

Unidad 1 completa (incluye paper No silver bullet)

- 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

- Gestión de Producto
- Requerimientos Ágiles
- User Stories
- Estimaciones Ágiles
- Framework Scrum

Unidad 3

- SCM

CON * LO QUE NO ESTÁ EN EL RESUMEN LILA

U1	1
Introducción a la Ing del Sw. ¿Qué es?	1
*Estado actual y antecedentes. La crisis del Sw	1
*Disciplinas que conforman la Ing de Sw	2
Problemas al construir SW (+ *ejs de grandes proy de sw fallidos y exitosos)	2
Ciclos de vida (Modelos de Proceso) - Adm de Proy de Sw y Proc	4
* Ventajas y desventajas de c/u de los ciclos de vida./Escenarios donde elegirlo	5
Componentes de un Proyecto de Sistemas de Información	5
Gestión tradicional de un proyecto	6
U2	8
Filosofía Ágil	8
Requerimientos ágiles	9
User stories	10
Estimaciones de sw	11
Gestión de productos	13
*Framework Scrum	15
U3	17
Software Configuration Management (SCM)	17
Paper No Silver Bullets pedilo	21

U1

Introducción a la Ing del Sw. ¿Qué es?

El **software** es un set de programas y la documentación que lo acompaña (no es solo código).

*La **ingeniería de software** es una disciplina que se encarga de todos los aspectos de la producción del sw. Desde la primera etapa de especificación del sistema hasta el mantenimiento del sist después de que se pone en operación. Nace tras la crisis del sw.

Tipos básicos de Sw:

- **System Software:** Controla y gestiona los recursos del hw. Interfaz.
- **Utilitarios:** subconjunto de Syst. Sw. Proporciona herramientas para mej y optimizar rendimiento.
- **Sw de aplicación:** *Para hacer tareas específicas en una compu. Se enfoca en satisfacer las necesidades de los usuarios.*

No comparar SW y manufactura (5)

- **Sw es menos predecible** que los productos en una línea de producción
- Casi **ningún prod de sw es igual a otro**. (reqs son distintas)
- **No todas las fallas en sw son errores**.
- El **sw no se gasta**. **Se debe ir adaptando** a los cambios y necesidades de user/org
- Sw **no gobernado por las leyes de la física**

*Estado actual y antecedentes. La crisis del Sw

Actualmente muchos sistemas desarrollados fracasan debido a fallas en el software, en consecuencia de :

- **Demandas crecientes:** a medida que tecnología y técnicas de desarrollo evolucionan, la demanda de sistemas más grandes y complejos aumenta.
 - Usuarios necesitan rápido que se construya y distribuya
 - Los métodos existentes (tradicionales) no permiten lograr esa "espontaneidad". Se deben desarrollar y probar nuevas técnicas.
- **Bajas expectativas:** muchas empresas de desarrollo actuales no usan métodos de ingeniería de software en su trabajo. Su sw termina siendo más costoso y menos confiable. Las empresas se conforman con que el sw solo "funcione". Sin tener en cuenta el valor agregado que el sw le tiene que aportar al negocio del usuario

Antes, en 1968 (*en una conferencia de otan zzz*), se presentaron hechos relativos al sw, a la **crisis del sw**. Se recalcó la dificultad para generar sw libre de defectos, fácilmente comprensibles y que sean verificables. Las causas son:

- La **evolución del hw** permitió que se desarrollen sistemas más grandes y complejos. Pero el salto de hw no fue acompañado de un salto de sw. Es porque el sw es una actividad humana, cuyo producto es abstracto e intangible
- Esto, junto con la **demanda creciente** de estos sistemas, la **subestimación de la complejidad del desarrollo** de sw, los **cambios para adaptarse** a las necesidades del cliente y la **falta de una disciplina** que intervenga en todos los aspectos de la producción del sw.

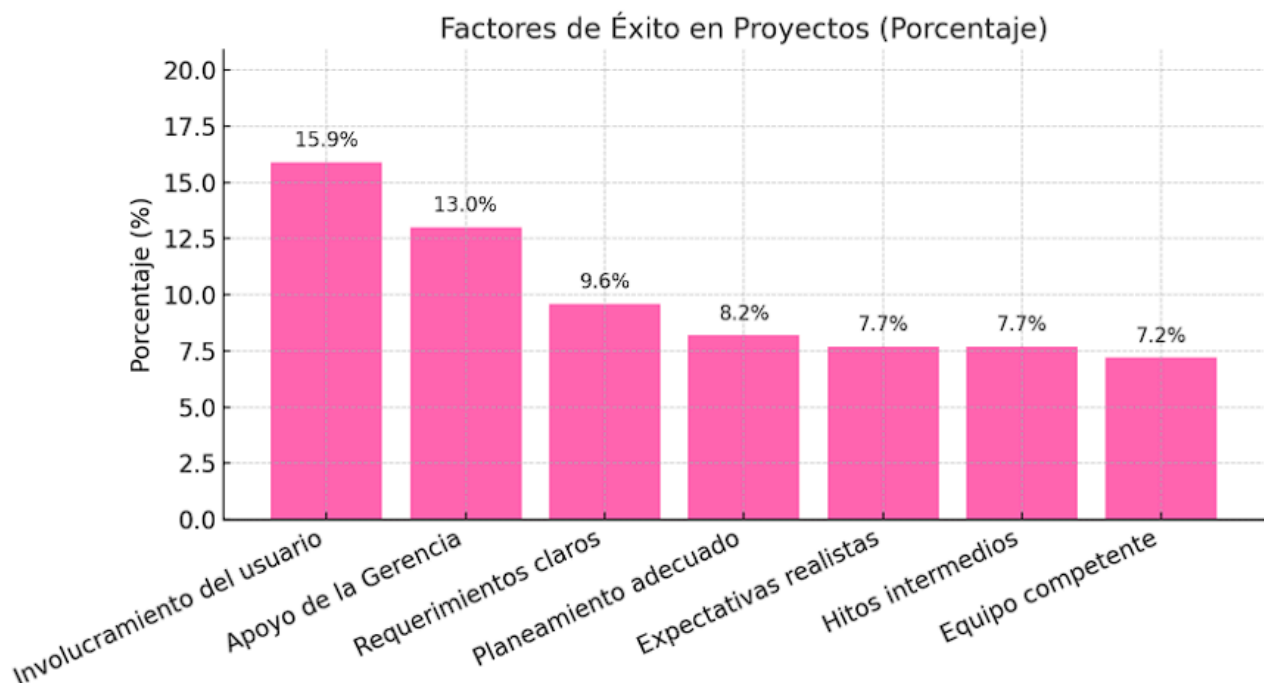
*Disciplinas que conforman la Ing de Sw

- **Disciplinas técnicas:** actividades que aportan al desarrollo de sw como *producto*.
 - Requerimientos
 - Análisis y diseño
 - Construcción
 - Prueba
 - Despliegue
- **Disciplinas de gestión:**
 - Planificación de proyecto
 - Monitoreo y control de proyectos
- **Disciplinas de software:** disciplinas transversales a los procesos de sw. Permiten verificar la integridad y calidad de un producto de sw.
 - Gestión de configuración de software (SCM)
 - Aseguramiento de Calidad (QA)
 - Métricas

Problemas al construir SW (+ *ejes de grandes proy de sw fallidos y exitosos)

- Más tiempos y costos que los presupuestados
- Que la versión final no satisfaga las necesidades del cliente
- Problema en la escalabilidad Y/O adaptabilidad de sw
- Mala documentación
- Mala calidad del sw (no relacionado al testing)

Software exitoso

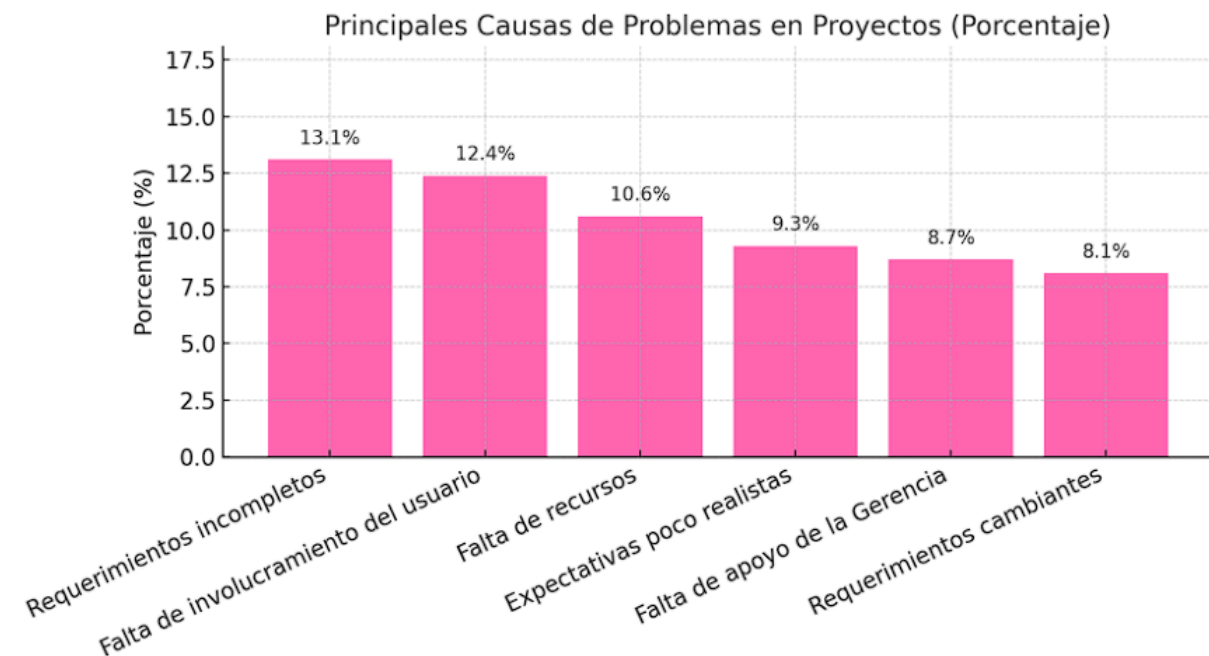


Se busca que estos factores de éxito estén embebidos en el proceso de desarrollo. Uno de los aspectos claves son los **requerimientos**, tenerlos claros e incentivar a que el usuario tenga participación para esclarecerlos. Es necesario que los requerimientos vayan “madurando” a

medida que avanzamos en el desarrollo del sistema. También debemos considerar la **retroalimentación**, es importante medir el avance del sistema y contar con info para corregir fallas.

El manifiesto ágil y metodologías como Scrum nos hablan sobre equipos capacitados y con habilidades cuya participación dentro del desarrollo del sistema sea competente.

Software no exitoso

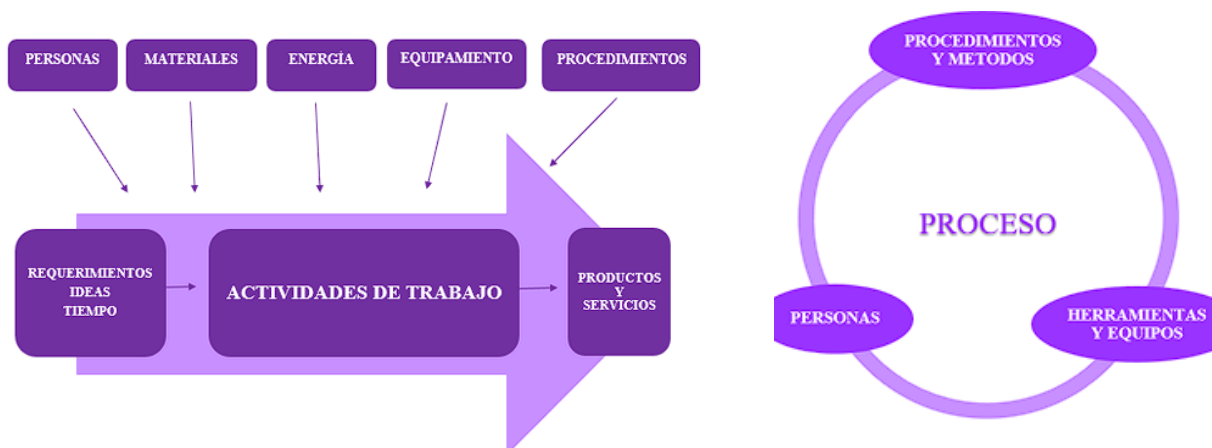


PARA LOS EJEMPLOS, VER MI GPT !!! LA ÚLTIMA CONVERSACIÓN!!

Procesos de Desarrollo Empíricos vs. Definidos.

El **proceso de Software** es el conjunto estructurado de actividades que a partir de un conjunto de entradas, producen una salida.

- Las acts dependen de la organización y el tipo de sist a desarrollar.
- Los inputs: requerimientos (definen alcance del prod), personas (el principal capital que se consume), materiales, energía, equipos, hw, etc



La definición de CMM: 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.

Las personas hacen uso de las herramientas, equipos, procedimientos y métodos y a partir de esos producen un producto de sw.

Proceso definido

Basados en modelo industrial (línea de producción), donde hay un conjunto de entradas utilizados en un conjunto de actividades que van a producir la misma salida siempre que los inputs sean los mismos.

Software no es tan definible. Es difícil aplicar este concepto.

La administración y el control provienen de la predictibilidad del proceso definido. (**PUD**)

Punto de mejora es el central (actividades, porque entrada y salida son fijas)

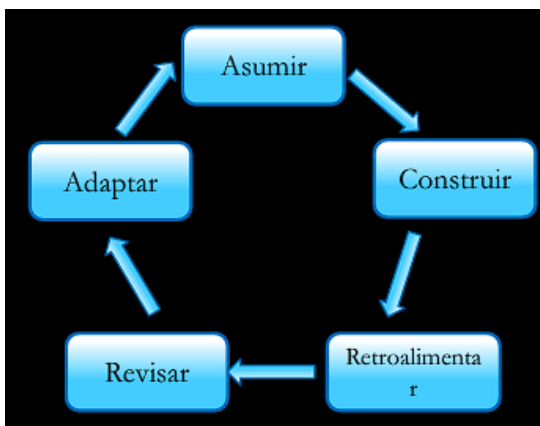
Proceso empírico

Se basa en capitalizar la experiencia de los actores involucrados en la producción para maximizar la calidad de lo que se está produciendo.

Las variables no pueden ser controladas en su totalidad. Implica la necesidad de **adaptación** para poder garantizar la producción de un sw de calidad.

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

No tiene punto de mejora tangible como los definidos.



Patrón de conocimiento en procesos empíricos

Se empieza con una hipótesis (asumir). Se construye, se hace retrospectiva de algunos puntos (retroalimentar), se revisa el resultado de la construcción comparándolo con la hipótesis inicial. Se adapta el proceso y se sigue construyendo.

CICLO DE VIDA ITERATIVO E INCREMENTAL

Ciclos de vida (Modelos de Proceso) - Adm de Proy de Sw y Proc.

El **ciclo de vida** es una abstracción. Define las etapas y el orden de cada una. (serie de pasos en las que un producto progresa)

El ciclo de vida de **un proyecto, empieza y termina** cuando se genera un producto.

El ciclo de vida de un **producto**, se mantiene mientras el objeto exista.

Ciclos de vida básicos

- **Secuencial**: ejecutar una etapa después de la otra sin retorno normalmente. Algunas devuelven información (como cascada)
- **Iterativo**: procesos empíricos la implementan. Iteraciones para ganar info y convertirla en un incremento. (iterativo/incremental)
- **Recursivo**: Para casos particulares, como proyectos de alto riesgo. (espiral). Se toma una característica específica y en una iteración me centro en esa sola funcionalidad. EN DESUSO POR INUTIL..

Relación con Procesos de desarrollo de sw

El **PROCESO** es una implementación del **CICLO DE VIDA** (que es una abstracción que nos guía en cuáles son las etapas y su orden) que tiene un **objetivo** que lo guía, que es la **creación de un PRODUCTO** (debe ser no ambiguo y alcanzable)

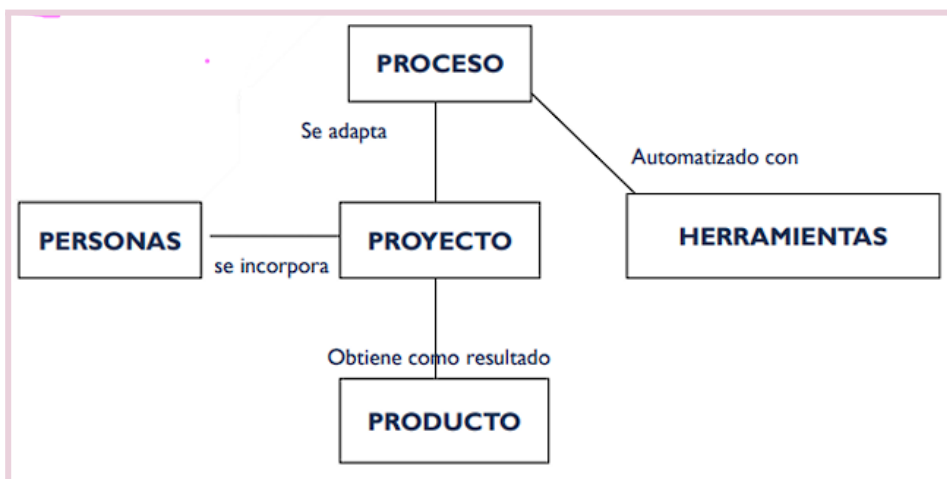
*Influencia en la Adm de Proy de Sw

?

* *Ventajas y desventajas de c/u de los ciclos de vida./Escenarios donde elegirlo*

Tipo de ciclo de Vida	Escenarios recomendados	Ventajas	Desventajas
Secuencial (cascada puro)	Requerimientos estables, entornos regulados, equipos grandes y distribuidos, plazos y costos fijos, necesidad de documentación formal	1. Planificación detallada. 2. Documentación completa. 3. Roles claros. 4. Control estricto de fases. 5. Ideal para proyectos críticos y auditables.	1. Muy rígido a cambios. 2. Entrega de producto al final. 3. Corrección de defectos costosa. 4. Dependencia de requisitos iniciales. 5. Menor motivación del equipo.
Iterativo (Iterativo-Incremental)	Requerimientos cambiantes, necesidad de entregas tempranas, productos evolutivos, entornos ágiles, prioridad de reducir riesgos tempranos	1. Feedback continuo. 2. Valor temprano al cliente. 3. Menor retrabajo. 4. Alta flexibilidad. 5. Priorización de funcionalidades	1. Planificación/documentación compleja. 2. Riesgo de desorganización. 3. Arquitectura puede degradarse. 4. Costos variables. 5. Requiere cliente activo.
Recursoivo (espiral)	Proyectos grandes y complejos, alta incertidumbre, control de riesgos crítico, necesidad de prototipos, adopción de nuevas tecnologías, máxima calidad y seguridad.	1. Gestión de riesgos en cada ciclo. 2. Prototipos sucesivos. 3. Flexibilidad total. 4. Comunicación constante con cliente. 5. Adopción controlada de tecnología.	1. Costos altos. 2. Planificación difícil. 3. Difícil estimar fin del proyecto. 4. Requiere personal experto. 5. Riesgo de pérdida de trazabilidad si no se documenta bien

Componentes de un Proyecto de Sistemas de Información.



Un **PROYECTO** es llevado a cabo por **PERSONAS** que implementan **HERRAMIENTAS** para automatizar los **PROCESOS** y que obtiene como resultado un **PRODUCTO**.

Proceso: plantilla. Toman como entrada requerimientos y a través de un conjunto de actividades obtienen como salida un producto o servicio de software, esas actividades están estructuradas y guiadas por un objetivo.

Proyecto: donde administro los recursos que necesito y las personas que van a formar parte del mismo, para obtener como resultado un producto de software. Guiados

Características de proyecto de sw:

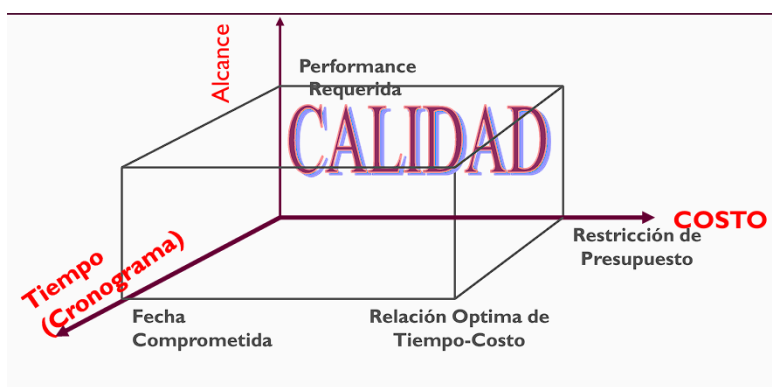
- **Duración limitada:** termina cuando alcanza el/los obj's
- **Orientado a un objetivo:** obj debe ser claro, no ambiguo y alcanzable
- **Son únicos**
- **Tiene tareas relacionadas entre sí basadas en esfuerzos y recursos para alcanzar el obj:** dependen una de otra. hace que la gestión de proyectos sea compleja.

Gestión tradicional de un proyecto

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

La **administración de proyectos** es la aplicación de conocimientos, habilidades, herramientas y técnicas a las actividades del proyecto para satisfacer sus requerimientos. Incluye identificar los requerimientos, establecer obj's claros y alcanzables, y adaptar las especificaciones, planes y el enfoque a los diferentes intereses de los involucrados

La triple restricción



Alcance: requerimientos del cliente

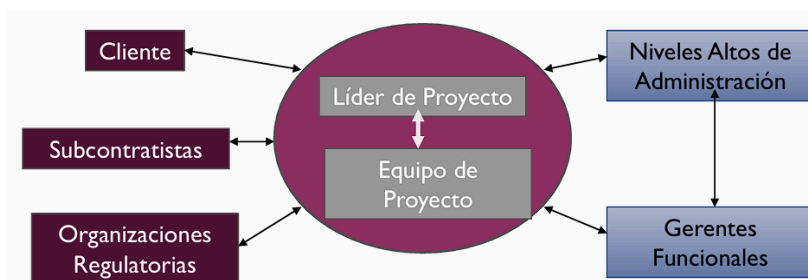
Tiempo: que lleva completar proyecto

Costo: en recursos y dinero

Es una de las bases de la gestión tradicional de proyectos. Es responsabilidad del **PM** balancear los tres objetivos. El balance de estas variables afecta directamente a la calidad del proyecto.

La calidad del producto no es negociable. Un proyecto de alta calidad **entrega el producto requerido, satisfaciendo los objetivos en el tiempo estipulado con el presupuesto planificado.**

Roles en un proyecto



PM: líder del proyecto. Se ocupa de todas las tareas de gestión. Administra todos los recursos, organiza el trabajo de las personas involucradas y hace el seguimiento de la planificación verificando que vaya como planeado.

No es necesario en procesos empíricos, porque los proyectos se autogestionan.

Equipo: grupo de personas. Se deben poder complementar. Distintos conocimientos y habilidades.

Plan de proyecto

Como hoja de ruta. Esboza lo que va a hacer el proyecto. Guía. Vivo a lo largo del proyecto. En permanente actualización (si no, no sirve). Se documenta:

- **QUÉ** hacemos → **Alcance** del proyecto
- **CUÁNDO** hacemos → **Calendarización**
- **COMO** hacemos → **Recursos** y decisiones disponibles
- **QUIEN** hacemos → Asignación de **tareas**.

Planificación de proyectos de sw.

1. Definición del alcance

- Alcance del proyecto: todo el trabajo para entregar el producto. Se mide contra el *Plan de Proyecto*
- Alcance del producto: todas las características que se pueden incluir en un prod. Se mide contra la *Especificación de Requerimientos* (Ej: CU)

Primero definir el **alcance del producto**, con ese puedo definir el del **proyecto**. (tareas dependen de lo que hay que hacer)

2. Definición de proceso y ciclo de vida

Proceso de desarrollo es el cpto de acts que tengo que hacer para construir el producto de sw.

Ciclo de vida es de qué manera ejecuto esas actividades.

3. Estimación: No tenemos certeza de nada. Estimamos las características necesarias para la planificación. Con rangos.

- **Tamaño**: del prod a construir
- **Esfuerzo**: hora persona lineales
- **Calendario**: que días, que horas y cuantas personas van a trabajar.
- **Costo**: presupuesto necesario
- **Recursos críticos**: tanto humanos como físicos.

4. Gestión de Riesgos

Identificar los riesgos más **probables** de ocurrir o los que **más impacten** en el sistema. Se multiplican estas dos variables (impacto y ocurrencia) y se obtiene la **exposición de riesgo**. (permite determinar riesgos principales y cuáles gestionar).

5. Asignación de recursos

6. Programación de proyectos

7. Definición de métricas: Métricas nos permiten dar cuenta si proyecto está en línea con lo planificado. Si métrica no tiene fin en particular, no tiene sentido tomarla. Según su *dominio* se clasifican:

- **Métricas de proceso**: saber que pasa en términos organizacionales. Son las del proyecto pero despersonalizadas de uno en particular (% de proy que terminan en termino y % que no)
- **Métricas de proyecto**: si proyecto va según planificado (comp tiempo planif con real)
- **Métricas de producto**: miden características que tienen relación directa con el producto que estamos construyendo.

Métricas básicas: Tamaño, Esfuerzo, Tiempo, Defectos.

8. Monitoreo y control: **PM** trabaja sobre lo definido y ve si está cumpliendo lo definido en el plan de proyecto.

FACTORES PARA EL ÉXITO

1. **Monitoreo y Feedback:** tener un monitoreo permanente y generar acciones correctivas cuando sea necesario para evitar una desviación mayor.
2. **Misión/Objetivo claro:** saber hacia donde vamos
3. **Comunicación:** en todos sus aspectos, con el líder de proyecto, con el equipo, con los clientes y con los stakeholders.

CAUSAS DE FRACASOS

1. Fallar al definir el problema
2. Planificar basado en datos insuficientes
3. La planificación la hizo el grupo de planificaciones
4. No hay seguimiento del plan del proyecto
5. Plan de proyecto pobre en detalles
6. Planificación de recursos inadecuada
7. Las estimaciones se basaron en “supuestos” sin consultar datos históricos
8. Nadie estaba a cargo
9. Mala comunicación

U2

Filosofía Ágil

Manifiesto ágil: compromiso que hacen todas las personas en un proyecto para trabajar de una manera determinada independientemente de las prácticas que realice cada uno.

El obj es lograr equilibrio entre seguir **procesos rigurosos y formales** y **trabajar eficiente** y efectivamente para entregar un prod de calidad

SE APLICA EN PROCESOS EMPÍRICOS: importante ciclos de retroalimentación cortos.

Agilismo: ideología con conjunto definido de principios que guían el desarrollo del producto.

Busca encontrar un equilibrio entre procesos definidos y nada de procesos.

COMPATIBLE SOLO CON CICLOS DE VIDA ITERATIVOS

Pilares del empirismo

- **Transparencia:** comunicación abierta y honesta de la información relevante a todas las partes interesadas. *Confianza y toma de decisiones más informada*
- **Adaptación:** capacidad de adaptarse y ajustar el trabajo y el proceso para hacer frente a las desviaciones y problemas detectados durante la inspección. *Rta rápida a cambios y mantener entrega de valor a cliente*
- **Inspección:** revisión regular y sistemática del progreso del trabajo y los productos resultantes para detectar problemas y desviaciones del plan. *Tomar decisiones informadas.*

Manifiesto ágil

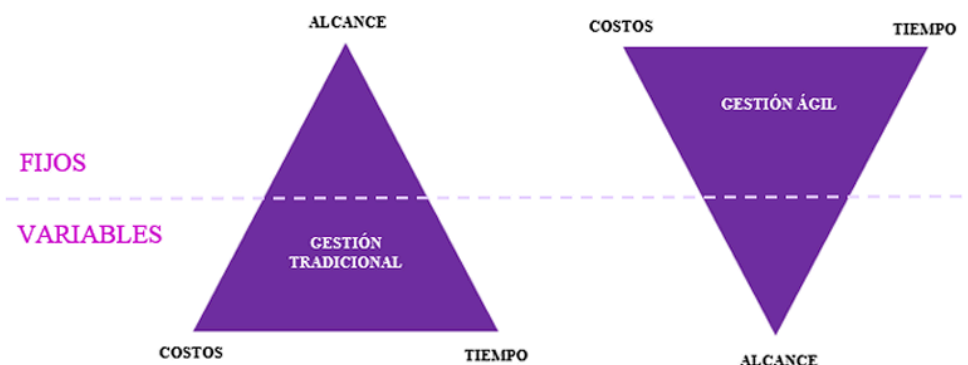
Valores (4)

1. Valorar a los individuos e interacciones por sobre procesos y herramientas
2. Sw funcionando por sobre documentación extensiva
3. Valorar más la colaboración con el cliente que la negociación de un contrato (alcance/costo/tiempo)
4. Valorar más la respuesta ante el cambio que seguir un plan

Principios

- | | |
|-----------------------------|---|
| 1. Satisfacción del cliente | 7. Sw funcionando |
| 2. Adaptación al cambio | 8. Continuidad (ritmo cte, ciclo de t fijo) |
| 3. Entregas frecuentes | 9. Excelencia técnica |
| 4. Trabajo en equipo | 10. Simplicidad |
| 5. Personas motivadas | 11. Equipos Autoorganizados |
| 6. Comunicación directa | 12. Mejora continua |

Triángulo de Hierro vs Triángulo ágil



Gestión tradicional:

- Los requerimientos son fijos. Iteraciones en base a porciones de funcionalidad (CU)

Gestión ágil

- busca maximizar valor al cliente, asegurar la calidad del producto y cumplir tiempo establecido.

Requerimientos ágiles

Se expresan como **historias de usuarios**, descripciones de requisitos en idioma de usuario. Usadas para definir el alcance y se priorizan en función del valor que aportan al producto. El **PO** es el responsable de identificar las necesidades y prioridades del negocio, priorizar los requerimientos del producto. Estos requerimientos se organizan en lista **Product Backlog**. A medida que se trabaja en esos, puede haber req movidos, o removidos, o agregados.

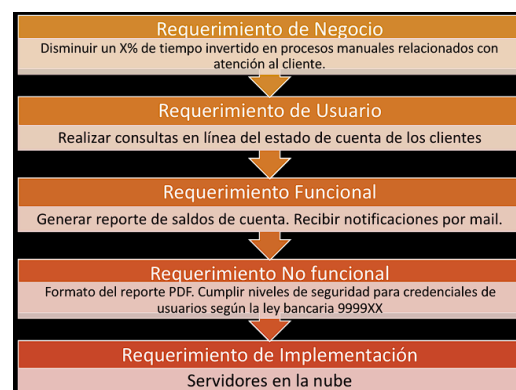
Just in Time: Concepto para evitar desperdicios. Análisis cuando lo necesito, no antes ni después

METODOLOGIA TRADICIONAL vs METODOLOGÍA ÁGIL

	TRADICIONAL	ÁGIL
Prioridad	Cumplir el plan	Entregar Valor
Enfoque	Ejecución ¿Cómo?	Estrategia (¿Por qué? ¿Para qué?)
Definición	Detallados y cerrados. Descubrimientos al inicio	Esbozados y evolutivos. Descubrimiento progresivo
Participación	Sponsors, stakeholder de mayor poder e interés.	Colaborativo con stakeholders de mayor interés (clientes, usuarios finales)
Equipo	Analista de negocios, Project Manager y Áreas de Proceso.	Equipo multidisciplinario.
Herramientas	Entrevistas, observación y formularios	Principalmente prototipado. Técnicas de facilitación para descubrir.
Documentación	Alto nivel de detalle. Matriz de Rastreabilidad para los requerimientos.	Historias de Usuario. Mapeo de historias (Story Mapping)
Productos	Definidos en alcance	Identificados progresivamente
Procesos	Estables, adversos al cambio.	Incertidumbre, abierto al cambio

Tipos de requerimientos

- **De negocio:** dom de problema
- **De usuario:** dom de problema
- **Funcional:** sol.
- **No funcional:** sol. los que el cliente no suele tener en claro
- **De implementación:** sol. Restricciones,



conceptos relacionados a principios ágiles que se aplican:

- Cambios son constantes
- Tener mirada de todos los involucrados (sh)
- “Usuario dice lo que quiere cuando recibe lo que pidió”
- No hay técnicas ni herramientas que sirvan para todos los casos
- Importante es entregar valor, no una salida.

Principios ágiles relacionados a los Req ágiles

1. Prioridad es **satisfacer al cliente** a través de releases tempranos y frecuentes (2 sem a 1 mes)
2. Recibir cambios de requerimientos, aun en etapas finales. **Adapt cambio**
4. Técnicos y no técnicos trabajando juntos todo el proyecto. **Trabajo en equipo**
6. El medio de comunicación por excelencia es cara a cara. **Comunicación directa**
11. Las mejores arquitecturas, diseños y requerimientos emergen de **equipos auto organizados**

User stories

Técnica para capturar requerimientos. Necesita poco o nulo mantenimiento y puede descartarse después de la implementación.

Las 3 “C” de una US

- **Conversation**: no queda escrita en ningún lugar. Es la parte no visible. es donde se obtienen todos los detalles para que el equipo pueda trabajar esa historia.
- **Confirmation**: Condiciones que se tienen que dar para dar por satisfecha la US. Esta dentro de la Card.
- **Card**: parte visible. Donde escribimos algo que nos indique cual es la US, de qué se trata y su valor del negocio.

US son multipropósito. Me plantea:

- Una necesidad del usuario
- Una descripción del producto
- Un ítem de planificación (al planificar las US)
- Token para una conversación (especificar los detalles cuando llega momento de trabajarla)
- Mecanismo para diferir una conversación

US está planteada como **implementación en porciones verticales**. Implementarla desde la lógica de interfaz de usuario hasta como resuelvo la lógica en la BD

Modelado de roles

Criterios de aceptación

Pruebas de aceptación: Lo que esperamos que suceda cuando ejecutamos la US

Criterio de ready: para definir que tiene que cumplir la US para que esté lista para ser implementada. (incluirla al Sprint). Lo ayuda a definir el modelo **INVEST**

I: Independiente:

N: Negociable:

V: Valuable:

E: Estimable: poder definir cuanto esfuerzo

S: Small: entrar en una sprint

T: Testable:

Definición de Done: para determinar si US está decentemente terminada para presentársela al PO/Cliente. Debe tener las pruebas unitarias y de aceptación en verde

Niveles de abstracción

- **Épicas:** US muy grandes que no fueran detalladas
- **Temas:** Conjunto de US relacionadas. Más grande que las épicas, ya que todavía no ha llegado el momento de detallarlas y no solo son ideas concretas
- **Spikes:** sirven para quitar riesgo o incertidumbre de otro requerimiento.
 - **Técnicas:** para investigar enfoques técnicos en el dominio de la solución.
 - **Funcionales:** hay cierta incertidumbre respecto de cómo el usuario interactuará con el sistema

Lineamientos para Spikes

- *estimables, demostrables y aceptables.* Deben cumplir con criterio de Ready.
- Es excepción, última opción.
- Implementar la spike en una iteración separada de las historias resultantes

Estimaciones de sw

Predecir en un momento donde no tenemos gran conocimiento, donde hay incertidumbre. NO es planear. Se va refinando a medida que el proyecto se va desarrollando

Razones por las que **nos equivocamos al estimar**:

- si el proyecto no está organizado, estimaciones son imprecisas}
- no tenemos en claro cuáles son los recursos disponibles
- si resultados muy distintos utilizando diferentes técnicas, tengo errores.

Lo primero que estimamos es el **TAMAÑO DEL SOFTWARE**. □ Técnica de **contar**: funcionalidades, requerimientos, cant de US, cant de CU.

Luego el **ESFUERZO**. □ Técnica de **contar** horas lineales de desarrollo

CALENDARIO. □

COSTOS y PRESUPUESTOS. □

Técnicas de estimación

clasificación de MÉTODOS BASADOS:

- **en la experiencia:** para determinar la estimación de esfuerzo y tiempo requerido
- **exclusivamente en recursos**
- **exclusivamente en el mercado:** costo de proyectos similares en el mercado para determinar la estimación de esfuerzo y tiempo para un nuevo proyecto.
- **en los componentes del prod. o en el proc. de desarrollo:** para estimar el esfuerzo y tiempo requerido para cada componente. Luego, se suman para obtener una estimación total del proyecto.
- **métodos algoritmos:** modelos matemáticos y estadísticos

Dentro de los **métodos basados en la exp**, la técnicas de estimación:

- **Datos históricos:** comparar un proyecto nuevo con uno anterior. El tamaño, el esfuerzo, el tiempo y los defectos. El problema es que los dominios pueden ser muy distintos
- **Juicio experto:** se basa en la experiencia y conocimiento de expertos, que proporcionen estimaciones basadas en su experiencia y conocimiento. Se basa en la subjetividad

- ◆ **Puro**: Un experto estudia las especificaciones y hace su estimación. Desventaja → Si desaparece el experto, la empresa deja de estimar
 - ◆ **Delphi**: Un grupo de personas. Las estimaciones individuales se recopilan de forma anónima. Desventaja → requiere más tiempo y recursos
- **Analogía**: se basa en datos históricos pero se seleccionan específicamente aquellos proyectos que son similares al proyecto actual para realizar la comparación y estimación.

Actividades omitidas en estimaciones:

- Requerimientos faltantes
- Actividades de desarrollo faltantes (documentación técnica, participación en revisiones, creación de datos para el testing, mantenimiento de producto en previas versiones...)
- Actividades generales (días de enfermedad, licencias, cursos, reuniones de compañía...)

Estimaciones en ambientes ágiles



Si las estimaciones se utilizan como compromisos son muy peligrosas y perjudiciales para cualquier organización.



Lo más beneficioso en las estimaciones es el “proceso de hacerlas”.



La estimación podría servir como una gran respuesta temprana sobre si el trabajo planificado es factible o no.



La estimación puede servir como una gran protección para el equipo.

En estimaciones ágiles puntualmente, se va a trabajar con las **features o stories**. Usando una medida de tamaño relativo conocida como **STORY POINTS (SP)**, cuyo peso es la combinación de su **incertidumbre, esfuerzo y complejidad**.

Medidas relativas. No absolutas por ser específicas de cada equipo

Velocity

Métrica del progreso de un equipo. Se calcula a partir de la suma de los Story Points asignados a cada US que el equipo completó al 100% durante la iteración que se está midiendo. También podemos derivar la duración de un proyecto.

Posibles métodos de estimación

Poker estimation: Combina el *juicio de experto* con *analogía* y utiliza algo de Wideband *Delphi*. Se basa en la Serie de Fibonacci. Si SP grande, no cumple con criterio de ready.

La **canónica**, debe tener complejidad 1 y si se puntúa una con 0 es que hay desconocimiento sobre la misma y es necesario detallarla.

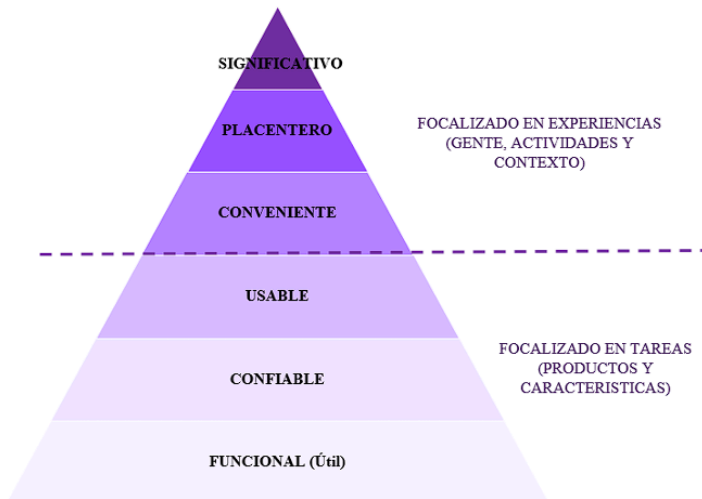
Los participantes de **poker planning** son devs. Rondas de estimación hasta que se llegue a un consenso en cuanto al valor de Story Points para cada User Story.

en las estimaciones también tenemos que cumplir con las características y **pilares del empirismo**, que son **ejecutar, inspeccionar y adaptar, retroalimentación**.

Gestión de productos

Creamos prods para lograr la satisfacción de los clientes, la obtención de dinero . y a visión de cambiar el mundo.

Evolución de los prod de sw



En la **base** las características del sw que tienen que existir para que sw funcione.

En el **tope**, las que son difíciles de alcanzar

El **desafío** tiene que ver con cómo hacemos para **construir un producto de software** donde **desechamos el desperdicio** y logramos abarcar los aspectos que estamos persiguiendo cuando creamos el producto de software.

UVP (User value Proposition): técnica de desarrollo de productos. Se centra en comprender las necesidades, deseos y problemas de los usuarios y en crear soluciones que satisfagan esas necesidades de manera efectiva. Planteamos una hipótesis y la validamos con un mínimo esfuerzo.

MVP (Minimum Viable Product):

- objetivo es comprobar que la hipótesis del producto a construir funciona
- APRENDER, no vender
- Si los feedbacks son distintos, no está claro el mercado en el que estoy.
- Hipótesis se puede mover según exigencia del mercado. Se hace nuevo MVP (MVP2)

MMF (Minimum Marketable Feature):

- Cuando ya se encontró el MVP exitoso
- definir los elementos mínimos que deben estar en el prod para que sea **comercializable**. VENDER
- Parte de un MMR o MMP
- Feature: más chico que MVP.

MVF (Minimum Viable Feature):

- Combinación MVP y MMF
- característica mínima que se puede construir e implementar rápidamente, utilizando recursos mínimos, para probar la utilidad de la misma.
- Para **validar la característica que hace la diferencia**.
- Si es exitosa, se pueden hacer más **MMF** en esa área (porque se comprobó)
- versión mini de MVP

MRF (Minimum Release Feature):

- aja

MMP: (Minimum Marketable Product):

- Primer release de MMP dirigido a primeros usuarios.
- Focused en características clave que satisface a ese grupo

MMR: (Minimum Marketable Release):

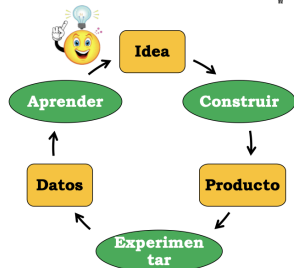
- Release de producto con el conjunto de características más pequeño posible.
- MMR1 = MMP

MLP: (Minimum Lovable Product):

- versión más simple de prod que genera conexión emocional en usuarios

Ciclo de Feedback

Build-Experiment-Learn Feedback Loop



permite descubrir las necesidades del cliente y alinearlas metodológicamente.

Fases de construir MVP

Cuando empezamos a construir MVP, la complejidad puede variar. La idea es tener rápidamente y con el menor esfuerzo el MVP para poder validarlo y retroalimentar.

Para minimizar el desperdicio y esfuerzo:

- En caso de duda, simplificar
- Evitar la construcción excesiva y la promesa excesiva
- Saber que cualquier trabajo adicional más allá de lo que necesita para comenzar el ciclo podría ser un desperdicio

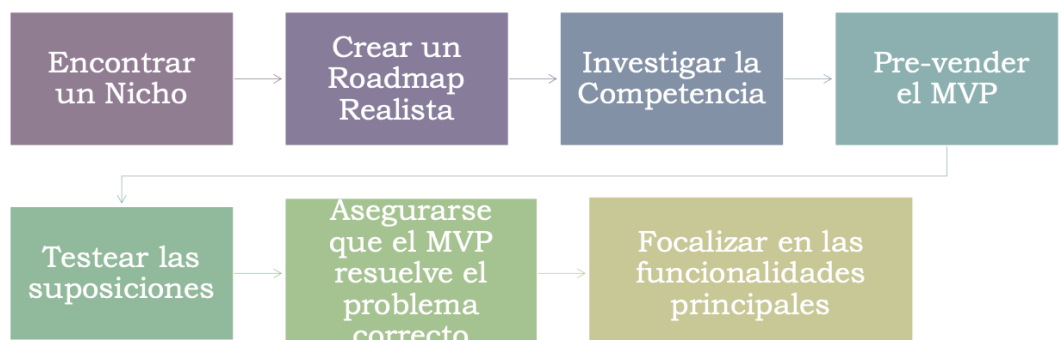
Audacia cero

El tener pocos recursos hace pensar si alguna vez vas a tener números grandes. Se aplaza el lanzamiento de cualquier versión de un producto hasta que esté seguro del éxito. Es esencial experimentar!!

Supuestos de “Saltos de Fe”

Son los elementos más riesgosos del plan. Las partes de las que todo depende.

Preparar un MVP



Características un MVP

- Diseño
- Usabilidad
- Confiabilidad
- Funcionalidad

Framework Scrum

Framework para gestión ágil de proyectos para poder generar de manera adaptativa soluciones complejas. No dice como contruir software

Resumen de la guía de scrum 2020 slds

Definición de scrum

Scrum es un marco de trabajo para generar de manera adaptativa soluciones complejas. No dice como construir software.

Requiere de un **Scrum Master** para fomentar un entorno donde

1. **PO** ordena el trabajo en un *Product Backlog*
2. **Scrum Team** convierte trabajo en un *Increment* de valor durante un *Sprint*
3. **Scrum Team** e interesados inspeccionan los rtados y se adaptan para prox *sprint*
4. Repetir.

Teoría de Scum

Se basa en el empirismo y el pensamiento Lean. Conocimiento viene de experiencia.

Enfoque iterativo e incremental para optimizar la previsibilidad y controlar el riesgo.

implementan los pilares empíricos de Scrum de **transparencia, inspección y adaptación.**

Transparencia:

Proceso y el trabajo emergentes deben ser visibles. La transparencia permite la inspección

Inspección:

detectar variaciones o problemas potencialmente indeseables. La inspección permite la adaptación. La inspección sin adaptación se considera inútil.

Adaptación:

ajuste debe realizarse lo antes posible para minimizar una mayor desviación.

Valores de Scrum

Compromiso, Foco, Franqueza, Respeto y Coraje

Scrum Team

Unidad cohesionada de profesionales enfocados en un objetivo a la vez. Generalmente 10 personas o menos. Más pequeños se comunican mejor y son más productivos.

Scrum define tres responsabilidades específicas dentro del Scrum Team: los *Developers*, el *Product Owner* y el *Scrum Master*.

Developers:

responsables de:

- Crear un plan para el Sprint, el Sprint Backlog;
- Inculcar calidad al adherirse a una Definición de Terminado;
- Adaptar su plan cada día hacia el Objetivo del Sprint; y,
- Responsabilizarse mutuamente como profesionales.

PO:

responsables de maximizar el valor del producto resultante

- Desarrollar y comunicar explícitamente el Objetivo del Producto;
- Crear y comunicar claramente los elementos del Product Backlog;
- Ordenar los elementos del Product Backlog; y,
- Asegurarse de que el Product Backlog sea transparente, visible y se entienda.

SM:

responsable de establecer Scrum como se define en la Guía de Scrum. Líderes del *Scrum Team*.

- Guiar a los miembros del equipo en ser autogestionados y multifuncionales;
- Ayudar al Scrum Team a enfocarse en crear Increments de valor que cumplan con la Def of Done;
- Procurar la eliminación de impedimentos para el progreso del Scrum Team; y,
- Asegurarse de que todos los eventos de Scrum se lleven a cabo y sean positivos, productivos y se mantengan dentro de los límites de tiempo recomendados.

Sirve al *PO*:

- *A encontrar técnicas para una definición efectiva de Objetivos del Producto y la gestión del Product Backlog;*
- *Ayudar al Scrum Team a comprender la necesidad de tener elementos del Product Backlog claros y concisos;*
- *Ayudar a establecer una planificación empírica de productos para un entorno complejo; y,*
- *Facilitar la colaboración de los interesados según se solicite o necesite.*

Sirve a la organización a liderar, capacitar y guiarla en su adopción de Scrum

Eventos de Scrum

Lo óptimo es que todos los eventos se celebren al mismo tiempo y en el mismo lugar para reducir la complejidad.

Sprint

Son el corazón de Scrum. Son eventos de duración fija. Todo el trabajo necesario para lograr el Objetivo del Producto, ocurre dentro de los Sprints. (NO se hacen cambios que pongan en peligro el objetivo) (2sem a 1 mes)

Cuando horizonte de Sprint es muy largo, el Objetivo se puede volver inválido

Sprint Planning

Establecer el trabajo que se realizará para el Sprint. Se seleccionan los elementos del *Product Backlog* para incluirlos en el *Sprint* actual. (máx 8hs)

Para cada elemento del backlog, los Devs planifican el trabajo necesario para crear un *Increment* que cumpla con la definición de Done.

El Objetivo del Sprint, los elementos del Product Backlog seleccionados para el Sprint, más el plan para entregarlos se denominan juntos Sprint Backlog.

Daily Scrum

El propósito es inspeccionar el progreso y adaptar el Sprint Backlog según sea necesario. (15 mins). Mejoran la comunicación, identifican impedimentos, promueven la toma rápida de decisiones y, en consecuencia, eliminan la necesidad de otras reuniones.

Sprint Review

El propósito es inspeccionar el resultado del Sprint y determinar futuras adaptaciones. Revisan lo que se logró en el Sprint y lo que ha cambiado en su entorno (máx 4hs)

Sprint Retrospective

El propósito es planificar formas de aumentar la calidad y la efectividad. Inspecciona cómo fue el último Sprint con respecto a las personas, las interacciones, los procesos, las herramientas y su Definición de Terminado. Concluye el Sprint (máx 3hs)

Artefactos de Scrum

Cada uno tiene un compromiso

- Product Backlog → Obj del Prod
- Sprint Backlog → Obj del Sprint
- Increment → Def of DONE

Product Backlog

Lista emergente y ordenada de lo que se necesita para mejorar el producto. Es la única fuente del trabajo realizado por el Scrum Team.

Sprint Backlog

Se compone del Objetivo del Sprint (por qué), el conjunto de elementos del Product Backlog seleccionados para el Sprint (qué), así como un plan de acción para entregar el Increment (cómo).

Increment

Peldaño concreto hacia el Objetivo del Producto. Debe ser utilizable

U3

Software Configuration Management (SCM)

Sistemas de software siempre cambian, pueden considerarse como un conjunto de versiones donde cada una de ellas debe mantenerse y gestionarse.

SCM: actividad de soporte transversal a todo el proyecto que aplica dirección y monitoreo administrativo y técnico. Cuyo propósito es **mantener la integridad del producto a lo largo de todo el ciclo de vida.**

1. **Identificar** características técnicas y funcionales de **ítems de configuración**
2. **Documentar** características técnicas y funcionales de **ítems de configuración**
3. **Controlar los cambios** de esas características identificadas y documentadas
4. **Registrar y reportar estos cambios**
5. Verificar **correspondencia con los requerimientos** (trazabilidad).

Su propósito es resolver problemas:

La integridad es el medio por el cual podemos garantizar que el producto a entregar tiene la calidad correspondiente. Y nos garantiza un nivel mínimo de confiabilidad.

PÉRDIDA DE COMPONENTES
PÉRDIDA DE CAMBIOS (LA VERSIÓN DEL COMPONENTE NO ES LA ÚLTIMA)
REGRESIÓN DE FALLAS
DOBLE MANTENIMIENTO
SUPERPOSICIÓN DE CAMBIOS
CAMBIOS NO VALIDADOS.

El problema de la calidad es que es subjetiva y el sw intangible. Es muy difícil medir y la calidad corresponde a las expectativas del cliente.

Se mantiene la integridad de un prod de sw cuando

1. Satisface las necesidades de usuario
2. Permite rastreabilidad durante todo su ciclo de vida
3. Satisface criterios de performance y RNF
4. Cumple con expectativas de costo.

Conceptos generales

Item de configuración: aquellos artefactos que forman parte del producto y pueden ser almacenados en un repositorio, sin importar su extensión. Pueden sufrir cambios.

Repositorio: contenedor de ítems de configuración, se encarga de mantener la historia

- **Centralizado:** un servidor contiene todos los archivos. Si falla, todo lo positivo se cae
- **Descentralizado:** cada cliente tiene una copia del repo completo. más complicado llevar control

Versión: Instancia de un ítem de configuración que difiere de otras instancias de este ítem. Cada versión de software es una colección de **elementos de configuración (ECS)**

El **control de versiones** se refiere a la evolución de un único ítem de configuración (IC)

Variante: versión de un ítem que evoluciona por separado

Configuración de Software: Conjunto de todos los ítems de configuración con su versión específica. Define una foto de los ítems de configuración en un momento de tiempo determinado.

Línea Base: conjunto de IC que fueron construidos y revisados formalmente. Sirve como base para desarrollos posteriores. Para cambiar una línea base, existe un protocolo formal de control de cambios. Sirve para tener un punto de referencia, saber cual es la última situación estable.

- **De especificación:** las primeras, no tienen código. Podría contener el documento de especificación de requerimiento.
- **Operacionales:** contiene versión de producto cuyo código es ejecutable.

Rama: conjunto de IC con sus respectivas versiones, que permiten bifurcar el desarrollo de un sw.

Actividades relacionadas a la gestión de configuración

Actividades que contienen las secciones de un plan de gestión de configuración. (Estas se hacen en metodología tradicional y ágil, menos la auditoría que no es compatible con la meto. ágil)

Identificación de IC

Id unívoca para cada ítem. En el equipo se definen políticas y reglas de nombrado y versionado, la estructura del repo y la ubicación de los IC dentro de la estructura.

Se debe tener en cuenta al id los ítems, la duración de su integridad:



Control de cambios

Se hace sobre los ítems de configuración que pertenecen a la línea base. Es llevado a cabo por el comité de control de cambios, el cual se reúne para autorizar la creación y cambios sobre la línea base. Etapas:

1. Se recibe propuesta de cambio (sobre línea, no ítem)
2. Análisis de impacto del cambio
3. Si se autoriza, se genera una orden de cambio donde se define que se va hacer
4. Después de hacer el cambio, aprobar la modificación de la LB
5. Notificar a los interesados

Auditorías de configuración

Controles autorizados. Auditor externo analiza las LB (NO EN ÁGIL). Tiene que ser objetivo.

- Auditoría física de configuración (PCA): asegura que lo que está indicado para cada IC en la línea base se alcanzó realmente
- Auditoría funcional de configuración (FCA): eval. independiente de los prod de sw, controlando que la funcionalidad y performance real de cada IC sea consistente con las especificación de reqs

Validación (que cada IC resuelva el problema apropiado) y verificación (producto cumple con los objetivos definidos)

Reportes e informes de estado

para mantener un registro de cómo evoluciona el sistema, y dónde está ubicado el software.

Reportes incluyen todos los cambios que han sido realizados a las líneas base durante el ciclo de vida del software.

Evolución de la gestión de configuración

Integración continua: integrar su código a un repositorio compartido varias veces al día. Es asegurar que el software pueda ser desplegado en cualquier momento (**TDD**). El componente terminado se sube a repo de integración así está en condiciones de ir a las pruebas de aceptación de usuario.

Entrega continua: Suma la automatización de las pruebas de aceptación y entonces el producto ya está listo para desplegarlo a producción. Es prerequisite la integración continua

Despliegue continuo: poner en el ambiente de producción del usuario final el producto. No existe la intervención humana para desplegar el producto en producción. Para esto, se utilizan pipelines. Se recomienda hacer el despliegue a poco tiempo de haber trabajado con los cambios en el código.

SCM en ambientes ágiles.

Posibilita el seguimiento y la coordinación del desarrollo en lugar de controlar a los desarrolladores.

Entre otras tareas, SCM en Agile:

- △ Hace seguimiento y coordina el desarrollo en lugar de controlar a los desarrolladores.
- △ Responde a los cambios en lugar de tratar de evitarlos.
- △ La automatización debe ser aplicada donde sea posible. Esto colabora con la entrega frecuente y rápida de software. (Continuous Integration)
- △ Coordinación y automatización frecuente y rápida.
- △ Eliminar el desperdicio - no agregar nada más que valor.
- △ Provee documentación Lean y trazabilidad.
- △ Provee retroalimentación continua sobre calidad, estabilidad e integridad

Paper No Silver Bullets

A ver si existe una bala de plata que abarate los costos del sw. El tipo este dice que no hay solución porque el software es muy complejo naturalmente. Para ver la tasa de crecimiento del sw, lo divide en dificultades esenciales y accidentales:

Dificultades esenciales: naturales, propias del sw

- complejidad: sw es complejo porque no hay dos partes iguales. Un incremento en escala de un sw no es repetir los mismos elementos pero en tamaño más grande. En la mayoría de los casos, los elementos interactúan entre ellos de una manera no lineal. Muchos de los problemas clásicos de desarrollar productos de sw derivan de esta complejidad esencial y su incremento no lineal con el tamaño .
De esta complejidad surge la dificultad de enumerar todos los posibles estados del programa (falta de confianza). De la complejidad de las funciones, viene la complejidad de poder utilizarlas. De la complejidad de la estructura viene la complejidad de agregar nuevas funciones, y las brechas de seguridad.
- conformidad: el sw se debe adaptar a otros aspectos externos.
- posibilidad de cambio: todo sw exitoso cambia.
- invisibilidad: sw es invisible y no visible. Si intentamos diagramar su estructura, no es uno, pero muchos gráficos, uno sobre otro.

dificultades accidentales:

- lenguajes de alto nivel: los leng. de alto nivel liberan a un programa de un complejidad accidental, sacando detalles de máquina. Pero un exceso de sofisticación puede complicar en lugar de ayudar.
- tiempo compartido: (uso del procesador al mismo tiempo). Ayuda a la inmediatez
- entorno de programación unificado: avance tecnológico, inclusión de herramientas a muchos programas usando formatos estándar.

esperanzas para la bala de plata

- ada y otros avances de lenguajes de alto nivel: salto de productividad con lenguajes de alto nivel .
- programación orientada a objetos: reducen la complejidad de como expresar el diseño (esencial).
- ia: divide en dos tipos de IA
 - 1: resuelve lo que antes requería inteligencia humana
 - 2: uso de técnicas de programación que establecen reglas.
- sistemas expertos: (el 2 de la IA). es un programa que contiene un motor de inferencias generalista y una base de reglas, que coge datos de entrada y suposiciones, explora las inferencias derivadas de la base de reglas, consigue conclusiones y consejos,
- programación automática: es un programa que resuelve un problema con una definición de lo que es el programa. (eeee?)
- programación gráfica: los diagramas visuales no llegan a describir la estructura del sw
- verificación de programa: es necesario para el programa la validación. no es una solución
- herramientas y entorno: capaz la ganancia más grande en el entorno de programación es el uso de db integradas.
- workstations:

ataques prometedores al concepto de esencia

- comprar vs construir: no reinventar la rueda
- refinado de requerimientos y prototipado rápido: lo más difícil es definir qué construir. los clientes no saben lo que quieren, no van a poder especificar todo.
- desarrollo incremental, crecer, no construir.: MVP !!!!!hacer algo mínimo y hacerlo crecer por capas
- gran diseñadores: importa el humano