

Unidad 2: Gestión Lean Ágil de Productos de Software

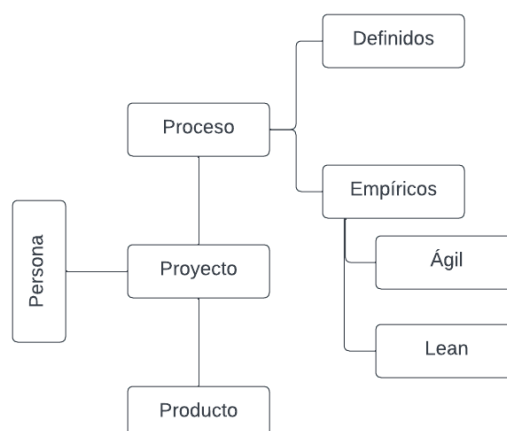
| | |
|-------------|--|
| 🕒 Created | @August 17, 2023 10:35 AM |
| 📁 Class | ISW |
| 📁 Type | Teórico |
| 📎 Materials | 02 Rqs Agiles USER STORIES.pdf |
| ☑ Reviewed | <input type="checkbox"/> |

Introducción

- Técnica para trabajar requerimientos funcionales
- Las personas hacen los proyectos → Las **personas** tienen un rol fundamental en el éxito o fracaso del proyecto.

Procesos

Nivel más abstracto. Está descrito a **niveles teóricos** que se supone de hacer. Se define en términos de roles, y se debe ajustar e instanciar en cada proyecto.



- **Definidos:** enfoque estructurado y bien documentado para la gestión y el desarrollo de software. En este caso, el proceso se sigue de manera predecible

y repetible, lo que ayuda a garantizar la consistencia y la calidad en el producto final.

Desagregamos un conjunto de actividades y definidos que hacer, como, quién y demás. Permiten ser predecibles y tener la ilusión de que tienen **control**.

La información previa, sirve de base para las próximas experiencias.



Intentan ser la **expresión de completitud**. Todo lo que necesitamos está allí: roles, entradas, salidas por actividad, métricas, herramientas. Definido de antemano por personas diferentes a las que van a hacer el trabajo.

- PUD

El proceso se ha **malinterpretado** a lo largo del tiempo. El proceso venía definido desde afuera.

- RUP

- **Empíricos:** enfoque de gestión y desarrollo que se basa en la adaptación continua y la mejora a través de la retroalimentación constante. En el contexto del desarrollo de software, esto implica que los equipos ajustan y modifican su proceso en función de las lecciones aprendidas y la información recopilada durante el desarrollo.

Los ciclos deben ser **cortos**, por lo que se basan en ciclos de vida **iterativos**.



Basados en **tres pilares**:

- Inspección
- Adaptación
- Transparencia: la información es de **todos**, el código es del equipo, se deben blanquear las situaciones actuales. Claridad, el objetivo y la situación de avance debe ser **visible**. Pasamos de conocimiento implícito a explícito que se puede **compartir**.

Todos los **frameworks** dan **pautas**. No arman un proceso, por lo que no podemos llamarlos **metodologías**.

Ciclos cortos donde puedo determinar que puedo **corregir**.
En la retroalimentación esta la **experiencia**.

- Ágil

Fuerza muy importante que exige que **las decisiones las toma el que hace el trabajo**.

- Técnica: **User Stories**

- Surge del Software

- Lean

- Surge de la Industria Automotriz

Manifiesto Ágil

Ideología con un conjunto definido de principios que guían el desarrollo del producto.

Ágil

Balance entre ningún proceso y demasiado proceso. La diferencia inmediata es la exigencia de una menor cantidad de documentación, sin embargo, no es eso lo más importante:

- Los métodos ágiles son **adaptables** en lugar de predictivos.
- Los métodos ágiles son **orientados a la gente** en lugar de orientados al proceso

Valores Ágiles

Los valores ágiles son los cuatro conceptos fundamentales que están en el corazón del manifiesto ágil.



Valoró una cosa sobre la otra. **No significa que no debemos hacer ambas.**

Cómo enfoque debo valorar lo que valora el **cliente**.

1. Individuos e interacciones sobre los procesos y las herramientas

Las dos cosas son importantes, pero las primeras son prioritarias.

Individuos e interacciones: priorizamos la interacción y el vínculo.

2. Software funcionando sobre la documentación extensiva.

- a. Interacciones cortas que se pueden poner en producción.

Generemos la información que haga falta, en ningún momento se dice que no.

Se prioriza el funcionamiento del software. Importancia en la permanencia de la información.

3. Colaborar con el cliente sobre negociación contractual.

Hacer parte del proyecto al cliente. Revisar los cambios de requerimientos, son puntos de fricción.

4. Responder al cambio sobre seguir un plan.

Debemos tener una actitud de tener respuesta. El trabajar sobre la marcha, permite que los cambios sean más fácilmente bienvenidos.

Principios Ágiles

Los principios ágiles son una serie de 12 directrices que complementan y expanden los valores ágiles. Estos principios proporcionan una guía más detallada para la implementación de los valores ágiles en la práctica.

Product Owner → de negocio. Es un representante del cliente. Debe tener capacidad de decisión y tener en cuenta la actualidad y el futuro.

1. **Nuestra principal prioridad es satisfacer al cliente a través de la entrega temprana y continua de software con valor.**
2. **Aceptamos los cambios en los requisitos, incluso en etapas tardías del desarrollo. Los procesos ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.**
3. **Entregamos software funcional con frecuencia, con preferencia por intervalos cortos, en un periodo de semanas a meses, con una preferencia por los periodos más cortos.**
4. **Los responsables de negocio y los desarrolladores trabajamos juntos de forma cotidiana a lo largo del proyecto.**

5. **Construimos proyectos en torno a individuos motivados. Les damos el entorno y el apoyo que necesitan, y confiamos en que harán bien su trabajo.**
6. **El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara.**
7. **El software funcional es la medida principal de progreso.**
8. **Los procesos ágiles promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios debemos ser capaces de mantener un ritmo constante de forma indefinida.**
9. **La atención continua a la excelencia técnica y al buen diseño mejora la agilidad.**



La calidad del producto **no se negocia.**

10. **La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es esencial.**



La maximización del trabajo no hecho.

11. **Las mejores arquitecturas, requisitos y diseños emergen de equipos que son autoorganizados.**



Equipos autoorganizados y autogestionados, el equipo toma las decisiones. No existen los **jefes.**

12. **A intervalos regulares, el equipo reflexiona sobre cómo ser más efectivo para luego ajustar y perfeccionar su comportamiento en consecuencia.**

Requerimiento Emergentes: requerimientos que aparecen mientras el producto está naciendo, creciendo y evolucionando. Representan un 50% de los requerimientos.

User Stories

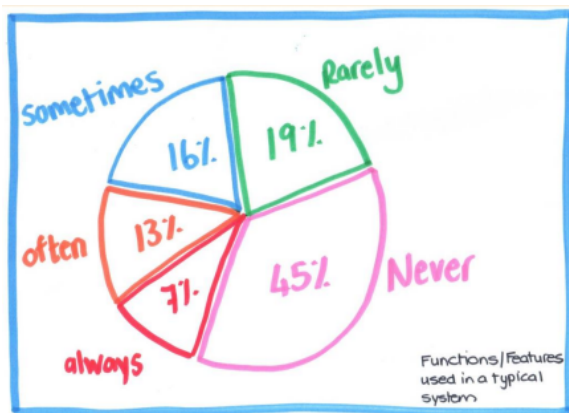


Lo importante es el **valor de negocio** que agregamos al cliente. El software es un **medio para un fin**.

Nivel de abstracción más alto, a nivel de usuario y negocio.

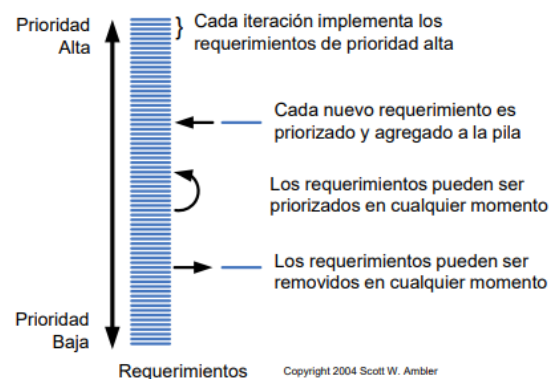
- Descripción corta que describe una necesidad del usuario.
 - Terminología del usuario. Se busca que lo escriba el mismo usuario.

Priorización. Debemos poder entender cuando parar. **Lo bueno es suficiente.** Definida por el product owner.



Existe un porcentaje muy grande de funcionalidad que **no se utiliza**.

Product Backlog: gestionar y priorizar los requisitos, características y elementos de trabajo que deben ser desarrollados en un proyecto. Es una **lista dinámica que contiene todas las funcionalidades, mejoras y tareas** que se espera implementar en el producto. Propiedad del **product owner**.



Just in Time: refiere a la idea de **entregar o producir algo en el momento exacto en que se necesita**, evitando la creación excesiva o el almacenamiento innecesario. Cuando se aplica en el desarrollo ágil, se busca minimizar el trabajo que no agrega valor al producto y maximizar la eficiencia.

Eliminar Desperdicios: busca **identificar y eliminar** cualquier actividad, proceso o recurso que **no agrega valor** al producto o servicio final. Esto se logra al enfocarse en maximizar la eficiencia y la calidad al tiempo que se minimiza el desperdicio en todas sus formas.

Cara a Cara: la importancia del enfoque cara a cara en Agile radica en su **capacidad para mejorar la comunicación, la colaboración, la retroalimentación y la adaptación**, lo que conduce a un desarrollo más eficiente y a la entrega de productos de mayor calidad que satisfacen las necesidades del cliente de manera más efectiva.

Tipos de Requerimientos:

1. Dominio del Problema:
 - a. Requerimientos de Negocio
 - b. Requerimientos de Usuario
2. Dominio de la Solución:
 - a. Requerimientos de Software

Triangulo de Hierro



Amarillo habla de enfoque tradicional, mientras que azul habla de enfoque ágil.

Dimensiones en las que se mueve un proyecto en función de las decisiones que tenemos que tomar cuando estamos trabajando.

Refleja la idea de que existe una **relación de equilibrio** entre el alcance del proyecto, el tiempo que se tiene para completarlo y los recursos disponibles para ello. Si uno de estos elementos cambia, los otros dos también se verán afectados.

Enfoque Tradicional: En el enfoque tradicional de gestión de proyectos, representado a menudo por el modelo en cascada, el triángulo de hierro comprende tres elementos: alcance, tiempo y costo. En este modelo, el alcance es definido al

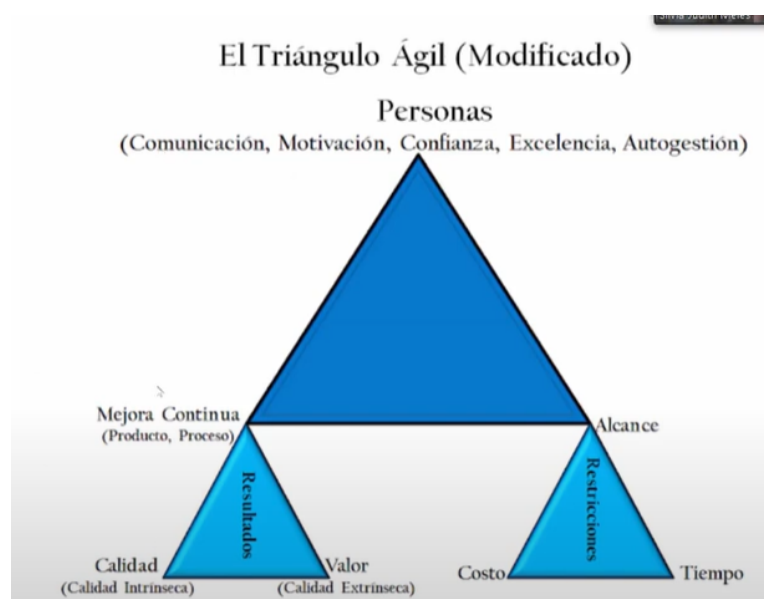
inicio del proyecto y se considera como algo fijo y estable. El tiempo se planifica con antelación, y el costo se asigna según el alcance y el tiempo estimado.

Enfoque Ágil: En los enfoques ágiles, como Scrum, el triángulo de hierro sigue incluyendo alcance, tiempo y costo. Sin embargo, el enfoque es diferente. En lugar de considerar el alcance como algo fijo y definitivo al inicio del proyecto, se priorizan las características en función de su valor y se dividen en elementos más pequeños llamados "historias de usuario".

El tiempo en Agile se divide en intervalos llamados Sprints, que tienen una duración fija y generalmente corta. Durante cada Sprint, el equipo trabaja en un conjunto de historias de usuario priorizadas.

El costo no solo se refiere al presupuesto financiero, sino también al esfuerzo del equipo y los recursos disponibles.

Las personas son lo más importante, la intención de mejora continua y luego las demás opciones.



User Stories

Descripción breve y simple de una funcionalidad o característica que un usuario necesita del producto. Son **multipropósitos**.

No son una especificación de requerimientos. Son una necesidad del usuario, una descripción del producto, un ítem de planificación, un token para una conversación y un mecanismo para diferir una conversación.

Artículo de Lectura Obligatoria:

<https://www.cgl.ucsf.edu/Outreach/pc204/NoSilverBullet.html>



La parte más difícil de hacer software es decidir que software queremos hacer. **REQUERIMIENTOS**

Partes de una User Story

1. Conversación: No es visible.



De un lado tendremos la tarjeta - Del dorso tendremos la confirmación/pruebas de identificación de usuario.

2. Tarjeta (Card):

3. Confirmación:

Sintaxis

1. ¿Quién?

2. ¿Qué necesita?

3. ¿Para qué? → Lo más importante, implica el valor de negocio. El valor de negocio nos permite priorizar.



En **ningún** momento se especifica el **cómo**. En lenguaje común de negocio.

Ejemplo:

Buscar Destino por Dirección

Como **Conductor** quiero **buscar un destino a partir de una calle y altura** para **poder llegar al lugar deseado sin perderme**.

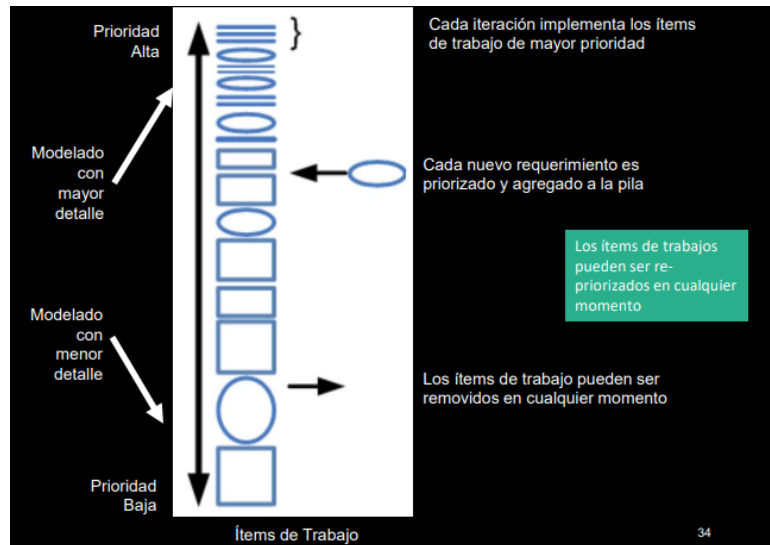
Expresión mínima de la Card

Se recomienda acompañar esta **card** con **criterios de aceptación**.

Frase Verbal: Nombre corto. Tiene que ver con el qué. Permite un acceso más rápido a las user stories. **No reemplaza la user story.**

El **product owner** debe priorizar las historias en el **product backlog**.

Existen **niveles de granularidad** distintos en las historias. El nivel utilizado es definido por el equipo.



| Story 1 | Story 2 |
|----------------|---------|
| GUI | |
| Business Logic | |
| Database | |
| | |

Porciones Verticales: las user stories debe ser una unidad funcional de software. Deben poder ser construidas. Están formadas por una parte de cada capa.

Modelado de Roles

Tarjeta que explica los roles de usuario. Se anotan las características generales del perfil de usuario.

Proxies Product Owner: persona designada para desempeñar el rol de Product Owner en ausencia del Product Owner principal. **Se deben evitar. ROL NO TÉCNICO.**

Criterios de Aceptación

Definen **límites** para una user story. Es un requerimiento no funcional.

Información concreta que nos permiten evaluar si lo que hacemos se va a aceptar o no.

Formalización de las cosas que el product owner nos va a exigir que estén especificados en las user stories. Se escriben a nivel de usuario, deben ser específicos y concretos.

Ejemplo

Buscar Destino por Dirección

Como **Conductor** quiero **buscar un destino a partir de una calle y altura para llegar al lugar deseado sin perderme.**

Criterios de Aceptación:

- La altura de la calle es un número.
- La búsqueda no puede demorar más de 30 segundos.

Derivan las **pruebas de aceptación**: debemos tener una que lo cumpla y otra que no.

Las pruebas de aceptación son un acuerdo con el product owner.

Describimos lo que debemos probar y como debería funcionar el software.

- Pruebas de Usuario**
 - ❑ Probar buscar un destino en un país y ciudad existentes, de una calle existente y la altura existente (pasa).
 - ❑ Probar buscar un destino en un país y ciudad existentes, de una calle inexistente (falla).
 - ❑ Probar buscar un destino en un país y ciudad existentes, de una calle existente y la altura inexistente (falla).
 - ❑ Probar buscar un destino en un país inexistente (falla).
 - ❑ Probar buscar un destino en País existente, ciudad inexistente (falla).
 - ❑ Probar buscar un destino en un país y ciudad existentes, de una calle existente y demora más de 30 segundos (falla).

Si al presentar la historia las pruebas de aceptación funcionan como debería, la story se debería aprobar.



¿Dónde están los detalles?

En la conversación y en las pruebas de aceptación. Estos expresan detalles no expresados en la card y nos permiten decidir si se acepta la card.

Definición de Listo/Definition of Ready (DoR)

Se aplica a la user story, si está aprobado la misma está a una iteración de ser implementada. Si no, quedará más abajo en el product backlog.

Es definido por el **equipo**.

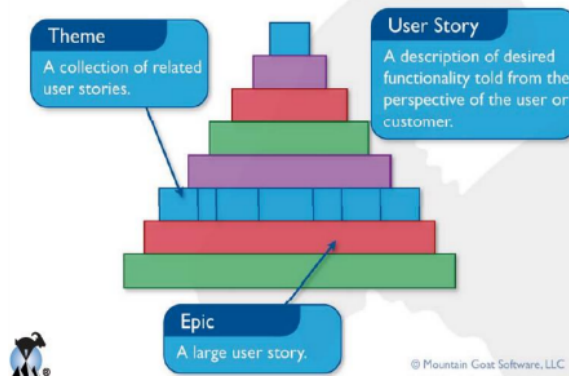
Es un checklist que tiene una base de mínima (**acuerdo mínimo**), un Invest Model.

INVEST Model:

- **Independent:** cada user story en vertical se puede aplicar en cualquier orden, decidido por el product owner.
- **Negotiable:** la historia está escrita en términos de qué y no de cómo.
- **Valuable:** debe tener un porque o para que, generando valor de negocio.
- **Estimable:** debe poder estimarse un costo/valor numérico.
 - Si una user no es estimable debido a su complejidad e incertidumbre se llama **spike/item de investigación**.
- **Small:** deben ser consumible en una iteración, **Sprint**. Poder comenzar y terminar la user en el mismo sprint.
- **Testable:** puedo demostrar que la user se implementó en los términos que el product owner exige.

Mountain Goat - Diferentes Niveles de Abstracción

Stories, themes and epics



Epica - Tema - User Story

Requerimientos Agiles

| | |
|-------------|--|
| 🕒 Created | @August 24, 2023 10:48 AM |
| 📁 Class | ISW |
| 📁 Type | Teórico |
| 📎 Materials | 04 Estimaciones Ágiles.pdf |
| ☑ Reviewed | <input type="checkbox"/> |

Requerimientos Ágiles

- Foco en el **valor de negocio**. Software como un medio para agregar valor de negocio.
- Los requerimientos se encuentran de a poco. Comenzamos con lo mínimo y necesario y a lo largo de tiempo vamos mejorando.

Requerimientos Agiles: necesidades, funcionalidades o características que deben ser desarrolladas en un proyecto utilizando metodologías ágiles, como Scrum. A diferencia de los enfoques tradicionales, donde los requerimientos suelen estar completamente definidos al inicio del proyecto, en Agile, los requerimientos son flexibles y se adaptan a medida que avanza el desarrollo y se obtiene feedback.

Trabaja a nivel de requerimiento de negocio o requerimiento de usuario. Trabajamos sobre el dominio del problema.

Stakeholders: son todos los que toman decisiones con respecto al producto.

Los principios ágiles relacionados a los requerimientos son el 1, 2, 4, 6 y 11.

- Parto de una visión de producto, que debe responder al valor del negocio, va a ser el punto de partida de la primera visión del producto.
- Está visión va a dar lugar al **product backlog**.
 - Como mínimo debe tener la cantidad suficiente de **user stories** para poder ejecutar la primera iteración del producto.
 - **Se define como una fila priorizada de user stories.**



Just In Time: Diferir decisiones hasta el último momento responsable.

Enfoque que busca entregar, producir o realizar actividades exactamente cuando son necesarias, minimizando el tiempo y los recursos desperdiciados en el proceso.

La **esencia de los requerimientos ágiles** está basada en varios principios del manifiesto, la mejor comunicación es cara a cara y los mejores requerimientos vienen de los equipos autoorganizados. El entregar software funcionando facilita la retroalimentación.

La gestión ágil es una **gestión binaria**, sólo 0 y 1 (estás o no embarazada)

- ¿Tenés la historia implementada o no?

Definition of Ready (DoR)

Definition of Ready: conjunto de criterios predefinidos que una historia de usuario o tarea debe cumplir antes de ser considerada apta para su inclusión en el **Sprint backlog** de un equipo ágil.

Se construye de mínima con el INVEST Model.

Sprint Backlog: lista detallada y específica de las historias de usuario o tareas individuales que el equipo de desarrollo ha comprometido completar durante un Sprint en la metodología ágil Scrum. El Sprint Backlog se crea a partir del Product Backlog.

| | | |
|-------|-------|------|
| To Do | Doing | Done |
|-------|-------|------|

Definition of Done (DoD)

Definition of Done: conjunto de criterios acordados y predefinidos que una historia de usuario o tarea debe cumplir para considerarse completamente finalizada y lista para ser **mostrada** al cliente o usuario. Modelo en **checklist**.

El equipo se pone de acuerdo en el tablero de **Definition of Done** y **Definition of Ready**. Código comentado, código documentado, usuario documentado, debe estar subido a repositorio, se deben hacer revisiones técnicas del código.

Ágil: iteraciones, tiempo y excelencia técnica no negociables. Se negocia la cantidad de requerimientos a generar.



Si utilizo user muy grandes y se compromete todo el equipo a hacerla, pero no llegan, no se **entrega nada**.

Product Backlog

La mayor responsabilidad de Product Owner es poder priorizar los requerimientos que le parecen más importantes.

- Mientras más abajo vemos la lista, nos encontraremos con temas y épicas que al subir en prioridad se dividirán en stories.
- Esta permitido cambiar el orden cuando se desee. No se puede cambiar luego de que la user pase el DoR.
- Va cambiando su condición dependiendo del estado del producto.

Spike

Spike: nivel de incertidumbre con una historia que no permite hacer una correcta estimación.

- **Técnica:** indefiniciones tecnológicas. Investigación y comprensión de aspectos técnicos o tecnológicos específicos.
- **Funcional:** indefiniciones del negocio. Se debe seguir trabajando con el **Product Owner**. Investigar y comprender aspectos funcionales o de negocio de una historia de usuario o de una funcionalidad específica.



Si decidís que parte del equipo debe hacer la investigación. La spike va en un sprint backlog y las user entran luego de resolver el problema.

Estimable - INVEST

Estimar: refiere a la capacidad de evaluar o estimar el esfuerzo y el tiempo requeridos para completar una tarea, historia de usuario o funcionalidad específica.

Obtiene como resultado algo **cuantitativo**. Tienen un valor de probabilidad asociado.

Tiene asociado un nivel de **incertidumbre** debido a la **predicción** necesaria para poder crearla. Se planifica con una estimación como base, hay que estar preparado a hacer una reestimación.

Previamente se estimaba sobre líneas de código, debido a que es importante poder hacer cuantificable el software para estimar.

Estimar basado en experiencia no necesariamente funciona, debido a que puede y suele variar debido a los equipos y proyectos variantes.

Estimaciones Ágiles

Características:

Se aprende del proceso. Hay que estar dispuestos al cambio.

- Son **relativas**:
 - Las personas no saben estimar en términos absolutos
 - Se prefiere la comparación: más rápido y fácil.
 - Se realizan estimaciones por **comparación**.
- Foco en la **certeza** y no en la precisión:
 - En los enfoques tradicionales se utiliza la precisión, pero las mismas no suelen ser cumplidas.
- **Diferir** las decisiones hasta el último momento responsable:
 - No es necesario estimar todo el producto.
 - Momentos de Estimación:
 - Generalizada (talles de remera). Específico en el Product Backlog.
En esta etapa, antes de que comience un Sprint, el equipo y el Product Owner trabajan juntos para estimar el esfuerzo relativo de las historias de usuario y las tareas en el Product Backlog.
 - Durante la Sprint Planning.
Durante la Sprint Planning, el equipo selecciona las historias de usuario o tareas del Product Backlog que se comprometerá a completar en el próximo Sprint. En esta fase, el equipo realiza estimaciones más

detalladas de cuánto esfuerzo se necesita para cada elemento seleccionado.

- Se estiman las historias.
- Se estiman como se llevarán a cabo las historias.



Poker Planning - Utiliza Juicio Experto Delphi, donde muchos estiman.

- Estima el que **hace el trabajo**:
 - En las tradicionales estima el líder del proyecto, que luego no tiene un rol activo en el trabajo. Utilizando juicio experto puro.
 - El grupo autoorganizado permite que los que estiman, deben ser los que llevan a cabo el trabajo.

Story Point

Story Point: unidad de estimación de user stories. Representa el tamaño de la historia de usuario. Unidad homogeneizadora, que permite comparar.

Utiliza la **serie de Fibonacci**, el número siguiente se obtiene en base a los dos anteriores.

Serie de Fibonacci: Secuencia de números en la que cada número es la suma de los dos números anteriores.

La serie tiene un crecimiento exponencial, al igual que la complejidad del software.

0, 1, 2, 3, 5, 8, 13

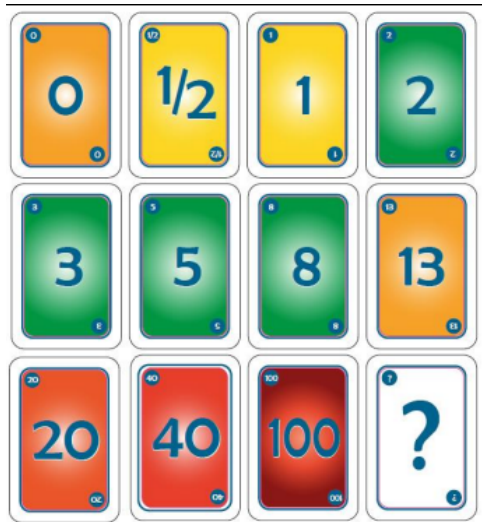
En el contexto de estimaciones de puntos de historia en metodologías ágiles, la serie de Fibonacci se utiliza comúnmente como una escala para asignar valores a la complejidad o el esfuerzo relativo de las historias de usuario. Los números de la serie de Fibonacci se convierten en unidades de medida para las estimaciones, como "story points".

Poker Estimation

Estima el equipo completo.

User Story Canónica: elemento que se utiliza para **comparar**.

Se debe estimar bajo el concepto de completar todos los pasos necesarios para llegar la **Definition of Done**.



1. Todos tenemos nuestras cartas de estimación. Se presenta la primer user story.
2. Cada uno hace su estimación en privado. Se espera a que todos terminen, para no influir en la decisión del otro.
3. Se muestran todas las cartas.
4. Comienza una explicación personal del razonamiento detrás. Se empiezan por los extremos. Se **escucha**.

5. Se hace otra ronda individual de nuevo. Considerando la opinión de los demás
6. Se repite hasta que se converge en un acuerdo. Por lo general no se hacen más de dos iteraciones.
 - a. Si es necesario alguien toma una decisión sobre la estimación.

La canónica se utiliza como punto de comparación para poder estimar, es propia del proyecto sobre el que estamos estimando. Se suele utilizar una canónica de uno, conlleva el problema de que sucede si es más sencillo.

Una canónica de tres es recomendada, al haber espacio para más complejidad y sencillez.

Se debe esperar que cada sprint realice users de entre 1 y 5 puntos.

Story Point

Se divide en tres dimensiones una story al momento de asignar un story point:

1. **Complejidad:** dificultad de implementación desde un punto de vista técnico. Incluye consideraciones como la integración con sistemas existentes, la complejidad de la lógica de negocio, la necesidad de utilizar tecnologías nuevas o desconocidas, y la presencia de desafíos técnicos.

2. **Esfuerzo:** relaciona con la cantidad de funcionalidades o interacciones que debe tener la historia de usuario. Cuantas más funcionalidades o interacciones estén involucradas, mayor puede ser la complejidad funcional. También considera si la historia involucra flujos de trabajo complejos o excepciones. Siempre debe estar relacionado al esfuerzo de construir que, el tamaño y la complejidad.



Tamaño vs Esfuerzo vs Calendario. Las estimaciones basadas en tiempo son más propensas a errores debido a varias razones. Se debe tener en cuenta el esfuerzo en horas de trabajo ideal y de ahí estimar el calendario.

3. **Duda/Incertidumbre:** refiere a la cantidad de incertidumbre que rodea la historia de usuario. Si hay incertidumbre sobre cómo implementar ciertos aspectos, si hay dependencias desconocidas o si la historia está relacionada con áreas del sistema que no están completamente definidas, esto puede aumentar el riesgo y, por lo tanto, la estimación de puntos de historia.

Valor de Homogenización: llevamos las tres dimensiones a un sólo número y la vuelvo el story point. Enfoque o técnica que busca convertir las tres dimensiones (complejidad técnica, complejidad funcional y riesgo/incertidumbre) de una historia de usuario en un único punto de estimación, como un número de puntos de historia. Esta técnica puede emplearse en metodologías ágiles para simplificar la asignación de puntos de historia al considerar varias complejidades en una única medida.

Velocidad

Métrica de progreso de un equipo. Se calcula sumando el número de story points que el equipo completa durante la iteración. NO SE ESTIMA.

Se cuentan los story points de las User Stories que están completas. Permite corregir errores de estimación.



Debido a que se trabaja con fechas de entrega fijas, es mejor trabajar con piezas de producto más pequeñas. Recordar que se trabaja de manera binaria, está terminado o no lo está.

Gestión de Productos

| | |
|------------|---------------------------|
| 🕒 Created | @August 31, 2023 10:47 AM |
| 📁 Class | ISW |
| 📁 Type | Teórico |
| ☑ Reviewed | <input type="checkbox"/> |

Gestión de Productos

Práctico

Salir al mercado con una versión del producto que justifique el gasto que implicaría desarrollar todo el producto.

- **MVP:** producto mínimo viable →
 - Identificar MVP
 - Escribir User Stories
 - Estimarlas y Justificarlas

Proceso - Proyecto - Producto

- **Proceso:** Conjunto de pasos o actividades que se realizan de forma secuencial para lograr un objetivo determinado. Da una definición teórica de que es lo que debería hacerse para hacer software. Toman como entrada requerimiento y obtienen como salida un producto de software, mediante actividades especificadas.
- **Proyecto:** Esfuerzo temporal que se lleva a cabo para crear un producto, servicio o resultado único. Necesita de personas, recursos y un método de como hacerse el trabajo para cumplir con el objetivo que se plantea. Debe tomar del proceso lo que parece que es necesario en base a un criterio de necesidad. Unidad de gestión, medio por el cual gestiona los recursos para obtener como resultado un producto o servicio. Cada tarea debe tener un entregable.

- Alcance: trabajo que hay que hacer para cumplir con el objetivo. Se toma del proceso, y son tareas que cumplir.
- **Producto:** Artículo producido, cuantificable y que puede ser un elemento terminado o un componente. Software: conocimiento empaquetado. La definición, bd, diseño, caso de uso son software. Todos los entregables de los proyectos son software.



Las empresas que no tienen como core hacer software subestiman el producto.

¿Por qué creamos productos?

1. Para satisfacer a los clientes.
2. Para tener muchos usuarios logueados.
3. Para obtener mucho dinero.
4. Realizar una gran visión, cambiar el mundo.

¿Qué características realmente utilizamos de un producto de software?

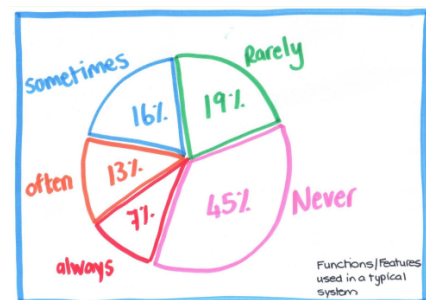


La mayoría de las características de un producto de software no se usan.

No hay ningún indicador que hable de entregar el 100% en la primer entrega.

Time to market: el tiempo que tarda una empresa en llevar un nuevo producto o servicio al mercado. Se mide desde la concepción del producto o servicio hasta su lanzamiento.

El TTM es un factor importante para el éxito de un negocio, ya que puede dar a una empresa una ventaja competitiva sobre sus rivales.



Evolución de los Productos de Software

La funcionalidad del producto tiene que ver con la utilidad y no la usabilidad.

Usabilidad implica cuanto disfruto o sufro utilizando este producto de software.



Nos debemos focalizar en la gente y no en las tareas



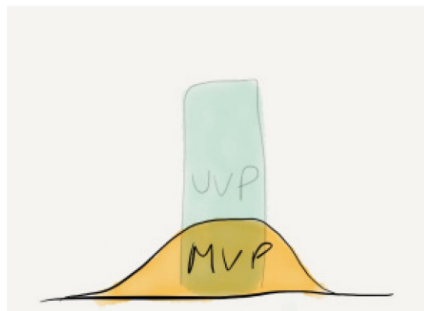
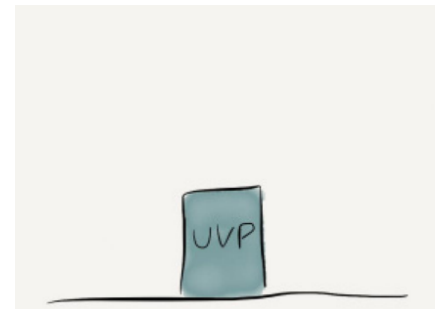
Un **producto de software significativo** le cambia la vida a la gente.

MVP - MVF - MMF



Comprender un producto nuevo tiene una hipótesis de valor único

UVP idea de un producto único y con valor



El siguiente paso es crear un producto mínimo viable para probar su hipótesis

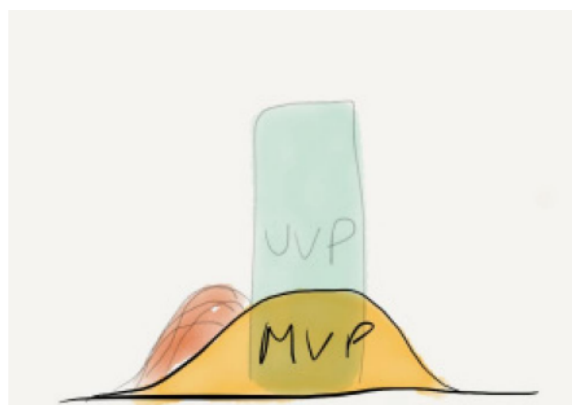
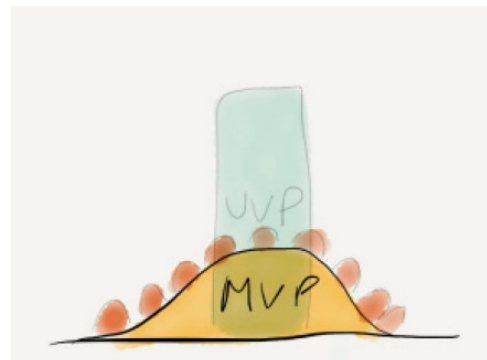
El MVP se crea para validar una hipótesis: va a ser querido por los clientes.

Poco tiempo y poca plata dedicada a esta etapa.

Al no encontrar un mercado específico para el producto, debido a las recomendaciones recibidas.

Minimal Marketable Feature: las características mínimas de un producto para poder comercializarlo.

Salgo al mercado para que el cliente lo evalúe.



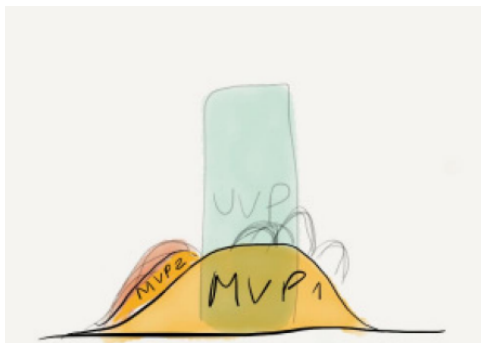
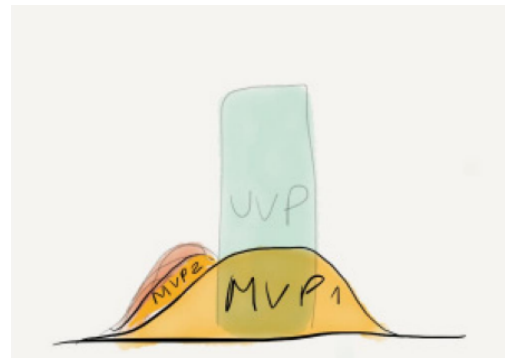
Si por el contrario ves cada vez más respuestas apuntando al MISMO X entonces tiene sentido revisar la hipótesis de Cliente/Problema/Solución

Minimal Viable Feature: hay veces que con una sola función podemos probar en el mercado.

Ejecución de un PIVOT. Construcción de un MVP2 centrado en la nueva hipótesis basada en el aprendizaje reciente de desarrollo de clientes generado por el anterior MVP.

Estamos creando el producto. Validamos si es el producto que el mercado necesita.

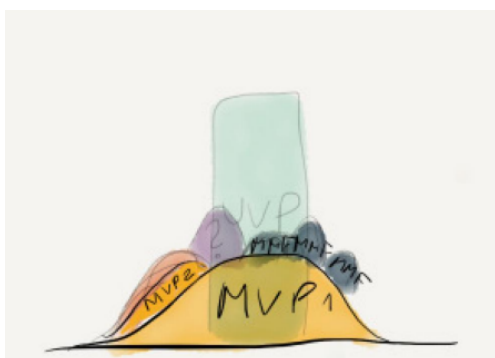
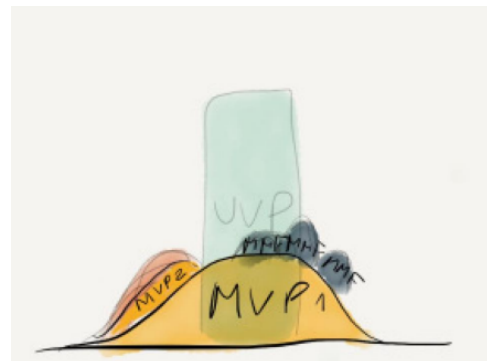
Desviación del producto.



En función del feedback recopilado y la investigación de su gestión de productos, tiene un par de áreas que potencialmente pueden traer este crecimiento. Algunos de ellos por cierto amplían su propuesta de valor única y algunos hacen que su producto actual sea más robust

Debemos sacar al mercado con algo.

Buscamos agregar MMF para agregar valor. Supone una alta certeza de que existe valor en esta área



Ahora tu hipótesis se centra en una característica en lugar del producto. Tienes un área con alto potencial pero también alta incertidumbre

La forma de afrontarlo es crear una función "pionera": MVF (Característica Mínima Viable). La característica mínima que aún puede ser viable para uso real y aprendizaje de los usuarios reales.

Si la MVF resulta exitosa (hit gold), puede desarrollar más MMF en esa área para tomar ventaja (si eso tiene sentido).



- El producto se cultiva en mercados inciertos al intentar varios MVP.
- Cuando se logra ajustar el producto en el mercado de productos se combinan MMF y MVF según el nivel de incertidumbre del negocio / requisitos en las áreas en las que se está enfocando.
- Si bien los MVP / MMF / MVF son atómicos desde una perspectiva empresarial (no puede implementar y aprender de algo más pequeño) pueden ser bastante grandes desde la perspectiva de la implementación.
- El dinosaurio carpaccio se obtiene cortando cada una de esas piezas en pequeñas porciones destinadas a reducir el riesgo de ejecución / tecnología (normalmente se denominan User Stories).



Esas porciones más pequeñas pueden tener un valor comercial tangible o no.



Minimal Viable Product (Producto Mínimo Viable)



Minimal Viable Feature (Característica Mínima Viable)

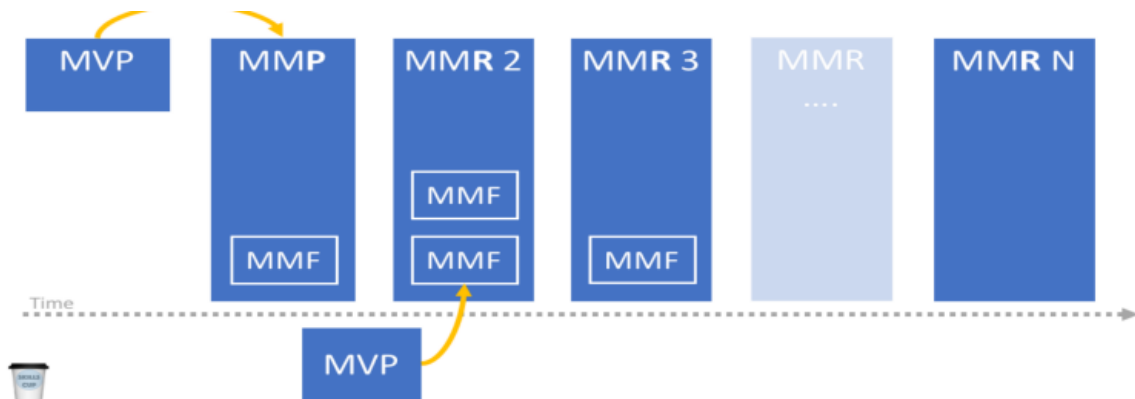


Minimal Release Feature (Características Mínimas del Release)



Minimal Marketable Feature (Característica Mínima Comercializable)

- **MVP** (Mínimo producto viable) es un prototipo que contiene las características y funcionalidades más básicas de un producto. El objetivo de un MVP es validar la idea de un producto y recopilar comentarios de los clientes para mejorarlo.
- **MVF** (Mínimo producto valioso) es un prototipo que contiene las características y funcionalidades más importantes de un producto. El objetivo de un MVF es crear un producto que satisfaga las necesidades de los clientes y tenga éxito en el mercado.
- **MRF** (Mínimo producto requerido) es un prototipo que contiene las características y funcionalidades mínimas que se requieren para cumplir con un requisito o necesidad. El objetivo de un MRF es crear un producto que cumpla con un requisito específico o necesidad del cliente.
- **MFF** (Mínimo producto factible) es un prototipo que contiene las características y funcionalidades que son factibles de implementar con los recursos y el tiempo disponibles. El objetivo de un MFF es crear un producto que sea posible de desarrollar y lanzar en un tiempo razonable.



1. Si parto de un MVP, parto de una validación de una hipótesis. Superada esa hipótesis, pasamos a marketable (comercializable) MMP. Aca se puede agregar un MMF.
2. Luego pasamos a un Release MMR donde agregamos otra feature, producto con otro nivel de madurez.



La diferencia entre ellos es que MMP (Minimum Marketable Product) es el primer producto completo que se puede ofrecer a los clientes, mientras que MMR (Minimum Marketable Release) es el primer lanzamiento interno que se puede convertir en un producto comercializable

MVP



TIENE EL VALOR SUFICIENTE PARA QUE LAS PERSONAS ESTÉN DISPUESTAS A USARLO O COMPRARLO INICIALMENTE.



DEMUESTRA SUFICIENTE BENEFICIO FUTURO PARA RETENER A LOS PRIMEROS USUARIOS.





PROPORCIONA UN CICLO DE RETROALIMENTACIÓN PARA GUIAR EL DESARROLLO FUTURO.


Concepto de Lean Startup que enfatiza el impacto del aprendizaje en el desarrollo de nuevos productos.


Usted produce un producto real que puede ofrecer a los clientes y observar su comportamiento real con el producto o servicio.

Ver lo que la gente realmente hace con respecto a un producto es mucho más confiable que preguntarle a la gente que harían.

 Confundir a un MVP, **que se enfoca en el aprendizaje**, con Característica Comercializable Mínima (MMF) o con Producto Comercializable Mínimo (MMP), ambos se enfocan en “ganar”.

 El riesgo de esto es entregar algo sin considerar si es lo correcto que satisface las necesidades del cliente.

 Enfatizar la parte **mínima** de MVP con exclusión de la parte **viable**. El producto entregado no es de calidad suficiente para proporcionar una evaluación precisa de si los clientes utilizarán el producto.

 Entregar lo que consideran un MVP, y luego no hacer más cambios a ese producto, independientemente de los comentarios que reciban al respecto.

MVP vs MMF o MMP: Errores comunes

21

Valor vs Desperdicio

| Todo lo que no genera valor es desperdicio.

La productividad de un Startup no puede medirse en términos de cuánto se construye cada día, por el contrario, se debe medir en términos de averiguar la cosa correcta a construir cada día



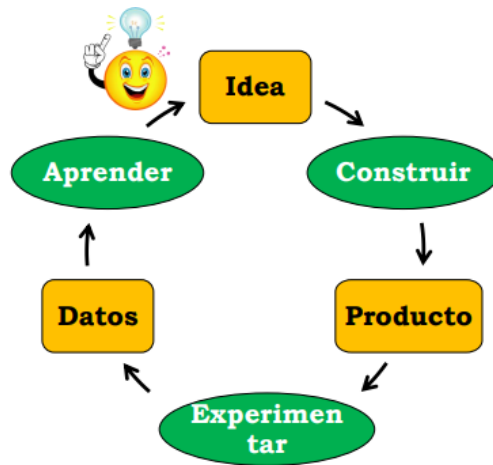
Se busca no dedicarle tiempo a esfuerzos que no generarán valor.

No se debe priorizar el valor de negocio por sobre el valor de usuario. El valor de negocio debe venir por consecuencia del valor del usuario.



El trabajo intelectual se mide con resultados concretos.

Ciclo construir-implementar-aprender



Creación de Valor

El éxito no es entregar un producto, sino entregar un producto que el cliente usará.

Se busca alinear los esfuerzos continuamente a las necesidades reales de los clientes.

Construir un MVP



Si tienes duda, simplifica. Es mejor salir con algo chico que con nada.

Audacia Cero

No hay nada para perder, esto permite que grandes cosas.

A menudo es más fácil recaudar dinero cuando tiene:

cero ingresos

cero clientes

tracción cero

que cuando tienes una pequeña cantidad de cada uno!

Cero invita a la imaginación, pero los números pequeños hacen preguntas sobre si los números grandes alguna vez se materializarán

Saltos de Fé

Son los elementos más riesgosos del plan/concepto de un startup.

Las siguientes hipótesis permiten sostener o justificar los saltos de fé:

- Hipótesis de Valor
 - Prueba si el producto realmente está entregando valor a los clientes después de que comienzan a usarlo
 - Métrica: tasa de retención
- Hipótesis de Crecimiento
 - Prueba como nuevos clientes descubrirán el producto
 - Métrica: Tasa de Referencia - Net Promoter Score (NPS)

Preparar un MVP



1. Encontrar un nicho de mercado, que tengo contenido un salto de fé.
2. Crear un roadmap, paneo de que características tendría.
3. Investigo competencia.
4. Armo un MVP
5. Testeo
6. Asegurarse que el MVP resuelve una problema
7. Salir al mercado. Recordar la ventana de tiempo que hay para salir al mercado.



Lo perfecto es el enemigo de lo nuevo.

Componentes de un Proyecto de Software

| | |
|----------------|--|
| 🕒 Created | @September 7, 2023 10:47 AM |
| 📁 Class | ISW |
| 📁 Type | Teórico |
| ☑ Reviewed | <input type="checkbox"/> |
| ☰ Bibliografía | Capítulo 7 de Desarrollos de Proyectos Informáticos Mcconell |

Componentes de un Proyecto de Software

Procesos

Definidos

Definición: Los procesos definidos son procesos que están bien documentados, estructurados y predecibles. Siguen un conjunto de reglas y procedimientos claramente definidos y pueden ser repetibles en múltiples ocasiones.

Características: Los procesos definidos son altamente estructurados y se basan en estándares y mejores prácticas establecidas. Son adecuados para situaciones donde la repetibilidad y la consistencia son esenciales, como en la fabricación de productos altamente regulados.

- Establecer de antemano todas las cosas que necesito hacer para cumplir con el objetivo del producto de software que quiero hacer.
- Establecer roles, plazos, paso a paso del proceso.
- Está definida por **otro**. Los que definen no son quienes van a realizar ese trabajo después.
- Los cambios son costosos



Motivación: tener visibilidad de donde estamos parados y que es lo que falta.

Empíricos

Definición: Los procesos empíricos se basan en la observación y la adaptación continua. Estos procesos reconocen que no se puede prever todo y que el aprendizaje a través de la experimentación y la retroalimentación es fundamental.

Características: Los procesos empíricos son más flexibles y adaptables. Se enfocan en inspeccionar y adaptar continuamente el trabajo a medida que se avanza. Se aplican en situaciones donde la incertidumbre es alta y los requisitos cambian con frecuencia.

- Lo que sirve es la experiencia de hoy, la del equipo y proyecto de este momento.
- Servirá para realizar las correcciones.
- La única experiencia que me sirve es la de este equipo en este momento en particular, no sirve la experiencia extrapolar.
 - Ágil
 - Lean

Proceso vs Proyecto vs Ciclo de Vida

Proceso: define todas las cosas que deberíamos hacer, conjunto de actividades relacionadas.

- **Definición:** El proceso de desarrollo de software es un conjunto de actividades, métodos, prácticas y pasos organizados que se siguen para diseñar, construir, probar, desplegar y mantener software de manera efectiva.
- **Propósito:** El proceso proporciona una estructura y un enfoque para desarrollar software de manera coherente y de alta calidad. Define cómo se deben realizar las tareas, quién es responsable de ellas y en qué secuencia.

Proyecto:

Guiado por un objetivo, que debe ser claro para todos los involucrados y debe ser alcanzable. Debemos sentir que se **puede lograr**.

Objetivo define el trabajo que hay que hacer para satisfacer al cliente.

Obtiene un resulta **único**. Se lo puede entender como un medio de organización.

Tiene fecha de inicio y fin, luego de ella se reasignan los recursos.

Se divide el trabajo en tareas interrelacionadas del proceso, que generan dependencias inevitables, pero facilitan el trabajo.

Elaboración gradual: se logra mediante tareas interrelacionadas.

- **Definición:** Un proyecto de desarrollo de software es un esfuerzo temporal que tiene como objetivo crear un producto de software específico, que puede ser un sistema, una aplicación o un componente.
- **Propósito:** Los proyectos de desarrollo de software tienen un alcance, objetivos y recursos definidos. Se crean para entregar un resultado específico en un plazo y con un presupuesto determinados.

Planificación del proyecto, la importancia es el acto de planificar no el resultado. El ejercicio de pensar. Decidir como equipo que cosas dejamos por escrito en documentación. Permite compartir la responsabilidad de opinion grupal.

Administración de Proyecto - Gestión Tradicional

Buscamos que el proyecto cumpla con el objetivo, administrando los recursos, organizando el trabajo, haciendo un seguimiento.

Hay un líder del proyecto que es el responsable.

Actividades: planificación y seguimiento y control.

Lider → Indica que hacer, como, quien y demás. Responsabilidades de gestión.

Plan de Proyecto → Documenta qué hacemos, cuándo, cómo y quién. Las métricas nos permiten ser objetivos.

Ciclo de Vida: orden de las tareas, cuanto vas a ejecutar de esa tarea, en qué momento. Le da información para que el proceso que es lineal se puede ejecutar en el contexto del proyecto.

- **Definición:** El ciclo de vida del software se refiere a las fases o etapas a través de las cuales pasa el software desde su concepción hasta su retiro. Estas etapas pueden variar según la metodología y el enfoque utilizados.
- **Propósito:** El ciclo de vida del software proporciona una estructura para la planificación, el diseño, el desarrollo, las pruebas y la gestión del software a lo largo de su existencia. Ayuda a garantizar que el software se mantenga y mejore de manera efectiva a lo largo del tiempo.



Ciclo de vida recursivo vs iterativos.

Ciclo de Vida Recursivo:

Definición: En un ciclo de vida recursivo, el proyecto se divide en subproyectos o entregables más pequeños, y cada uno de ellos sigue un ciclo de vida independiente y completo. Estos subproyectos pueden ser manejados por equipos separados.

Ciclo de Vida Iterativo:

Definición: En un ciclo de vida iterativo, el proyecto se divide en ciclos o iteraciones más pequeñas, cada una de las cuales implica la planificación, el diseño, la implementación, la revisión y la retroalimentación. Estas iteraciones se repiten varias veces.

Alcance

Alcance del Producto: sumatoria de todos los requerimientos funcionales y no funcionales que el producto tiene. Va guardado en la ERS. Se mide en rendimiento.

Alcance del Proyecto: tareas que se deben hacer para cumplir el objetivo del proyecto. Se guardan en el plan del proyecto. Es necesario que previamente se defina el alcance del producto. Se mide en tareas.

Estimaciones

1. Tamaño
2. Esfuerzo: horas que trabajo
3. Calendario
4. Costo
5. Recursos Críticos

Ciclo de Vida del Producto

Nace cuando se quiere crear el producto, y finaliza cuando el producto se saca del mercado. Cada ciclo de vida de producto contiene muchos ciclos de vida de proyecto

Desde que nace hasta que se discontinua. Más grande, dura más.

El ciclo de vida del producto describe las etapas y fases que atraviesa un producto desde su concepción hasta su retirada del mercado.

Objetivo: El objetivo principal del ciclo de vida del producto es gestionar y planificar el desarrollo, la comercialización, la distribución y la retirada del producto, teniendo en cuenta factores como la demanda del mercado, la competencia y las necesidades del cliente.

Ciclo de Vida del Proyecto

Representación de un proceso.

Estados de evolución del proyecto que determina el orden en el cual se pueden realizar.

El ciclo de vida del proyecto se refiere a las etapas y fases a través de las cuales pasa un proyecto desde su inicio hasta su cierre.

Objetivo: El objetivo principal del ciclo de vida del proyecto es planificar, ejecutar y controlar todas las actividades necesarias para entregar el producto, servicio o resultado que el proyecto busca lograr.

Tipos de Ciclos de Vida

El proceso definido se puede utilizar con cualquier tipo de ciclo de vida. El proceso empírico recomienda el modelo iterativo-incremental.

Secuencial

- **Características:** En un ciclo de vida secuencial, las fases del proyecto se realizan de manera secuencial, una después de la otra, y cada fase debe completarse antes de pasar a la siguiente.
- **Metodologías Ejemplo:**
 - **Modelo en Cascada:** En este modelo, las fases, como requisitos, diseño, implementación, pruebas y mantenimiento, se siguen en una secuencia lineal y rigurosa.
 - **Modelo en V:** Similar al modelo en cascada, pero con énfasis en la validación y verificación en cada fase de desarrollo. Ejemplo: V-Model.

Iterativo

- **Características:** En un ciclo de vida iterativo, el proyecto se divide en iteraciones, y cada iteración sigue un ciclo de desarrollo completo. Las

iteraciones permiten obtener feedback temprano y realizar ajustes en el proyecto.

- **Metodologías Ejemplo:**

- **Iterativo - Incremental:** RUP utiliza ciclos iterativos para desarrollar software y se enfoca en la arquitectura y el diseño temprano.

Rekursivo

- **Características:** En un ciclo de vida recursivo, el proyecto se divide en subproyectos o entregables más pequeños, y cada uno de ellos sigue un ciclo de vida independiente y completo. No genera versiones intermedias del producto, solo saca la versión al final de todo.
- **Metodologías Ejemplo:**
 - **Ciclo de Vida en Espiral:** Aunque no es puramente recursivo, tiene elementos de iteración y evaluación en cada fase. Cada ciclo en espiral puede considerarse un ciclo de vida independiente.

SCM

| | |
|------------|------------------------------|
| 🕒 Created | @September 21, 2023 11:04 AM |
| 📁 Class | ISW |
| 📁 Type | Teórico |
| ☑ Reviewed | <input type="checkbox"/> |

Gestión de Configuración del Software (SCM)

Disciplina de soporte de estilo paraguas, transversal a todo el proyecto que busca mantener la **integridad del software** en todo su ciclo de vida. Item de configuración, es cada cosa que se quiere configurar. Es responsabilidad del equipo completo.

Software es cualquier producto de trabajo que sale de cualquier actividad del ciclo de vida del proyecto y en general del producto.

Integridad del producto: implica satisfacer las necesidades del usuario, debe ser fácil y rastreable durante la totalidad de su ciclo de vida, satisfacer los criterios de performance y cumplir con la expectativa de costo.



El software cambia, y es muy fácil el cambio en el software. Buscamos que el cambio sea rastreable.

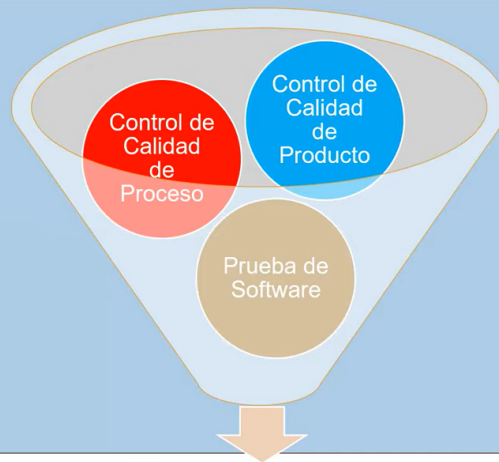
Es importante que cada item de configuración tenga su **versión**.

Versión: forma particular de un artefacto en un instante o contexto dado. El **control de versiones** refiere a la evolución de único item de configuración por separado.

Disciplinas de Soporte del Software

Administración de Configuración de Software

Administración de Configuración de Software



Aseguramiento de Calidad de Software

Se la define como una disciplina que permite que las demás disciplinas sigan su objetivo de manera inequívoca.

Propósito: establecer y mantener la integridad de los productos de software a lo largo de su ciclo de vida (del producto).

Conceptos Básicos

- **Item de Configuración:** todos y cada uno de los artefactos que forman parte del producto o del proyecto, que puedan sufrir cambios o necesitan ser compartidos entre los miembros del equipo y sobre los cuales necesitamos conocer su estado y evolución.
- **Repositorio:** contenedor de los items de configuración. Mantiene una estructura, que permite definir que item de configuración se guarda en cada lugar, esto permite facilitar el trabajo en grandes aspectos.
Da contención debido a la maleabilidad y facilidad de borrar del software.
- **Línea Base:** conjunto específico y bien definido de items de configuración, que deben especificar su estado, que se utiliza como referencia para medir el progreso, realizar comparaciones y realizar un seguimiento de los cambios a lo largo del ciclo de vida del proyecto. Se debe especificar el estado de configuración y versión de cada item que forma parte de la línea base, deben ser estables, para cambiarlos deben pasar por un proceso formal de control de cambios.

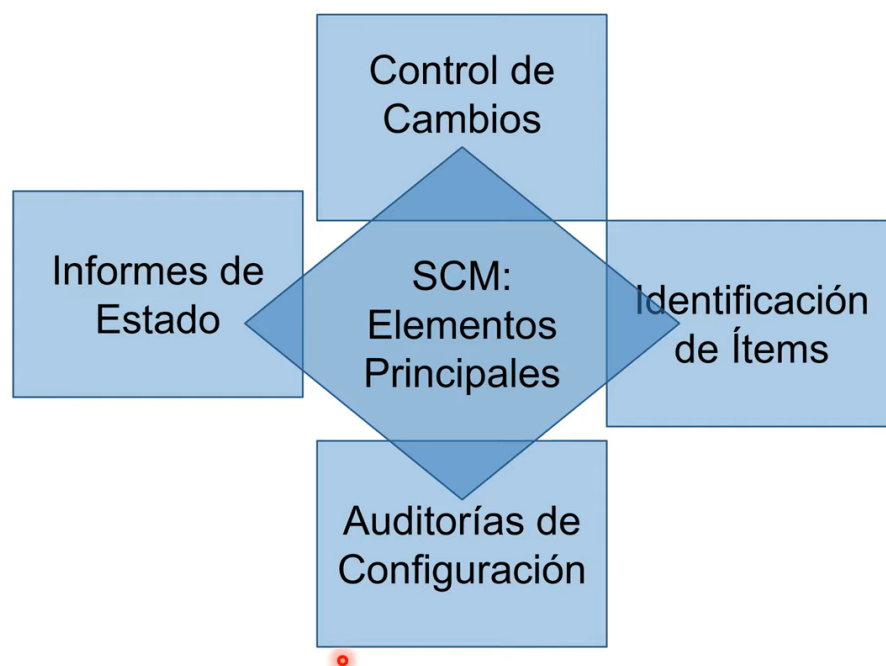
Se utiliza para poder tener un punto de referencia. Se ven como situaciones estables. Los items de la línea base no permiten una libertad completa de acción.

- De especificación: no tienen código. Si no información de ingeniería del producto.
- De productos que han pasado por un control de calidad definido previamente: código de producto.
- **Ramas:** bifurcación de los items de configuración, que permiten trabajar sobre la rama principal sin modificarla explícitamente. Cuando el item de configuración vuelva a estar estable, se agregan las modificaciones a la rama principal.

Integración de ramas: operación llamada **merge**, lleva los cambios de una rama a la rama principal. Todas las ramas deberían eventualmente integrarse a la principal o ser descartadas.

- **Configuración del Software**

Actividades Fundamentales de la Administración de Configuración de Software



- **Identificación de Items:** aquí se define la estructura del repositorio, se identifican los items unívocamente y se asignan los items a un lugar específico dentro de la estructura del repositorio. Se definen reglas de nombrado y esquemas genéricos.

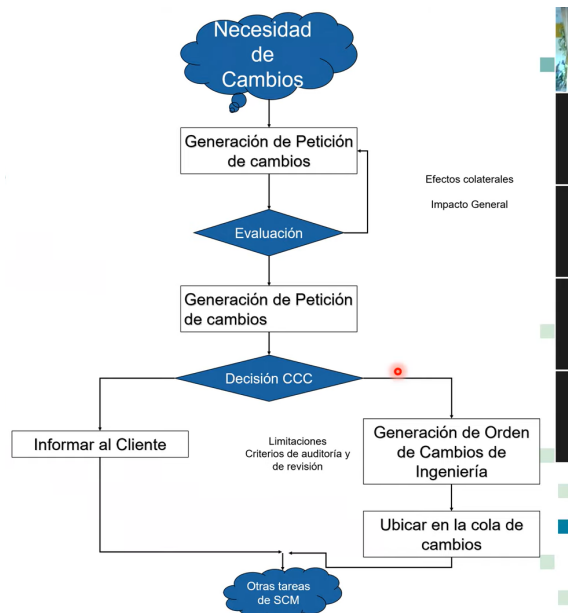
Tipos de Items de Configuración:



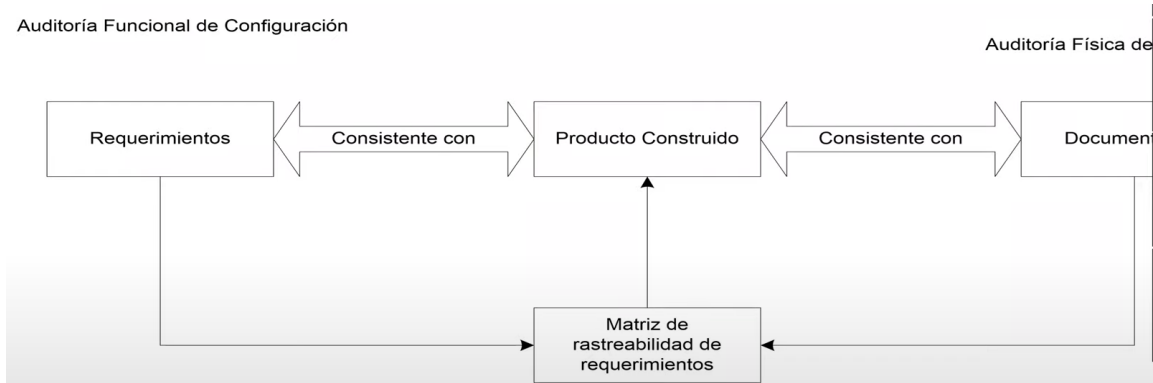
- Debido a los distintos ciclos de vida, tenemos diferentes tipos de items. La forma de identificación debe tener estos tipos en cuenta. Luego del ciclo de vida propio del tipo, los items no serán más gestionados.

- **Control de Cambios:** mantener la integridad de las líneas base. Tiene su origen en un requerimiento de cambio a uno o varios items de configuración que se encuentran en una línea base.

Es un procedimiento formal que involucra diferentes actores y una evaluación del impacto del cambio. La gente que forma parte del comité de control de cambios forma parte de las distintas áreas involucradas del desarrollo del producto, la cliente también se puede sumar.



- **Auditorías de Control de Software:**



Se audita sobre una línea base determinada del producto.

El auditor debe ser alguien externo al equipo, debido a que la auditoría debe ser una revisión independiente y objetiva del producto.

Necesitan un **plan**, debido a que es un proceso de control.

1. Física: hace verificación. Vela por la integridad del repositorio, que el mismo este, que exista en el lugar físico donde se acordó y que se sigan las reglas establecidas.
 2. Funcional: hace validación. Analiza si el producto es el producto correcto, si se responde a los requerimientos.
- **Informes de Estado:** generamos reportes para facilitar la toma de decisiones y dar visibilidad. El básico es el inventario que lista todos los items de configuración de un repositorio y demás cambios.

Plan de Gestión de Configuración: debe contener respuestas a las preguntas de cómo se realizarán las cuatro actividades básicas.

- Reglas de nombrado de los items de configuración
- Herramientas a utilizar para SCM
- Roles e integrantes del Comité
- Procedimiento formal de cambios
- Plantillas de formularios
- Procesos de auditoría



La gestión de configuración de software es una disciplina que es fundacional para mantener la integridad del producto, y que ha permitido que los equipos evolucionen en su forma de construir software. Permitiendo un trabajo más productivo y buscando automatizar la mayor parte posible de los procesos.