INGENIERÍA Y CALIDAD DE SOFTWARE

Unidad 2: Gestión Lean Ágil de Productos de Software

Proceso

- Definido: PUD y RUP
- Empírico: dos corrientes que los adoptan y utilizan. Filosofía Agile (surge en el software, 4 valores y 12 principios) y Lean (nace en la industria automotriz – Toyota System)

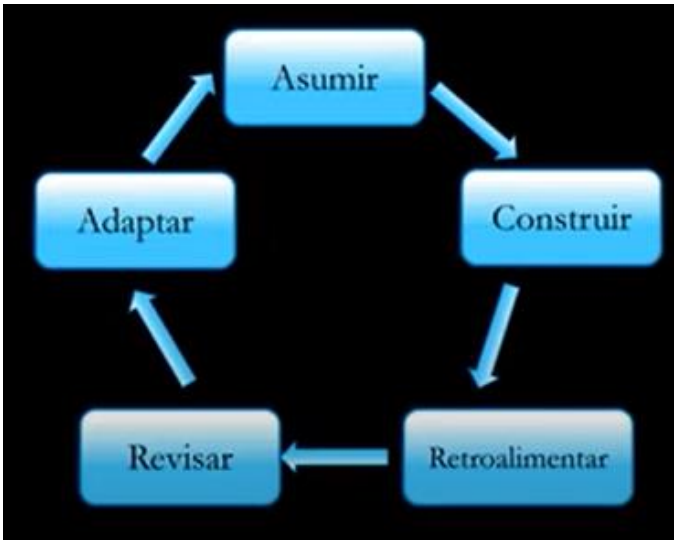
TIPOS DE PROCESO:

PROCESOS DEFINIDOS

- Origen: inspirados en las líneas de producción.
- Basados en: repetibilidad → asumen que se puede repetir el mismo proceso una y otra vez, indefinidamente, y obtener los mismos resultados.
- Intentan ser la expresión de completitud: establecer de antemano e identificar todas las cosas que se van a necesitar hacer para cumplir con los objetivos del producto; orden, artefactos (lo que se va a generar en cada actividad (métricas y herramientas)), roles responsables de llevar a cabo las tareas, descripción detallada del paso a paso.
- Definido de antemano a nivel organizacional y por gente diferente de la que va a hacer el trabajo. Organización define y quienes definen son roles que no son los que van a hacer el trabajo después.
- Cambios más costosos porque los procesos se definen organizacionalmente, no sobre los que lo van a hacer.
- Pueden elegir cualquier ciclo de vida. A veces terminan con procesos obsoletos.
- Motivación: tener visibilidad → necesidad natural de todos, que quieren saber lo que están haciendo.
- PUD y RUP: cuenta la máxima cantidad de opciones posibles para hacer un producto de software. Se adapta a cada situación en particular, si bien hay veces que no se toma así.
- Quieren poder predecir lo que va a pasar en cada momento, gente que se basa en estos procesos quiere tener la ilusión de control, aunque la gente hace lo que quiere. → IDEA poder predecir hacia adelante en base a acciones que tuvo la organización anteriormente.

PROCESOS EMPÍRICOS:

- Origen: en una universidad sembraron pasto, esperaron un año, y luego de eso se fijaron por donde la gente pasaba ya que se había hecho un “caminito”, y allí construyeron las sendas peatonales.
- Basados en: experiencia, pero no la de antes, la de hoy en este momento en particular y no extrapolada, este equipo, este proyecto. Se tiene que generar, no es fácil de conseguir. Asume procesos complicados con variables cambiantes. Proceso se puede repetir pero resultados diferentes.
- Experiencia y conocimiento es el que va a servir para hacer las concepciones.
- **Si o si utilizan ciclos de vida iterativos para la retroalimentación**.
- Quien decide lo que se va a hacer, y cómo y cuándo se revisa, acá lo ve el propio equipo, que son quienes realmente hacen el trabajo. No recibir estimaciones impuestas de gente de afuera.
- No hay ninguno completo, son lineamientos, buenas prácticas de alguna parte, frameworks, vías que cubren algún aspecto: cómo se gestiona, ingeniería, pero NUNCA cubre la totalidad de las cosas para hacer porque justamente es lo que deja librado a cada equipo que decida basado en la experiencia qué quiere hacer, cómo y cuándo.
- Se basa en: ciclos de entrega cortos para poder generar retroalimentación que sirva como experiencia para poder seguir trabajando. Si algo no sale bien, avanzan. (EXPERIENCIA LA GANAN SOBRE LA BASE DE LA PROPIA RETROALIMENTACIÓN).



Patrón de conocimiento en procesos empíricos

- Empezar con algo → ASUMIR
- Después se construye → CONSTRUIR
- Y eso nos da la retroalimentación que nos da la información para
- REVISAR
- Y en base a eso ADAPTAR

- Dicen que la experiencia es aplicable a cierto equipo, no se puede extrapolar a otros equipos, contextos, proyectos.

De dónde → **3 pilares** () que hacen que se pueda retroalimentar y ahí es donde se gana la experiencia:

- **Inspección**: no de alguien externo, sino de alguien del equipo. Permite feedback, retroalimentación, ver espacios de mejora y realizar ajustes para ser más efectivos
- **Adaptación:** capacidad de adecuación a lo inspeccionado. Adaptarse rápidamente a lo nuevo y a los cambios.
- **Transparencia**: visible para todo el mundo sin haber nada oculto. La información es de todos, hacerse cargo de hacer una tarea y blanquear en qué situación se está. Claridad, objetivo y situación del proyecto tienen que ser visible para todos en todo momento. Es el que permite crecer como equipo y transformar el conocimiento en el producto.

BUSCAR RELACIÓN CON USER STORIES

Se basan en ciclos de vida iterativos y cortos. Frameworks que no son completos, dan solo algunas pautas para ciertas cosas, no procesos ni metodologías. Ágil es un pensamiento.

Proceso lo definen con lo menos que el equipo necesite para funcionar, y lo que hace, aferrándose a ciclos de vida iterativos (SI O SI porque es como se gana la experiencia, a través de la retroalimentación rápida y constante, ciclos cortos donde se ve si se tiene algo que corregir y se adapta).

Cada proyecto tiene su particularidad y para aprender se necesitan los ciclos cortos de retroalimentación.

FILOSOFÍA ÁGIL

- Movimiento que se gestó desde los programadores, se juntaron e hicieron ↓
- MANIFIESTO ÁGIL: es un compromiso, voluntad de trabajar de determinada manera, independientemente de las prácticas concretas que se realizan.
- Al principio no había ingeniería.

MANIFIESTO ÁGIL

- Forma de trabajo sustentada en los conceptos de los procesos empíricos.
- 4 valores y un conjunto de 12 principios

VALORES ÁGILES

- **Individuos e interacciones por sobre procesos y herramientas**.

  Más importante la comunicación en el equipo de trabajo.

  No implica que no deban utilizar procesos y para modelar se deban utilizar sólo herramientas simples como pizarrón y afiches.

  Procesos ayudan y herramientas mejoran la eficiencia, pero no consiguen resultado por sí solas.

Deben adaptarse a la organización y el equipo, y no al revés.

Personas son lo más importante → capacidad de ser creativas e innovar

- **Software funcionando sobre documentación extensiva**. *

NO implica no documentar, sino hacerlo cuando se necesita (ver lo que vale la pena, y no todo es permanente), decisiones que se toman deben quedar sentadas en algún lugar.

Valor del cual muchísima gente se aprovecha para no generar ninguna documentación. Existe la necesidad de mantener información sobre el producto de software y el proyecto. Decisiones arquitectónicas que el equipo tomó tienen que quedar documentadas.

Documentos son transferencia de conocimiento y permite mantener la integridad del producto.

Modelos del producto debe quedar documentado para saber desde dónde se parte, estándares de programación.

Importa más el feedback, retroalimentación y valor que genera ver al usuario utilizando el software, que un documento que especifique los requisitos detallados del mismo.

- **Colaborar con el cliente sobre la negociación contractual**. **

Sinceridad, generar un vínculo con el cliente, hacerlo formar parte del proyecto, cliente es un miembro más del equipo.

Punto donde se pone a prueba → cambios de requerimientos, se empieza a discutir.

Problemas de las negociaciones → requisitos.

Tiene más sentido ir creando el producto con una retroalimentación continua durante su desarrollo, que cumplir a rajatabla lo pactado de antemano.

De nada sirve entregarle un software al cliente que ya no es relevante para ellos → entornos cambiantes.

En vez de estar resistiéndose a querer refutar cosas que se dijeron, los requerimientos cambian, entonces tarde o temprano hay que responder a esos cambios.

Colaboración continua con él genera más valor que el cumplimiento estricto de un contrato.

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

No tiene sentido utilizar planteamientos rígidos en escenarios cambiantes como lo es el desarrollo de software. Factor inherente = cambio, evolución rápida y continua.

Justamente las metodologías ágiles promueven la anticipación y la adaptación, frente a la planificación y control que evite las desviaciones, como plantea la gestión de proyectos tradicional.

PRINCIPIOS (12 principios del manifiesto ágil)

1) Mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software de valor. *

No esperar hasta el final para entregar el producto completo

Clientes obtienen valor rápidamente

Lo más importante es el cliente porque es quien define el producto

2) Aceptar que los requisitos cambien. *

Cambiar requisitos incluso en etapas avanzadas → nuevos conocimientos o cambios en el mercado

Al principio no se sabe casi nada

Se fomenta la capacidad de adaptación y flexibilidad para ajustarse

3) Entregar software funcional frecuentemente.

Preferiblemente en semanas o meses

Entrega continua de partes funcionales para realizar ajustes en función de su retroalimentación

4) Responsables de negocios, diseñadores y desarrolladores deben trabajar juntos día a día durante el proyecto.

Técnicos y no técnicos colaborando constantemente

Interacción continua entre los interesados y el equipo → alineados → objetivos y requisitos

5) Desarrollamos proyectos en torno a individuos motivados.

Proporcionar el entorno y apoyo que necesitan para que estén comprometidos y puedan lograr crecer tanto profesional como personalmente

6) Método más eficiente de comunicar información es conversaciones cara a cara.

Si bien la tecnología brinda muchas formas de comunicación, comunicación directa es lo más efectivo para transmitir información

Evita malentendidos y ayuda a entender realmente lo que se quiere

7) Software funcionando es la principal medida de éxito. *

Más valor → capacidad de entregar un software que funcione correctamente y cumpla a centrarse en documentación exhaustiva y promesas

Software mide el progreso

8) Procesos ágiles promueven el desarrollo sostenible.

Plazos de entrega fijos

Mantener ritmo constante y productivo desde el inicio del proyecto y evitar sobrecargas

9) La atención continua a la excelencia técnica y al buen diseño mejor la Agilidad.

La calidad del negocio no se negocia, se puede ajustar cualquier otra cosa, pero no la calidad

Asegurar la calidad, escalabilidad y mantenibilidad a largo plazo

10) Simplicidad es esencial.

Maximizar la cantidad de trabajo no realizado → centrarse en lo esencial y evitando agregar características porque al fin y al cabo no se usan

11) Las mejores arquitecturas, requisitos, y diseños emergen de equipos auto-organizados.

Contrarresta a que las decisiones las toman expertos ajenos al equipo en los definidos

La mejor gente que define las características del producto es el equipo que lo desarrolla

Va surgiendo a medida que el producto va creciendo con el equipo que lo hace

Equipo tiene la capacidad de tomar decisiones de manera colaborativa

12) A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo y de acuerdo con esto ajustan su comportamiento.

Mejora continua → identificar áreas de mejora y realizar ajustes

Mentalidad de aprendizaje y adaptación constante para ser más efectivos y eficientes en la entrega de valor al cliente

- Product Owner: capacidad de decisión, qué se necesita en la actualidad y en el futuro. Éxito del proceso de un desarrollo ágil.
- Equipos auto organizados: hacerse responsable de las decisiones que se toman, no hay un jefe que asigna trabajo a nadie.
- Requerimientos emergentes: aparecen mientras el producto crece y evoluciona. Muchas cosas hasta que no se ven no se da cuenta que lo necesitamos. Terminan siendo el 50%, el otro 50% son los conocidos. Los NO conocidos son el margen de error que se tiene como profesionales.

DESARROLLO ÁGIL DE SOFTWARE (AGILE)

- Agilismo es mucho más estricto respecto de sus formas de ejecutarse.
- Agilismo no es la legalización del lío.
- Ágil → ideología con un conjunto de principios que guían el desarrollo del producto.
- Orientados a la gente en lugar del proceso
- Métodos adaptables y no predictivos

Ejemplos de frameworks ágiles:

- FDD, ATDD, CRYSTAL, SCRUM, XP

Lo importante es el valor de negocio que se entrega al cliente.

REQUERIMIENTOS ÁGILES

Software es una herramienta, un medio para un fin, no un fin en sí mismo.

User Stories → ver requerimientos más arriba, nivel de abstracción más alto, nivel de negocio.

- Descripción: necesidad que tiene el usuario con su terminología, en la medida de lo posible que lo escriba él, siendo cortas.
- Priorización: saber cuándo parar. Determinar qué es lo bueno suficiente para entregarle al cliente. Viene de la mano del PO (su responsabilidad está sobre la base de la lista priorizada de producto = product backlog).
- Cambios son bienvenidos aún en etapas finales del proyecto.
- **Just in time** → pelea con la definición completa de antemano de todos los requerimientos. Dice que hay que describir los requerimientos conforme haga falta. Analizo solo cuando necesito, y de esta manera, se da la **eliminación de los desperdicios**, no invertir tiempo en requerimientos que después van a cambiar.
- **Co locados** → trabajar todos en el mismo espacio físico, para poder construir y debatir juntos el producto que se va a realizar.

TRIPLE RESTRICCIÓN o TRIÁNGULO DE HIERRO → dimensiones en las que se mueve un proyecto en función de las decisiones que se tienen que tomar.

- Alcance: requerimientos. Fijo porque de este surgen los otros dos.
- Costo:
- Tiempo:
- En realidad lo que tiene más importancia es el valor que el producto tiene para el cliente, y por otro lado, las características de la calidad que le ponemos al producto.
- Después se le agregan las personas como lo más importantes, porque son las que desarrollan el producto, y son los que piden el producto.

En cada iteración → dejar fijos los recursos y el tiempo. La iteración va a durar lo que el equipo decida, pero una vez definido, no se cambia → time box.

24/08

¿Qué caracteriza a los ***REQUERIMIENTOS ÁGILES***?

- ***Hay un FOCO en valor de negocio***: **Software es una herramienta, un medio para un fin, no un fin en sí mismo**. ES EL MEDIO POR EL CUAL LE ENTREGAMOS VALOR AL CLIENTE. No solo crear valor en el cliente, sino también darle valor al negocio en hacer mejorar las cosas, cumplir con objetivos, entre otros. Entregar valor de negocio, no características de software.
- ***Usar historias y modelos para mostrar qué construir***. Se trabaja con el cliente también, por ejemplo el PO que también está del lado del negocio, juntos a través de historias se usan como base para armar la definición del producto. Mejor forma → comunicación. Usa las US. Proceso de descubrimiento en conjunto con el cliente, trabajo con el equipo para ver qué es lo que se quiere.
- Se parte de una visión de producto y es lo que va a determinar la definición de la primera versión de producto que se va a tener (primera versión de producto: objetivo y alcances, y conformación principal del artefacto)
- ***Determinar que es "solo lo suficiente"*** → hay que empezar con lo mínimo necesario, y después vamos completando. Realidad muestra situación no favorable en que sólo se usa siempre 7% de las características del producto.
- Product backlog: pila PRIORIZADA. Contiene la cantidad suficiente de historias para ejecutar la primera iteración. Problema de la priorización recae en el PO, que proviene del negocio.
- ***Just in time***: diferir decisión. Eliminación de desperdicios, solo detallar lo que hace falta, porque si se detalla todo se pierde tiempo porque después cambian. Ni antes ni después, analizar cuando se necesita.
- ***El cara a cara permite que fluya información con realimentación rápida***. Basada en varios principios del manifiesto que tiene que ver con la comunicación.
- Los mejores requerimientos emergen de equipos auto organizados.
- Acá surgen las User Stories. (no son un artefacto nativo de scrum)

Gestión ágil intenta equilibrar y contrarrestar → agregar cosas en el producto que no se usa, no cumple, está de más.

Dueño del producto es el que realmente tiene claro de cuáles son sus necesidades → su responsabilidad PRINCIPAL → PRIORIZAR NECESIDADES → decidir qué es lo que se tiene que hacer primero, en base a sus necesidades, qué es lo que tiene mayor valor para el negocio, lo cual se ve plasmado en ↓

PRODUCT BACKLOG → contenedor de "características" que hacen falta, LISTA PRIORIZADA ORGANIZADA, lo más importante se ubica arriba, y lo que menos abajo. Entonces → priorización → PO (por eso para poder trabajar de manera ágil se necesita un cliente que **realmente quiera hacerlo**, alguien que sepa **sobre el producto**, **capacidad de tomar decisiones**, ganas de estar **disponible** para el equipo para contestar dudas frecuentemente → mejor medio de comunicación es cara a cara en vez de un documento formal que diga todo lo que se tiene que hacer). Justamente → pila → no está una cosa al lado de la otra, sino una debajo de la otra.

Donde cada iteración implementa los requerimientos de prioridad alta.

Justamente lo que se busca es cambiar el enfoque, de dejar fijos los requerimientos e ir estimando los recursos y el tiempo ↓ ahora

- Dejar fijo el tiempo con iteraciones de duración fija → sprints. El valor se asocia a la utilidad, beneficio o satisfacción que se le ofrece a los usuarios finales por cada funcionalidad completa que se le entrega.
- Recursos fijos: equipo de trabajo con una determinada capacidad asignado a trabajar.

En base a eso se acuerda el alcance en esa iteración con el equipo → Acordar cuánto es lo que el equipo puede entregar de software funcionando para ese tiempo. Cambio importante en términos de GESTIÓN.

Tipos de requerimientos:

- Requerimiento de negocio
- Requerimiento de usuario (es por medio del que se llega a cumplir el de negocio) FOCO DE LO ÁGIL
- Requerimiento funcional
- Requerimiento no funcional
- Requerimiento de implementación

RESUMEN: trabajar juntos hablando técnicos y no técnicos entendiendo las necesidades del negocio y seguidamente del usuario. Si no se entiende el negocio, no vamos a poder construir un software que lo ayude. Luego, junto con los usuarios descubrir cuál es la mejor forma de satisfacer esas necesidades → product backlog: es lo que se usa para determinar en una iteración qué es lo que se va a entregar. De la entrega y obtenemos feedback, que sirve como retroalimentación. Asumimos:

- Cambios son la única constante. Siempre van a existir.
- Mirada de quienes son los involucrados → stakeholders: todos lo que tienen que decir algo sobre el producto. PO: representante de ellos, parte de su responsabilidad trabajar con ellos.
- El usuario dice lo que quiere cuando recibe lo que pidió → el usuario va a decir si está contento cuando lo vea. La mejor retroalimentación es la que se obtiene cuando se ve al usuario utilizando el producto o servicio.
- No todas las herramientas sirven para todos los casos → saber identificar cuáles son, ir aprendiendo en cada situación.
- Lo importante no es entregar una salida, un requerimiento, lo importante es entregar, un resultado, una solución de "valor".

Principios ágiles que tributan la gestión ágil de los requerimientos ↓

USER STORY (Historia de usuario) → usuario quiere contarnos algo que considera importante que necesita. Descripción corta de una necesidad que tiene el usuario respecto de un producto de software, entonces se transforma en una técnica PARA la especificación de requerimientos de usuario. Son:

- Necesidad del usuario → expresiones de intención.
- Descripción del producto
- Item de planificación → sirven como entrada a la documentación
- Token para una conversación
- Mecanismo para diferir en una conversación
- No son especificaciones detalladas de requerimientos

- Necesita poco o nulo mantenimiento y puede descartarse después de la implementación

Parte más difícil de construir → sistema de SW: decidir precisamente qué construir, establecer los requerimientos técnicos detallados. Ninguna otra parte del trabajo afecta tanto al sistema resultante si se hace de forma incorrecta.

Lo más difícil del SW → REQUERIMIENTOS. Porque su determinación es un proceso social.

EN EL PARCIAL VAN PREGUNTAS DEL ARTÍCULO → NO SILVER BULLET. Software es complejo, y si no se entiende no se va a poder entregar un producto de calidad.

Partes de una USER STORY → 3 C:

- **TARJETA (CARD)** → frente: tarjeta, dorso: confirmación (práctica: pruebas de aceptación de usuario)
- **CONVERSACIÓN** → la más importante. No queda guardada en ningún lugar formal. Idea → usuario disponible para sacar dudas, y después el equipo se pone a trabajar.
- **CONFIRMACIÓN** → pruebas de usuario que se identifican necesarias para hacerle a la funcionalidad y sirven para que el PO acepte las características de software que se construyen a partir de la US.

Formas de expresar las US → 3 W:

As who, I want what so that why

- **Who** → qué es lo que necesita. Representa quién está realizando la acción o quién recibe el valor de la actividad. NO es usuario, toda persona que usa el sistema es un usuario, no da representación.
- **What** → quién lo necesita. Representa la acción que realizará el sistema.
- **Why** → para qué lo necesita. Comunica porque es necesaria la actividad, da el valor de negocio y es lo que sirve para PRIORIZAR.

En base a esto, **EL PO PRIORIZA LAS US EN EL PB**. Agrega, quita, saca, prioriza.

US son PORCIONES VERTICALES → tiene parte de interfaz de usuario, de lógica de negocio y persistencia o base de datos. Si se cortan horizontalmente NO se entrega valor, el usuario no puede hacer nada con eso.

Todo en Agile es relativo.

Cuando el PO no está disponible → PROXIES (usuarios representantes) → perfiles del negocio, no son los que desarrollan software, sino, gerentes de usuarios/desarrollo, alguien de marketing, vendedores, clientes.

CRITERIOS DE ACEPTACIÓN → Formalización de las cosas que el PO nos va a exigir que estén especificados en las user stories, información concreta que va a definir si lo que se implementó es correcto o no. Van a servir para definir las pruebas de aceptación y generar la 3C: Confirmación.

- Definen límites para una US.
- Ayudan a que los PO respondan lo que necesitan para que la US provea valor. (req func mínimos)
- Ayudan a que el equipo tenga una visión compartida de la US.
- Ayudan a desarrolladores y testers a derivar las pruebas.
- Ayudan a los desarrolladores a saber cuándo parar de agregar funcionalidad en una US.
- Definen una intención no una solución.
- Deben ser objetivos y verificables.
- Independientes de la implementación.
- Relativamente de alto nivel → no es necesario que se escriba cada detalle.

¿DETALLES?

- Cada equipo va a definir dónde va a guardar los detalles → documentación interna de los equipos.
- Los detalles no están en la carta, gran parte está en la conversación y la otra en las pruebas.

PRUEBAS DE USUARIO

- Se deben contemplar situaciones de éxito y fracaso, pasa y falla.

- Pruebas deben ser detalladas sino no sirven.
- Lo mejor es que las pruebas las defina el usuario, porque sabe lo que hace la gente, conoce.

DEFINITION of READY (DoR) → medida de calidad que lo crea el equipo y se aplica a la user para saber si está en condiciones de entrar en una iteración de desarrollo, lista para empezar a implementarla. Si no lo cumple, queda más abajo en el product backlog. Sprint backlog → que salga del pb para entrar a las iteraciones, o al sprint backlog.

- **INVEST Model** → serie de características que ayuda a saber si una US está en condiciones de incluirse en una iteración de desarrollo:
    - Independiente: cada user story se puede implementar en cualquier orden, que no dependan las unas de las otras. Calendarizables e implementables en cualquier orden.

Una US puede ser desarrollada, testeada y potencialmente incluso entregada por sí misma.

Produce valor agregado único por si sola, por más que sea secuencialmente dependiente de otra.

    - Negociable: historia está escrita en términos de qué quiero, no cómo quiero hacerlo o cómo se va a implementar. Es el QUÉ no el COMO.

No es un contrato para una funcionalidad específica, sino un espacio para que los requerimientos puedan ser discutidos, tratados entre los involucrados.

    - Valuable: el why, debe tener valor de negocio.

Se puede considerar como el más importante.

US debe proveer valor al usuario, cliente o stakeholder.

Product backlog son priorizados por el valor y justamente las empresas triunfan o fallan basado en el valor que el equipo puede entregar.

Idea de que una US es un corte vertical y no horizontal → así crea valor. *

    - Estimable: es asignarle un peso a la US, un número que permita compararla y determinar el esfuerzo, complejidad, incertidumbre que conlleva.

A veces una historia es tan compleja o cosas que no tienen rta, que no se pueden estimar. Si no se puede estimar, NO es US, sino que es una SPIKE.

Una US de cualquier tamaño puede estar en el PB, pero para poder ser desarrollada y testeada en una iteración, el equipo le debe poder dar una estimación aproximada de lo que conlleva.

Principal beneficio de estimar no es darle un peso, sino dar cuenta de cosas nuevas, criterios que faltaban, la clarificación del entendimiento de la historia → el proceso de la conversación para la estimación termina siendo tan importante como la propia estimación.

    - Small: consumible en una iteración (sprint), empezar y terminar en un sprint. Depende mucho del equipo, experiencia, duración de la iteración. Suele traer problemas.

Si no es considerada que puede completarse en una iteración, no entrega valor ni puede ser considerada *done* en ese punto.

    - Testeable: demostrar que esa US efectivamente se implementó cumpliendo los criterios de aceptación que se definieron.

Si no es testeable → mal formada/pensada, muy compleja o dependiente de otras en el PB.

    - Algunos equipos le suman que debe tener un prototipo, reglas de negocio, clases, entre otros.

DEFINITION of DONE (DoD) → definición de hecho o terminado, Código comentado, respetando los estándares, documentado, etc. Acuerdo entre el PO y el equipo de cuándo un trabajo está completado y es potencialmente entregable y utilizable (se puede hacer un release) al finalizar cada iteración.

- Permite: **tener siempre un producto «potencialmente entregable y usable»** al finalizar cada iteración, con el mínimo esfuerzo.
- Define qué entregables y mínimos de calidad tienen se tienen que cumplir en TODOS los objetivos / requisitos que se van a ir aceptando durante cada iteración del proyecto.
- 3 mínimos componentes: cumplir con requerimientos funcionales, no funcionales y ser de calidad, ENTREGAR VALOR AL CLIENTE.
- Si no llega a cumplir con el DoD, entonces no se cuenta en la velocidad y debe volver al product backlog para corregir lo que sea necesario.

SPIKES: nivel de incertidumbre (falta de información) tal que esa US no se puede estimar. Dos tipos:

- Técnica: tiene que ver con el cómo. Vienen más del lado de la tecnología.
- Funcional: tiene que ver con el qué. Vienen más del lado del negocio.

ESTIMACIONES → valoración cuantitativa de la realidad que se quiere considerar, no es precisa tiene alto riesgo de incertidumbre y si estamos en el inicio del proyecto aún más todavía, no es planear, no son compromisos, son la base de los planes pero no tienen que ser lo mismo.

Lo primero que se estima:

- Tamaño del producto que el cliente espera.

Realidad → proyectos terminan en un orden de 4 veces más de lo que se estima.

Siempre va a haber errores → hay incertidumbre siempre.

Errores más comunes:

- Actividades omitidas: por ejemplo el retrabajo, gestión del proyecto, no sólo la programación, revisiones, testing, mantenimiento, días de enfermo, conversaciones.
- Proceso de estimación: todos tienen sesgo, margen de error.
- Falta de información: se asumen ciertas cosas pero que en realidad no se saben.
- No claridad del proceso que se va a utilizar para trabajar.

Métodos utilizados en el enfoque tradicional

- Basados en la experiencia
    - Datos históricos: partir de la idea que la experiencia de otros equipos en otros proyectos sirve para estimar el nuevo proyecto. Tecnologías, perfil de gente, esfuerzos.
    - Juicio experto: Puro. Conocimiento organizacional está en la cabeza de uno que es el experto, hablando de software no se puede ser experto en todo. Delphi: grupo de expertos.
- Basados exclusivamente en los recursos
- Método basado exclusivamente en el mercado
- Basados en los componentes del producto o en el proceso de desarrollo
- Métodos algorítmicos: fórmulas con distintas variables

¿Cómo se estima en los ambientes ágiles?

1) **Son relativas**: en contraposición a las tradicionales que son absolutas. Los seres humanos son mejores comparando, además es más rápido, qué tanto más. Se hace estimaciones por comparación en el enfoque ágil (poker planning o poker estimation → con US canónica)
   ¿Dónde se estima? → En dos momentos: primero más generalizado y después durante la sprint planning (dos momentos: uno en el que se estiman las historias y después cómo se piensas que se lleva a la implementación esas historias).
   Priorización → tiene q ver con el valor de negocio.
   Estimación → tiene q ver con cuánto esfuerzo y recursos le conlleva al equipo.
2) **Foco en certeza** → Se hace foco en la certeza y no en la precisión, porque esta es cara. Los enfoques tradicionales hacen mucho énfasis en la precisión y luego no suelen cumplirlos. No hacer el esfuerzo de estimar todo el producto.
3) **Diferir las decisiones** → hasta el último momento porque en el proceso se va obteniendo retroalimentación y se va obteniendo certeza de lo que se está haciendo, al principio no se sabe. Estimaciones tempranas, las menos posibles.
4) **Estima el que hace el trabajo**: que es el equipo, a diferencia de los tradicionales que quien estima es el líder del proyecto que muchas veces no tiene nada que ver con el día a día del equipo. No quedarse con una sola idea, tener en cuenta los demás.
5) Principal beneficio de estimar no es darle un peso, sino dar cuenta de cosas nuevas, criterios que faltaban, la clarificación del entendimiento de la historia → el proceso de la conversación para la estimación termina siendo tan importante como la propia estimación.
6) Sirven como respuesta temprana sobre si el trabajo planificado es factible o no.
7) No se deben utilizar como compromiso → NO SON UN COMPROMISO NI UN PLAN
8) A mayor diferencia entre lo estimado y lo planeado MAYOR es el riesgo

Forma de estimar las US ↓

STORY POINT (punto de historia): unidad de estimación de la US. Se necesita dar un valor cuantificado de alguna manera para después comparar. Representa el tamaño de la US, cuan grande, compleja y trabajo requiere. Tamaño no es esfuerzo NO ESTIMAR EN HORAS DE TRABAJO (depende de habilidades, conocimiento, experiencia, familiaridad con los dominios de aplicación/negocio)

- **Serie de Fibonacci** → 0, 1, 1, 2, 3, 5, 8, 13, 21 … **Tiene un crecimiento exponencial al igual que la complejidad del software, por eso se asimilan y se puede utilizar**. Algunos usan ½ para describir algo nada complejo.
- Tamaño por números: del 1 al 10.
- Talles de remeras: S, M, L, XL, XXL.
- Serie 2^n: 1, 2, 4, 8, 16, 32, 64.
- Estima el equipo

- User story canónica: elemento que se va a utilizar para comparar. Contra esta se comparan todas las demás.
- Se tiene que cumplir el DoD (es lo que el PO va a usar para…), para decir TERMINÉ.

Velocidad → métrica más importante porque mide producto. Principio: la mejor métrica del proceso es el software funcionando. No se estima, se calcula al final del sprint → Velocidad = Σ US que el PO aceptó al final de la iteración. Sirve para ir viendo si logro → principio: desarrollo sostenible, equipo estabilizado con ritmo de trabajo sostenible, para tener previsibilidad de lo que se puede esperar del equipo, y cuánto software puede entregar en cada iteración. Sirve para ver cómo va funcionando el equipo.

POKER PLANNING → forma de estimar, donde cada uno estima las tareas por su cuenta para no estar influenciados por los otros. Se pone un número individualmente y después se compara en una puesta en común. Primero empieza explicando el que mayor número de todos, así todos se escuchan. Y después se hace otra ronda de nuevo ahora escuchando los puntos de vista de los demás. Y ahí se supone que debería haber convergencia al mismo valor. Nos basamos en la US canónica a la hora de estimar por nuestra cuenta.

Lo mejor una canónica de 3 porque da margen para abajo y para arriba. La mayoría de los equipos eligen una canónica de 1 que es lo más fácil y sencillo.

3 dimensiones que puede tener una US al momento de asignar un SP (SP es un número que homogeniza estas 3 dimensiones):

- Complejidad: cuan dificultosa es.
- Esfuerzo: trabajo, cuántas horas ideales se necesitan para hacerlo. No es tiempo (calendario). Se mide siempre en horas ideales lineales.
- Duda o incertidumbre: falta de información.

Se llevan las 3 dimensiones a un solo número → Story Point.

TAMAÑO NO ES ESFUERZO. Concretamente cuánto tiempo en HORAS va a llevar realizar la tarea.

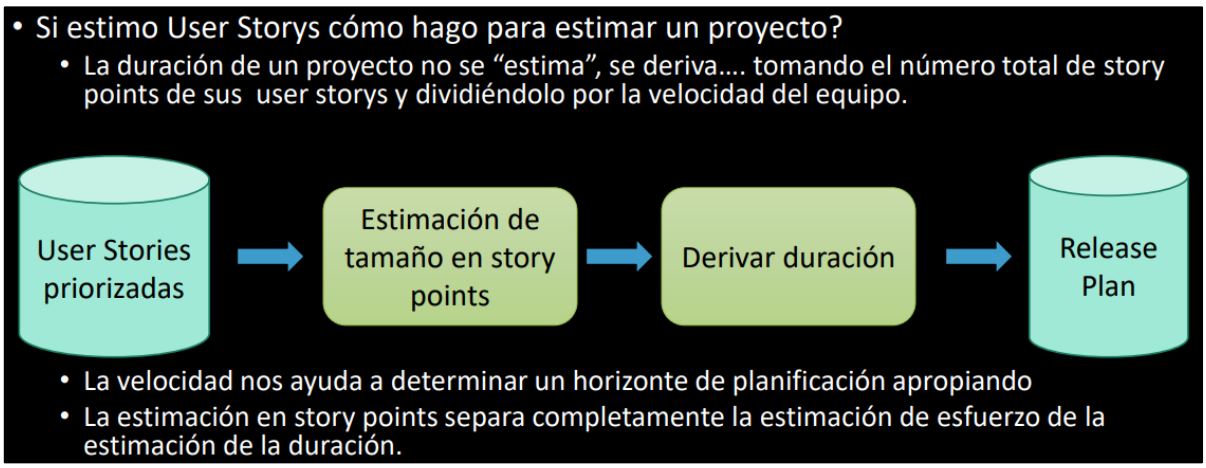
ESTIMAR → tamaño de lo que vamos a hacer. Se da como equipo y que el mismo pueda definir. COMPARAR.

Para medir el tamaño → STORY POINT.

La base story o US canónica:

- No sea algo demasiado grande
- Que tenga la menor duda posible
- Que sea relativamente chico

Idea producto → US = 150. Equipo compromete 15 SP por iteración. Iteración = 1 mes. Total = 10 meses.



GESTIÓN DE SOFTWARE COMO PRODUCTO

Gestionar productos en niveles ágiles

¿Por qué creamos productos?

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

IDEA → Sí se puede desarrollar un producto de manera iterativa e incremental. Empezar por una versión resumida que vaya a la esencia del producto y a partir de eso obtener realimentación para corregir y salir al mercado después con otra versión corregida y mejorada. No hay nada que te diga que un producto tiene que salir al mercado con el 100% de su funcionalidad.

Evolución de los productos de SW

- **FUNCIONAL** → base: que haga lo que tenga que hacer.
- **CONFIABLE** → usuarios se sientan cómodos y seguros usando el resultado que brinda ese producto, se puede depender de ellos.
- **USABLE** → cuando disfruto o uso haciendo lo que tengo que hacer usando el producto de software. Tiene que ver con la experiencia de usuario, realmente mejora su calidad de vida.
- **CONVENIENTE** → si uso el producto es más productivo, logro cosas que antes no lograba.
- **PLACENTERO** → no sea un esfuerzo usarlo, cuando se disfruta utilizarlo.
- **SIGNIFICATIVO** → cambia la vida de las personas, cambiar el hilo de la historia.

idea es focalizar el producto en las experiencias de quienes le darán uso.

Tiene que ver con la utilidad en vez de la usabilidad.

Difícil → focalizado en las tareas en vez de en los usuarios.

¿Cómo logro hacer un producto significativo?

No se puede lograr hacer todo en una sola vez, lo primero que surge:

UVP → idea de un producto único y de valor para quien se le ofrece el producto. Comprender que un nuevo producto tiene una hipótesis de valor único. Una hipótesis → siempre basada en el usuario.

Siguiente paso, crear un MVP para probar su hipótesis.

MVP: PRODUCTO MÍNIMO VIABLE

- Lo mismo que se necesita para probar si se está alineado en el producto correcto. Conjunto de características para armar un producto que necesitamos salir a validar.
- Propósito → evaluar/validar una hipótesis: si realmente satisface una necesidad, si el producto va a conseguir clientes, las descargas que se necesitan, retención, que se use. ES EL PRODUCTO CORRECTO O NO.
- Se da la posibilidad de que, si la hipótesis sale mal validada, se pueda cambiar.
- Idea de que estamos creando el producto, estamos validando si el producto es el que realmente se necesita. Se sale a la búsqueda de clientes potenciales que lo validen, mostrarlo para que el cliente interactúe y dé feedback, retroalimentación.
- Hay veces que el validar la hipótesis termina generando una variación importante del producto, o incluso sale otro producto → surgen + MVPs.
- Clave del éxito → que esté clara la hipótesis.
- Objetivo: feedback corto a un costo razonable. No vender. MVP no siempre es un producto, puede ser un video, un prototipo.
- Ver lo que la gente realmente hace con respecto a un producto es mucho más confiable que preguntarle a la gente qué harían. Observar su comportamiento real con el producto o servicio.

MVF: CARACTERÍSTICA MÍNIMA VIABLE

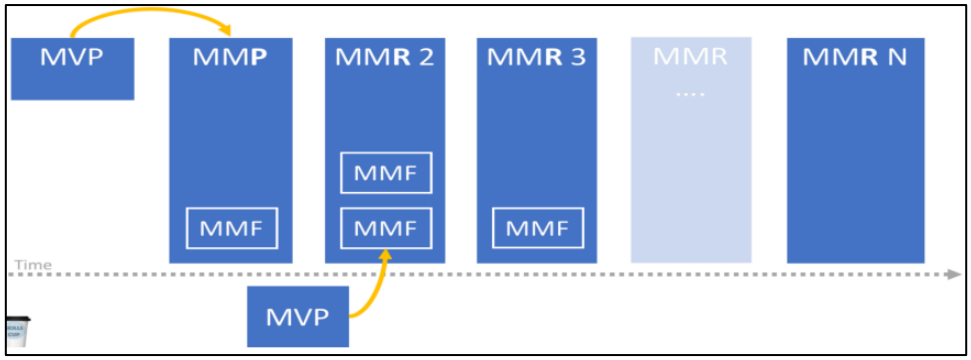
- Feature. Una característica.
- Conjunto de MVF arman MVP.
- Es parte de un MVP.
- A veces un MVP en un MVF.

MMF: CARACTERÍSTICA MÍNIMA COMERCIALIZABLE

- Marketing, comercializable. Ya es un producto que va a ofrecerse con un resultado concreto QUE SE VA A OBTENER.
- Tiene que ver con las características mínimas que debe tener un producto para salir a comercializarlo.
- Cuando se tiene es porque ya hubo un MVP que se validó, y se tiene un respaldo de que se va a utilizar.
- Pieza más pequeña de funcionalidad que puede ser liberada.
- Objetivo: vender, ganar dinero.

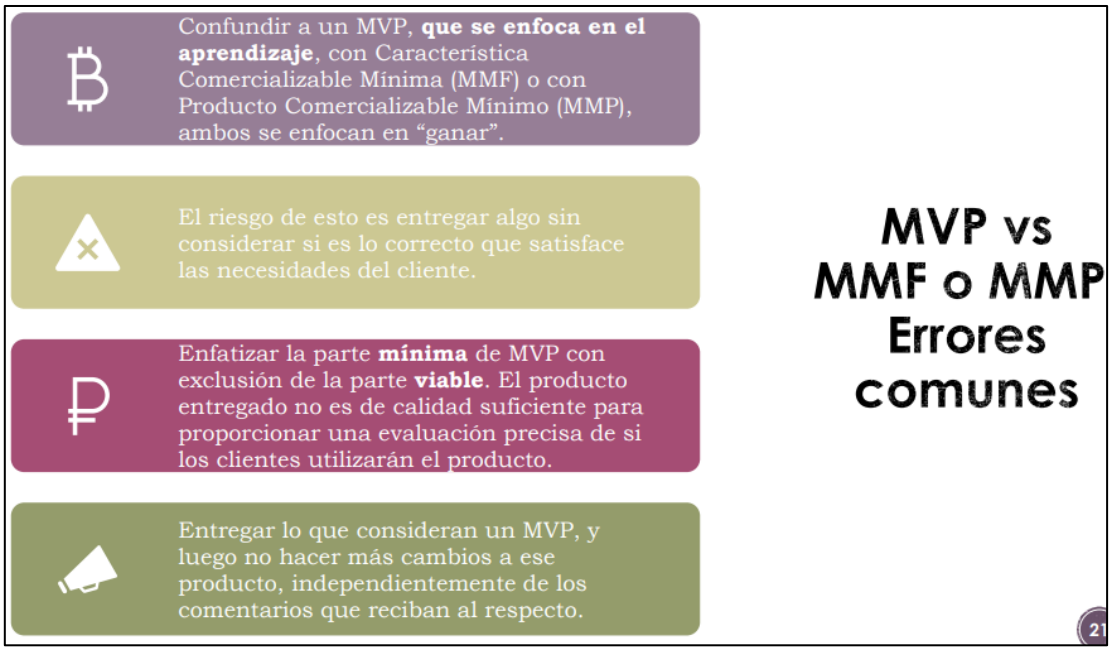
MRF: CARACTERÍSTICAS MÍNIMAS DEL RELEASE

- Release: versión del producto que tiene la madurez suficiente para sacarlo al mercado.
- Un release capaz se obtiene en muchas iteraciones.
- Ya tiene que ver con la puesta en producción, características mínimas para que un producto pueda ponerse en producción.



- Cuando parto de un MVP parto de la validación de una hipótesis.
    - Versión de un nuevo producto creado con el menor esfuerzo posible
    - Dirigido a un subconjunto de clientes potenciales
    - Utilizado para obtener aprendizaje validado
    - Más cercano a los prototipos que a una versión real funcionando de un producto
- Superada esa hipótesis y cuando se dice sí y se cierra el momento de validación.
- Se evoluciona a algo que es vendible la v desaparece, y aparece la m que es con lo que se va a salir al mercado → MMP: mínimo producto vendible.
    - Primer release de un MMR dirigido a primeros usuarios
    - Focalizado en características clave que satisfarán a este grupo clave
- Cuando salió al mercado, a la vez, es r ya es producto con un nivel de madurez, es el primer mínimo release → MMP = MMR1.
    - Incremento más pequeño de un producto que tiene el conjunto de características más pequeño posible y ofrece un valor nuevo a los usuarios y satisface sus necesidades actuales.
- Y a partir de ahí se va evolucionando con distintos release del producto.

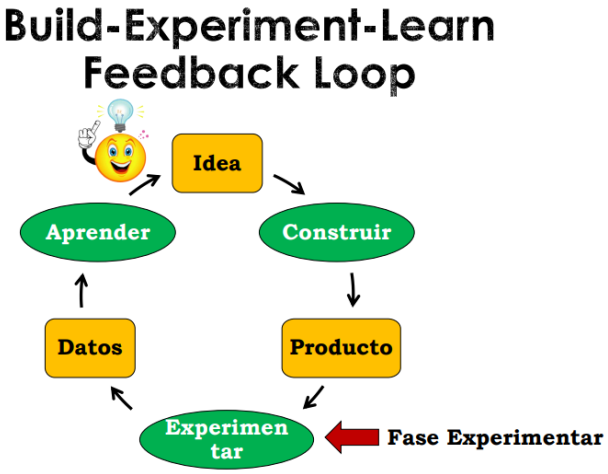
IDEA → EL PRODUCTO SE DESARROLLA INCREMENTALMENTE. SE PUEDE EMPEZAR CON UN CONJUNTO PEQUEÑO DE CARACTERÍSTICAS, Y EN BASE A ESTO IR MEJORÁNDOLAS.



Método Lean Start – Up

Desperdicio: todo lo que no genera valor. Es la motivación para mejorar servicios. Hay relación de que la calidad del producto final está relacionada con la calidad del proceso que se utiliza para llevarlo a cabo.

- Relación: se usa el MVP para generar un producto que dé valor → eliminar/evitar desperdicio.



MVP → fase experimentar

Se usa como medio para validar lo que construí

Ciclo como medio para crear el MVP.

Fase CONSTRUIR: MVP no necesariamente puede ser un producto terminado, es cualquier forma de mostrar una idea.

- Simplificar
- Evitar construcción excesiva
- Cualquier trabajo adicional → desperdicio
- Si duda → quedarse con lo sencillo

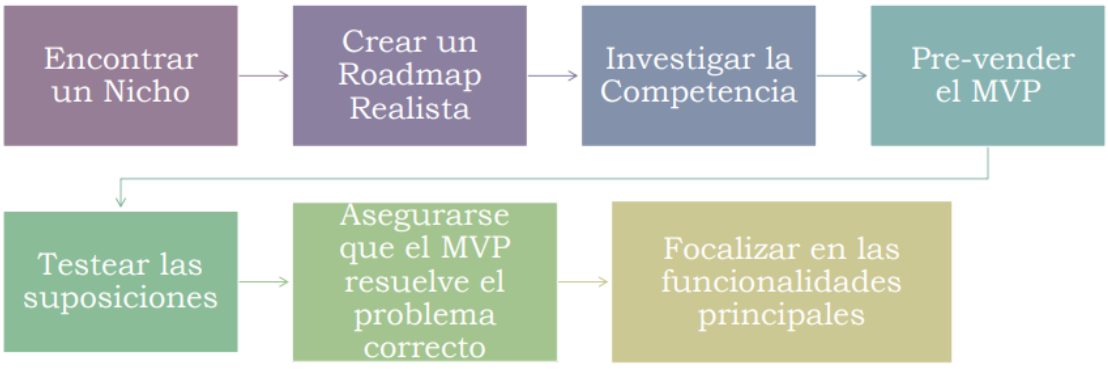
¿Cuál esfuerzo vale la pena y cuál no?

Si se prioriza el valor de negocio por sobre el valor del usuario no vale. El valor de negocio es justamente lo que se logra cuando se le da valor al usuario.

Surgen dos dilemas que resolver ↓

- DILEMA DE AUDACIA CERO: cuando no hay nada para perder, es cuando uno más se anima a arriesgarse. No se tiene clientes, ni ingresos. Te permite crear libremente sin restricciones.
- SUPUESTO DE: SALTO DE FE → lleva a validar dos hipótesis:
    - Hipótesis del valor: se mide con la tasa de retención, qué clientes instalan el producto y se mantienen usándolo
    - Hipótesis de crecimiento: cuántos clientes nuevos voy a incorporar.

PROCESO PARA PREPARAR UN MVP



1. Encontrar un nicho de mercado: porción del producto a la que quiero hablarle, en donde esté contenido un salto de fe
2. Paneo de qué características el producto haría
3. Lo que más se puede saber
4. Retroalimentación
5. Validar
6. Centrarse en las características más importantes, hacer foco en lo mínimo

---

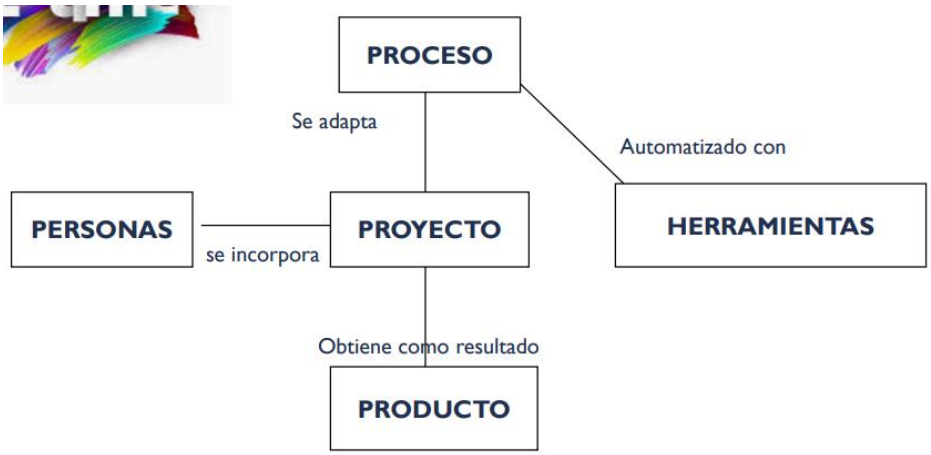
07/09

SOFTWARE

- Primera concepción: programa andando en una computadora.
- Conocimiento empaquetado a distintos niveles de abstracción.

- BD, diseño, caso de uso, US → también es software.
- Todo lo que sale como ejecución de las tareas de un proyecto de software.
- Cada pieza que la gestión de configuración genera/mantiene ES SOFTWARE.
- Colección de programas necesarios para convertir a una computadora (de propósito general) en una máquina de propósito especial diseñada para una aplicación de un dominio particular, incluyendo documentación producto del desarrollo de un sistema.

Disciplina ISW se mueve en tres dimensiones:



- **Proceso** → nivel más abstracto, como una definición de diccionario donde está escrito en términos teóricos qué es lo que se debería hacer para hacer software. Al momento de iniciar un proyecto se tiene que adaptar en caso de que sea PD, o terminar de armar en caso de que sea PE. Proceso se define en términos de roles porque para cada proyecto las personas asumen uno o más roles, por eso el proceso se instancia en cada proyecto. Automatizado con **Herramientas**.
- **Proyecto** → instancia de un proceso, toma de esa definición teórica lo que le hace falta, en el momento de la definición se decide el ciclo de vida que va a gestionar la gente y los recursos para poder obtener un producto de software que sea de calidad y satisfaga los requerimientos de los usuarios.
- **Producto** → lo que se obtiene como resultado.
- **Personas** → se incorpora, es lo más importante, dimensión que no se puede dejar de lado, fundamental. Personas tienen un rol fundamental decisorio en el desarrollo de cualquier proyecto de software.

Diferencia entre ↓ Se complementan, pero no son lo mismo.

- Proceso:
  - Define todas las cosas que debemos hacer para hacer algo.
  - Conjunto de actividades interrelacionadas que toman como entrada requerimientos y obtienen como salida un producto o servicio, en nuestro caso de software.
  - Descripción de las cosas que tenemos que hacer para alcanzar un objetivo.
- Ciclo de vida: te dice el orden a ejecutar las tareas del proceso, en qué momento se van a llevar a cabo. Hace que el proceso se ejecute bien en el contexto de un proyecto.
  - Afecta la forma en la que se gestiona el proyecto. Elegir el correcto ayuda a mejorar la rapidez de desarrollo, mejorar la calidad, el seguimiento y el control, minimizar los riesgos. Mientras que, el equivocado puede que ayude a que el trabajo sea lento, se repita o sea innecesario.
  - Serie de pasos a través de los cuales el producto o proyecto progresa
  - Productos tienen su ciclo de vida y proyectos también.

Tiene el proyecto: fases o etapas, estados de evolución que determinan qué cosas se pueden hacer y en qué orden, y el producto: estados por los que pasa desde la idea hasta que se descontinúa, más grande que el del proyecto. Hay varios proyectos a lo largo del ciclo de vida del producto.

Es una representación de un proceso: especifican las fases y el orden en el cual se llevan a cabo.

Función principal: establecer el orden en el que un proyecto especifica, hace prototipos, diseña, implementa, revisa, testea

Tipos de ciclo de vida DE UN PROYECTO:

- **Secuencial**: cascada, + fácil de gestionar, da ilusión de visibilidad de avance más ordenada, pero en realidad no es así. No se mezcla SCRUM con cascada. Cascada requiere que:
    - La fase de prueba comience solo si la de codificación empezó. Se realiza una revisión al final de cada fase para determinar si se está listo para pasar a la fase siguiente, sino se queda en la que esté.
    - Requerimientos se conozcan en etapas tempranas del proyecto. No es flexible.
    - Es bueno cuando se tiene una definición estable del producto y se está trabajando con metodologías conocidas.
    - Cada fase tenga una salida tangible. Está manejado principalmente por documentos, es decir, el trabajo principal son documentos.
    - No provee resultados tangibles en términos de software hasta el final del ciclo. No es bueno para proyectos que requieren ver progreso en etapas tempranas.
    - El equipo no se entera que se olvidó algo hasta que entra en la etapa de testing, lo cual es un error costoso.
- **Iterativo**: iterativo e incremental de scrum. Conceptualmente quieren decir lo mismo: producto lo voy a entregar en distintas iteraciones o vueltas, donde las primeras van a tener funcionalidad limitada. Cantidad mínima de iteraciones: 2, porque sino, es en cascada.
    - Iel PUD: ciclo de vida de alcance FIJO. Casos de uso significativos para la arquitectura, iteración no termina hasta que no se implementan todos los casos de uso que me comprometí a hacer en esa iteración. Es variable el tiempo.
    - Iel Scrum: ciclo de vida de duración FIJA. Se presenta tal día y no se mueve. Se negocia el alcance.

  En todas las iteraciones se repite un proceso de trabajo similar (de ahí el nombre "iterativo") **para proporcionar un resultado completo sobre producto final**, de manera que el cliente pueda obtener los beneficios del proyecto de forma incremental.

  En cada iteración el equipo **evoluciona el producto** (hace una entrega incremental) a partir de los resultados completados en las iteraciones anteriores, añadiendo nuevos objetivos/requisitos o mejorando los que ya fueron completados. Un aspecto fundamental para guiar el desarrollo iterativo e incremental es la **priorización de los objetivos/requisitos en función del valor que aportan al cliente**.
- **Recursivo**: espiral. No hacen entregas parciales como el ↑.
    - Hace énfasis en la gestión de riesgos.
    - 5 pasos en cada iteración: Define objetivos, evalúa riesgos, construye un entregable (no funcional), entrega y verifica, planea la próxima iteración. Se basa en tomar una característica específica, te centras en esa, avanzando en los incrementos.
    - No tuvo mucho éxito en el mercado.
    - Da muchas vueltas, pero no entrega versiones intermedias del producto, sino que se genera recién al final.
    - Al cliente se le va mostrando prototipos y cosas no funcionales.
    - Tiene control → checkpoints al final de cada iteración, si el proyecto no puede hacerse por razones técnicas, por ejemplo, te das cuenta temprano.

Ciclo de vida de los productos

- Son distintos y más largos.
- Inicia con la idea de generar un producto de software y termina cuando se saca del mercado.
- En un ciclo de vida de UN producto puede haber N ciclos de vida de un proyecto.

PROYECTO:

- Unidad de gestión que permite **obtener un producto o servicio único**.
- **Medio de organización**, administrar recursos y gestionar personas.

- **Incorpora personas** a trabajar con distintos roles, los cuales están definidos teóricamente en el proceso. Responsables de hacer el trabajo que se tiene que hacer para obtener el producto.
- **Instancia de un proceso**. Tareas se ejecutan concretamente en el ámbito del proyecto.
- Usa procesos con un ciclo de vida para poder cumplir con ese objetivo. Lo primero que se define es el objetivo (describe lo que el proyecto quiere obtener), que está relacionado con el resultado. Cada proyecto genera una versión del producto que es un RELEASE.

Características:

- Orientados a objetivos: están dirigidos y guiados por objetivos, con dos características:
  - **Claro** → que no sea ambiguo, que todos interpreten lo mismo.
  - **Alcanzable** → que sea realmente realizable, se pueda hacer.
- Resultado es único: cada resultado es distinto del anterior y del siguiente, aún que sean del mismo producto. Define el trabajo que se tiene que hacer para satisfacer las necesidades del cliente.
- Fecha de inicio y fin bien identificadas: no es algo constante, empieza y termina. Cuando se termina, se reasignan los recursos a otros proyectos. Son temporarios, se alcanzan los objetivos y terminan.
- Tareas interrelacionadas: manera en que se alcanza el objetivo. Se divide el trabajo en tareas, algunas tienen que estar relacionadas, otras no, armar conjunto según el proceso. Elaboración gradual: el todo se va a dividir en partes, es la que permite cumplir con el objetivo.

Proyecto planificado a veces fracasa → imaginar lo no planificado.

Importante de la planificación → acto de planificar, no lo planificado. Ponerse en el acto de pensar el objetivo, alcance, riesgos, recursos, estimación.

Decisiones queden escritas en algún lado → más fácil de comprender.

Cómo está relacionado el objetivo del proyecto con el del producto.

- Producto: gestionar las notas de los estudiantes. Alcance: registrar alumnos, comisiones, notas.
- Proyecto: desarrollar un producto de software que gestione las notas de los estudiantes. Alcance: va el proceso, porque es el trabajo q tengo q hacer.

Cronograma: hace énfasis en la gestión de las tareas y del tiempo.

GESTIÓN de PROYECTO → PROJECT MANAGEMENT

- Se quiere que el proyecto cumpla con el objetivo exitosamente: administrar recursos, organizar el trabajo, seguimiento si las cosas se están dando como se dijo.
- Gente, recursos, objetivo.
- Jefe/Líder de proyecto → responsable
- Dos actividades: planificación y seguimiento y control.
- De ahí aparecen distintos enfoques de planear el proyecto.

Para poder lograr un objetivo, el LP debe lograr TRIPLE RESTRICCIÓN. Para poder cumplir con el trabajo → lograr equilibrio en los tres pilares:

- Alcance: requerimientos que el cliente expresó. Es lo que no se puede cambiar, porque es del cliente. Determina los dos de abajo.
- Tiempo: cronograma, cuánto tiempo debería llevar completarlo
- Costo: cuánto debería costar, personas, recursos críticos.
- Balance → afecta CALIDAD. Los proyectos de alta calidad entregan el producto requerido, el servicio o resultado, satisfaciendo los objetivos en el tiempo estipulado y con el presupuesto planificado.

Líder de proyecto → solo tareas de gestión, no se le asignan tareas de programación. Necesita

PLAN DE PROYECTO → mapa que define lo que vamos a hacer, cuándo, cómo, quiénes. Descripción de las interrelaciones de los recursos que serán utilizados en el proyecto.

1)Alcance: todo el trabajo y solo el trabajo que hace falta hacer para cumplir con el objetivo, son las tareas que se deben realizar para entregar el producto con todas las características y funciones especificadas.

- Todo: que no esté incompleto, no te comas nada.
- Solo: no hacer de más.
- ≠ → **alcance del PRODUCTO**: características que pueden incluirse en un producto o servicio. Relación: antes tiene que estar este definido, porque si este es grande, hay que hacer más tareas para realizarlo. Está definido en la **ERS: Especificación de Requerimientos.**

2)Proceso: tareas que hay que hacer y **ciclo de vida**: cómo se van a hacer y cuánto de cada tarea.

3)Estimaciones:

- Tamaño: qué se quiere que haga el producto. En CU o LoC (líneas de código sin comentar). QUÉ.
- Esfuerzo: horas persona lineales ideales. Solo una persona, sin solapamiento, que lo va a hacer y lo va a terminar. Siempre se mide en HORAS. CÓMO. Hay que entender que cuando se traduce el esfuerzo en el calendario hay que tener en cuenta bien todo. Tener en cuenta la gestión de las dependencias.
- Tiempo: CUÁNDO. Calendario. En desarrollo de software suele ser MESES. Ver gente disponible, cuántos días a la semana se va a trabajar y cuantas horas se tienen disponibles para trabajar. Tener en cuenta Índice de solapamiento: si se pueden hacer cosas en paralelo o se necesitan terminar algunas para empezar otras.
- Costo: al último porque las decisiones que se toman en términos del tiempo pueden impactar en el costo. Por ejemplo, hacer trabajar horas extra, traer más gente. CUÁNTO.
- Recursos críticos: cosas que hacen falta más allá de lo normal, para que no generen desfasajes. Por ejemplo, alarmas para incendios con sensores, licencias de software, herramientas. Más o menos ver cuándo se necesitan, cuánto salen.

4)Gestión de riesgos

Riesgo también se estima: porque son cosas malas que pueden llegar a suceder probabilidad de ocurrencia de algo que puede impactar negativamente en mi proyecto, pérdida o daño, y dependiendo de lo grave que sea puede ser que incluso haga fracasar.

Tiene un impacto → consecuencia que tengo que pagar en mi proyecto por si eso ocurre.

No se puede destinar tiempo y recursos a cada uno de los riesgos, por eso ↓

Se mide por: probabilidad de ocurrencia e impacto. Multiplico la PxI = exposición al riesgo → valor numérico que permite comparar un riesgo con otro.

Cuando el riesgo se hace realidad → PROBLEMA.

Probabilidad de que salga bien o no → lo que tiene en común el riesgo y la estimación.

Lo que impacta en las estimaciones es la probabilidad del riesgo.

Gestionar los riesgos → tratar de bajar o la probabilidad de ocurrencia, o el impacto, o ambos → disminuir la exposición. En algunos casos se puede y en otros no. Después es hacer seguimiento y control, porque como todo en un proyecto de software, cambia.

5)Asignación de recursos → equipo de trabajo que realice las actividades y tareas planteadas por el proceso, tomar los roles. Tecnologías.

6)Definición de métricas → definición cuantitativa que permite dar cuenta si el proyecto está en línea con lo que se planificó, o bien se está desviando. Se mide la realidad y se compara → monitoreo y control del proyecto.

Métricas se % en función del dominio:

- Métricas de proceso: saber en términos de organización cómo se está trabajando, independientemente del proyecto. Ej. porcentaje de proyectos que se terminan en término.
- Métricas de proyecto: permiten saber si un proyecto de software se está realizando según lo que se planificó. Ej. comparar con el cronograma.
- Métricas de producto: tienen relación directa con el software que se está realizando. Ej. tamaño.

Métricas básicas para un proyecto de software:

- Tamaño del producto: producto.
- Esfuerzo: proceso.
- Tiempo (calendario): proyecto.
- Defectos: producto.

7)Monitoreo y control

Plan de proyecto y métricas. Proyecto se atrasa de un día a la vez → si en lugar de esperar a que el proyecto se haya desviado, se va realizando un seguimiento, se puede corregir: tipo de controles, informes, reuniones para ver el avance del proyecto.

Lo que transita por el tablero son las US → que son **producto**.

Adaptación → instancia. Un proceso se instancia para un proyecto. En el proyecto se define qué se hace del proceso y qué no → tayloring?.

Hay veces en que el costo del esfuerzo no es relevante.

Nunca es lineal la traducción del esfuerzo.

Cronograma → necesita las tareas y sus estimaciones, se realiza luego de determinar el alcance, es parte del planeamiento.

- SOFTWARE CONFIGURATION MANAGEMENT (SCM)

GESTIÓN DE CONFIGURACIÓN DE SOFTWARE -

¿Qué es? → actividad es acotado. Es una DISCIPLINA de soporte. Paraguas, protectora que va ocurriendo transversalmente a lo largo de todo el proyecto, e incluso más allá para darle soporte al producto en todo su ciclo de vida.

SCM: 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.

ÍTEM DE CONFIGURACIÓN → 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. Puede ser una foto, Word, código, prototipo. Cada uno debe tener su versión.

Visión de software más abarcativa → cualquier cosa que sale ↑

Responsabilidad: si bien existe un gestor de configuración, todo el equipo es responsable por el producto de software que se está construyendo. Cada vez que alguien genera un item es responsable de respetar los lineamientos para mantener la integridad del producto.

Propósito: establecer y mantener la integridad del producto a lo largo de todo su ciclo de vida.

Integridad:

- Satisfacer necesidades del usuario
- Puede ser fácil y completamente rastreado durante su ciclo de vida
- Satisface criterios de performance

- Cumple con sus expectativas de costo

Problema → software cambia, y es fácil de modificar.

REPOSITORIO → contenedor de ítems de configuración, donde se guardan. Tiene una estructura que permite definir dónde va cada ítem. Es importante para la TRAZABILIDAD y evaluar el impacto de un cambio. Estructura es conocida y acordada entre todos los que van a trabajar en un determinado proyecto implica que se pueda encontrar más fácilmente las cosas, además ayuda a la seguridad (ver quiénes pueden acceder), controles de acceso, políticas de seguridad, evitar tener las cosas de manera local, evitar que se modifique el software fácilmente.

VERSIÓN → estado que tiene que ver con el tiempo de cada ítem de configuración, se define, desde el punto de vista de la evolución, como la forma particular de un artefacto en un instante o contexto dado.

LÍNEA BASE → puede contener en un momento de tiempo un solo ítem.

- Especificación y/o producto, conjunto de ítems estable, que se los puede tomar como referencia, han pasado formalmente por niveles de revisión y aprobación, y que para cambiarlos deben pasar por un proceso formal de control de cambios. Conjunto de ítems se marcan para identificar la línea base: Nombre, versión, identificación única.
- Objetivo: poder volver atrás y asegurarse que la línea tenía ítems confiables.
- Dos tipos:
    - De especificación (Requerimientos, Diseño), no requieren código.
    - De productos que han pasado por un control de calidad definido previamente

RAMAS → bifurcación del desarrollo, pueden ser descartadas o integradas, y justamente sirven para colaborar al hecho de que el software se puede cambiar fácilmente. Entonces, cuando ya se prueba que lo que se hizo en la rama funciona, está aceptado, ahí recién se hace un merge a la rama principal.

4 funciones que integran la SCM (secciones de un plan de configuración):

- **Identificación de ítems**: definir estructura del repositorio, definición de ítems, y asignación de estos a un lugar específico dentro del primero. Definir reglas de nombrado, porque algunos no se saben cómo se van a llamar al principio.
    - De acuerdo con el ciclo de vida de cada ítem se clasifican en 3:
        - Producto: ERS, arquitectura, código, manual de usuario. Necesitan mantenimiento y control porque trasciende los otros.
        - Proyecto: plan de proyecto, cronograma.
        - Iteración: plan de iteración, reporte de defectos. Es el que dura menos, porque cuando termina, no es relevante mantenerlos, ya no evolucionan más.
- **Control de cambios**: mantiene integridad de las líneas base y evalúa impactos del cambio para su aceptación o rechazo. Comité de control de cambios: referentes del equipo que está trabajando en el desarrollo del producto, representantes de los roles que aportan diferentes visiones sobre el cambio y su naturaleza. Básicos: arquitecto, desarrollador, líder del proyecto, alguien de requerimientos. Objetivos: que todos los que se tengan que enterar de un cambio en la línea base, se enteren.
- **Auditorías de configuración de software**: la hace un auditor, alguien externo al proyecto porque debe ser algo objetivo. Es un proceso de control, debe tener un plan para comparar contra él. Si no tiene línea base no tiene QUÉ controlar. Se hacen 2:
    - **Física** → vela por la integridad del repositorio, ítems respeten nombrado y lugar. Se audita una línea base. Primero se hace esta. Verifica.
    - **Funcional** → asegura si el producto es el producto correcto. Si los ítems responden a lo que los requerimientos dicen lo que tienen que hacer. Items consistentes con la especificación de requerimientos. Valida.

Sirven para dos procesos básicos:

- **Validación:** el problema es resuelto de manera apropiada que el usuario obtenga el producto correcto.
- **Verificación:** asegura que un producto cumple con los objetivos preestablecidos, definidos en la documentación de líneas base (línea base). Todas la 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.

- **Informes de estado**: generar reportes para dar visibilidad, aseguran que la información de los cambios llega a todos los involucrados, enterar de un estado de situación.

Plan debe contener respuestas a las preguntas de cómo se van a realizar las 4 tareas de la SCM:

- Debe contener sección que identifique los ítems de configuración a ser administrados.
- Es requerido para realizar una auditoría.
- Es un ítem de configuración.