# Resumen parcial 1:

## Temas: unidad 1, gestión de producto, Req. Agiles y user stories, estimaciones agiles, SCM, No silver bullets.

Unidad 1: introducción a la ingeniería de software, estado actual, la crisis del software, disciplinas de ISW, ejemplos de grandes proyectos de software fallidos y exitosos, ciclos de vida y su influencia en proyectos de software, procesos de desarrollo empíricos vs. Definidos, ciclos de vida y procesos de desarrollo, ventajas y desventajas de ciclo de vida, criterios para la elección de ciclos de vida, componentes de un proyecto de sistemas de información, vinculo proceso, proyecto, producto en la gestión de un proyecto.

# Ingeniería de software:

La **ingeniería de software** es una disciplina de la ingeniería que tiene como objetivo el desarrollo de sistemas de software costeable. El software es algo abstracto, intangible y fácil de modificar. Comprende todos los aspectos de la producción de software, desde las etapas iniciales de la especificación del sistema hasta el mantenimiento de este.

* El **software** es información estructurada con propiedades lógicas y funcionales que fue creada y es mantenida de varias formas, y es confeccionada para ser procesada por computadoras en su estado más desarrollado. El software no es solo código, sino que cuenta con varias cosas que hacen que el software sea más que el código, el software es toda información que se puede presentar en varios niveles de abstracción
* **Crisis del software:** surge de la introducción de nuevas computadoras basadas en circuitos integrados que provoco la posibilidad de desarrollar software más grande y complejos. En consecuencia, los proyectos de software se atrasaban, otros se cancelaban, se sobrepasaban de presupuesto, es decir, se generaba software de mala calidad y desempeño pobre que requerían intensas actividades de mantenimiento. Se debieron crear nuevas técnicas y métodos para formalizar el desarrollo de software.

**Un proceso de** software se define como un conjunto estructurado de actividades que a raíz de un conjunto de entradas producen una salida. Debe ser explícitamente modelado si va a ser administrado o llevado a cabo. Dentro del desarrollo de software el objetivo es obtener un producto o servicio de software a partir de distintas entradas. Entre estas entradas tenemos los requerimientos que nos indican los límites del alcance, las personas que son un aspecto fundamental y el principal recurso en el desarrollo, y también hay materiales, energía, equipamiento, hardware.

A diagram of a process

Description automatically generated

Las **personas** hacen uso de las **herramientas, equipos, procedimientos y métodos** y a partir de estos producen un producto de software. Dentro de este proceso están definidas las responsabilidades, actividades y herramientas a partir de las cuales las personas transforman los requerimientos de software. Diferencia entre estos dos tipos de procesos:

* **Proceso definido:** dentro de estos procesos todo lo que nosotros necesitamos para llevar a cabo de software tiene que estar descripto ahí y es definido de antemano por gente del negocio que no forma parte del equipo de desarrollo. Esto es difícil de aplicar al software ya que el software no es definible. Se asume que podemos repetir el proceso indefinidamente y obtener los mismos resultados. Hacer ajuste y cambios son costosos porque el proceso viene de la organización ya que, al venir de la organización, un cambio en el proceso es más difícil porque tiene que haber un acuerdo organizacional. La resistencia al cambio resulta en procesos obsoletos. Ejemplo: el PUD.
* **Procesos empíricos:** en estos procesos centramos el foco en la experiencia de no solo los usuarios, sino que también del negocio. El conocimiento y la toma de decisiones se adquieren a través de la experiencia y la experimentación. Estos procesos de utilizan para abordar proyectos en los que los detalles exactos y los resultados futuros no pueden preverse con precisión debido a la complejidad del software y la incertidumbre que conlleva. Se basa en tres pilares fundamentales:
  + **Transparencia:** todos los aspectos del proceso son visibles para todos los miembros del equipo y las partes interesadas. Permite que todos comprendan lo que está sucediendo y fomenta la comunicación y la colaboración.
  + **Inspección:** implica una revisión regular y detallada de los resultados y el progreso del trabajo en el proyecto. Se hacen revisiones, demostraciones, pruebas para evaluar el trabajo realizado. Ayuda a identificar problemas y oportunidades de mejora de manera temprana.
  + **Adaptación:** implica tomar medidas basadas en la información recopilada durante la inspección. Si se detectan problemas o si se identifican oportunidades para mejorar, el equipo debe ser capaz de ajustar su enfoque y realizar cambios en el proceso. Se debe poder responder a la incertidumbre y al cambio en el entorno del proyecto.

El ciclo de vida es una abstracción. Me define todos los estados del proyecto por los que va a pasar y en qué orden desde que este comienza hasta que termina. Son una serie de pasos a través de los cuales un producto o procesos progresa. El proyecto y el producto ambos tienen ciclo de vida. **El ciclo de vida de un proyecto de software es una representación de un proceso**. Grafica una descripción del proceso desde una perspectiva en particular. Nos define las fases del proyecto y el orden en el cual se llevan a cabo las mismas. El ciclo de vida de un proyecto termina cuando se genera el producto.

Para el producto, una vez que este se genera el ciclo de vida no termina si no que se debe mantener mientras el producto exista. Hasta que el producto no se deseche, es decir donde ya no requiere más mantenimiento, el ciclo de vida no termina. Hay tres ciclos de vida básicos:

* **Secuencial:** se basan en ejecutar una etapa después de la otra sin retorno generalmente. Ejemplo ciclo de vida en cascada.
* **Iterativo – incremental:** hago iteraciones para ganar información y convertir esa información en un incremento que será incorporado en el sistema. Al producto lo voy a entregar en distintas iteraciones que cada iteración va a tener funcionalidades limitadas, se la voy a presentar al cliente y este me va a dar información para que yo pueda mejorar, voy a ir incrementando funcionalidad a medida que avanzo en iteraciones. Los procesos empíricos implementan este tipo de ciclo de vida ya que permita la adaptación y mejora del proceso.
* **Recursivo:** se basa en tomar una característica especifica y en una iteración me centro en esa funcionalidad, en la siguiente iteración en otra y así sucesivamente. Se utiliza para casos particulares como proyectos de alto riesgo.

## El **proceso** es una implementación del **ciclo de vida** (que es una abstracción que nos guían en cuales son las etapas y su orden) que tiene como objetivo la creación de un **producto** no ambiguo y alcanzable.

**Componentes de proyecto de desarrollo de software:**

software en contexto


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

* El **proceso** es una plantilla, una definición abstracta que se materializa a través de los proyectos, en donde se adapta a las necesidades concretas del mismo. En un proceso se expresa en forma teórica lo que se debe hacer para hacer software y debo adaptarlo al proyecto o terminar de armarlo. Los procesos toman como entrada requerimientos y a través de un conjunto de actividades estructuradas obtienen como salida un producto o servicio de software.
* El **proyecto** es la unidad organizativa o de gestión, y necesita de personas, recursos y procesos para poder existir. A través de este, administro los recursos que necesito y las personas que van a formar parte de este, para obtener como resultado un producto de software.

# Gestión de productos:

La **gestión de productos** se refiere al proceso de planificación, desarrollo y supervisión de un producto a lo largo de su ciclo de vida, desde su concepción hasta su retiro del mercado. Hay varias razones por las cuales decidimos crear un producto, como, por ejemplo, satisfacer a nuestros clientes, tener masividad de usuarios conectados, obtener dinero, pero también algo muy importante es realizar un producto con visión de cambiar al mundo. Sabemos que muchas veces invertimos mucho tiempo en construir algunas características de software que no van a ser utilizadas frecuentemente o nunca. Debemos elegir conscientemente estas características útiles y eliminar el desperdicio.

A diagram of a pyramid

Description automatically generated

Esta **pirámide** nos muestra la evolución de los productos de software:

* La base de la pirámide comprende la funcionalidad de nuestro producto de software, aquí se ubican todas las características que el software debe tener si o si para ser útil y cumpla con el propósito esperado. Esto viene acompañado de que el software sea confiable, es decir que opere de manera consistente y que sea usable, es decir que tenga en cuenta la experiencia del usuario para que este tenga una experiencia placentera al utilizarlo. En la punta de la pirámide se ubican aspectos relacionados con otras cuestiones de la visión del producto que son difíciles de alcanzar. Hablamos de un producto conveniente, placentero y significativo.
* El desafío surge de 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.

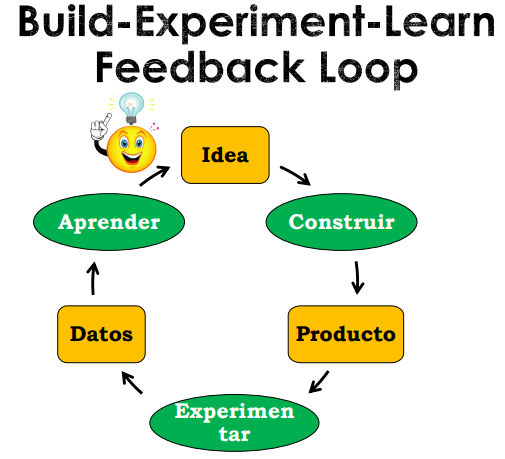
Necesitamos desarrollar el mínimo producto o mínima característica para saber si el producto que vamos a construir va a cubrir las expectativas del usuario o las propias al desarrollar el producto. Planteamos una hipótesis y con un mínimo desarrollo, un mínimo esfuerzo y validarlas para hacer que el desperdicio tienda a 0. Si nosotros desarrollamos un producto que resuelva nuestra hipótesis con un producto completo con muchas funcionalidades y recursos utilizados, corremos el riesgo de que al lanzarlo al mercado no tenga la aceptación que esperábamos ya que puede ser algo que no haya demanda o que no cumpla las expectativas del usuario. Es por esto por lo que tratamos de minimizar el esfuerzo para poder construir un producto que nuestro usuario y poder averiguar si es que la hipótesis plantea algo que el usuario necesita.

* Concepto de **MVP** o **mínimo producto viable** donde trabajamos con el mínimo producto que necesito obtener. Contiene las características mínimas que debe tener el producto para que los posibles usuarios lo puedan evaluar y decirnos si es atractivo o no. Puede ser run video o una presentación, no necesariamente tiene que ser código.
* El **objetivo** del MVP es **comprobar que la hipótesis** que yo plantee sobre el producto es válida o si debo hacer cambios. Se genera un espacio de aprendizaje gracias a la participación de los usuarios que envían retroalimentación sobre si el producto cumple con las expectativas. Si obtengo opiniones distintas de cada usuario (a algunos les gusta, a otros no, algunos piensan que le falta y otros que le sobra) nos indica que tengo que seguir trabajando sobre la visión del producto.
* La hipótesis puede moverse o modificarse, mi MVP se va moviendo hacia el foco en el cual los clientes se van a interesas. Debemos definir un nuevo MVP basado en este llamado MVP2 y lo que hacemos es pivotear hasta encontrar específicamente cual es el producto que nosotros queremos llegar a construir.
* Concepto de **MVF** o **característica mínima viable**, se trata de una característica mínima que se puede construir e implementar, utilizando recursos mínimos para probar la utilidad de esta. Buscamos la característica que por sí misma ofrezca un valor agregado a el producto. Un MVP puede estar formado por una sola o varias MVF.
* Concepto de **MMF** o **característica mínima comercializable** define las características mínimas que debe tener el producto pueda ser comercializable. Que voy a tener que construir para que el producto sea rentable. Es importante que sea mínimo ya que no tiene sentido construir el producto completo hasta no saber si va a ser rentable o no y también para poder sacarlo rápidamente al mercado (no te roben la idea o se vaya la ilusión del producto)
* Concepto de **MRF** o **características mínimas del release** que se trata del release del producto que tiene le conjunto de características más pequeño posible. El incremento más pequeño que ofrece un valor nuevo a los usuarios y satisface sus necesidades actuales. Lo obtengo en varias iteraciones hasta que tengo una cierta cantidad de características a lanzar.
* Concepto de **MMP** o **producto mínimo viable** es el primero que sea hace para poder comercializarlo.

Error común es confundir un MVP o con MMF o con el MMP

* Esto puede llevar a la entrega de un producto sin considerar si realmente satisface las necesidades del cliente, ya que el enfoque se coloca en obtener ganancias en lugar de validar la utilidad del producto.
* Enfatizar en la parte mínima y no darle bola a la parte viable del producto. El producto entregado no es de calidad suficiente para proporcionar una evaluación de si los clientes utilizaran el producto.
* Entregar lo que se considera un MVP y no hacer más cambios a ese producto (sin importar el feedback que reciban)

**Build experiment learn feedback loop**, aspecto muy importante de esta metodología es ciclo de Feedback o aprendizaje. Nuestro objetivo es realizar un producto que el cliente va a querer utilizar, para esto necesitamos alinear los esfuerzos constantemente con las necesidades reales de los clientes. A través de este ciclo podemos aprender sobre las características reales de los clientes y alinearlas a nuestro desarrollo.



La idea es tener rápidamente y con el menor esfuerzo tener un MVP que podemos validar y retroalimentar la hipótesis para entrar en el ciclo de aprendizaje. Como minimizar desperdicio y esfuerzo: simplificar, evitar construcción excesiva, evitar promesas, entender que trabajo adicional a lo que se necesita puede ser un desperdicio. El **MVP** está dentro de la **fase de experimentar**.

* Concepto de audacia cero, usamos el MVP para experimentar con los primeros usuarios en el mercado verificando mi hipótesis probando todos los elementos disponibles comenzando por los más riesgosos.
* Concepto de supuestos saltos de fe se les dice a las cuestiones más riesgosas del plan o concepto del nuevo producto.



# Filosofía Ágil con Req. Agiles, user-stories y estimaciones:

La **gestión ágil** de los requerimientos apunta a trabajar y a poner el esfuerzo en lo que brinde valor y a trabajarlo en el momento adecuado.

Enfoque ágil plantea de manera diferente la definición de los requerimientos.

Los **requerimientos** se irán descubriendo a medida que avanza el proyecto y se busca definir solo lo suficiente para poder comenzar a trabajar con lo mínimo necesario y obtener retroalimentación lo antes posible para mejorar continuamente.

* Se expresan en forma de **historias de usuario** que son breves descripciones de los requisitos del cliente que se escriben en lenguaje natural y se enfocan en las necesidades del usuario. Definen el alcance de cada iteración del proyecto y se prioriza en función del valor que aportan al producto.

Razón por la cual los requerimientos se definen a medida que avanza el proyecto para tratar de determinar solo lo que es suficiente ya que es muy común en los productos de software haya funcionalidades desarrolladas que no se utilicen, por ende, hay desperdicio**. El enfoque ágil busca eliminar ese desperdicio, priorizando aquellas funcionalidades que le aportan valor al cliente**.

* Para ello se introduce una figura clave en el proceso de desarrollo que es el **Product Owner** o dueño de producto, que es el encargado de identificar las necesidades del negocio y priorizar los requerimientos del producto del que más necesitan al que menos necesitan.
* Se utiliza un concepto llamado **Product Backlog** que nos muestra en una lista ordenada la prioridad de los requerimientos, en donde los que requerimientos más importantes se ubican en la parte superior de la lista mientras que los requerimientos de prioridad baja en la parte inferior de la lista. Tiene forma de pila, en donde las historias de usuario se van acomodando una encima de la otra según su prioridad, arrancamos a trabajar con los requerimientos que estén en el tope de la lista con un mayor nivel de detalle y continúo en orden decreciente.
* Es importante aclarar que a medida que el proyecto avanza hay requerimientos que pueden ser removidos, o agregados, o que su prioridad cambie.
* **Concepto Just inTime:** analizo un requerimiento cuando lo necesito y analizo con detalle cuando tenga el suficiente valor. Esto nos ayuda a nosotros a evitar desperdicios, ya que no tiene sentido especificar requerimientos que no voy a necesitar ahora y que posiblemente puedan cambiar a medida que pasa el tiempo. A medida que avanzamos vamos a ir encontrando los requerimientos y describiéndolos conforme haga falta.

Las **User Stories** a través de una historia expresan una necesidad que tiene el usuario, no son especificaciones detalladas de los requerimientos. No necesitan de mantenimiento, puede ser descartadas y son entrada a la documentación que se desarrollara después. Las 3 C de una US:

* **Conversación:** es la parte más importante de la user story y que no queda escrita en ningún lugar. Es el momento en el cual el equipo de desarrollo dialoga con el Product owner y capturamos las necesidades del cliente. Obtenemos los detalles para que podamos trabajar en esa historia.
* **Confirmación:** condiciones que se tiene que cumplir para que una user story sea realizada con éxito. Se expresa dentro de la tarjeta como criterios de aceptación, los cuales formaliza lo que el product owner solicita y definen los límites de una user story. Vienen acompañados de las pruebas de aceptación que van a validar si los criterios se cumplen o no. Se utilizan pruebas tanto positivas y negativas para comprobar los criterios.
* **Card o tarjeta:** es la parte visible de la user story. Es donde escribimos o expresamos algo que nos indique cual es la historia de usuario, de que se trata y su valor de negocio. Muy importante que contenga estas tres cosas: quien realiza la acción, que necesita ese quien y para que lo necesita o que valor recibe.

Forma de expresar las historias de usuario:

Como <Quien> quiero <Que necesita> para <Valor de negocio que recibe>

La US modelan o representan distintas cosas:

* Una necesidad del usuario
* Una descripción del producto
* Un ítem de planificación (cuando priorizamos las historias)
* Token para una conversación: acordarse de que hay que hablar sobre esto
* Mecanismo para diferir una conversación: definir otras conversaciones que debemos tener a futuro.
* Esta planteada como una **implementación en porciones verticales,** no solo describen funcionalidad, sino que debemos implementarla desde la lógica de interfaz de usuario hasta como resuelvo la lógica en la base de datos. Tenemos que entregar software funcionando, cada user story tiene un poco de cada capa para que pueda funcionar, solo desarrollas lo que te hace falta para que funcione.

**Criterios de aceptación** definen los límites de la user story. Ayudan a que el product owner responda los criterios que necesita para que la US provea valor al negocio. No contiene cuestiones de implementación, siempre definen una intención y no pensamos en cómo resolverlo. Decimos que define los limites ya que ayuda al equipo de desarrollo a agregar la funcionalidad necesaria, y no agregar de más ni de menos. Se utiliza las palabras debe o puede, dependiendo si es obligatorio u opcional. Están dentro de la tarjeta de la historia.

**Pruebas de aceptación** nos dan la confirmación de que lo que estamos haciendo (si todas las pruebas pasan) lo que el PO espera recibir al finalizar el producto. Agregamos también lo que esperamos que suceda cuando ejecutamos la historia de usuario (falla o pasa).

* **Definition of Ready** nos permite definir que tiene que cumplir la US para asegurarnos de que esta lista para ser implementada. Se acuerda con el equipo y se arma una checklist con todas estas condiciones y se van tildando a medida que se van cumpliendo. Si se cumple con todo entonces esa US esta lista para ser implementada, si no se cumple con alguno tengo que seguir trabajando con la user story.
* **Modelo INVEST** si se cumple con este modelo la US esta lista para ser implementada. No importa el orden, generalmente lo prioriza el PO.
  + Independiente: no depende de otra US. No me afecta a otra US si la muevo dentro del product backlog.
  + Negociable: especificamos el que en la user story y no detallamos el cómo, o sea la implementación de la user story.
  + Valuable (de valor): debe tener un valor concreto para el cliente.
  + Estimable: tengo que poder definir cuanto esfuerzo me va a llevar hacer la US.
  + Small: debo poder estimarla.
  + Testeable: tengo que poder demostrar que fueron implementadas.
* **Definition of Done** también la define el equipo y se valida con un checklist donde especificamos todas las características que debe tener la US para poder ser presentada al cliente o al Product owner.

Existen distintos **niveles de abstracción** para los requerimientos agiles ya que estos evolucionan con el tiempo. Por esto podemos tener distintos tipos de US que pueden entrar en el product backlog.

* Las **Épicas** son historias de usuario muy grandes, que en principio son así de grandes porque están en un lugar de la pila en donde todavía no fueron detalladas. Como todavía no debo implementarlas las dejo plasmadas como una gran US sabiendo que en algún momento voy a tener que llegar a disolverla, detallarla y hacerla más pequeña.
* Los **Temas** pueden ser incluso hasta más grandes que una épica, conjunto de US relacionadas que defino en función de un título para recordarme que todo lo que está incluido en ese tema lo voy a tener que tratar en algún momento. Nivel de abstracción más grande que las épicas ya que todavía no han sido detalladas aun, pueden ser iniciativas o propuestas de valor.
* Un tipo especial de User Stories son las **Spikes** que sirven para quitar riesgo o incertidumbre de otro requerimiento, de otra US o de alguna faceta del proyecto que nosotros queremos investigar. Sirven para: familiarizar al equipo con una nueva tecnología o dominio, analizar un comportamiento de una historia compleja y poder dividirla en piezas manejables, ganar confianza frente a riesgos tecnológicos, enfrentar riesgos funcionales donde no está claro como de be interactuar el sistema con el cliente. Deben ser estimables, demostrables y aceptables. Son una excepción.
  + **Spikes técnicas:** sirven para investigar enfoques técnicos en el dominio de la solución. Se hacen uso de estas cuando el equipo necesita una comprensión más fiable sobre alguna tecnología a aplicar antes de desarrollar una nueva funcionalidad en un tiempo fijo.
  + **Spikes funcionales:** se usan cuando hay cierta incertidumbre respecto de cómo el usuario interactuara con el sistema. Son mejor evaluadas con prototipos para obtener retroalimentación.

Al planificar un proyecto de software debemos realizar **estimaciones** sobre el uso de recursos, costos y otras características que vamos a implementar durante el proceso de desarrollo de producto. Estimar en el contexto de un proyecto de software se refiere al proceso de predecir o calcular cuánto tiempo, esfuerzo y recursos serán necesarios para completar dicho proyecto. Si estamos al inicio del proyecto la incertidumbre es mayor, pero a medida que avanzamos con el proyecto la incertidumbre disminuye ya que tenemos más información. No hay que esperar hasta tener toda la información para estimar, debo realizar la primera estimación y luego estas estimaciones se van ajustando a medida que avanzamos.

* Lo primero que estimamos es el **Tamaño** que nos referimos a que tan grande va a ser el trabajo que tengamos que hacer. Tenemos que contar funcionalidades, requerimientos, cantidad US y cantidad de casos de uso. También se puede utilizar datos históricos que nos van a servir para comparar el tamaño del software con algún software ya desarrollado que tenga características similares.
* Estimamos también el **Esfuerzo** que se refiere a la cantidad de horas lineales que voy a necesitar para construir el software. Cuento las horas.
* **Calendario** o sea una fecha de entrega
* **Costos y presupuesto**.

Existen diferentes técnicas de estimación que nos ayudan a disminuir esa incertidumbre, importante aclarar que no van a hacer que la incertidumbre desaparezca, pero se va a contextualizar.

* **Métodos basados en la experiencia:** se basa en la experiencia pasada del equipo en proyectos similares para determinar la estimación de esfuerzo y tiempo requerido para este nuevo proyecto. (agregar lo de juicio experto)
* **Métodos basados exclusivamente en recursos:** basada en la cantidad y el tipo de recursos necesarios para llevar a cabo un proyecto.
* **Métodos basados exclusivamente en el mercado:** basado en el costo de proyectos similares en el mercado para determinar la estimación de esfuerzo y tiempo.
* **Métodos basados en los componentes del producto o en el proceso de desarrollo:** identificación y descomposición de los componentes del software y el proceso de desarrollo para estimar el esfuerzo y tiempo requerido para cada componente.
* **Métodos algorítmicos:** utiliza métodos matemáticos y estadísticos para determinar la estimación de esfuerzo y tiempo requerido para un proyecto.

Estimaciones en ambientes agiles se deben tener en cuenta ciertas cuestiones:

* Si las estimaciones se usan como compromisos son muy peligrosas y perjudiciales para cualquier organización. Por eso las estimaciones no son un plan.
* Lo más beneficioso de las estimaciones es el proceso de hacerlo.
* Puede servir como una gran respuesta temprana sobre si el trabajo planificado es factible o no.
* Puede servir como una gran protección al equipo.
* Las US van a ser estimadas usando una unidad de medida de tamaño relativo llamada **Story Point**. Es un número que se le da a cada historia de usuario que resulta de la combinación de estimar su incertidumbre, esfuerzo y complejidad (no se tiene en cuenta la duración del proyecto, o sea no es una medida basada en el tiempo). Son específicas para cada equipo, por eso son relativas, y se estima de forma colectiva, es decir, nos interesa la visión/opinión de cada uno. Se utiliza la serie de Fibonacci.
* Para estimar US debemos definir una User Story llamada la canónica que nos sirve como referencia para estimar las otras US.
* Para poder estimar una US debo estimar el **complejidad, esfuerzo e incertidumbre**:
  + **Complejidad:** Evalúa cuán complicada resulta la tarea, considerando los desafíos y obstáculos que implica su realización.
  + **Esfuerzo:** Cuantifica las horas ideales requeridas para llevar a cabo la tarea, excluyendo distracciones y otras responsabilidades laborales, lo que proporciona una estimación precisa del tiempo de dedicación necesario.
  + **Incertidumbre:** Refleja el grado de falta de claridad en aspectos técnicos o relacionados con el negocio. Esta incertidumbre puede tener un impacto notable en la estimación, ya que no podemos medir con precisión lo que no conocemos
* Concepto de **Velocity** es una medida del progreso de un equipo durante el desarrollo de un proyecto y sobre una iteración. Se suma los Story points asignados a cada US que el equipo completo durante la iteración que se está midiendo. Nos sirve para corregir errores de estimación.
* Método **Póker Planning** consiste en que todos los miembros del equipo de desarrollo (son ellos porque son los que saben resolver la tarea) seleccionen una carta que representa en valor numérico que puede ser la posible estimación de la US, la deben colocar boca abajo. Una vez que todos los miembros han elegido, se dan vuelta las cartas simultáneamente y en donde cada participante debe justificar el puntaje que le haya otorgado a la US. Luego se lleva a cabo una nueva ronda de estimación hasta que se llegue a un acuerdo al valor del Story Point de cada US.

# Software configuration management o scm

La **gestión de configuración de software o SCM** es una disciplina de soporte cuyo propósito es mantener la integridad del software lo largo de toda su vida. El **software** es información estructurada con propiedades lógicas y funcionales que fue creada y es mantenida de varias formas, y es confeccionada para ser procesada por computadoras en su estado más desarrollado. El software no es solo código, sino que cuenta con varias cosas que hacen que el software sea más que el código, el software es toda información que se puede presentar en varios niveles de abstracción. Los sistemas de software siempre cambian durante su desarrollo y uso, conforme se hacen cambios al software se crean **nuevas versiones** del sistema. Los sistemas pueden considerarse un conjunto de versiones donde cada una de ellas debe mantenerse y gestionarse para no perder la trazabilidad de los cambios que se introducen en cada versión. Al presentarse esto cambios es fundamental **mantener la integridad** del producto de software y para esto nos es de gran ayuda la gestión de configuración de software.

El **objetivo del SCM** es mantener la integridad del software y para esto debemos establecer ciertas cuestiones:

* Identificar y documentar características técnicas y funcionales d ellos ítems de configuración.
* Controlar los cambios de esas características identificadas y documentadas.
* Registrar y reportar esos cambios.
* Verificar correspondencia con los requerimientos.

La gestión de configuración de software **resuelve problemas** tales como: la perdida de componentes, perdida de cambios, regresión de fallas, doble mantenimiento, superposición de cambios, cambios no validados. Esto lo hace a través identificar la configuración en un momento dado, controlar sistemáticamente sus cambios y mantener su integridad y origen.

Como vemos nosotros que la integridad del producto es consistente y por ende podemos garantizar de que nuestro producto sea de buena calidad:

* Cumple con las necesidades del cliente, hace lo que el cliente espera que haga.
* Permite el seguimiento del producto y todos los ítems de configuración a lo largo del ciclo de vida. Existen vínculos o conexiones entre los ítems de configuración que nos dejan ver a nosotros en donde es que va a impactar un cambio en un ítem.
* Cumple con requerimientos de performance y requerimientos no funcionales.
* Cumple con expectativas de costo, no solo por parte del cliente, sino que también el equipo de desarrollo está conforme con la remuneración.

Conceptos generales:

* **Ítem de configuración:** aquellos artefactos que forman parte del producto o proyecto que pueden ser almacenados en un repositorio y tiene una versión en un momento de tiempo. Pueden sufrir cambios y se desea conocer su estado y evoluciona lo largo del ciclo de vida (producto o proyecto). La única manera de identificarlos unívocamente es a través de la versión. Algunos ejemplos: requerimientos, casos de uso, diagramas de diseño, código, casos de prueba, ERS, arquitectura de software, etc.
* **Repositorio:** es el contenedor de los ítems de configuración, se encarga de mantenerla historia de cada ítem con sus atributos y relaciones, además es usado para hacer evaluaciones de impacto de los cambios propuestos. Una o varias bases de datos. Tiene una estructura que nos ayuda a mantener la integridad y el orden, y también cuestiones como la seguridad, las políticas de backup, los controles de acceso. Pueden ser de dos tipos: **centralizado**, en donde un servidor contiene todos los archivos con sus versiones (falla el servidor y todo muere). El otro tipo es el **descentralizado** en donde cada cliente tiene una copia exactamente igual del repositorio completo (más complicado llevar control sobre el mismo).
* **Versión:** instancia de un ítem de configuración que difiere de otras instancias de este ítem. Cuando controlamos las versiones hacemos referencia a la evolución de un solo ítem. Es un punto particular en el tiempo de ese ítem de configuración.
* **Variante:** es una versión de un ítem de configuración que evoluciona por separado. Un producto de software puede adoptar distintas formas dependiendo del lugar donde se instale.
* **Configuración de software:** conjunto de todos los ítems de configuración con su versión especifica. Es la sumatoria de todos los ítems de configuración que tiene en un momento determinado, equivale a una foto de todos los ítems de configuración con su versión en un momento del tiempo.
* **Línea base:** es un conjunto de ítems de configuración que han sido construidos y revisados formalmente, de manera que pueden ser tomados como referencia para demostrar que se ha alcanzado cierta calidad en ellos y que sirve como base para desarrollos posteriores. Es lo mismo que decir que una configuración de software que ha sido formalmente revisada y aprobada que sirve como base para desarrollos futuros.
  + Para hacer cambios en una línea base hay que seguir **protocolo formal de control de cambios** que es definido por el comité de control de cambios, que permite definir el procedimiento a seguir para manejar peticiones de cambios y en caso de aceptarse, acordar nuevamente los parámetros y comunicar los cambios a todo el equipo, para que tomen la nueva línea base como modelo a seguir.
  + Sirve para hacer **Rollback** o sea ir atrás en el tiempo y reproducir el entorno de desarrollo en un momento dado del proyecto.
  + Dos tipos de línea base, primero **de especificación** estas no cuentan con código. El otro tipo es **operacion**al en donde si hay una versión del producto como código ejecutable. La primera línea base operacional corresponde con la primer Release.
* **Rama o Branch:** es un conjunto de ítems de configuración con sus correspondientes versiones, que permiten bifurcar el desarrollo de un software para experimentación, resolución de errores o desarrollar el mismo software, pero para otra plataforma. Las ramas se pueden integrar y se fusionan a la rama principal o descartar.

Existen actividades relacionada a la gestión de configuración que forman parte de un plan de gestión de configuración. La actividad de auditoria es la única que solo se hace en el entorno tradicional.

* **Identificación de ítems de configuración:** identificación univoca de cada ítem y en el equipo se definirán políticas y reglas de nombrado y versionado para todos ellos. También se debe definir la estructura del repositorio y la ubicación de los ítems de configuración dentro de esa estructura. Esta identificación y documentación proveen un camino que une todas las etapas del ciclo de vida del software, lo que permite a los desarrolladores controlar y velar por la integridad del producto, como así a los clientes evaluar esa integridad.
  + **Ítems de producto:** ciclo de vida más largo y se van a mantener mientras el producto exista.
  + **Ítems de proyecto:** el plan del proyecto tiene un ciclo de vida del proyecto.
  + **Ítems de iteración:** planes de iteración, cronogramas de iteración y reporte de defectos.
* **Control de cambios:** surge de cuando se quiere hacer un cambio en algún ítem de configuración de una línea base. Es un procedimiento formal sobre los ítems de configuración que pertenecen a la línea base que involucra a diferentes actores y se evalúa el impacto del cambio. Lo hace el comité de control de cambios. El comité evalúa el cambio y se autoriza o rechaza, si se autoriza deben definir qué día va a suceder. ¿Quién forma parte del comité? El PO, el arquitecto, el líder del proyecto, analista funcional, testigo, el cliente si el cambio es algo muy grande, o sea cualquiera que se pueda ver afectado por ese cambio. Permite tener una trazabilidad entre los ítems de configuración y ante un cambio saber cuáles ítems están afectados por este.
* **Auditorias de configuración:** son controles autorizados a realizar por el equipo de desarrollo en un momento determinado, donde un auditor externo al equipo analiza las líneas base, las cuales permiten analizar en un momento determinado cual es el estado del software y si están cumpliendo las pautas que plantea esta disciplina de SCM. Los auditores tienen que ser independientes y objetivos. Se hace mientras el producto se está construyendo y su objetivo es que el producto tenga calidad e integridad.
  + **Auditoria física de configuración:** asegura que lo que está indicado para cada ítem en la línea base se ha alcanzado realmente.
  + **Auditoria funcional de configuración:** evaluación independiente de los productos de software, controlando que la funcionalidad y performance reales de cada ítem de configuración sean consistentes con la especificación de requerimientos.