Plan Promocionar ISW

1er Parcial

Unidad 1: Ingeniería de Software en Contexto

Introducción a la Ingeniería del Software. ¿Qué es?

Primero hay que saber que es Software. Es un conjunto de programas con la documentación requerida para definir, desarrollar y mantener los entregables para el cliente como los son los archivos de configuración, documentos que describen la arq. Del sistema, etc.

Ahora, no alcanza con saber programar, y luego de una crisis del software surge la Ing. De Software.

La Ing. De Software es una disciplina que se encarga de crear Software, desde la primera etapa de requerimientos hasta el mantenimiento del sistema. Esta se apoya en varias realidades:

- Se debe hacer un esfuerzo para entender el problema antes de empezar a desarrollar software.
- El diseño es una actividad crucial. Esto trae dos claros beneficios, calidad de software y facilidad de mantenimiento.

Estado actual y antecedentes. La Crisis del Software.

La crisis del software es (porque sigue existiendo) un conjunto de dificultades o errores ocurridos en la planificación, estimación de costos, productividad y calidad de software.

Causas:

- La evolución de tecnología a nivel hardware. Esto permitió la posibilidad de desarrollar sistemas más grandes y complejos. Pero el salto del hardware no fue acompañado por un salto en desarrollo de software.
- La demanda creciente de sistemas complejos. Los productos deben ser adaptados a las necesidades del cliente y la falta de una disciplina que intervenga en todos los aspectos de la producción del software fueron las cusas de esta crisis.

En la actualidad, los sistemas fallan debido a demandas crecientes de construir software de manera rápida y los métodos existentes (tradicionales) de ingeniería no permiten lograrlo. También, muchas de las compañías de desarrollo de software se conforman con que el software funcione, sin tener en cuenta el valor agregado que el software le debe aportar al negocio del usuario. Por estas razones es necesario una mejor educación y capacitación en ingeniería de software.

Disciplinas que conforman la Ingeniería de Software.

La ingeniería de software está compuesta por:

- Disciplinas técnicas: Requerimientos, análisis y diseño, construcción, prueba, despliegue. (actividades que aportan al desarrollo como producto).
- Disciplinas de gestión: Planificación de proyectos, monitoreo y control de proyectos. (actividades que aportan al desarrollo como proyecto).
- Disciplinas de soporte: SCM, aseguramiento de calidad, métricas. (actividades transversales a los procesos de software).

Ciclos de vida (Modelos de Proceso) y su influencia en la Administración de Proyectos

Son la serie de pasos a través de los cuales le producto o proyecto progresa. Son una representación simplificada de un proceso, es decir, son modelos genéricos de los procesos de software y establecen criterios que se utilizan para determinar si se debe pasar de una tarea a otra.

Estas especifican: las fases del proceso y el orden en el cual se llevan a cabo.

El ciclo de vida del producto siempre es mayor que el del proyecto, ya que el ciclo de vida del proyecto dura lo que dura el desarrollo de software. En cambio, el ciclo de vida del producto dura hasta que el software se deje de utilizar. Entonces, es posible que un producto tenga varios proyectos en su ciclo de vida, debido a contantes cambios y/o actualizaciones que se vayan realizando. En otras palabras, dentro del ciclo de vida del producto, pueden que se desarrollen varios ciclos de vida de proyectos.

Los ciclos de vida se clasifican en:

- **Secuenciales:** El proyecto progresa a través de una secuencia ordenada de pasos y una actividad no puede iniciar sin que la precedente haya sido finalizada.
  Generalmente, están dirigidos por documentos, es decir, el trabajo principal del producto es la documentación del software entre las distintas fases.
  Dificultades:
  - Los procesos raramente siguen un flujo secuencial.
  - Exige que los reqs. sean completamente explícitos desde un principio y esto no suele suceder.
  - Software entregable en etapas muy avanzadas del proyecto.
  - Un defecto implica un gran rediseño en la solución.
  - Documentación muy burocrática y excesiva.
- **Iterativo/Incremental:** Se aplican sucesivas iteraciones en forma escalonada a medida que avanza el calendario de actividades, donde cada iteración produce un incremento de software funcional potencialmente entregable. El sistema se desarrolla como una serie de versiones y cada una añade funcionalidad a la versión anterior.
  Dificultades:
  - Se invisibiliza el proceso.
  - Alto costo de documentar cada incremento.
  - La estructura del software tiende a degradarse frente a una cantidad considerable de incrementos.
- **Recursivo:** Es utilizado para gestionar riesgos del desarrollo de sistemas complejos a gran escala y requiere de una intervención del cliente.
  Dificultades:
  - Puede ser más costoso en tiempo y dinero readaptarlos para reutilizarlos en otros proyectos.
  - La tecnología puede ser obsoleta.
  - Puede carecer de mantenimiento o documentación.
  - Se genera una versión del producto recién al final.

Como vemos, la elección de un ciclo de vida afecta de forma directa a la administración que se debe realizar sobre el proyecto, ya que cada uno de los modelos de proceso poseen características y enfoques distintos.

Procesos de desarrollo Empíricos vs Definidos.

El proceso de desarrollo de software es un conjunto de actividades para desarrollar un sistema de software. Estas varían dependiendo de la organización, del tipo de sistema que debe desarrollarse y del ciclo de vida que elijamos.

Existen dos tipos de procesos de desarrollo:

- Definidos: Son considerados deterministas. Estos asumen que podemos repetir el mismo proceso una y otra vez, indefinidamente, y obtener los mismos resultados, incluso cambiando de equipos.
  Están inspirados en líneas de producción.
  La administración y el control provienen de la predictibilidad del proceso.
  Puedo usar cualquiera de los ciclos de vida.
- Empíricos: No son considerados deterministas. Se conforman en base a la experiencia interna y externa de las personas que intervienen en un contexto particular.
  Se basan en ciclos cortos de inspección y adaptación.
  La administración y el control es por medio de inspecciones frecuentes y adaptaciones para lograr buenas prácticas.
  No se pueden combinar con cualquier ciclo de vida, solo con el modelo iterativo-incremental.
  Se basan en tres pilares fundamentales: adaptación, inspección y transparencia.

Ciclos de vida (Modelos de Proceso) y Procesos de Desarrollo de Software.

Se tienen en cuenta tres factores determinantes del proceso:

- Procedimientos y métodos: La adaptación de los procesos establecidos es esencial para la calidad resultante.
- Personas motivadas, capacitadas y con habilidades: Son primordiales para obtener la calidad del producto del proceso, ya que éste se determina del esfuerzo de las personas.
- Herramientas y equipos: Son los materiales necesarios para llevar a cabo el proceso. Se recomienda automatizar la mayor cantidad de actividades posibles, lo que mejora la eficiencia de las personas y puedan concentrarse en tareas que requieran de su capacidad.

Actividades fundamentales de un proceso de desarrollo de software:

- Especificación de software: Clientes junto con ingenieros definen el software a producir y sus restricciones.
- Desarrollo: Diseño y programación del software.
- Validación: Asegurarse que el software construido es lo que el cliente quiere.
- Evolución: Donde se modifica el software para reflejar los reqs. cambiantes del cliente y el mercado.

Criterios para la elección de ciclos de vida en función de las necesidades del proyecto y las características del producto.

Ciclo de vida	Procesos de desarrollo que lo admite	Características que lo hacen elegible
Secuencial	Definidos	Alta certeza. La organización tiene una forma tradicional de trabajar. No se pueden realizar entregas parciales del software. Eliminar riesgos de planificación.
Iterativa/Incremental	Definidos o Empíricos	Existe incertidumbre. Volatibilidad de reqs. Posibilidad de entregas parciales. Uso en etapas tempranas del producto
Recursivo	Definidos	Mucho control de riesgos en cada iteración. No se pueden realizar entregas parciales.

Componentes de un Proyecto de Sistemas de Información.

Primero definamos que es un proyecto de software: Es un esfuerzo temporal que requiere del acuerdo de un conjunto de especialidades y recursos para la creación de un producto único. Este define el proceso y tareas que se van a realizar, el personal que se encargará de las actividades y los mecanismos que se implementarán para valorar riesgos, controlar el cambio y evaluar la calidad.

Características:

- Únicos: El resultado de un proyecto es único e irrepetible, por más parecido o igual que sean sus componentes con otro proyecto.
- Orientados a objetivos: Están dirigidos a obtener resultados y ello se refleja a través de objetivos. Estos objetivos guían al proyecto, por lo tanto, deben ser no ambiguos, claros y alcanzables.
- Son de duración limitada: Los proyectos son temporarios, cuando se alcanza el objetivo, el proyecto se termina. Tienen un principio y un fin, liberando los recursos y personas.
- Tienen tareas interrelacionadas basadas en esfuerzos y recursos: Se definen las tareas, sus dependencias, los recursos y personas asignadas, permitiendo manejar la complejidad inherente de los sistemas.

Administración de un proyecto de software: Es la aplicación de conocimientos, habilidades, herramientas y técnicas a las actividades del proyecto para satisfacer los requerimientos del proyecto en tiempo, con el presupuesto acordado. Administrar un proyecto incluye:

- Identificar los requerimientos.
- Establecer objetivos claros y alcanzables.
- Adaptar las especificaciones, planes y el enfoque a los diferentes intereses de los involucrados.

Lo importante de la planificación es el acto de planificar, no el resultado.

La Restricción Triple:

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

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

Un proyecto va a tener un Líder de Proyecto y éste es el que realiza la toma de decisiones en su totalidad sin la inclusión de la opinión del equipo. El líder asigna las tareas que deben realizar los integrantes del equipo y además maneja todas las relaciones con todos involucrados del proyecto.

Luego, el Equipo de Proyecto, es un grupo de personas comprometidas con alcanzar los objetivos de los cuales se sienten mutuamente responsables. Estos cuentan con diversos conocimientos y habilidades, la posibilidad de trabajar juntos efectivamente desarrollando sinergia y tienen un sentido de responsabilidad como una unidad. Es usualmente un grupo pequeño.

Y, por último, se crea un Plan de Proyecto. Este es un artefacto de gestión que cumple la función “hoja de ruta” de un proyecto, en el cual se documentan todas las decisiones que se toman a lo largo del desarrollo:

- ¿Qué es lo que hacemos? ←Alcance del proyecto.
- ¿Cuándo lo hacemos? ← Calendario.
- ¿Cómo lo hacemos? ← Actividades o tareas para cubrir el alcance.
- ¿Quién lo va a hacer? ← Responsables de cubrir las actividades.

Responder a estas preguntas implica una planificación del proyecto de software:

- Definición del Alcance del Proyecto.
- Definición del Procesos y su Ciclo de vida.

- Estimación.
- Gestión de Riesgos.
- Asignación de Recursos.
- Programación de Proyectos.
- Definición de Controles.
- Definición de Métricas.

Vinculo proceso-proyecto-producto en la gestión de un proyecto de desarrollo de software.

El proceso, automatizado con herramientas, se adapta al proyecto, al cual se incorporan personas, y de este se obtiene como resultado un producto.

El proceso se adapta al proyecto porque depende del tipo de producto que se desea desarrollar, ya que puede que la complejidad de este sea un determinante en la elección del proceso. De esta manera se utiliza solo lo que verdaderamente hace falta, ya que no existe un proceso ideal que sirva para cualquier tipo de proyecto.

Un proyecto esta integrado por un equipo de trabajo, dentro del cual deben estar los roles bien definidos, para que cada uno asuma la responsabilidad que le corresponda.

Hay que tener en claro que las personas son la parte más importante, después de todo el desarrollo de software es una actividad humano-intensiva y no nos servirá de nada usar el "mejor proceso" y además planificar mucho si no tenemos un equipo capacitado.

En el desarrollo de un proyecto, se busca utilizar herramientas que nos permitan automatizar o facilitar las actividades que están definidas en el proceso adoptado.

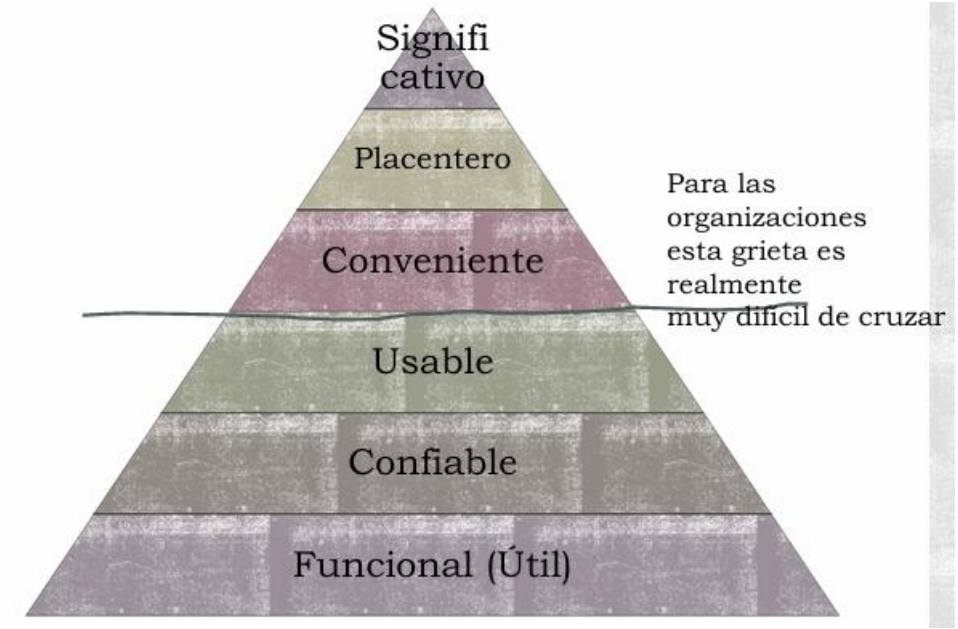
Unidad 2: Gestión Agile de Productos de Software.

Gestión de Producto.

Un producto de software se construye para satisfacer una necesidad específica en base a ciertos requerimientos. Es una solución que entrega valor real al usuario con el menor desperdicio posible.

Las organizaciones que usan el software se enfocan en aquello que consideran significativo y placentero. Usualmente no se fijan si el producto es funcional o confiable. Esa debe ser la base, pero el producto debe proveer una experiencia que cumpla con todas las expectativas del usuario.

Evolución de los productos de software:



Pero ¿cómo podemos saber que un producto pueda generar valor al cliente?

MVP (Producto Mínimo Viable): Es un concepto de Lean Startup y se define como la versión de un nuevo producto que permite a un equipo recopilar la cantidad máxima de aprendizaje sobre los clientes con el menor esfuerzo.

Con los MVPs podemos ver lo que la gente realmente hace con el producto antes de preguntarle a la gente que harían.

Características:

- Diseño:
  - Diseño adecuado.
  - Consigue una UX que deleita.
  - Logra satisfacer el aspecto visual y de interacción.
- Usabilidad:
  - Tiene suficiente valor para que la gente esté dispuesta a usarlo/comprarlo.
  - Resulta útil para su público objetivo.
- Confiabilidad:
  - Involucra a los early adopters puedan confiar en la solución plenamente, incluso cuanto tenga poco tiempo en el mercado.
- Funcionalidad:
  - Tiene las funciones necesarias para solucionar un problema específico.
  - Satisface las demandas y permite evaluar las funciones a implementar más adelante.

Un MVP permite validar diferentes hipótesis:

- Hipótesis del valor: Prueba si el producto realmente está entregando valor a los clientes después de usarlo. También sirve como métrica de prueba (tasa de retención).
- Hipótesis de crecimiento: Prueba cómo nuevos clientes descubrirán el producto. También sirve como métrica de prueba (tasa de referencia o Net Promoter Score (NPS).

Un MVP puede tener otras piezas:

Relación entre MVP, MMF, MMP, MMR:

- MVP:
  - Dirigido a un subconjunto de clientes potenciales.

    - Más cercano a los prototipos que a una versión real funcionando de un producto.
- MMF (Característica Mínima Comerciable):
    - Es la pieza más pequeña de funcionalidad que puede ser liberada.
    - Tiene valor tanto para la organización como para los usuarios.
    - Es parte de un MMR o MMP.
- MMP (Producto Mínimo Comerciable):
    - Primer reléase de un MMR dirigido a primeros usuarios (early adopters).
    - Focalizado en características clave que satisfarán a este grupo clave.
- MMR (Características Mínimas del Release):
    - Release de un producto que tiene el conjunto de características más pequeño posible.
    - El incremento más pequeño que ofrece un valor nuevo a los usuarios y satisface sus necesidades actuales.
    - MMP = MMR1.

Idea → MVP (aprendo si sirve) → MMF (pequeña funcionalidad útil) → MMR (conjunto de MMFs listos para lanzar) → MMP (el primer MMR que se lanza)

MVF (Característica Mínima Viable): Es una versión mini del MVP, es decir, una versión muy básica de una funcionalidad, hecha con el menor esfuerzo posible para validar si vale la pena desarrollarla más a fondo.

Se usa para aprender si esa funcionalidad interesa al usuario, igual que un MVP, pero a nivel feature.

Ejemplo: Antes de invertir en la funcionalidad "mensajes de voz", lanzas una versión simple que graba y envía sin edición ni vista previa. Si los usuarios la usan mucho, la mejoras.

Diferencias entre un MVF y un MMF:

MMF	MVF
Lista para lanzarse	Lista para validar
Tiene valor por sí sola	Busca si tendrá valor
Parte de una reléase (MMR)	Parte de un experimento

Requerimientos Ágiles.

¿Qué significa Ágil?

Balance entre un ningún proceso y demasiado proceso. La diferencia inmediata es la exigencia de una menor cantidad de documentación. Los métodos ágiles son adaptables en lugar de predictivos, y son orientados a la gente en lugar de orientados al proceso.

Ahora ¿cómo sacamos los requerimientos en este enfoque?

“La parte más difícil de construir un sistema de software es decidir precisamente qué construir. Ninguna otra parte del trabajo conceptual es tan difícil como establecer los requerimientos técnicos detallados, ninguna otra parte del trabajo afecta tanto al sistema resultante si se hace incorrectamente, ninguna otra parte es tan difícil de rectificar más adelante.” No silver Bullet.

La etapa de capturar los requerimientos se transforma en una etapa de trabajo muy integrado entre el cliente y el equipo, ya que el enfoque de requerimientos ágiles esta adherido al manifiesto ágil, es decir, cumple con los valores y principios que el manifiesto plantea.

¿Cuál va a ser nuestra fuente de requerimientos?

Se obtienen conversando cara a cara con el cliente (principio de que el mejor medio de comunicación es el cara a cara). En Agile no se escribe una ERS (como en los ambientes tradicionales), pero se compensa con la disponibilidad del cliente para resolver las dudas. Pero, el desarrollo ágil no implica la no documentación o la no especificación de requerimientos, sino que prioriza documentar y especificar aquellos que sean de valor para el cliente, antes de hacer toda una ERS errónea.

También es importante saber que los requerimientos son relevados continuamente durante todo el proyecto. El inicio de la construcción del software se da en un contexto donde los requerimientos no están completamente definidos, pero se tiene una visión.

Tipos de requerimientos:

En el enfoque ágil, se trabaja con los requerimientos de negocio y los requerimientos de usuario.

Al usuario le interesa el requerimiento de negocio y a través del requerimiento de usuario vamos a lograr ese valor de negocio.

¿Cómo vamos a gestionar esos requerimientos?

Con el Product Backlog. Éste, es una lista priorizada de características y requerimientos del software. Es el Product Owner quien se encarga de definirlo y es él quien tiene en claro las necesidades y requerimientos principales.

Al principio, se empieza con una visión del producto y con algunos requerimientos identificados. A partir de esta base, se comienza a construir el producto y se incorpora al cliente para obtener retroalimentación, por esto el Product Backlog nunca está al 100%.

Los requerimientos se analizan cuando se los necesitas, no antes. A esto se le llama requerimientos “just in time” y el enfoque ágil lo utiliza para cumplir con el principio de eliminar desperdicio y diferir compromisos.

¿Qué hay que tener en cuenta?

- Los cambios son la única constante.
- Stakeholders: no son todos los que están (solo los involucrados).
- Siempre se cumple eso de que: “El usuario dice lo que quiere cuando recibe lo que pidió”.
- No hay técnicas ni herramientas que sirvan para todos los casos.
- Lo importante no es entregar una salida, un requerimiento, lo importante es entregar un resultado, una solución de valor.

Principios Ágiles relacionados a los Requerimientos Ágiles:

1. La prioridad es satisfacer al cliente a través de releases tempranos y frecuentes.

2. Recibir cambios de requerimientos, aun en etapas finales.

4. Técnicos y no técnicos trabajando juntos todo el proyecto.

6. El medio de comunicación por excelencia es cara a cara.

11. Los mejores arquitecturas, diseños y requerimientos emergen de equipo autoorganizados.

User Stories.

La elicitación tradicional no da buenos resultados, ya que todas las decisiones se toman al principio del proyecto cuando menos información se tiene. Por lo que se estableció que esta toma de decisiones se traslada a lo largo del proyecto, a medida que la incertidumbre baja y la información sube, buscando que se obtenga más a menor tiempo posible. Aquí surgen las User Stories:

Son una técnica para trabajar requerimientos de usuario en ambientes ágiles.

Compuesta de 3 partes: Tarjeta, Conversación, Confirmación (triple “c” en inglés),

Son multipropósito, ya que son:

- Una necesidad del usuario.
- Una descripción del producto.
- Un ítem de planificación.
- Token para una conversación.
- Mecanismo para diferir una conversación.

Se consideran verticales dentro del producto, ya que pueden incluir aspectos desde diseños de interfaces hasta diseños de tablas en la base de datos.

Una de las partes en la tarjeta de la User Story son los **criterios de aceptación**, éstas:

- Definen límites para una user story.
- Ayudan a que los PO respondan a lo que necesitan para la US provea valor (requerimientos funcionales 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.

Los detalles como el encabezado de la columna se nombra “saldo” o el formato del saldo es 999.999.999,99 se pueden capturar en lugares como la documentación interna de los equipos, pruebas de aceptación automatizadas.

Y la última parte de una US son las **pruebas de aceptación**: Son declaraciones de intención de que hay que probar. Se prueban escenarios exitosos y escenarios que fallen. Pero solo se encuentran las pruebas más importantes. El PO acepta la US como implementada si todas las pruebas de aceptación se cumplen.

Hay dos definiciones que debemos tener en cuenta para validar US:

DoR (Definition of Ready): Es una medida de calidad que determina si la US está en condiciones de entrar a una iteración de desarrollo.

Para que una US pueda ser implementada, es decir, para que cumpla con DoR, mínimamente debe satisfacer el **INVEST Model**:

- Independiente: La US es calendarizable e implementable en cualquier orden.
- Negociable: Las US no son contratos estrictos, sino más bien que se dejan para negociar en conversaciones futuras entre clientes y el equipo de desarrollo.

- Valuable: Las USs deben aportar valor de negocio a quienes estará destinado el producto, no para el equipo de desarrollo.
- Estimable: La US debe contener la cantidad de información suficiente para estimar tamaño, complejidad y esfuerzo.
- Small: La US debe ser lo suficientemente pequeña como para ser finalizada en una iteración.
- Testeable: La US debe cumplir con las pruebas de aceptación para poder probar si la misma fue o no implementada.

DoD (Definition of Done): Determina si la US está terminada. El PO decide si se pasa a producción o no viendo que se cumpla con los criterios pactados. Si una US cumple con el DoD, se considera que está lista para ser desplegada y aportar valor al cliente.

Hay un tipo especial de US llamada **Spike**. A veces, una US presenta mucha incertidumbre, cosa que imposibilita que pueda ser estimada y por lo tanto no cumple con la DoR. Las Spikes se utilizan para quitar el riesgo e incertidumbre de una US y una vez resuelta la incertidumbre, la Spike se convierte en una o más USs.

Se clasifican en:

- Técnicas:
    - Utilizadas para investigar enfoques técnicos en el dominio de la solución (evaluar performance potencial, decisión hacer o comprar, evaluar la implementación de cierta tecnología).
    - Utilizadas para cualquier situación en la que el equipo necesite una comprensión más fiable antes de comprometerse a una nueva funcionalidad en un tiempo fijo.
- Funcionales:
    - Utilizadas cuando hay cierta incertidumbre respecto cómo el usuario interactuará con el sistema.
    - Usualmente son evaluadas con prototipos para obtener retroalimentación de los usuarios o involucrados.

Estimaciones Ágiles.

Las features/US son estimadas usando una medida de tamaño relativo conocido como **story points**.

Las estimaciones nos pueden servir como una gran respuesta temprana sobre si el trabajo planificado es factible o no.

Estimamos **tamaño**: es una medida de la cantidad de trabajo necesaria para producir una story. Éste indica:

- Cuán compleja es una story.
- Cuánto trabajo es requerido para hacer o completar una story.
- Cuán grande es una story.

Tamaño NO ES esfuerzo, ya que las estimaciones basadas en tiempo son más propensas a errores debido a varias razones:

- Habilidades.
- Conocimiento.
- Experiencia.

Al tamaño lo podemos estimar con **Story Points**: Es una unidad de estimación que especifica la complejidad, incertidumbre y esfuerzo propio del equipo respecto a una US en términos relativos respecto a otra US (canónica). Nos da idea del “peso” de cada US y decide cuan grande (compleja) es. Podemos elegir alguna escala para medir con SP, por ejemplo, Fibonacci.

La filosofía Ágil plantea que quien estima, es quien debe hacer el trabajo, con lo cual es el equipo de desarrollo quien realiza las estimaciones.

Uno de los métodos más utilizados para estimar es el Poker Estimation.

De las estimaciones podemos sacar una métrica importante, la **velocidad (Velocity)**. Esta métrica mide el progreso de un equipo. Se calcula sumando el número de SP (asignados a cada US) que el equipo completa durante la iteración (solo los que cumplen con el DoD y aprobados por PO). La Velocidad permite corregir errores de estimación.

Unidad 3: Gestión del Software como Producto.

SCM (Software Configuration Management)

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

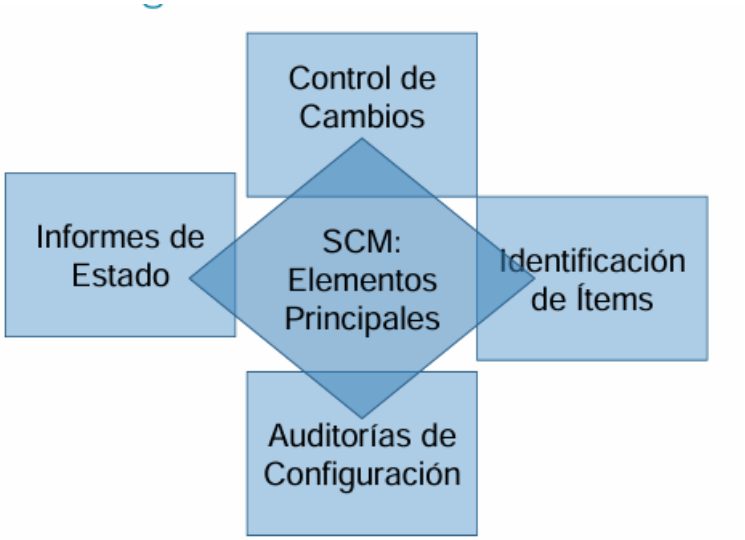
Su propósito es establecer y mantener la **integridad** de los productos de software a lo largo de su ciclo de vida. El producto de software es íntegro cuando:

- Satisface las necesidades del usuario.
- Puede ser fácil de rastrear durante su ciclo de vida.
- Satisface criterios de performance.
- Cumple con sus expectativas de costo.

Conceptos claves para la Gestión de Configuración de Software:

- 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.
- Versión: Estado particular en el tiempo de un ítem de configuración. Cada versión de software es una colección de ítems de configuración en un instante de tiempo.
    - Control de versiones: Hace referencia a la evolución de un único ítem de configuración.
- Variante: Versión de un ítem de configuración que evoluciona por separado. Las variantes representan configuraciones alternativas, y si la diferencia que existe entre las instancias de un mismo ítem de configuración es muy pequeña, también se la denomina variante.
- Configuración: Conjunto de todos los ítems de configuración con su versión específica.
- Repositorio: Contenedor de ítems de configuración, se encarga de mantener la historia de cada ítem con sus atributos y relaciones.
    - Centralizado: Un servidor contiene todos los archivos con sus versiones.
    - Descentralizado: Cada cliente tiene una copia exactamente igual del repositorio completo.
- Rama (Branch): Es un conjunto de ítems de configuración con sus correspondientes versiones, que permiten bifurcar el desarrollo de un software.
- Línea Base: Configuración de Software que ha sido formalmente revisada y aprobada y que sirve como base para desarrollos futuras. Pueden ser:
    - De Especificación (son las primeras línea base, dado que no cuentan con código).
    - De productos que han pasado por un control de calidad definido previamente. La primera línea base corresponde con la primera reléase.

Actividades Fundamentales de la SCM:



Identificación de Ítems de configuración:

- Identificación unívoca de cada ítem de configuración.
- Convenciones y reglas de nombrado.
- Definición de la estructura del Repositorio.
- Ubicación dentro de la estructura del repositorio.
- Se clasifican en:
    - De producto.
    - De proyecto.
    - De iteración.

Control de Cambios:

Tiene su origen en un requerimiento de cambio a uno o varios ítems de configuración que se encuentran en la línea base. Por lo que se define un proceso formal que involucra diferentes actores y una evaluación del impacto del cambio.

Para el control de cambios se forma el Comité de Control de Cambios.

Auditorías de Configuración de Software:

- Auditoría Física de Configuración: Asegura que lo que está indicado para ICS en la línea base o en la actualización se ha alcanzado realmente.
- Auditoría Funcional de configuración: Evaluación independiente de los productos de software, controlando que la funcionalidad de cada ítem de configuración sea consistente con la especificación de requerimientos.

Informes de Estado:

- Se ocupa de mantener los registros de la evolución del sistema.
- Maneja mucha información y salidas por lo que se suele implementar dentro de procesos automáticos.
- Incluye reportes de rastreabilidad de todos los cambios realizados a las líneas base durante el ciclo de vida.

Plan de Gestión de Configuración. Incluye:

- Reglas de nombrado de los IC.
- Herramientas a utilizar para SCM.
- Roles e integrantes del Comité.
- Procedimiento formal de Cambios.
- Procesos de Auditoría.

SCM en ambientes Ágiles:

Los equipos ágiles son autoorganizados, por lo que las Auditorías de configuración no son una actividad propia de la gestión de configuración en los ambientes ágiles, tampoco existe un comité para el proceso de Control de Cambios.

La diferencia es que el manifiesto ágil se enfoca en los proyectos, mientras que SCM se enfoca en el producto.

Entonces,

- Dada la naturaleza de los requerimientos ágiles, SCM responde a los cambios, en lugar de evitarlos.
- La automatización debe ser aplicada donde sea posible.
- Provee retroalimentación continua sobre calidad, estabilidad e integridad.
- Las actividades de SCM se encuentran embebidas en las demás tareas para alcanzar el objetivo del sprint.
- Es responsabilidad del equipo.
- Se adoptan prácticas continuas.