CLASE GRABADA N°2 – FILOSOFÍA ÁGIL

Filosofía ágil – Manifiesto ágil

- **Introducción**.

  Proviene desde los equipos de desarrollo, es decir, es un movimiento que se gestó desde los desarrolladores/programadores. Ellos impulsaron a que surja este nuevo pensamiento y enfoque/visión para hacer software.

  En septiembre de 2001 se juntaron un grupo de referentes de software en un hotel en Utah a proponer sus frameworks de trabajo y sus prácticas. Debatieron y acordaron un conjunto de cosas que llamaron: "Manifiesto Ágil". Desde ahí salió un "compromiso", en el cual todas esas personas que firmaron el manifiesto ágil de alguna manera están acordando una manera de trabajar. Las filosofías lo que definen son formas de manejarse en las situaciones que debe enfrentar en la vida real.

  Significo una evolución cultural que tuvo que ver con las experiencias que cada uno de los presentes en esa reunión vivió. Debido a que en los inicios todo se realizaba de manera bastante artesanal.

  Seguidamente, aparecieron los militares imponiendo una forma muy rígida de desarrollar software, con estándares muy estrictos y muchas restricciones. Procesos muy determinados de antemano, con mucha exigencia, mucha carga de informes y documentación formal.

  Se dan comportamientos pendulares de las personas que van de un extremo a otro.

  La idea del movimiento ágil fue de alguna manera lograr un equilibrio. Una situación de: "no hagamos las cosas como si no fuéramos profesionales, no hagamos las cosas como si fuera un trabajo artesanal, ni tampoco vayamos al otro extremo de estar 100% encarcelados en restricciones"

  Se resumen en: "Compromiso útil entre nada de proceso y demasiado proceso" (Fowler, 2001)

  Filosofía ágil se materializa en el movimiento ágil.

  El movimiento ágil esta basado en que la forma de trabajo va a ser sustentada en los conceptos de los PROCESOS EMPÍRICOS. Basados en la experiencia del propio equipo, con ciclos de retroalimentación cortos. Comenzamos con algo, asumimos la acción de construir algo y lo controlamos y el mostrarlo hace que consigamos realimentación para analizar lo que se deba y corregir los errores. Asi, vamos avanzando.

- **3 pilares del empirismo**
  - Inspección: La inspección en este contexto no es una inspección realizada por un inspector o un auditor, sino una inspección realizada por todos los miembros del Equipo Scrum. La inspección puede realizarse para el producto, procesos, aspectos de personas, prácticas y mejoras continuas. Por ejemplo, el equipo muestra abierta y transparentemente el producto al final de cada Sprint al cliente para obtener valiosos comentarios. Si el cliente cambia los requisitos durante la inspección, el equipo no se queja, sino que se adapta al usar esto como una oportunidad para colaborar con el cliente para aclarar los requisitos y probar la nueva hipótesis.

  - Adaptación: la adaptación en este contexto se trata de la mejora continua, la capacidad de adaptarse en función de los resultados de la inspección. Todos en la organización deben hacer esta pregunta regularmente: ¿Estamos mejor que ayer? Para las organizaciones basadas en ganancias, el valor se representa en términos de ganancias. La adaptación eventualmente debería volver a una de las razones para adaptar Agile: por ejemplo, un tiempo de comercialización más rápido, un mayor retorno de la inversión a través de una entrega basada en el valor, un menor costo total de propiedad a través de una mejor calidad de software y una mayor satisfacción de los clientes y empleados.

- Transparencia: nos permite crecer como equipo y transformar el conocimiento implícito en conocimiento explicito. Conocimiento que se pueda transmitir, que sea del equipo, y luego se traslade a la organización. Lo que yo hago no es MIO, es producto del equipo que está construyendo. Si tengo algún problema hay que blanquearlo y pedir ayuda. Informar en que estado estamos. Esto significa presentar los hechos tal como están. Todas las personas involucradas —el cliente, el CEO, los contribuyentes individuales— son transparentes en sus tratos cotidianos con los demás. Todos confían el uno en el otro, y tienen el coraje de mantenerse al tanto de las buenas y malas noticias. Todos se esfuerzan y colaboran colectivamente por el objetivo organizacional común, y nadie tiene una agenda oculta.

Manifiesto ágil

4 valores: Ambas cosas son importantes, solo que las primeras se priorizan por sobre las segundas. Se valora tanto lo de la izquierda como lo de la derecha, pero valoramos MAS lo que esta a la izquierda.

- **Individuos e Interacciones SOBRE procesos y herramientas**
  - Individuos e Interacciones: vínculos, comunicación con el equipo y con el cliente.
  - Aunque los procesos son de ayuda para guiar el trabajo, deben adaptarse a la organización, los equipos y las personas, y no al revés. Asimismo, defiende que aunque las herramientas mejoran la eficiencia, no consiguen resultados por sí solas
- **Software funcionando SOBRE documentación excesiva**
  - Importa mas el software en si mismo MAS que el respaldo que debemos documentar en largos registros (modelos, manuales). Priorizar primero el desarrollo de las funcionalidades de este.
  - Valor del cual muchísima gente se aprovecha para no crear ninguna documentación.
  - La realidad es que existe la necesidad de mantener información sobre el producto de software y sobre el proyecto que estamos transcurriendo para obtener ese producto. El enfoque ágil plantea: generemos la información que haga falta cuando haga falta. No especifica que NO se genere información.
  - Ágil dice: “Los procesos de generación de información son lo más importante.”
  - Las decisiones que se van tomando del producto se necesita asentarlas y que trasciendan.
  - El conocimiento tiene que ser de la organización no de los individuos.
  - Generar poca información (dado que los tiempos iterativos son cortos) formal y la que haga falta. No toda la documentación que se genera es necesario que sea persistente eternamente.
- **Colaborar con el cliente SOBRE negociación contractual**
  - Conocer primero al cliente, generarle confianza, serle sincero en todos los aspectos, contenerlo antes de pasar a organizar los acuerdos financieros y contractuales
  - Hacerlo formar parte al cliente.
  - CAMBIOS que pide el cliente son IMPORTANTES
  - El problema de las negociaciones contractuales en muchos casos en las empresas empieza cuando el cliente tiene un acuerdo formal firmado con la organización con un presupuesto, un plazo, términos y características de SW a implementar y el cliente quiere agregar una nueva funcionalidad o recuerda alguna necesidad o restricción. Comienzan las fricciones/frustraciones/malentendidos.
  - El cliente debería integrarse al equipo y ayudar en la construcción del sw.
  - Es un punto clava saber si el cliente con el que estamos tratando esté dispuesto a formar parte y tomar acciones a medida que el software se construye. Determina si seremos agiles o no.

- **Responder al cambio SOBRE seguir un plan**
  - El cuarto de los valores de Agile habla de que no tiene sentido utilizar planteamientos rígidos en escenarios volátiles como el del desarrollo de software.
  - Es más valiosa la capacidad de respuesta y adaptación a los cambios que la de seguir y asegurar el cumplimiento de los planes preestablecidos.
  - Las metodologías ágiles promueven la anticipación y la adaptación, frente a la planificación y el control que proponen las fórmulas de gestión tradicionales.

12 principios del Manifiesto Ágil

1. Nuestra mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software de valor.

2. Aceptar que los requisitos cambien. Cambiar los requisitos incluso en etapas avanzadas.

3. Entregar software funcional con frecuencia, preferentemente en semanas o meses.

4. Los responsables de negocios, diseñadores y desarrolladores deben trabajar juntos día a día durante el proyecto. Colaboración constante entre los interesados y los desarrolladores.

5. Construir proyectos en torno a individuos motivados y proporcionarles el entorno y apoyo que necesitan.

6. El método mas eficiente de comunicar información es por medio de conversaciones cara a cara.

7. El software que funciona es la principal medida de éxito. Software funcional es la principal medida de progreso.

8. Los procesos agiles promueven el desarrollo sostenible. Mantener un ritmo sostenible de desarrollo.
   Hay que recordar que los plazos de entrega en ágil son fijos.

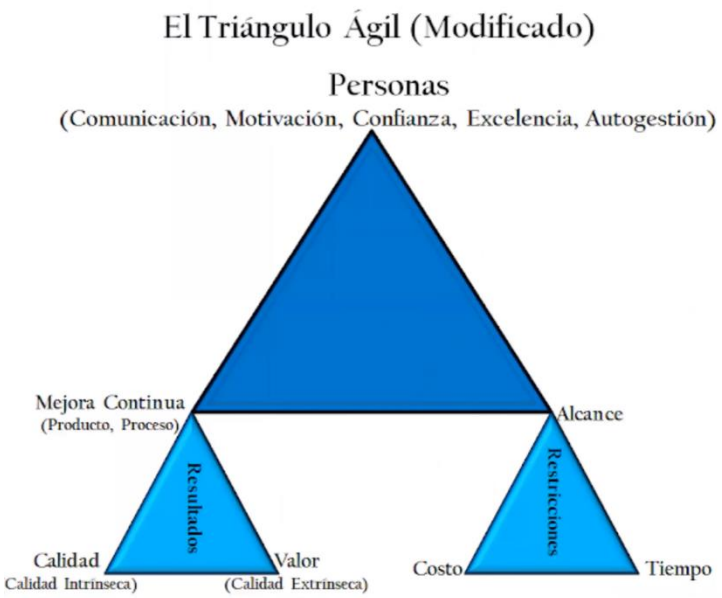
9. La atención continua a la excelencia técnica y al buen diseño mejora la agilidad.
   La calidad del producto no es negociable. Hagamos otros ajustes como: cantidad de sw que entregamos, menos funcionalidades, pero NO SE TOCA LA CALIDAD.

10. La simplicidad es esencial
    Maximizar la cantidad de trabajo no realizado.
    El enfoque consiste en maximizar la cantidad de trabajo que no se realiza, centrándose en las funcionalidades esenciales y evitando agregar características superfluas. La simplicidad permite una mayor agilidad al facilitar la comprensión del sistema y la capacidad de realizar cambios de manera más rápida y efectiva.

11. Las mejores arquitecturas, requisitos y diseños emergen de sistemas autoorganizados.
    Que las decisiones sean tomadas por personas que formen parte del equipo y que las decisiones no sean tomadas por alguien externo al mismo.

12. A intervalos regulares, el equipo reflexiona sobre cómo ser mas efectivo y luego ajusta y perfecciona su comportamiento en consecuencia.

- El triangulo ágil. El triangulo de Hierro.



CLASE GRABADA N°1 – CONCEPTOS BÁSICOS

Proceso de Software.

En definitiva, es la transformación de ideas, requerimientos y tiempo en productos y servicios de software, realizando un conjunto actividades estructuradas de trabajo en las que utilizamos recursos como materiales, energía, equipamiento, procedimientos e intervienen personas.

Las actividades varían dependiendo la organización y el tipo de sistema que debe desarrollarse. Un proceso de software debe ser explícitamente modelado si va a ser administrado.

Objetivo = Entregar un producto de software de calidad a los clientes.

Definición de proceso de Software (SW-CMM). Un conjunto de actividades, métodos, practicas y transformaciones que la gente usa para desarrollar o mantener software y sus productos asociados.

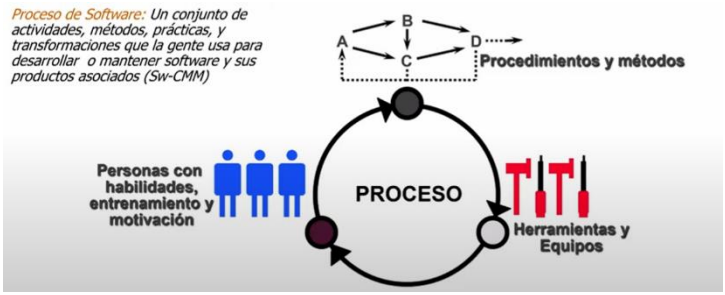


Foto = enfoque de mesa de 3 patas. Todo es importante para lograr el equilibrio que hace falta para que un proceso realmente puesto en una organización funcional obtenga un producto de software de calidad.

Contexto dado por Judith = La disciplina de sistemas, es una industria que es humano intensiva, significa que el software es creado por personas y que el aporte mas importante para que el producto llegue a las manos de los usuarios, es el aporte que hacen las personas.

Somos el elemento más importante en un proyecto para entregarle software a alguien.

En términos de costos, lo mas caro de hacer software es pagarle a la gente que trabaja haciendo software, el 80% del costo de un producto de software es el producto del esfuerzo.

Costo de un producto de software:

- 80% - costo del esfuerzo de las personas
- 10% - conectividad
- 10% - costos menores

Gente que interviene debe …

- Estar formada y capacitada, conocimiento técnico para hacer su trabajo
- Motivada. Ganas.

Necesitamos

- Herramientas de soporte y que nos ayuden a automatizar lo mas que podamos el proceso de hacer software. Existen actualmente y por suerte son cada vez más. Ayudan a que nuestra cabeza haga foco en lo que no es reemplazable por una herramienta automatizada.

Tipos de Procesos.

Procesos Definidos. Procesos que vienen del taylorismo y de las líneas de producción de Ford. Plantean la necesidad de tener definido ampliamente paso a paso qué vamos a hacer en cada momento. Asume que podemos repetir el mismo proceso una y otra vez indefinidamente, y obtener los mismos resultados.

La administración y control provienen de la predictibilidad del proceso definido. Definidas las tareas a realizar por cada persona, cuando la realizará, que entradas y salidas tiene, qué artefactos se generan y en qué orden.

Tienen la intención de ser REPETITIVOS. Se busca lograr la repetitividad para lograr la visibilidad y saber qué esperar a cada momento de tiempo y para poder predecir situaciones posibles. Las personas que implementan estos procesos quieren tener la ilusión de CONTROL. Todo el mundo sabe lo que tiene que hacer.

Intentan ser procesos COMPLETOS que describen la mayor cantidad de cosas posibles.

Ejemplo: PUD (Proceso Unificado de Desarrollo). Explica paso a paso la mayor cantidad de opciones posibles que deberíamos tener en cuenta para hacer un producto de software. Se adapta a cada situación en particular. Nivel de definición de las actividades es anticipado. + formales, con + definiciones sobre las cosas, + explícitos. RUD

Procesos Empíricos. Tienen un fuerte arraigo en la experiencia. La experiencia no siempre es fácil de conseguir debido a que la misma es basarse en algo anterior para tomar una decisión en la actualidad. La experiencia no se puede comprar, ni es intrínseca a las personas, la experiencia se tiene que generar.

Surgen en contraposición de los procesos definidos.

Basado en ciclos de entrega cortos para poder generar retroalimentación que sirva como experiencia para evolucionar y seguir avanzando.

Son procesos que no están COMPLETOS. Sino mas bien se basan en la creación de guías, lineamientos, heurísticas, prácticas que cubren determinados aspectos del proceso. Hacen foco en la gestión o en la ingeniería, pero ningún proceso empírico cubre la TOTALIDAD de las cosas que hay que hacer. Esta característica es la que permite que los equipos al iniciar el trabajo decidan (basado en la experiencia) qué es lo que quiere hacer, cómo y cuándo.

Patrón de conocimiento en procesos empíricos. Asumen una hipótesis de que algo funciona de X forma, actúan/prueban, obtienen retroalimentación, revisan y si algo no salió bien, adaptan. Hay retroalimentación. Y el ciclo vuelve al inicio. De esta forma ganan experiencia.

El empirismo sostiene que la experiencia es aplicable al mismo equipo, y no se puede extrapolar a otros equipos distintos con contextos distintos. CONTRARIO A REPETITIVO.

No son procesos completos, por eso no se les suele llamar procesos. Mas bien son lineamientos, buenas practicas de alguna parte de un proceso. La idea es que el equipo las use y cierre y cree su propio proceso para el proyecto.

SCRUM NO ES UN PROCESO. SCRUM es un framework de trabajo para la gestión de productos y proyectos.

Ejemplo: movimiento Lean, movimiento Ágil.

Concepto de Ciclo de Vida.

No es lo mismo ciclo de vida que proceso.

Consiste en una serie de pasos a través de los cuales el producto o proyecto progresa.

Uno elige para un determinado proyecto un ciclo de vida.

Los productos también tienen su ciclo de vida. Son distintos de los de los de PROYECTOS, son mas largos. Un ciclo de vida de un producto inicia con la idea de crear un producto de software y termina cuando a ese producto lo retiro del mercado. Productos que tienen años de ciclo de vida. Puede haber n ciclos de vida de proyectos. Cada proyecto genera una versión que suele conocerse como un reléase o entregable del producto que tiene ciertas características y que es el que se pone en producción. Mientras se use un determinado producto de SW va a necesitar cambios.

El ciclo de vida para el desarrollo de un producto, que es el ciclo de vida del proyecto lo que hace es establecer cómo afrontar las actividades que plantea el proceso, en qué orden, cuanto de cada actividad vas a hacer.

Un ciclo de vida de un proyecto de software es una representación de un proceso. Grafica una descripción del proceso desde una perspectiva particular.

Los modelos especifican:

- Fases de proceso
  - Ejemplo: requerimientos, especificación, diseño, etc.
- Orden en el cual se llevan a cabo.

Clasificación de los ciclos de vida. Hay 3 tipos básicos de ciclos de vida para un proyecto de desarrollo de software.

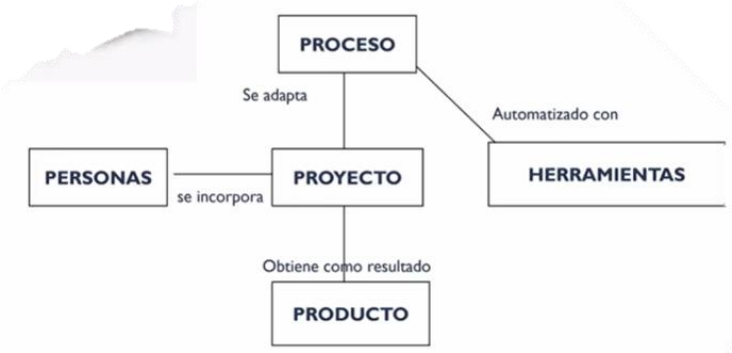
- Secuencial
  - Ciclo de vida en cascada. Hacer el 100% de cada etapa antes de pasar a la siguiente.
- Iterativo e incremental
  - Ciclo de vida iterativo e incremental. No se hace el 100% de las etapas, sino que se va haciendo un poco de todo y obtiene una versión o iteración.
- Recursivo: fracaso. Paso entre el secuencial al iterativo. Muchas vueltas, pero sin genera versiones intermedias del producto. Generas una versión que se pueda probar recién al final. No es el mismo resultado como cuando poner en producción el producto y el cliente puede ver las funcionalidades andar.
  - Ciclo de vida en espiral.

Para cerrar el vínculo entre procesos y ciclos de vida, cuando trabajamos con procesos definidos podemos elegir cualquier tipo de ciclo de vida de cualquier categoría.

Procesos empíricos no se pueden combinar con cualquier ciclo de vida. Solo con los iterativos e incrementales. No se puede implementar un proceso empírico con ciclo de vida de cascada porque el empirismo con una retroalimentación al final no es viable.

VINCULO ENTRE EL CICLO DE VIDA DEL PRODUCTO Y EL PROYECTO. Necesidad de cambios es el disparador en ambos.

RELACIÓN ENTRE: PROCESO, PERSONAS, PROYECTO, PRODUCTO, HERRAMIENTAS.



La disciplina de ingeniería de software se mueve en estas 3 dimensiones. Se agrega la 4ta dimensión que son las PERSONAS. El peopleware. Es el mayor factor de éxito. Hacer énfasis en las personas, no se puede dejar de configurar.

El proceso es como una definición de diccionario porque es donde está escrito en términos teóricos que es lo que se debería hacer para hacer software. Mas teórico**. Al momento de iniciar el proyecto, yo tendré que adaptarlo (proceso definido) o terminar de armarlo en el caso que sea empírico con las personas que están involucradas**.
El proceso se define en términos de roles. Para cada proyecto, las personas asumen uno o mas roles. El proceso se instancia en cada proyecto. Proceso es teórico y lo instancio en el proyecto.
En ese momento, se elige el ciclo de vida a utilizar. El ciclo de vida es el ciclo que va a tener el proyecto que será el encargado de gestionar a la gente y los recursos para poder obtener un producto de software que sea de calidad y satisfaga las necesidades de los usuarios.

La disciplina de software se ocupa en el producto como objetivo final. Un producto de calidad es la motivación fundamental.

Proyecto es el medio por el cual obtenemos un producto de software. Unidad organizativa. Medio por el cual administro los recursos que necesito y las personas que van a estar involucradas. El proyecto para poder existir necesita de una serie de elementos:

- Personas
- Recursos
- Método que establezca cómo hacer el trabajo. Aquí entra el proceso

El proceso da una definición teórica de qué es lo que debería hacerse para hacer software. Definición de un conjunto de actividades que toman como entrada requerimientos y obtienen como salida un producto o servicio en este caso de software. Ese conjunto de actividades está estructuradas y guiadas por un objetivo.

El proyecto debe tomar de la definición teórica del proceso de acuerdo y conforme lo necesite con un criterio de necesidad, corrección y adecuación y procede a armar el conjunto de tareas que es necesario hacer. Estas tareas conforman el alcance del proyecto (tareas que hay que hacer para cumplir con el objetivo).

El proyecto comienza decidiendo la gente que va a sumarse a trabajar o los distintos roles. Incorpora personas que van a tener roles para el determinado proyecto, estos roles están definidos en el proceso. Una persona puede asumir mas de un rol en un proyecto. Un rol puede ser adquirido por muchas personas.

El proceso se instancia en el proyecto. El proyecto le da vida al proceso. Las tareas se ejecutan en ámbito de proyecto, porque los procesos conforman la teoría a seguir.

CLASE GRABADA – COMPONENTES DE PROYECTO

¿Qué es software? Conocimiento empaquetado a distintos niveles de abstracción. Los entregables, bases de datos, diseño, casos de uso, user stories, manuales de configuración. Cada artefacto que sale de una tarea que se realizo en el contexto de un proyecto, es software.

¿Qué es un proyecto?

Característica = OBJETIVO

- Los proyectos están dirigidos a obtener resultados únicos. Cada resultado de un proyecto es distinto del resultado de otro proyecto. Aunque sea el mismo producto, el cambio está en que serán versiones distintas.
- Los objetivos guían al proyecto. Objetivo: obtener el producto único.
- Los objetivos no deben ser ambiguos. Debe ser claro
- Un objetivo claro no alcanza, también debe ser alcanzable. Debe poder lograrse con los recursos que podemos conseguir.
- Proyecto con objetivo CLARO y ALCANZABLE.

Objetivo del proyecto vs Objetivo del producto.
Se relacionan, pero no es lo mismo.
Objetivo del proyecto define el trabajo que el proyecto tienen que hacer para dejar contento al cliente.

Característica = DURACIÓN LIMITADA

- Un proyecto no es algo de flujo constante. No es una línea de producción en la que entramos materia prima y va saliendo producto.
- Empieza y termina sí o sí.
- Son temporarios, cuando se alcanzan los objetivos el proyecto termina.
- Cuando termina, liberamos los recursos.

Característica = tareas interrelacionadas basadas en esfuerzos y recursos

- Complejidad sistémica de los problemas.
- Divide y vencerás.
- Dividimos el trabajo que debemos hacer en tareas que tienen una relación. La interrelación genera dependencias inevitables y pueden ser determinantes.
- La definición de qué tareas tenemos que hacer la sacamos del proceso.

Característica = son únicos

- Todos los proyectos por similares que sean tienen características que los hacen únicos.

¿Los proyectos se planifican o no se planifican? Deberían estar planificados. Puede ser que no se empiece planificando, pero en cierto punto será necesario. Un proyecto planificado fracasa a veces, si no se planifica las posibilidades de fracasar son mayores.

¿Qué es la administración de proyectos o Project Management en inglés?

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

Incluye:

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

Se trata de lo siguiente: Quiero que el proyecto consiga el objetivo exitosamente entonces administramos todos los recursos que nos dieron y vamos a organizar el trabajo de la gente que está afectada y haremos un seguimiento de si las cosas se están dando como nosotros establecimos que se iban a dar. Esas actividades que intentan controlar y conseguir el objetivo se llama administrar un proyecto.

Tenemos gente, recursos y objetivo/s. gestión tradicional de los proyectos. Hay una figura de un líder de proyecto que es el responsable de administrarlo. En términos concretos, la administración de proyectos tiene 2 grandes actividades:

- Planificación
- Seguimiento y monitoreo de que las cosas estén pasando como fueron planificadas.

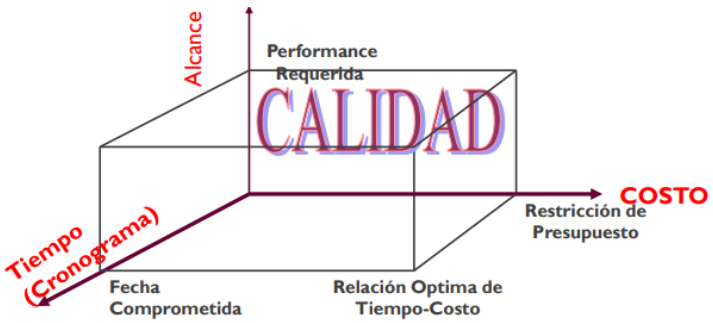
Son dos disciplinas a las que hacemos referencia a cuando hablamos de administrar un proyecto.

Distintos enfoques para administrar un proyecto:

- Administración de proyectos agiles. Se analizan en comparación con los tradicionales.
- Administración de proyectos tradicionales.

Triple restricción.

- Objetivos de proyecto: ¿qué está el proyecto tratando de alcanzar? Alcance/requerimientos que el cliente dio.
- Tiempo: ¿cuánto tiempo debería llevar completarlo?
- Costos: ¿cuánto debería costar? Recursos y gente que terminan generando un costo.



El balance de estos 3 factores afecta directamente la calidad del proyecto.

“Proyectos de alta calidad entregan el producto requerido, el servicio o resultado, satisfaciendo los objetivos en el tiempo estipulado y con el presupuesto planificado”

No esta bueno que los 3 aspectos sean FIJOS. Dimensión que normalmente no se controla, nunca depende de nosotros. ALCANCES. El producto es del cliente.

Dimensiones tiempo y costos si debiéramos poder determinarlas.

Si el cliente cambia el alcance, por consecuencia cambian las otras 2.

Rol del líder de proyecto/equipo. Enfoque tradicional

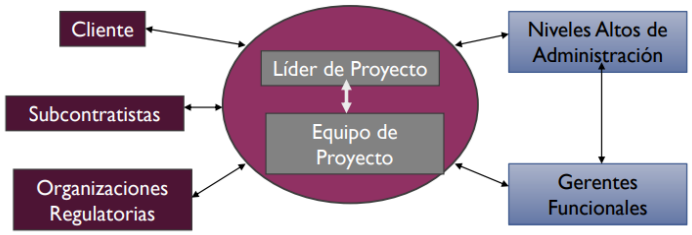
Líder es el que se relaciona con los demás involucrados por fuera del equipo.

Es conductista. Establece lo que hay que hacer y tiempos de demora.

Encargado de informar. Tanto de logros como desviaciones.

Responsabilidad de estimar, planificar, asignar trabajo a la gente, realizar seguimiento.

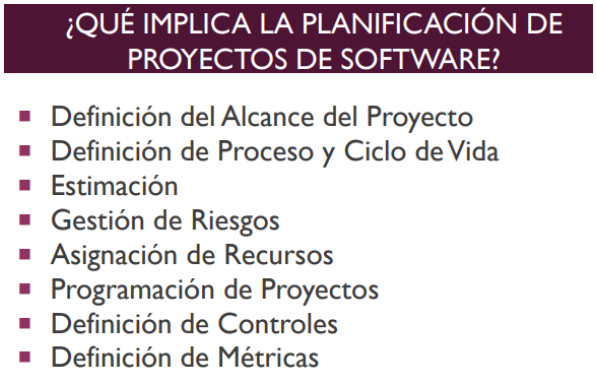
No suele tener trabajo técnico.



Plan de proyecto.

Documenta y debemos responderlo en un plan de proyecto:

- Qué es lo que hacemos
- Cuando lo hacemos
- Como lo hacemos
- Quien lo va a hacer



Alcance de producto vs alcance de proyecto.

Producto. La sumatoria de todos los requerimientos funcionales y no funcionales que ese producto tiene que satisfacer. Se guardan en la ERS.

Proyecto. Trabajo, tareas que tengo que hacer para cumplir con el objetivo del proyecto. Se guardan en el plan de proyecto. Necesito tener definido antes el alcance del producto.

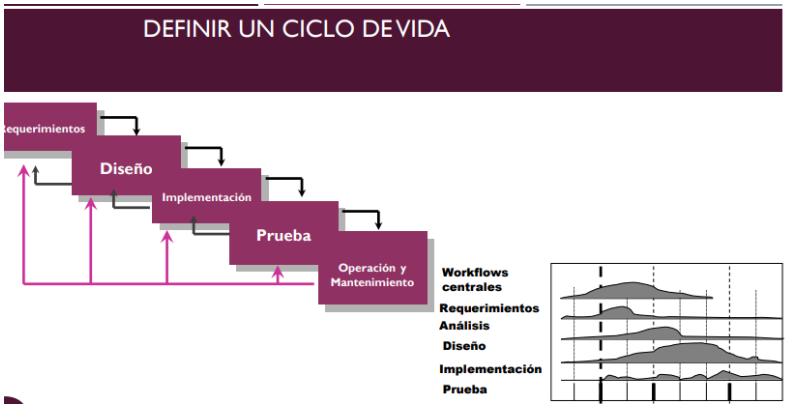
Si el producto que tengo que construir es mas grande, las tareas del proyecto van a ser mas y necesitar mas tiempo o gente.

Ciclo de Vida.

No es lo mismo que proceso.

El proceso define las tareas que hay que hacer y el ciclo de vida te dice como hacer esas tareas, cuanto de cada tarea vas a hacer cada vez.

Saber que cuando inicia el proyecto, tenemos que elegir que proceso quiero usar y qué ciclo de vida quiero usar.



Estimaciones de Software en la gestión tradicional

Tienen un orden.

1. Tamaño del producto de software.
2. Esfuerzo. Horas de trabajo concentrado, personas.
3. Calendario. Plasmarlo en días del calendario. Fecha de entrega
4. Costo.
5. Recursos críticos. Cosas raras que nos hacen falta. No entra una pc.

CLASE GRABADA N° 4 – USER STORIES Y REQUERIMIENTOS AGILES

FALTA ANOTAR LO PRIMERO QUE DIJO

Requerimientos Agiles.

Pilares de los requerimientos agiles.

- Construir el producto correcto que le agregue valor al negocio. Solemos estar muy enfocados en el software, pero hay que entender que el software es una herramienta de trabajo para el cliente. Lo que le tenemos que entregar al cliente es valor de negocio, no características de software.

- Determina que es solo lo suficiente. Mínimo y necesario para comenzar. Encontramos los requerimientos de a poco. La realidad nos muestra una situación en la que del total de las características que tiene un producto se software, únicamente el 7% de ellas se usan siempre y un 45% de las mismas no se usan nunca.

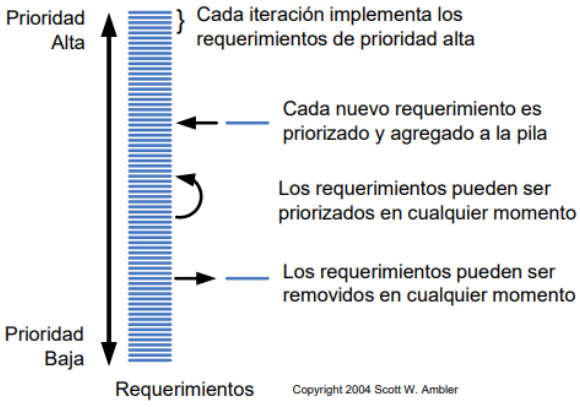
- La parte de desarrollo de requerimientos está muy enfocada a la idea de construir junto con el cliente. Proceso de descubrimiento muy integrado entre el cliente y el nosotros. A través de historias de usuarios y modelos que construiremos las usamos como base para ir armando y modelando la definición del producto.

PRINCIPIO del Manifiesto Ágil RELACIONADO.

- El método más eficiente de comunicar información es con conversaciones cara a cara. Porque la idea es armar equipos de trabajo en los que estas técnicas de descubrimiento se pueden lograr crear la visión del producto que queremos conseguir y a partir de ahí avanzar en las siguientes etapas.

Gestión ágil de requerimientos. Los requisitos cambiantes son una ventaja competitiva si se puede actuar sobre ellos. Intenta equilibrar la situación en la que nos encontramos con muchos requerimientos en el producto que no terminarán utilizándose.

El dueño del producto que se llama PO, es el que realmente tiene claras cuales son sus necesidades y su responsabilidad principal el PRIORIZAR REQUERIMIENTOS. Decide qué es lo que el necesita primero. De modo que el equipo se ocupe primero de eso.



Product backlog. Es una lista organizada y priorizada donde lo mas importante se acomoda arriba y lo que tiene menos prioridad se ubica debajo. En este espacio el dueño es el PO. Es el que toma la decisión de priorizar y ordenar lo mas urgente.

Enfoque just intime de los requerimientos agiles. Se usa fundamentalmente para evitar desperdicios. Desperdicio es especificar requerimientos que después cambian y yo inverti un montón de tiempo en especificarlos. El producto no estará especificado al 100% desde el inicio, se irá encontrando requerimientos y describiendo los requerimientos conforme sea necesario. Ni antes ni después. Si llega antes es un desperdicio, pero si llega después es un problema porque el equipo tendrá que hacer correcciones.

Tradicional vs Ágil.

Desde el punto de vista de los requerimientos agiles hay que ver al triangulo de forma invertida.

Dejamos fijo el tiempo con iteraciones de duración fijas llamadas sprints por Scrum.

Dejamos fijo los recursos que es el equipo de trabajo con una determinada capacidad que se estima junto con los recursos de infraestructura y comunicación necesarios.

Si tenemos tiempo y equipo fijos, decimos: cuanto del producto podemos construir con este equipo en este tiempo. Se acuerda un determinado alcance para esa estimación de tiempo.

Se llega a un acuerdo de qué es lo que el equipo puede entregar del producto funcionando en esta iteración.

Tipos de requerimientos.

A nivel de la gestión ágil de requerimientos, trabaja a nivel de requerimiento de **negocio y de usuario**.

Los requerimientos del usuario se alinean con los requerimientos del negocio.

Tarea del PO. Identificar necesidades de usuario que satisfagan necesidades de negocio.

User stories. Sirven para especificar requerimientos de usuario.

RESUMEN DE GESTIÓN ÁGIL DE REQUERIMIENTOS

Debemos trabar juntos técnicos y no técnicos (PO) entendiendo las necesidades del negocio.

Luego, junto con los usuarios descubrir cual es la mejor forma de satisfacer esas necesidades. Aquí aparece el producto backlog. Determinar y priorizar los requerimientos más urgentes a entregar en la primer iteración.

De allí obtenemos feedback. Nos ayuda a ir modificando el producto backlog.

Hay que saber que:

- Los cambios son una constante. Siempre pueden surgir.
- Debemos tener una mirada de quienes son los involucrados. Son todos los que tienen algo para decir respecto del producto. Stakeholders, son todos ya sea estén o no. El producto owner es el representante de todos los que en el negocio tienen algo para decir sobre el producto.
- El usuario dirá si realmente está contento o no cuando reciba un producto funcional. Por eso es el apuro en que se entreguen versiones en cortas iteraciones. Permite eliminar incertidumbre de forma más rápida.
- No hay técnicas ni herramientas que sirvan para todos los casos. Aprender en cada situacion, cual es la herramienta mas adecuada.
- Lo importante no es entregar una salida, un requerimiento, lo importante es entregar un resultado que dé una solución de valor.

Principios del manifiesto ágil relacionados a los requerimientos agiles.

1. La prioridad es satisfacer al cliente a través de releases tempranos y frecuentes. De 2 semanas a un mes.

2. Recibir cambios de requerimientos aún en etapas finales.

4. Técnicos y no técnicos trabajando juntos todo el proyecto

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

11. Las mejores arquitecturas, diseños y requerimientos emergen de equipos auto organizados. Decisiones técnicas dejárselas al equipo y no exigirles desde el inicio.

User Stories.

Una historia de usuario proviene de un usuario del futuro producto de software que estamos creando. Este usuario nos dirá algo que él mismo considera importante que necesita para su negocio.

Descripción corta de una necesidad que tiene un usuario respecto del producto de software.

La parte más difícil de hacer software es DECIDIR QUÉ SOFTWARE QUEREMOS. Esa parte descansa sobre la figura del product owner.

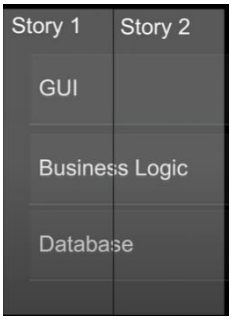
Es una técnica de identificación de requerimientos de usuarios.

Partes de una User Story.

- Conversación. No es visible.
- Tarjeta. Nomenclatura definida.
    - **¿Quién es el que necesita esto?** Rol
    - **¿Qué necesita?** Acción que realizará el sistema
    - **¿Para qué?** Valor de negocio. IMPORTANTE para que el PO pueda priorizar el producto backlog. Es muy difícil priorizar si no.
- Confirmación. Pruebas de usuario que se identifican necesarias para hacerle después a la funcionalidad una vez que esté construida, que servirían para que el PO apruebe la característica de software que estamos construyendo a partir de la US.

Las user stories son **multipropósito**.

- Una necesidad del usuario
- Una descripción del producto
- Un ítem de planificación
- Token para una conversación. Recordatorio
- Mecanismo para diferir una conversación.
- **NO** son una especificación de requerimiento.



Las user stories se incorporan al producto balcklog en el orden que el PO indique según prioridades. El dueño del producto backlog es el PO.

Son **porciones verticales**. Si las cortamos horizontales no entregan valor. Solo entregamos interfaz, o lógica de negocio o base de datos. No funcionalidad. El usuario no puede hacer nada con eso.

Modelado de roles. Describir los tipos de usuarios. No sirve que los definamos únicamente como USUARIOS.

Mas características:

- 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.

Diferentes niveles de abstracción.

- User story
- Epicas. Una user story grande. No la puedo ejecutar en una iteración
- Tema. Colección de user stories relacionadas.

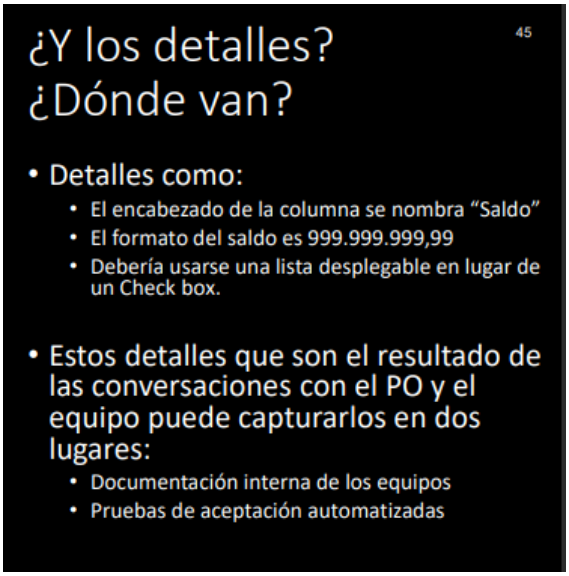
Usuarios Representantes. Proxies. Reemplazan al PO.

Criterios de aceptación de User Stories.

Es información concreta que tiene que servirle al equipo para saber si lo que implementamos es correcto o no.

- Definen limites para una user story
- 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 cuando parar de agregar funcionalidad en una US.
- Definen una intención, no una solución
- Son independientes de la implementación.
- Relativamente de alto nivel, no es necesario que escriba cada detalle.
- No se escriben en términos informáticos ni de software.
- Debe ser medible.

Pruebas de aceptación de User Stories.



Expresan detalles resultantes de la conversación.
Tienen que contemplar situaciones de éxito y fracaso.

No es conveniente que el usuario final no participe en la elección de las pruebas de usuario. Si los hacemos parte luego sabremos que es lo que el querrá que funcione y saber los escenarios que el tomaría como exitosos.

Complementan la US

Proceso de 2 pasos:

- Identificarlas al dorso de la US
- Diseñar las pruebas completas.

Definición de listo – Definition of Ready

Es una medida de calidad que construye el equipo en su conjunto para poder determinar que la user está en condiciones de entrar a una iteración de desarrollo. Lista para empezar a implementarla.
Como mínimo debe cumplir el invest model.
Si no lo cumple, queda en una posición mas abajo en el producto backlog.

Definición de Hecho – Definition of Done

La Definition of Done es un conjunto de reglas que determinan cuándo un elemento está terminado. Terminado significa listo para poner en producción a disposición del usuario.

Un checklist exhaustivo que asegura que únicamente los aspectos completamente hechos son entregados al cliente, no sólo en términos de funcionalidad, si no de calidad también.

Modelo de calidad de base.

Ayuda a saber que las US está en condiciones de incluirse en una iteración de desarrollo.

- **Independiente**. Le doy la libertad al PO de que las priorice para que se puedan desarrollar en cualquier orden. Calendarizables e implementables en cualquier orden.
- **Negociable**. Escrita en términos de qué necesita el usuario, no de cómo lo vamos a implementar. El qué no el cómo.
- **Valuable**. Debe tener el para qué. Debe brindar valor al negocio.
- **Estimable**. Asignarle un peso que me permita compararla con otras y poder determinar cuanto esfuerzo voy a necesitar yo para implementarla. the conversation surrounding the estimation process is as (or more) important, than the actual estimate
- **Chica**. Debe ser consumida en 1 iteracion. No hay trabajo a medias. Pequeño. Depende del equipo y su experiencia.
- **Testeable**. Debe demostrar que fueron implementadas.

CLASE GRABADA – GESTIÓN DE PRODUCTOS

¿Por qué hacemos productos? Qué nos motiva.

- Satisfacer a los clientes
- Para tener muchos usuarios logueados
- Para obtener mucho dinero
- Realizar una gran visión, cambiar el mundo.

Las motivaciones son las que nos impulsan.

Evolución de los productos de software.

La base es que cumpla con los alcances establecidos siendo funcional y util para lo esperado.

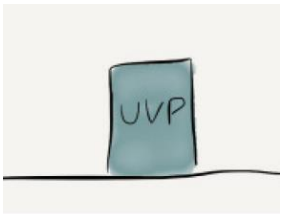
Confiable. Un usuario tiene que sentirse cómodo de que no corre peligro al utilizar ese producto. Si dependemos de los resultados de ese producto de software la confiabilidad es esencial.

Usable. Mejore la calidad de vida.

Conveniente. Supere otros. soy mas productivo que antes, logro cosas que antes no lograba. Genere un diferencial con respecto a otros.

Poner la atención en = focalizarlo en las experiencias.

Explicación MVPs, MVFs, MMFs con el dinosaurio de Lean/Agiles

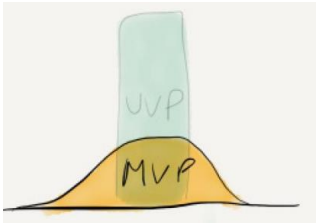


Comprender → un producto nuevo tiene una hipótesis de valor único. Producto/servicio será único.

Comienza con una idea de valor y producto único. Podemos tener una idea de hacia dónde llegar, pero no podre llegar de un solo salto, de allí nace MVP.

El siguiente paso es crear un producto mínimo viable (MVP) → para probar su hipótesis.

MVP. Producto Mínimo Viable. Es la definición de lo mínimo que yo necesito para como hipótesis probar si estoy alineado en el producto correcto o no. El propósito fundamental es validar un concepto. ¿es esto? ¿o estoy equivocado? La idea es hacerlo con poco tiempo y poca plata. A esa hipótesis la podemos tirar a la basura porque no sirve. No debemos tardar ni invertir mucho en obtenerla.



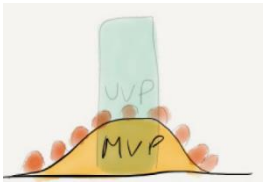
Esto se centra en su propuesta de valor única, pero normalmente también proporciona un poco de funciones de "Table stakes" sólo para asegurarse de que sea "viable" como producto.

En el contexto de gestión de productos, las funciones de tipo "table stakes" son elementos fundamentales que los clientes y usuarios esperan y dan por sentado en un producto o servicio.

Estas funciones son necesarias para ingresar al mercado y mantenerse competitivo, pero no necesariamente proporcionan una ventaja competitiva significativa por sí solas. Por lo tanto, cumplir con los "table stakes" es un requisito previo antes de que una empresa pueda considerar ofrecer características adicionales o diferenciarse de la competencia.

El término proviene del mundo del póker, donde "table stakes" se refiere al dinero mínimo que un jugador debe tener en la mesa para participar en una partida.

Generamos el MVP y salgo a la búsqueda de clientes potenciales que lo validen, es decir, salgo a mostrarlo para que el cliente interactúe y nos de feedback.



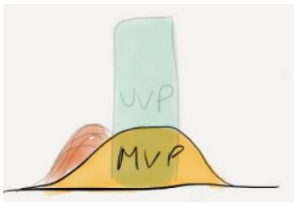
Tu MVP también es una hipótesis. Podría ser lo suficientemente bueno para encontrar un mercado o no.

Pueden suceder dos escenarios, que el feedback era esperable o puede suceder que el cliente le haría ciertos cambios.

Se muestra el caso en el que cada cliente potencial con el que interactúas te dice "Esto es genial, pero para poder usarlo necesito X" y X es diferente para cada cliente / usuario.

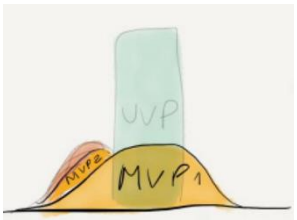
Esto muestra que aún no se encuentra un mercado para el producto.

Es útil y valorable la posibilidad de poder valorarlo con los clientes. Nos abre la posibilidad a encontrar características que hasta el momento no habíamos precisado.



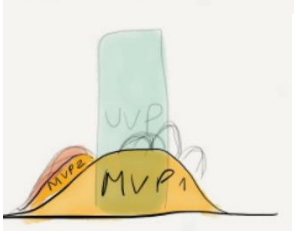
Si por el contrario ves cada vez más respuestas apuntando al MISMO X entonces tiene sentido revisar la hipótesis de Cliente/Problema/Solución

Esta situación produce una variación significativa del producto (como si cambiara de eje). En la que haya una tendencia de varios clientes hacia determinados cambios.



Básicamente, estás ejecutando un pivot. Está construyendo MVP2 centrado en la nueva hipótesis basada en el aprendizaje reciente de Desarrollo de clientes generado por el anterior MVP.

Supongamos que MVP2 es exitoso y está viendo una tracción real de los primeros usuarios.

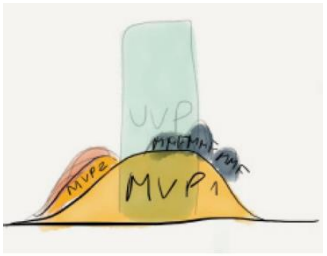


Algunos de ellos por cierto amplían su propuesta de valor única y algunos hacen que su producto actual sea más robusto.

MVF. Caracteristica minima viable.

El MVF es parte del MVP porque un producto va a tener un conjunto de caracteristicas, hay caracteristicas minimas que van a formar parte del MVP. El proposito es el feedback. Ver si voy o no por el camino correcto o no. Análisis de viabilidad para validar una hipotesis.

MMF. Conjunto de caracteristicas minimas comercializables.



Hablamos del conjunto minimo de caracteristicas tiene que tener mi producto para que yo lo pueda vender en el mercado. El proposito es dinero.

El objetivo del MMF es aportar valor. Supone una alta certeza de que existe valor en esta área y que sabemos cuál debe ser el producto para proporcionar este valor.

La razón para dividir una característica grande en MMF más pequeños es principalmente el tiempo de comercialización (Time to market) y la capacidad de aportar valor en muchas áreas.

Una indicación de que está trabajando en MMF es que cuando al liberarse uno se siente cómodo trabajando en el próximo MMF en esa área.

Asociado con una presion proveniente del marketing. Conseguir inversores.

Ahora tu hipótesis se centra en una característica en lugar del producto.

Tienes un área con alto potencial pero también alta incertidumbre.

Escenario de mucho riesgo.

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). Si no es así, puede cambiar a otro enfoque hacia ese área de características, o en algún momento buscar una ruta de crecimiento alternativa.

Esencialmente, **el MVF es una versión mini del MVP**.

Diferencia entre MVP y MMF.

MMF es un producto que va a ofrecerse con un resultado concretos que se va a obtener.

MVP es una hipotesis de producto, te permitis que no ande y que sea un prototipo. Un video.

MRF. Caracteristicas minimas del release. Concepto encaminado hacia un producto maduro que sale al mercado. Un release es una version del producto que tiene la madurez suficiente como para salir al mercado. Conjunto de caracteristicas minimas que se esperan que ese release tenga para cumplir con el objetivo del release. Un release se va a obtener pasadas varias iteraciones de desarrollo.

La idea se centra en saber que un producto se desarrolla incrementalmente, y que podemos empezar con un conjunto muy pequeño de caracteristicas y mientras mas cerca estemos de formar una hipotesis que queremos validar, mientras menos tengamos, MEJOR. Perdemos menos tiempo y plata. Sacarnos la idea que el producto tiene que estar 100% especificado al inicio.

Nos motiva generar un producto de valor. El análisis es complejo, si no genera valor es desperdicio.

Relación entre MVP, MMF, MMP, MMR.

Comenzamos con un MVP (Producto minimo viable), es la idea de hipotesis de caracteristicas que queremos validar. Es raro que salgamos a validar con 1 sola caracteristica que seria un MVF, una sola es raro.

El MVP tiene que evolucionar a algo que sea vendible, da origen al producto minimo comercializable (MMR) Generalmente un MMR está compuesto de una sumatoria de MMF.

Si cuando ese MMR salio al mercado y funciono exitosamente ese MMR, pasa a ser el primer release. Aparece el MRF1.

Si sucede que en algun punto nos encontramos con una realidad de cierta caracteristica del producto para la que se va desviando mi producto que quiero validar, puede aparecer un nuevo MVP.

Mientras el producto se mantenga en el mercado siendo exitoso siempre se generará nuevos releases MRF2, 3 … Mientras el producto se use, va a necesitar cambios agregados, nueva tecnología, etc.

Usamos al MVP como un medio para generar un producto que de valor y eliminar desperdicio. La idea es que por medio de ciclos cortos, poder aprender y retroalimentar para determinar cual es el producto correcto que vamos a construir.

Dijo profe práctico:

Minimo producto que necesito construir para salir a al mercado. La idea de este producto es poder validar una hipotesis.

Se hace para invertir lo menos posible en desarrollo y ese producto tendrá éxito, si habrá usuarios que deseen usarlo. Antes de gastar fortunas desarrollando algo que no se sabe si tendrá éxito, se debe desarrollar el MVP.

Puede ser:

- Prototipo funcional
- Video
- Maqueta

RESUMEN.

MVP. Version de un nuevo producto creado con el menor esfuerzo posible. Dirigido a un subconjunto de clientes potenciales. Utilizado para el aprendizaje validado. Mas cercano a los prototipos que a una version real funcionando de un producto.

MMF. Es la pieza más pequeña de funcionalidad que puede ser liberada.tiene calor tanto para la organización como para los usuarios. Es parte de un **MMP** o **MMR**.

MMP. Primer release de un MMR dirigido a primeros usuarios. Focalizado en caracteristicas clave que satisfarán a este grupo clave.

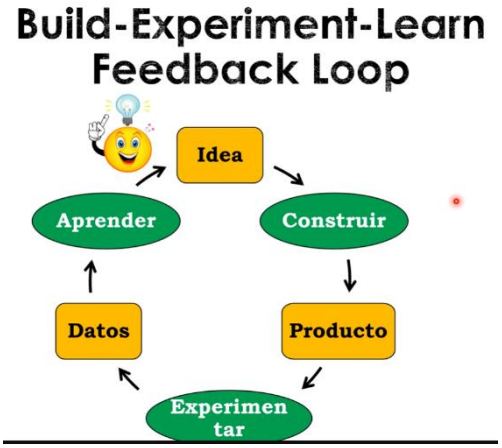
MMR. Release de un producto que tiene el conjunto de caracteristicas más pequeño posile. El incremento más pequeño ofrece un valor nuevo a los usuarios y satisface sus necesidades actuales. MMP = MMR1

FOCO EN EL MVP.
Tiene un objetivo concreto que es una hipotesis a validar. Necesito saber si voy a tener clientes que me compren el producto o no.

Lo que busco con el MVP es feedback rápido a un costo razonable y una decisiones de si vale la pena invertir en eso o no. Hipotesis a validar. ¿Es el producto correcto o no?

ERROR. Confundir a MVP con producto comercializable MMF o MMP. Porque los dos se encuentran en estados distintos uno está apto para la validación y el otro está apto para la comercializacion. Si queres salir a vender un producto que no está en estado de madurez de comercializacion sino que en estado de madurez de validacion de hipotesis puede conducir a un fracaso.

Valor vs. Desperdicio
Si no genera valor de negocio es desperdicio.
Creacion de valor es proveer beneficios a los clientes.



Ciclo en el que se enmarca el MVP →

Lo que nos motiva llevar a la creacion de valor es definir una HIPOTESIS de valor que sea nuestro motor. Vamos a utilizar este ciclo como un medio para generar el MVP.

El MVP puede moverse en la linea de ser algo funcional dinamico. Codigo ejecutable, prototipo, prototipo en papel, video. No necesariamente un producto funcionando.

CLASE GRABADA – ESTIMACIONES

Una estimación tiene que terminar entregando un número. El proceso de estimación obtiene como resultado un valor cuantitativo, una medida que tiene que ser comparable con otra medida. Puede ser mas o menos precisa, pero tiene que ser medible y comparable.

¿en qué momento se estima? Al principio. Pero posteriormente, debemos estar preparados para re-estimar. A más información, más eficiente será la estimación.

El problema de las estimaciones es que la palabra estimación tiene asociado un concepto que muchas veces se olvida, ese concepto es PROBABILIDAD. Las estimaciones tienen un valor de probabilidad asociado. Puede salir bien o no.

Uno planifica basándose en una estimación.

IDEA. Pensar todo lo que puede salir mal, agregar el tiempo, y multiplícalo x4 y aun así te vas a quedar corta.

¿Para qué estimamos?

- Por definición, **una estimación no es precisa**. Habrá errores, intentar que sean mínimos. Pueden venir del mismo proceso de estimación o por actividades omitidas por ejemplo cuando solo se estima el tiempo de programación. Problema de quién estima. No es lo mismo que estime un experto que alguien que no conoce sobre el trabajo a realizar.
- **Estimar no es planear y planear no es estimar**.
- Las estimaciones son la base de los planes, pero los planes no tienen que ser lo mismo que lo estimado.
- Sirve como respuesta temprana sobre si el trabajo planificado es temprana o no.
- A mayor diferencia entre lo estimado y lo planeado, mayor riesgo
- Las estimaciones no son compromisos.

Generalmente al principio no tenemos demasiada información para estimar.

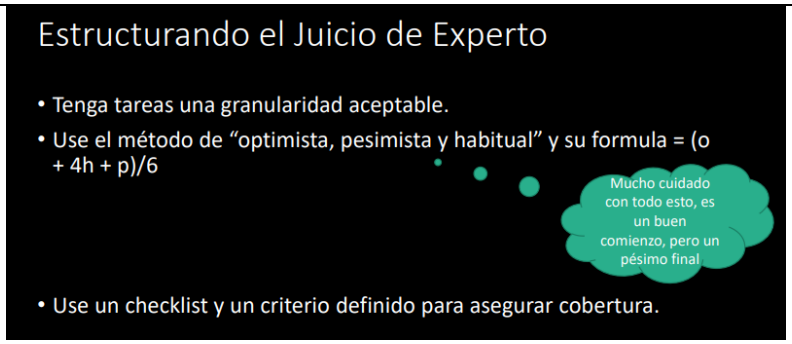
Si los requerimientos cambian, muy probablemente cambien las estimaciones también.

Técnicas fundamentales de estimación.

Cuando estimamos tenemos que contar algo. Al trabajar con algo intangible se dificulta el hecho de realizar el conteo.

Métodos utilizados.

- Basados en la experiencia
  - **Datos históricos**: lo usan los procesos definidos, asumiendo la base que la experiencia de otros proyectos y de otra gente, me puede servir a mí. Se refiere a la práctica de utilizar información y registros previos de proyectos o eventos similares para predecir el tiempo, costo, recursos u otros aspectos relacionados con un nuevo proyecto o evento. Este enfoque se basa en la premisa de que las experiencias pasadas pueden proporcionar información valiosa para estimar el rendimiento futuro. Datos históricos que necesito: tiempo que me llevo, gente que trabajó, tecnología utilizada.
  - **Juicio experto**
    - **Puro**: adentro de la empresa tenemos un gurú estimador que establece que tal cosa estará para X día. Nunca se sabe experto en qué es, estima desde lo que es complejo o no para él. Desviaciones en la realidad. Un experto estudia las especificaciones y hace su estimación. Se basa fundamentalmente en los conocimientos del experto. Si desaparece el experto, la empresa deja de estimar. Es un riesgo.



  - **Delphi**: se toma como base para el poker planning. Se estima en grupo. Grupo de personas que estime y se pongan de acuerdo en un numero que converja y que esté ajustado en la realidad.
    Un grupo de personas son informadas y tratan de adivinar lo que costará el desarrollo tanto en esfuerzo, como en duración.
    Las estimaciones en grupo suelen ser mejores que las individuales.
    Se dan las especificaciones a un grupo de expertos. Se les reúne para que discutan tanto el producto, como la estimación.
    Remiten sus estimaciones individuales al coordinador. Cada estimador recibe información sobre su estimación y las ajenas, pero de forma anónima.
    Se reúnen de nuevo para discutir las estimación. Cada uno revisa su propia estimación y la envía al coordinador.
    Se repite el proceso hasta que la estimación converge de forma razonable.

  - **Analogía:** Ajuste del método de datos históricos. De la base de los datos históricos, hacer una abstracción de los que son parecidos a este proyecto. De todos los datos históricos me fijo los que con coincidentes con el trabajo actual.
- 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

Actividades omitidas.

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

Estimaciones ágiles

Enfoque de estimaciones agiles apunta a un corregir los problemas que se identificaron en las estimaciones de software tradicionales.

Algunos de los problemas de estimaciones tradicionales.

- Estimar demasiado pronto
- Resistencia a las re-estimaciones
- Estimaciones que buscan precisión
- Estimaciones que las hace gente distinta que la que va a hacer el trabajo.
- Estimaciones que pretender ser absolutas.

Hay una fuerte propuesta de que estime quien hace el trabajo.

Las estimaciones son un proceso y uno aprende en el proceso, en el día a día. Todo el mundo debe entender que una estimación tiene un valor de probabilidad asociado que no hay certeza y que no son compromisos. La idea es empezar con algo que después se pueda revisar todas las veces que haga falta revisarlo.

Aplica el principio ágil: La mejor métrica de progreso es el software funcionando para llegar a trabajar exitosamente ese principio, tenemos que entender que todo lo que nosotros definimos en cuanto a estimaciones tienen que ver con producto.

A diferencia del orden establecido en el enfoque tradicional:

Tienen un orden.

1. Tamaño del producto de software.
2. Esfuerzo. Horas de trabajo concentrado, personas.
3. Calendario. Plasmarlo en días del calendario. Fecha de entrega
4. Costo.
5. Recursos críticos. Cosas raras que nos hacen falta. No entra una pc.

Características de las estimaciones agiles:

- Si las estimaciones se utilizan como compromisos son muy peligrosas y perjudiciales para cualquier organización.
- Lo más beneficioso en las estimaciones es el “proceso de hacerlas”.
- La estimación podría servir como una gran respuesta temprana sobre si el trabajo planificado es factible o no.
- La estimación puede servir como una gran protección para el equipo

1) Son relativas: en contraposición a las tradicionales que son absolutas. Los seres humanos son mejores comparando, además es más rápido, qué tanto más. Se hace estimaciones por comparación en el enfoque ágil (poker planning o poker estimation → con US canónica)

¿Dónde se estima? → En dos momentos: primero más generalizado y después durante la sprint planning (dos momentos: uno en el que se estiman las historias y después cómo se piensas que se lleva a la implementación esas historias).

Priorización → tiene q ver con el valor de negocio.

Estimación → tiene q ver con cuánto esfuerzo y recursos le conlleva al equipo.

2) Foco en certeza → Se hace foco en la certeza y no en la precisión, porque esta es cara. Los enfoques tradicionales hacen mucho énfasis en la precisión y luego no suelen cumplirlos. No hacer el esfuerzo de estimar todo el producto.

3) Diferir las decisiones → hasta el último momento porque en el proceso se va obteniendo retroalimentación y se va obteniendo certeza de lo que se está haciendo, al principio no se sabe. Estimaciones tempranas, las menos posibles.

4) Estima el que hace el trabajo: que es el equipo, a diferencia de los tradicionales que quien estima es el líder del proyecto que muchas veces no tiene nada que ver con el día a día del equipo. No quedarse con una sola idea, tener en cuenta los demás.

La gente de ágil sostiene que debemos estimar el peso de las características de software. Se puede estimar a nivel de user story pero también a partir de cosas mas grandes como las éticas o mas aun como los temas. Mientras mas grande, usamos unidades de medida más laxas.

Las US son la unidad de gestión dentro de ágil. Que son las que entran a la planificación de una iteración para ser desarrolladas, se utiliza una unidad de medida STORY POINT. Las medidas relativas no son absolutas.

Story Points no es una medida basada en tiempo. Peso de la US y lo mas importante es que ese peso lo utilizamos, pero por comparación. De ahí proviene la …

Medida relativa según el equipo.

ESTIMACIÓN RELATIVA

- Somos mejores estimando por comparación, además de que es más rápido.
- No en términos absolutos.
- Se obtiene una mejor dinámica grupal y pensamiento de equipo mas que individual.

El Story Point es una unidad homogeneizadora que nos permite comparar. No sabríamos cómo estimar el peso o valor de cada US.
Es una unidad de medida especifica del equipo que representa complejidad, riesgo y esfuerzo. Sotry point da la idea del peso de cada story y decide cuan grande es.

3 dimensiones que conforman el peso del Story Point:

- Complejidad.
- Esfuerzo.
- Incertidumbre. Cuanta información nos falta para ser más adecuados en lo que estamos trabajando.

Tratan de no estimar todo al principio.

Tratan de introducir en la cabeza del cliente que las cosas van a cambiar y que no tiene mucho sentido dar una estimación completa al principio si tenemos en cuenta los cambios. Se hace gradualmente para ir corrigiendo y perfeccionando.

Como el esfuerzo es la materia prima más significativa del costo de un proyecto que se representa concretamente en horas. Tendencia a confundir tamaño con esfuerzo. Y confundir esfuerzo con calendario.

El esfuerzo es único y personal por persona.

Se toman horas ideales. En cuanto a horas dedicadas al trabajo es mas sencillo. El calendario debería tomarse como una derivación.

Velocidad. Sumando story points puedo calcular una métrica ágil llamada velocidad que es cuanto producto yo le entregue al cliente al final de una iteración. Métrica que no se estima. Se calcula al final de al iteración.

Corrige los errores de estimación.

CLASE GRABADA – GESTIÓN DE CONFIGURACIÓN DEL SOFTWARE

Concepto de Gestión de Configuración de Software. Disciplina protectora, que ocurre transversalmente a lo largo de todo el proyecto, pero más allá para darle soporte a todo el ciclo de vida del producto.

Una disciplina que aplica dirección y monitoreo administrativo y técnico a: identificar y documentar las características funcionales y técnicas de los ítems de configuración, controlar los cambios de esas características, registrar y reportar los cambios y su estado de implementación y verificar correspondencia con los requerimientos. (ANSI/IEEE 828, 1990)

Disciplina protectora. Es la responsable de mantener la integridad del producto.

Mantiene la integridad del producto. ¿Qué es la integridad del producto?

- Satisfacer las necesidades del usuario
- Fácil y rastreable durante su ciclo de vida
- Satisface criterios de performance
- Cumple con expectativas de costo

De quién es responsabilidad el SCM → todos tenemos responsabilidad por el producto de software que estamos construyendo entre todos, cada vez que alguien crea un item de configuración es responsable por respetar los lineamientos que el equipo definió respecto de como mantener la integridad del producto. Todos son responsables.

Gestor de configuración → Tareas adicionales, mantener la herramienta, marcar líneas base, organizar los procesos de cambio.

Principal problema del software:

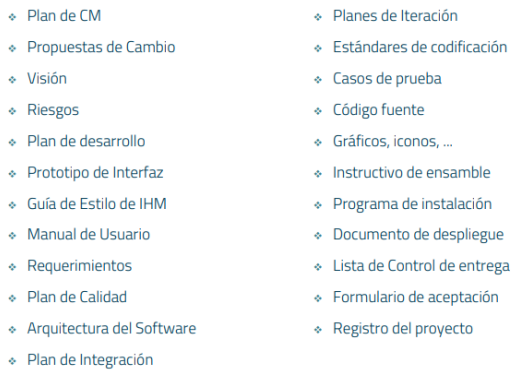
- Cambios en el software, es muy fácil modificarlo y con un solo click cambiar todo. Maleabilidad, facilidad de destruirlo, cambiarlo.

Propósito de la disciplina: evitar que ocurra este problema. Tratar que el producto esté integro, si vamos a hacer un cambio, se tenga control de qué es lo que cambió en un determinado momento, por eso se asocia el concepto de versión. Cada ítem de configuración debe tener su versión. Certeza de cómo es la conformación del producto en un momento determinado. Establecer y mantener la integridad de los productos de software a lo largo de su ciclo de vida, desde que se concibe, incluyendo todos los ítems de configuración que se conciben durante su desarrollo y evolución hasta que el producto se deja de usar.

Disciplinas de soporte del software

- Administración de configuración de Software.
  - Control de calidad de proceso
  - Control de calidad de producto
  - Prueba de Software

ítems de configuración → Word, foto, prototipo, código. No importa el formato, mientras que se trate de un archivo almacenable. Todos y cada uno de los artefactos que forman parte del producto o del proyecto, que pueden sufrir cambios o necesitan ser compartidos entre los miembros del equipo y sobre los cuales necesitamos conocer su estado y evolución.



Ej: user stories, requerimientos, fotos de tableros. Tiene que ser un elemento que se pueda guardar en un file sistema

Repositorio → es donde van a parar los ítems de configuración. Contenedor de ítems. Debe tener una estructura. Mantiene la historia de cada IC con sus atributos y relaciones. Usado para hacer evaluaciones de impacto de los cambios propuestos. Necesaria una estructura para poder establecer vínculos entre los IC para que haya una trazabilidad.

Versión → estado de un ítem de configuración en un determinado momento. Se define como la forma particular de un artefacto en un instante o contexto dado.

Línea Base → No necesariamente se marca sobre toda la configuración de software, sino que ciertos IC la forman. Puede contener en un determinado momento, 1 solo ítem. Conjunto de IC estables, que pueden ser tomados de referencia, que han pasado por distintas aprobaciones y los marca para identificar que tales IC forman parte de la Línea Base. La línea base debe deber tener una identificación univoca. Su modificación es restringida, se necesita acceso. Deben pasar por un proceso de control de cambios.

¿Para qué marco líneas base en un repositorio? Para saber qué testear, qué desplegar. Punto de referencia a un momento de tiempo determinado y saber cómo fue evolucionando. Ultima situación aprobada que tenemos para un producto.

Tipos de líneas base

- De especificación
- De productos que han pasado por un control de calidad definido previamente.

Rama → sirven para bifurcar el desarrollo. Permiten la experimentación. Pueden ser descartadas o integradas.

Actividades Fundamentales de la → Administración de Configuración de Software.

- Identificación de ítems. Se define la estructura del repositorio, se identifican unívocamente los ítems y se asignan a la estructura del repositorio. Especificamos las bases. Acuerdos y lineamientos de cómo nombrar los ítems.

- Auditorias de configuración. Son controles autorizados a realizar por el equipo de desarrollo en un momento determinado, donde un auditor externo al equipo (independiente y objetivo) analiza las líneas base, las cuales permiten "congelar" y analizar en un momento determinado cuál es el estado del software, y si se están cumpliendo todas las pautas que plantea esta disciplina de SCM. En metodologías ágiles, esta es la única actividad de la disciplina SCM que no se soporta por la filosofía.
  Es objetiva cuando hay independencia de criterio, por lo que el auditor no debe tener ningún condicionamiento respecto de lo auditado.
  Auditor es alguien externo al equipo.
  Se hace primero → Auditoria física: vela por la integridad del repositorio.
  Se hace después de la física → Auditoria funcional: se fija si los IC son correctos.
  Necesitan un plan para guiarse.

- Informes de estado: Provee un mecanismo para mantener un registro de cómo evoluciona el sistema, y dónde está ubicado el software, comparado con lo que está publicado en la línea base.

  Esto permite mantener a todo el equipo informado sobre la última línea base, y que se trabaje en base a su última versión, y no sobre una versión obsoleta.

  Estos reportes incluyen todos los cambios que han sido realizados a las líneas base durante el ciclo de vida del software. Normalmente esto consta de grandes unidades de datos, por lo que se utilizan herramientas automatizadas por una computadora.

  El objetivo principal es asegurarse que la información de los cambios llegue a todos los involucrados.
  Tomar decisiones, corroborar la integridad.
  Dar visibilidad para → tomar decisiones.

- Control de cambios. Asociado a las líneas base. Tiene su origen en un requerimiento de cambio a uno o varios ítems de configuración que se encuentran en una línea base. Es un procedimiento formal que involucra diferentes actores y un evaluación del impacto del cambio. Una vez definida la línea base, no es posible cambiarla sin antes pasar por un proceso formal de control de cambios. Es decir que el control se hace sobre los ítems de configuración que pertenecen a la línea base, ya que todos los trabajadores del software tienen a dichos ítems como referencia

- **Unidad 1 completa (incluye paper No silver bullet)**
  - Introducción a la Ingeniería del Software. ¿Qué es?
  - Estado Actual y Antecedentes. La Crisis del Software.
  - Disciplinas que conforman la Ingeniería de Software.
  - Ejemplos de grandes proyectos de software fallidos y exitosos.
  - Ciclos de vida (Modelos de Proceso) y su influencia en la Administración de Proyectos de Software.
  - Procesos de Desarrollo Empíricos vs. Definidos.
  - Ciclos de vida (Modelos de Proceso) y Procesos de Desarrollo de Software
  - Ventajas y desventajas de c/u de los ciclos de vida. Criterios para elección de ciclos de vida en función de las necesidades del proyecto y las características del producto.
  - Componentes de un Proyecto de Sistemas de Información.
  - Vinculo proceso-proyecto-producto en la gestión de un proyecto de desarrollo de software.
- **Unidad Nro. 2. Gestión Lean Ágil de Productos de Software**
  - Gestión de Producto
  - Requerimientos Ágiles - User Stories
  - Estimaciones
- **Unidad Nro. 3: Gestión de Software como producto**
  - SCM (Gestión de Configuración de Software)

Definición de Ingeniería de software e introducción.

Según la IEEE.
La ingeniería de software es: La aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento de software; es decir, la aplicación de la ingeniería al software. La ingeniería de software incluye un proceso, métodos y herramientas para administrar y hacer ingeniería con el software.

Fritz Bauer la definió como el establecimiento y uso de principios de ingeniería estándar. Según él, ayuda a obtener, económicamente, un software que es confiable y funciona de manera eficiente en las máquinas reales.

Boehm define la Ingeniería de Software como la aplicación práctica del conocimiento científico al diseño creativo y la construcción de programas de computadora. También incluye la documentación asociada necesaria para desarrollarlos, operarlos y mantenerlos.

Definición de Software.

No hay que caer en la asociación de Software con código, que es una visión acotada.

“Software es conocimiento, información empaquetada a diferentes niveles de abstracción”

Conocimiento empaquetado a distintos niveles de abstracción. Los entregables, bases de datos, diseño, casos de uso, User Stories, manuales de configuración. Cada artefacto que sale de una tarea que se realizó en el contexto de un proyecto, es software.

Saber programar nada más, no alcanza para hacer software.

Software es cualquier producto de trabajo que sale de cualquier actividad dentro del ciclo de vida del proyecto y después del ciclo de vida del producto.

Problema del software:

- Fácil de modificar. Los clientes se aprovechan para pedir cambios
- Dificultad para gestionar las versiones del software.
- Es la problemática que aborda la gestión de configuración de SW.
- Cambios en el negocio afectan al SW.

Definición de Ingeniería de software e introducción.

Según la IEEE. La ingeniería de software es: La aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento de software; es decir, la aplicación de la ingeniería al software. La ingeniería de software incluye un proceso, métodos y herramientas para administrar y hacer ingeniería con el software.

Ingeniería del Software disciplina técnica que tiene por objetivo la construcción de sistemas de software en presencia de restricciones de tiempo, dinero, recursos y demás.

La práctica de la Ingeniería del Software representa la capacidad de entregar programas ejecutables, correctamente documentos, que cumplen con las exigencias explícitas e implícitas de los clientes, que se integran correctamente en su ambiente de producción y que sean útiles, generando valor de negocio a los clientes.

Ingeniería de software es un concentrador de muchas disciplinas. Tipos de disciplinas según la función

Técnicas: Asociadas al PRODUCTO. Lo que se le entrega el cliente es producto. Creación del producto con calidad.

- Requerimientos
- Análisis
- Diseño
- Implementación/Programación
- Prueba
- Despliegue

Gestión: asociadas al PROYECTO.

- Planificación
- Seguimiento y supervisión

Soporte: asociados al PRODUCTO y al PROYECTO. Protectoras, que intentan atender y prevenir para que la calidad del producto sea la que comprometimos con el cliente. Cumple las expectativas del cliente (de tiempo, funcionalidad). Ocurre transversalmente a lo largo de todo el proyecto y más allá inclusive porque trasciende al proyecto para darle soporte al producto en todo su ciclo de vida.

- SCM
- Aseguramiento de calidad – PPQA Aseguramiento de calidad de proceso y producto.
- Métricas

Definición de proceso de software.

En definitiva, es la transformación de ideas, requerimientos y tiempo, en productos y servicios de software, realizando un conjunto actividades estructuradas de trabajo en las que utilizamos recursos como materiales, energía, equipamiento, procedimientos e intervienen personas.

Las actividades varían dependiendo la organización y el tipo de sistema que debe desarrollarse. Un proceso de software debe ser explícitamente modelado si va a ser administrado.

Definición de proceso de Software (SW-CMM). Un conjunto de actividades, métodos, practicas y transformaciones que la gente usa para desarrollar o mantener software y sus productos asociados.

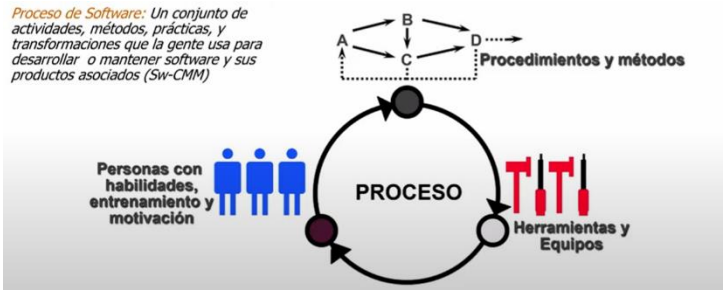


Foto = enfoque de mesa de 3 patas. Todo es importante para lograr el equilibrio que hace falta para que un proceso realmente puesto en una organización funcional obtenga un producto de software de calidad

El proceso es como una definición de diccionario porque es donde está escrito en términos teóricos que es lo que se debería hacer para hacer software. Mas teórico. Al momento de iniciar el proyecto, yo tendré que adaptarlo (proceso definido) o terminar de armarlo en el caso que sea empírico con las personas que están involucradas.

El proceso se define en términos de roles. Para cada proyecto, las personas asumen uno o mas roles. El proceso se instancia en cada proyecto. Proceso es teórico y lo instancio en el proyecto.

En ese momento, se elige el ciclo de vida a utilizar. El ciclo de vida es el ciclo que va a tener el proyecto que será el encargado de gestionar a la gente y los recursos para poder obtener un producto de software que sea de calidad y satisfaga las necesidades de los usuarios.

Definición de procesos definidos y empíricos.

Procesos Definidos. Procesos que vienen del taylorismo y de las líneas de producción de Ford. Plantean la necesidad de tener definido ampliamente paso a paso qué vamos a hacer en cada momento. Asume que podemos repetir el mismo proceso una y otra vez indefinidamente, y obtener los mismos resultados.

La administración y control provienen de la predictibilidad del proceso definido. Definidas las tareas a realizar por cada persona, cuando la realizará, que entradas y salidas tiene, qué artefactos se generan y en qué orden.

Tienen la intención de ser REPETITIVOS. Se busca lograr la repetitividad para lograr la visibilidad y saber qué esperar a cada momento de tiempo y para poder predecir situaciones posibles. Las personas que implementan estos procesos quieren tener la ilusión de CONTROL. Todo el mundo sabe lo que tiene que hacer.

Intentan ser procesos COMPLETOS que describen la mayor cantidad de cosas posibles.

Ejemplo: PUD (Proceso Unificado de Desarrollo). Explica paso a paso la mayor cantidad de opciones posibles que deberíamos tener en cuenta para hacer un producto de software. Se adapta a cada situación en

particular. Nivel de definición de las actividades es anticipado. + formales, con + definiciones sobre las cosas, + explícitos. RUD

Procesos Empíricos. Tienen un fuerte arraigo en la experiencia. La experiencia no siempre es fácil de conseguir debido a que la misma es basarse en algo anterior para tomar una decisión en la actualidad. La experiencia no se puede comprar, ni es intrínseca a las personas, la experiencia se tiene que generar.

Surgen en contraposición de los procesos definidos.

Basado en ciclos de entrega cortos para poder generar retroalimentación que sirva como experiencia para evolucionar y seguir avanzando.

Son procesos que no están COMPLETOS. Sino mas bien se basan en la creación de guías, lineamientos, heurísticas, prácticas que cubren determinados aspectos del proceso. Hacen foco en la gestión o en la ingeniería, pero ningún proceso empírico cubre la TOTALIDAD de las cosas que hay que hacer. Esta característica es la que permite que los equipos al iniciar el trabajo decidan (basado en la experiencia) qué es lo que quiere hacer, cómo y cuándo.

Patrón de conocimiento en procesos empíricos. Asumen una hipótesis de que algo funciona de X forma, actúan/prueban, obtienen retroalimentación, revisan y si algo no salió bien, adaptan. Hay retroalimentación. Y el ciclo vuelve al inicio. De esta forma ganan experiencia.

El empirismo sostiene que la experiencia es aplicable al mismo equipo, y no se puede extrapolar a otros equipos distintos con contextos distintos. CONTRARIO A REPETITIVO.

No son procesos completos, por eso no se les suele llamar procesos. Mas bien son lineamientos, buenas practicas de alguna parte de un proceso. La idea es que el equipo las use y cierre y cree su propio proceso para el proyecto.

SCRUM NO ES UN PROCESO. SCRUM es un framework de trabajo para la gestión de productos y proyectos.

Ejemplo: movimiento Lean, movimiento Ágil.

Ciclos de Vida

No es lo mismo ciclo de vida que proceso.

Consiste en una serie de pasos a través de los cuales el producto o proyecto progresa.

Uno elige para un determinado proyecto un ciclo de vida.

Los productos también tienen su ciclo de vida. Son distintos de los de los de PROYECTOS, son mas largos.

Un ciclo de vida de un producto inicia con la idea de crear un producto de software y termina cuando a ese producto lo retiro del mercado. Productos que tienen años de ciclo de vida. Puede haber n ciclos de vida de proyectos. Cada proyecto genera una versión que suele conocerse como un reléase o entregable del producto que tiene ciertas características y que es el que se pone en producción. Mientras se use un determinado producto de SW va a necesitar cambios.

El ciclo de vida para el desarrollo de un producto, que es el ciclo de vida del proyecto lo que hace es establecer cómo afrontar las actividades que plantea el proceso, en qué orden, cuanto de cada actividad vas a hacer.

Un ciclo de vida de un proyecto de software es una representación de un proceso. Grafica una descripción del proceso desde una perspectiva particular.

Los modelos especifican:

- Fases de proceso
  - Ejemplo: requerimientos, especificación, diseño, etc.
- Orden en el cual se llevan a cabo.

Clasificación de los ciclos de vida. Hay 3 tipos básicos de ciclos de vida para un proyecto de desarrollo de software.

- Secuencial
  - Ciclo de vida en cascada. Hacer el 100% de cada etapa antes de pasar a la siguiente.
- Iterativo e incremental
  - Ciclo de vida iterativo e incremental. No se hace el 100% de las etapas, sino que se va haciendo un poco de todo y obtiene una versión o iteración.
- Recursivo: fracaso. Paso entre el secuencial al iterativo. Muchas vueltas, pero sin genera versiones intermedias del producto. Generas una versión que se pueda probar recién al final. No es el mismo resultado como cuando poner en producción el producto y el cliente puede ver las funcionalidades andar.
  - Ciclo de vida en espiral.

Para cerrar el vínculo entre procesos y ciclos de vida, cuando trabajamos con procesos definidos podemos elegir cualquier tipo de ciclo de vida de cualquier categoría.

Procesos empíricos no se pueden combinar con cualquier ciclo de vida. Solo con los iterativos e incrementales. No se puede implementar un proceso empírico con ciclo de vida de cascada porque el empirismo con una retroalimentación al final no es viable.

VINCULO ENTRE EL CICLO DE VIDA DEL PRODUCTO Y EL PROYECTO. Necesidad de cambios es el disparador en ambos.

Relación entre PROCESOS DE DESARROLLO y CICLO DE VIDA. Un ciclo de vida de software es un representación de un proceso.

PARCIALES

Alcance de Proyecto. El Alcance del Proyecto es todo el trabajo y sólo el trabajo que debe hacerse para entregar el producto o servicio con todas las características y funciones especificadas.

Generación de reportes de estado. Asegurarse que la información de los cambios llega a todos los Involucrados

Solicitud de cambio. Debe ser analizada para determinar su impacto. Requiere tratamiento por parte de la CCB

El comité de control de cambios. Los representantes de los roles que aportan diferentes visiones sobre el cambio y su naturaleza.

Software. Colección de programas necesarios para convertir a una computadora (de propósito general) en una máquina de propósito especial diseñada para una aplicación de un dominio particular, incluyendo documentación producto del desarrollo de un sistema

Cronograma → necesita las tareas y sus estimaciones, se realiza luego de determinar el alcance, es parte del planeamiento.

Product Backlog. User stories, spikes, épicas, defectos.

Plan de administración de configuración.

- Debe contener una sección que identifique los ítems de configuración a ser administrados
- Es requerido para realizar una auditoria
- Es un ítem de configuración.

Linea Base. Especificación y/o producto que ha sido revisado formalmente y se está de acuerdo con los resultados de la revisión; y a partir de allí sirve como base para el desarrollo ulterior, solo puede cambiarse con un procedimiento formal.

Ciclo de Vida en Cascada. La fase de prueba comience solo si la de codificación ha finalizado. Los requerimientos se conozcan en etapas tempranas del proyecto. Cada fase tenga una salida tangible