

INTRODUCCION A LA INGENIERIA EN SOFTWARE

Software: es un set de programas y la documentación que acompaña.

Existen 3 tipos básicos de software:

1. System Software
2. Utilitarios
3. Software de Aplicación

El Software y la Manufactura no se deben comparar, las razones son:

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

Problemas comunes con el desarrollo de Software

- ❖ La versión final del producto no satisface las necesidades del cliente.
- ❖ No es fácil extenderlo y/o adaptarlo.
Es decir, agregar más funcional en otra versión es casi imposible.
- ❖ Mala documentación.
- ❖ Mala calidad.
- ❖ Mas tiempos y costos que los presupuestados.

Aspectos para el Éxito y Fracaso en el desarrollo de Software

Éxito

- Involucramiento del Usuario (15,9%)
- Apoyo de la Gerencia (13%)
- Enunciado claro de los requerimientos (9,6%)
- Planeamiento adecuado (8,2%)
- Expectativas realistas (7,7%)
- Hitos intermedios (7,7%)

Fracaso:

- 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%)

➔ Saber programar nunca es suficiente para llevar a cabo el desarrollo de un producto de software.

Ingeniería de Software: “multi person construction of multi version software”.



- **Cuerpo de Conocimiento de la Ingeniería de Software**
- **Versión 3.0 del 2014 de la IEEE**
- **Está conformado por 15 áreas de conocimiento**

| |
|--|
| Software Requirements |
| Software Design |
| Software Construction |
| Software Testing |
| Software Maintenance |
| Software Configuration Management |
| Software Engineering Management |
| Software Engineering Process |
| Software Engineering Models and Methods |
| Software Quality |
| Software Engineering Professional Practice |
| Software Engineering Economics |
| Computing Foundations |
| Mathematical Foundations |
| Engineering Foundations |

La Ingeniería en Software está compuesta por distintas disciplinas:

- **Disciplinas Técnicas:**

Requerimientos

Análisis y Diseño

Prueba

Despliegue

- **Disciplinas de Gestión**

Planificación de Proyecto

Monitoreo y Control de Proyectos

- **Disciplinas de Soporte**

Gestión de Configuración de Software

Aseguramiento de Calidad

Métricas

El Proceso de Software

Es un conjunto estructurado de actividades para desarrollar un sistema de software.

Estas actividades varían dependiendo de la organización y el tipo de sistema que debe desarrollarse.

→ Debe ser explícitamente modelado si va a ser administrado.



Proceso: una secuencia de pasos ejecutados para un propósito dado.

Proceso de Software: un conjunto de actividades, métodos, prácticas y transformaciones que la gente usa para desarrollar o mantener software y sus productos asociados.

→ Dentro de un proceso están involucrados:

Procedimientos y Métodos.

Herramientas y Equipos.

Personas con habilidades, entrenamiento y motivación.

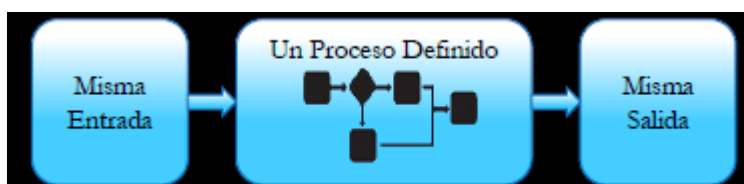
→ Tipos:

Definidos

Son aquellos que están inspirados en líneas de producción.

Asume que podemos repetir el mismo proceso una y otra vez, indefinidamente y obtener los mismos resultados.

La administración y control provienen de la predictibilidad del proceso definido.

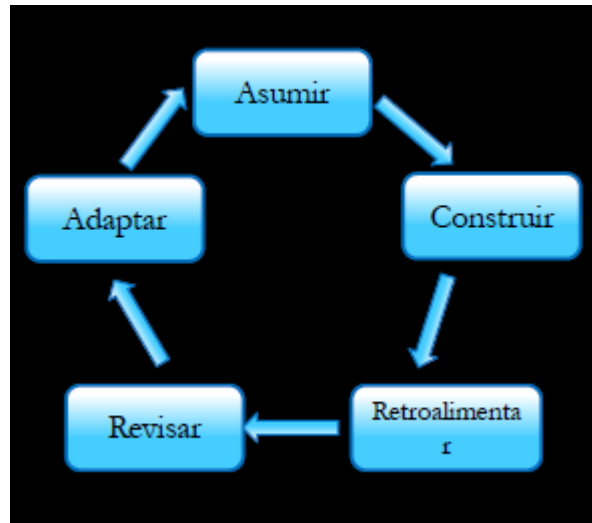


Empíricos

Asume procesos complicados con variables cambiantes. Es decir, cuando se repite el proceso se pueden llegar a obtener resultados diferentes.

La administración y control es a través de inspecciones frecuentes y adaptaciones.

Son procesos que trabajan bien con procesos creativos y complejos.

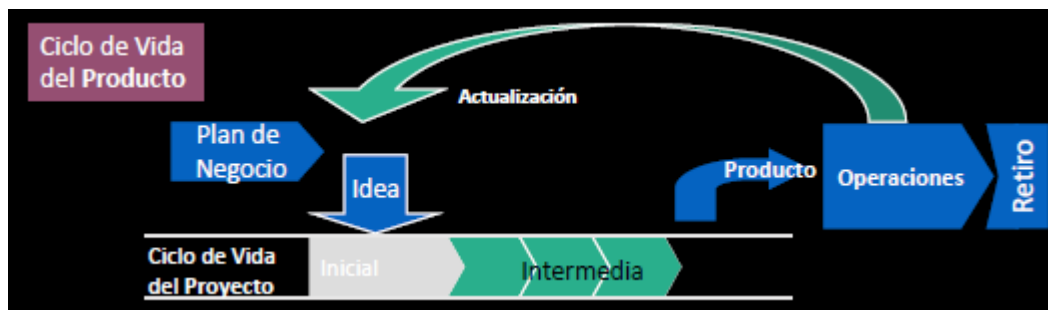


Ciclos de Vida

Serie de pasos a través de los cuales el producto o proyecto progresa.

- Los productos tienen su ciclo de vida.
- Los proyectos tienen su ciclo de vida.

Relación Ciclo de Vida Proyecto y Ciclo de Vida Producto



➔ Ciclo de Vida de Proyectos de Software:

Es una representación de un proceso, grafica una descripción del proceso desde una perspectiva particular.

Los modelos especifican:

- Las fases del proceso (requerimientos, especificación, diseño).
- El orden en el cual se llevan a cabo.

Clasificación (Tipos Básicos):

1. Secuencial
2. Iterativo / Incremental
3. Recursivo

COMPONENTES DE PROYECTO DE DESARROLLO DE SOFTWARE

Procesos Definidos: son procesos que están inspirados en líneas de producción, se definen secuencias de actividades en donde se asume que a determinada entrada se ejecutan determinada manera definida la salida va a tener los mismos atributos de calidad.

Tiene que ver con el orden de la ejecución de las actividades.

Ejemplo: PUD

Procesos Empíricos: tienen su desarrollo en aprender de la experiencia y es particular de cada equipo de trabajo.

Se trabaja en la definición de un proceso, de manera diferente.

→ La base es aprender / retroalimentarse de la experiencia.



Todo proyecto se crea a base de un proyecto que contiene herramientas que utilizan las personas obteniendo como resultado un producto.

Proceso: es una definición abstracta, existe y se materializa en función de los proyectos determinados.

El resultado de un proyecto es un producto de software.

Proyecto: empieza y termina.

No tiene una sobre vida, no acompaña a la vida del producto.

Producto: sobrevive a la vida del proyecto.

Sobre el mismo producto de software se pueden hacer varios proyectos

Personas: son claves, no hacemos nada sin ellas.

Herramientas: se utilizan para la automatización del proyecto.

Es inconcebible trabajar sin las herramientas que automaticen el proceso.

¿Qué es un Proyecto? (gestión tradicional de procesos definidos)

Independientemente si son de desarrollo de software o no, cualquier proyecto tiene ciertas características que lo definen como tal.

1. Están orientados a objetivos, se debe poder definir hacia dónde va y poder medir cuando se alcance el mismo para decir que se terminó el proyecto.

Deben estar descriptos de tal manera para distinguir si se finaliza el proyecto o no. (darnos cuenta si realmente alcanzamos el mismo).

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

2. Duración Limitada, se define el objetivo esperando en algún momento del tiempo alcanzarlos. Por eso tienen un inicio y un fin.

Una línea de producción No es un proyecto.

Debe tener el objetivo bien determinado.

Debe terminar en algún momento finito cuando se cumple con el objetivo.

Los proyectos son únicos y diferentes porque el resultado que vamos a obtener son distintos.

Y siempre va a haber algo que los diferencie. Hasta el momento en el que el proyecto está situado produce que tenga características diferentes, por más que el objetivo supuestamente sea el mismo.

- Los proyectos son temporarios, cuando se alcanza el o los objetivos, el proyecto se termina.

3. Tienen tareas que se relacionan entre sí, tienen precedencia y una depende de la otra.

A las mismas se le designan esfuerzos y recursos, de esta manera la planificación de la gestión del proyecto no es sencillo sino algo complejo.

Estas tareas tienen una complejidad inherente ().

- Complejidad sistémica de los problemas.

4. Los proyectos son únicos, por más similares que puedan ser tienen características que los diferencian, en todo sentido.

Si bien algunos tienen características similares, por más que respondan al mismo proceso definido (se obtienen el mismo producto)

¿Qué es la Administración de Proyectos? (tradicionales)

Es la aplicación de conocimientos, habilidades, herramientas y técnicas a las actividades del proyecto para satisfacer los requerimientos del mismo.

En términos de un proceso definido y proyectos tradicionales, incluyen necesariamente ciertos aspectos:

1. Poder definir el alcance, definir los requerimientos.

No es lo mismo el alcance del proyecto que el alcance del producto.

Con esto estamos hablando del alcance del producto,, los requerimientos que vamos a adquirir son relacionados y definidos en el producto

2. Establecer objetivos claros y alcanzables: es del proyecto.

El alcance:

- Proyecto: se mide en términos de lo que hay que realizar en función de lo que queremos obtener. (taras a realizar)
Trabajo necesario para hacer que se cumplan los objetivos del proyecto.

Lo relacionado a los objetivos del proyecto quedan ahí, no se los usa más.

- Producto: se mide en términos de requerimientos

Los requerimientos del producto, la vida de los mismos van a sobrevivir con el producto.

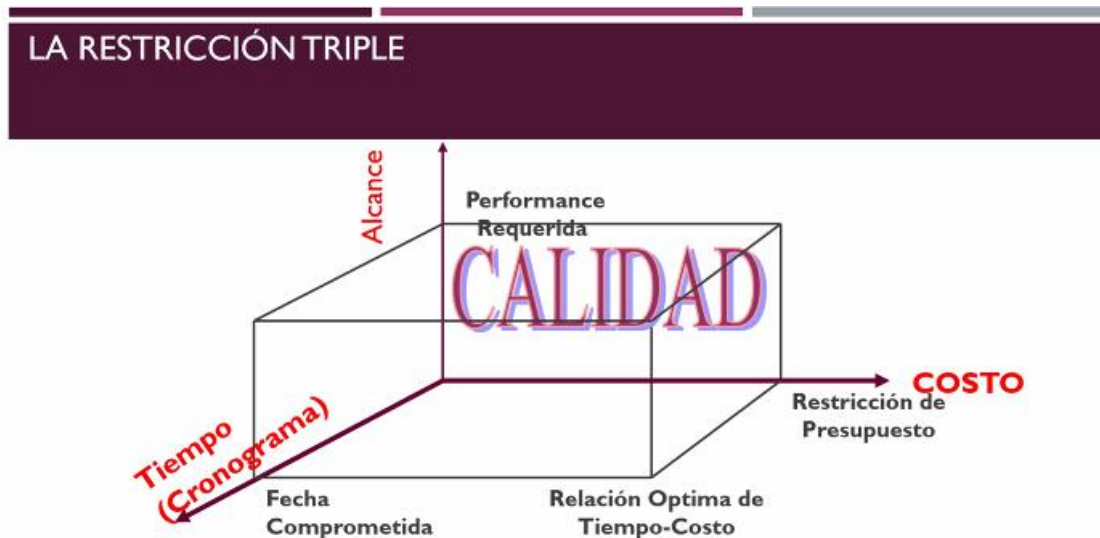
3. Adaptar especificaciones, planes y enfoque a los diferentes intereses de los involucrados.

- ➔ Aplicar conocimientos, habilidades, personas, técnicas y herramientas para poder realizar las actividades del proyecto para satisfacer los requerimientos de proyecto.

Triple Restricción:

Nos dice que en un proyecto vamos a tener una restricción de tiempo (cronograma con una determinada fecha comprometida), por otro lado un costo (determinado presupuesto para alcanzar los objetivos), Alcance (relacionado con la performance requerida del producto que se debe lograr en el tiempo determinado y con la calidad que va a tener / alcance del producto que se va a construir).

Las tres variables: alcance tiempo y presupuesto están relacionadas entre sí, de manera que si una varía, debe variar todas o alguna de las otras. (necesariamente las demás se ven afectadas)



Está relacionada con la habilidad que debe desarrollar el líder del proyecto, para dentro de la triple restricción lograr el objetivo que se espera del proyecto con la conformidad del cliente.

La habilidad del líder del proyecto es hablar con el cliente, para determinar cuál es la restricción que más pesa. La más importante para el cliente que es la que menos se debe mover.

- Objetivos de proyecto: ¿Qué está el proyecto tratando de alcanzar?
- Tiempo: ¿Cuánto tiempo debería llevar completarlo?
- Costos: ¿Cuánto debería costar?

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

“Proyectos de alta calidad entregan el producto requerido, el servicio o resultado, satisfaciendo los objetivos en el tiempo estipulado y con el presupuesto planificado”

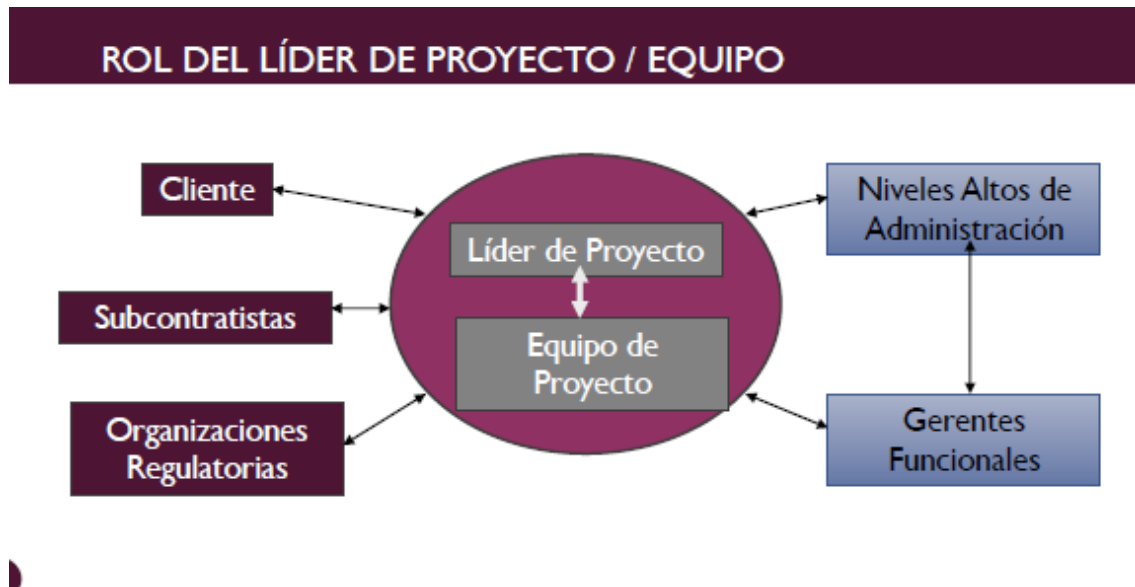
- Es responsabilidad del Líder de Proyecto balancear estos tres objetivos (que a menudo compiten entre ellos)

El Desarrollo de Software

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

Hay un primer ciclo de desarrollo donde se obtiene el producto de software y a lo largo de la vida del mismo podemos ir obteniendo nuevas versiones que son incrementos de la versión original / primera del producto.

Roles



- **LIDER DE PROYECTO:** es quien se ocupa de la gestión del proyecto, asignar tareas, gestionar al equipo, realiza la planificación, gestiona la triple restricción y mantener relaciones con clientes organizaciones externas, usuarios.

Pero no necesariamente hace todas, son su responsabilidad. Puede tener personas del equipo para resolver cada una de estas cuestiones, por lo que consulta con algunas personas del equipo, solo tiene bajo su cargo las actividades.

Independientemente de que el equipo de trabajo, (**completar con clase**)

- **EQUIPO:** grupo de personas comprometidas para alcanzar el o los objetivos del proyecto que se deben alcanzar.

Una de las características fundamentales es la posibilidad de trabajar en equipo, se buscan normalmente que sean grupos pequeños por cuestiones de comunicación.

Si es un grupo muy grande se dividen en más pequeños para que fluya más la comunicación.

Características:

- Diversos conocimientos y habilidades.
- Posibilidad de trabajar juntos efectivamente, desarrollando sinergia.
- Usualmente es un grupo pequeño.
- Tienen sentido de responsabilidad como una unidad.

¿Qué es el Plan del Proyecto?

Es una hoja de ruta, sobre la cual se va trabajando o teniendo definido como avanzar, que hacer y cómo hacer si nos desviamos (como nos recuperamos de una desviación, como solucionamos eso, como corregimos las dificultades que se nos presentan para llegar al objetivo del proyecto)

Se va definiendo a la largo del tiempo, a medida que logramos tener más definiciones.

Contiene: las actividades, fechas (cuando vamos a hacer las actividades / si se pueden superponer o no), como, quien lo hace.

Documenta:

- ¿Qué es lo que se hace?
- ¿Cuándo se hace?
- ¿Cómo se hace?
- ¿Quién lo hace?

¿Qué implica la planificación de Proyectos de Software?

El plan de proyecto implica:

- Definir el alcance del proyecto (más allá del producto que se construye)
- Qué proceso es el que nos va a guiar y que ciclo de vida vamos a utilizar.

(Proceso: definimos el conjunto de pasos a seguir

Ciclo de Vida: nos dice como ejecutamos esos paso / la instanciación concreta de los pasos. Ejemplo: cascada / iterativo e incremental, etc.)

- Estimaciones
- Gestionar Riesgos (cosas que pueden pasar que condiciones e impacten)
- Asignación de Recursos
- Programación de Proyectos
- Definición de Controles
- Definición de Métricas

Definición del Alcance

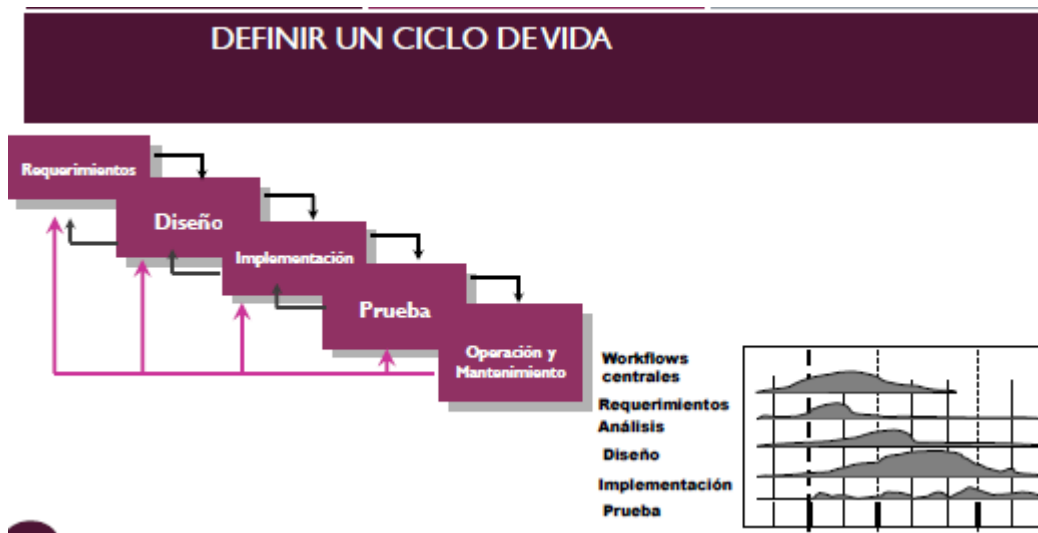
Alcance del Producto: son todas las características que pueden incluirse en un producto o servicio

Alcance del Proyecto: es todo el trabajo y solo el trabajo que debe hacerse para entregar el producto o servicio con todas las características y funciones especificadas.

¿Cómo se mide?

Alcance de Producto: se mide contra la especificación del requerimientos

Alcance de Proyecto: se mide contra el plan del proyecto.



Estimaciones de Software

La estimación no tiene que ver con la definición ni planificación final del proyecto o del cronograma.

Sirve de "input" para no comprometerse de más con algunas cuestiones que no se van a poder realizar.

Se estima:

- 1) **Tamaño:** lo podemos medir con casos de uso, definir el alcance de lo que vamos a construir.
- 2) **Esfuerzo:** cuantas horas hombre lineal se necesitan para construir lo que se definió en el tamaño. Medida en cantidad de horas de esfuerzo. Es necesario conocer con cuantos recursos y de que tipo contamos.
- 3) **Calendario:** se distribuyen las horas lineales en tareas a realizar, algunas veces se extienden o achican.
- 4) **Costo:** el costo está totalmente vinculado a la mano de obra, pero también depende de variables como el costo o uso de herramientas y sus disponibilidades.
- 5) **Recursos Críticos:** son aquellos momentos del tiempo y recursos que pueden generar algún inconveniente si no contamos con los mismos. Tienen que ver con lograr estimar cuales son los recursos que impactan directamente en el cronograma, solo los identificamos. Incluye recursos de todo tipo: humanos, herramientas.

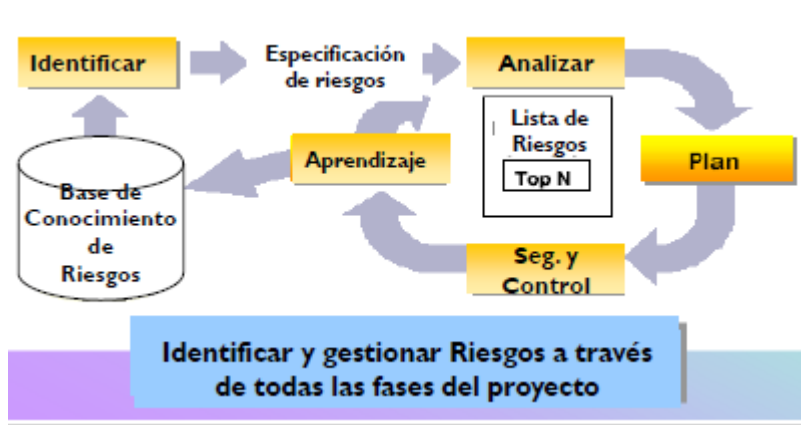
Esta es la manera en la que podemos ir definiendo la estimación. Se deben realizar en este orden.

Riesgos

Es un problema o un evento que puede llegar a suceder (si ya sucedió no es un riesgo) y que podría comprometer el éxito del proyecto.

Gestión de Riesgos:

Uno trata de identificar aquellos riesgos que aparecen en un top de lista y se gestionan solo esos.



Todos tienen asociada:

- 1) **Una probabilidad de ocurrencia**, que incluso puede variar a lo largo de la vida del proyecto.

Por ejemplo: cuando el cliente no tiene idea de que es lo que realmente quiere.

- 2) **Impacto:** si el riesgo ocurre cual es el daño que causa.

Se puede medir en riesgos cualitativos, cuantitativos o en términos de dinero (muchas veces no es fácil).

Entre estas dos variables se multiplican entre sí, y nos permite obtener la lista de riesgos.

Gestionar significa hacer algo para disminuir la probabilidad de ocurrencia o el impacto.

Ejemplos: rotación de personal, que cambie la versión del sistema operativo.

Métricas de Software

Se definen para que podamos a través de valor concretos saber cómo vamos con nuestro proyecto.

- ➔ Las métricas del Proyecto se consolidan para crear métricas de Proceso que sean publicas para toda la organización del software.

Lo definimos en términos de 3 dominios:

1. Métricas de Proceso:

Son las mismas métricas del proyecto, despersonalizadas, lo hacemos de una manera más estratégica con el foco de saber si hay algunas cuestiones específicas que hay que cambiar porque está mal definido el objetivo.

Nos podemos dar cuenta con esto por ejemplo que hay una mala planificación del proyecto.

2. Métricas de Proyecto:

Permiten ver cómo vamos con el trabajo realizado proyecto y cuánto falta para terminarlo.

Permiten tomar acciones correctivas.

Lo hace el líder del proyecto, tiene una mirada más táctica, solo se concentra en el proyecto y cómo hacer para terminar el mismo en tiempo y forma.

- ➔ Se consolidan para crear Métricas de Proceso que sean públicas para toda la organización del software.

3. Métricas de Producto:

Nos mide concretamente cuestiones relacionadas al producto de software que se está creando.

Generalmente tienen que ver con defectos. Miden características del producto final.

Métricas Básicas:

- Tamaño: producto
- Esfuerzo: proyecto
- Tiempo : proyecto
- Defecto: producto

Ejemplos:

Métricas Organizacionales son Métricas de Proyecto.

¿Cada cuánto se miden las Métricas?

Depende de las características del proyecto.

Ejemplo:

- Si el mismo dura 3 años, no se hace una medición diaria.
- Si el mismo dura 2 meses, las mediciones deben ser más frecuentes para poder tomar con tiempo acciones correctivas

Hay que tener en cuenta que hay métricas que no son necesarias medirlas porque no representan ningún tipo de información importante / ningún riesgo.

Es importante por lo tanto tener una lista de métricas posibles y de acuerdo a eso tener en cuenta cuales son las que vamos a usar, cada cuanto las vamos a tomar, como las vamos a tomar (tiempo y equipo implicado)

IMPORTANTE:

¿Qué hacemos con las Métricas?

Hacemos el monitoreo y control del proyecto, es decir, nos vamos dando cuenta si vamos a poder cumplir o no el objetivo del proyecto.

De no ser así, debemos tomar las medidas necesarias para encaminarnos y cumplir los objetivos.

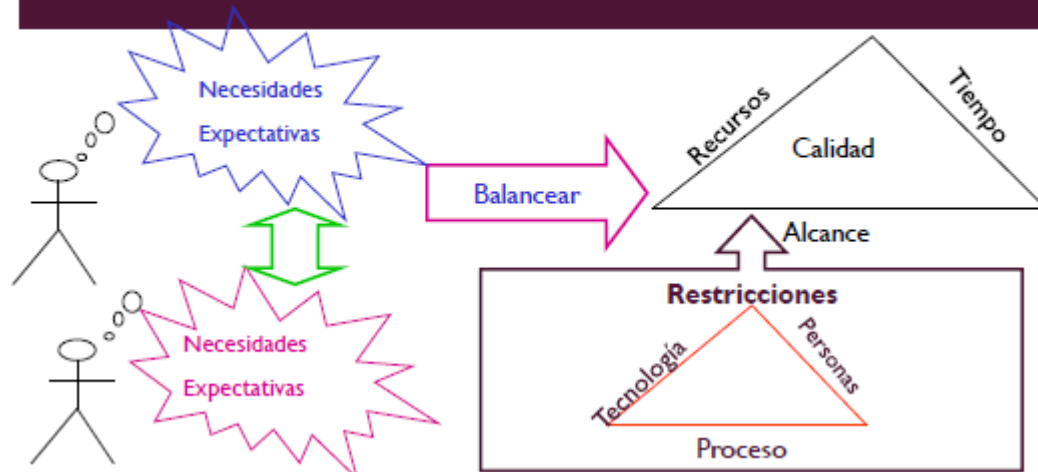
“De a un día por vez”

De esta manera podemos ir comparando lo “planificado” y lo “real”, puede que no estén alineados completamente pero depende de la observación.

No hay una perfecta igualdad entre lo planificado y lo real, lo ideal es que sean algo similares.

Es parte del trabajo del Líder del Proyecto y como herramienta se utilizan las Métricas.

ALGO MÁS SOBRE LAS MÉTRICAS...



Tres Factores para el Éxito de un Proyecto

1. Monitoreo y Feedback

2. Tener una misión / objetivo claro

Saber hacia dónde vamos.

Es clave que el equipo del proyecto este alineado con los objetivos que queremos lograr.

3. Comunicación

En todos sus aspectos: clientes, equipo, líder

Causas y Fracasos de Proyectos

- Fallas al definir el proyecto.
- Planificar con datos insuficientes.
- Las planificaciones la hizo el grupo de planificaciones.
- No hay seguimiento del plan.
- No hay plan de proyecto que se va retroalimentando en detalles.
- Planificación de recursos inadecuada.
- Las estimaciones se basaron en “supuestos” sin consultar datos históricos.
- Nadie estaba a cargo.

Nota: las métricas nunca deben estar definidas para evaluar el desempeño de las personas.

REQUERIMIENTOS AGILES

El Manifiesto Ágil presenta valores, que compara el costo vs el beneficio, se priorizan algunas cuestiones pero no implica que se deje de lado las demás.

El manifiesto tiene 4 valores y 12 principios.

Hay algunos principios que definen específicamente como vamos a trabajar con los requerimientos a lo largo de todo el ciclo de vida.

Se plantean 3 conceptos

- ➔ Algo que nos guía a la hora de definir los requerimientos tiene que ver con aquello que le da valor al producto que estamos construyendo. Esto da el hincapié / es lo más importante a la hora de definir los requerimientos.

(Más importante)

- ➔ La manera tradicional es encontrar historias, modelos, buscar maneras tradicionales.
- ➔ Avanzar con lo suficiente, es decir no trabajar de más pero si lo suficiente en el momento más adecuado y apropiado. No antes ni después.

Por qué en parte tiene que ver en base a los principios del manifiesto. Todos los productos de software aun los exitosos tienen un desperdicio que es significativo. Tienen un desperdicio en la construcción y definición de la funcionalidad

Hay muchas funcionalidades que se pueden usar muy rara vez o no se usan.

Las cuestiones que le dan valor al software se deben trabajar en el momento adecuado.

- Usar el “valor” para construir el producto correcto.
- Usar historias y modelos para mostrar que construir.
- Determinar que es “sólo lo suficiente”.

¿Qué es Ágil?

No es una metodología o proceso. Es una ideología con un conjunto definido de principios que guían el desarrollo del producto.

→ Es el balance entre ningún proceso y demasiado proceso.

La diferencia inmediata es la exigencia de una menor cantidad de documentación.

Importante:

- Los métodos ágiles son adaptables en lugar de predictivos.
- Los métodos ágiles son orientados a la gente en lugar de orientados al proceso.

Valores de los Equipos Ágiles:

- Planificación continua, multi nivel.
- Facultados, auto organizados, equipos competentes.
- Entregas frecuentes, iterativas y priorizadas.
- Prácticas de ingeniería disciplinadas.
- Integración continua.
- Testing Concurrente.

¿Cuándo es aplicable?

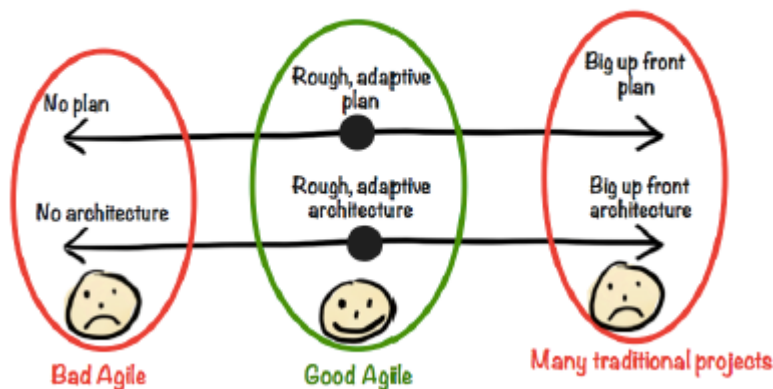
Cuando los problemas a ser resueltos caen dentro del espacio “Complex”, utilizar metodologías ágiles da mejores resultados.

El desarrollo de nuevos productos y Knowledge Work tienden a estar en el espacio “Complex”.

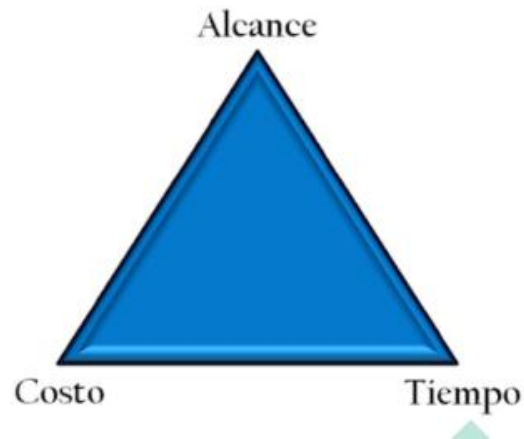
Investigación dentro de Anarchy.

→ Ser ágil no es ser indisciplinado.

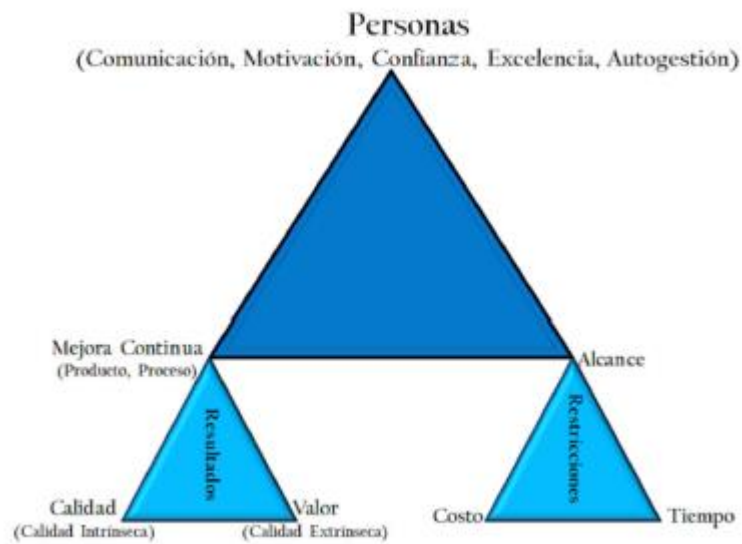
Don't go overboard with Agile!



El Triángulo de Hierro



El Triángulo Ágil (Modificado)



Gestión Ágil de Requerimientos de Software

1. Definir los requerimientos que tienen prioridad más alta.

Aquellos que den valor agregado al software de lo que quiere el cliente.

Implica que cuando hay que trabajar en la implementación de los mismos vamos a trabajar primero con los que están al inicio de la lista. Y luego vamos bajando hacia los que están al final de la misma.

Se pueden agregar al final nuevos requerimientos o se pueden sacar algunos que se consideren demasiado pocos importantes y no tengan valor.

(Esto está relacionado con la manera en la que nosotros modelamos los requerimientos, el cliente puede materializar el valor que le da a cada requerimiento para verificar la importancia.)

Van cambiando en el tiempo sin que nos afecte, hasta que tienen prioridad en el trabajo.

Just In Time: Analizo cuando lo necesito (en el momento en el que tiene la suficiente prioridad), detallo cuando necesito el requerimiento en el límite justo cuando lo necesito pero no demasiado tarde. Es parte de lo que se trabaja con la gestión de requerimiento.

Comunicación “cara a cara”: es la manera más eficiente de comunicación (mayor efectividad) a medida que utilizamos medios de comunicación más fría o distante se pierde la calidez del canal de comunicación.

Debemos priorizar este tipo de comunicación. Es un aspecto muy importante a la hora de definir los requerimientos.

Gestión Ágil de Requerimientos de Software

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





→ Valor

Es la obtención de beneficio tangible o intangible.

El valor lo asociamos a la utilidad, beneficio o satisfacción que se le ofrece a los usuarios finales por cada funcionalidad completa que se entrega.

Diferencias entre la forma de obtención de requerimientos:

| Tradicional vs Ágil | | |
|----------------------|---|--|
| | Tradicional | Ágil |
| Prioridad | Cumplir el plan. | Entregar Valor. |
| Enfoque | Ejecución ¿Cómo? | Estrategia (¿Porqué? ¿Para qué?). |
| Definición | Detallados y cerrados. Descubrimientos al inicio. | Esbozados y evolutivos – Descubrimiento progresivo. |
| Participación | Sponsor, 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 |

Se da de acuerdo a los requerimientos que tenemos y la manera de priorizar / trabajar sobre los mismos.

Diferencia más específica “Triángulo de Hierro”

Teniendo en cuenta la “Triple Restricción”

Tradicional

Si trabajamos en el contexto de un proyecto tradicional: nos ponemos de acuerdo en dejar fijo el alcance y en función del mismo definimos los recursos y luego el tiempo.

Está dirigido por el plan del proyecto, que debemos seguir a lo largo de todo el mismo hasta finalizarlo.

Están fijos los requisitos / requerimientos / alcance.

Ágil

En la gestión ágil las variables son las mismas, pero tener en cuenta que el valor es clave. A diferencia de la gestión tradicional se dejan fijos los recursos y el tiempo (período de tiempo pequeño), el alcance puede variar en función de estos.

Definimos un determinado tiempo en el que vamos a trabajar con una determinada (cantidad de recursos horas de trabajo, cantidad de personas) y en función de eso definimos que le entregamos al cliente.

Apunta al principio de las entregas tempranas y frecuentes (el cliente recibe de manera rápida software funcionando), en lugar de definir interacciones fijas y por alcance hacemos: cada 4 semanas por ejemplo entregamos un alcance en función de los recursos que tenemos. Ofrecemos un tiempo para que el cliente siempre sepa que le entregamos algo que le de valor.

- ➔ Como el lapso de tiempo es corto, las estimaciones de los alcances que vamos a entregar es más certera.
- ➔ El riesgo de construir un software que no cumpla con lo que el cliente quiere es mucho menor.
- ➔ Se pueden producir cambios en los demás requerimientos que todavía no se trabajaron. Se es más flexible al cambio.



Tipos de Requerimientos

Funcionales: proporcionan detalle de cómo debe comportarse un producto y especifican que se necesita para su desarrollo. Son las funcionalidades básicas de lo que debe hacer el sistema.

No Funcionales: también denominados de Calidad, detallan características que el producto debe poseer para mantener su efectividad y prever problemas o limitaciones del producto. Describen como el sistema llevará a cabo determinada funcionalidad.

Implementación: son puramente de implementación. se usan para detallar cambios en los procesos, roles en el equipo, migraciones de un sistema a otro, etc.

Negocio: definen los objetivos y problemas que la empresa quiere resolver con el producto. Son los que le dan valor al producto.

Están comúnmente basados en una necesidad real del usuario, sea esta conocida o no por él.

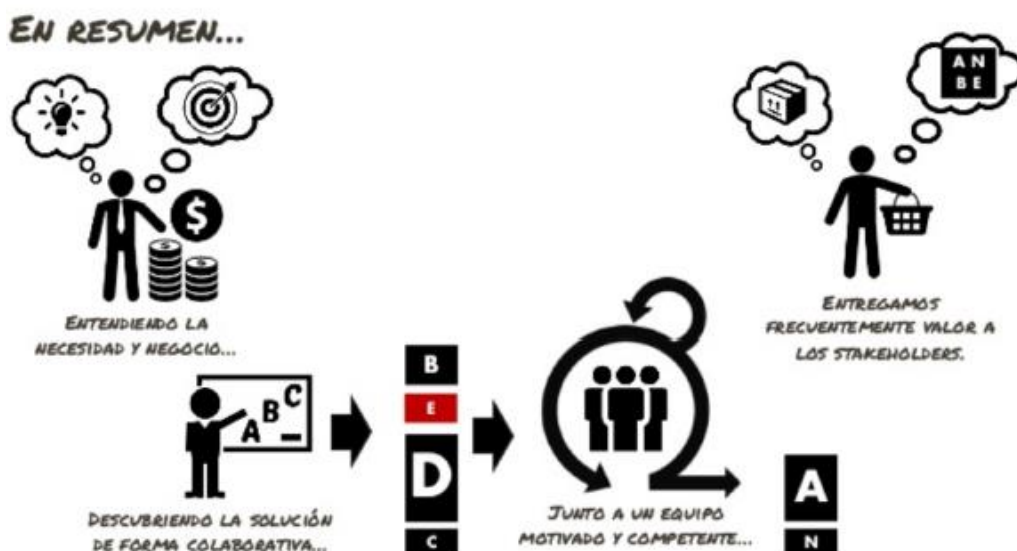
Usuario: describen las expectativas de los usuarios y como este interactuará con el producto de software que construyamos.

Los usuarios son los que reciben el valor de negocio que proporciona el producto.

Si no son similares a los Requerimientos del Negocio, el proyecto irá mal encaminado.

Resumen Requerimientos Ágiles:

1. Entender que le da valor al producto y como resolverlo: priorizando
2. Trabajar con un equipo motivado y competente.
3. Hacer entregas frecuentes del software funcionando.



Tips:

- Los cambios son la única constante

Buscamos gestionar requerimientos aceptando cambios, no podemos evitarlos.

Si el producto no platea cambios hay algún problema.

El cambio es parte de la vida de cualquier producto de software

- El valor es un aspecto sumamente importante.
- Stakeholders : no son todos los que están.
- Siempre se cumple que “el usuario dice lo que quiere cuando recibe lo que pidió”

Principios del Manifiesto Ágil que están relacionados con los Requerimientos Ágiles:



Son los que están más fuertemente relacionados.

En términos de la Metodología Tradicional

Sponsor: es normalmente el que pone la plata o en términos de una empresa tiene la suficiente fuerza en la organización para promover el proyecto. No conoce en detalle el valor del producto que se va a construir .

Cliente: es la persona que define los requerimientos que vamos a trabajar, no siempre es usuario pero sabe del valor que debe tener el producto.

Stakeholder: roles personas u organizaciones que son externas y partes del equipo o no. Los debemos tener en cuenta porque hay cuestiones del producto que van a estar relacionados con tareas o regulaciones de estos.

Ejemplo: como organizaciones ambientales, entres regulatorios.

USER STORIES

Es una técnica que utilizamos para gestionar los requerimientos ágiles.

La manera de utilizarla es como si contáramos una historia.

➔ No es tan importante lo que uno escribe, sino de lo que se habla de esa historia.

Una de las cuestiones por las cuales se comenten más errores a la hora de llevar a cabo un proyecto de software es la mala definición / establecer los requerimientos. El error se arrastra a lo largo de toda la vida del proyecto.

¿Qué es?

Es una manera de trabajar con requerimientos ágiles y relacionada a contar una historia.

¿Cómo?

Tiene 3 partes:

1. **Conversación**: es la más importante.
2. **Confirmación**
3. **Tarjeta**: se expresa / escribe la historia del usuario. Expresamos algo que nos indique de que se trata y el valor de negocio.

Forma de expresar las Historias de Usuario

➔ **Parte Frontal de la Tarjeta**:

Regla: Se identifica:

- **Como** <nombre del rol>

Representa quién está realizando la acción o quién recibe el valor de la actividad.

- **Yo puedo** <actividad que va a realizar el sistema>

Representa la acción que realizará el sistema.

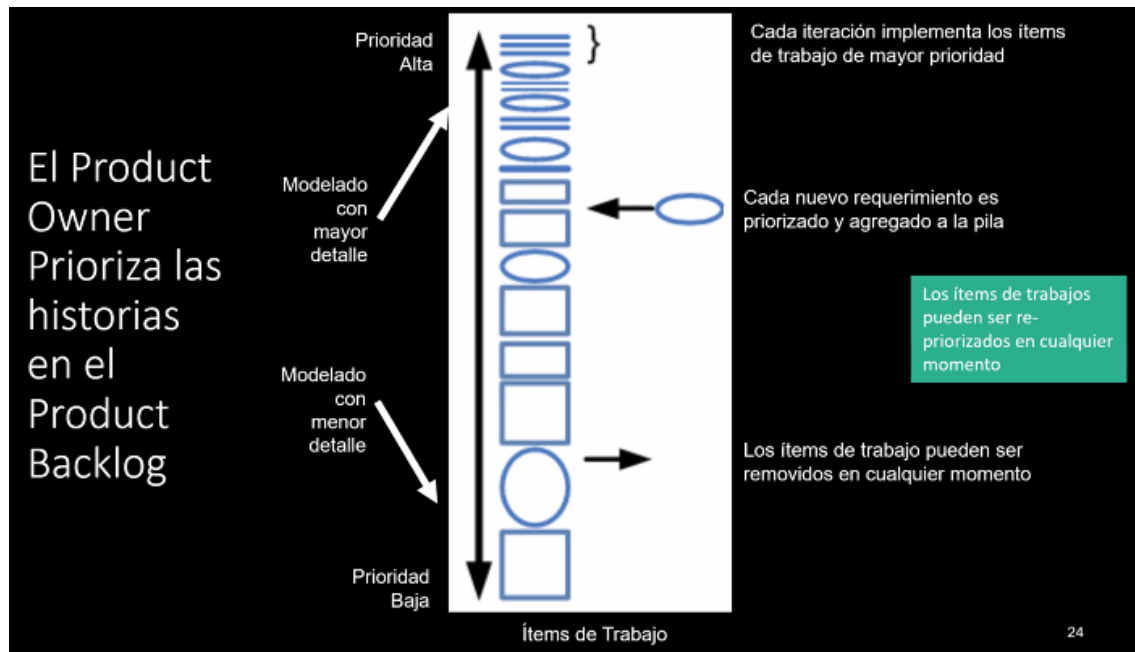
- **De forma tal que** <valor de negocio que recibo>

Comunica porque es necesaria la actividad.

➔ Siempre se expresan de la misma manera.

Las historias son: (multi usuarios)

- Una necesidad del Usuario.
- Una descripción del producto.
- Un ítem de planificación: priorizamos las historias / cada una es un ítem de planificación.
- Token para una conversación: hay que recordar de que hay que hablar.
- Mecanismo para definir una conversación: cuando eso tenga un suficiente valor de negocio lo hablamos.



Ejemplo: "USER STORY para un GPS"

Buscar Destino por Dirección

Como Conductor quiero buscar un destino a partir de una calle y altura para poder llegar al lugar deseado sin perderme.

Criterios de Aceptación:

- **La altura de la calle es un número.**
- **La búsqueda no puede demorar más de 30 segundos.**

- Lo que está en negrita representa: el rol, la actividad y el valor.
- Esto es la parte "frontal" de la tarjeta.

La Tarjeta nos da lugar a una conversación.

Criterios de Aceptación: cuestiones que definen límites / consideraciones, fundamentalmente para que el equipo pueda evaluar si lo que hacemos al implementar la USER STORY es lo que quiere el usuario.

Son cosas que efectivamente se ponen como límite / requerimientos mínimos para que la historia esté aprobada.

Pueden ser pocos o muchos, depende de la charla entre el equipo y el usuario.

Nos aseguramos de presentarle al usuario lo que realmente nos pidió.

➔ Pararnos siempre desde el lugar del usuario.

- Definen límites para una USER STORY
- No contienen detalles de implementación (son perspectivas del usuario) (son independientes de la implementación)
- Definen una intención de acuerdo al valor del negocio, no pensamos en la solución.
- Se escribe en alto nivel. (ayudan a los desarrolladores a saber cuándo parar de agregar funcionalidad en una US).
- Permite al equipo derivar las pruebas de usuario.
- Ayudan a que los Product Owner responsan lo que necesitan para que las US provean valor (requerimientos funcionales mínimos)
- Ayudan a que el equipo tenga una visión compartida de la US.
- Ayudan a los desarrolladores a saber cuando parar de agregar funcionalidad en una US.

➔ Consideraciones que define el usuario a la hora de hablarnos de como el espera, visualiza, imagina esa funcionalidad.

Los detalles que son resultados de la conversación del usuario y el equipo puede guardarse en la documentación interna o pruebas de aceptación automatizadas.

Este nivel de detalle queda fuera de la US.

➔ Parte de Atrás de la Tarjeta:

Confirmación

Pruebas de aceptación: pruebas que imaginamos que va a hacer el usuario cuando le entreguemos el software funcionando.

- Expresan detalles resultantes de la conversación.
- Complementan la US.

Proceso de dos pasos:

1. Identificarlas al dorso de la US.
2. Diseñar las pruebas completas.

Ejemplo:

Ejemplo: User Stories / Casos de Prueba

Como compañía quiero pagar por una búsqueda de puestos con una tarjeta de crédito, así resuelvo mi necesidad en forma más eficiente.

Criterio de Aceptación:

- Se acepta Visa, MasterCard y American Express
- En compras mayores de \$100 se piden el número del dorso de la tarjeta

Probar con Visa (pasa)

Probar con MasterCard (pasa)

Probar con American Express (pasa)

Probar con Dinner's Club (falla)

Probar con números de tarjeta buenos

Probar con números de tarjeta malos

Probar con números de tarjeta faltantes

Probar con tarjetas vencidas

Probar con montos menores de \$100

Probar con montos mayores de \$100

➔ No hay una cantidad determinada de US para hacer.

Directamente las armamos y priorizamos en una lista.

Conceptos Importantes

1. Definición de Listo

Está relacionado con que las US que están al principio de la lista tienen mayor nivel de detalle.

Cuando la US está en condiciones de ser implementada, definida de una manera que cumple con las condiciones para ser incluida en un release.

Es una definición que acuerda el equipo y hace una check list.

- Si no cumple con alguno de los requerimientos debemos verificarla.
- Si esta lista la podemos implementar.

¿Qué nos ayuda a definir cuando una US esta lista?

Hay un modelo INVEST MODEL para recordar cuales son las características que debe tener una US

1. **Independiente:** no depende de otra US.
2. **Negociable:** se negocia el “que”
3. **Valor**
4. **Estimable:** debemos poder definir cuanto esfuerzo nos va a llevar hacer esta US.
Sirve para poder determinar si la podemos hacer en un sprint o no.
5. **Pequeña:** si no puede ser implementada en un sprint hay que achicarla.
6. **Evaluable:** demostrar que fueron implementadas.

2. Definición de Hecho

Es propia del equipo y cumple con un check list y también está relacionada al requerimiento.

Es un acuerdo del equipo de cuando la US se encuentra lista para ser presentada al usuario.

Diferentes Niveles de Abstracción



Épicas: son US largas que no tienen demasiada prioridad. Es de granularidad menor.

Tema: es más amplia y nos interesa porque en algún momento lo vamos a tratar, solamente lo ponemos como un título (porque todavía no es momento de tratarla)

SPIKE

Es un tipo especial de US utilizada para quitar riesgo e incertidumbre de una US.

La falta de estimación de la US puede estar relacionada con desconocer la tecnología.

Para ello creamos esta US, lo que incluimos en ella es el resultado de la investigación de lo que no podíamos estimar.

Trabajamos en prototipo y a partir de esta información podemos estimar la US y luego poder implementarla.

- Se trabaja en un sprint anterior al que se va a introducir nuestra US.
- Uso (Funcionales):

Cuando hay cierta incertidumbre respecto de cómo el usuario interactuará con el sistema.

Son mejor evaluadas como prototipos para obtener retroalimentación de los usuarios involucrados.

- Técnicas:

Investigar enfoques técnicos en el dominio de la solución.

(evaluar performance potencial, decisión hacer o comprar, evaluar la implementación de ciertas tecnologías)

En cualquier situación en la que el equipo necesite una comprensión más fiable antes de comprometerse a una nueva funcionalidad en un tiempo fijo.

- No por cualquier cosa lo vamos a hacer, debe ser algo rápido. Se debe justificar su planteo, no todas las dudas son SPIKE.

Lineamientos

- Estimables: cuanto tiempo nos va a llegar o cuan grande va a ser y una manera de demostrar que se logró resolver.

La estimación siempre tiene incertidumbre, de acuerdo al nivel de incertidumbre las vamos a usar o no.

- Excepción, no regla.

Toda historia tiene incertidumbre y riesgos.

El objetivo del equipo es aprender a aceptar y resolver cierta incertidumbre en cada iteración.

Los SPIKES deben dejarse para incógnitas más críticas y grandes.

Utilizar SPIKES como última opción.


- Implementar la SPIKE en una iteración separada de las historias resultantes.

Salvo que el SPIKE sea pequeño, sencillo y sea probable encontrar una solución rápida, en cuyo caso, SPIKE e historia pueden incluirse en la misma iteración.


Pueden utilizarse para:

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


Algunas cosas para dejar en claro



DIFERIR EL ANÁLISIS DETALLADO TAN TARDE COMO SEA POSIBLE, LO QUE ES JUSTO ANTES DE QUE EL TRABAJO COMIENCE.



HASTA ENTONCES, SE CAPTURAN REQUERIMIENTOS EN LA FORMA DE "USER STORIES".



LAS USER STORIES NO SON REQUERIMIENTOS, NO NECESITAN SER DESCRIPCIONES EXHAUSTIVAS DE LA FUNCIONALIDAD DEL SISTEMA.

GESTION DE PRODUCTOS DE SOFTWARE

Cuando hablamos de productos de software debemos pensar a que apuntamos cuando construimos un producto.

- Diferenciar del proyecto con el producto.

¿Por qué creamos un producto?

Que surge de disparador a la hora de pensar crear un producto:

- Satisfacer a los clientes a los que podemos prestar servicio.
- Tener muchos usuarios logueado, el producto es muy popular.
- La obtención de dinero que ganamos.
- Realizar una visión que logre trascender y cambiar el mundo.

Primero reforzamos el punto de que el producto sobrevive a la vida del proyecto. Debe haber un ciclo de evolución donde le vamos agregando funcionalidades y cumpliendo los objetivos de crear el proyecto.

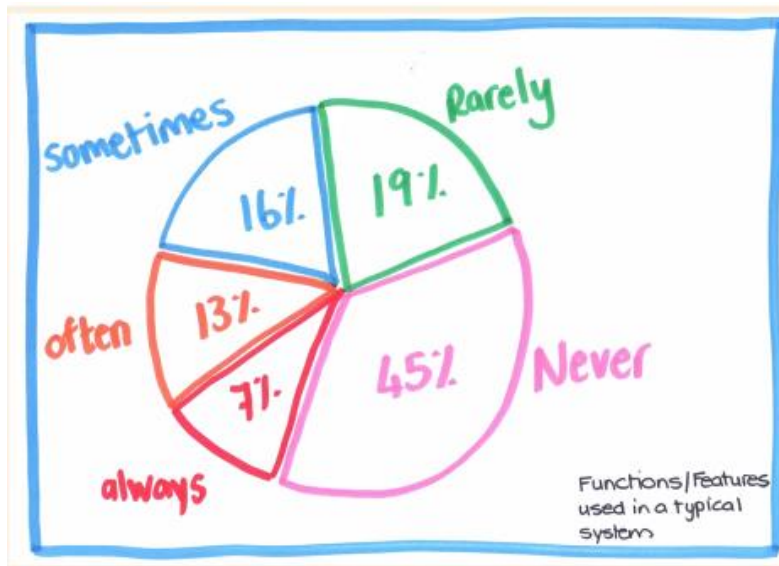
Tips para que las user stories sean útiles para el equipo

| | | | |
|---|--|---|---|
|  | Un paso a la vez (evitar la palabra "Y") |  | Usar palabras claras en los criterios de aceptación |
|  | No olvides la parte invisible: la conversación |  | Las user stories se escriben desde la perspectiva del usuario |
|  | | No forzar todo para escribirlo como user stories | |

Recordar:

Tenemos diferentes usos de diferentes funcionalidades de acuerdo al valor de negocio que se le entregue al cliente.

No debemos trabajar de más, solo lo que nos pide el cliente (trabajamos concretamente en lo que sabemos que el cliente espera y quiere que funcione)



¿Qué características realmente utilizamos de un producto de software?

3

➔ Nos enfoquemos en el 7% o 13% de funcionalidades, son las más esenciales.

Si nos paramos en analizar un producto de software que este en uso, podemos armar una Pirámide

Esto representa que cuando focalizamos la evolución de un producto de software en función de lo que quiere el cliente.

Primero: debemos satisfacer que nuestro producto cumpla con las funcionalidades esenciales. Que nuestro producto sea **Útil**.

Si el producto no cumple con las expectativas que quiere el cliente, no estamos haciendo bien las cosas.

Segundo: no solamente el producto debe brindar las funcionalidades que se esperan, sino brindar la sensación de seguridad al usarlo. Quedarnos tranquilos de que realmente funciona bien. El producto debe ser **Confiable**.

Dependiendo del producto la confiabilidad tendrá mayor o menor grado.

Tercero: quienes vayan a utilizar el producto deben sentirse cómodos, utilizarlo con facilidad. El producto debe ser **Usable**.

➔ Por lo general un producto de software de calidad básico cumple con estas 3 condiciones.

➔ Las otras condiciones son aquellas que cuestan más que se cubran y muchas veces el desarrollo o evolución de productos de software no llegan a esos niveles.

Cuarto: **Conveniente**. Mas allá de ofrecer la funcionalidad que espera el cliente, vamos un paso más allá y justo en el momento indicado el producto de software ofrece lo que se estaba buscando.

Quinto: **Placentero**

Sexto: Significativo, tiene que ver con para que creamos productos de software: tener una visión trascendente que cambie / afecte de manera significativa y beneficiosa al mundo.

Cambiar la vida de las personas.

Ejemplo: WhatsApp, Mercado Libre (en el contexto de la pandemia, ayudo a muchos negocios a aumentar o mantener ventas y funcionando)

No muchos productos de software cumplen con los últimos niveles de la pirámide.



➔ De esta manera podemos comparar distintos productos de software que cumplen o no cumplen con toda la pirámide.

Creación de Productos

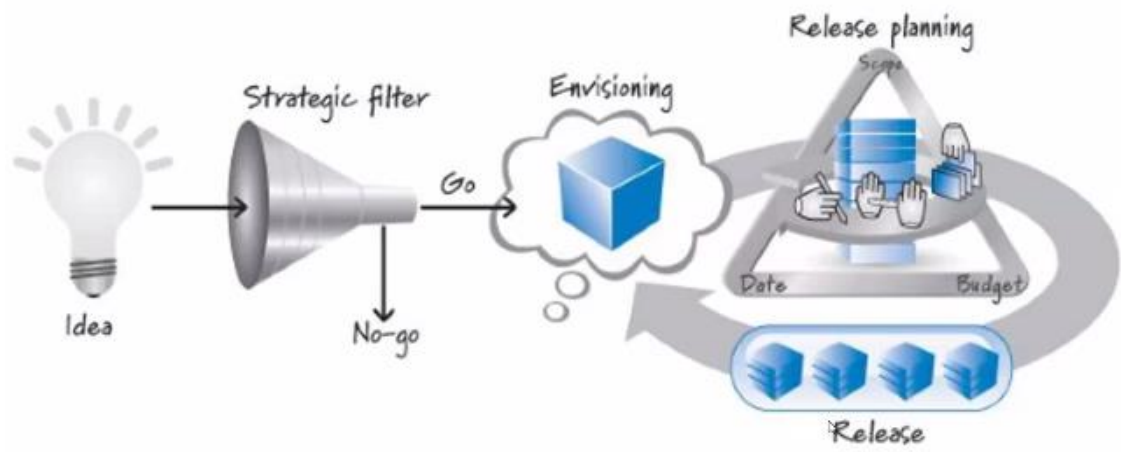
Se parte de una idea de una persona, necesitamos de un sponsor o alguien que nos respalde para que la misma se pueda materializar en un producto de software.

Empezamos con la visión del producto, viendo cómo hacer para que la idea materializada en algo más concreto se convierta en al menos un producto básico de software.

Priorizamos características, para decir cuál de ellas van en los distintos release del producto (retroalimentan la idea de funcionalidad del producto).

Dentro del triángulo de release planing se habla de la frontera de la triple restricción, como hacemos para administrarlo / gestionarlo / dividirlo de manera estratégica.

- ➔ Tiene que ver con poder trazar en términos genéricos como se imagina el producto de software a través del tiempo de acuerdo a las funcionalidades que le vamos agregando.

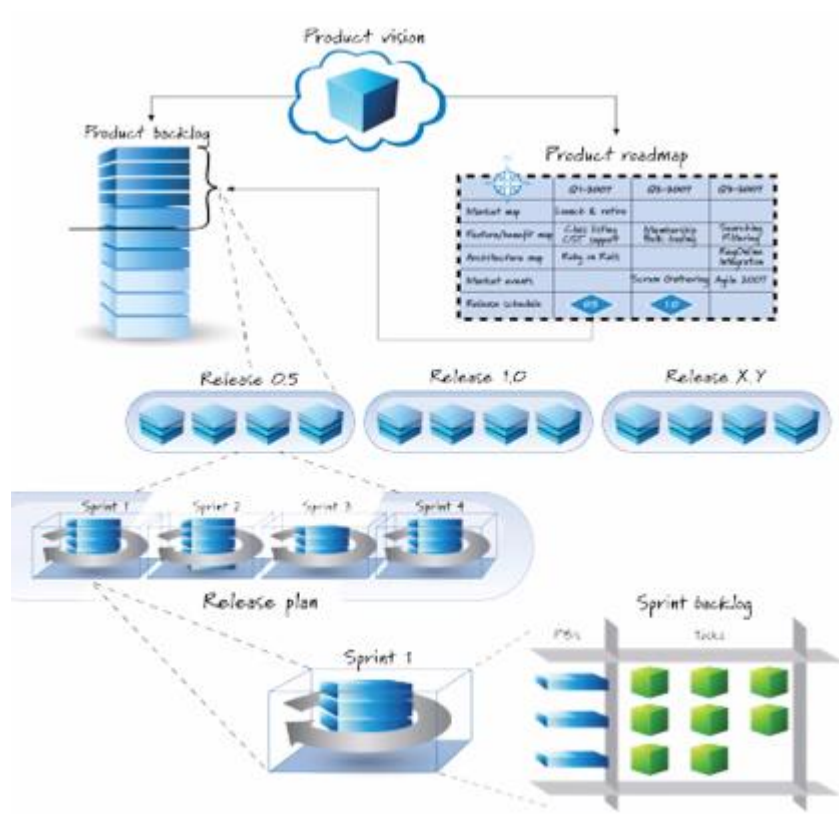


- USER STORIES
- Características.

Lo más importante para el negocio es lo que está arriba. Con el resto de las características no están del todo detalladas (tienen poco peso y en el momento no nos dan valor al negocio), trabajamos de acuerdo a lo que necesitamos en el momento.

Evitamos correr el riesgo de trabajar en aspectos que luego pueden cambiar.

Por esta razón listamos todas las características, detallando lo más prioritario.



Determinar de acuerdo a las características del inicio de la pila, cuáles van a estar en cada release.

- Cumplir con el Modelo INVEST

Dentro de un sprint, o iteración, donde un release tiene una x cantidad de sprint y en cada uno vamos a definir concretamente cuales son las tareas que se planifican y ejecutan para finalmente alimentar el plan de release.

Así vamos a tener las distintas etapas del producto, utilizando los distintos planes de release, nos permite saber en qué instantes de tiempo vamos a contar con que funcionalidades.

- ➔ No vamos a empezar con el producto de software hasta no saber si el mismo va a responder a las funcionalidades planteadas / si cumple con el objetivo que planteamos al inicio.

Si se cumple con el ¿Por qué lo creo al producto?

Valor vs Desperdicio

¿Cuáles de nuestros esfuerzos crean valor y cuáles son desperdicio?

El punto en este momento es: yo quiero crear un producto de software para un objetivo, debo elegir por donde empiezo pensando en que tengo que validar para que mi producto realmente se va a materializar de la manera en la que yo quiero que sea así.

- Lean Thinking define la creación de valor como proveer beneficios a los clientes, cualquier cosa es desperdicio.
- La productividad de un Startup no puede medirse en términos de cuánto se construye cada día, por el contrario, se debe medir en términos de averiguar la cosa correcta a construir cada día.

A partir de esto surge el concepto de:

MVP (Producto Mínimo Viable)

La idea es construir el MVP de tal manera que en lugar de vender a alguien una idea en términos conceptuales se le muestre el producto funcionando con las funcionalidades básicas.

A partir de ello obtener una retroalimentación de si el producto funciona y si es conveniente. Organizar a partir de allí como ir realizando las demás funcionalidades hasta terminar de concretarlo.

No necesariamente es un producto para entregar y comercializar, sino que sirve para validar hipótesis en lugar de que mis usuarios y potenciales clientes lo puedan visualizar mediante su uso.

- Es un concepto de Lean Startup que enfatiza el impacto del aprendizaje en el desarrollo de nuevos productos.
- Premisa Clave detrás del MVP, es que se produce un producto real que puede ofrecerse a los clientes y observar su comportamiento real con el producto o servicio.
- Eric Ries:

“versión de un nuevo producto que permite a un equipo recopilar la cantidad máxima de aprendizaje validado sobre clientes con el menor esfuerzo”.

Este aprendizaje validado vienen en forma de si sus clientes realmente comprarán el producto.

- Ver lo que la gente hace con respecto a un producto es mucho más confiable que preguntarle a la gente que harán.

Método Lean Start – Up

Es la Base del MVP.

Se creó para reducir el tiempo y costos a la hora de crear empresas a través de Hipótesis – Implementación.

Permite ofrecer a los clientes productos que el crea que es real y a partir de esto nos dé retroalimentación para corregir la idea que tenemos pensada.

A veces podemos desarrollar característica del producto o hacer que el cliente piense que lo que le mostramos es real y así sabemos y determinamos que es lo que quiere.

Características Claves:

1. Tener las características suficientes para que a las personas les interese, estén dispuestas a usarlo o comprarlo así como esta (inicialmente).
2. Que los usuarios quieran seguirlo usando en el futuro.

Demuestra suficiente beneficio futuro para retener a los primeros usuarios.

3. Retroalimentación, para cambiar un poco la visión que tenemos de nuestro producto (cambiar la idea del negocio)

Proporciona un ciclo de vida de retroalimentación para guiar el desarrollo futuro.

Errores Comunes:

1. Confundir el objetivo: confundir el comercializar con la validación de hipótesis (el objetivo no es comercializarlo, siempre tiene que ver con el aprendizaje)
2. Que no ofrezca el valor que el cliente está esperando (lo que no nos permite hacer la retroalimentación)

Plasmar una idea en términos concretos, experimentar la hipótesis a partir del uso de los clientes, tener métricas y tomar datos de lo que paso con el uso, aprender y cambiar la idea.

3. Enfatizar la parte mínima con exclusión de parte viable.

El producto entregado no es de calidad suficiente para proporcionar una evaluación precisa de si los clientes utilizarán el producto.

4. Entregar lo que consideran un MVP y luego no hacer más cambios a ese producto, independientemente de los comentarios que se reciban al respecto.

➔ Un MVP varia en complejidad desde pruebas de humo, extremadamente simples a prototipos.

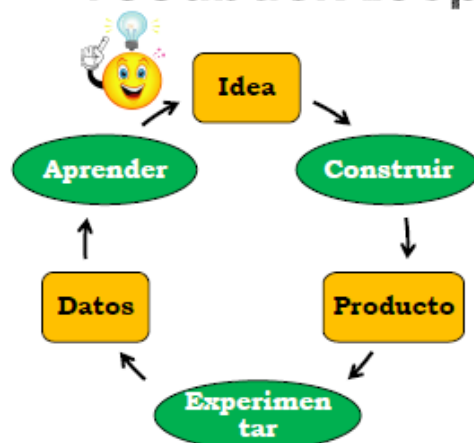
Éxito – Hacia la creación de valor

➔ El éxito no es entregar un producto, el éxito se trata de entregar un producto (o característica de un producto) que el cliente usaría.

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

The Build Experiment Learn feedback loop: permite descubrir las necesidades del cliente y alinearlas metodológicamente.

Build-Experiment-Learn Feedback Loop



Fase de Construir MPV

- Ingresar lo más rápido con un Producto Mínimo Viable.
- Un MVP varia en complejidad desde pruebas de humo extremadamente simples hasta prototipos tempranos.



Ejemplo: Dropbox

Para poder desarrollar este producto para que cualquier persona desde sus distintos dispositivos pueda acceder a datos.

1. Validar si alguno de los futuros usuarios les interesaba usarlo.

Se hizo un video donde narraba de manera coloquial a los primeros usuarios y mostrando (materializado en términos concretos) dando algunos elementos para que a los usuarios les quedara la idea.

Plantearlo de tal manera de poder retroalimentarse de usuarios para saber si valía la pena o no llevarlo a cabo.

Ni siquiera se construyeron las características básicas.

Lo primordial para construir un MVP es:

(completar)...

- Tener en cuenta cual es la hipótesis que queremos corroborar.
- Plantearlo desde el lugar donde no tengo nada.
- Lograr que los usuarios tengan ciertos saltos de fe sobre lo que les estamos prometiendo.
- Se debe materializar, mostrarla de alguna forma para que las personas (usuarios/clientes/beneficiarios) puedan darnos una retroalimentación de lo que opinan y les parece el producto.

Ejemplo Facebook:

Se destaca el que se enfocaron en no solo captar los nuevos usuarios sino que los mismos la sigan usando a la aplicación.

Dilema: La Audacia de Cero

A menudo es más fácil recaudar dinero cuando tiene:

- Cero ingresos
- Cero clientes
- Tracción cero.

Que cuando tenemos una pequeña cantidad de cada uno.

➔ Cero invita a la imaginación, pero los números pequeños hacen preguntas sobre si los números grandes alguna vez se materializaran.

Este fenómeno crea un incentivo brutal.

Aplazar el lanzamiento de cualquier version del producto hasta que se esté seguro del éxito.

Si se pospone experimentar con MVP, surgirán algunos problemas:

- La cantidad de trabajo desperdiciado puede aumentar.
- Se perderán los comentarios esenciales.
- El riesgo de que su startup construya algo que nadie quiere puede aumentar.

Compensaciones:

- ¿Preferible atraer capital de riesgo y potencialmente derrocharlo?
- ¿O preferiría atraer capital de riesgo y utilizarlo sabiamente?

Usar MVP para experimentar (inicialmente en silencio) con los primeros usuarios en el mercado.

Verificar su concepto probando TODOS sus elementos, comenzando por los más riesgosos.

Supuestos “Saltos de Fe”

Son los elementos más riesgosos del plan / concepto de una startup (es decir, las partes de las que todo depende).

En casos de problemas, la mayoría de las personas no conocen una determinada solución (o incluso un problema), pero una vez experimentada la solución no pueden imaginar cómo vivirían sin ella.

- **Hipótesis del Valor:**

Prueba si el producto realmente está entregando valor a los clientes después de que comienzan a usarlo.

Una métrica de prueba: Tasa de Retención.

- **Hipótesis de Crecimiento:**

Prueba como nuevos clientes descubrirán el producto.

Una métrica de prueba: Tasa de Referencia / Net Promoter Score (NPS)

Preparar un MVP



Roadmap: todas las características que tengo explicitadas, cuales voy a desarrollar primero, cuales después y en qué momento aproximadamente las voy a tener.

La idea no es cumplir lo específicamente pero si definirlo en términos realistas. No como una presión, sino que se pueda especificar en términos concretos.

Pre ver el MVP: encontrar un sponsor.

Testear las Suposiciones: tener un grupo de personas que retroalimenten.

Funcionalidades Principales: son las que nosotros queremos validar en términos del producto que queremos crear, nos permite ver si la idea funciona o no. Las mismas pueden tener que verse modificada.

➔ La idea es que las personas compren / utilicen el producto y sigan sosteniendo su uso a través del tiempo.

SOFTWARE CONFIGURARION MANAGEMENT (SCM)

Gestión de Configuración de Software

¿Qué es?

Es una disciplina protectora, es decir, que va ocurriendo transversalmente a lo largo de todo el proyecto. Pero más allá inclusive porque trasciende el mismo para darle soporte al producto en todo su ciclo de vida.

→ Recordar: La ingeniería de software está alimentada por disciplinas de distintos tipos: técnicas, de gestión, de soporte.

Desde el momento 1 en el que tratamos de concebir al producto empezamos a generar producto de trabajo o artefactos, componentes que en la gestión de configuración tiene un nombre específico: **ITEM DE CONFIGURACION**

Ítem de Configuración

Es cada cosa que yo quiero controlar / gestionar.

Desde el primer ítem de configuración que imaginamos/pensamos, plasmamos en algo que se puede meter dentro de la computadora (sin importar el formato que tenga el mismo: foto, Word, código, prototipo, archivos que se pueden poner en la computadora).

Por esto, debemos comprender porque debemos tener una visión de la palabra SOFTWARE más abarcativa / más grande que lo que es solo código.

→ Software: es cualquier producto de trabajo que sale de cualquier actividad dentro del ciclo de vida del proyecto y después del ciclo de vida del producto.

Esos ítems de configuración que en general se gestan en el contexto de un proyecto van conformando el producto.

→ La disciplina de Gestión de Configuración es la responsable (su propósito es) de mantener la integridad del producto.

Integridad del Producto

¿Qué tiene que tener un producto para tener Integridad?

- Satisfacer las necesidades del usuario.
- Ser fácil y rastreable durante su ciclo de vida.
- Satisfacer los criterios de la performance.
- Cumplir con la expectativa de costos.

En todo nuestro trabajo transversal (porque empieza desde el momento cero y sigue mientras el producto existe) hay actividad de gestión de configuración.

Responsabilidades

Todos tenemos responsabilidad por el producto de software que estamos construyendo entre todos.

Cada vez que alguien genera/crea un ítem de configuración es responsable por respetar los lineamientos que el equipo definió, respecto a cómo mantener la integridad de producto. (todos somos responsables)

- Hay algunos roles específicos como el Gestor de Configuración que tienen tareas adicionales. Normalmente mantener la herramienta, marcar líneas base y organizar los procesos de cambios.

No implica un rol full time, sino que tiene momentos de su trabajo dedicados a esa tarea y el resto del tiempo se dedica a su función principal.

Configuration Manager (Gestor de Configuración): es la cabeza principal de la configuración de gestión del proyecto de software, se encarga principalmente de coordinar todo lo que se va a hacer al respecto.

Los demás integrantes de equipo participan constantemente en el desarrollo del software por lo que tienen que seguir los pasos del patrón de proyecto.

Por ejemplo:

- ➔ En caso de cambios... Es responsabilidad de un equipo de delegados de cada área que debe participar para en el caso de que haya un cambio estar todos al tanto y todos evalúen como afecta a su área.

Soporte de Herramientas

Como es algo que se integra, hay un soporte importante de herramientas que nos ayuda a automatizar las actividades que se puedan, por lo que uno tiene mucho más que ver por una cuestión de cultura que con otra cosa.

- Es acostumbrarse a trabajar teniendo en cuenta los lineamientos que se habían definido en el equipo para controlar y mantener la integridad del producto.

Problema

El problema grave que tiene el Software es que cambia y que además es muy fácil modificar / cambiar un software.

- Ejemplo:

Es muy fácil, sin querer, seleccionar un conjunto de carpetas en un repositorio y borrarlas. Lo que genera seis meses de pérdida de trabajo de un equipo y fue en un “clic”.

Es una de las características que complica el software, la maleabilidad de destruirlo y la facilidad también de cambiar cosas que hagan que: el producto deje de funcionar sin mucho esfuerzo.

→ La **Gestión de Configuración** (esta disciplina) tiene como propósito que tratar que el producto este integro.

Es decir, tratar de que si vamos a hacer un cambio, tengamos control de que es lo que cambio en un determinado momento.

Recordar: siempre asociado al concepto de Software va el concepto de Versión. Cada ítem de configuración tiene que tener su versión.

Para eso la disciplina intenta tener una certeza de cómo es la conformación del producto en un momento determinado.

Disciplinas de Soporte del Software (Disciplinas Protectoras del SW)

La idea de la materia es dar visión de lo que es “hacer software” desde otra perspectiva que no es solo la técnica (no solo programar o diseñar). Sino con otras perspectivas de disciplinas que nos ayudan a que el producto sea el producto integro y de calidad que se espera que sea.

El Marco de Trabajo para que otras disciplinas como el “Aseguramiento de Calidad de Software” puedan existir, es la “Administración de Configuración de Software”.

No se puede asegurar la calidad de un producto que no se sabe dónde está, se desconoce la última versión, no se conocen los componentes que forman parte de un determinado release.

- Esto es lo que debemos garantizar con la disciplina de Administración de Configuración de Software.

Gráficamente lo podemos ver como un “contenedor”, que es la disciplina que ayuda a que el resto de las disciplinas puedan funcionar y hacer su trabajo.

¿Por qué deberíamos Gestionar la Configuración?

Su propósito es establecer y mantener la integridad de los productos de software a lo largo de su ciclo de vida.

Involucra para la Configuración:

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

Recordar: el ciclo de vida del producto es más grande que el ciclo de vida del proyecto.

Desde que se concibe el producto, todos los ítems de configuración que se crean y evolucionan, hasta que ese producto se saca de línea, se deja de usar, se tira a la basura.

Conceptos Clave para la Gestión de Configuración de Software

Ítem de Configuración

Son todos y cada uno de los artefactos que forman parte del producto o del proyecto, que pueden sufrir cambios o necesitan ser compartidos entre los miembros del equipo y sobre los cuales necesitamos conocer su estado y evolución.

- ➔ Debe ser un elemento que se pueda meter dentro de un File System / un repositorio de un disco de una computadora. Puede tener cualquier formato.

Ejemplos: un requerimiento: en un archivo de Word, en una herramienta que gestione solicitudes de requerimientos, una Base de Datos (un archivo en Excel), una USER STORIE en foto.

Documentos de diseño, código Fuente, Código Ejecutable, etc.

Repositorio

Es el contenedor de ítems de configuración (es donde van a parar los ítems de configuración) que debe tener una estructura que permite definir que ítem de configuración vamos a colocar en cada lugar.

La estructura dentro del repositorio (conocida, publicada y acordada entre todos los que van a trabajar en un determinado proyecto) es necesaria para facilitar la búsqueda de los ítems de configuración.

Además, ayuda a la seguridad, controles de acceso, políticas de Back Up que se aplican a un repositorio. Sumado al concepto de compartir, evitando así que cada integrante del equipo tenga cosas diferentes en su máquina local (produciendo errores).

Lo ideal es tener un repositorio para un producto y todos los ítems de configuración que tiene el mismo y que nos interesa controlar, alojados en algún lugar de la estructura del repositorio, con un esquema de configuración de permisos tal que permita estar tranquilo de quien accede a realizar cada tarea.

Versión

Desde el punto de vista de la evolución es, la forma particular de un artefacto en un instante o contexto dado.

El Control de Versiones refiere a la evolución de un único ítem de configuración o de cada ítem por separado.

- La evolución puede representarse gráficamente en forma de grafo.

Línea Base

Es aquella que puede contener en un momento de tiempo un solo ítem de configuración (es raro pero se puede), o un conjunto de ítems estables (se los puede tomar como referencia, pasaron todos los niveles de revisiones y aprobaciones).

- Se utiliza para “marcar” las baseline. Se los marca para identificar que los mismos forman parte de la línea base, que también debe tener una identificación unívoca.

La Línea Base debe tener: nombre, versión e identificación única en un momento. Indicando claramente cuáles son los ítems de configuración con su estado que forma parte de ella.

Sirve para: tener un punto de referencia para los ítems de configuración de los que se determinó que están en un estado tal que se los puede tomar como “base” para avanzar en el desarrollo.

- No confundir con Versión de Producto.

Hay dos tipos de líneas base que se pueden identificar a lo largo de un proyecto.

- ➔ Las Líneas Base se pueden gestionar de la manera a la que cada equipo de desarrollo le resulte más conveniente.

Hay equipos que solo mantienen una línea base identificada al mismo tiempo (las otras son históricas), o hay equipos que gestionan teniendo líneas base por fases (requerimientos, diseño, código), manejándose así con varias líneas base activas.

Pueden ser:

- **Especificación (Requerimientos, Diseño)**

No tienen código, sino que tienen información de ingeniería del producto.

- **Operacionales**

Tienen código que ya es operativo, se marca para entrar a una serie de pruebas de aceptación, o entrar a una versión de producción.

Productos (que han pasado por un control de calidad definido previamente)

- ➔ Recordar: que podemos trabajar en un proyecto en un repositorio y si utilizamos ítems de configuración que no forman parte de ninguna línea base, hay libertad de acomodarlos, cambiarlos y modificarlos.

Mientras los ítems de configuración NO formen parte de una línea base, no necesitan ningún mecanismo formal, ni ninguna autorización para tocarlo o cambiarlo o modificarlo.

Es una de las cosas que caracterizan a los ítems de configuración que forman parte de una línea base, sobre los no se tiene esa libertad de acción.

Ramas (Branch)

Ramas de versionado en el contexto de la gestión de configuración.

- Existe una Rama Principal.
- Sirven para bifurcar el desarrollo
- Pueden tener razones de creación con semántica.
- Permiten la experimentación.
- Pueden ser descargadas o integradas.

Integración de Ramas

- La operación se llama MERGE.
- Lleva los cambios a la rama principal.
- Pueden surgir conflictos
- Todas las ramas deberían eventualmente integrarse a la principal o ser descartadas.

Proceso:

Se arma una bifurcación de la línea donde están los ítems de configuración (normalmente la línea base) debido a que no se recomienda trabajar ni accionar directamente sobre los ítems que están en la rama principal.

Cuando se termina de trabajar, se revisa y está estable, se integra sobre la rama principal modificando la situación de la misma.

En algunos casos queda descartada, no se la llega a integrar nunca porque no sirvió (por la razón que fuere). La rama se descarga sin llegar a integrarse en la rama principal.

Configuración del Software

Actividades Fundamentales de la Administración de Configuración de SW

son las cosas que contienen las secciones de un plan de gestión de configuración.

- La Gestión de Configuración debe ser planificada.

El conjunto de actividades que van a involucrar a una serie de personas que hay que coordinar en su trabajo, suele organizarse usando planes.

Son un recurso útil si se usan de manera adecuada, a servicio de uno y no a servicio de los planes.

1. Identificación de Ítems

Implica la definición de la estructura del repositorio, se identifican los ítems unívocamente y se asignan los ítems a un lugar específico dentro de la estructura del repositorio.

Puede ocurrir que hay ítems que dé ante mano se puede especificar su nombre y hay ítems (como los componentes de código o los casos de prueba, que dé ante mano al inicio del proyecto no sabemos cómo cada desarrollador los va a nombrar).

- ➔ Se definen reglas / esquemas de nombrado genéricos, los cuales siguen estándares de lineamientos que están vinculados al lenguaje de programación con el que se trabaja, tantos caracteres para el nombre + la extensión + la fecha, etc.

Esa regla de nombrado es la que todos después van a utilizar para definir cómo se los va a llamar a los ítems de configuración.

Normalmente el Gestor de Configuración arma la estructura del repositorio y es el responsable de ponerlo a disposición al equipo.

Luego los miembros del equipo que comienzan a generar ítems de configuración desde el momento cero que inicia el proyecto, consultando el plan de configuración para saber cómo deben llamar a los ítems y donde guardarlos.

Ejemplo: el plan de configuración puede ser un afiche pegado en la pared para que todos los miembros del equipo puedan con facilidad, ver el nombre y crear los ítems de configuración de manera adecuada.

No importa el formato que tenga, pero si debe cumplir el propósito de que todo el mundo sepa los acuerdos y los lineamientos para la identificación de los ítems y para el resto de las actividades que se desarrollarán.

Tipos de Ítems de Configuración

De acuerdo al ciclo de vida que tenga cada ítem, por lo general en un proyecto de desarrollo de software trabajamos con 3 tipos de ítems distintos.

➔ **Producto**

ERS, Arquitectura, Código, Manual de Usuario

➔ **Proyecto**

Plan de Proyecto, Cronograma

➔ **Iteración**

Plan de Iteración, Reporte de Defectos

Se hace esta distinción porque el ciclo de vida del producto trasciende el ciclo de vida del proyecto.

A lo largo de un ciclo de vida de un producto se empiezan y terminan mucho proyectos que van haciendo evolucionar los ítems de configuración del producto.

Por lo que cuando se identifican los ítems y se les asigna un lugar en el repositorio se debe tener en cuenta que tipo de ítem es.

- ➔ Se define la forma de identificación de los ítems y donde se los va a guardar teniendo en cuenta de que ciclo de vida forman parte.

2. Control de Cambios

Proceso de Control de Cambios

Está relacionada a mantener la integridad de las líneas base conforme se van realizando cambios.

- ➔ Línea Base es un ítem o conjunto de ítems de configuración con su estado, su versión que han sido revisados y se los considera estables y que para cambiarlos deben pasar por un proceso formal de control de cambios.

Por esta razón los ítems de configuración que no forman parte de una línea base, se los puede cambiar a voluntad y necesidad con total libertad.

El Proceso Formal de Cambios posee varias actividades, pero como primer instancia, el cambio que se solicita debe ser autorizado y verificar si lo que se cambio es lo que se había autorizado o no.

- Es la actividad que se encarga no solo de definir las líneas base, sino también de mantenerlas a lo largo del tiempo.

Tiene su origen en un **Requerimiento de Cambio** a uno o varios ítems de configuración que se encuentran en una Línea Base.

Es un procedimiento formal que involucra diferentes actores y una **Evaluación del Impacto del Cambio**. (determina si el comité de control de cambios va a autorizar un cambio o no)

Comité de Control de Cambios: son referentes del equipo que está trabajando en el desarrollo del producto.

Se definen a quienes les interesa enterarse del cambio que apareció y en función de eso se convoca al comité.

Hay un conjunto de roles que son “básicos” que tienen que integrar el comité: arquitecto, quien vela por los requerimientos funcionales, desarrolladores, líder de proyecto (vela por los aspectos de los recursos: tiempo, costos, personas). A partir de eso se pueden sumar más personas si es necesario.

- ➔ El propósito es que todos los que se tengan que enterar que se cambió una línea base se enteren.
- ➔ Se hace asociado a las líneas base, si no hay ninguna marcada todavía entonces no hay procesos formales para cambiarlas

3. Auditorías de Configuración de Software

Auditoría: Es una revisión independiente y objetiva sobre el producto.

Lo que se audita es una línea base, la línea base que se acordó que se va a auditar con quienes están en el proyecto.

En una Auditoría, el auditor debe ser alguien externo al equipo. Se lo convoca para hacer una auditoría y se le pide la misma sobre una determinada línea base del producto.

Se hacen dos auditorías:

- **1. Física** → verificaciones

Vela por la integridad del repositorio, es decir, que el repositorio este, que este alojado en el lugar donde se estableció que debe estar, que los ítems de configuración respeten el esquema de nombrado, que los ítems de configuración estén guardados donde se dijo.

- ➔ Ve la situación del repositorio, verificando que los ítems de configuración dentro son los que deben estar para poder satisfacer los requerimientos.

- **2. Funcional** → validaciones

Ve si el producto es el correcto, si en los ítems de configuración se responde / hacen lo que los requerimientos dicen que tienen que hacer.

- ➔ Una de las primeras cosas que se preguntan antes de iniciarla es si está disponible el reporte de la auditoria de configuración física.

Porque si esta no sale bien, la funcional directamente no se hace.

La Matriz de Rastreabilidad de Requerimientos es una herramienta muy útil para hacer auditorias. Debido a que las auditorias se hacen por muestreo, no se auditan todos los ítems que hay en un repositorio, se elige un requerimiento al azar y revisamos si fue desarrollado correctamente (auditoria física) y que haga lo que debe hacer (auditoria funcional).

- ➔ Las auditorias necesitan un plan, de lo contrario no hay un lugar donde este definida la estructura del repositorio, el esquema de nombrado de los ítems, la conformación de comité.

Como la auditoria es un proceso de control, si no se tiene un plan, no hay contra que controlar.

Y si no se tiene una línea base, no se tiene que controlar.

Una auditoria toma un plan y controla una línea base, y en función de ese proceso de control se obtiene un informe de auditoría que muestra todas las desviaciones que se detectan en el momento en el que la auditoria se realiza.

Luego el equipo realiza un plan de acción, debe informar cómo se piensa resolver las cosas que el auditor mostro como desviaciones.

4. Informes de Estado

El propósito de la actividad es generar reportes para dar visibilidad y poder tomar decisiones.

La información nos sirve para hacer visible la situación de la configuración (hablando de informes de estado)

- ➔ En general cualquier pedido de informe sirve para dar visibilidad, avisar, enterar a quienes necesitan, acerca de un estado de situación de una determinada cosa.

El reporte básico es el inventario, que lista todos los ítems de configuración que tiene el repositorio con su estado en un momento de tiempo, o listar cuantas solicitudes de cambio se atendieron en un determinado periodo de tiempo. O listar el contenido de una línea base / o listar cuantas líneas base hay para un determinado proyecto, saber quién autorizo un determinado cambio a una línea base y cuando se implementó.

Sobre la base del repositorio y usando la herramienta de gestión de configuración que se tiene, obtenemos reportes de manera automática.

El plan debe contener respuestas a las preguntas de cómo se va a hacer las cuatro actividades básicas de la Gestión de Configuración.

- Cuantas auditorias y de que tipo se van a hacer.
- Que informes de estado se van a sacar.
- Estructura del repositorio, etc.
- Como se conforma el comité de control de cambios.

Evolución de la Gestión de Configuración de Software

La Gestión de Configuración es una disciplina fundacional para mantener la integridad del producto y que ha permitido que los equipos evolucionen en su forma de construir software y traten de hacer un proceso que sea cada vez más productivo.

Y que para lograr esa productividad tratan de automatizar la mayor parte de los procesos que se pueda.

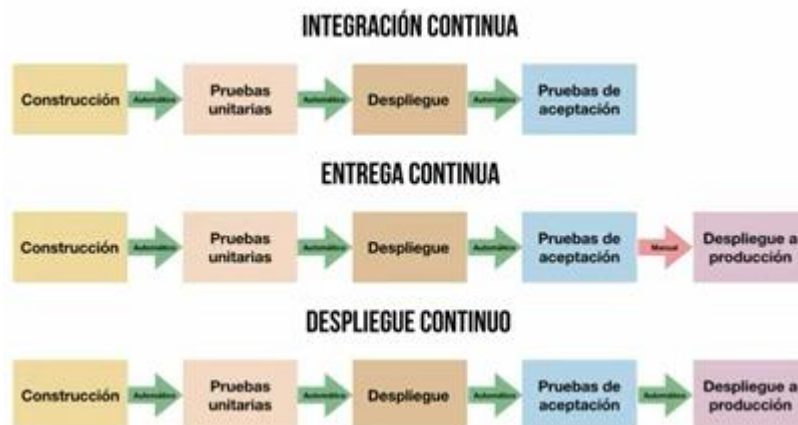
La automatización nos ayuda, no solo haciendo las cosas más rápidamente, si no evitándonos errores que a veces para nosotros son inevitables y que automatizándolos ese tipo de errores no se cometen.

Prácticas Continuas

Integración, Entrega y Despliegue

Estas disciplinas son la evolución de la Gestión de Configuración de Software.

Con evolución refiere a que si no hay la base de gestión de configuración puesta (como primera capa) ninguna de estas cosas son posibles. Es como llevar la gestión de configuración a un nivel de automatización que nos permite a nosotros ir implementando cada una de estas prácticas (que en términos generales se les llaman Prácticas Continuas).



Todas estas prácticas continuas llegaron con las corrientes ágiles, que apuntan a automatizar lo más que se pueda las actividades del proceso de desarrollo.

Todas las metodologías ágiles formalizan mucho y son muy rigurosos respecto a la calidad y forma en la que se va a construir el código.

Le dan una especial importancia al testing, con la idea de que esté lo más automatizado posible.

Integración Continua

Se empieza con cada desarrollador en su entorno trabaja, hace pruebas unitarias (en la medida lo más posible automatizadas) desarrollando con algo que se llama TDD (Desarrollo Conducido por Testing).

Cuando termina con el componente de código sabiendo que funciona, lo sube a un repositorio de integración.

Hay pruebas de integración que se van corriendo automáticamente, para tener una versión del producto que permanentemente funciona. Evitando que el día que se deba entregar el producto al cliente esto deje de funcionar o haya cualquier problema, garantiza que ande el producto.

Entrega Continua

Es el delivery, sumado la automatización de las pruebas de aceptación, permitiendo que entonces el producto esté listo para desplegarlo a producción.

Las pruebas de aceptación no deberían de hacerse en el mismo ambiente donde el software va a funcionar, sino en otro.

Las pruebas de aceptación ya automatizadas dejan una versión para que manualmente se pase a producción. Cuando este último paso también es automatizado tenemos el Despliegue Continuo.

Despliegue Continuo

Es la última actividad que se realiza, ponemos en ambiente de producción del usuario final del producto.

- ➔ La diferencia entre estas tres practicas es el nivel de automatización de las pruebas, hasta donde se llega con la transparencia de que las pruebas se hacen con productos de software (que las realizan) y que ni nos enteramos que se despliega una nueva versión de un producto en producción.

Gestión de Configuración de Software en Ambientes Ágiles

La disciplina de Gestión de Configuración de Software es aplicable en ambientes ágiles.

SCRUM no habla de gestión de configuración, pero va de la mano con metodologías ágiles debido a que busca responder a los cambios y mantener la integridad del producto.

El Manifiesto Ágil va aplicado hacia el proyecto (como trabajar en el contexto de desarrollo de un proyecto) y la Gestión de Configuración esta aplicada al producto (por lo que es más extenso que el manifiesto).

- ➔ La discusión que debe tener el equipo (debido a que es el equipo el que decide lo que quiere y no, cuanto hacer y cuanto no) es como se va a hacer la Gestión de Configuración (no se discute si la misma se hace o no).

Es que de todo lo que ofrece la gestión de configuración, lo que nos ofrece valor y lo que no (decidimos que hacer y cómo hacerlo).

Si o si es imprescindible realizar:

La auditoría se puede no realizar, debido a que el pensamiento ágil apunta a que el equipo es el que resuelve, decide. Y como la auditoría es independiente y objetiva, la realiza alguien externo al equipo.

Si la concepción del equipo ágil es resolver las cosas, del equipo puertas para adentro, no se necesita de nadie de afuera que controle. La auditoría de configuración es el elemento que el equipo va a decidir no realizarlo.

Es una cuestión de cultura realizarla o no. (si se la puede realizar de acuerdo al concepto del equipo)

De todas es la que más se evalúa si hace falta o no.

- ➔ Las demás actividades no se piensa si hacerlas o no directamente, solo se discute el cómo hacerlas. Si o si se llevan a cabo.

SCM en Ágil

- Es responsabilidad de todo el equipo.
- Automatizar lo más posible.
- Educar al equipo.
- Tareas de SCM embebidas en las demás tareas requeridas para alcanzar el objetivo del Sprint.

SCRUM

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

La diferencia en términos conceptuales entre una guía de SCRUM y la otra es que lo que se busca es hacer que el marco de trabajo sea aún menos prescriptivo que en la guía original.

Es decir, muchas cosas las deja planteadas / pautadas pero el resto deja que el equipo lo maneje de acuerdo a su criterio.

Valores de SCRUM

Se relacionan con los valores y principios del Manifiesto Ágil, es la base de sustento de SCRUM.

Principios de SCRUM

Time Boxing: es un principio con el que trabaja y define todo el tiempo. Se basa en que hay un tiempo fijo “punto de sprint” establecido (hay un tiempo máximo que pueden durar las cosas)

Desarrollo Iterativo: SCRUM utiliza un Ciclo de Vida Iterativo e Incremental (recordad que con distintos procesos elegimos este tipo de ciclo de vida)

Diferencias de las Iteraciones de PUD y las Iteraciones de SCRUM

- SCRUM : es un lapso de tiempo fijo, no más de un mes.
- PUD : lo que es fijo es el alcance, una vez definido el mismo se decide el tiempo de las iteraciones.

Colaboración

Auto Organización

Empirismo: es la base del proceso. Si bien tiene cuestiones prescriptivas también da lugar en el desarrollo a las decisiones del equipo.

- ➔ No hay hincapié específico en proyectos. Con lo que se puede entender que podemos usarlo para otra cosa que no sea gestión de proyectos.

Cuesta imaginar eso, pero es importante destacar que en la guía 2020 esto se omite.

Puede ser utilizado SCRUM para gestionar proyectos que no estén relacionados con software.

- ➔ Recordar: para mejorar procesos, debemos plantear proyectos.

Sprint

- Permite incluir dentro, es el contenedor, de los demás eventos
- Timeboxing definido: todo lo que se hace dentro de un sprint tiene que durar 1 mes o menos.
- Todo el trabajo para lograr el objetivo ocurre dentro del sprint.

SCRUM es un marco de trabajo que permite adaptarnos más fácilmente a los cambios.

Se define un objetivo para el sprint y se aceptan cambios en la medida que no cambien el objetivo planteado. Si se presentan hay que rechazarlos y ver si se continua con el sprint o no.

Los cambios que ocurren dentro del sprint no son de gran importancia.

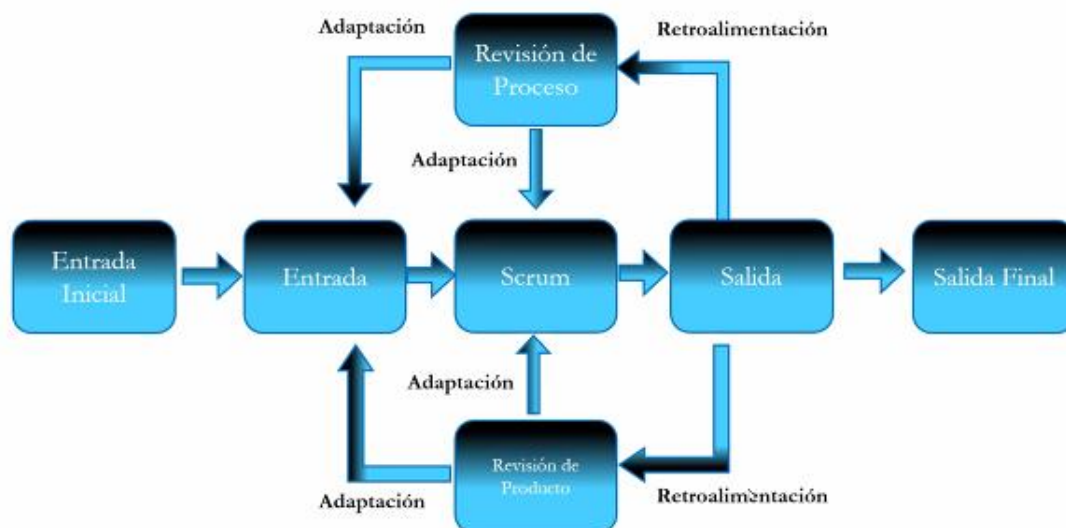
Si nosotros logramos tener la posibilidad de que el time box sea menor a un mes, vamos a tener ventajas.

Recordar: la salida del sprint debe ser software funcionando.

➔ Un aspecto importante cuando pensamos en sprint tiene que ver con que significa la ejecución dentro del mismo.

No trabajamos como una “mini cascada”, sino que ejecutamos de manera vertical. Una persona desglosa en tareas y se va ejecutando de manera vertical.

¿Cómo funciona SCRUM?



Categorías de Responsabilidades

Responsabilidades: las podemos distribuir como creamos necesario siempre y cuando se respeten las mismas.

- **Product Owner:** es una única persona
- **Scrum Master:** es el encargado de asegurar que scrum fluya, resolviendo problemas que pueda haber y asegurar que los developers y el product owner tengan las herramientas necesarias.
- **Developers:**

Eventos

Todos los eventos se realizan dentro del sprint.

Product Backlog Refinement: no se la menciona como un evento específico. Es opcional y también ocurre dentro del sprint.

Implica cambiar prioridades, refinar ítems, borrar ítems.

1. Sprint Planning

Tratar de tener US priorizadas, estimaciones realizadas para saber cuáles podemos definir dentro de un sprint.

Para determinar esto necesitamos saber: el objetivo del sprint. Y en función de eso sabemos que US incluir en el mismo.

Como salida vamos a tener el objetivo del sprint, las US del product backlog, tareas que vamos a ejecutar.

Participan: las 3 categorías de responsabilidades.

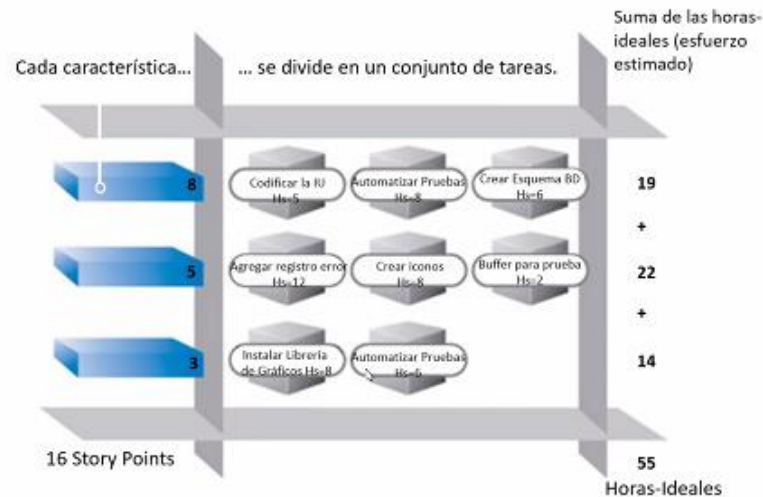
A medida que ejecutamos sprint, lo que hacemos en este evento cada vez va mejorando. Directamente aplicamos el empirismo, el equipo en función de su propio aprendizaje se adapta logrando la mejora.



➔ Quien planifica el trabajo es el que lo va a hacer.

➔ La misma persona que debe cumplir con la tarea se planifica y organiza como hacerla.

Artefacto: Sprint Backlog



- El equipo puede usar las herramientas que quiera.
- No está especificado la asignación de Story Points.

Capacidad

Utilizamos la capacidad para determinar y saber si vamos a poder llevar a cabo las características que queremos desarrollar.

- Horas de trabajo disponibles por día
Cuántas hora tenemos disponibles por día de cada una de las personas del equipo, que días están disponibles esas personas y en base a eso definimos la capacidad.

- **Horas de Trabajo Disponibles por día (WH) X Días Disponibles Iteración (DA) = Capacidad**

$$WH \times DA = \text{Capacidad}$$

Lo ideal es medir la capacidad en términos de Story Point.

Capacidad del Equipo: es lo que nosotros necesitamos definir para saber en la sprint Planning cuántas características del sprint backlog voy a poder ejecutar.

2. Daily Scrum

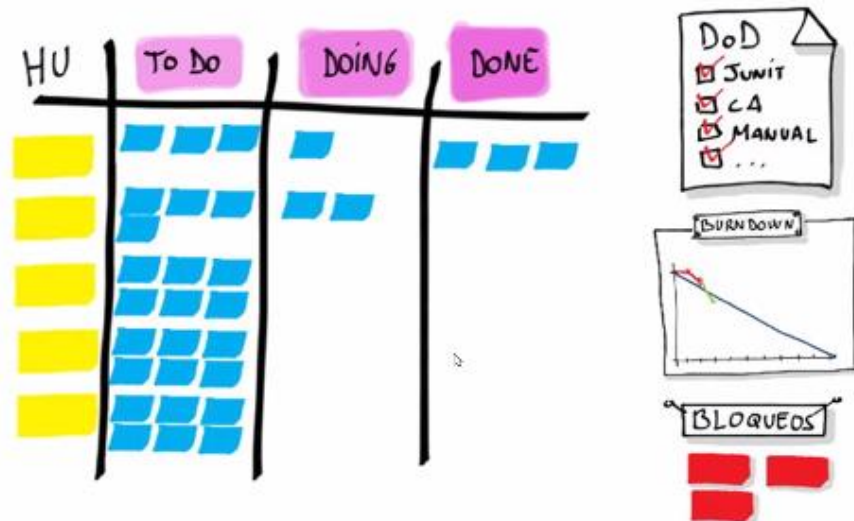
El objetivo es inspeccionar / retroalimentar y adaptar para darle transparencia a lo que estamos haciendo.

Tener en cuenta: Si el product owner está ejecutando tareas bajo responsabilidad del developer, también debería participar de esta ceremonia.

De acuerdo a la nueva guía todas las personas que forman parte del Sprint Team pueden participar de la Daily Scrum.

- ➔ Siempre revisar si nos estamos enfocando en el objetivo del sprint, de acuerdo a las tareas que vayamos realizando.

Tablero de SCRUM

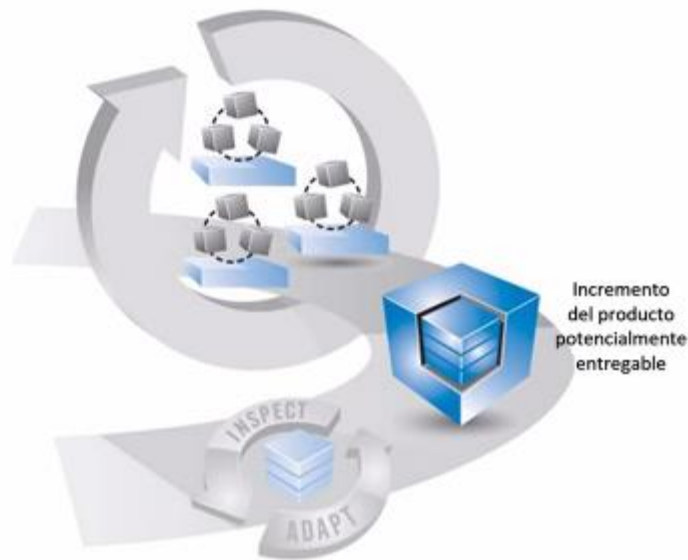


- ➔ Es un tablero donde colocamos las US y cómo vamos avanzando en su desarrollo.
- ➔ Criterio de Done: cuando el código está subido por el desarrollador, al menos un testing generado, etc.
Puede ser organizacional o si no fuera así debe definirlo el equipo.
- ➔ Las tarjetas rojas son impedimentos que tenemos y que el Scrum Master debería ayudar a los Developers y Product Owner a solucionarlos.

3. Sprint Review

Es un evento de presentación e inspección del sprint.

Artefacto: Versión del Producto



Métricas Ágiles

Regla de Oro Ágil sobre Métricas:

“La medición es una salida, no una actividad”

Recordar: medir la que sea necesario y nada más

Dos principios ágiles que guían la elección de las Métricas:

Con las métricas medimos cuanto producto funcional / software funcionando le entrego al cliente

Velocidad:

Hablamos de software entregado funcionando, lo medimos en términos de software aprobado por el Product Owner y en términos de Story Point.

- ➔ Nos permite en el ciclo de inspeccionar y adaptar, la capacidad del equipo en términos de Story Point.
- ➔ Requiere de un conocimiento y el ejercicio del equipo, ya con algunas US realizadas se obtiene mayor velocidad.
- ➔ La velocidad no es comparable entre equipos.

Capacidad:

se mide en una situación ideal en términos de Story Point, en las primeras realizaciones es más difícil identificar y trabajar con SP. Y en función de eso se determinan las US incluidas en el sprint.

Running Tested Features: (por lo general no se la utiliza)

Es una medida de las Features entregadas al cliente.

Cuenta solo características y cada una puede tener un peso muy relativo de la otra. Lo importante de hacer un gráfico es solo una medida de cuanto vamos avanzando pero no nos proporciona más valor.

¿Cuándo ocurre la Retrospectiva?

Se realiza una por cada sprint, lo que se hace es ver cuáles son los aspectos que pueden mejorarse (nos focalizamos en dos o tres), siempre en un contexto de transparencia.

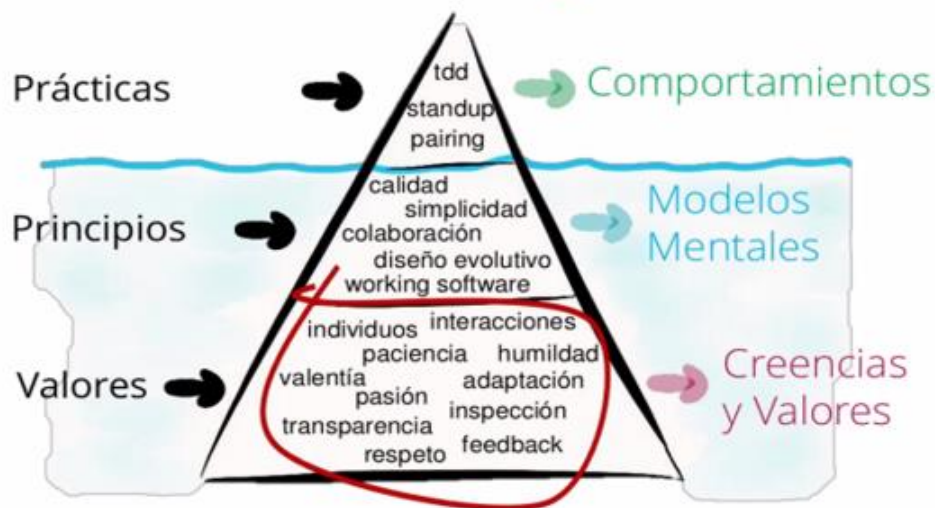
No se busca hacer catarsis ni buscar culpables, sino que definimos los aspectos que están bien y los que pueden mejorarse, es base a estos últimos definimos como vamos a hacer (que tareas y cómo vamos a realizar) para resolver esta situación.



Las diferencias del Framework 2020



Recordemos.... ¿Dónde está la Agilidad?



➔ Sin los valores implementados en la organización, ni las técnicas ni las practicas pueden llevarse a cabo.

Increment:

Vamos a tener tantos como características implementadas tenemos en el Sprint.

- Pueden entregarse antes del final del sprint.

En la Review se presenta la suma de los incrementos.

FRAMEWORKS PARA ESCALAR SCRUM

Repaso SCRUM:

Es un framework de trabajo de desarrollo de software ágil, que se utiliza para gestionar proyectos de software en equipo.

Equipo: Product Owner, Developers, Scrum Master.

Artefactos: Product Backlog, Sprint Backlog, Incremento.

Product Backlog: tiene el objetivo del producto, los ítems deben estar priorizados, se va adaptando a los cambios (aún en etapas avanzadas del proyecto).

Evento de Refinamiento: se realiza previo al planeamiento y consiste en redefinir los ítems, si hace falta reestimarlos y hasta eliminarlos o agregarlos. Es la adaptación del Product Backlog a los cambios que van surgiendo.

Es lo que necesitamos para que el Backlog se mantenga “vivo” y retroalimentado. Es una etapa opcional, de acuerdo a lo que resuelve el equipo, de no llevarse a cabo este proceso se integra en la Sprint Planning.

Sprint Planning: se toman los ítems priorizados, se hablan con el equipo, se estiman y se ve que se va a realizar durante el sprint a lo que se compromete el equipo.

Se dividen tareas y definimos la capacidad del equipo, en función de eso se construye el Sprint Backlog (definimos el objetivo del sprint, descomponemos cada una de las fixtures en tareas). Y a partir de allí empieza la ejecución del sprint.

➔ De acuerdo a la teoría:

Una vez que se arranca con la ejecución del sprint, no se puede modificar el alcance de ninguna de las fixtures incluidas en el sprint backlog, durante la ejecución del mismo.

Si bien los cambios son bienvenidos, en ese momento (en lo que dura la ejecución del sprint, máximo 1 mes), el compromiso con el Product Owner es que en el time box definido no ingresan cambios.

Es importante sobre todo en las primeras iteraciones, que es cuando el equipo va aprendiendo, intentar entregar al menos alguna de las características que no sean todas pero cumplir con el time box (entregar lo que uno llegue a entregar). Pero esto no es porque se modifique el alcance, sino porque el equipo todavía no tiene la experiencia.

En las primeras ejecuciones suelen haber errores en la definición de la capacidad del equipo o en las estimaciones.

➔ Cuando decimos que no se introducen cambios hablamos en el contexto de que por ejemplo estamos desarrollando la característica / USER STORY 1 y vienen Product Owner a hacernos cambios sobre la misma.

Existe la posibilidad de que después de realizarla nos demos cuenta de que la característica realizada no tiene valor actual para el producto, por lo que se acuerda (el PO lo decide) que el sprint se cancela teniendo en cuenta el costo que conlleva esto.

El PO debe tener la suficiente claridad de la visión del producto y el involucramiento.

Recordar: el Product Owner no es el concepto de cliente en los proyectos tradicionales, sino que es parte del equipo por lo que en cada decisión que se toma respecto al mismo este es parte y de alguna forma está comprometido con la misma.

La teoría (lo que está escrito en la guía) debe respetarse por todos los integrantes del equipo. Teniendo en cuenta lo que acuerda, hay cuestiones que son no negociables, de lo contrario no estamos trabajando con SCRUM y genera complicaciones en el equipo de trabajo.

Sprint Daily: el equipo en general establece lo que se va a realizar en el día (actual) cualquiera sea la acción que corresponda. Reporta la situación y dura aproximadamente 15 min.

Sprint Review: se realiza a inspección y adaptación sobre el producto.

Sprint Retrospective: se realiza inspección y adaptación sobre el proceso.

Recordar: los principios del agilismo son transparencia, inspección y adaptación.

Diferencia entre “Hacer Ágil” y “Ser Ágil”

“Hacer Ágil” tiene que ver con implementar las prácticas del agilismo, pero implementarlo no en términos de la cultura de la organización, los valores o principios. Es implementar las prácticas.

Los beneficios se obtienen cuando nosotros podemos incorporar la cultura de “Ser Ágil” y saber que las prácticas son la parte del iceberg que se ve, pero debajo de todo esto está la institucionalización de la cultura ágil que es lo que permite que estas prácticas se puedan materializar o cumplir pero sin todo el esfuerzo de sentir que lo que hacemos es tener “una piedra en el zapato” para cumplir con la práctica, sino que se naturaliza dentro del equipo de trabajo.

En el equipo de trabajo el PO es uno más del SCRUM Team, es clave debido a que la gestión tradicional de proyecto hay un cliente y el equipo trabaja para él.

PO: no es analista funcional y no es el cliente de la gestión de proyectos tradicionales.

¿Cómo hacemos nosotros para poner en prueba lo que se plantea en SCRUM?

En todas las técnicas y herramientas que se utilizan, lo que hace o lo que pone a prueba la efectividad de lo que estamos trabajando es que de alguna manera tenga la potencialidad de ser escalable.

Si no lo es, lo relacionado a la complejidad de desarrollar software no se resuelve.

¿Qué significa que se escalable?

De acuerdo a la guía los equipos en SCRUM están constituidos por entre 3 o 9 participantes (regla practica de 7 más menos 2)

Todos los productos que queramos construir con esta cantidad de personas, muchas veces, no es posible hacerlo. La realidad es que si pensamos en un producto grande necesitamos más personas para su realización.

De esta manera surge el escalamiento: poder trabajar con lo que plantea SCRUM y los beneficios y bondades, en un contexto donde se tenga equipos más grandes.

¿Por qué Escalamos?

Poder trabajar con metodologías ágiles y herramientas como SCRUM en un proyecto de software para el desarrollo de un producto, con equipos de mayor dimensión de lo que el framework plantea y aumenta la complejidad del contexto sobre el cual estamos parados.

Framework CYNEFIN

Se plantea es que, de acuerdo a los entornos en los cuales nos encontramos, vamos a tener distintas formas de resolver o encarar la complejidad (o escenario que el entorno plantea).

1. Contexto de Entornos Simples:

Los entornos simples funcionan de acuerdo a una causa y una única consecuencia.

Implementando las “Mejores Prácticas” si sabemos que ante una causa siempre tenemos las mismas consecuencias, es muy fácil de resolver.

Ejemplo: tengo dolor de cabeza, el dolor se me pasa tomando una pastilla.

2. Contexto de Entornos Complicados:

En los mismos se da la situación de que para una causa también tenemos efectos, pero en realidad tenemos más de uno.

Lo que requiere utilizar “Las Buenas Prácticas” para abarcar, a partir de una causa, los posibles efectos que tenemos identificados y poder resolverlo.

➔ Estos dos contextos no se relacionan ni tienen sentido con lo que plantea SCRUM para resolver la problemática del software.

Lo que se plantea en el framework de scrum para poder resolver la problemática del software es lo que entra en el cuadrante de contexto de entornos complejos.

3. Contexto de Entorno Complejo:

Implica que en este cuadrante de escenario, no se puede asociar una causa a un efecto o un efecto a una causa. Nos terminamos dando cuenta de esta relación probablemente después de que ocurran.

La calve de este tipo de escenario está relacionado a lo que propone SCRUM, la inspección y adaptación: las “Prácticas Emergentes”. Implica que no hay manera de resolver de ante mano, a partir de una causa cuales van a ser sus efectos, sino que a partir de lo que realmente va pasando vamos adaptando el escenario. Es decir, a partir de las prácticas que emergen hacemos el análisis de las relaciones causa efecto para poder irnos acomodando en esos escenarios.

4. Contexto de Entorno Caótico:

Son escenarios donde lo que uno intentaría es tratar de buscar “Prácticas Novedosas”, ideas, lo que se nos ocurra para salir de ese escenario.

Esto quiere decir que en verdad, en los escenarios caóticos, no hay nada que se pueda resolver (uno en esos contextos no debería pasar demasiado tiempo) y las prácticas novedosas son para salir del caos que no puede manejarse.

Ejemplo: situación año 2020 pandemia, no hay nada que alguien pueda hacer, se buscan ideas para salir de la situación.

5. Desorden:

Es la falta de información, no contamos con la información necesaria para catalogar nuestra situación en alguno de esos cuatro escenarios. Por lo que directamente no podemos trabajar, necesitamos saber dónde estamos parados.

- ➔ En el contexto de este framework para analizar la complejidad, cuando trabajamos con SCRUM o con los Framework para Escalar SCRUM, estamos parados en los escenarios COMPLEJOS.

Con la particularidad de que si trabajamos con SCRUM estamos hablando de escenarios Complejos y cuando hablamos de Escalar Scrum, seguimos hablando de escenarios complejos pero más complejo. Debido a que se agregan variables que utilizando el Framework tal como está planteado no existe.

¿Hacemos Proyectos o Productos?

Nosotros construimos productos y utilizamos proyectos debido a que son una unidad de gestión que nos sirve para guiar la construcción de ese producto.

- ➔ Debemos tener en claro que lo que construimos cuando hablamos de generar software, es que construimos productos. Es importante remarcarlo y tenerlo en cuenta porque SCRUM es un framework que nos permite gestionar proyectos para finalmente crear productos de software.

Frameworks

1. SAFE – DEAN LEFFINGWELL

Es uno de los primeros Frameworks que se crearon, posee una complejidad bastante importante, es más organizacional no está circunscripto solamente a lo relacionado con la escalabilidad de SCRUM.

Plantea el concepto de la arquitectura, es como una de las debilidades que tiene SCRUM corre el riesgo de perder de vista la arquitectura que también debe ser definida en términos de escalabilidad de acuerdo a lo que será el sistema en un futuro.

Es el único framework que plantea como un porfolio de productos, a diferencia del resto que solo trabajan con un único Product Backlog.

Es bastante complicado de entender, lo que un poco en contra de lo que se supone deberían ser los Frameworks ágiles.

2. LESS – CAIG LARMAN

Tiene dos versiones, básica que pueden trabajar hasta dos equipos y otra más completa para más de 8 equipos.

3. SCALE SCRUM – JEFF SUTHERLAND

Es más engorroso y confuso en algunas cosas.

4. NEXUS – KEN SCHWABER

Es muy similar a SCRUM es como poner una capa de este sobre otra.

Hace foco a la integración entre todos los equipos que trabajan y la minimización de la dependencia entre el trabajo de todos los equipos.

NEXUS

- ➔ Es un framework que está planteado para escalar SCRUM con un único Product Backlog, (o lista de productos).

Se plantea además que la ceremonia / evento de Refinamiento es obligatoria no opcional como en SCRUM (es exactamente la misma).

Permite trabajar con al menos 3 a 9 Scrum Team.

Incremento del Producto: es un incremento del producto para todos los equipos que están trabajando. Es decir, no hay pedazos de incremento de producto para cada uno de los equipos, sino que el artefacto incremento del producto, una vez terminada la ejecución del sprint es UNO solo.

Con lo que se produce una pequeña modificación en relación a lo que plantea SCRUM para todos los equipos relacionada a la ceremonia de Review, es la única que se lleva a cabo en conjunto (todos los scrum team) participan en la misma y no se realizan luego individuales por cada equipo.

En general las demás ceremonias como en la planificación, en la que se hace una planificación del sprint pensando en cómo vamos a particionar la parte del Product Backlog que vamos a hacer en cada uno de los equipos.

Se la denomina Planificación del Sprint Nexus, planificamos sobre que vamos a trabajar y lo dividimos en cada uno de los equipos, después cada uno de los equipos realiza su propia planificación en base a lo que se acordó entre todos. Sucede esto también con la Daily y con la Retrospective.

ROLES

- **Product Owner**
- **Scrum Master**
- **NIT Members**
- **Nexus Integration Team (NIT)**

NEXUS INTEGRATION TEAM

Es un rol que en SCRUM no existe, tiene como función resolver los aspectos relacionados a la integración y dependencias que surgen producto de que muchos equipos están trabajando sobre un mismo Product backlog para obtener un mismo incremento del producto.

Composición: Product Owner, SCRUM Master y 1 o más NIT Members.

Los miembros del NIT pueden ser miembros de equipos Scrum Individuales o se puede armar un equipo con personas aparte.

Generalmente se utilizan miembros de cada uno de los equipos individuales, pero debe estar establecido que la prioridad mientras pertenecen al NIT es trabajar dentro de ese contexto.

Esto hace que la composición del NIT vaya cambiando en el tiempo en función de las necesidades.

Ejemplo: si en algún momento del tiempo se debe trabajar en integración y dependencia del manejo de la arquitectura, el NIT puede estar compuesto por miembros que tengan mayor experiencia en lo relacionado con la definición de la arquitectura.

Por esta razón va cambiando de acuerdo a la prioridad de integración y disminución de dependencia con la que se trabaja en el momento.

➔ Es un rol que no existe en SCRUM, esta solamente planteado por este framework.

Dependencias: son perjudiciales aquellas que condicionan el avance de un equipo a lo que otro equipo pueda estar haciendo. Cuando se convierte en un impedimento debe disminuir y resolverse.

Integración: es muy importante debido a que en el framework se trabaja en la integración de un Product Backlog por parte de todos los equipos. Es muy difícil si no se tiene integración continua llevar a cabo esta forma de trabajo.

Responsabilidades:

- Declaración del DOD aplicable al incremento integrado.
- Coaching sobre estándares de desarrollo, arquitectura e infraestructura.
- Consultoría.
- Generación de advertencias sobre dependencias y problemas entre equipos.
- Puede trabajar en el Product Backlog.

EVENTOS

- **Refinamiento**

Se plantea como obligatorio, a diferencia de lo expuesto en Scrum que era opcional.

- **Nexus Sprint Planning**

Primero se lleva a cabo la partición para obtener el objetivo del Sprint Backlog de Nexus para cada uno de los equipos, y luego a partir de allí cada uno de los equipos realiza su Scrum Planning.

- **Nexus Daily Scrum**

Se llevan a cabo por un lado la Daily Scrum habitual de cada equipo y por el otro está la Daily Nexus, cuyo foco es la integración y dependencia relacionada a la información que se comparte entre los distintos equipos.

Normalmente primero se lleva a cabo la reunión entre cada equipo y luego la Daily Nexus. Es importante que los miembros del NIT sean parte de los equipos individuales permitiendo que cuando se charle entre todos los equipos se tengan ideas claras de lo que se habló individualmente.

- **Nexus Sprint Review**

La Review desaparece individualmente de cada uno de los equipos, se realiza una sola donde está resuelto el incremento del producto (suma) de todos los incrementos que realizaron los equipos.

Por esto también se habla de unificar el criterio de la Definición de Terminado, debe ser aplicable al incremento integrado (es el artefacto como uno solo)

- **Nexus Sprint Retrospective**

Consta de tres partes:

1. Los representantes del NIT identifiquen los problemas que pueden haber afectado a más de un equipo. Que de alguna manera está relacionado con la integración y las dependencias.
2. Cada equipo Scrum realiza su propia Sprint Retrospective y además de abordar las cuestiones específicas del equipo, abordan aspectos relacionados a la integración y dependencia.

¿Se dejaron trabajos sin realizar? ¿Nexus generó deuda técnica?

¿Todos los artefactos se integran con éxito al menos una vez al día?

¿El software se desplegó con la frecuencia suficiente para prevenir la acumulación abrumadora de dependencias sin resolver?

Deuda Técnica: el producto funciona pero hay cosas que se deben acomodar, emprolijar, documentar, terminar cosas para que tenga mejor performance.

3. Representantes apropiados de Scrum Team se reúnen y acuerdan como visualizar y rastrear las acciones identificadas.

ARTEFACTOS

- **Product Backlog**
- **Nexus Sprint Backlog**
- **Incremento Integrado**

➔ Conceptualmente, es como poner una capa de SCRUM sobre otra teniendo en cuenta que los artefactos son los mismos, los eventos iguales, solo que a partir del nuevo rol “NIT” que es el que resuelve en los eventos los problemas claves de integración y dependencias.

Estos últimos son de alguna manera los problemas que Nexus plantea que no se pueden resolver cuando cada uno de los equipos trabaja de manera individual por así decirlo.

Conclusión

MODELO DE MADUREZ LEAN AGILE “SHU-HA-RI”

Es un modelo que está relacionado a la filosofía ágil.

Plantea cual debería ser el camino a seguir para ser capaces, a partir de nuestro aprendizaje, de definir las reglas o nuevas reglas.

1. Seguir las Reglas

Antes de pensar cómo vamos a hacer las cosas y modificar o adaptar lo que ya está definido. Primero debemos aprender bien lo que está definido, dominarlo, entenderlo, usarlo, para tomar la experiencia que implica seguir las reglas.

2. Romper las Reglas

Una vez que esa instancia está superada, podemos hablar de “romper las reglas” implica elegir que usar, que no y en que contexto.

Normalmente las organizaciones ven el framework y adaptarlo para definir que se usa y que no.

3. Hacer las Reglas

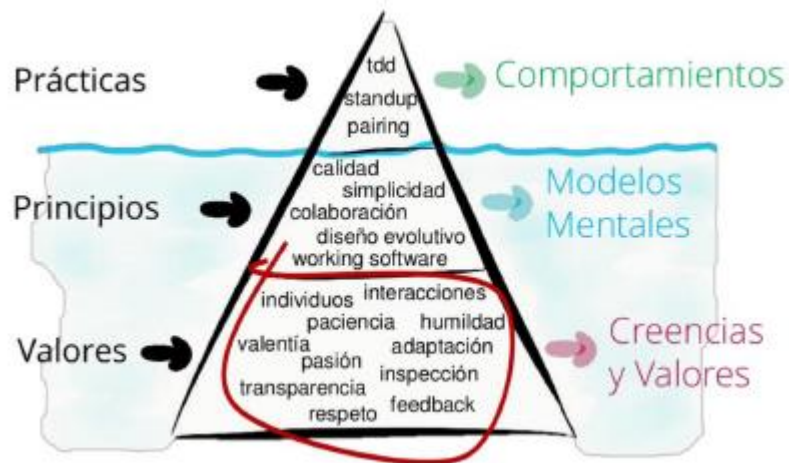
SHU-HA-RI - Madurez Lean-Agile



➔ De esto surge también el hecho de tener cuidado con implementar las prácticas sin antes haber hecho un cambio en la cultura y los valores de la organización.

Para que funcione lo que se plantea en Scrum y de beneficios, la base de la pirámide es cambiar la cultura y valore para que cada uno de los integrantes de la organización tengan estos valores.

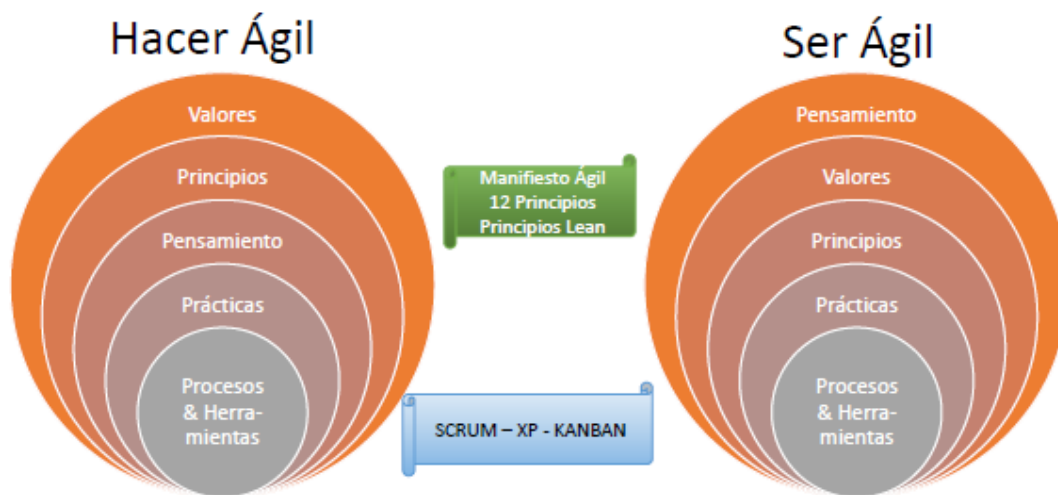
De lo contrario no lo que se plantea NO es Scrum y tampoco Nexus.



Hacer Ágil: nos paramos en el contexto simplemente de implementar métodos y prácticas ágiles. Estamos haciendo algo para lograr la agilidad.

Nos sirve para tener artefactos, definir las prácticas, pero no tenemos incorporado el pensamiento relacionado con Ser Ágil a nivel organizacional.

Ser Ágil: implica cambiar la mentalidad, el estilo de liderazgo. Es un cambio profundo que a su vez se promueve en todos los materiales relacionados al manifiesto ágil, principios y valores.



Aclaraciones:

USER STORIES

Si no consume tiempo podemos incluir la investigación dentro de la US como una tarea. Para que sea una SPIKE debemos tener la duda que nos impida realizar la US.

SCRUM

En SCRUM es o todo o nada, si no terminamos el desarrollo de una US para un Sprint, se cancela la misma y se la deja para realizar en la siguiente etapa.

No hay ninguna restricción dentro del Framework que implique que tareas debemos incluir, realizar en la Planning del Sprint. Las US deben cumplir con el criterio de READY. Lo más normal es que si se necesita refinamiento lo hagamos en la ceremonia del Refinamiento que se puede realizar a lo largo de todo el Sprint. No entra dentro de la Sprint Planning debido a que las US ya deberían estar refinadas.

Puede ocurrir que algunas actividades como la deuda técnica, a veces se implementan cosas o desarrollan que en términos de arquitectura hay que refinar, por lo que el tiempo que queda lo podemos dejar para terminar esto. La arquitectura de un producto no necesariamente se define claramente en un inicio, en algún momento hay cosas que se deben acomodar. No se planifica por adelantado pero se realiza en los espacios de tiempo disponibles antes de terminar la ejecución del Sprint.

LINEA BASE:

Es un pila fundamental del SCM, frente a determinados momentos del tiempo si es necesario introducir cambios los mismos deben ser aprobados por un comité.

Se busca siempre mantener la integridad del producto.

En un determinado momento del tiempo podemos colocar una etiqueta a una configuración del producto que si por algún motivo tenemos problemas de integridad podemos volver a esa version anterior que está integrada y controlada.

Ítem de Configuración estable: la definición de cuando, tiene que ver con lo definido en el plan de configuración (se define el criterio de en qué momentos vamos a tener líneas base).

Ejemplo: tenemos los ítems de configuración a los que les colocamos una etiqueta de línea base, seguimos trabajando, los ítems evolucionan. Si queremos volver a esa etiqueta, version estable del producto.

Importante: diferencia entre la SCM tradicional en procesos definidos, dentro de lo que vemos también hay que revisar cuales se trasladan a un ambiente ágil y cuáles no.