Resumen Ingeniería y Calidad de Software

No se encuentran entradas de índice.

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. 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**

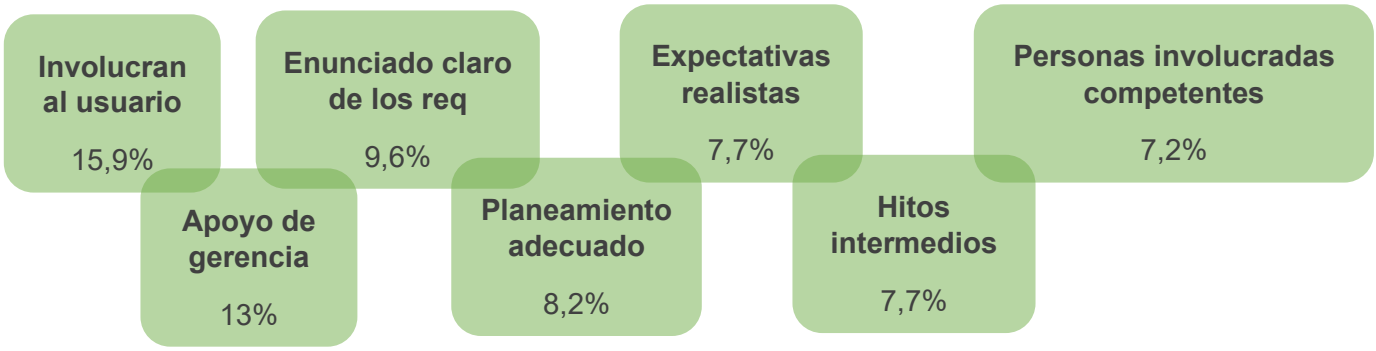
Claramente hay una relación entre estas características las cuales son que se trata la intangibilidad del software.

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.

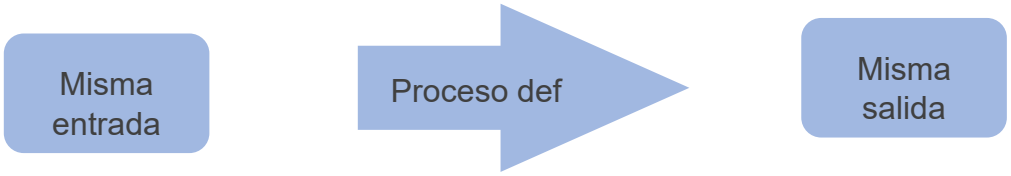
Empirismo y Definidos

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.

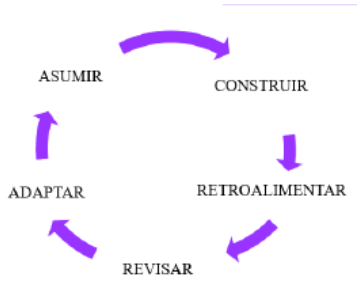
La idea es capitalizar la experiencia y así lograr la mayor calidad.

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

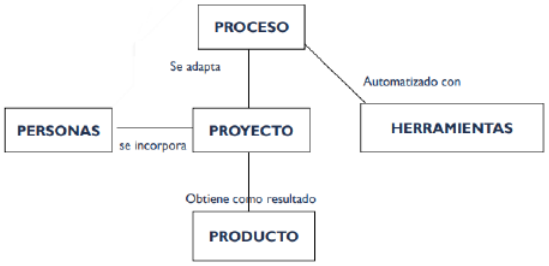
Utilizan para casos particulares, como proyecto de alto riesgo. Basado en la investigación de riesgos y ciclo de vida en B que permite hacer una vuelta en particular de cada etapa haciendo una funcionalidad e ir avanzando en los incrementos.

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.

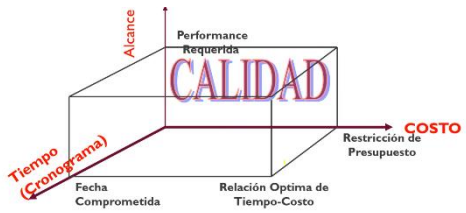
Se basa en ejecutar las actividades definidas para lograr el objetivo.

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.

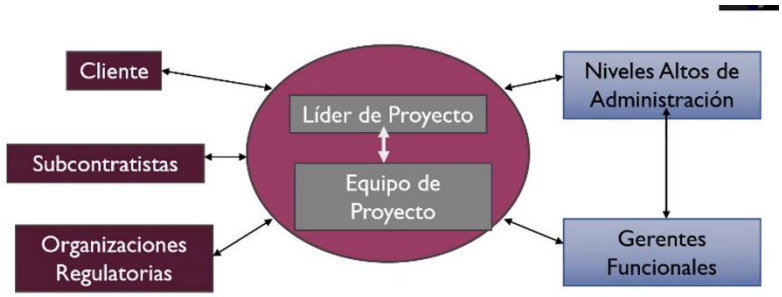
El balanceo de las mismas afecta directamente a la calidad del proyecto. Porque:

”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 • Misión/objetivo claro • Comunicación: En todos sus aspectos con el líder de proyecto y todos los involucrados	• Falta de comunicación • No existe monitoreo • Falta de planificación o pocos datos • No hay detalles en el plan • Mala planif de recursos • Estimaciones sin sustentos en datos históricos • Nadie a cargo • 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”

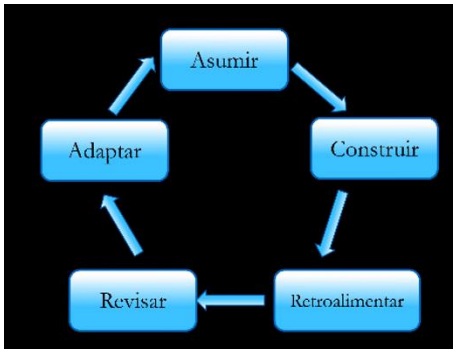
Lo podemos entender como un compromiso útil entre nada de proceso y mucho proceso.

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.

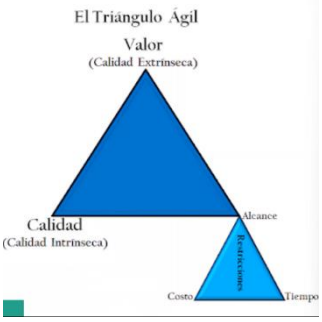
Triángulo Ágil vs Tradicional

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.

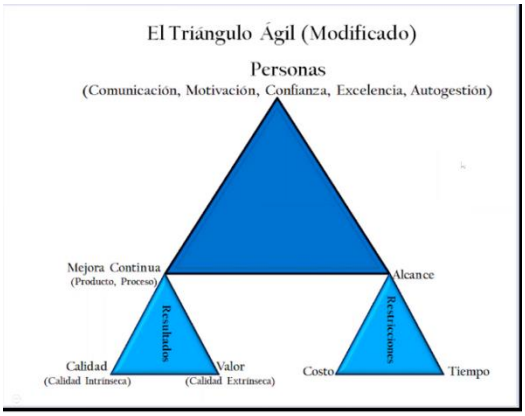


Llegando a nuestro proximo triangulo donde nos centramos en las personas porque son estás las más importante de nuestro proyecto. Según el empirismo son ellas las que van a capitalizar la experiencia que tienen y la del equipo para lograr una mayor calidad en el producto de software.

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:

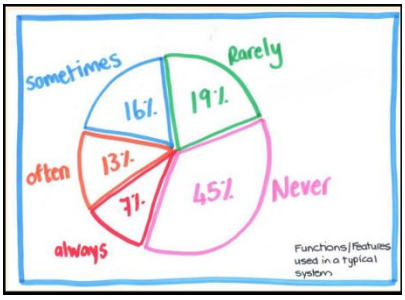
Valor: Usaremos valor para construir el producto correcto. Es decir, nos preguntaremos a quien queremos o vamos a ayudar con la creación de nuestro producto. La idea central de la construcción de software está centrada en el hecho de construir valor de negocio.

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)

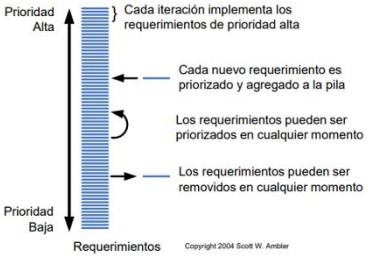
La necesidad de cambiar el enfoque a solamente lo suficiente surge del problema que solamente una pequeña porción 7% de las funcionalidades que construimos se utilizan siempre y una gran porción del 45% no se utilizan nunca lo que genera un desperdicio.



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. Matriz de Rastreabilidad para los requerimientos.	Historias de Usuario. Mapeo de historias (Story Mapping)
Productos	Definidos en alcance	Identificados progresivamente
Procesos	Estables, adversos al cambio.	Incertidumbre, abierto al cambio

Tipos de requerimientos

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 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:

Epicas: 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 epica. 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.

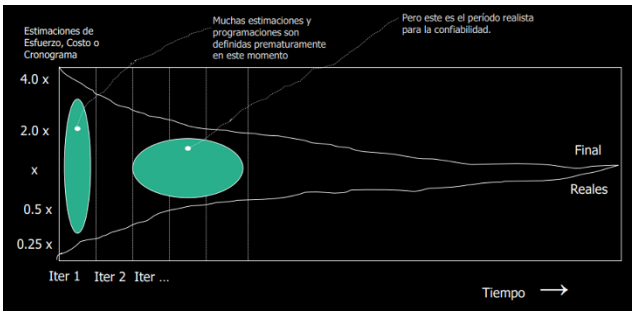
No son compromisos.

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.

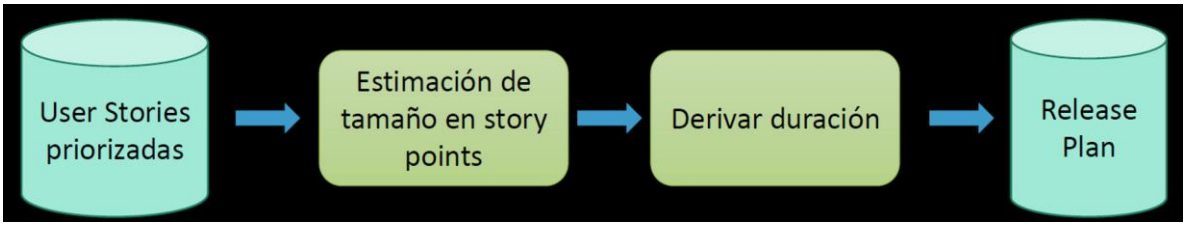
Para poder hablar de relatividad en los story points definiremos una US que será canónica. Es nuestro punto de comparación para el resto de las US. No necesariamente tiene que ser la más pequeña.

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.

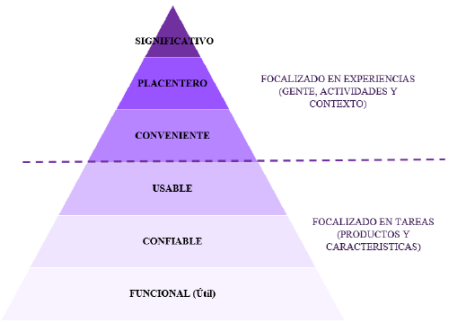
Gestión de productos

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

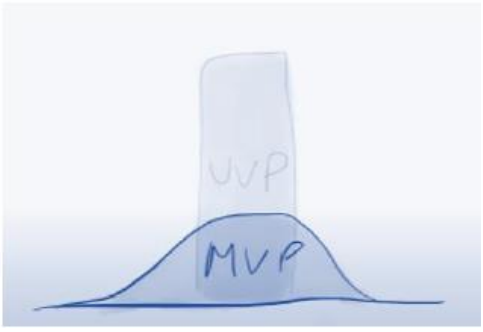
Vamos a iniciar un circuito de aprendizaje, poniendo a prueba la hipótesis y adaptándonos.

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.

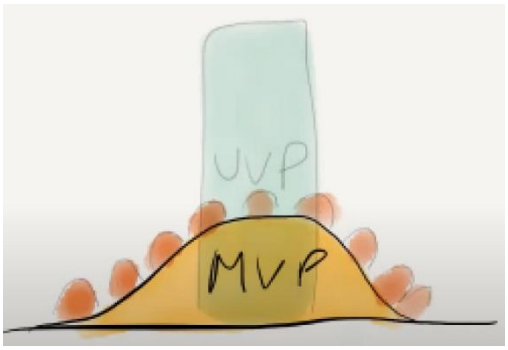
Queremos validarla para que nosotros no corramos el riesgo de armar un producto que no tiene un mercado. De esta manera buscamos minimizar el esfuerzo y el desperdicio. Construyendo un producto que el usuario quiere y demanda.

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.



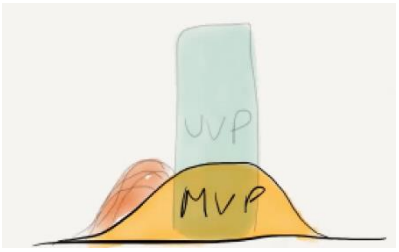
Objetivo es comprobar que la hipótesis del producto a construir funciona, o si necesitamos cambios.



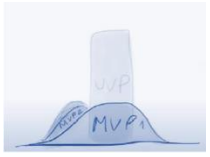
Nos puede suceder que la hipotesis como está definida sirva o que cada cliente potencial pida ciertas funcionalidades diferentes.

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.

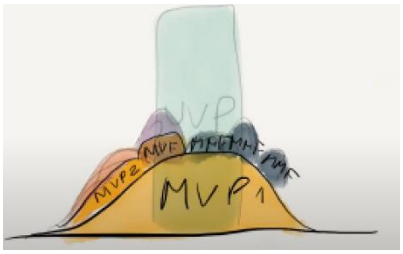
Dejamos de lado la hipótesis, ya fue validada, nos centraremos en definir elementos que deben estar presentes en un producto para que sea comercializable. validación del producto en términos de comercialización.

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

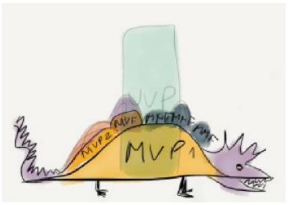
Se trata de una caracteristica mínima que se puede construir e implementar rápidamente. Quiero saber cual es la caracteristica mínima que hace la diferencia en mi producto.



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

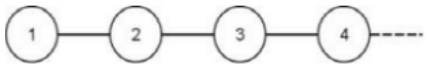
Dos tipos centralizados y descentralizados.

Versión

Es una instancia de un ítem que difiere del resto

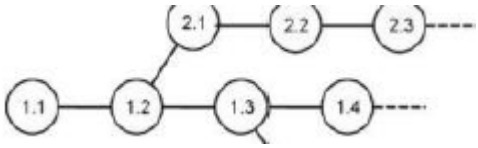
Es un estado del IC en un momento determinado. Se define desde el contexto de la evolución como la forma particular de un IC en momento determinado.

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 validación 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. Que el producto haga lo que tenga que hacer. Realiza una verificación.

Validación	Verificación
Se encarga de asegurar que IC resuelva el problema apropiado	Asegurar que un producto cumple con lo definido en la documentación de LB 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.