**Componentes de Proyecto de Desarrollo de Software**

Proceso: transforma ideas y necesidades en un producto de sw. Agrupamos un conjunto de actividades, recursos y personas para obtener un resultado.

Conjunto estructurado de actividades para desarrollar un sistema de sw. Esas actividades varían dependiendo la organización y el tipo de sistema a desarrollar. Debe ser explícitamente modelado si va a ser desarrollado.

Proceso de sw: conjunto de métodos, actividades, practicas y transformaciones que la gente usa para desarrollar y mantener sw y sus productos asociados.

Tiene tres patas: las personas, los procedimientos y métodos, y las herramientas y equipos.

Procedimientos: las tareas que vamos a hacer, como las vamos a medir.

Herramientas: son el soporte que nos ayuda a automatizar el proceso de sw.

En nuestra industria, el aporte mas importante lo hacen las personas (humano-intensivo). Lo mas caro de hacer sw es pagarles a las personas (el %80 es el costo del esfuerzo).

Esa gente debe estar capacitada (conocimiento técnico) y motivada (ganas de trabajar).

**Proyectos Definidos y Empíricos**

Los proyectos empíricos tienen un fuerte arraigo en la experiencia, basarse en algo anterior, se tiene que generar y no es fácil de conseguir.

Surgen en contraposición de los definidos, que plantean la necesidad de tener definido que vamos a hacer en cada momento, quien va a hacer que tarea, cuando la va a hacer, que artefacto va a generar, en que orden, etc. Por ejemplo, el PUD.

La diferencia entre estos dos es la intención que tienen los definidos de ser repetitivos, de manera de tener visibilidad y poder predecir que va a pasar, para tener mayor control. Además, intentan ser procesos completos, que describen la mayor cantidad de cosas posibles para obtener el producto. Tienen entradas y salidas bien definidas.

Los empíricos (por ejemplo, LEAN y ágil) son procesos que no están completos para nada, solo son practicas o guías que cubren algún aspecto, pero nunca cubre la totalidad, eso lo deja librado a elección de la decisión del equipo basada en la experiencia (como no es completo no se considera por ejemplo a SCRUM como un proceso, sino un framework o buenas prácticas). Se basa en ciclos de entrega cortos que generen rápida retroalimentación que les permitan readaptar. Otra diferencia es que esa experiencia es aplicable a ese equipo, pero no puede extrapolarse a otro, a diferencia de lo que esperan los definidos.

**Ciclos de Vida**

El ciclo de vida del desarrollo de sw es la estructura que contiene los procesos, actividades y tareas relacionadas con el desarrollo y mantenimiento de un producto de software, abarcando la vida completa del sistema, desde la definición de los requisitos hasta la finalización de su uso.

Cada esfuerzo de desarrollo de sw pasa por un "ciclo de vida", que consta de todas las actividades entre el momento en que la versión 1.0 de un sistema comienza su vida y el momento en que la última versión expira. Un ciclo de vida es un modelo prescriptivo de lo que debería suceder entre el primer destello y último aliento.

Para nuestros propósitos, la función principal de un modelo de ciclo de vida es establecer el orden en que un proyecto especifica, prototipa, diseña, implementa, revisa, prueba y realiza sus otras actividades. Establece los criterios que utilizas para determinar si se procede de una tarea a la siguiente.

No es lo mismo que proceso, un proceso define el paso a paso de lo que tenemos que hacer para obtener el producto y elige un ciclo de vida, el cual dice como se van a encarar esas actividades, en que orden y cuanto hacer de cada una. Por lo tanto, se complementan, pero no son lo mismo.

Hay ciclos de vida para los proyectos (cascada, incremental, iterativo…) y ciclos de vida para los productos. Los primeros representan un proceso, graficando una descripción del proceso desde una perspectiva particular. Especifican las fases del proceso y el orden para llevarlas a cabo. Estos últimos son más largos, inician con la idea de crear un producto de sw y termina cuando lo retiro del mercado. En un ciclo de vida de un producto puede haber n ciclos de vida de proyectos. Cada proyecto genera una versión.

Hay muchos de ciclos de vida, pero en general se agrupan en 3 tipos:

**Secuencial**: por ejemplo, cascada. Ordena rigurosamente las etapas del proceso para el desarrollo de software, de tal forma que el inicio de cada etapa debe esperar a la finalización de la etapa anterior. Al final de cada etapa, el modelo está diseñado para llevar a cabo una revisión final, que se encarga de determinar si el proyecto está listo para avanzar a la siguiente fase. Este modelo fue el primero en originarse y es la base de todos los demás modelos de ciclo de vida.

Ventajas

Realiza un buen funcionamiento en equipos débiles y productos maduros, por lo que se requiere de menos capital y herramientas para hacerlo funcionar de manera óptima.

Es un modelo fácil de implementar y entender.

Está orientado a documentos.

Es un modelo conocido y utilizado con frecuencia.

Promueve una metodología de trabajo efectiva: Definir antes que diseñar, diseñar antes que codificar.

Desventajas

En la vida real, un proyecto rara vez sigue una secuencia lineal, esto crea una mala implementación del modelo, lo cual hace que lo lleve al fracaso.

El proceso de creación del software tarda mucho tiempo ya que debe pasar por el proceso de prueba y hasta que el software no esté completo no se opera. Esto es la base para que funcione bien.

Cualquier error de diseño detectado en la etapa de prueba conduce necesariamente al rediseño y nueva programación del código afectado, aumentando los costos del desarrollo.

Una etapa determinada del proyecto no se puede llevar a cabo a menos de que se haya culminado la etapa anterior.

**Iterativo/Incremental**: por ejemplo, el que vemos en el PUD. Son aquellos en los cuales se repiten las actividades del proyecto en fases o iteraciones y en cada una de ellas se incrementa el entendimiento del producto por parte del equipo del proyecto. Las iteraciones desarrollan el producto mediante una serie de ciclos repetidos que van añadiendo sucesivamente funcionalidad al producto.

Cada iteración, se habrá completado con un conjunto de entregables. Las futuras iteraciones pueden mejorar los entregables o crear nuevos. El producto final será la acumulación de funcionalidad construidas en las interacciones.

Se opta por los ciclos de vida iterativos es necesario gestionar objetivos poco definidos o de una alta complejidad, siendo la calve para el éxito. Este tipo de ciclo de vida permite al equipo del proyecto incorporar la retroalimentación e ir incrementando la experiencia del equipo durante el proyecto.

Ciclo de vida iterativo. Responde a la alta incertidumbre del proyecto realizando iteraciones, que no son más que una división del proyecto en fases cíclicas en las que el proyecto va avanzando progresivamente. A cada uno de estos ciclos se le denomina iteración y al inicio de cada uno de ellas debe planificarse el trabajo a realizar en la misma.

Este ciclo de vida permite ir detallando el plan conforme avanza el proyecto y se va conociendo más sobre el mismo (disminuye la incertidumbre).

Ciclo de vida incremental. Es una particularización del anterior, mediante la cual cada ciclo que se realiza va obteniendo una porción de producto, servicio o resultado completa. A cada porción generada en una iteración se le denomina incremento.

Es decir, vamos produciendo porciones del resultado del proyecto que están acabadas al 100% e iteramos hasta tener todas las porciones, esto es, todo el resultado esperado.

El entregable se produce a través de una serie de iteraciones que sucesivamente añaden funcionalidad dentro de un marco de tiempo predeterminado. El entregable contiene la capacidad necesaria y suficiente para considerarse completo sólo después de la iteración final.

En proyectos en los que no está claro el alcance final o el plazo o coste disponible, un enfoque incremental dará sin duda mayor valor final al cliente.

**Recursivo**: por ejemplo, el espiral. Las actividades de este modelo se conforman en una espiral, en la que cada bucle o iteración representa un conjunto de actividades. Las actividades no están fijadas a ninguna prioridad, sino que las siguientes se eligen en función del análisis de riesgo, comenzando por el bucle interior. Puede verse como un modelo que conjuga la naturaleza iterativa con los aspectos controlados y sistemáticos del Modelo Cascada, con el agregado de gestión de riesgo.

Ventajas

El análisis del riesgo se hace de forma explícita y clara.

Une los mejores elementos de los restantes modelos.

Reduce riesgos del proyecto.

Incorpora objetivos de calidad.

Desventajas

Genera mucho tiempo en el desarrollo del sistema

Es un modelo costoso en recursos

Requiere experiencia en la identificación de riesgos

Aunque trabaja con iteraciones no produce versiones intermedias del producto, no se genera una versión del producto que pueda ponerse en producción hasta el final, con lo cual no se tiene una posibilidad de una retroalimentación real con el cliente al hacerlo con prototipos o versiones no productivas.

Cuando trabajamos con procesos definidos podemos elegir cualquier ciclo de vida de cualquier tipo, más allá de ciertas recomendaciones. Los empíricos no, solo recomiendan el iterativo/incremental, para aprender lo más rápido posible.

**Las 4 P**

La disciplina de ingeniería de sw se mide en 3/4 dimensiones: proceso, proyecto, producto y personas.

El proceso es el nivel más abstracto y teórico, es donde se escribe que se debería hacer para hacer sw, se define en término de roles, ya que para cada proyecto las personas asumen uno o más roles, por eso decimos que el proceso se instancia en cada proyecto para tener un lineamiento de que rol y responsabilidad tiene cada persona y que tareas tendrá asignadas para obtener el producto de sw. Es decir, yo puedo decir que en este proyecto el líder del proyecto será tal persona, pero en otro proyecto será otra. Cuando hablamos de ciclo de vida se aplica al proyecto. El proceso debo adaptarlo en caso de que sea definido o terminar de armarlo en caso que sea empírico con la gente que va a estar involucrada.

Entregar un producto de sw de calidad es la motivación. El proyecto es el medio por el cual obtengo un producto de sw. Es la unidad organizativa o de gestión.

El proyecto para existir necesita de personas, recursos y un método o forma para hacer el trabajo para conseguir el objetivo, es decir el producto de sw. El proceso es el que da una definición teórica de que debería hacerse para hacer sw. El proyecto toma de esa definición lo que parece que le hace falta con un determinado criterio, y con eso arma el conjunto de tareas a realizar, también llamado alcance del proyecto.

Proceso: definición de un conjunto de actividades que toman como entrada requerimientos y obtienen como salida un producto o servicio.

Proyecto: es un medio por el cual administro los recursos y las personas para obtener un producto o servicio. Incorpora personas que van a asumir roles.

Producto: es el sw.



**Software**

No es solamente un set de programas, sino también tiene que ver con la documentación que lo acompaña. Es conocimiento empaquetado. Todo lo que sale como resultado de la ejecución de las tareas de un proyecto es sw.

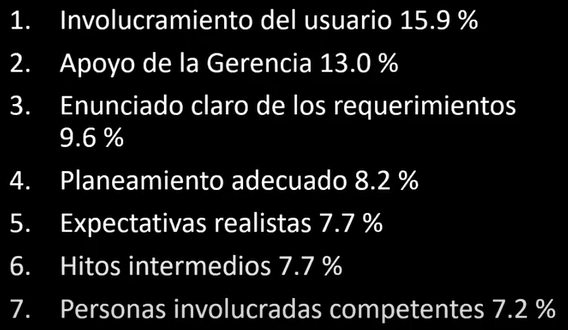
Tres tipos: system sw, utilitarios y sw de aplicación.

No hay forma de comparar sw con un proceso de manufactura. Esto se debe a que casi ningún producto de sw es igual a otro, ya que las necesidades y requerimientos siempre son diferentes. Además, el sw no es predecible y tampoco se gasta. Así mismo no todas las fallas son errores. Por último, no está gobernado por las leyes de la física.

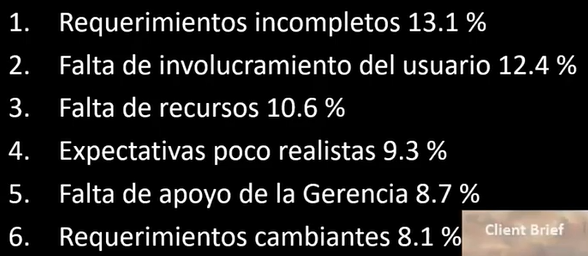
**Problemas del Desarrollo de SW**

* Que la versión final del producto no satisfaga las necesidades del cliente.
* Poder extender y escalar el sw en el tiempo
* Mala documentación
* Mala calidad del sw
* Más tiempo y costos que los presupuestados

**Ejemplos de sw fallidos**

**Cuando sale bien se debe a…**

**Cuando sale mal se debe a…**



**Ingeniería de SW**

Se divide en 3 ramas: disciplinas técnicas, de gestión y de soporte.



**Proyecto**

Características:

* Deben tener un objetivo único, el cual guía al proyecto.
* EL proyecto debe tener un inicio y un fin.
* El objetivo debe ser claro y alcanzable.
* Conseguimos el objetivo en base a un conjunto de tareas interrelacionadas.

**¿Los proyectos se planifican o no?**

En algún punto se debe planificar si o si, pero lo importante de la planificación es el acto de planificar y no el resultado, ya que es el acto del cual aprendemos.

**Administración de proyectos**

Administración de proyectos es la aplicación de conocimientos, habilidades, herramientas y técnicas a las actividades del proyecto para satisfacer los requerimientos del proyecto.

Administro los recursos y organizo a la gente afectada, y hago un seguimiento para ver si se está haciendo lo que dijimos. Finalmente, todas las actividades que intentan controlar y alcanzar el objetivo del proyecto, eso es administrar un proyecto. El líder o jefe del proyecto es el encargado de llevarla a cabo. En la gestión tradicional el líder quien tiene que hacer que y cuando, es el que se relaciona con todos. Lo ideal es que solo gestione y no tenga trabajos técnicos.

Tiene dos grandes disciplinas, la planificación y el seguimiento y monitoreo.

Administrar un proyecto incluye:

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

**La Restricción Triple**

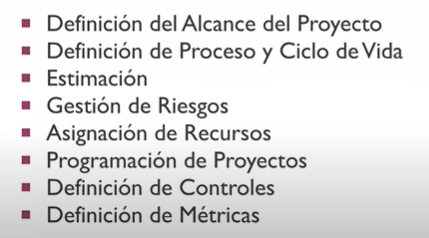
Para una buena administración se debe lograr un equilibrio entre 3 dimensiones, el alcance o producto (son los requerimientos), el tiempo y los recursos y gente que terminan implicando costos. El balance de estos tres factores afecta directamente la calidad del proyecto.

Lo ideal es poder controlar dos variables o por lo menos una. La que nunca depende de nosotros es el alcance, lo maneja el cliente.

**Plan de Proyecto**

Es el que nos va a guiar, define que vamos a hacer, cuando, como…

Documenta que hacemos (alcance), cuando (calendarización), como (afectación de recursos) y quien lo va a hacer (asignación de las personas).

Que implica la planificación:

Las métricas hacen falta ya que son definiciones cuantitativas, es decir, son objetivas. Nos ayudan a evitar percepciones subjetivas.

**Alcance**

Del producto: todas las características que pueden incluirse en un producto o servicio (va en la ERS – Especificación de requerimientos de sw). Se mide en requerimientos.

Del proyecto: es todo el trabajo y solo el trabajo que debe hacerse para entregar el producto servicio con todas las características y funciones especificadas (va en el plan del proyecto). Se mide en tareas.

Se relacionan en que, si el producto a construir es mas grande, las tareas del proyecto también serán más.

**Definición de Proceso y Ciclo de Vida**

Se define el proceso y ciclo de vida a utilizar, se puede combinar cualquier proceso con cualquier ciclo de vida en la gestión tradicional.

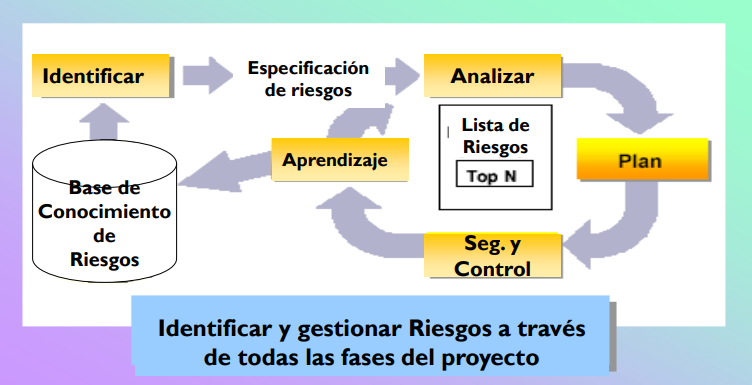
**Estimación**

Se estima en el siguiente orden: Tamaño - Esfuerzo – Calendario – Costo - Recursos críticos.

El esfuerzo son las horas de trabajo lineales. El costo se deja para después para hacer una estimación más ajustada a la realidad. Los recursos críticos son cosas más raras que necesitamos, no siempre hay. Por ejemplo, si desarrollamos un sistema de detección necesito el sensor. Si hay debo detectarlos para saber cuantos voy a necesitar y cuando deberé sumarlos al proyecto.

**Riesgos**

Eventos que podrían comprometer el éxito de proyecto.



**Métricas**

El dominio de las métricas del software se divide en:

* Métricas de proceso.
* Métricas de proyecto.
* Métricas de producto.

Las métricas del proyecto se consolidan para crear métricas de proceso que sean públicas para toda la organización del software.

Algunas métricas básicas para un proyecto de sw son el tamaño del producto, el esfuerzo, el tiempo y los defectos.

**Monitoreo y Control**

Se compara lo planificado y lo real. Encaso de que se requiera atrasar un proyecto se hace de un día a la vez.

**Tres factores para el éxito de un proyecto:**

* Monitoreo & Feedback
* Tener una misión/objetivo claro
* Comunicación

**Causas de Fracaso en los Proyectos:**

* Fallas al definir el problema
* Planificar basado en datos insuficientes
* La planificación la hizo el grupo de planificaciones
* No hay seguimiento del plan de proyecto
* Plan de proyecto pobre en detalles
* Planificación de recursos inadecuada
* Las estimaciones se basaron en “supuestos” sin consultar datos históricos
* Nadie estaba a cargo

**Unidad 2**

**Filosofía Ágil – Manifiesto Ágil**

Este enfoque surge de los propios programadores. Se busca lograr un equilibrio entre nada de proceso y demasiado proceso.

Esta basado en los procesos empíricos, en la experiencia que sale del propio equipo y de los ciclos de retroalimentación cortos.

El empirismo tiene 3 pilares: la adaptación, la inspección y la transparencia. Este ultimo nos permite crecer como equipo y transformar el conocimiento implícito que tenemos cada uno en conocimiento explicito. Conlleva pedir ayuda, informar en donde estoy, etc.

El manifiesto ágil tiene 4 valores y 12 principios.

Ojo, las dos cosas son importantes, pero se valora más lo de arriba.

1. Es más importante el vínculo y la comunicación que las herramientas y el proceso.
2. Nos interesa mas el sw en sí que el cómo, pero no significa que no se debe documentar o generar información del producto. Está la documentación del producto y la del proyecto. Se genera la información que hace falta y cuando hace falta. No toda esa documentación será persistente a lo largo de las iteraciones.
3. Puede haber un problema si hay un contrato formal y luego surge un cambio de requerimientos, pueden surgir fricciones. Es importante vincular al cliente con el proyecto y que el cliente esté dispuesto a participar.
4. No se define tanto de entrada, se empieza a trabajar y luego se va cerrando la idea.

Principios:

1. Satisfacer al cliente con sw funcionando, es decir, en condiciones de ponerse en producción
2. Los requerimientos cambian
3. Iteraciones lo mas cortas posibles (máximo dos meses)
4. Técnicos y no técnicos (responsables del negocio) trabajando juntos. Se suma al equipo un referente del negocio que decide o que debe priorizarse, es decir que debe tomar decisiones
5. Como debe ser la gente que trabaje (capacitados, motivados, con habilidades, comprometidos). Un desafío es que se queden durante todo el proyecto
6. Permite ver la actitud, gestos, cara… del otro
7. Es la mejor métrica, la mejor forma de mostrar avance
8. Necesita que se mantengan juntos los equipos. Se debe lograr un buen ritmo de trabajo para lograr las entregas de cada iteración
9. La calidad no se negocia, el sw que quiere el cliente no se ajusta sacrificando calidad. Tiene calidad cuando cumple con los requerimientos
10. Maximización del trabajo no hecho, para poder llegar con el tiempo. Evitar el “y ya que estamos…”
11. Las mejores decisiones arquitectónicas y de diseño emergen en el propio equipo que va a desarrollarlo. Que no decida alguien de fuera del equipo que debe hacer cada quien, opuesto a los jefes de la gestión tradicional. Soy yo quien voy, ve que hay que hacer, lo elijo y lo hago. Esto es MUY DIFICIL de llevar a cabo, ya que requiere compromiso, conocimiento técnico y motivación. Se espera proactividad del equipo
12. En intervalos regulares de tiempo, se determinan acciones concretas para mejorar, lo cual es esencial para lograr el empirismo

El enfoque tradicional fija el alcance en cada iteración, el ágil fija el tiempo, y sino llego con todo entrego hasta donde llegué.

**¿Qué es Ágil?**

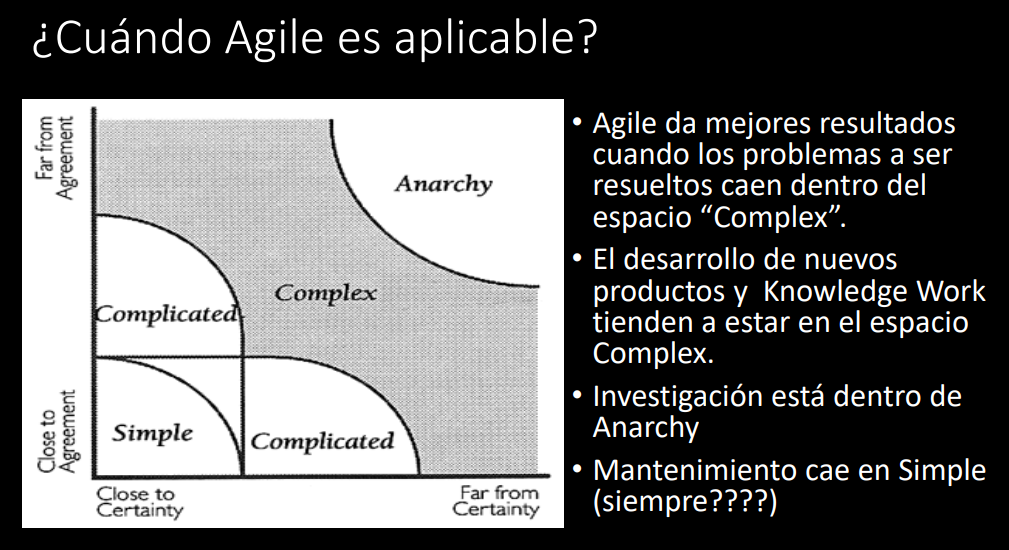
NO es una metodología o proceso, Ágil es una ideología con un conjunto definido de principios que guían el desarrollo del producto

Valores de los equipos ágiles …

* Planificación continua, multi-nivel
* Facultados, auto-organizados, equipos completos
* Entregas frecuentes, iterativas y priorizadas
* Prácticas de ingeniería disciplinadas
* Integración continua
* Testing Concurrente

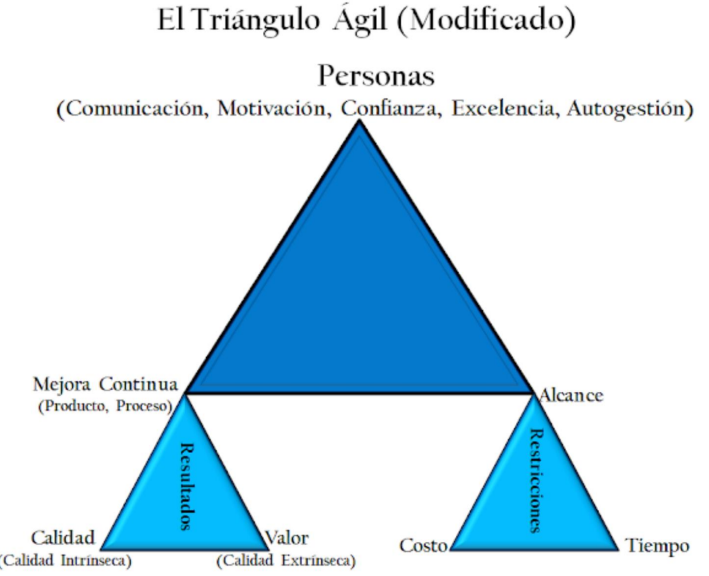
**¿Pero qué significa Ágil?**

* 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.

Algunos frameworks ágiles son XP, SCRUM, ATDD, FDD, Crystal… OJO, no son procesos, son buenas prácticas.

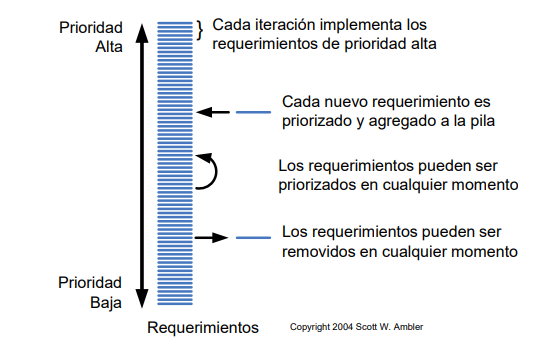
**El Triángulo Ágil**

El del enfoque tradicional hace mucho foco en el proyecto y en cumplir con los recursos del proyecto. El ágil le da mucha mas importancia al valor del producto. Hace mucho mas foco en la gente.



**Requerimientos Ágiles**

Se apunta a construir valor de negocio. Vamos a ir encontrando los requerimientos de a poco, empezando con lo mínimo necesario, solo lo suficiente. Esto es clave ya que por lo general la mayoría de las características que realmente se usan de un producto de sw son pocas, por eso es importante detectar y hacer foco en el valor. Se construye junto con el cliente o un referente del mismo. Se utilizan historias y modelos para mostrar que vamos a construir.

El dueño del producto es el que tiene clara sus necesidades, y su responsabilidad esencial es priorizar requerimientos.

En los enfoques tradicionales el usuario no suele estar dispuesto a comprometerse, por lo tanto, esto no pasa. El enfoque ágil busca cambiar esto, por eso es vital que el dueño del producto (referente) tenga ganas de estar disponible para el equipo y responder las dudas, que tenga conocimiento sobre el producto y que tenga posibilidades de tomar decisiones.

**Just in Time**

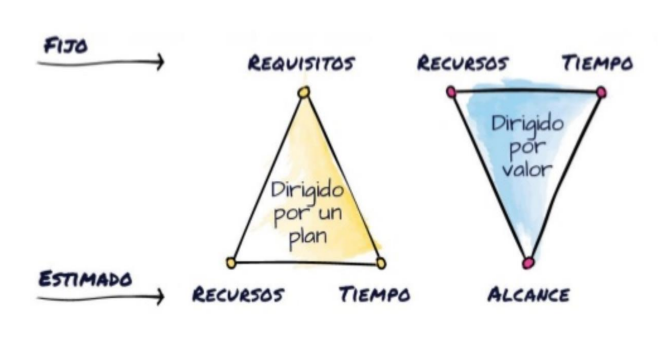
Este concepto busca evitar desperdicios, dice que el producto no va a estar especificado al %100 de entrada, sino que iremos viendo los requerimientos conforme nos hagan falta. OJO, no deben llegar antes pero tampoco tarde.

El cara a cara permite que la comunicación sea mas efectiva y que fluya.

**Valor de Negocio**

Debemos entregar valor de negocio, no características de sw. Estas ultimas son solo el medio por el cual entregamos valor.

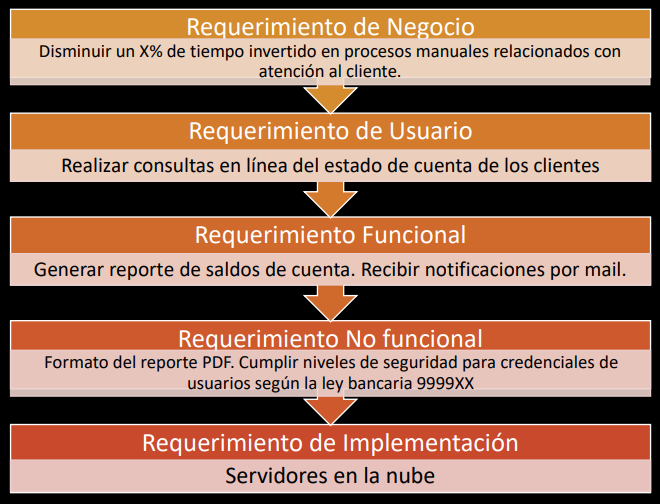
Podemos definirlo como la obtención de beneficio tangible o intangible, o asociarlo a la utilidad, beneficio o satisfacción que le ofrezco al usuario final por cada funcionalidad completa que entrego.

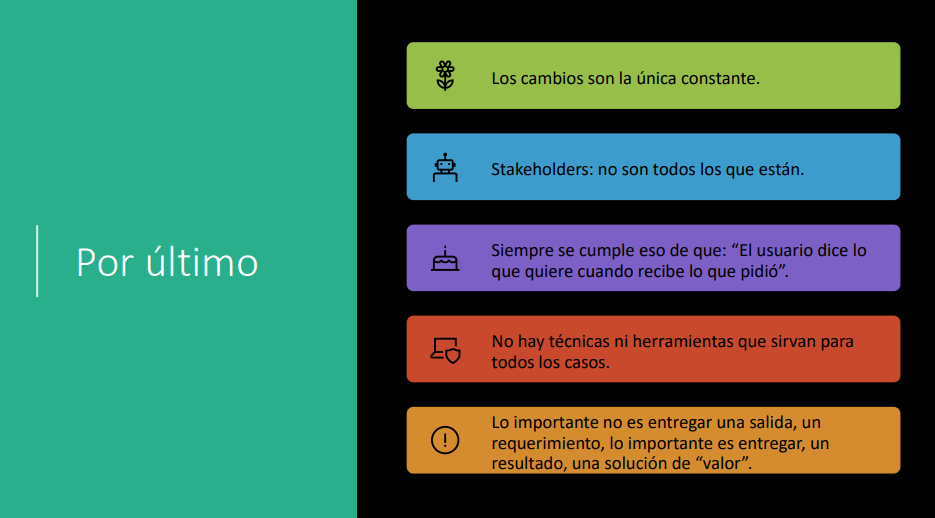
**Tradicional VS Ágil**

En vez de dejar fijo el alcance y derivar recursos y tiempo como la tradicional, fijamos el tiempo con iteraciones de duración fija y recursos fijos asignados al trabajo, a partir de eso determinamos el alcance de la iteración, es decir cuanto sw funcionando puedo entregar.

**Tipos de Requerimientos**

A nivel de gestión ágil de requerimientos, se trabaja a nivel de requerimientos de negocio y requerimientos de usuario (user stories). Al usuario le interesa el requerimiento de negocio, el de usuario, es el medio por el cual el cree que va a lograrlo, en caso que no sea así debemos decirlo.



En síntesis… técnicos y no técnicos debemos trabajar juntos entendiendo las necesidades y el negocio, y junto con los usuarios descubrir cual es la mejor forma de satisfacer esas necesidades, entregando frecuentemente valor para poder obtener feedback.

Stakeholders: todos los involucrados con el producto, el dueño del producto los representa.

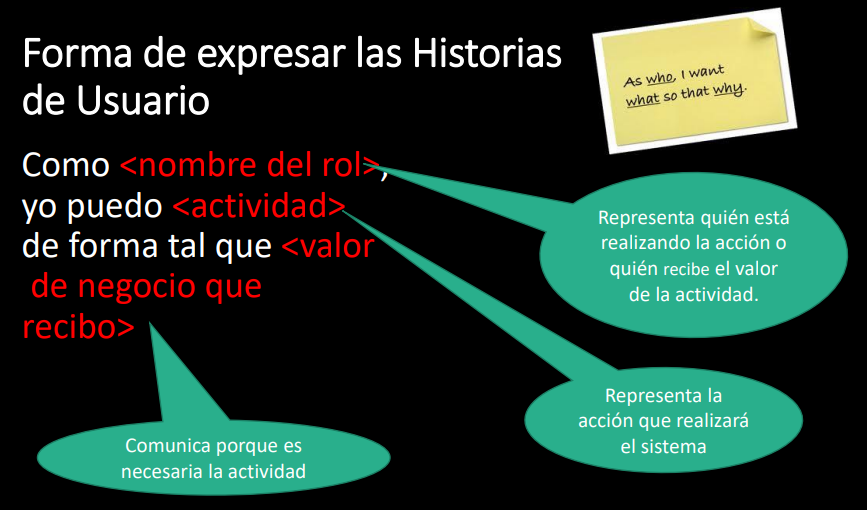
**User Stories**

Es una técnica de identificación de requerimientos. Es una descripción corta de una necesidad que tiene el usuario respecto del producto de sw. Es algo que el usuario quiere contarnos.

“La parte más difícil de construir un sistema de software es decidir precisamente qué construir. Ninguna otra parte del trabajo conceptual es tan difícil como establecer los requerimientos técnicos detallados... Ninguna otra parte del trabajo afecta tanto el sistema resultante si se hace incorrectamente. Ninguna otra parte es tan difícil de rectificar más adelante”

Tiene 3 partes: la tarjeta, la conversación y la confirmación. La mas importante es la conversación, la cual no queda guardada en ninguna parte, la gente se pone a trabajar en base a lo charlado. La parte visible de la user es la tarjeta, donde está la parte que queda de la conversación, y al dorso va la confirmación, que son las pruebas que se identifican como necesarias que debe cumplir el sw para que el PO me acepte esa característica de sw.

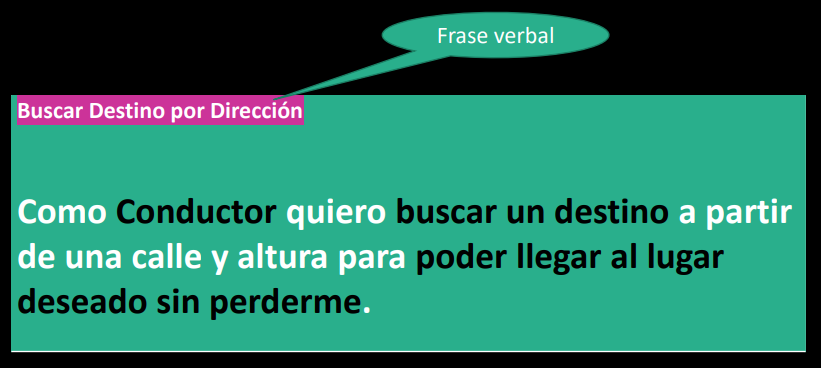
**Tarjeta**

Tiene una nomenclatura.

Mas que por que, es un para que.

El valor de negocio es clave para que el PO pueda priorizar.

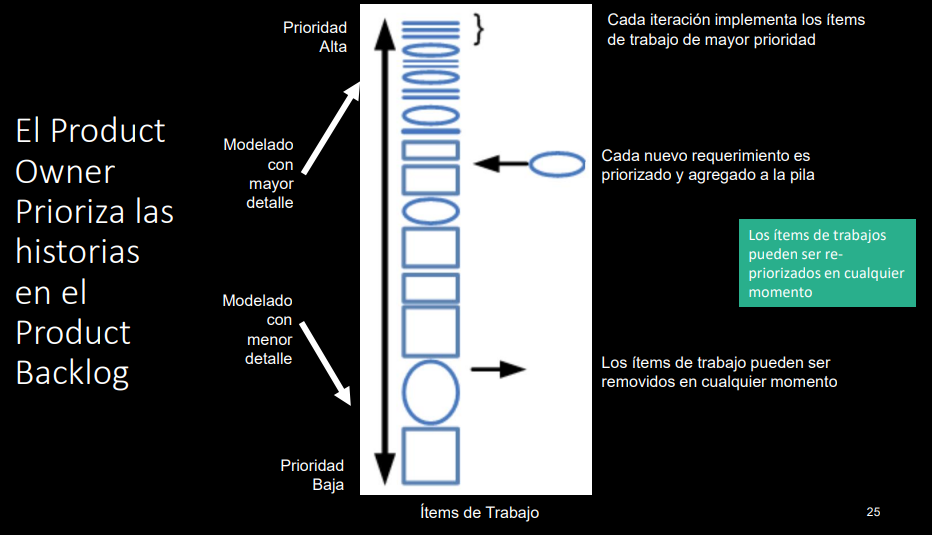
Ejemplo:



La frase verbal NO es la user en sí. Solo una forma mas corta de referenciarla, y no es obligatoria.

Las User Stories son Multipropósito. Las historias son:

* Una necesidad del usuario
* Una descripción del producto
* Un ítem de planificación
* Token para una conversación (recordatorio)
* Mecanismo para diferir una conversación

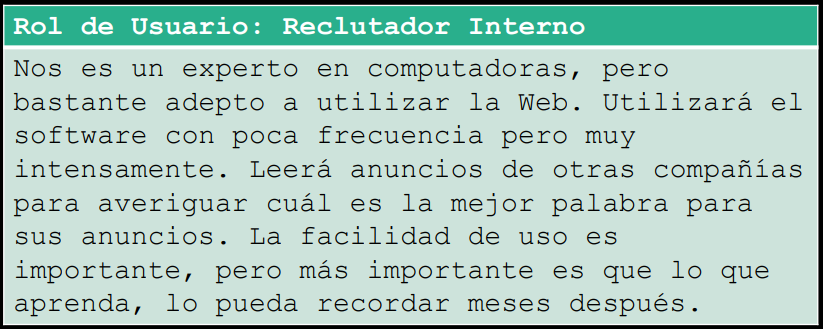
Esto NO es una especificación de requerimiento.

Las user stories son porciones verticales, en relación a las capas de una arquitectura. Si las cortara en horizontal, no estaría entregando ningún valor al cliente.



**Descripción de Roles**

Nos permite responder a quien le estamos haciendo el sistema, a quien le va a ser útil o no. Por ejemplo, una técnica es:



Describimos el perfil de usuario para entenderlo y conocerlo de forma que luego la persona que diseñe la experiencia de usuario lo haga pensando en ese perfil.

La idea es refinar esto con técnicas más específicas, por ejemplo una llamada “personas”, describiendo solo una persona en particular.

**Usuarios Representantes (Proxies)**

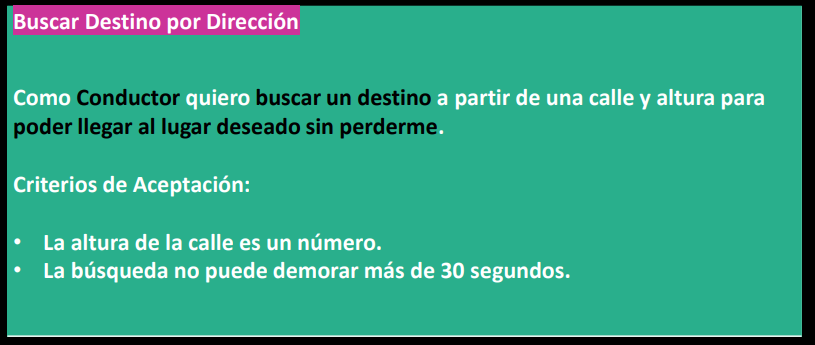
Cuando la situación ideal del PO no es posible, buscamos alternativas, los cuales siguen siendo perfiles de negocio. Tipos de usuarios representantes:

* Gerentes de Usuarios
* Gerentes de Desarrollo
* Alguien del grupo de marketing
* Vendedores
* Expertos del Dominio
* Clientes
* Capacitadores y personal de soporte.

**Criterios de Aceptación**

Es información concreta que nos sirve para saber si lo que implementamos es correcto o no. Se acuerdan con el PO. Nos definen limites y son la base con la cual se escriben las pruebas de aceptación.

* Definen límites para una user story (US)
* Ayudan a que los PO respondan lo que necesitan para que la US provea valor (requerimientos funcionales mínimos)
* Ayudan a que el equipo tenga una visión compartida de la US
* Ayudan a desarrolladores y testers a derivar las pruebas.
* Ayudan a los desarrolladores a saber cuándo parar de agregar funcionalidad en una US

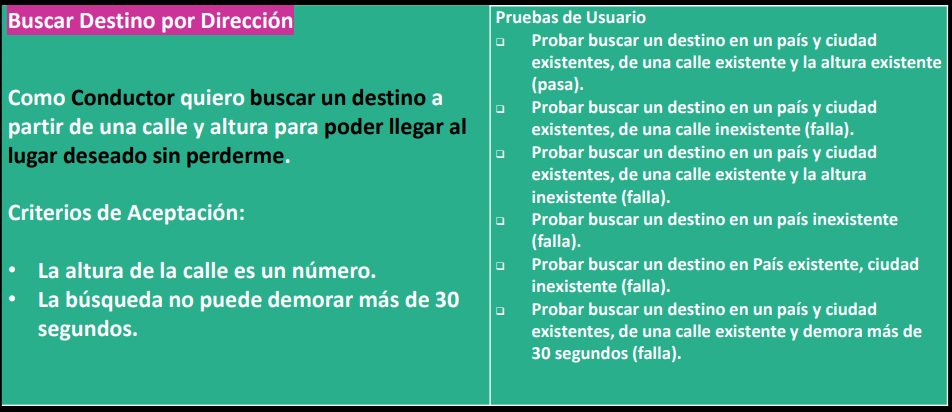


Se escriben a nivel de usuarios, pero deben ser objetivos y verificables.

Los detalles que antes estaban en la ERS, este enfoque propone moverlos a las pruebas de aceptación, y en la medida de lo posible automatizadas. De esta forma se genera esfuerzo de documentación una sola vez y permiten que las pruebas se ejecuten, ya que sin detalle no es posible.

**Pruebas de Aceptación**

Están relacionadas con los criterios de aceptación. Lo que va en la user no es el caso de prueba, es solo el título.



Debemos contemplar tanto situaciones de éxito como de fracaso.

**Definition of Ready**

Es una medida de calidad que construye el equipo para determinar que la user está en condiciones de entrar en una iteración de desarrollo. Debe tener como mínimo lo que plantea el INVEST model.

**INVEST Model**

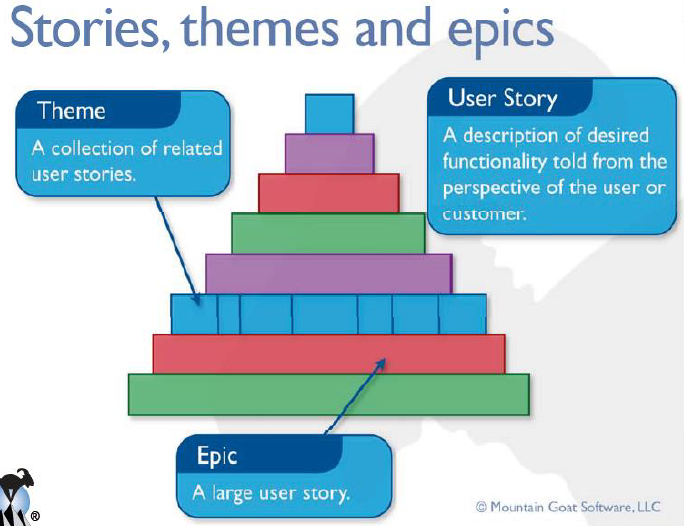
* Independent - calendarizables e implementables en cualquier orden, y le doy la libertad al PO para que las priorice en cualquier orden
* Negotiable – el “qué” no el “cómo”, que necesita el usuario, no como lo vamos a implementar
* Valuable – debe tener valor para el cliente
* Estimatable – para ayudar al cliente a armar un ranking basado en costos, debemos asignarle un peso a la user, un numero que me permita compararla con otras
* Small – deben ser “consumidas” en una iteración, es la más difícil de todas de imaginar, ya que depende del equipo y de la duración de la iteración
* Testable – poder demostrar que fueron implementadas, relacionado a las pruebas

**Definition of Done**

Determina cuando se termina una user, es decir, cuando está lista para ser entregada. Este criterio también lo decide el equipo.

**Algo más sobre las User Stories…**

* No son especificaciones detalladas de requerimientos (como los casos de uso)
* Son expresiones de intención, “es necesario que haga algo como esto…”
* No están detallados al principio del proyecto, elaborados evitando especificaciones anticipadas, demoras en el desarrollo, inventario de requerimientos y una definición limitada de la solución.
* Necesita poco o nulo mantenimiento y puede descartarse después de la implementación.
* Junto con el código, sirven de entrada a la documentación que se desarrolla incrementalmente después.

**Niveles de Abstracción**

Si una user es muy grande se llama épica.

Un conjunto de users relacionadas se pueden juntar en un tema.

**Spikes**

Tipo especial de historia, utilizado para quitar riesgo e incertidumbre de una User Story u otra faceta del proyecto.

Se clasifican en : técnicas y funcionales.

Pueden utilizarse para:

* Inversión básica para familiarizar al equipo con una nueva tecnología o dominio.
* Analizar un comportamiento de una historia compleja y poder así dividirla en piezas manejables.
* Ganar confianza frente a riesgos tecnológicos, investigando o prototipando para ganar confianza.
* Frente a riesgos funcionales, donde no está claro como el sistema debe resolverla interacción con el usuario para alcanzar el beneficio esperado.

Técnicas

* Utilizadas para investigar enfoques técnicos en el dominio de la solución.
* Evaluar performance potencial
* Decisión hacer o comprar
* Evaluar la implementación de cierta tecnología.
* Cualquier situación en la que el equipo necesite una comprensión más fiable antes de comprometerse a una nueva funcionalidad en un tiempo fijo.

Funcionales

* Utilizadas cuando hay cierta incertidumbre respecto de cómo el usuario interactuará con el sistema.
* Usualmente son mejor evaluadas con prototipos para obtener realimentación de los usuarios o involucrados.

Algunas User Stories requieren de ambos tipos de spikes. Por ejemplo:

Como un cliente, quiero ver mi uso diario de energía en un histograma, para poder comprender rápidamente mi consumo de energía pasado, presente y proyectado.

En este caso un equipo puede crear dos spikes

* Spike Técnico:

Investigar cuanto tiempo requiere actualizar un display de un cliente al uso actual, determinando requerimientos de comunicación, ancho de banda y si los datos se actualizan en formato push o pull

* Spike Funcional:

Crear un prototipo de histograma en el portal web y obtener la retroalimentación de algunos usuarios respecto del tamaño, el estilo de la presentación y los atributos gráficos.





**Tips para que las US sean útiles:**

* Un paso a la vez (evitar la palabra “Y”)
* No olvides la parte invisible: la conversación
* Usar palabras claras en los criterios de aceptación
* Las user stories se escriben desde la perspectiva del usuario
* No forzar todo para escribirlo como user stories

**Gestión de Productos**

Las motivaciones son el disparador para crear un producto. Es importante tener en cuenta que son pocas las características de un producto que realmente se usan a menudo. En parte es responsabilidad del usuario y en parte nuestra.

Por eso es importante entender que podemos hacer un producto de manera iterativa e incremental yendo a la esencia del mismo en un comienzo y luego ir mejorándolo. No estamos obligados a entregar todo de entrada.



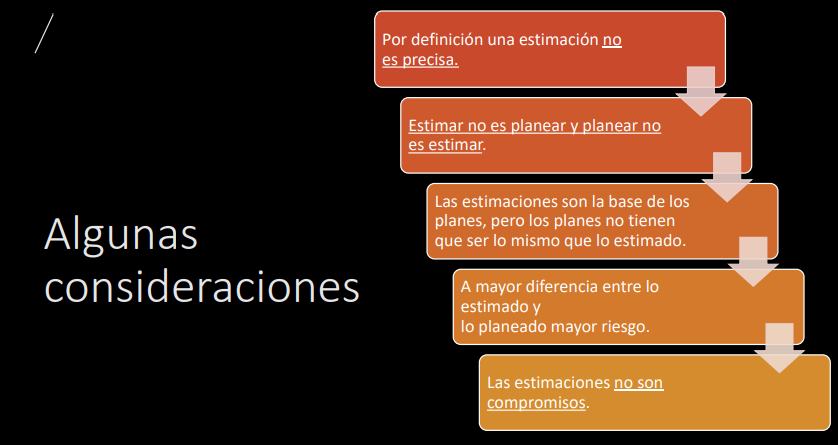
Funcional: que haga lo que tiene que hacer.

Confiable: los usuarios tienen que sentirse cómodos en relación a que no corren peligro.

Usable: logro mis objetivos productivamente usando el producto. El usuario siente que el producto mejora su calidad de vida. Tiene que ver con la experiencia de usuario.

Conveniente: soy mejor al usar el producto. Genera un diferencial con respecto a otros productos.

Debo poner el foco en la cima de la pirámide y no en la base.

**Estimaciones de SW**

Al principio no tenemos tanta información para estimar, con lo cual es mas difícil. A medida que nos vamos acercando al final del proyecto vamos teniendo más información y nos equivocamos menos. Por lo tanto, ninguna cosa que asumamos al comienzo se debe ver como inamovible. Debemos asumir que, si o si va a haber errores en las estimaciones, lo que se debe tratar es que sean los menos posibles. Hay errores que vienen del propio proceso de estimación, otros mas graves relacionados a actividades omitidas, ya que muchas veces solo se estima el proceso de programación, la cual solo es aproximadamente solo un %35.

La estimación debe ser cuantificable, debo contar algo. En un principio se contaban líneas de código, pero no hay forma de saber cuántas líneas tendrá el producto de entrada.

**Métodos utilizados**

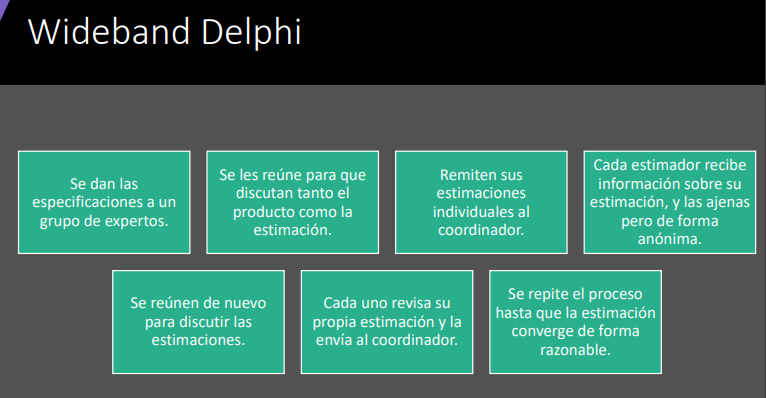
* Basados en la experiencia.
* Basados exclusivamente en los recursos.
* Método basado exclusivamente en el mercado.
* Basados en los componentes del producto o en el proceso de desarrollo.
* Métodos algorítmicos

Métodos basados en la experiencia:

•Datos Históricos, lo usan los procesos definidos, asumiendo que la experiencia de otros proyectos o equipos me pueden servir. A diferencia del enfoque ágil que opina que la experiencia es algo que solo puede capitalizar la misma gente.

•Juicio experto

* Puro, tenemos un estimador en la empresa. El problema es que a saber de que es experto, hay mucha dependencia de la organización con él, y estima desde su perspectiva. Es el enfoque de estimación mas usado. Se le puede dar algo de estructura usando el método de “optimista, pesimista y habitual” y su fórmula = (o + 4h + p)/6.
* Delphi, es un juicio experto grupal, el cual se toma como base para el poker planning. El grupo trata de adivinar lo que costará el desarrollo tanto en esfuerzo como en duración. Cada uno comienza haciendo una estimación individual y luego se ponen de acuerdo y llegan a un numero mas relacionado a la realidad.

•Analogía, se hace una abstracción de los datos históricos con solo los que son parecidos a este proyecto.

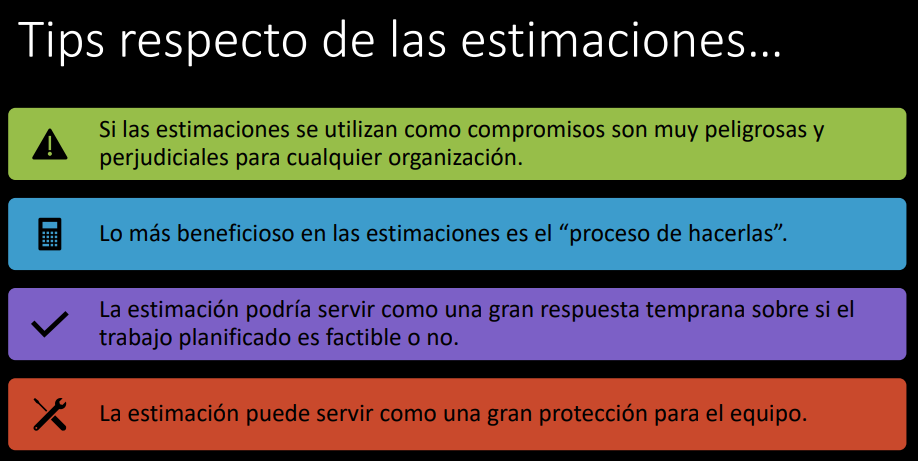
**Estimaciones Ágiles**

Buscan resolver los problemas de la estimación tradicional, así como estimar demasiado pronto, resistencia a las reestimaciones, estimaciones que buscan precisión, estimaciones que buscan ser absolutas, estimaciones que las hacen personas que no harán el trabajo…

Algunas posturas dicen que estimar es desperdicio, ya que puede no salir bien.

Estas estimaciones las debe hacer el que hace el trabajo. Las estimaciones son un proceso, y uno aprende en el proceso. Se debe entender que las estimaciones no son certezas y que tienen un valor de probabilidad asociado.

Todo lo que definimos con estimaciones tienen que ver con producto, a diferencia de las estimaciones tradicionales. Se estima el tamaño del producto.



Las features/stories son estimadas usando una medida de tamaño relativo conocida como story points (SP). Es una unidad de medida específica (del equipo) de, complejidad, riesgo y esfuerzo. Representa el peso de la user. Usamos ese peso por comparación, de manera relativa. Las medidas relativas no son absolutas.

Las personas no saben estimar en términos absolutos. Somos buenos comparando cosas. Comparar es generalmente más rápido. Se obtiene una mejor dinámica grupal y pensamiento de equipo más que individual Se emplea mejor el tiempo de análisis de las storys.

Story Points no es una medida basada en tiempo. Es una medida homogeneizadora que se basa en 3 dimensiones: la complejidad, el esfuerzo (por eso es importante que estime el que hace el trabajo) y la duda o incertidumbre (cuanta información nos hace falta).

El equipo debe justificar los puntos que le dan a una user.

El tamaño es una medida de la cantidad de trabajo necesaria para producir una feature/story. El tamaño indica: cuán compleja es una feature/story, cuánto trabajo es requerido para hacer o completar una feature/story y cuán grande es una feature/story.

La mayor problemática de esto es que tendemos a confundir tamaño con esfuerzo y esfuerzo con calendario. El esfuerzo siempre debe ir atado a construir que, es decir el tamaño. El tamaño esta representado por la complejidad y habla del producto, es decir, independientemente de quien lo haga que tanta dificultad tiene. Luego el esfuerzo refiere a cuantas horas necesito yo para hacerlo, no es lo mismo que lo haga uno a que lo haga otro.

El esfuerzo se mide en horas ideales, despejando todo lo que no sea el trabajo.

No está bueno estimar solo en calendario, las estimaciones basadas en tiempo son más propensas a errores debido a varias razones. ▪ Habilidades ▪ Conocimiento ▪ Asunciones ▪ Experiencia ▪ Familiaridad con los dominios de aplicación/negocio.

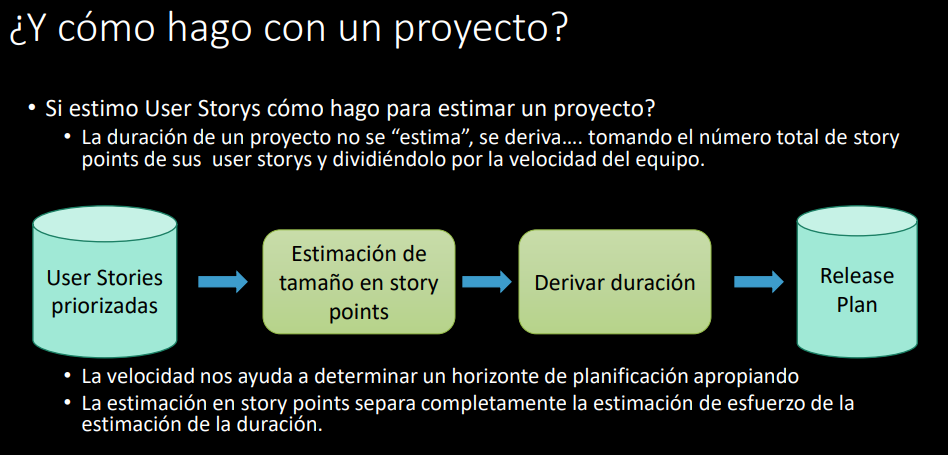
Además, pueden surgir otros trabajos de por medio.

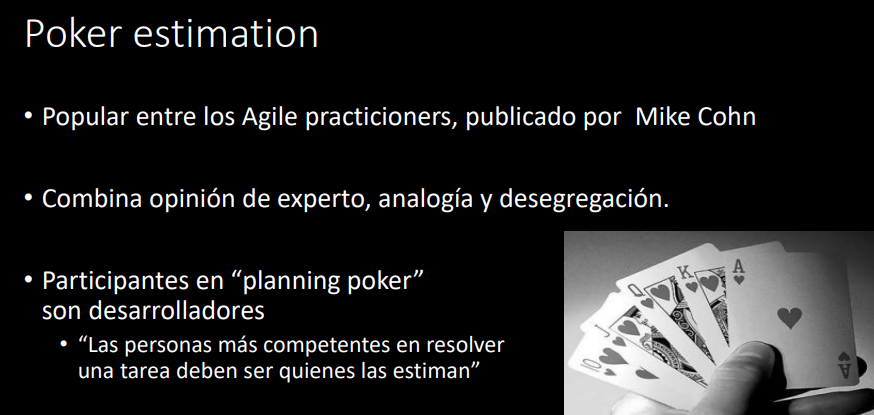
Ojo, no significa que no se deba estimar tiempo, pero siempre en relación/derivación del esfuerzo.

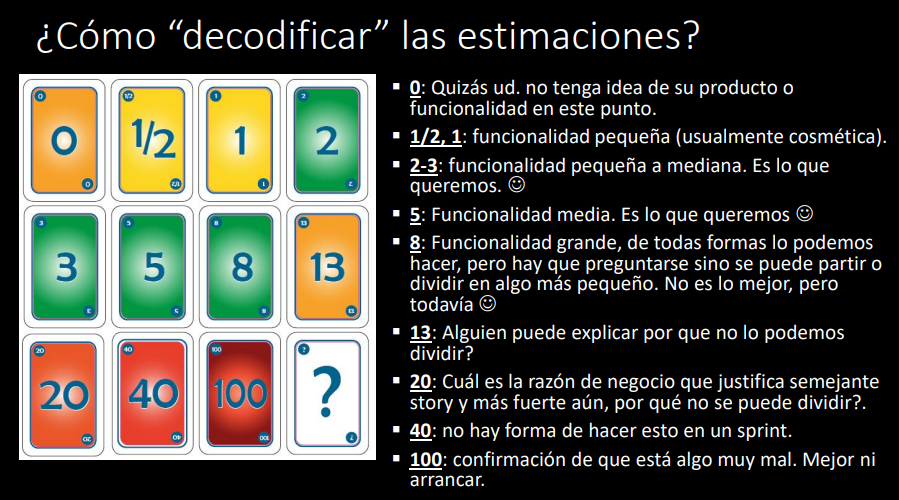
Se debe elegir una escala para medir el tamaño, la cual una vez elegida no se cambia (pueden ser números del 1 al 10, talles de remera, Fibonacci…). La complejidad de una user tiende a incrementarse exponencialmente.

Teniendo en cuenta el story point como unidad de estimación, puedo sumarlos para obtener una métrica ágil llamada velocidad. Es cuanto producto le entrego al usuario al final de una iteración, no se puede estimar, se calcula al final del sprint.

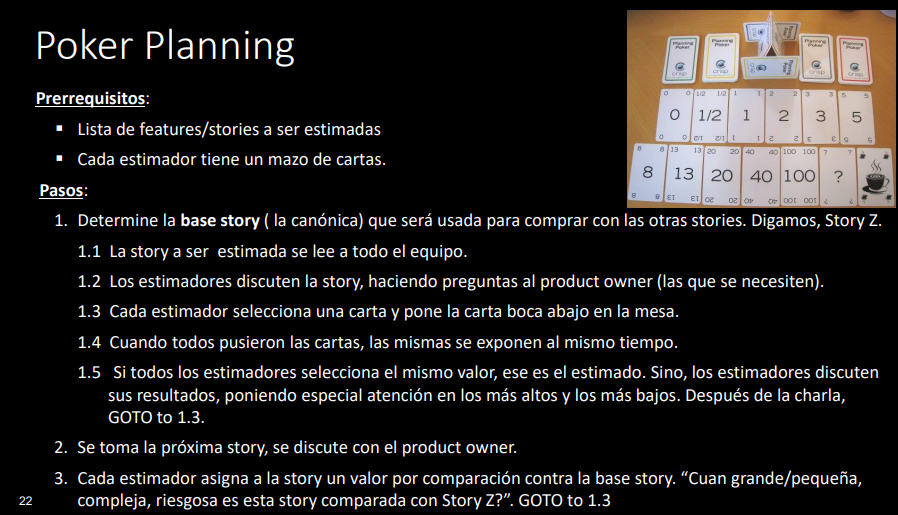
Velocidad/ Velocity es una medida (métrica) del progreso de un equipo. Se calcula sumando el número de story points (asignados a cada user story) que el equipo completa durante la iteración. Se cuentan los story poins de las Users Storys que están completas, no parcialmente completas. La Velocidad corrige los errores de estimación.







Cuando trabajo con iteraciones fijas, el tiempo no se mueve. Debo trabajar con piezas de productos lo mas pequeñas posibles. Mientras más pequeñas más fácil es determinar si las terminaste o no. La única manera de meter una user de 21 o mas en un sprint es cuando no se puede partir de ninguna manera y el equipo blanquea que no se va a poder terminar en ese sprint. La mayoría de las user se pueden partir.



Al estimar debemos tener en cuenta que la probabilidad nos puede salir a favor o en contra, por lo tanto no debemos aferrarnos a esas estimaciones.