mejora.

**Entendimiento**

Conocimiento de si mismo y de la organización para ir hacia adelante. Conocer el punto de inicio.

**Acuerdo**

Compromiso de avanzar juntos hacia los objetivos respetando las diferencias.

**Respeto**

Valorando, entendiendo y mostrando consideración por las personas.

## Principios directores

Kanban tiene 3 principios directores que no están relacionados al propósito o manera de cometer el cambio. Busca las llamadas a la acción y se basan en las necesidades en la organización.

### Sostenibilidad

Encontrar ritmo sostenible y un foco en la mejora.

*Mira hacia adentro de la organización*. Buscamos construir servicios evitando la sobrecarga en los que la demanda esta equilibrada con la capacidad del sistema (aquí nace el WiP y el sistema pull)

### Orientación al Servicio

Conseguir rendimiento y satisfacción del cliente

*Mira hacia el exterior*: Desde el propósito de la organización hacia sus clientes.

Tiene que ser el más claro y explícito para todas las organizaciones. Debemos entregar los servicios que el cliente espera y cubrir y superar sus necesidad y expectativas.

Como Kanban trata sobre la entrega y mejora de valor en los servicios es clave.

### Supervivencia

Mantenimiento de la competitividad y adaptabilidad

*Mira hacia el futuro.*

Buscamos garantizar que una organización sobreviva y prospere donde hay cambios e incertidumbre.

Nunca podremos asumir que con los procesos y tecnología actual será suficiente para el futuro.

## Principios de gestión de cambio

Entendemos que son comunes a todas las implementaciones.

Kanban no es una transformación de un día para el otro. Sabemos que utiliza un enfoque de cambio evolutivo, basándose en la forma de trabajo ya existente.

Se busca siempre mejorar a través de la retroalimentación y la colaboración.

Este método genera un cambio evolutivo a través de los conocimientos adquiridos por las personas que utilizan el tablero Kanban.

Son 3

- Empezar con lo que estes haciendo ahora
- Acordar buscar la mejora a través del cambio evolutivo
- Fomentar el liderazgo en cada nivel de la organización

El primer principio esta respaldado por el hecho de que se minimiza la resistencia al cambio cuando respetamos practicas y roles actuales y que además de que los procesos actuales tengan deficiencias también tienen fortalezas de los que se puede aprender para aplicar.

## Principios de entrega de servicios

Con Kanban tenemos que tomar un enfoque de orientación al servicio para comprender nuestra organización como ocurre el trabajo en ella. Para mejorar las prestaciones de servicios debemos de seguir estos principios (3):

- Entender las necesidades y expectativas del cliente y enfocarse en ellas
- Gestionas el trabajo, dejar que la gente se autoorganice alrededor de las tareas
- Evolucionar las políticas para mejorar los resultados hacia el cliente y negocio.

Están muy alineados con el principio director de orientación a servicios y entrega de valor al cliente.

## Practicas

Estás son las 6 practicas generales de Kanban.

### Visualizar:

Clave para tener colaboración eficaz e identificar oportunidades de mejora.

Hay que visualizar el trabajo, su flujo y los riesgos.

Se mejora mucho la transparencia.

Se puede realizar con un tablero. Pero lo importante es que debemos poder visualizar todo, tanto trabajo como políticas.

### Limitar el trabajo en progreso:

Indica el número de elementos de trabajo en un determinado momento. Los sistemas eficaces son los que se centran en el flujo de trabajo.

Limitamos para equilibrar la ocupación y asegurarnos el flujo. Nos cambia de un sistema "*push*" a uno "*pull*"

### Gestionar el flujo

Poder terminar el trabajo de la forma más fluida y predecible con un ritmo sostenido

El seguimiento y medición del flujo da come resultado información valiosa para gestionar expectativas del cliente y poder hacer predicciones.

Debería maximizar la entrega de valor y minimizar los tiempos de entrega y ser tan fluido como sea posible.

### Hacer las políticas explicitas

Manera de articular y definir un proceso que va más allá de la definición del flujo.

Políticas deben ser acordadas entre todas las partes interesadas y deben estar expuestas en un lugar destacado.

Las mismas permiten a las personas a la autoorganización dentro del grupo.

Estas deben ser pocas, sencillas, bien definidas, visibles, aplicables en todo momento y fácilmente modificables.

### Implementar ciclos de feedback

Necesarios para una entrega coordinada y mejorar la entrega de tu servicio. Son importantes para un cambio evolutivo.

### Mejorar colaborativamente, evolucionar experimentalmente.

Kanban es un método de mejora. Y de mejora continua.

Logramos este cambio se realice colaborativamente utilizando experimentos diseñados basados en modelos y en el método científico.

## En software (*Anderson*)

- El propósito es la mejora de los procesos.
- No es un proceso de desarrollo o metodología de administración de proyectos.
- Permite mejorar o introducir cambios en mis procesos permitiendo la evolución continua.
- Mejora continua.
- Mejora gradual de procesos para poder entregar producto.
- No es prescriptivo y tampoco define muchas cosas. Por sobre todo tenemos que entender de empezar con lo que tenemos y de ahí avanzar.

Para aplicar Kanban tenemos que:

- **Dividir el trabajo en piezas:**

  Una vez definido mi tablero tenemos el proceso mapeado. Donde cada columna representa un paso.

  Definimos que tipo de trabajo tenemos que atender. Ósea las piezas que pasan por las columnas.

  Tenemos diferentes niveles de granularidad: US, solicitudes de req, Defectos, Features, CU y tareas.

  > Hay una aclaración que en este caso Kanban no define un producto backlog y por eso podemos poner tareas.

- **Visualizarlo:**

  Al flujo de trabajo con herramientas como el tablero.

  Cada uno lo configura en función de lo que necesita hacer y las características propias del trabajo.

  Se visualizan también el WiP y las políticas

- **Usar colores diferentes para tipos de trabajo diferentes**
- **Escribir cada tarea en una nota diferente**

**Los pasos:**

1. Mapear el proceso al tablero
    a. Pensándolo como una cadena de valor donde cada paso suma valor.
    b. Empezamos con lo que tenemos, revisar las actividades y ver si suman o no para agregarlo -> Mapearlo
2. Ver las unidades de trabajo que manejamos
3. Definir los limites WiP
    a. Hay decisiones por si queremos o no limitar la cola de producto.
    b. Se puede ir corrigiendo.
4. Definir las políticas de servicio asociado a los tipos de trabajo
    a. Tenemos acuerdos de que tipo de servicios (asociados mayormente a tiempos de entrega)
    b. Sugerencias no obligación:
        i. Expreso (Maneja las urgencias)
            a. No guardo capacidad, pero si aparece se suspende todo
            b. Max 1
            c. Todo el equipo tiene que trabajar para que salga del tablero.
        ii. Fecha fija
            a. Sabemos cual es el momento máximo que podemos entregar lo que tenemos que hacer en la tarjeta
            b. Uno puede jugar con elegirla en base a la fecha fija.
            c. Riesgo: Relajarse y dejar al último (vuelve expreso)
        iii. Estándar
            a. Situación normal
            b. Adherirse a la forma de trabajo y límites del WIP
            c. Intangible
            d. Duda técnica
            e. El sistema funciona, pero estaría bueno hacerlo.
5. Empezar a trabajar
    a. Ver si aparecen cuellos de botella y tratarlos
    b. Clases de servicios en donde diferentes trabajos con diferentes políticas
    c. Flujo

## Conceptos generales

- El proceso se mapea a un conjunto de colas.
- Inventan el Justo a tiempo. No hacer sobre especificación y no hacer cosas anticipadamente
- 2 tipos desperdicios evitables e inevitables. Reuniones es un ejemplo de inevitables.

# Métricas

Tenemos 3 enfoques para tomar métricas: Lean, Ágil y Tradicional

Tomarlas tiene un costo y su beneficio debe ser mayor al costo.

> Medida cuantitativa o presencia o grado del valor sobre un atributo que quiero medir. *Es un número.*

La obtengo como resultado de un proceso de medición que nos permite construir identificadores para la toma decisión.

Tiene que ser objetiva (por ese es cuantitativa) medible y debe ser representada numéricamente.

*Es grado valor o presencia de un atributo en el aspecto que yo quiero medir*

En el software mediremos siempre 3 cosas: Proceso, Proyecto y Producto. Que coincide que son los 3 ejes de la ISW. A la gente acá no se la mide.

Las métricas no sirven para castigar la gente. No debemos de cuestionar a las personas porque terminamos haciendo que la gente termine escondiendo resultado o demás y se va la **transparencia.**

La elección de usar métricas surge por el hecho de generar visibilidad o transparencia sobre el proyecto, proceso o producto. Con esto generamos información adecuada para la toma de decisiones de calidad.

## Tradicional

En tradicional se habla de unas métricas básicas que justamente se denominan básicas porque son lo mínimo para obtener un poco de visibilidad.

Tiene un foco en TODO. Es importante resaltarlo porque es un punto de diferencia entre los 3 enfoques.

Toma métricas

- Tamaño del producto **Que**
  Primero se habla de producto. Podemos contar las líneas de código, pero la mejor forma seria medirlo con requerimientos. (Funciones, alcances, CU, opciones de menú, etc.)
  Medido en lo que el proyecto considere conveniente. Mejor con requerimientos.
  Una aclaración es que se puede hablar de complejidades para no tener 200 CU que son simples y valgan lo mismo que 1 muy complejo.

- Esfuerzo **Cómo**
  Esfuerzo que voy a necesitar para construir el producto.
  Hs personas lineales. No se tiene en cuento los solapamientos (lineal)
  Nivel de granularidad lo define uno si es por proyecto, CU, workflow

- Tiempo Calendario **Cuando**
  Derivado del esfuerzo entra más variables como la cantidad de días a trabajar, las horas por día, índice de solapamiento, cantidad de gente.
  Armamos la fecha que vamos a trabajar
  Se mide en meses, días, semanas. Medir tiempo en horas confunde.

- Defectos
  Medir sobre el producto cuales son las cosas que se detectaron que no son coincidentes con lo que se esperaban.
  Debe de construirse según la severidad. Es lo clásico y si no lo tenemos no sirve.
  Otra interesante es la densidad de defectos. Cuantos defectos tengo por bloque de código.

Las métricas tienen un costo asociado y la precisión de estas es cara. Mantengamos las cosas lo más simple posible.

Todas las métricas sirven para un propósito lo que nos podemos encontrar es que ese propósito varie según el que esté trabajando en el proyecto.

Las métricas con costo muy altos no son viables y debemos mantenerlo lo más simple posible.

NOSOTROS debemos de satisfacer necesidades y expectativas de todos los interesados. Buscamos un equilibrio y en función de eso armar las métricas que voy a necesitar para poder darle a información a los interesados que deben de recibirla.

Cuando armo mi política de métricas las cosas que elijo medir están sumamente relacionado con la triple 3. Además, cuando no tenemos equilibrio de la triple restricción lo primero que se sacrifica es la calidad.

Existe una técnica denominada **GQM** que ayuda a derivar las métricas a partir de los objetivos de negocio. Para llegar a ese objetivo tenemos que hacernos ciertas preguntas y de allí derivamos las métricas.

Por último, el enfoque tradicional tiene foco en las 3 variables y acá podemos ver:

- **Producto**
  - Miden al software. Requerimientos es lo más optimo. Calidad, satisfacción, cumplimiento.
    - No se pueden medir directamente. Tenemos que medir lo de alrededor.
      - Porcentaje de requerimientos que el cliente esperaba cumpliste. (Calidad)
  - Directamente hablar de producto
  - Tamaño del producto
  - Defectos
- **Proyecto**
  - Tiempo calendario
  - Esfuerzo
    - Relacionado con la 3 restricción. Buscamos dar visibilidad de la situación de cómo estamos funcionando para entregar el producto o servicio.
  - Privada, y solamente visible para los involucrados en el proyecto.
    - No va en contra de la transparencia porque se debe poder acceder de las puertas para adentro
  - Hacer publica métricas de proyecto es muy difícil despersonalizar.
- **Proceso**
  - Las métricas anteriores se consolidan para crear métricas de proceso que sean públicas para toda la organización.
  - Se utiliza la extrapolación de la experiencia
  - Son generales no se habla de ningún proyecto o producto en particular.
  - Denota comportamiento organizacional y hablan de la organización.
  - Son públicas.
  - Se utilizan fundamentalmente para mejorar el proceso.

## Ágil:

La visión cambia. No queremos tener actividades específicas para la toma de métricas, sino que buscamos que este integrado que sea una consecuencia del trabajo.

Se cambia un poco el tablero y está bien.

Se busca lo mínimo porque en ágil ya se establece una métrica por excelencia que es que la mejor métrica para medir el progreso es el software funcionando.

Sobre esto el agilismo arma sus métricas (3)

No hay extrapolación de experiencia y su foco está en el producto.

Las métricas que asume el agilismo para medir su framework son las siguientes. Sin embargo, debemos de tener en cuenta agregarles las métricas de producto indispensable como la de defectos:

- Velocidad: Mide cuanto SP le entregamos al PO y este nos lo acepto.
    - Se mide en SP
    - Se calcula al final de un sprint **no se estima**.
    - Es la métrica de ágil porque mide software funcional.
- Capacidad: Determinar a cuanto nos podemos comprometer como equipo realizarlo para el próximo sprint.
    - Mide en hs ideales o SP
        - Los equipos no tan maduros utilizan horas ideales porque desagregan las US en tareas cosa que los maduros no suelen usar tanto por eso utilizan los SP.
    - Cuanto trabajo nos comprometemos para el próximo sprint.
        - Define mi compromiso
        - Para formar el sprint backlog.
    - Consumo interno. Se usa en el sprint planning.
- Running tested features
    - No se usa mucho. Porque mide cantidades absolutas.
        - No tiene en cuenta la complejidad que puede llegar a tener una feature.
    - Cantidad de características de software funcionando
    - Contando cuantas características entregue por sprint.

* Velocidad y capacidad se miden en el contexto de un sprint.

La velocidad -> Producto y capacidad -> Proyecto.

No hay métricas de proceso porque la experiencia no se extrapola. Entonces no tiene sentido.

## Kanban o Lean

Método para la gestión de la mejora de procesos. Por lo tanto las métricas medirán proceso. Que tan bueno somos trabajando en el proceso.

La unidad básica de medición es para cada pieza de trabajo que pasa por el tablero.

Tiene un foco en medir el proceso.

Las métricas son:

### Lead Time

Le sirve al cliente porque es el tiempo que tarda en entrar una funcionalidad al backlog y que se la entrega al cliente.

Desde que te lo pedí hasta que lo entregaste.

Suele medirse en días de trabajo y mide el ritmo de entrega.

### Cycle time

Sirve a los desarrolladores.

Es el tiempo que tardo desde que salió del backlog y se lo entrega al cliente. Mide en días de trabajo o esfuerzo

Mide mi ritmo de producción.

Ritmo de terminación: Desde que yo empecé a trabajar. Medio sesgada.

Si hay mucha diferencia entre lead y cycle no podemos estar satisfecho con nuestro cycle.

### Touch time

Toma las columnas de trabajo solamente (aquellas donde fue tocado).

Mide el tiempo en cual una pieza de trabajo fue realmente trabajada por el equipo.

> $$Touch\ Time \leq Cycle\ Time \leq Lead\ Time$$

### Eficiencia del proceso

Cuanto tiempo es el que realmente se trabajó. Mas cerca de uno es más eficiente.

$$\%\ Eficiencia\ ciclo\ proceso = Touch\ Time\ /\ Elapsed\ Time.$$

Porque el touch time y cycle time son iguales. Es casi imposible, pero podemos tener valores cercanos al 1.

Métricas más específicas de Kanban vinculadas a medir nuestra entrega de calidad respecto a servicios:

- Expectativa de nivel de servicio que los clientes esperan
- Capacidad del nivel de servicio al que el sistema puede entregar.
- Acuerdo de nivel de servicio que es acordado con el cliente.
- Umbral de la adecuación del servicio el nivel por debajo del cual este es inaceptable para el cliente.

El SLA se hace relacionado a los defectos. Estableciendo los tiempos máximos que un equipo puede resolver un defecto según la severidad.

Independientemente del foco elegido **tenemos que siempre medir el producto**. Ya sea haciendo por tamaño o por defectos.

> En defectos se mide cobertura. Cuanto hemos probado de todo lo que se ha entregado. %CosasProbadas/TotalProducto. Se debería tener un mínimo

Además, buscamos siempre automatizar lo más posible la recolección de las métricas. Además, deben tener una línea clara para saber para qué va a ser utilizada.

Debemos de definir en el plan de proyecto que métricas vamos a tomar. El líder de proyecto se fijará si hay métricas a nivel organizacional y las tomara y después elegirá la que cree que es mejor.

Definir la métrica hay que definir de donde voy a sacar los datos y los cálculos que voy a hacer.

Existen errores en todas las métricas y que nos permita obtener una visión objetiva y alejarnos del 90/90 y nos alejamos de las percepciones de la gente.

Cualquier métrica siempre la tomare en el ámbito del proyecto. El ámbito de captura de la métrica es en el proyecto.

# Calidad

Es un concepto subjetivo relacionado con expectativas y necesidad.

Es un concepto complejo y de facetas múltiples. Existen varias vistas, capaz lo más interesante para nuestro ambiente son:

- **Producto:** Sugiere que la calidad tiene que ver con las características inherentes (funciones y características) de un producto
- **Valor:** Mide el acuerdo con lo que el cliente está dispuesto a pagar por un producto.

También tenemos la **calidad de diseño** que se refiere a las características que los diseñadores especifican para un producto. En ISW se incluye que el mismo las funciones y características especificadas en el modelo de requerimientos.

---

Todos los aspectos y características de un producto o servicio que se relacionan con habilidad de alcanzar las necesidades.

---

Podemos entenderlo como cumplir con las necesidades del cliente y su satisfacción. Siendo la misma:

satisfacción del usuario = producto que funciona + buena calidad + entrega dentro del presupuesto y plazo

Los factores de la calidad según ISO son:

- Funcionalidad
- Confiabilidad
- Usabilidad
- Eficiencia
- Facilidad de recibir mantenimiento
- Portabilidad

De todos se habla de grado o de la facilidad (últimos 2 casos) de realizar algo.

Tenemos que entender nunca podremos hacer software completamente de calidad entonces hablamos de hacerlo suficientemente bueno. Entendiendo que el tiempo para llegar al mercado actúa contra la mejora calidad.

**Suficientemente bueno:** Contiene las funciones y características de alta calidad que desean los usuarios, pero tiene errores.

> Hay que entender que se puede correr el riesgo de causar un daño permanente a la reputación. Además existen ciertos dominios donde no puede haber errores conocidos porque es una negligencia (salud).

## Aspectos donde que influyen a la calidad

- Atrasos en las entregas
- Costos excedidos
- Falta cumplimiento de compromiso
- Requerimientos no claros
- Software no hace lo esperado
- Trabajo fuera de hora
- Fenómeno 90-90
- ¿Dónde está el componente?

## Aspectos donde se evidencia la calidad

La calidad no es solamente producto. Existe un área que rodea a todo lo que es calidad como proyecto o las personas.


Hablando de calidad en software debemos dejar contenta a demasiadas personas sus expectativas y sus necesidades.

Tenemos diferentes personas interesadas en los conceptos de calidad. Pero siempre son Cliente y Usuario con sus expectativas. Necesidades de la gerencia, equipo de desarrollo y mantenimiento.

## Principios de la calidad

Estos principios sustentan la necesidad de aplicar PPQA

Tiene que ser algo que se concibe y se realiza desde el momento cero.

- Calidad no se inyecta ni se compra, está embebida
- Esfuerzo de todos
- Personas son la clave
- Se necesita sponsor a nivel gerencial
- Liderar con el ejemplo
- No se puede controlar lo que no se mide
- Simplicidad
- PPQA planificarse
- El aumento de las pruebas no aumenta calidad
- Debe ser razonable para mi negocio.

## Visión

Hay diferentes visiones que están basado en las necesidades y expectativas de las personas.

La visión trascendental: Es una visión para lograr cosas grandes y es una motivación y motor para avanzar.

### Calidad programada

Expectativa cuando uno empieza a trabajar en un producto sobre la calidad que uno espera del mismo.

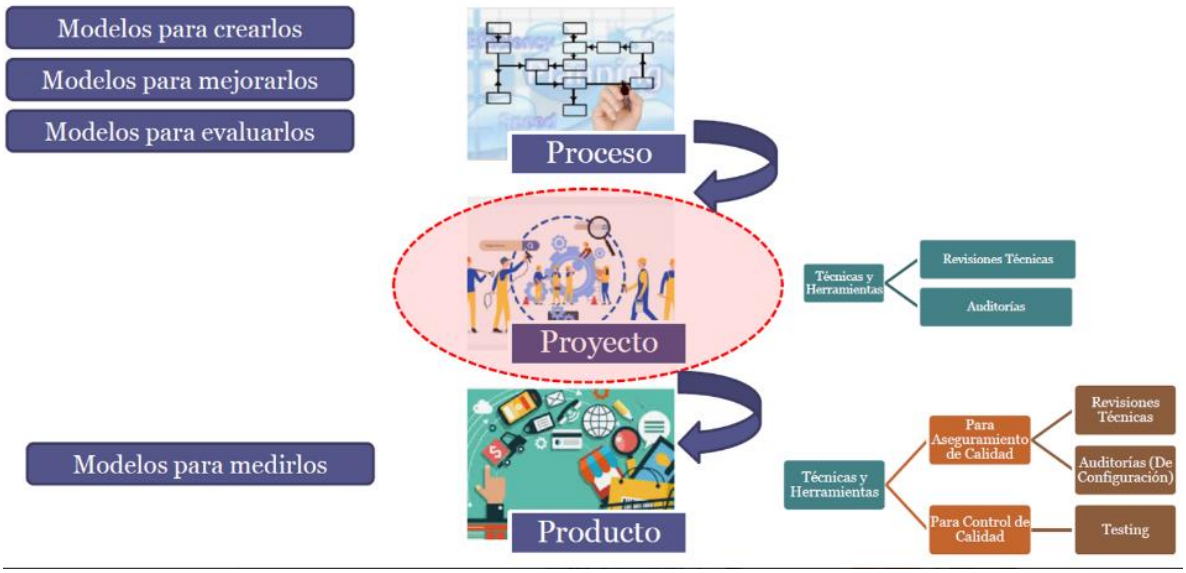
### Calidad necesaria

La programada está relacionada con esto. Es lo mínimo que el producto debe tener para satisfacer los requerimientos de usuario.

### Calidad realizada

Lo que realmente hicimos nosotros con el producto.

La intersección de las tres calidades es la calidad esperada. Mientras más grande sea la intersección más cubrimos dicha calidad. Lo que queda por fuera termina siendo insatisfacción.




La calidad existe en el producto, software y proyecto.

**Calidad en el software:**

Para realizarlo necesitamos una guía o estructura que es el proceso. Conociendo 2 enfoques lo definido y lo empírico. Nos basamos en modelos para poder definir procesos para una organización.

El propósito de los modelos que aparecen para mejorar los procesos como lo son el IDEAL o SPICE es darnos un marco de referencia para mejorar nuestro proceso.

Estos son los modelos para mejorarlos. Los modelos para evaluarlos intentan ver el grado de adherencia de los procesos a los modelos que decidimos tomar.

Entendemos que el proceso sigue siendo una **definición teórica** de cómo la gente debería hacer las cosas. Por lo tanto, cuando se instancian estos lo realizan dentro del contexto de un proyecto.

Durante la ejecución de un proyecto incluiremos actividades para ver como se están haciendo las cosas en el proyecto. Asegurándonos que estoy haciendo lo que dije que haremos en el modelo. Es acá donde se integra el concepto de calidad.

Hablamos de contexto porque es donde realmente aplicamos las formas de hacer calidad. Si bien podemos hablar calidad en producto, proceso ambas calidades se verán aplicadas en el proyecto.

**Técnicas y herramientas**

En el proyecto es el ámbito donde uno trabaja para hacer producto.

Las revisiones técnicas y auditorias son técnicas para asegurar la calidad en un proyecto. En la primera se tiene la característica que se realiza por pares con formación técnica similar. Por otro lado las auditorias son objetivas e independientes, se revisa si hay desviaciones respecto a lo que se comprometió, se realiza sobre el proyecto.

Una de las cuestiones que nos hace ruido SCRUM es con el tema de las auditorias.

Por otro lado, resuelve la identificación de oportunidades de mejoras del proceso en un evento particular que es la **restrospective**.

> Esta ceremonia está centrada principalmente en ver oportunidades de mejoras en el proceso. Lo hacemos en el proyecto porque es allí donde está instanciado y donde podemos aplicar mejoras.

Acá también insertare tareas para asegurar y controlar la calidad del producto.

Para el producto tenemos técnicas para asegurarnos la calidad y para controlarla.

- Aseguramiento de calidad
    - Revisiones técnicas
        - Se revisa varios aspectos del producto NO solo código

        - Nunca se discute a la persona
    - Auditorias de configuración
        - Funcional -> Valida
        - Física -> Verifica
        - Las dos se realiza a un tipo de producto de la línea base. Sobre producto
- Control de calidad
    - Testing

En SCRUM la ceremonia que tiene foco en el control del producto es la sprint review.

La diferencia principal entre la visión de lo tradicional y lo ágil radica en el concepto de empirismo y definidos.

En donde lo tradicional y definidos se guían con un principio o base la cual es que:

“Un proceso que tiene calidad y es respetado por las personas involucrados va a generar siempre un producto de calidad.” Entonces hablamos de procesos definidos que nos generan la calidad.

No significa que esté mal, solamente que no podemos ser tan rígidos en una industria intangible. Un proceso con calidad por si solo no me asegura que el producto tenga calidad.

En el caso de los empíricos la calidad del producto depende de la capacitación de la persona. Nuevamente los empíricos tienen el foco en la persona y no en el proceso.

El foco es distinto cuando tratamos de asegurar la calidad en el proceso o en el producto.

### Calidad de producto

No se puede sistematizar y no existen modelos en la industria que logren evaluar la calidad. Esto sucede porque es muy subjetiva e influyen variables como las características del producto, contexto y del cliente y usuario.

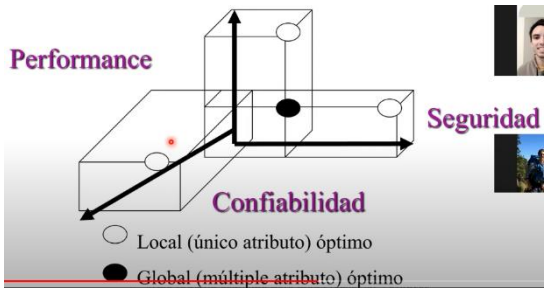
Todos los modelos que existen como el de ISO 25010 (req no funcionales) son teóricos, debemos de ponerlo en práctica para cada uno de los productos. Siendo esta calidad muy compleja de acreditar por lo anterior.

Podemos hablar de modelos, pero estos se definen en términos generales:

**Barbacci/SEI**

Donde nos dice que evalúa la calidad en base a 3 cosas y la calidad está definida como el equilibrio que mejor le quede al cliente.

- Performance
- Seguridad
- Confiabilidad

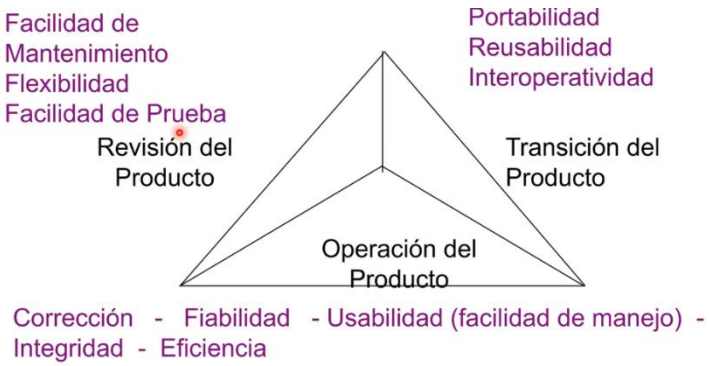



**MCCAL**

La calidad tiene 3 grandes agrupadores que son factores determinantes que tienen que ver con la visión de calidad

Establece ciertos factores los cuales son:

- Operación del producto:
  - Corrección
  - Confiabilidad
  - Eficiencia
  - Integridad
  - Usabilidad
- Revisión del producto:
  - Facilidad para recibir mantenimiento
  - Susceptibilidad de someterse a pruebas
  - Flexibilidad
- Transición del producto:
  - Portabilidad
  - Reusabilidad
  - Interoperabilidad

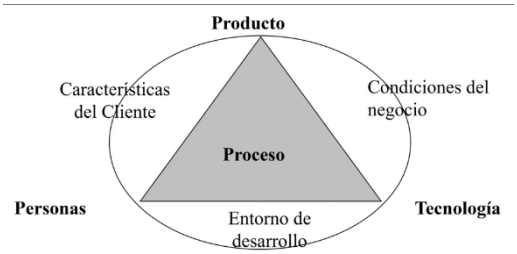



Es muy difícil y a veces imposible desarrollar mediciones directas sobre estos factores por lo que mucha de estas solo se mantienen de manera indirecta

## Calidad de proceso

Bajo la premisa que no siempre un proceso sirve excelente para todos los productos. Tenemos que poder encontrar un proceso útil a un trabajo especifico.

Insiste con la calidad del proceso porque el único factor controlable que tenemos es el proceso.




Veremos que ni las personas, tecnologías o requerimientos (producto) pueden controlarse por nosotros.

La única posibilidad de tener una sensación de control es sobre un proceso definido.

## Proceso en PPQA

No es solamente las tareas escritas. Es un triángulo de 3 cosas.

- Tenga escrito los lineamientos de cómo vamos a trabajar
- Las personas con habilidades, entrenamiento y motivación
- Mejor soporte de herramientas automatizadas y equipamiento.

Es una visión más amplia a la ya conocida. Lo mismo pasa cuando definimos el proceso.

Abrimos la caja y empezamos a hablar de etapas o WF que a su vez se dividen en tareas que tendrán roles y entradas y salidas.


Ingeniería es la parte de diseño, requerimientos y análisis.

Debemos incorporar disciplinas de gestión y de control que son las 3 que nos encontramos abajo del todo.

Estas son transversales y empiezan desde el momento cero del proyecto. Se hacen mientras se hace software.

Cuando hablamos de un proceso deberíamos poder saber como vamos a hacer cada una de estas cosas.

## Disciplina de Aseguramiento de calidad de software

Es insertar en cada una de las actividades acciones para poder detectar lo más temprano posible las oportunidades de mejora sobre el proceso y producto.

Es algo cultural que debe estar inserto en la cabeza de todos en donde se ve a la calidad como una responsabilidad de todos y cada uno de los involucrados.

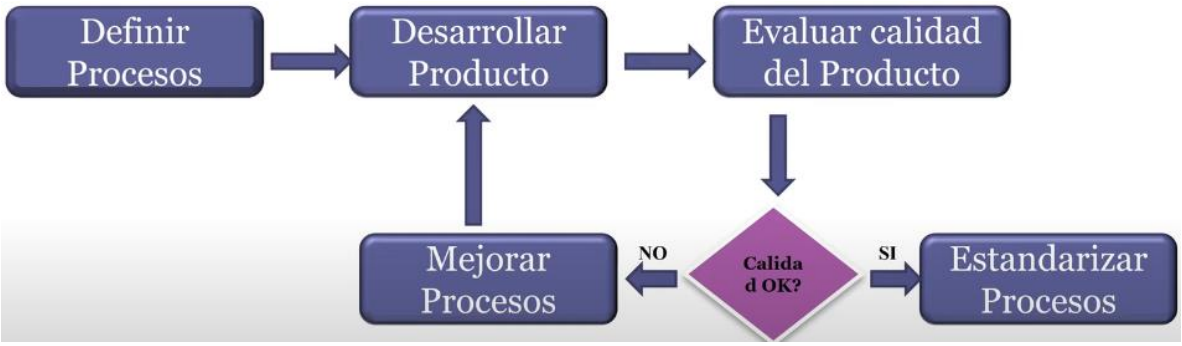
Para materializarlo hay consideraciones como:

- No debe reportar la gente que hace calidad al mismo gerente que reportan proyectos.
    - Esto quita independencia y objetividad
    - Reporte independientemente al de proyecto y lo más cerca al nivel más alto.
- No debería haber más de una posición entre la gerencia de primer nivel y la de calidad.
    - Reporte debe ser directo a la gerencia de primer nivel y lograr la independencia.

Además, las personas que trabajan en calidad deberían:

- Aseguramiento de calidad:
    - Nos establece la infraestructura de apoyo a los métodos solidos de la ISW.
    - Consiste en un conjunto de funciones de auditoria y reportes para evaluar la eficacia y completitud de las acciones de control de calidad.
    - Vamos a definir estándares, procesos y los modelos contra los cuales haremos comparaciones con la revisiones técnica y auditoria.
- Planificar la calidad
    - Decidir en que momento que proyecto le realizo que actividad de calidad.
    - Tenemos que el plan de testing es separado, pero se encuentra referenciado acá.
- Control de calidad:
    - Conjunto de acciones que ayudan a asegurar que todo el producto del trabajo cumpla con sus metas de calidad.
    - Seria ejecutar las actividades de calidad para ver en que situaciones está el proyecto.

Entonces aseguramiento de calidad es insertarse en el proceso de desarrollo mientras se esta ejecutando. No esperar a que este terminado para agregarle la calidad.




Parte en definir los procesos que se someten a evaluación desarrollando software. Se evaluara la calidad y si el producto esta aceptado significa que el proceso que se utilizo está bueno y lo estandarizamos, caso contrario lo mejoro.

En la estandarización para la organización se hace ruido nuevamente con lo agil porque la experiencia le sirve a un solo equipo. Entonces no nos sirven procesos estandarizados. Ahora en los definidos si nos sirve, porque entienden que da visibilidad y se puede extrapolar en el resto de los equipos.

## Modelos de calidad

En una empresa tenemos que hacer software con una unidad de gestión la cual es el proyecto. Ese debe generar un producto. Por lo general hablamos de que se genera de manera iterativa e incremental a lo largo del CV del producto.

Para generar un producto los proyectos utilizan diferentes procesos aplicados o adaptados al proyecto en si. De esta manera vemos los 3 ámbitos de un software.

En estos 3 ámbitos siempre existe la necesidad de que sea el mejor posible. Para ello utilizaremos modelos de mejora.

---

**Modelo:** Representación simplificada y abstracta de un sistema, proceso o componente de software permitiendo comprender, diseñar o comunicar cualquier cosa del software o su desarrollo.

---

Bajo este concepto nos encontramos con 4 modelos durante estos 3 ambientes:

- Creación de software
  - Guían el proceso de desarrollo de software estableciendo una estructura y secuencia de pasos para concebir y construir un producto de software.
- Mejorar el software
  - Optimizar y perfeccionar continuamente el proceso de desarrollo de software y la calidad del producto.

- Evaluarlo
    - Valoran la calidad y el rendimiento del software desarrollado, así como para identificar posibles problemas o deficiencias.
- Medirlo
    - Los modelos de medición se centran en cuantificar y registrar datos específicos relacionados con el software y el proceso de desarrollo.

## Mejora

Esquema o plantillas o recomendaciones de trabajo para encarar un proyecto.

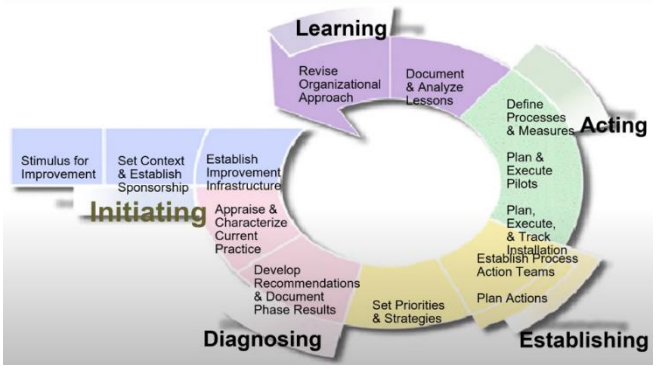
El propósito es analizar el proceso de la organización y armar un proyecto cuyo objetivo sea un proceso mejorado que vuelve a la organización.

Siguiendo la idea de que un proceso mejorado -> mejoro indirect producto

Nos sirve para definir un proyecto que ayude a mejorar el proceso que la empresa tiene.

### IDEAL

Modelo que nos da un contexto a nosotros para crear un proyecto cuyo resultado es un proceso definido. Nos plantea como nosotros podemos encarar un proyecto de mejora de un proceso.




Se empieza por buscar un sponsor dentro de la organización. Es un paso muy importante porque nunca son críticos los proyectos para la organización.

Este paso es importante no solo por el dinero sino para la disponibilidad de recursos que obtengamos.

Luego realizamos un diagnostico sobre lo que tenemos en el momento. ¿Qué hacemos bien y que mal? Donde estamos y hacia dónde queremos ir (Este es el análisis de brecha)

En base a esto definimos planes de acción con lo que tenemos que hacer para lograr el objetivo que me plantee.

Donde entran los modelos de calidad porque son ellos los que nos darán las pautas para seguirlos y saber hacia dónde apuntamos.

Luego de haber definido los planes de acción y que fueran aprobados, definimos los procesos. Escribimos Roles, actividades y todo lo que regula el proceso.

En este punto es importante aclarar que los modelos te dicen lo que tenes que hacer para obtener el proceso no el como. Entonces cada organización definirá su propio como.

Lo ponemos a prueba en uno o varios de proyecto para tener un control y ver si salen bien.

Y si sale bien se estandariza.

**CMMI**

Empezó en el software y se expandió para diferentes áreas. Luego se integran todas las áreas bajo un solo modelo -> CMMI

Es descriptivo y no prescriptivo.

- Se ofrece buenas practicas que dicen como llegar a un objetivo
- Elegimos que practicas tomamos para garantizarlas.
- Cuando quiero realizar una evaluación CMMI el modelo me sirve para ver que practicas debo incluir en el proceso para lograr el objetivo.

El CMMI entonces nos dice los objetivos que debemos de alcanzar y nos da buenas practicas para lograrlo. NO dice como hacer las cosas.

Una vez aplicado CMMI y obtenido el resultado nosotros podemos proceder a modelos de evaluación para evaluar lo que hicimos.

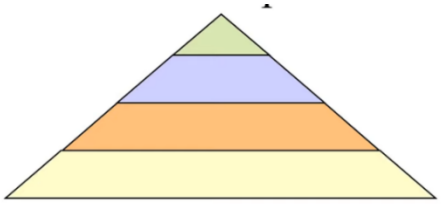
> SCAMPI es el modelo de evaluación o auditoria para darle un nivel de madurez de CMMI.

Cuando CMMI se integro se unieron 3 puntos fuertes

- DEV
  - Modelo para desarrollo de software
  - Guía de mejores prácticas que me ayuda a definir procesos de desarrollo de software
- SVC
  - Para servicios
  - Provee la guía para entregar servicios internos o externos
- ACQ
  - Adquisición
  - Modelo que guía a la empresa respecto de cómo manejar la adquisición de productos o servicios.

## Representaciones

Existen dos representaciones para CMMI



### Por etapas

En donde una empresa puede ser madura o no serlo.

Existen 5 niveles el primero se lo considera inmaduro mientras que del 2 al 5 son maduras. La madurez es una relacion directa con la capacidad para cumplir con sus objetivos. Mejorando la calidad y reduciendo riesgos.

La ventaja de esta representacion es que se habla de organización y se sabe que cuando nos referimos a una organización nivel X lo que podemos esperar de ella. Proveyendo una comparación entre ellas.

Los niveles nos definen que areas de proceso debo cumplir yo para lograr ese nivel. Ademas estos son acumulativos y tienen una relacion directa con la productiva y calidad e inversamente directa con el riesgo.

El nivel 1 nos explica que no tiene ningún tipo de visibilidad sobre el proceso. Existe mucha incertidumbre respecto a costos y tiempos y mucho riesgos. Entonces hablamos de una gestión de crisis.

- Administración de Requerimientos.
- Planeamiento de Proyectos.
- Monitoreo y Control de Proyectos.
- Administración de Acuerdo con el Proveedor.
- Medición y Análisis
- Aseguramiento de Calidad del Proceso y del Producto.
- Administración de Configuración.

El nivel 2 es la que incluye a la gestión y el soporte. Resuelve el problema mayor del nivel 1. Obteniendo un proceso administrado y organización con la capacidad de gestión y soporte.

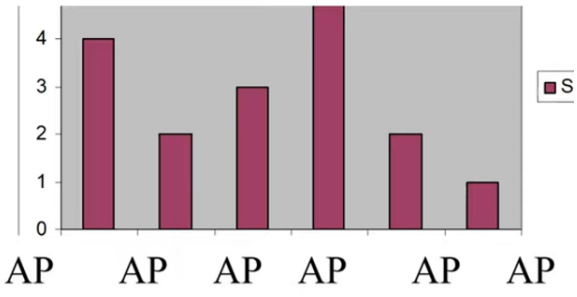
Lo rojo es porque es opcional a si contrato o no proveedor.

Cuando obtenemos un nivel 2 sabemos que tiene madurez para administrar sus proyectos y que el resultado va a ser un producto que sabremos que vamos a esperar.

El nivel 3 se encarga de la ingeniería haciendo foco en la triple P

Nivel 4 es una gestión cuantitativa. Pretendiendo trabajar bajo control estadístico.

**Continua**




Se eligen áreas de proceso dentro de las que el modelo te ofrece.

Elijo yo cuales son los procesos que quiero mejorar (diferencia).

En vez de medir la madurez de toda la organización se mide la capacidad de un proceso en particular.

Los niveles van de 0 a 5 (6 niveles) donde 0 significa que no se ejecuta.

**Auditoria CMMI**

Entonces si yo defino que quiero un proceso tomando como referencia nivel 2.

Encaro un proyecto para obtenerlo

Llamo a una evaluación (SCAMPI), antes debo haberlo aplicado al proceso y generar evidencia:

- Subjetiva: Donde se pregunta a los involucrados que hacen y como lo realizan
- Objetiva: Voy y me dijo en la evidencia que se deja por la ejecución del proceso. Artefactos.

Se comparan las dos y luego contra el modelo para ver si se cumple o no. Tengo que cumplir todas sino no soy nivel 2.

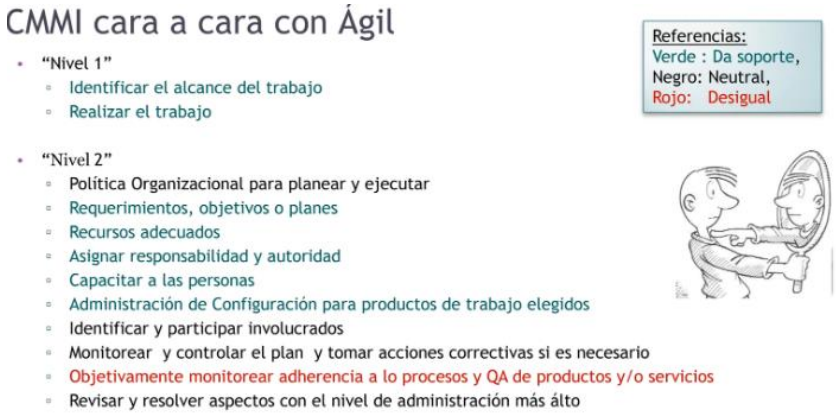
**CMMI y Ágil**

Como CMMI me dice que es lo que tengo que hacer y no como yo puedo realizarlo con las practicas que quiera (Ágil). Lo que tenemos que entender es que ambas van a ceder un poco.

Ágil es un cultura de trabajo que define practicas y CMMI es un modelo de mejores practicas en términos de objetivos que define que tenes que hacer no como hacerlo.

Allí está la unión.

Lo que más duele es lo rojo donde apunta a las auditorias.



En este aspecto CMMI nos dice que tenemos que definir un proceso y después cumplirlo. Entonces cuando hay una evaluación no sólo se mide que tengas los productos, sino que además hayas construido el proceso.

En ágil no hacemos eso. En ningún momento se evalúa en detalle si cumpliste o no con el proceso que definiste.

En una organización que quiere CMMI tiene que existir actividades que se encarguen de controlar si cumplimos o no con el objetivo.

# Testing

Las pruebas de software se realizan en el contexto de PPQA.

---

Proceso **destructivo** para tratar de encontrar defectos que se asumen su existencia en el código.

---

- Proceso destructivo porque el objetivo es encontrar defectos.
  - Destructivo es porque implica ir con una actitud negativa para demostrar que algo es incorrecto.
- Asumimos la existencia
  - Por eso demostramos. No verificamos que no haya defectos
  - Objetivo es encontrarlos.
- Tenemos éxito cuando encontramos defectos.
  - Si no lo encontramos no significa que no existan solo que no los encontramos.
  - Se lleva un 30 y 50% del costo

## Error vs Defecto

La diferencia principal radica en el momento en que lo descubrimos

Un **ERROR** se descubre en la misma etapa en el que fue creado. Es decir si es un error de implementación se descubre durante la implementación.

Mientras que un **DEFECTO** es una falla que se traslada a etapas posteriores. Es este el que se busca y encuentra en el testing.

## La prioridad y la Severidad

**SEVERIDAD**

Gravedad del defecto encontrado.

Esta es una clasificación tentativa y no es la única.

1. Bloqueante: No puedo seguir con el CP

2. Crítico: Compromete la ejecución del CP en cuanto a sus resultados.

3. Mayor

4. Menor

5. Cosmético

**PRIORIDAD**

Tiene que ver con la urgencia para resolver los defectos.

1. Urgencia

2. Alta

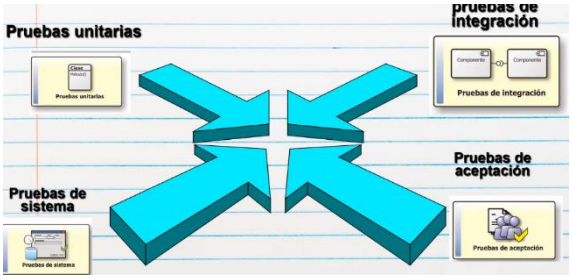
3. Media

4. Baja

Entonces nosotros siempre encontramos defectos porque los buscamos siempre en la etapa de testing.

## Niveles de pruebas

1. Unitarias
2. Integración
3. Sistema
4. Aceptación




### Unitarias

Pruebas donde pruebo un componente individual.

Es algo acotado que tiene que ver con el desarrollo que estoy haciendo, lo realiza el mismo *desarrollador* (única).

Se hace con respecto a componentes o aspectos específicos del software

Son las únicas que encuentran errores.

### Integración

Implica integrar los componentes ya pasados por las pruebas unitarias y fijarnos si funcionan correctamente juntos.

Se realiza de manera incremental sumándole los distintos componentes y ver como se conectan.

Siempre se deben de probar los críticos de la manera más rápida.

Podemos hasta llegar a probar una funcionalidad completa.

Se habla de interfaces y es llevado a cabo por el *Tester.*

### Sistema

Más amplia y no pruebo solamente la integración, sino que también el sistema en toda su escala.

Prueba la aplicación funcionando como un todo, buscando asegurarnos que el sistema en su totalidad funcione correctamente.

Acá entran los requerimientos no funcionales como lo son el performance.

Necesitamos que el entorno de prueba sea el más parecido (ambiente) al de producción

*Tester*

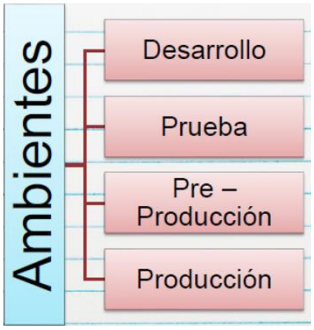
### Aceptación

Ejecutadas muchas veces por el *cliente* o *usuario final*.

En este punto nuestro foco no es encontrar defectos. El objetivo es establecer la confianza en el sistema. Independientemente de que el usuario pueda llegar a encontrar defectos.

El ambiente de pruebas tiene que ser el de producción.

## Ambientes




Lugares donde se trabaja para la construcción del software.

- **Desarrollo:** Lugar donde los desarrolladores escriben el código.
- **Prueba:** Se hacen las pruebas de integración, es un entorno parecido al de producción pero no tiene todas las características.
- **Pre-Producción:** Pruebas de sistema y a veces la de adaptación.
- **Producción:** Software funcionando y en operación. Se puede utilizar para las pruebas de aceptación si todavía el software no está en operación.

El problema viene con que es caro mantener y lograr ambientes parecidos al de producción porque tiene ciertas características como: Performance, concurrencia, seguridad, redundancia.

Por eso utilizamos lo más parecido para lograr hacer las pruebas que se requieran.

## Caso de prueba

---

El caso de prueba es un conjunto de condiciones o variables que nos va a permitir determinar si el software está funcionando correctamente o no.

---

Cuando decimos que es un conjunto de condiciones o variables, nos referimos a que tengo definido:

- Acciones a ejecutarse
- Valores específicos para realizarlos

Se especifican los datos y valores especifico para cada paso del CP.

Una buena definición nos ayuda a reproducir lo que queremos. Es decir que el objetivo no es solamente determinar si un software está funcionando o no, sino que también nos ayuda a reproducir los defectos.

Un caso de prueba es la unidad de la actividad de la prueba y consta de 3 partes:

- Objetivo: Característica del sistema a comprobar
- Datos de entrada y de ambiente: Datos a introducir al sistema que se encuentra en condiciones preestablecidas
- Comportamiento esperado: La salida o la acción espera en el sistema de acuerdo a los requerimientos del mismo.

El inconveniente es que no puedo definir los CP de manera infinita, necesito tener un mecanismo para lograr definir la menor cantidad de CP pero que permitan cubrir el software de la forma más completa posible.

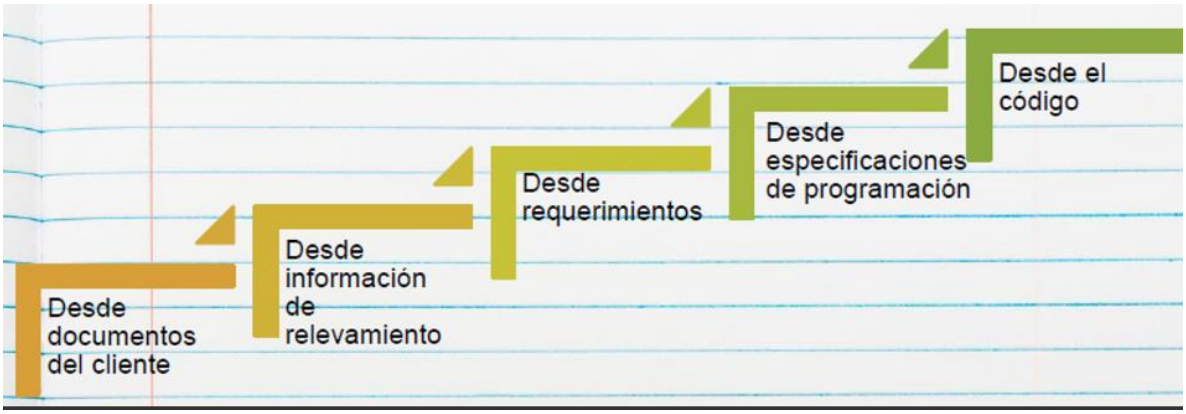
### Derivando CP

¿De donde obtengo yo los CP?

Mientras mayor documentación tengamos es mejor y más fácil.

Si tenemos una especificación de requerimientos bien detallada y casos de usos definidos, la derivación de CP se hará casia automática.

Si los requerimientos están descriptos poco preciso esa precisión se traslada a los CP.




De todos estos lugares puedo derivar, siendo el peor el código porque no hay casi nada de información.

### Estrategias

Buscamos definir la menor cantidad de CP pero buscando lograr la mayor cobertura sobre el software.

Esto es porque el testing exhaustivo, completo, no es viable porque es imposible. Necesitaríamos años de testing.

Como sabemos que nunca vamos a poder cubrir el software en su totalidad, debemos de buscar las maneras de cubrir el software encontrando la mayor cantidad de defectos posibles.

Para estas estrategias nosotros nos basamos en varias cosas:

- En los limites, esquinas y bordes es normalmente donde se encuentran los bugs.
    - Apuntemos allí cuando definamos CP.
- Mi objetivo es abarcar la mayor cantidad de defectos con el menor esfuerzo.
- Lo utilizamos porque el tiempo y $ es limitado y quiero pasar por la mayor cantidad de funcionalidades con la menor cantidad de CP.

**CAJA NEGRA**

Definimos las entradas y el resultado esperado. No sabemos lo que pasa en el medio.

Frente a ciertas entradas esperamos obtener un resultado especifico. Como se llega allí no nos interesa, si nos interesa comparar el resultado obtenido con el esperado.

Tenemos 2 tipos

- Basados en especificaciones
    - Partición de equivalencia
        - Partición de equivalencia es un conjunto de valores que puede tomar una condición para el cual para cualquier miembro del conjunto es equivalente.
    - Análisis de valores limites
- Basados en experiencia
  Tiene que ver con el testing que lo realiza un tester experimentado que sabe donde encontrar los defectos por su experiencia.
    - Adivinanza de defectos
    - Testing exploratorio

**CAJA BLANCA**

Mira adentro del código. Revisamos la estructura del código y revisar si hay alguna parte del código donde exista un defecto.

Vemos la estructura del software o componente de software.

- Enunciados
- Sentencias
- Decisión
- Condición
- Decisión/condición
- Múltiple

**Importante**

Hay muchas técnicas para obtener CP pero ninguna es completa. Las distintas técnicas atacan distintos problemas. Entonces es mejor coordinarlas.

Sin requerimientos todo es mucho más difícil.

Los CP se pueden empezar a definir desde el momento que tenemos los req.

### Condiciones de prueba

Contexto del sistema que tengo que tener en cuenta para poder hacer la prueba.

Son importante y tienen que ver con las condiciones de entradas

Indicar información adicional requerida.

## Ciclo de prueba o de test

Un ciclo implica la ejecución de todos esos casos de prueba en una versión del sistema a probar.

Ósea realizamos el primer CP, mando los defectos a corregir y luego vuelvo a hacer otro ciclo para controlar.

Esto se repite hasta que llegamos a un **criterio de aceptación.**

> El mismo tiene que estar definido con el cliente, y es necesario definirlo porque es imposible probarlo hasta que no existan defectos.
>
> Puede ser cualquier cosa desde cant de ciclos hasta que no haya más defectos de algún tipo de severidad.
>
> Se debe realizar una evaluación del nivel de riesgo y los costos asociados al proyecto.

Se deben ejecutar tantos ciclos como sean necesarios para la aprobación del criterio.

### Regresión

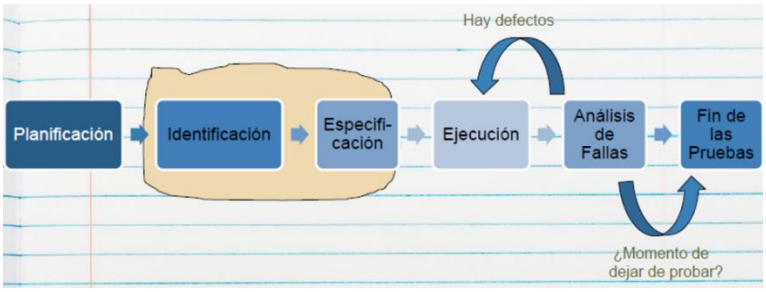
Es un concepto interesante en testing puesto a que cuando corregimos un error siempre ingresan más defectos.

Entonces lo que hacemos es que en el próximo ciclo no nos basta o sirve controlar solamente lo que corregimos tenemos que volver hacer testing a todo el producto.

Pero no siempre, se debe controlar.

## Proceso de pruebas

Cuando hablamos de un proceso de prueba lo estamos haciendo desde un contexto de lo definido.




Primero se empieza con la planificación y se arma el plan de pruebas.

Luego empieza el diseño de CP con la identificación y la especificación de los mismos.

Luego se ejecuta. Esto puede ser automatizado, se deben crear los datos de los CP y ejecutarse.

Terminando se realiza el análisis de fallas, chequeo que sea o no el resultado esperado. Si no lo es vuelvo a la ejecución.

Por último si cumplo con el criterio se finaliza.

## Testing y ciclo de vida

Retomando los conceptos de verificación y validación:

- Verificación: Estamos construyendo el sistema correctamente. Libre de defectos
- Validación: Es el sistema correcto. Con respecto a lo que el cliente quiere.

Cuando hablamos de ciclo de vida queremos hacer que las actividades de testing estén involucradas lo más temprano posible.

Una vez definido los requisitos yo ya puedo empezar a trabajar definiendo los CP. Por lo que no necesito código para hacer testing.

Cuanto antes empecemos es mejor porque nos da visibilidad de manera temprana de como van a ser probados los productos y disminuyen costos de correcciones.

- No se empieza cuando se termina de escribir código
- No es probar software que funciona
- No es calidad de producto o proceso
- No es el enemigo del dev

## Principios

- Muestra presencia de defectos
- Exhaustivo es imposible
- Temprano
- Agrupamiento de defectos

- Paradoja del pesticida
    - Si ejecutamos muchas veces el mismo CP las fallas pueden quedar ocultas.
- Dependiente del contexto
- Presencia de defecto asumida
- Un programa no debe probar su propio código
- Examinar software para probar que no hace lo que se espera
- No planificarlo sobre la suposición de que programamos bien

## Tipos de prueba

### Smoke test

Se realiza en la primera versión. Es una corrida rápida por todo el software en donde no se prueban casos alternativos ni rebuscados.

Para ver que no haya defectos enormes.

No es Sanity test en donde se corren todos los test para asegurarse que no hay fallas críticas.

### Testing Funcional

Ejecutar las pruebas de una manera relacionada a los requerimientos.

Vemos si se cumple con lo especificado allí.

Basados en requerimientos y procesos de negocios y se derivan automáticamente de los requerimientos funcionales.

### Testing No Funcional

Como lo realiza el sistema

Hay importancia en que los ambientes de prueba sean lo más parecido al entorno de producción.

- Performance Testing
- Pruebas de Carga
- Pruebas de Stress
- Pruebas de usabilidad,
- Pruebas de mantenimiento
- Pruebas de fiabilidad
- Pruebas de portabilidad

### Interfaces de usuario

### Performance

Medimos tiempo de respuesta y concurrencia

### Configuración

Verificar si en cierta configuración como trabaja el sistema. (que el soft funcione correctamente).

## TDD

Es un diseño conducido por CP. En donde implica que antes de implementar lo que pienso es que voy a probar y en base a eso implemento.

Primero escribo los CP, luego escribo el código necesario para que ese CP sea un éxito.

Hay muchísimo refactory porque hay mucha mejora del código.

Elijo requerimiento > Escribo CP > Implemento > Pasada la prueba lo factorizo > vuelvo a probar.

Se enfoca solamente en lo Funcional porque tienen que ver con el código.

# Auditorias

> Evaluaciones independientes de los productos o procesos de software para asegurar que se cumplan estándares, lineamientos, especificaciones y procedimientos. Basada en un criterio objetivo

Incluye documentación acerca de:

- La forma o contenido de los productos a ser desarrollados
- Proceso por el cual los productos van a ser desarrollados
- Como debería medirse el cumplimiento de estándares.

Actividad incluida dentro de las disciplinas de soporte. Implica evaluación independiente. Son un instrumento para el aseguramiento de calidad en el software.

Existen diferentes tipos de auditorías (3)

## Proyecto

- Valida el cumplimiento del proceso de desarrollo
- Responsable de evaluar si el proyecto se ejecutó en base a como dijimos que se iba ejecutar
- Apunta a ver el nivel de cumplimiento del proceso que como equipo nos comprometimos.

Se lleva a cabo de acuerdo a lo establecido en las PACS (Planeamiento de Aseguramiento de Calidad de Software) y este establece la persona responsable en auditar.

Verificar objetivamente las consistencia del producto a medida que evoluciona.

Inspecciones de software y revisiones de documentación de diseño y prueba deberían incluirse.

**Roles:**

- Auditado
    - Suele ser el líder de proyecto
    - Acuerda una fecha, proporciona la evidencia
    - Propone y comunica el cumplimiento de los planes de acción.

- Auditor
    - Objetivo y por fuera del proyecto
    - Acuerda la fecha, comunica alcance de auditoría, la realiza
    - Realiza reporte y seguimiento de planes de acción
- Gerente de SQA
    - Prepara el plan de auditoria
    - Calcula costos
    - Asigna recursos

El objetivo de la auditoria es verificar objetivamente la consistencia del producto a medida que evoluciona a lo largo del proceso de desarrollo. Determinando que:

- Interfaces de hardware y software consistente con la ERS
- Requerimientos funcionales de la ERS se validan en plan de verificación y validación de software.
- Diseño de producto satisface los requerimientos funcionales de la ERS
- Código es consistente con el DDS (Sistema de diagnóstico digital)

### Auditoria de configuración funcional

Valida que el producto cumpla con los requerimientos. Compara el software construido con los requerimientos de la ERS.

Asegurarnos de que el código implementa solo y completamente los requerimientos.

### Auditoria de configuración física

Valida que el ítem de configuración tal como está construido cumpla con la documentación técnica que lo describe. Permite trazabilidad y satisfacción de req.

Auditorias física compara el código con la documentación de soporte.

Las PACS deberían indicar la persona responsable de realizar la auditoria física.

## Proceso de Auditoría


**Preparación y planificación:** Planifica y prepara la auditoria en forma conjunta con el auditado y auditor.

**Ejecución:** Se pide documentación y se hacen preguntas. Buscando evidencia objetiva (lo que está documentado) y subjetiva (lo que el equipo expresa que hace).

**Análisis y reporte del resultado:** Se analiza la documentación, se prepara un reporte y se lo entrega al auditado.

**Seguimiento:** Dependiendo de cómo funcione el acuerdo entre auditado y auditor. Se puede hacer seguimientos de las desviaciones encontradas hasta considerarse resueltas.

## Checklist

Para que independientemente de quien realiza la auditoria el foco siempre sea el mismo.

- Fecha de la auditoría
- Lista de auditados
- Nombre del auditor
- Nombre del proyecto
- Fase actual del proyecto (si aplica)
- Objetivo y alcance de la auditoria
- Lista de preguntas.

## Resultados

- **Buenas prácticas:** Practica que se desarrollo mucho mejor de lo que se esperaba. Se encontró algo superador a lo que se esperaba. En el informe se comienza por acá.
- **Desviaciones:** Requieren un plan de acción del auditado. Cualquier cosa que no se hizo como el proceso indicaba que debía hacerse.
- **Observaciones:** Condiciones que deberían mejorarse pero que no requieren un plan de acción. Son cosas que el auditor advierte que no son desviaciones, pero son riesgosas.

## Métricas de auditoria

- Esfuerzo por auditoria
    - Cantidad de tiempo y recursos dedicados a la realización de auditoria
- Cantidad de desviaciones

    - Registra el numero total de desviaciones identificadas durante la auditoría.
- Duración de auditoría
    - Mide el tiempo total que lleva realizar la auditoria, desde la planificación hasta la presentación de los resultados. Útil para evaluar la eficiencia y capacidad de gestión.
- Cantidad de desviaciones clasificadas por AP de CMMI
    - Desglosa las desviaciones según las AP. Permite analizar fortalezas y debilidades

## Revisiones Técnicas

La verificación y validación.

Nosotros en un ciclo de vida completo iniciamos con las revisiones de los requerimientos y luego las de diseño y código hasta la prueba.

Debemos realizar validaciones (¿Estamos haciendo el producto que el cliente espera?) y verificaciones (¿Estamos construyéndolo correctamente?).

**Falla**: Error en un producto de trabajo

**Producto de trabajo:** Salida de cualquier actividad correspondiente al CV

Las Fallas existen por problemas de comunicación o limitaciones de memoria.

Se consideran severidades dentro de estas fallas.

### Principios

- Prevención es mejor que curar
- Evitar es más efectivo que eliminar
- La retroalimentación enseña efectivamente
- Priorizar lo rentable
- Olvidarse de la perfección
- Enseñar a pescar en vez de dar el pescado

Existen dos aproximaciones que son complementarias:

- Revisiones Técnicas
- Pruebas de software (Testing)

## Revisión técnica - Peer Review

La realiza un colega (misma capacidad formativa). El propósito es mejorar el software mediante la detección temprana de errores en cualquier artefacto que se genere.

Proceso estático de validación y verificación el cual no corrige errores.

El objetivo es introducir el concepto de verificación y validación. Buscamos evitar el retrabajo motivando a realizar un mejor trabajo.

Revisamos el artefacto no el autor de este. Debemos de brindar apoyo no culpar.

| Ventajas | Desventajas |
|---|---|
| • Pueden descubrirse muchos errores<br>• Pueden inspeccionarse versiones incompletas<br>• Pueden considerarse otros atributos de calidad | • Difícil introducir las inspecciones formales<br>• Sobrecargan al inicio los costos y conducen un ahorro sólo después de que los equipos adquieran experiencia<br>• Requieren tiempo para organizarse y parece ralentizar el proceso |

## Tipo de revisiones

### Formales

Tiene un proceso definido con roles

Tenemos las **inspecciones**.

**Objetivos:** Detectar y remover todos los defectos eficiente y efectivamente

**Atributos Típicos:** Proceso Formal, Checklists, Mediciones, Fase de verificación

**Actividad de garantía de calidad de software**

Tiene un proceso formal con roles. Utiliza un checklist que ayuda para saber que cosas controlar. Se toman métricas y se realiza un reporte de la revisión final de la inspección para analizar defectos encontrados.

Como objetivos tenemos que:

- Descubrir errores
- Verificar que el software alcanza sus requisitos
- Garantiza que el software ha sido representado según estándares.
- Conseguir un software desarrollado de manera uniforme
- Hacer que los proyectos sean más manejables.

Se lleva a cabo con una reunión y su éxito depende de la planificación.

| SON | NO SON |
|---|---|
| Forma más barata y efectiva de encontrar fallas<br>Proveer métricas al proyecto<br>Promover el trabajo en grupo<br>Proveer conocimiento cruzado<br>Método probado para mejorar calidad | Utilizadas para encontrar soluciones a fallas<br>Usadas para obtener la aprobación de un producto de trabajo<br>Evaluar desempeño de personas |

Roles que participan:

- Autor
- Moderador
- Anotador
- Lector
- Inspector

Proceso de inspección

1. **Planificación:** El moderador a pedido del autor planifica una inspección. Define lugar, tiempo y los roles
2. **Visión general:** Opcional, Autor realiza una descripción del producto a inspeccionar
3. **Preparación:** Preparación de cada rol para la reunión. Cada rol adquiere una copia del producto de trabajo que deberá leer y analizar para encontrar defectos.
4. **Reunión de inspección:** Equipo realiza un análisis de recolectar los potenciales defectos previos. Se lee el producto de trabajo y se anotan los defectos encontrados. Se concluye con la decisión de si se acepta o no el producto de trabajo y se realiza un informe detallando que se reviso, quien, lo que se descubrió y la conclusión
5. **Corrección:** Autor realiza las correcciones

6. **Seguimiento:** Dependiendo de la gravedad puede existir una re-inspección. Depende de la severidad del defecto se puede hacer inspecciones nuevamente o no.

**Informales**

Cuando no existe un proceso de como realizarlo.

Tenemos los **Walkthrough**

**Objetivos:** Mínima Sobrecarga; Capacitación de desarrolladores; Rápido retorno.

**Atributos Típicos:** Poca o ninguna preparación; proceso informal; no hay mediciones; no FTR (No es una revisión técnica formal)

**Es una técnica de análisis estático** en donde un programador o diseñador dirige miembros del equipo de desarrollo y otros partes interesados a través de un producto de software. Se comentan y preguntan acerca de posibles errores, violaciones de estándares etc.

No existe un proceso formal, son reuniones informales de colegas que se juntan a debatir correcciones a aplicar al producto de trabajo. No hay control del proceso.

No obtiene métricas pero es muy elegida en los enfoques agiles.