

# Resumen - 1er Parcial

## ▼ Contenidos (Según modalidad 2022)

### Unidad 1

- ☒ Introducción a la Ingeniería del Software. ¿Qué es?
- ☒ Estado Actual y Antecedentes. La Crisis del Software.
- ☒ Disciplinas que conforman la Ingeniería de Software.
- ☐ Ejemplos de grandes proyectos de software fallidos y exitosos.
- ☒ Ciclos de vida (Modelos de Proceso) y su influencia en la Administración de Proyectos de Software.
- ☒ Procesos de Desarrollo Empíricos vs. Definidos.
- ☒ Ciclos de vida (Modelos de Proceso) y Procesos de Desarrollo de Software
- ☐ Ventajas y desventajas de c/u de los ciclos de vida. Criterios para elección de ciclos de vida en función de las necesidades del proyecto y las características del producto.
- ☒ Componentes de un Proyecto de Sistemas de Información.
- ☒ Vinculo proceso-proyecto-producto en la gestión de un proyecto de desarrollo de software

### Unidad 2

- ☒ -Manifiesto Ágil/Filosofía Lean
- ☒ Requerimientos en ambientes lean-ágil
- ☒ Introducción al Desarrollo Ágil.
- ☒ Requerimientos en ambientes ágiles—User Stories
- ☒ Estimaciones en ambientes ágiles
- ☒ Frameworks de SCRUM a nivel equipo y escala
- ☒ Métricas Ágiles
- ☒ Gestión de Productos de Software—Planificación de Productos—Herramientas para Definición de Productos de Software
  - ☐ Lean UX
  - ☐ Design Thinking

### Unidad 3

- ☒ Conceptos Introductorias de la Gestión de Configuración.
- ☒ Versiones, variantes, release.
- ☒ Planificación de la Gestión de Configuración de Software.
- ☒ Actividades relacionadas con la Gestión de Configuración.
- ☒ El rol de las líneas base y su administración.

- ✓ ~~Elementos de configuración del Software.~~
- ✓ ~~Identificación de ítems de configuración en la Configuración de un software.~~
- ✓ ~~Gestión de Configuración en ambientes ágiles~~
- ✓ ~~Continuous Integration~~
- ✓ ~~Continuous Delivery~~
- ✓ ~~Continuous deployment — Estrategias de deployments — Canary Deployments — Blue/Green~~
- ✓ ~~Deployment~~

## Unidad 1

### ▼ Software



**Software** es un set de programas junto con la documentación que lo acompaña necesaria para desarrollar y mantener los programas ejecutables que se entregan al cliente.

Esta definición contempla:

- Código
- Archivos de configuración de la ejecución
- Documentación para el usuario
- Documentación del sistema
- Herramientas utilizadas para la construcción

Existen tres tipos básicos de software:

- Utilitario
- System software
- Software de aplicación

No hay manera de comparar software y manufactura por los siguientes motivos:

- El software es menos predecible, es intangible
- No hay producción en masa, casi ningún producto de software es igual a otro, incluso cuando la idea resulte similar
- No todas las fallas son errores
- El software no se gasta, puede dejar de tener vigencia porque los requerimientos del negocio van cambiando y se tiene que adaptar pero no se desgasta en términos de materiales
- El software no está gobernado por las leyes de la física
- No se construye software en una línea de producción.

### ▼ Ingeniería de Software

Es la disciplina de la ingeniería que se preocupa de todos los aspectos de la producción de un software; **desde** las primeras etapas de la especificación **hasta** el mantenimiento del sistema una vez operando.

## **|** *Multiperson construccion of multi-version software* - Parmas, 1987

De aquí, dos conceptos claves:

- Funcion del ingeniero: Los ingenieros aplican teorías, métodos y herramientas de la manera más conveniente siempre tratando de descubrir soluciones a los problemas teniendo en cuenta que deben trabajar con restricciones financieras y organizacionales por lo que buscan soluciones contemplando estas restricciones
- Disciplinas que conforman la ingeniería de software:
  - Técnicas: ayudan a construir el producto. Ej: Análisis, diseño, implementación, prueba, despliegue, etc.
  - De gestión: planificación, monitoreo, control
  - De soporte: gestión de configuración, métricas, aseguramiento de la calidad

La ingeniería de software nace tras las crisis del software, en la cual existía (y existe) un conjunto de dificultades o errores ocurridos en la planificación, estimación de los costos, productividad y calidad de un software, debido, principalmente, a la baja eficacia que presentan una gran cantidad de empresas al momento de desarrollarlo.

### ▼ Crisis del software

El término crisis del software hace referencia a un conjunto de hechos relativos al software, planteados en la conferencia de OTAN en 1968 por Friedrich Bauer, quien recalco la dificultad para generar software libre de defectos, fácilmente comprensibles y que sean verificables. Sin embargo, este término fue utilizado anteriormente por Dijkstra en El Humilde Programador.

Causas:

- Evolución del hardware que permite crear sistemas mas complejos no acompañada con una evolución de los procesos de desarrollo de software.
- Demanda creciente de sistemas complejos, difíciles de estimar, con muchas solicitudes de cambios.
- Falta de una disciplina que intervenga en los aspectos de la producción del software

## **Proceso de software**

### ▼ Concepto

El Proceso de Software (el que transforma ideas/necesidades en un producto 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 y el proceso debe ser explícitamente modelado si va a ser administrado.

**Objetivo:** obtener un producto de software o un servicio asociado

**Inputs:** Son dinámicos. Incluyen requerimientos pero tambien personas, materiales, energía, equipamiento y procedimientos

Como ingenieros en sistemas de información, trabajamos en una industria que es humana-intensiva, significa que el software lo hacen personas y que el aporte más importante para que el producto llegue a las manos de los usuarios interesados, es el aporte que hacen las personas, y de hecho en términos de costos, lo más caro de hacer software es pagarles a las personas que participan del proyecto.

Según la IEEE, un **proceso** es una secuencia de pasos ejecutados para un propósito dado. Mientras que un **proceso de software** es 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 vamos a encontrar la parte de procedimientos y métodos, herramientas y equipos y personas con habilidades, entrenamiento y motivación. Estos tres elementos son importantes y conforman esta visión de proceso de software.

#### ▼ Clasificación

Los procesos (a nivel general) se dividen en procesos definidos y procesos empíricos:

Los **procesos definidos**, inspirados en las líneas de producción, asumen 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. Estos plantean la necesidad de tener definidos ampliamente paso a paso que se debe hacer en cada momento. En la industria del software, un ejemplo claro es el PUD (Proceso Unificado de Desarrollo) que lo que hace es definir paso por paso, que es lo máximo para poder hacer un producto de software, este proceso se adapta a cada situación en particular.

ENTRADAS IDENTIFICADAS → PROCESO DEFINIDO → SALIDAS ESPERADAS

Por otro lado, los **procesos empíricos** asumen procesos complicados, con variables cambiantes (no pueden ser definidas o controladas en su totalidad). Cuando se repite el proceso, se puede llegar a obtener resultados diferentes; la administración y control se realiza a través de inspecciones frecuentes y adaptaciones. Son procesos que trabajan bien con procesos creativos y complejos. Tienen un fuerte arraigo en el aprendizaje y la experiencia. Esto no es tan fácil de conseguir, porque la experiencia es basarse en algo anterior para hacer lo actual. Estos surgen en contraposición a los procesos definidos.

Ciclo: ASUMIR → CONSTRUIR → RETROALIMENTAR → REVISAR → ADAPTAR → ASUMIR

Los procesos empíricos trabajan basados en una hipótesis, asumen que el proceso puede funcionar de alguna manera, hacen algo, obtienen realimentación, revisan y si algo no salió bien, adaptan y comienzan el ciclo nuevamente, ganando experiencia.

#### ▼ Optimización de procesos

En un **proceso definido** claramente si tengo las mismas entradas y busco las mismas salidas, para poder mejorar el proceso voy a ir a cada una de las etapas intermedias y las voy a tratar de optimizar y necesariamente el resultado va a ser óptimo si optimizo las etapas intermedias.

En cambio, en los **procesos empíricos**, el punto de mejora del proceso difícilmente sea tan tangible, voy a tener que ir atacando diferentes aristas para poder optimizar el proceso y principalmente centrarme en las personas, quienes son la clave del empirismo y quienes capitalizan la experiencia; a esa capitalización de la experiencia la voy a poder hacer en un ciclo de realimentación, en el cual yo voy a tomar información, voy a contrastarla con lo que esperaba obtener, contra mi hipótesis y en base a eso voy a readaptar ese proceso y lo voy a modificando para poder mejorarlo.

#### ▼ Visibilidad y repetición

Otra de las diferencias entre ambos procesos es la intención que tienen los procesos definidos de ser repetibles, y esa repetibilidad la quieren lograr para tener visibilidad, es decir para saber qué es lo que pueden esperar en cada momento y de tiempo y predecir que lo que va a pasar, estos tienen la “ilusión” de tener control. Estos procesos al ser más formales, más explícitos y tener más definiciones sobre las cosas, categoricen como procesos definidos.

Los procesos definidos intentan ser procesos completos, que describen la mayor cantidad de cosas posibles que se deben hacer para desarrollar software, en contraposición a los procesos empíricos que son procesos que no están completos, esto deja que cada equipo al iniciar un trabajo decida basado en la experiencia que es lo que quiere hacer, cuando y como. El empirismo se basa en ciclos de entrega cortos para poder generar realimentación que sirva como experiencia para evolucionar y seguir avanzando.

Los procesos empíricos dicen que la experiencia es aplicable al mismo equipo, este equipo puede generar su propia experiencia en este proyecto, en este contexto particular para los distintos ciclos, pero la experiencia de ese equipo no se puede extrapolar en otros equipos, proyectos, que es lo que si esperan los procesos definidos. Los procesos definidos esperan repetibilidad.

### Ciclo de vida

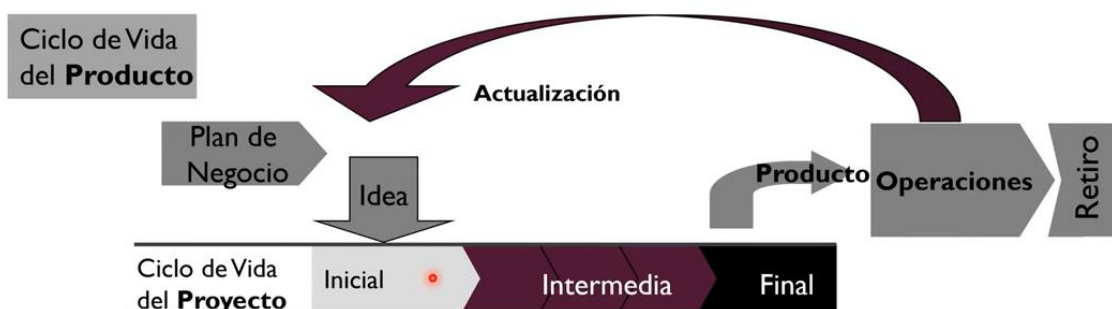
#### ▼ Proceso ≠ Ciclo de vida

Para un proyecto determinado, se define un ciclo de vida a utilizar en el proceso. El proceso define el paso a paso para alcanzar un objetivo, el ciclo de vida define cómo se van a realizar dichas actividades, su orden, duración y (si así lo establece el ciclo de vida), repeticiones.

Ciclo de vida y proceso se complementan pero son asuntos diferentes.

#### ▼ Relación ciclo de vida del proyecto y del producto

### RELACIÓN: CICLO DE VIDA DEL PROYECTO Y DEL PRODUCTO



Son distintos los ciclos de vida para un proyecto que para un producto. Un ciclo de vida de un producto inicia con la idea de crear un producto de software y termina cuando a ese producto de software lo retiro del mercado. En un ciclo de vida de un producto puede haber **n** ciclo de vidas de un proyecto

#### ▼ Ciclos de vida de un proyecto de software

Hay tres tipos básicos de ciclos de vida para los proyectos de desarrollo de software:

- **Secuencial:** se basa una etapa después de otra y no tienen generalmente retorno. Ej, en cascada.
- **Iterativo e incremental:** la mayoría de los procesos empíricos implementan este tipo de ciclo de vida para poder dar instancia de adaptación y mejora del proceso.
- **Rekursivos:** se utilizan para casos bastantes particulares como proyectos que tienen bastantes riesgos. Ejemplo: ciclo de vida en espiral que se basa en la mitigación de riesgos.

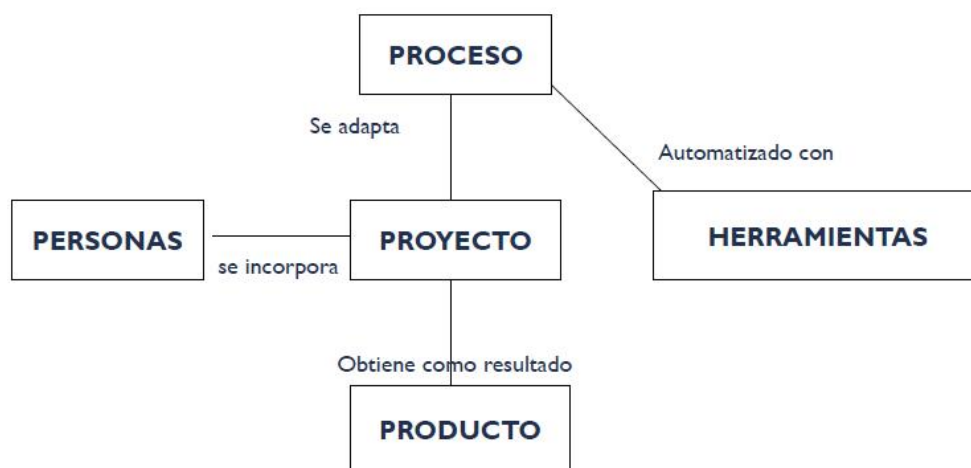
Según las características del proyecto y del producto que se deba construir, se va a elegir el ciclo de vida que vayan a implementar para poder llevarlo delante de la mejor forma posible.

***El proceso es una implementación del ciclo de vida que tiene como objetivo final construir un producto de software.***

Procesos definidos se pueden combinar con cualquier tipo de ciclo de vida, los procesos empíricos, como requieren de su constante inspección y adaptación solo se combinan con el ciclo de vida iterativo e incremental.

## Proyectos de Software

### ▼ Componentes



### ▼ Proceso

El proceso de una plantilla, una definición abstracta que se materializa a través de los proyectos donde se adapta a las necesidades concretas del mismo. Es el nivel más abstracto donde está escrito en términos teóricos que es lo que se debería hacer para hacer software. Debe adaptarse en caso de ser un proceso definido o completarse con las personas involucradas en caso de ser un proceso empírico.

El proceso se define en término de roles porque para cada proyecto, las personas asumen uno o más roles.



**El proceso se instancia en cada proyecto, se instancian los roles definidos en el proyecto, en el proceso. En el momento en el que instancio el proceso, elijo el ciclo de vida, que gestiona las personas y los recursos para obtener finalmente el producto**

#### ▼ Proyecto

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

El proyecto es la unidad organizativa, la unidad de gestión, y para poder existir necesita de personas, recursos y necesita de un método de cómo hacer el trabajo, esto lo define el proceso.

El proyecto es una unidad de gestión, es un medio por el cual yo administro los recursos que necesito y las personas que van a formar parte, para obtener como resultado un producto o servicio de software.

El proyecto comienza incorporando a las personas que van a asumir distintos roles definidos en el proceso.

#### ▼ Producto

Producto es el software, conocimiento empaquetado, no es solo código instalado en una computadora funcionando, la definición de ese producto es software, es decir, la base de datos, el diseño, las user stories son software. Todo lo que sale como ejecución de las tareas de un proyecto, es software.

#### ▼ Características

Los proyectos se caracterizan por:

1. **Orientación a un objetivo:** (esto permite que los proyectos sean únicos, para ello el objetivo tiene que ser claro y alcanzable), tengo que poder definir hacia donde voy y poder establecer/medir cuando alcancé ese objetivo para poder dar por finalizado el proyecto; estos objetivos son los que guían el proyecto, no deben ser ambiguos y deben ser también alcanzables. Los proyectos tienen un objetivo único, es decir que es distinto de los productos resultantes de otros proyectos. Cada resultado de un proyecto, es distinto del resultado de otro proyecto.
2. **Duración limitada:** son temporarios, cuando se alcanza el/los objetivo/s, el proyecto termina; una línea de producción no es un proyecto. El proyecto tiene un inicio y un fin.
3. **Tareas interrelacionadas basadas en esfuerzos y recursos para alcanzar el objetivo:** tienen precedencia, tienen vínculos entre ellas, que una tarea depende de otra y que a su vez se le asocian/designan esfuerzos y recursos lo cual hace que la gestión de proyectos sea una tarea compleja. Dividimos todo el trabajo a realizar en tareas que tienen una relación en término de que algunas pueden hacerse en conjunto o unas dependen de otras. La definición de que tareas hay que realizar se obtienen del proceso.
4. **Son únicos:** todos los proyectos por similares que sean tienen características que los hacen únicos. Cada resultado de un proyecto es diferente al resultado de otro proyecto.

#### ▼ Administración de proyectos

En la gestión tradicional de proyectos (aplicada a proyectos que adaptan procesos definidos) es la aplicación de conocimientos, habilidades, herramientas y técnicas a las actividades del proyecto para

satisfacer los requerimientos del proyecto y poder obtener como resultado el producto esperado, para ello voy a administrar todos los recursos que me dieron para el proyecto, organizar el trabajo de la gente afectada y hacer un seguimiento de si las cosas se están dando como estaban planificadas. Tenemos una figura de líder de proyecto que es el responsable de hacer todo lo mencionado.

Administrar un proyecto incluye:

- Poder definir los requerimientos, el alcance del producto.
- Establecer objetivos claros y alcanzables.
- Adaptar las especificaciones, planes y el enfoque a los diferentes intereses de los involucrados (stakeholders)

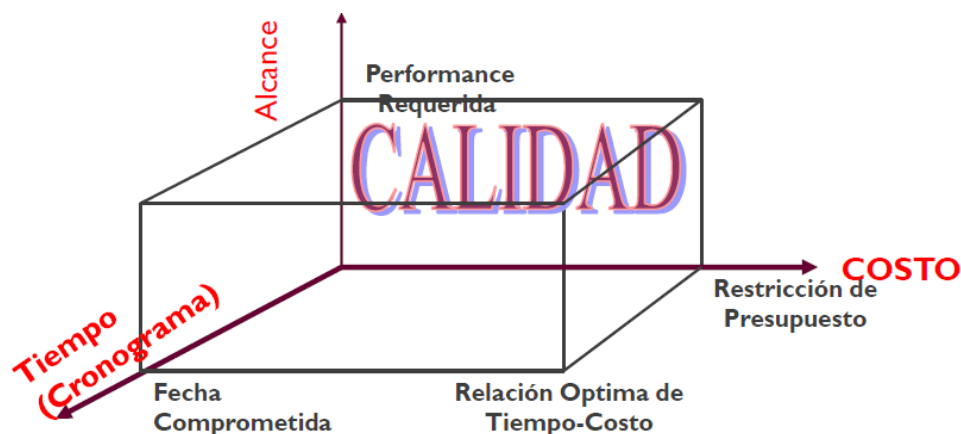
#### ▼ Triple restricción

La gestión tradicional habla sobre la necesidad de que el líder del proyecto encuentre un equilibrio entre 3 factores que actúan como restricciones del proyecto.

Esta triple restricción plantea que en un proyecto se van a tener restricciones de:

- Tiempo
- Costo o presupuesto
- Alcance

**La calidad no es negociable**



Se la define como triple restricción ya que las tres variables están íntimamente relacionadas entre sí de tal manera que, si yo modifico alguna de las variables voy a tener que hacer variar alguna de las otras variables, o ambas.

#### ▼ Líder de proyecto

En la gestión tradicional tenemos un equipo de proyecto y un líder de proyecto. El líder es quien se relaciona con los demás involucrados; es conductista, le dice a la gente que es lo que tiene que hacer, para cuando lo tiene que hacer, y el equipo recibe las asignaciones del trabajo y trabaja. Informa cuando ya está terminado, informa cuanto se demoró, informa el porcentaje de avance y si hay una desviación respecto a los planes informarlo al líder para que tome ciertas correcciones.

Dentro de este enfoque, como líderes tenemos la responsabilidad, estimar, planificar, asignar trabajo a la gente y hacer seguimiento de lo que está pasando. No solemos tener trabajo técnico,



ya que, si tuviéramos trabajo técnico y de gestión, se suele priorizar el técnico dejando de lado el de gestión.

Como líderes de proyecto debemos tener un plan de proyecto, un mapa que nos va a guiar durante todo el proyecto. Este es el artefacto resultante de la planificación.

#### ▼ Equipo del proyecto

El equipo de proyecto, por otro lado, es un grupo de personas comprometidas en alcanzar un conjunto de objetivos de los cuales se sienten mutuamente responsables. Una de las características fundamentales del equipo de proyecto es tener la posibilidad de trabajar en equipo y desarrollar sinergia; normalmente se buscan que sean equipos pequeños ya que hay cuestiones de comunicación que teniendo un equipo muy grande no se consiguen lograr.

#### ▼ Plan de proyecto

| Un plan es a un proyecto lo que una hoja de ruta a un viaje

En un plan de proyecto se debe definir:

- Alcance del proyecto → ¿Qué se va a hacer?
- Calendarización → ¿Cuándo se va a hacer?
- Recursos y decisiones → ¿Cómo se va a hacer?
- Asignación de tareas → ¿Quién lo va a hacer?

Asimismo, deben tomarse las decisiones respecto a:

- Alcance del proyecto
- Proceso y ciclo de vida
- Estimación
- Gestión de riesgos
- Asignación de recursos
- Definición de métricas

Cuando se realiza la definición del alcance, puede tratarse de:

- **Alcance del producto:** Son todas las características que pueden incluirse en un producto o servicio. Sumatoria de todos los requerimientos funcionales y no funcionales que el producto tiene que satisfacer, esto se define/mide en la ERS (Especificación de requerimientos de software).
- **Alcance el 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, este alcance se define/mide en el plan de proyecto o Plan de Desarrollo de Software; este plan cuenta con la definición del ciclo de vida, la estimación, la gestión de riesgo, la definición de las métricas, etc.

La relación que hay entre ambos es que, si el producto que yo tengo que construir es grande, las tareas a definir en el proyecto van a ser más o van a requerir más tiempo y recursos, por lo tanto,

para definir el alcance del proyecto debo primero definir el alcance del producto.

Proceso  $\neq$  Ciclo de vida  $\rightarrow$  Cuando inicia el proyecto debo definir qué proceso quiero usar y que ciclo de vida voy a utilizar. Dentro de esta gestión tradicional, podemos elegir el proceso y cualquier ciclo de vida para llevar a cabo el proceso, en gestión ágil si hay limitaciones (el único ciclo de vida que puede utilizarse es el iterativo).

#### ▼ Estimaciones de software

La estimación no se aborda directamente en la planificación del proyecto y el cronograma sino que sirve de input para la planificación. La definición de compromisos, fechas y objetivos tienen que ver con otras variables y no solo con lo estimado.

##### ▼ Concepto

Estimar es predecir el trabajo, esfuerzo que será necesario realizar en un momento donde no se tiene conocimiento sobre lo que se estima.

Por definición no es precisa, acarrea riesgo es incertidumbre propio del inicio del proyecto.

##### ▼ Errores

Los errores en las estimaciones provienen de:

- caos del proyecto
- la incertidumbre
- bajo nivel de conocimiento de la capacidad de la empresa
- falta de claridad en el alcance
- no utilización de técnicas adecuadas

En la gestión tradicional, se plantea una secuencia de estimaciones que tiene que tener el siguiente orden:

1. **Tamaño:** Definir el producto a construir
2. **Esfuerzo:** Horas persona lineales, una bolsa de hora ideales (no se tienen en cuenta las horas de ocio).
3. **Calendario:** Determinar qué días y que horas trabajar, y cuántas personas van a trabajar.
4. **Costo**
5. **Recursos Críticos:** A esas cosas que nos hacen falta para desarrollar, ejemplo, si desarrollamos un software que maneja sensores de alarma para incendio, el desarrollador necesita el sensor y el de testing también. No es propio de todos los proyectos pero es necesario tener en cuenta la posibilidad de su existencia.

Primero se determina que es lo que se debe construir (**tamaño**) y su alcance. Es difícil medir el tamaño del software, pero una de las posibilidades es, por ejemplo, la cantidad de casos de uso. Luego, se estima el **esfuerzo**, es decir, cuantas horas lineales necesito para construir lo que definí en el tamaño; todavía no pensamos quienes lo van a hacer ni qué disponibilidad de recursos tengo, simplemente es una medida en cantidad de horas de esfuerzo.

El siguiente paso es llevarlo a **calendario**, a esas horas lineales las distribuyo en las tareas que hay que realizar que tienen que ver con estas horas y en las secuencias que tienen estas tareas entre sí.

Luego se estima el costo, que esta directamente vinculado con la mano de obra; en función de la cantidad de horas y del calendario voy a poder determinar el costo de lo que tiene que ver con la mano de obra, pero también el costo de otras variables, como el uso de herramientas, disponibilidad de almacenamiento, disponibilidad de las distintas instalaciones que puedo llegar a necesitar.

Por último, estimo los recursos críticos, cuales son aquellos momentos del tiempo y recursos que son críticos y que me pueden generar algún inconveniente en el caso de no contar con ese recurso.

#### ▼ Técnicas de estimación

- Contar funcionalidades / features / requerimientos / historias de usuario
- Basadas en la experiencia
  - Datos históricos sobre procesos anteriores
  - Juicio experto: una persona experta estima → quien realiza la tarea. Puede incluir técnica  $(o+4h+p)/6$
  - WIDEBAND Delphi (Variante del juicio experto) → Conjunto de personas informadas hacen una estimación y discuten hasta llegar a un acuerdo.
    - Poker planning
- Basadas en los recursos
- Basados en el mercado
- Basados en los componentes del producto o proceso de desarrollo
- Métodos algorítmicos

#### ▼ Gestión de riesgo

Cuando se piensa en el riesgo, abordamos un problema o evento que podría llegar a suceder y comprometer el éxito del proyecto.

Todos los riesgos tienen asociados una probabilidad de ocurrencia y un impacto (en tiempo o dinero). Asociando una escala numérica para ambas variables y multiplicando obtenemos la **exposición al riesgo**

$$\text{Impacto} \times \text{Prob. de Ocurrencia} = \text{Exposición al riesgo}$$

Priorizando los riesgos según su exposición, deben tomarse una lista del top con mayor valor y gestionarlos para:

- Mitigar: evitar que ese riesgo suceda
- Elaborar planes de contingencia: ocurrido, acciones que se tomarán para disminuir su impacto

#### ▼ Métricas de software

Las métricas se definen fundamentalmente para que nosotros podemos medir a través de valores concretos, saber como vamos con nuestro proyecto y para poder hacer esto lo definimos en términos de tres dominios de las métricas.

- Métricas de proceso: usamos las mismas métricas del proyecto pero despersonalizadas del proyecto, es decir, lo hacemos de una manera más estratégica, con el foco en saber si hay algunas cosas que nuestro proceso de software tiene que cambiar porque está mal definido.

- Métricas de proyecto: me permiten ver cómo voy con el desarrollo de mi proyecto, como, por ejemplo, métricas de esfuerzo y métricas de tiempo (Calendario)
- Métricas de producto: miden cuestiones/características que tienen que ver con el producto de software que yo estoy creando, como, por ejemplo, métricas de defectos y métricas de tamaño.

Las métricas del proyecto se consolidan para crear métricas de proceso que sean públicas para toda la organización del software.

**¿Cada cuánto se toman las métricas?** Depende de las características y tiempos del proyecto. Deben ser lo suficientemente frecuentes para detectar desvíos a tiempo.

**¿Qué se debe hacer con las métricas?** Se desea saber si el proyecto va a poder cumplir con los objetivos propuestos. Si las métricas obtenidas indican que no, debo tomar acciones para corregirlo e intentar cumplirlo

Los proyectos se atrasan de un día a la vez, si puedo corregir los desvíos de mi proyecto en el momento adecuado, estoy a tiempo de poder cumplir mi objetivo.

#### ▼ Monitoreo y control

El **monitoreo y control**, que es para lo que usamos las métricas, tiene que ver con ir comparando lo planificado y lo real; la línea perfecta entre lo planificado y lo real no existe, suele estar algo dibujado. De esto se encarga el Líder de proyecto.

#### ▼ Éxito del proyecto

##### Factores de éxito de un proyecto

- Monitoreo y Feedback
- Tener una misión y objetivo claro
- Comunicación en todos sus aspectos

##### Causas de fracasos comunes:

- Fallar al definir el problema
- Planificar basado en datos insuficientes
- La planificación la hizo el grupo de planificaciones
- No hay seguimiento del plan del proyecto
- Plan de proyecto pobre en detalles
- Planificación de recursos inadecuada
- Las estimaciones se basaron en “supuestos” sin consultar datos históricos
- Nadie estaba a cargo

## Unidad 2

### Filosofía ágil

## ▼ Origen

El agilismo es un movimiento que se gestó desde los desarrolladores.

Se juntaron hace aproximadamente 20 años un grupo de referentes de la industria de software en un hotel de Utah a discutir y generaron un acuerdo al que llamaron **Manifiesto ágil**.

Este manifiesto ágil es un compromiso de todas las personas involucradas para trabajar de una determinada manera independientemente de las prácticas que realice cada uno. Esto ha sido una evolución cultural que tiene que ver con las experiencias que cada uno de los referentes ha vivido.

El movimiento ágil tenía como objetivo lograr un equilibrio entre hacer las cosas como si no fuéramos profesionales, y tampoco hacer las cosas en el otro extremo llegando al punto de cumplir con el proceso en lugar de entregar un producto de calidad.

| *Es un compromiso útil entre nada de proceso y demasiado proceso.*

El manifiesto ágil se sustenta en los procesos empíricos que tienen como base la experiencia, y la misma sale del propio equipo, por ello es importante tener ciclos de realimentación cortos. Empezamos con algo, lo construimos y los mostramos para obtener la retroalimentación para corregir cosas si es necesario y mejorar.

“Ágil” es una ideología con un conjunto definido de principios que guían el desarrollo del producto. Busca un balance entre ningún proceso y demasiado proceso.

## ▼ Procesos empíricos

Los procesos empíricos son solo compatibles con ciclos de vida iterativos.

El empirismo tiene 3 pilares:

- **Transparencia:** Es el que nos permite a nosotros crecer como equipo y transformar el conocimiento implícito, el conocimiento de cada uno de los miembros del equipo, en conocimiento explícito, conocimiento que sea del equipo. Debemos acostumbrarnos a la transparencia, a lo que hacemos no es propio, sino el producto que el equipo está construyendo. Si tengo un problema, informarlo, pedir ayuda. Todo esto ayuda para que estos procesos empíricos puedan fluir.
- **Adaptación**
- **Inspección**

## ▼ Manifiesto ágil

### ▼ Valores

**Individuos e interacciones por sobre procesos y herramientas:** Las dos cosas son importantes sólo que los individuos e interacciones es más valorado. Esto quiere decir que es más importante los vínculos y la comunicación entre los miembros del equipo por sobre aferrarse a la herramienta que tenemos que usar y el proceso a aplicar.

**Software funcionando por sobre documentación extensiva:** Este valor es del que se agarra mucha gente para no generar documentación, (Ejemplo, hacer SCRUM porque el mismo dice que no hay que documentar), esto es erróneo porque existe la necesidad de mantener la información del producto y sobre el proyecto que estamos cursando para obtener este producto. El enfoque ágil plantea generar la información cuando haga falta. Debemos recordar que, “lo más importante es el acto de planificar no el resultado” no es que no vamos a definir la arquitectura, esto es muy importante, y las decisiones de arquitectura que el equipo tomó deben quedar

documentadas, el equipo se pondrá de acuerdo en cómo va a documentar. Las decisiones tomadas con respecto al producto de software deben quedar documentadas y van a trascender más allá de la iteración que generó ese producto de software. Hay que decidir qué aspectos del producto van a quedar documentados, el conocimiento del producto debe ser transparente y de toda la organización no de un individuo. Con respecto al proyecto lo mismo, veamos que necesitamos documentar del proyecto, generamos poca información, pero se genera. Otro enfoque a tener en cuenta es la permanencia, no toda la información que se genera es necesaria que este siempre.

**Colaboración con el cliente por sobre negociación contractual:** Este enfoque está muy relacionado con la triple restricción. Los problemas de las negociaciones contractuales comienzan cuando el cliente tiene previamente un acuerdo formal/contrato firmado con el equipo con un determinado alcance/costo y tiempo, y luego el cliente quiere cambiar algunos aspectos que firmó, generando así conflictos, que pueden ser evitados si se involucra más al cliente dentro del proyecto, y no se desentiende. El cliente debe tener una actitud de integrarse en el proyecto. El punto clave para determinar si vamos a poder a hacer ágil o no, es si el cliente está dispuesto a sumarse a este enfoque o no (Si el mismo no participa, no contesta, se desentiende, no podemos aplicar la metodología). No sólo hay que formar a los equipos que van a trabajar con ágil, sino también al cliente (o Product Owner en Scrum).

**Responder al cambio por sobre seguir un plan:** Construir en conjunto con el cliente, no definamos tanto, empecemos a trabajar con una idea del producto y después vamos más a profundo para darle la posibilidad al cliente de cambiar de opinión. En lugar de tener una ERS firmada por el cliente que seguro no leyó, mejor tener una actitud más ajustada con la realidad de que los requerimientos cambian, la gente se equivoca, se olvida, se da cuenta de lo que quiere cuando lo ve, y no por un contrato.

#### ▼ Principios

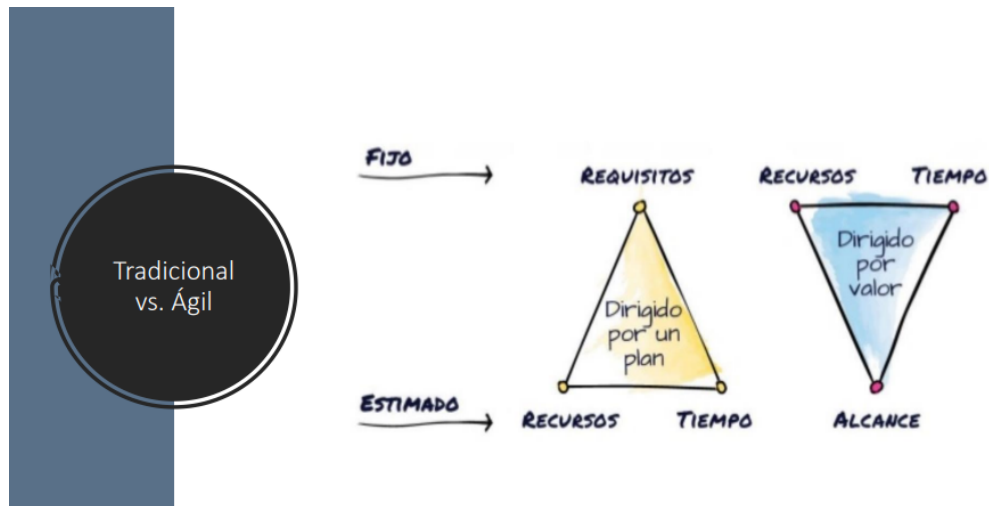
12 principios ágiles:

1. Nuestra mayor prioridad es satisfacer al cliente
2. Aceptar que los requisitos cambien
3. Entregar software funcional frecuentemente
4. Los responsables de negocios, diseñadores y desarrolladores (técnicos y no técnicos) deben trabajar juntos día a día durante el proyecto
5. Desarrollamos proyectos en torno a individuos motivados
6. El método más eficiente de comunicar información es conversaciones cara a cara
7. El software funcionando es la principal medida de éxito
8. Los procesos ágiles promueven el desarrollo sostenible → El equipo debe trabajar a un ritmo sostenible que le permita entregar software funcionando en cada iteración.
9. La atención continua a la excelencia técnica y al buen diseño mejoran la Agilidad
10. La simplicidad es esencial → La simplicidad se define como la maximización del trabajo no hecho. Lo mejor es lo simple, no agregar funcionalidades porque sí, si el cliente no lo pidió
11. Las mejores arquitecturas, requisitos y diseños emergen de equipos auto-organizados → Esto apunta a contrarrestar del enfoque tradicional que las decisiones de requerimientos, arquitectónicas, y de diseño que más impacto tienen en el producto, son tomadas por

consultores que no son parte del proyecto. El agilismo dice que la mejor gente para definir las características del producto, luego de estimarlas es la que va a hacer el trabajo.

12. A intervalos regulares, el equipo debe reflexionar sobre cómo ser más efectivo y de acuerdo a esto, ajustar su comportamiento.

#### ▼ Triple restricción ágil



Desde el punto de vista de los requerimientos, con el enfoque ágil, debemos en lugar de dejar fijo los requerimientos o el alcance, y a partir de ahí derivar recursos y tiempo (como lo plantea el enfoque tradicional), vamos a dejar fijo el tiempo (con iteraciones de duración fija, las que SCRUM llama Sprint) y los recursos (tener un equipo de trabajo con una determinada capacidad) asignado a trabajar, en base a esto, defino cuanto del producto funcionando puedo construir en este tiempo y así se acuerda el alcance, luego se repite para la siguiente iteración.

Teniendo en cuenta esta forma de pensar, se puede hablar de **gestión ágil de los requerimientos**

## Requerimientos Ágiles

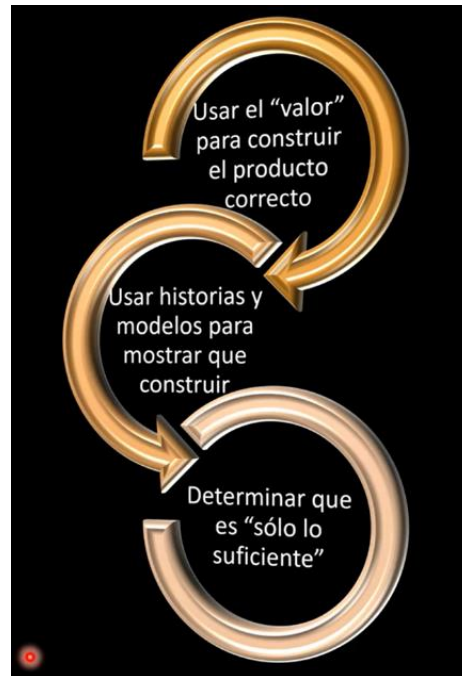
#### ▼ Concepto

Así como el agilísimo realiza una propuesta de trabajo con respecto al desarrollo propiamente dicho del software, del mismo modo lo hace con los requerimientos.

En este caso, se asume que:

- Los cambios van a existir todo el tiempo de forma constante.
- Tenemos que tener una mirada de los que están involucrados, es decir, todos los que tienen algo para decir del producto (StakeHolders). El product owner es el representante de todos los stakeholders.
- Hay una realidad de que el cliente dice si está satisfecho o no, cuando tenga algo para ver. Es por ello que estamos apurados para mostrarle algo al cliente.
- Los requerimientos se van a ir conociendo y/o "encontrando" de a poco. Se debe contar con lo mínimo y necesario para comenzar y luego generar retroalimentación con el software funcionando.
- No todas las herramientas sirven para todos los casos. Para cada situación hay una determinada herramienta más adecuada.

- Lo importante no es entregar una salida, un requerimiento, lo importante es entregar, un resultado, una solución de “valor”, construyendo el producto correcto.
- En la gestión tradicional de los requerimientos, muchas funcionalidades se construían y nunca eran utilizadas por los clientes.



La gestión ágil de requerimientos intenta contrarrestar este último punto incorporando la imagen de Product Owner. Esta persona es quien tiene realmente claras cuales son sus necesidades y su responsabilidad principal es priorizar requerimientos (Decidir qué es lo que necesita primero, que da un valor al negocio más importante).

#### ▼ Tiempos - Just in Time

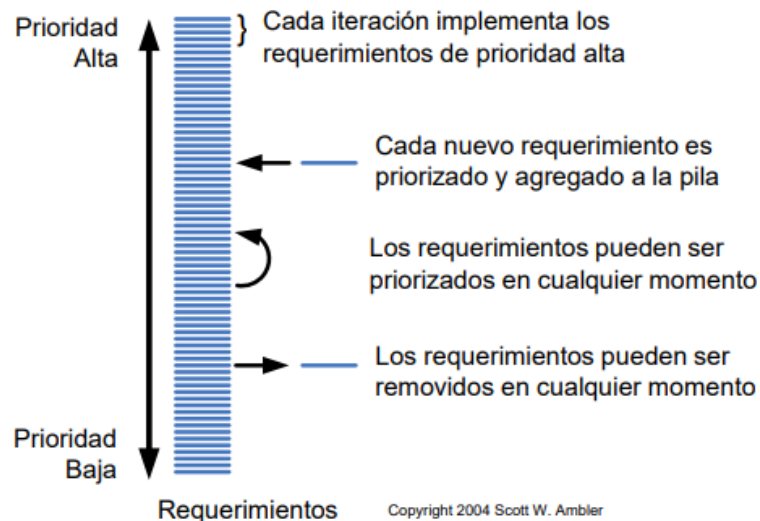
Just in time (Ni antes ni después, si llega antes es un desperdicio, pero si llega después tampoco sirve porque ya alguien tomó la decisión), es un concepto que se usa fundamentalmente para evitar desperdicios (especificar requerimientos que después cambian e invertí un montón de tiempo en especificarlos). El producto no va a estar especificado 100% desde el principio, se irán encontrando requerimientos y describiéndolos conforme haga falta.

#### ▼ Tipos de requerimientos

- Requerimientos de negocio
- Requerimientos de usuario
- Requerimientos funcionales
- Requerimientos no funcionales
- Requerimientos de implementación

#### ▼ Gestión ágil de requerimientos





En la gestión ágil, los requerimientos se encuentran contenidos en un contenedor denominado Product Backlog que es una lista priorizada y organizada por el Product Owner según su valor de negocio.

El valor se relaciona con la utilidad, beneficio o satisfacción que le ofrecemos a los usuarios finales por cada funcionalidad completa que se entrega.

Para la planificación del equipo, se tratará al product backlog como una pila donde siempre se extrae lo más prioritario (lo de la cabeza)

#### ▼ Principios ágiles

Los principios ágiles relacionados a la gestión ágil de los requerimientos son:

- La prioridad es satisfacer al cliente (entregando software funcionando) a través de releases tempranos y frecuentes
- Toda la gestión ágil se arma para estar preparados a recibir cambios de requerimientos aún en etapas finales.
- Técnicos y no técnicos (Product Owner) trabajando juntos todo el proyecto.
- El medio de comunicación por excelencia es cara a cara.
- Las mejores arquitecturas, diseños y requerimientos emergen de equipos auto-organizados. El enfoque ágil dice de darle responsabilidad al equipo en cuanto a decisiones técnica porque es un equipo capacitado.

#### ▼ Historias de usuario

Es forma de trabajar con requerimientos ágiles, es una definición de una funcionalidad que el software debe cumplir, expresada en forma de historia y desde la perspectiva del usuario.

Una historia de usuario es una descripción corta de una necesidad que tiene el usuario respecto del producto de software.

Representa en cierta forma, la necesidad del usuario, una descripción del producto, un ítem de planificación (al priorizar en historias, cada una representa un ítem de planificación) y es un token para una conversación (es decir que nos dice sobre que tenemos que hablar)

Una historia de usuario representa:

- Una necesidad del usuario
- Una descripción del producto
- Un ítem de planificación para determinar que va a entrar en una próxima iteración
- Token para una conversación, recordatorio para hablar con el cliente de una determinada funcionalidad.
- Mecanismo para diferir una conversación

▼ Características de las Historias de Usuario

- Las USER STORIES capturan las necesidades e ideas de los usuarios pero no llegan a ser especificaciones detalladas de los requerimientos (como los cu o la ERS).
- Son porciones verticales: Las US se cortan verticalmente incluyendo todas las capas a nivel arquitectónico (Interfaz de usuario, lógica de negocio, base de datos), para entregarle al usuario algo de software funcionando que le sirva y pueda trabajar con eso. Si las corto horizontalmente no entrego valor al cliente.
- Son expresiones de intención, ("es necesario que haga algo no esto...")
- No están detallados al principio del proyecto, elaborados evitando especificaciones anticipadas, demoras en el desarrollo, inventario de requerimientos y una definición limitada de la solución.
- Necesita un poco o nulo mantenimiento y puede descartarse después de la implementación
- Junto con el código, sirven de entrada a la documentación que se desarrolla incrementalmente después

▼ Partes de una User Story

▼ **1-Tarjeta**

Es la parte visible, donde escribimos la historia de usuario.

Siguen el siguiente formato.

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

- Nombre del rol: representa quien está realizando la acción o quien recibe el valor de la actividad
- Actividad: representa la acción que realizará el sistema
- Valor de negocio que recibo: comunica porque es necesaria la actividad. El valor de negocio le sirve al Product Owner priorizar lo que quiere.

Criterios de aceptación

- Definen limites para la user y son la base para definir las pruebas de aceptación
- Muestra lo necesario para que la user provea valor según lo que establece el Product Owner
- Genera una vision compartida con el equipo

- Guía a los equipos en las pruebas (desarrolladores y testers)
- Ayuda a determinar cuando parar de agregar funcionalidades

Pruebas de aceptación: Son las pruebas que se entiende va a hacer el Product Owner y los usuarios, cuando se entregue la funcionalidad lista. Están relacionadas con los criterios de aceptación. Se describe lo que se tiene que probar, estos no son casos de prueba, son los títulos. Expresan detalles resultantes de la conversación. Probar que lo que quiero que haga funcione y como. Que cosas deberían poder hacerse y que cosas no se aceptan porque contradicen los criterios de aceptación. Complementan la US.

Frase verbal: suele agregarse una frase resumen de lo que trata la US, a modo de "título"

## ▼ 2-Conversación

La conversación se considera la parte más importante de las User Stories. Esta no queda guardada en ningún lado, porque según los principios ágiles el mejor medio de comunicación es la comunicación cara a cara.

## ▼ 3-Confirmación

Las pruebas de usuario que se identifican necesarias para hacerle después a la funcionalidad una vez realizada, para que el product owner me acepta la característica de software construida a partir de la US.

## ▼ Definition of Ready

Una medida de calidad que construye el equipo para poder determinar que la user está en condiciones de entrar a una iteración de desarrollo. La User está lista cuando cumple con la definición de listo (o Ready) impuesta por el equipo.

Para determinar si una user esta lista nos guiamos en el **INVEST MODEL**:

- **Independiente:** No depende de ninguna otra user
- **Negociable:** la user tiene que estar escrita en términos de que necesita el usuario no como lo vamos a implementar.
- **Valuable:** debe tener un valor para el cliente
- **Estimable:** debo poder definir cuanto esfuerzo me conlleva hacer esa user o que tan grande
- **Small (Pequeña):** Debe poder ser consumida en una iteracion. Tiene que ver con no hacer el trabajo a medias, que algo sea pequeño depende mucho del equipo
- **Testeable:** se debe poder demostrar que la user se implementó cumpliendo los criterios de aceptación definidos

## ▼ Definition of done

Nos define si la historia esta decentemente terminada/presentada para poder presentársela al PO

## ▼ Spikes

Son un tipo especial de US que se producen por la incertidumbre que la misma presenta, la cual imposibilita que pueda ser estimada y por lo tanto no cumple con la definición de listo, Una vez

resuelta la incertidumbre, la Spike se convierte en una o más US. Es una característica más del producto a la cual el equipo le tiene que dedicar tiempo para:

1. Familiarizarse con una nueva tecnología o dominio.
2. Investigar y prototipar para ganar confianza frente a:
  - a. Riesgos tecnológicos.
  - b. Riesgos funcionales, donde no está claro cómo debe reaccionar el sistema para satisfacer las necesidades del usuario.

Se usan para quitar el riesgo e incertidumbre de una US y otra faceta del proyecto. Se clasifican en técnicas (asociada a la implementación y tecnología) y funcionales (utilizada cuando hay incertidumbre respecto de como el usuario interactúa con el sistema).

Deben ser:

- Estimables, demostrables y aceptables
- Excepcionales: Los spikes deben dejarse para incógnitas más críticas y grandes. Se deben utilizar spikes como última opción.
- Implementar la spike en una iteración separada de las historias resultantes. (Salvo sea muy pequeña)

#### ▼ Estimaciones ágiles

Las estimaciones ágiles, a diferencia de las estimaciones absolutas de la gestión tradicional, tienden a incorporar un método de estimación relativa basado en obtener la estimación a partir de la comparación

- Las personas no saben estimar en términos absolutos
- Somos buenos comparando cosas.
- Comparar es generalmente más rápido, es más fácil saber si algo es grande comparándolo con algo mas chico.
- Se obtiene una mejor dinámica grupal y pensamiento de equipo más que individual
- Se emplea mejor el tiempo de análisis de las US.

Se puede trabajar con Historias de Usuario y Story Points



Story Points son una medida de tamaño relativo asignable a las historias de usuario que combinan la incertidumbre, esfuerzo y complejidad

#### ▼ Definición de Listo - Definition of Ready

INVEST MODEL

## Scrum

#### ▼ Concepto

Scrum es un marco ligero que ayuda a las personas, equipos y organizaciones a generar valor a través de soluciones adaptables para problemas complejos.

En pocas palabras, Scrum requiere un Scrum Master para fomentar un entorno donde:

1. Un propietario del producto (Product Owner) ordena el trabajo de un problema complejo en un Product Backlog.
2. El equipo de Scrum convierte una selección del trabajo en un Incremento de valor durante un Sprint.
3. El equipo de Scrum y sus partes interesadas (stakeholders) inspeccionan los resultados y realizan los ajustes necesarios para el próximo Sprint.
4. Repetir

El marco de Scrum es deliberadamente incompleto, solo define las partes necesarias para implementar la teoría de Scrum. Scrum se basa en la inteligencia colectiva de las personas que lo utilizan. En lugar de proporcionar a las personas instrucciones detalladas, las reglas de Scrum guían sus relaciones e interacciones.

#### ▼ Enfoque

Scrum emplea un enfoque iterativo e incremental para optimizar la previsibilidad y controlar el riesgo.

Scrum involucra a grupos de personas que colectivamente tienen todas las habilidades y experiencia para hacer el trabajo y compartir o adquirir tales habilidades según sea necesario.

#### ▼ Pilares empíricos (T.I.A)

**Transparencia:** el proceso y trabajo son visibles para los que realizan el trabajo. La visibilidad sobre los artefactos es fundamental, la falta de transparencia de ellos conduce a decisiones que disminuyen el valor y aumentan el riesgo.

**Inspección:** Los artefactos de Scrum y el progreso hacia objetivos acordados deben ser inspeccionados con frecuencia para detectar varianzas o problemas potencialmente indeseables.

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

**Adaptación:** Si algún aspecto de un proceso se desvía fuera de los límites aceptables o si el producto resultante es inaceptable, el proceso que se está aplicando o los materiales que se producen deben ajustarse. El ajuste debe realizarse lo antes posible para minimizar la desviación adicional.

Se espera que un equipo de Scrum se adapte en el momento en que aprenda algo nuevo por medio de la inspección

#### ▼ Valores de Scrum (CERCA)

- Compromiso
- Enfoque
- Respeto
- Coraje
- Apertura

#### ▼ Scrum Team

El Scrum Team es la unidad fundamental de Scrum, un equipo pequeño de personas. Consta de:

- **Scrum Master:** es responsable de establecer Scrum tal como se define en la Guía de Scrum. Lo consigue ayudando a todos a comprender la teoría y la práctica de Scrum, tanto dentro del Equipo como en toda la organización. Asegura los eventos de Scrum, hace respetar sus timeboxes, ayuda al equipo a eliminar los impedimentos.
- **Product Owner:** El Propietario del Producto (una sola persona) es responsable de maximizar el valor del producto resultante del trabajo del equipo de Scrum. También es responsable de la gestión eficaz de la pila del producto
- **Desarrolladores:** personas del equipo Scrum que se comprometen a crear cualquier aspecto de un Incremento útil (funcional) en cada Sprint.

No hay una distinción de jerarquías ni sub-equipos

El equipo de Scrum es lo suficientemente pequeño como para permanecer ágil y lo suficientemente grande como para completar un trabajo significativo dentro de un Sprint, por lo general 10 o menos personas. → Equipos pequeños se comunican mejor y son más productivos

#### ▼ Eventos

Scrum combina cuatro eventos formales para la inspección y adaptación dentro de un evento contenedor, el Sprint.

Los sprints son el latido del corazón de Scrum, donde las ideas se convierten en valor


Los sprints tienen duración fija de un mes o menos.

Durante el Sprint:

- No se hacen cambios que pongan en peligro el Objetivo Sprint
- La calidad no disminuye;
- El trabajo pendiente del producto se refina según sea necesario;
- El alcance se puede clarificar y renegociar con el Propietario del Producto a medida que se aprende más

Un Sprint podría ser cancelado si el Objetivo del Sprint se vuelve obsoleto. Solo el Propietario del Producto tiene la autoridad para cancelar el Sprint

#### ▼ Sprint Planning

 **Momento:** Inicio del Sprint

 **Duración:** 8hs máxima para sprints de 1 mes


Por qué este sprint es valioso? → Objetivo del sprint

Que se puede hacer este sprint? → Selección de items del product backlog

Cómo se realizará el trabajo elegido? → Planificación del trabajo para cada item seleccionado.  
División en elementos de trabajo que puedan ser realizados en un día o menos.

#### ▼ Daily Scrum

 **Momento:** Todos los días laborables


 **Duración:** 15 minutos máximo

Tiene como objetivo inspeccionar el progreso hacia el Objetivo Sprint y adaptar el Sprint Backlog según sea necesario, ajustando el próximo trabajo planeado.

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

#### ▼ Sprint Review


 **Momento:** Previo al final del Sprint


 **Duración:** 4hs máximo para sprints de 1 mes

Es una sesión de trabajo y el equipo de Scrum debe evitar limitarla a que se convierta en una simple presentación.

Tiene como propósito inspeccionar el resultado del Sprint y determinar futuras adaptaciones. Se presenta el resultado del trabajo a las partes interesadas y se discute el progreso hacia el objetivo del producto.

#### ▼ Sprint Retrospective

 **Momento:** Final del Sprint

 **Duración:** 3hs máxima para sprints de 1 mes

El propósito de la retrospectiva Sprint es planificar formas de aumentar la calidad y la eficacia.

El equipo de Scrum inspecciona cómo fue el último Sprint con respecto a individuos, interacciones, procesos, herramientas y su definición de Hecho. Los elementos inspeccionados a menudo varían según el dominio del trabajo. Las suposiciones que los desviaron se identifican y se exploran sus orígenes. El equipo de Scrum analiza qué fue bien durante el Sprint, qué problemas encontró y cómo esos problemas fueron (o no fueron) resueltos.

El equipo de Scrum identifica los cambios más útiles para mejorar su eficacia. Las mejoras más impactantes se abordan lo antes posible. Incluso se pueden agregar al Sprint Backlog para el próximo Sprint.

#### ▼ Artefactos

Los artefactos de Scrum representan trabajo o valor. Cada artefacto contiene un compromiso para garantizar que proporciona información que mejora la transparencia y el enfoque con el que se puede medir el progreso

##### ▼ Product Backlog

El trabajo pendiente del producto es una lista emergente y ordenada de lo que se necesita para mejorar el producto. Es la única fuente de trabajo emprendida por el equipo Scrum.

El refinamiento de Backlog del producto es el acto de descomponer y definir aún más los elementos de trabajo pendiente del producto en artículos más pequeños y precisos.

**Compromiso:** Objetivo del producto

##### ▼ Sprint Backlog

El Trabajo pendiente de Sprint se compone del objetivo sprint (por qué), el conjunto de elementos de trabajo pendiente de producto seleccionados para el Sprint (qué), así como un plan accionable para entregar el incremento (cómo).

**Compromiso:** Sprint Goal

### ▼ Incremento

Cada Incremento es aditivo a todos los Incrementos anteriores y verificado a fondo, asegurando que todos los Incrementos funcionen juntos. Para proporcionar el valor, el incremento debe ser utilizable.

Se pueden crear varios incrementos dentro de un Sprint. La suma de los Incrementos se presenta en la Revisión Sprint apoyando así el empirismo. Sin embargo, un Incremento puede ser entregado a las partes interesadas antes del final del Sprint. La revisión de Sprint nunca debe considerarse una puerta para liberar valor.

**Compromiso:** Definition of Done → es una descripción formal del estado del Incremento cuando cumple con las medidas de calidad requeridas para el producto.

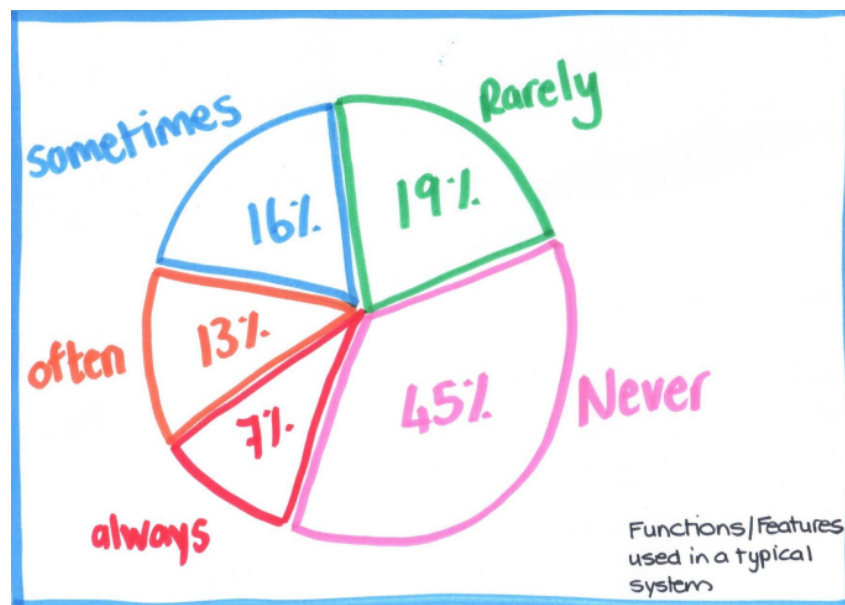
## Gestión de producto

### ▼ Introducción

¿Por qué creamos productos? Cual es nuestra motivación?

- Para satisfacer a los clientes
- Para tener muchos usuarios logueados
- Para obtener mucho dinero
- Para realizar una gran visión, cambiar el mundo

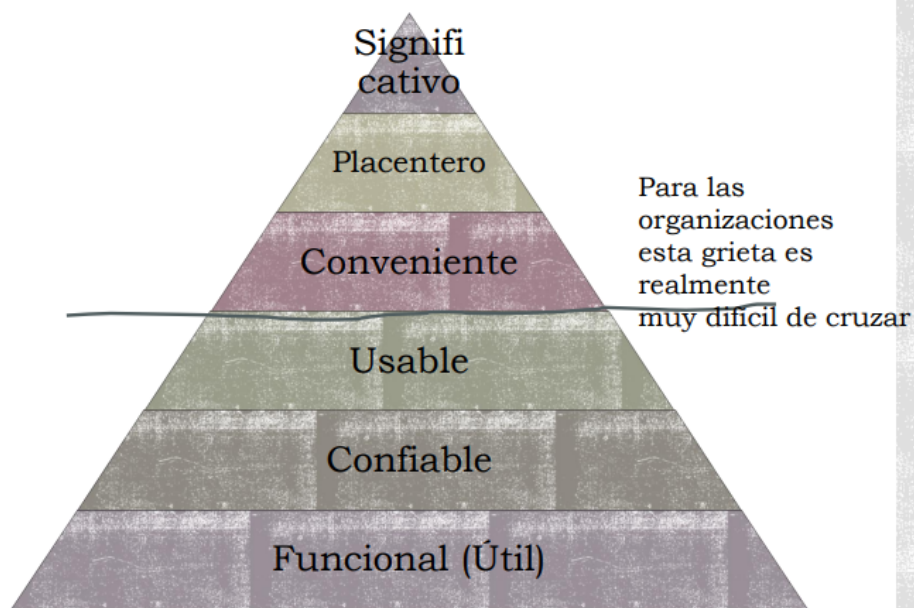
¿Qué características realmente utilizamos del producto de software? Tener features/funciones que no utilizamos es gastar plata innecesariamente. Hay que eliminar el desperdicio.



### ▼ Evolución de los productos de software



**Focalizado en experiencias (gente, actividades, contexto)**



**Focalizado en tareas (productos y características)**

Para controlar el esfuerzo que realizamos, podemos aplicar la técnica UVP.

Se define idea o hipótesis, se comienza a materializar la idea del producto priorizando las features.

▼ Conceptos:

▼ MVP

Se crea un Producto Mínimo Viable (MVP) para poder probar la hipótesis. Es solo para asegurarse de que sea viable como producto, ver si sirve o no y de ahí ver como seguir.

MVP también es una hipótesis. Sirve para ver si los clientes están interesados en el producto que tengo para ofrecer.

- Podría ser lo suficientemente bueno para encontrar un mercado o no.
- Puede pasar que en lugar de validar la hipótesis, haya una característica del producto que más le interesa al cliente. Ahí es cuando tengo que cambiar el foco, y puede convertirse en MVP2. Esto puede ocurrir repetidas veces.
- Tiene el valor suficiente para que las personas estén dispuestas a usarlo o comprarlo inicialmente
- Demuestra suficiente beneficio futuro para retener a los primeros usuarios
- Proporciona un ciclo de retroalimentación para guiar el desarrollo futuro.

▼ MMF

Es una característica mínima comerciable, se descompone el producto y vemos qué features se pueden comercializar. Objetivo: aportar valor.

En el caso de áreas con una fuerte indicación de valor, puede directamente definir un MMF (Características mínimas comercializables. Para encontrar la pieza mínima que pueda empezar a traer crecimiento.

La razón para dividir una característica grande en MMF más pequeños es principalmente el tiempo de comercialización (Time to market) y la capacidad de aportar valor en muchas áreas.

#### ▼ MVF

característica Mínima Viable. No me centro en el producto viable. Si la MVF es exitosa puedo realizar más MMF en esa área para tomar ventaja. Sino, se puede cambiar a otro enfoque. El MVF es una versión mini del MVP. (el MVP esta compuesto de MVF)

Un MVF debe proporcionar un valor claro a los usuarios y ser fácil de usar.

El producto se cultiva en mercados inciertos al intentar varios MVP.

Cuando se logra ajustar el producto en el mercado de productos se combinan MMF y MVF según el nivel de incertidumbre del negocio / requisitos en las áreas en las que se está enfocando.

Si bien los MVP / MMF / MVF son atómicos desde una perspectiva empresarial (no puede implementar y aprender de algo más pequeño) pueden ser bastante grandes desde la perspectiva de la implementación.

#### ▼ MMP

Primer release de un MMR dirigido a primeros usuarios (early adopters),

Focalizado en características clave que satisfarán a este grupo clave.

Ya se puede vender el producto, y al ser el primer release tiene que captar una buena impresión

#### ▼ MMR

Release de un producto que tiene el conjunto de características más pequeño posible.

El incremento más pequeño que ofrece un valor nuevo a los usuarios y satisface sus necesidades actuales.

MMP = MMR1 (pueden salir mas releases después, que mejoren el producto/release inicial)

lo obtengo en varias iteraciones hasta q tengo una cierta cantidad de características a lanzar

## Unidad 3

### Gestión de configuración

#### ▼ Concepto

Los cambios en el Software se dan por cambios del negocio y nuevos requerimientos, por la reorganización de las prioridades de la empresa por crecimientos, cambios en el presupuesto y defectos que se quieren corregir u oportunidades de mejora.



SCM es una disciplina que aplica dirección y monitoreo administrativo y técnico a: identificar y documentar las características funcionales y técnicas de los ítems de configuración, controlar los cambios de esas características, registrar y reportar los cambios y su estado de implementación y verificar correspondencia con los requerimientos

Software Configuration Management - SCM (o Gestión de la configuración de software) es una disciplina de soporte cuyo propósito es mantener la integridad del producto a lo largo de todo el ciclo de vida. Tiene aplicación en diferentes disciplinas:

- Control de calidad de proceso

- Control de calidad de producto
- Prueba de software

Así, es responsable de la integridad pero ¿Que es la integridad?

- Satisfacer las necesidades del cliente
- Poseer cambios fácil y completamente rastreables durante su ciclo de vida
- Buena performance
- Cumplir con expectativas de costo

#### ▼ Definiciones clave

ÍTEM DE CONFIGURACIÓN (IC): cada artefacto que forma parte del producto o del proyecto que puede sufrir cambios o necesitan ser compartidos entre miembros del equipo. Necesitamos conocer su estado y evolución.

VERSIÓN de un IC: forma particular de un artefacto en un instante o contexto dado. Se refiere a la evolución de cada IC por separado

VARIANTE: es una versión de un IC que evoluciona por separado. Representan configuraciones alternativas, ya que un producto puede adoptar distintas formas dependiendo su contexto

REPOSITORIO: contenedor de ICs, mantiene la historia de cada IC con sus atributos y relaciones. Usado para hacer evaluaciones de impacto de los cambios propuestos. Pueden ser 1 o + bases de datos

- Centralizados
- Descentralizados

LÍNEA BASE: es una configuración de software especial, que ha sido revisada formalmente y sobre la cual se llegó a un acuerdo. Se “marca” con una etiqueta. Sirve como base para desarrollos posteriores y puede cambiarse solo a través de un procedimiento formal de control de cambios. Permiten ir atrás en el tiempo y reproducir el entorno de desarrollo en un momento dado del proyecto. Pueden ser:

- De especificación (Requerimientos, Diseño)
- De productos que han pasado por un control de calidad definido previamente

RAMAS: sirven para bifurcar el desarrollo. Permiten experimentar con el proyecto diferentes formas y después se elige la alternativa que se adapte mejor y se unifica al proyecto principal (rama principal/trunk/master), el resto de las ramas se descartan.

#### ▼ Actividades

##### ▼ Identificación de Items de configuración

- Identificación unívoca de los items de configuración
- Definición de convenciones y reglas de nombrado
- Definición de la estructura del repositorio
- Ubicación dentro de la estructura

Los IC pueden ser de diferentes tipos (Prod/ Proy/ Iter):



#### ▼ Control de cambios

Puede haber necesidad de cambio para uno o varios IC que se encuentran en una línea base. Es un procedimiento formal que involucra diferentes personas y una evaluación del impacto del cambio

**Comité de control de cambios:** formado por representantes de las áreas interesadas

#### ▼ Auditoría de gestión de configuración

##### Auditoría física de la configuración (PCA)

Asegura que lo que está indicado para cada IC en la línea base o en la actualización se ha alcanzado realmente.

Consistencia entre lo que definimos en el plan de configuración para cada IC, con lo que tengo en el repositorio

Verificación (física PCA): asegura que un producto cumple con los objetivos preestablecidos, definidos en la documentación de líneas base (línea base). Todas las funciones son llevadas a cabo con éxito y los test cases tengan status “ok” o bien consten como “problemas reportados” en la nota de release.

##### Auditoría funcional de configuración (FCA)

Evaluación independiente de los productos de software, controlando que la funcionalidad y performance reales de cada Item de configuración sean consistentes con la especificación de requerimientos.

Valida que el producto a construir sea consistente con los Requerimientos identificados. Se usa una matriz de rastreabilidad para encontrar donde se implementan cada requerimiento

Validación: (funcional FCA) el problema es resuelto de manera apropiada que el usuario obtenga el producto correcto.

#### ▼ Informes de estado

Se ocupa de mantener los registros de la evolución del sistema.

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

Incluye reportes de rastreabilidad de todos los cambios realizados a las líneas base durante el ciclo de vida

#### ▼ Plan de gestión de la configuración

- Reglas de nombrado de los IC
- Herramientas a utilizar para SCM

- Roles e integrantes del comite
- Procedimiento formal de cambios
- Plantillas de formularios
- Procesos de auditoria → el plan de SCM es una parte del Plan de proyecto(hoja de ruta para construir el producto)

#### ▼ Prácticas continuas

- **Integración continua:** se refiere a la práctica de combinar frecuentemente y de manera automatizada los cambios de código realizados por diferentes miembros del equipo en un repositorio compartido.  
Esto permite detectar y solucionar errores de integración tempranamente, lo que ayuda a garantizar la calidad del código
- **Entrega continua:** es una extensión de la integración continua, que implica la automatización del proceso de construcción, prueba y empaquetado del software en cada cambio de código, y la entrega de estos paquetes a un entorno de pruebas o de producción para su evaluación. La entrega continua permite a los equipos de desarrollo detectar y corregir errores en etapas tempranas del ciclo de vida del software.
- **Despliegue continuo:** es una extensión de la entrega continua, que implica la automatización de todo el proceso de despliegue del software en un entorno de producción. Esto permite que los cambios de código sean entregados a los usuarios finales con mayor rapidez y frecuencia, lo que reduce el tiempo de lanzamiento de nuevas funcionalidades y mejora la experiencia de los usuarios.