Análisis y Metodologías

Para el desarrollo de esta materia haremos énfasis en parte de las metodologías:

- UP.
- Scrum y Kanban (combinadas).
- UML (como lenguaje de documentación y modelado).

El enfoque estará en la forma en la que cada una :

- Captura de requerimientos y requisitos.
- En el uso de artefactos.
- En la mecánica documental y de seguimiento.

Vamos a ver metodologías tradicionales y ágiles

trabajo práctico
lectura del material teórico
1 solo parcial teórico (noviembre)

Sistema y análisis

¿Que era un Sistema Informático ?

"Permite ingresar, almacenar y procesar datos, transformándolos y produciendo información de utilidad. Es el conjunto de partes interrelacionadas (HW, SW, personal, etc)..."

¿Qué es Analizar ?

"Examinar detalladamente, separando o considerando por separado sus partes, para conocer sus características o cualidades, o su estado, y extraer conclusiones".

Analizar un sistema implica detectar (muchas veces siguiendo una metodología):

- Los requerimientos funcionales,
- Los requisitos no funcionales,
- Los atributos de calidad necesarios y,
- Las reglas de negocio.

Metodología

¿ De qué se trata una Metodología ?

Metodología significa "un conjunto de métodos que siguen una determinada lógica".

Una metodología de desarrollo de software se refiere a un framework (entorno o marco de trabajo) usado para estructurar, planear y controlar el proceso de desarrollo en sistemas de información.

Todo sistema fue construido, o es mantenido siguiendo una metodología de acuerdo a un Ciclo de Vida

Las metodologías obedecen, se rigen o responden también a un tipo de ciclo de vida al que responde el proceso de construcción del sistema o por el cual se mantiene un determinado sistema.

Una filosofía de desarrollo de programas de computación con el enfoque del proceso de desarrollo de software. Parte de esa metodología es el análisis de los requerimientos y requisitos.

*Requisitos funcionales y no funcionales:
El análisis es vital y es consecuencia del relevamiento, después del relevamiento va el análisis .*

Ciclo de vida

¿Y qué es un ciclo de vida entonces ?

Cada sistema suele responder a algún tipo de ciclo de vida. Cada tipo suele tener características, pueden responder a...

- El Estilo de la Organización.
- El momento en el que nos encontramos.
- El Condicionamiento sobre el Proyecto en un determinado momento.

¿ y por qué lo anterior es importante en el análisis ?

- porque influye en el análisis de los requerimientos y requisitos de un sistema.
- Porque la técnica de detección de requerimientos y requisitos viene influenciada por la metodología, ésta por el ciclo de vida, y esta última por la elección de la organización.
- Como profesionales debemos poder entender contextos

Proyecto informático de un sistema

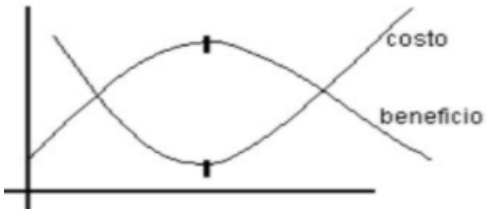
El análisis en un proyecto informático, implica identificar a actores, su punto de vista, los requerimientos, requisitos, necesidades, y puntos de vista: **Actores => entender su punto de vista ->** Actitud; Aptitud; Poder de observación y acción

Información

- **Reflejo de los hechos**: Elementos que reflejan los hechos de una organización. Se representa con Reportes, Tableros de Comando, Balances, Información Contable, etc.
- **Interpretación Precisa:** hay que saber interpretarla para saber que sucede en la organización.
- **Exactitud y Precisión**: debe ser exacta, oportuna y consensuada, esta mejorará las decisiones y acciones a ejecutar en los proyectos.

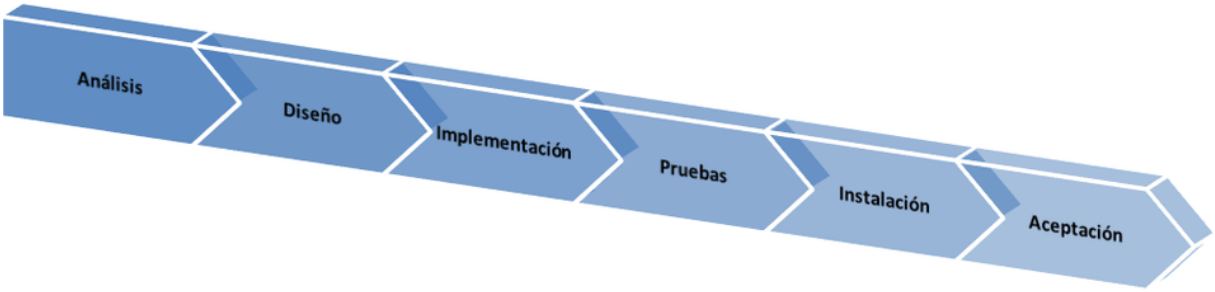
Costo vs Beneficio

"Costo para generar la información / Beneficio que generan las decisiones tomadas".
El mínimo costo es posterior al momento de toma de decisión. Si la decisión se toma después de un tiempo, el beneficio es menor. ----> relación costo beneficio --->



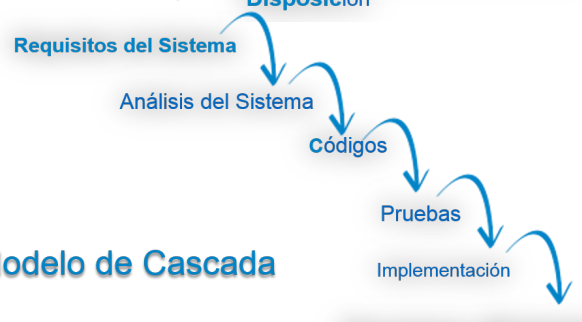
Ciclos de vida

- Lineal
  - El ciclo de vida en cascada es uno de los más antiguos, simples y sencillos.
  - Está pensado para proyectos informáticos cortos, concretos y por sobre todo finitos.
  - Su estructura va a coincidir muy fácilmente con los procesos usuales de un proyecto.
  - No está pensado para "evolutivos". Cuando el software nace, se termina el ciclo.

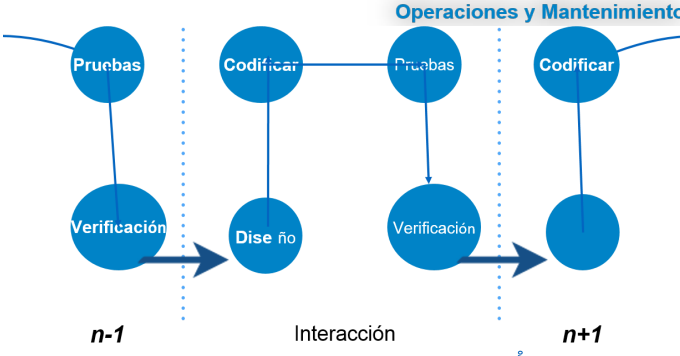


- SDLC
  - El ciclo de vida del desarrollo Software (SDLC en sus siglas), secuencia estructurada y definida de las etapas en Ingeniería de software para desarrollar el producto software deseado.
  - Es consensuado y adoptado por entidades financieras, o empresas multinacionales (quizás por su trazabilidad).
  - Es un estilo consensuado, reúne, preserva y reestructura elementos (aún vigentes) de estilos más antiguos.
  - Suele ser, y es cuestionado por eso, algo burocrático pero presenta buena trazabilidad y buen seguimiento para un proyectos de software.

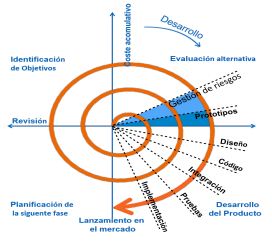
- Cascada
  - Sigue un modelo en que las fases del SDLC funcionarán una detrás de la otra de forma lineal.
  - Lo que significa que solamente cuando la primera fase se termina se puede empezar con la segunda, y así progresivamente.
  - Este modelo concibe una período de operación y mantenimiento, posterior al nacimiento en el que se entiende que pueden haber evolutivos o mejoras.



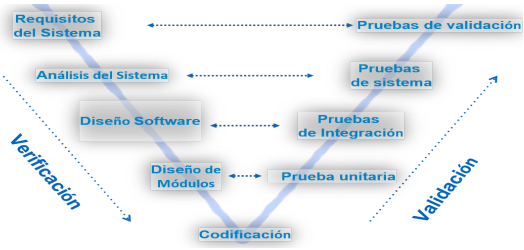
- Repetitivo
  - Este modelo guía el proceso de desarrollo de software en repeticiones.
  - Proyecta el proceso de desarrollo de forma cíclica repitiendo cada paso después
  - de cada ciclo en el proceso de SDLC.
  - Está pensado directamente para tener ciclos evolutivos recurrentes de mejora (del software).



- Espiral
  - El modelo en espiral es una combinación de ambos modelos: el repetitivo (o iterativo incremental) y uno del modelo SDLC.
  - Se puede ver como si se combina un modelo de SDLC combinado con un proceso cíclico (modelo repetitivo).
  - Fue usado ampliamente para lanzamientos de productos al mercado.

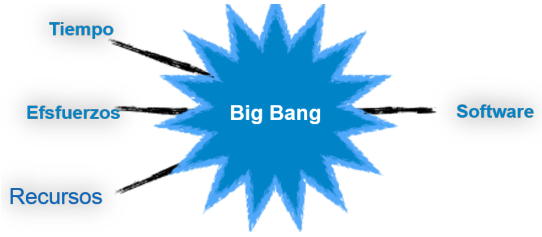


- V

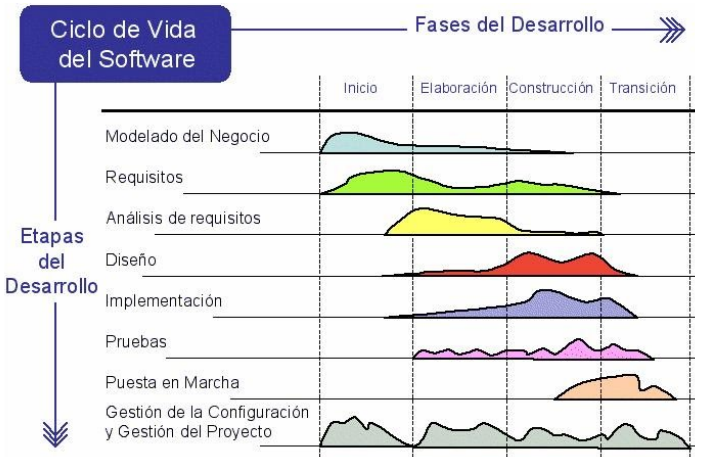


- El mayor inconveniente del modelo de cascada es que solo se pasa a la siguiente fase cuando se completa la anterior, por tanto no es posible volver atrás si se encuentra algún error en las etapas posteriores.
- El Modelo V aporta opciones de evaluación del software en cada etapa de manera inversa, de esta manera se puede monitorear el avance y acompañarlo con validaciones permanentes.

- Big bang
  - modelos más simples con los que podemos contar.
  - Requiere poca planificación, mucha programación (concurrente) y muchos fondos.
  - Simula el concepto de la teoría de creación del universo 'Big Bang'.
  - Tal como cuentan los científicos, después del big bang muchas galaxias, planetas y estrellas evolucionaron.
  - De la misma manera, si reunimos muchos fondos y programación, quizá podamos conseguir el mejor producto de software.

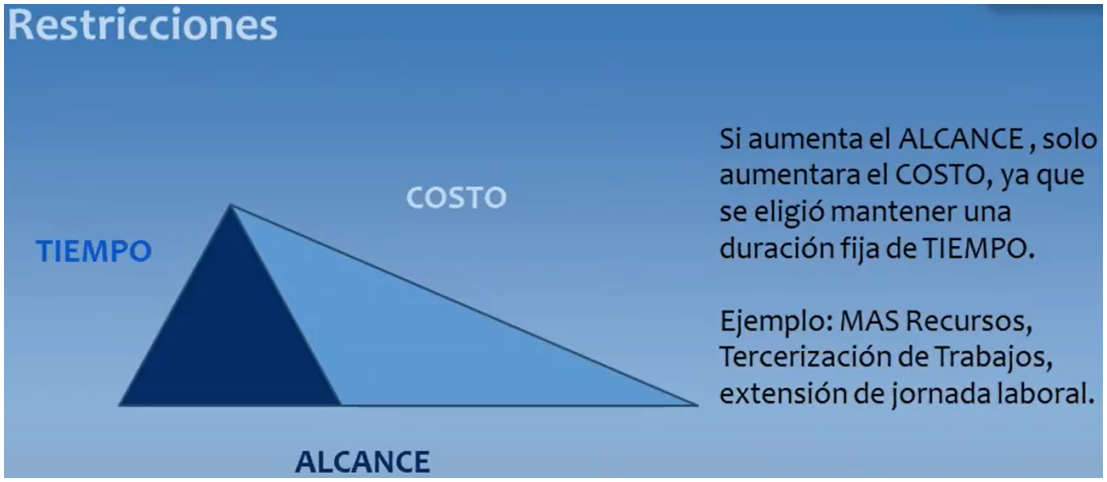


- Iterativo Incremental
  - Este ciclo de vida, usualmente llamado Proceso Unificado o UP (por sus siglas en Inglés), representa también una metodología que contribuye a estructurar un proyecto de software, proveyendo fases, tareas, entregables todo estructurado en interacciones que producen incrementos de funcionalidad.
  - Fue postulado por la firma Rational (hoy parte del grupo IBM) por lo que también es conocido por la sigla RUP. De este ciclo de vida se desprende además una metodología de desarrollo de software que lleva el mismo nombre.
  - El mejor aporte de este modelo, fue una determinación precisa de artefactos (con un buen ensamble con la UML).
  - Otro gran aporte de este modelo, fue la determinación de los ritmos o cadencias, en las cuales se determinaban “iteraciones” de corto plazo.
  - Más aportes fueron, la posibilidad de validar y permitirse corregir errores o defectos. También aportó elementos de medición de rendimiento y de predicción.



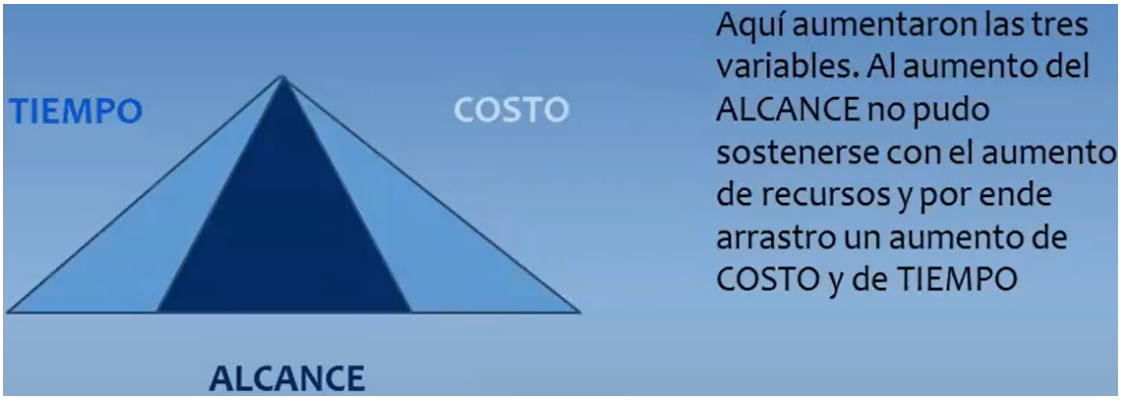
- Ágiles
  - En los últimos tiempos, producto del ejercicio y experiencias de los ciclos de vida anteriores, surgió un nuevo estilo comunicado a través del “Manifiesto Ágil”.
  - Del mismo se desprenden una serie de metodologías de desarrollo, más que ciclos de vida, que buscan producir más y mejor software en lugar de tanta documentación.
  - Actualmente son ampliamente aceptadas y muchas de ellas han desplazado o están desplazando a las anteriores.
  - En alguna medida se deben dar una serie de condiciones para que las mismas se puedan ejecutar a su máxima capacidad.
  - Algunas de ellas son la metodología Scrum, Kanaban, Proceso Unificado Ágil (AUP, por sus siglas en Inglés), eXtreme Programming, etc.
  - Aquí también, la frontera entre ciclo de vida y metodología no es clara (aunque no muy relevante), de todas formas también nos genera condicionamientos en la Dirección y Gestión de Proyectos y en el proceso de Análisis que se deben saber para poder mitigar.

Gestión Predictiva de Proyectos



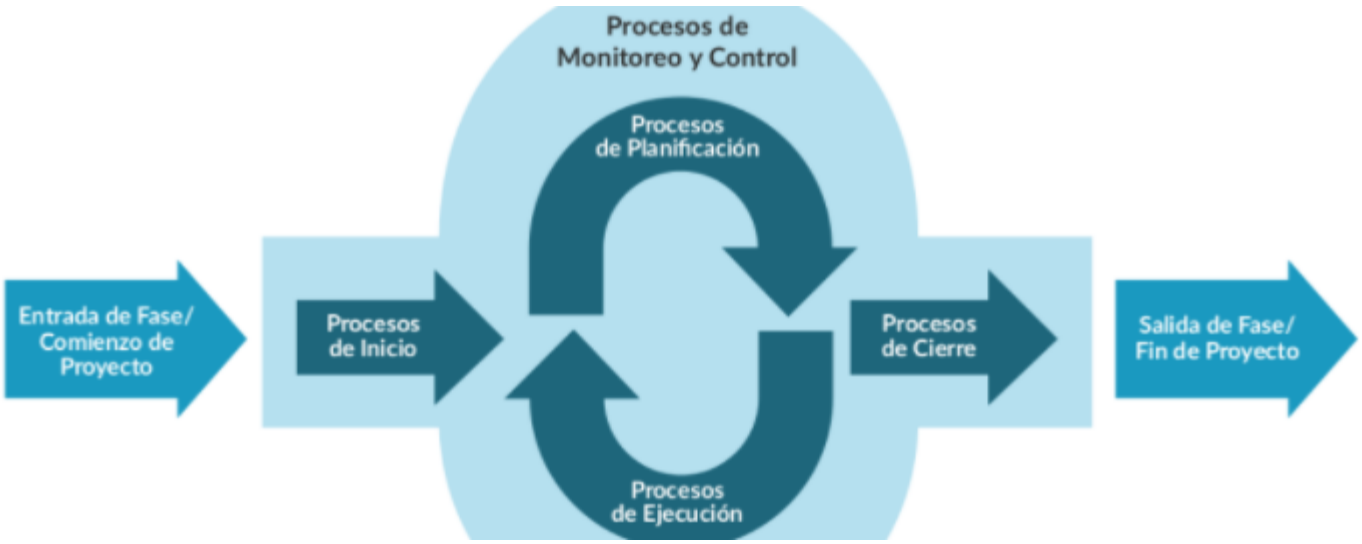
Esto se conoce como la triple restricción

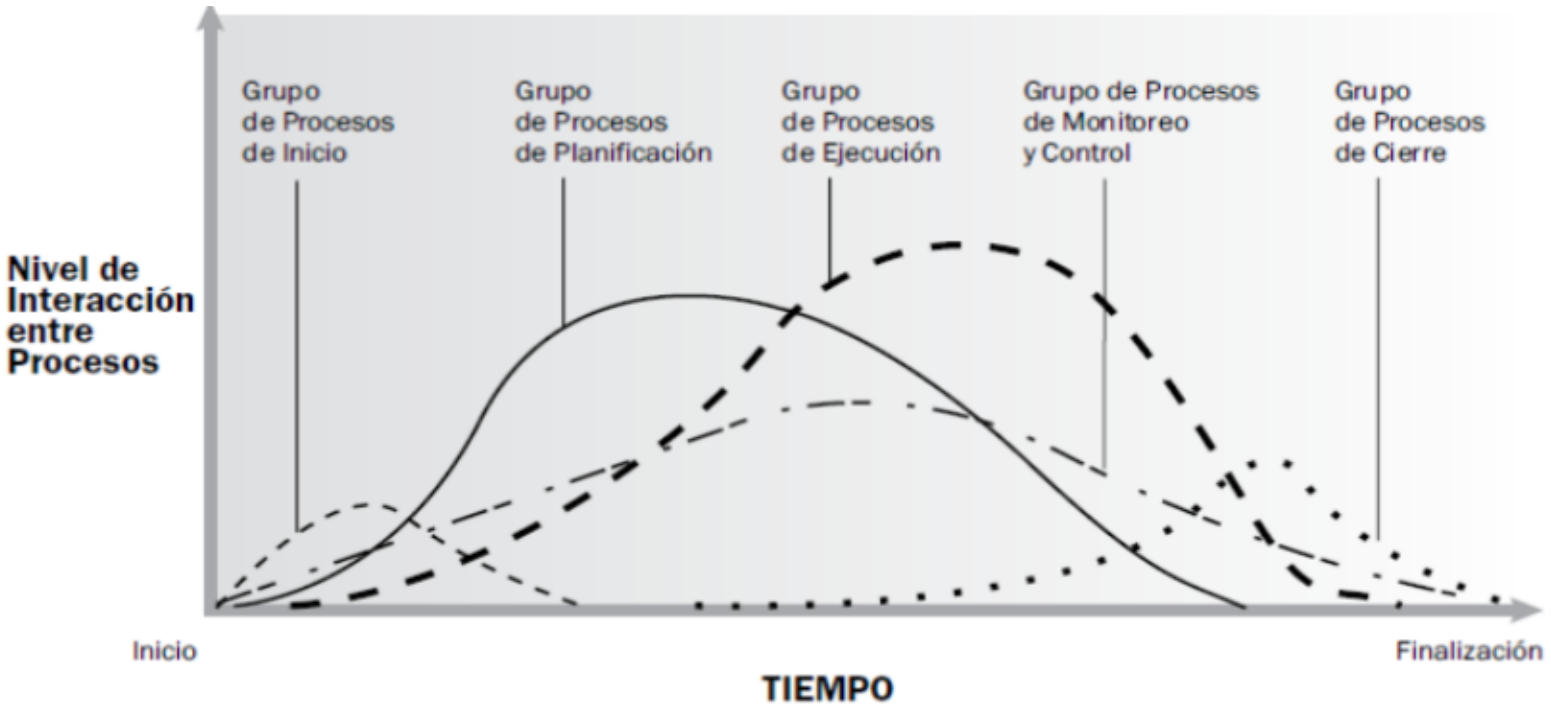
(Si aumenta el alcance aumenta el costo y el tiempo)



Variables que pueden jugar dentro de la restricción

Grupo de procesos





Áreas de conocimiento

• **Integración** 1. Acta de constitución del proyecto 2. enunciado del alcance preliminar 3. plan de gestión del proyecto 4. dirigir y gestionar la ejecución 5. Supervisar y controlar el trabajo 6. Control integrado de cambios 7. Cerrar el proyecto	• **Alcance** 1. Planificación del alcance 2. Definición del alcance 3. Crear EDT 4. Verificación del alcance 5. Control de cambios del alcance	• **Tiempo** 1. Definición de actividade 2. Secuenciación de actividades 3. Estimación de recursos 4. estimación de la duración de las actividades 5. Desarrollo del cronograma 6. Control del cronograma
• **Costos** 1. Estimación de costos 2. Asignación del presupuesto de costos 3. Contorno de los costos	• **Calidad** 1. Planificación de la calidad 2. Aseguramiento de la calidad 3. Control de calidad	• **RR.HH** 1. Planificación de los RRHH 2. Adquirir el equipo del proyecto 3. Desarrollar el equipo 4. Gestionar el equipo del proyecto
• **Comunicaciones** 1. Planificación de las comunicación 2. Distribución de ia información 3. Informes de rendimiento 4. Gestionar a los interesados	• **Riesgos** 1. Planifican gestión de riesgos 2. Identificación de riesgos 3. Análisis cualitativos de riesgo 4. análisis cuantitativos de riesgos 5. Plan de respuesta de riesgos 6. Supervisión y control de riesgos	• **Adquisiciones** 1. Planificación de las adquisiciones 2. Planificación de la contratación 3. SOlicitar respuestas de endores 4. Selección de proveedores 5. Administración del contrato 6. Cierre del contrato

Roles tradicionales

- Gerente de proyectos
  - Planificación de Tareas
  - Identificar necesidades de recursos
  - Mantener Clima Laboral
  - Asegurar cumplimiento de tiempos
  - Reportar a niveles de Gerenciamiento superiores
  - Aportar a mantener la Visión del cliente sobre las tareas a realizar
- Lider de equipo
  - Guía del Equipo
  - Mantener armonía y cordialidad
  - Asegurar la performance del equipo
  - Organizar y Supervisar el equipo de Trabajo
  - Asesorar en la definición e implementación de mejoras
  - Reportar periódicamente avances y riesgos
- Analista funcional o de negocio
  - Realizar el relevamiento de necesidades
  - Estimar esfuerzos y generar propuesta de solución
  - Prever cambios en base a su experiencia
  - Organizar los requerimientos
- Roles tradicionales - Desarrolladores y Testers
  - Construir la Solución
  - Colaborar en el Diseño Gráfico
  - Desarrollar Prototipos
  - Ejecutar testing unitario, integral , funcional y de performance
  - Trabajar en equipo con el resto de los colaboradores

Técnicas y herramientas

- Estimación
  - Por Analogía
  - Paramétrica
  - Por 3 Valores (Te=(T0+4*Tm+Tp)/6
  - Top-Down vs Botton -Up
- EDT (o WBS) : estructura de desglose de tareAS

Scrum y Kanban

Enfoques ágiles

- Los principales enfoques ágiles son XP (eXtreme Programming), SCRUM, Crystal Clear y DSDM (Dynamic Systems Development Method), entre otros
- Herramientas de modelado ágil: tarjetas CRC, lenguaje natural, castellano estructurado, etc.
- Existen procesos marco tradicionalmente formales que contempla la aplicación de técnicas de modelado ágil, como por ejemplo RUP y actualmente CMMi

Valores de manifiesto ágil:

- Valorar más a los individuos y su interacción que a los procesos y las herramientas
- Valorar más el software que funciona que la documentación exhaustiva
- Valorar más la colaboración con el cliente que la negociación contractual
- Valorar más la respuesta al cambio que el seguimiento de un plan

Principios del manifiesto ágil

1. Nuestra prioridad más alta es la de satisfacer al cliente a través de la entrega temprana y continua de software de valor
2. Son bienvenidos los requisitos cambiantes, incluso si llegan tarde al desarrollo. Los procesos ágiles se subordinan al cambio como ventaja competitiva para el cliente
3. Entregar con frecuencia software que funcione, en periodos de un par de semanas hasta un par de meses, con preferencia de los periodos más cortos
4. Las personas del negocio y los desarrolladores deben trabajar juntos de forma cotidiana a lo largo del proyecto
5. Llevar a cabo proyectos en torno a individuos motivados, dándoles la oportunidad y el respaldo que necesitan y recordándoles confianza para que realicen la tarea
6. La forma más eficiente y efectiva de comunicar información de ida y vuelta dentro de un equipo de desarrollo es mediante la conversación cara a cara
7. El software que funciona es la principal medida del avance del proyecto
8. Los procesos ágiles promueven el desarrollo sostenido. Los interesados, desarrolladores y usuarios deben mantener un ritmo constante de forma indefinida
9. La atención continua a la excelencia técnica y al buen diseño promueve la agilidad
10. La simplicidad –el arte de maximizar la cantidad de trabajo que no se hace- es esencial
11. Las mejores arquitecturas, requisitos y diseños surgen de equipos autoorganizados
12. El equipo reflexiona cada cierto tiempo, en intervalos regulares, sobre la forma de ser más efectivo y ajusta su conducta en consecuencia

-Scrum se compone de: el dueño del producto; el scrum master y el equipo de desarrollo (recursos humanos)

-El product owner es el dueño del producto y es el que trae los requerimientos

-El scrum master es quien conoce la metodología y hace cumplir cada detalle

-El Backlog de producto está formado por historias de usuario (las historias de usuario son definiciones)

Acerca de Scrum

Marco de trabajo dentro de las metodologías ágiles, que administra el proceso de desarrollo de software desde una perspectiva empírica basada en la teoría del control de procesos, introduce conceptos de flexibilidad, adaptabilidad y productividad.

-los procesos de desarrollo de software no son definidos, no pueden ser ni repetibles ni predecibles: se trata de un proceso caótico de adaptación del equipo al caos para lograr un Objetivo auto organizándose y tomando decisiones con total libertad, logrando una cohesión interna y una productividad cada vez mayor

-Ocupa menos tiempo en planificación, definición de tareas y producción de reportes y más tiempo en la comprensión del problema y la provisión de soluciones empíricas

Historia

-Es una visión contrapuesta al Manufacturing Maturity Model del que se creó el Capability Maturity Model

Beneficios

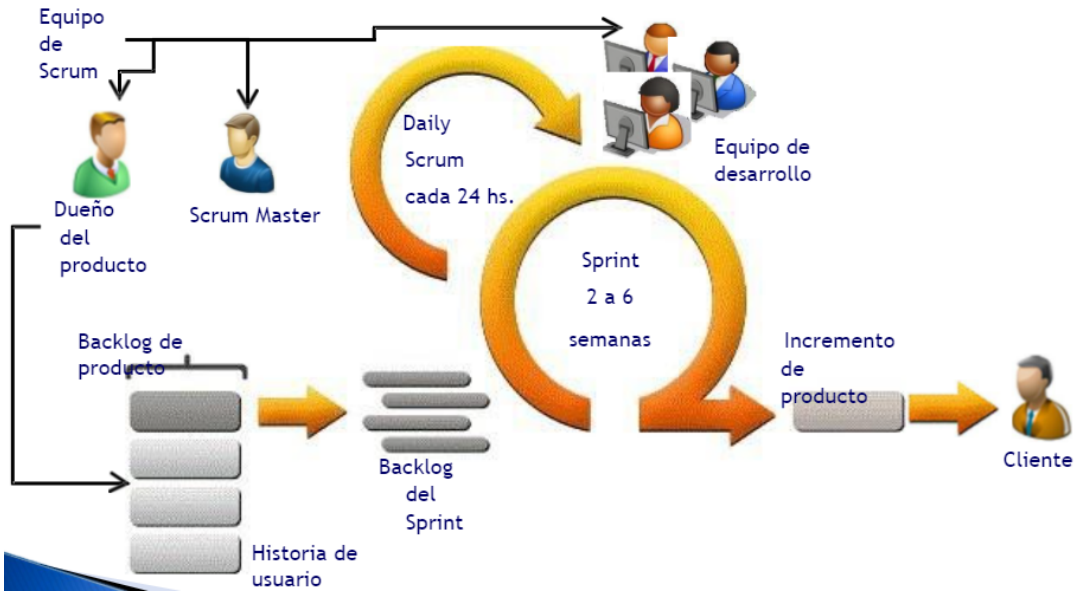
-Permite coordinar los recursos humanos de modo de que trabajen juntos en forma efectiva y puedan lograr desarrollos complejos

-Permite lograr incrementos exponenciales en la productividad

-Con Scrum se espera estar entregando software funcionando correctamente ya en el primer mes de desarrollo

-Posibilita el desarrollo de software en entornos complejos y cambiantes

Esquema de scrum



El product owner

-Es el dueño del producto y responsable del mismo.
-Siempre es una persona y NO un comité... (Pueden existir comités para asesorar al dueño del producto)
-el dueño del producto representa los stakeholders
-Es quien se encarga de administrar los requisitos que se entregan, establece prioridades y decide contenido de los "releases"
-Convierte los "temas" en requisitos del proyecto o sea el "Backlog del producto" y asegura que el equipo de desarrollo comprenda esos elementos
-Responsabilidades: visión del producto, prioridades, características mínimas, fechas de releases, ROI, costo
-define plan de desarrollo del producto; está orientado al negocio; consigue los interesados, usuarios etc; proporciona retroalimentación y aceptación

-el product owner es quien trae las tareas y los criterios de aceptación de cada una al proyecto

Product Backlog

Lista evolutiva y priorizada que tiene los requisitos funcionales y no funcionales del proyecto, características y aspectos

-Es administrada por una única persona, el Dueño del Producto (Product owner)

-la agenda de desarrollo del proyecto se hace a partir de esta lista

-cada ítem del backlog debe tener su propia estimación de tamaño y su valor

-el backlog incluye “temas” (issues) se mantienen hasta que sean convertidos en trabajo

- Sólo pueden entrar para ser atendidos en cualquier momento los requisitos de soporte y mantenimiento del código preexistente

-Los requisitos del Backlog de más alta prioridad están más detallados que los de más baja prioridad

-Si un ítem del Backlog no está lo suficientemente claro, se lo debe transformar en un tema o se le debe reducir su prioridad

-El Backlog se puede dividir en partes, llamadas Release Backlog, correspondientes a los releases planificados

Historias de usuario

Descripción de los requisitos de software expresados en una o dos frases: “como (rol) necesito esta (característica) para obtener este (valor)” → se pueden conocer como “ítems” o “elementos”

-los ítems de mayor prioridad tienen más detalle que los de menor prioridad

-A veces se las llama simplemente ítems o elementos del backlog

-se pueden describir con: ID, nombre, importancia, estimación inicial, criterios de aceptación, observaciones

-importancia: no importa el valor, sólo que más alto es más importante

-Estimación inicial: se miden en “puntos de historia”, que son días-hombre ideales para implementarla y exige que los valores sean correctos (2 puntos de historia = 2 días de 1 persona) y sean relativos (una historia de 2 puntos se hace en la mitad del tiempo que una de 4 puntos)

-criterios de aceptación: describe cómo se deberá demostrar que lo solicitado funcione como se espera

(cada historia de usuario no debe estar incluida en otra)

Campos adicionales:

*Categoría: agrupa las historias por su mayor relación (seguridad, front end, arquitectura, etc.)

*Componentes: indicando como check boxes los componentes empleados

*Solicitante: quién solicitó la historia

*ID del bug tracking: código otorgado por la herramienta de seguimiento de defectos para seguir el estado de los defectos reportados para la historia

Modelo INVEST

- Independiente: una historia debe ser lo más independiente posible de otras

-Negociable: la historia es sólo una descripción corta y sus detalles se obtienen durante la etapa de conversación con el cliente

-Valiosa: cada historia debe tener valor para el cliente; seguramente lo serán más si el propio cliente las escribe

-Estimable: se deben poder estimar para permitir priorizar y planificar

-Pequeña (Small): una buena historia debe ser pequeña en esfuerzo, de no más de 2 a 3 personas/semana de trabajo

-Verificable (Testable): si no se la puede probar, nunca vamos a saber si se la terminó

-Las historias no funcionales (técnicas) carecen de valor para el Dueño del Producto conviene: describirlas como funcionalidades con valor para el negocio; incluirlas como tareas dentro de otras historias; dejarlas como no funcionales y calcular la velocidad de desarrollo considerando un tiempo para implementarlas dentro de cada Sprint

Sesion de especificación

sesiones donde se busca: clarificar el contenido de cada historia; identificar las posibles inconsistencias; revisar los casos extremos de cada historia; discutir detalles que permitan su implementación/. se realizan al menos a un Sprint de distancia de donde se debe desarrollar el item

//Deben participar el Dueño de Producto, el Scrum Master y todo el Equipo de Desarrollo, incluyendo los responsables de las pruebas si los hubiera

-Los miembros del **Equipo de Desarrollo** hacen consultas hasta agotar todas sus dudas

-Los criterios de aceptación son consistentes y comprendidos por todos

-Se debe discutir el "qué" y no el "cómo"

-definir historias realizables y razonables, que crecerán con el tiempo

Criterios de aceptación

- Son las condiciones que permiten determinar que una historia está "terminada"
- Se escriben de manera colaborativa al cabo de las sesiones de especificación
- Deben ser sencillos, fáciles de comprender, específicos, relevantes, observables casi inmediatamente y, obvio, verificables
- Se deben enfocar en el "qué" y no en el "cómo"
- Se pueden obtener transformando los ejemplos, los diferentes caminos alternativos y las reglas de negocio

Story map #1

- Se agrupan las historias por algún criterio de uniformidad: épica, funcionalidad, tema, módulo del sistema, etc. y se compone la importancia
- Se dividen las historias por Sprint

Durante los **Sprints** quedan identificadas las historias hechas, en marcha o por hacer, los objetivos de cada Sprint, y todo lo que se desee representar

Scrum Master

- Es el principal responsable del éxito de **Scrum**
- Se ocupa del cumplimiento del proceso, de proteger al **Equipo de Desarrollo** de los pedidos fuera del **Backlog**, de hacer que los clientes, usuarios y stakeholders en general se atengan a las reglas del proceso, etc.
- Se encarga de controlar que el **Equipo** no desarrolle nada fuera de lo acordado dentro del **Sprint** en curso
- Representa a la otra parte, a la gerencia o al **Equipo**, frente a cada uno de éstos
- Sin lugar a dudas este rol debe ser ocupado por quien tenga el perfil adecuado; no todos pueden llevarlo a cabo
- Mientras el **Equipo** trabaja, el **Scrum Master** y el **Dueño del Producto** continúan agregando ítems al **Backlog** de producto
- Ambos son quienes establecen la visión del proyecto

Sprint

- Es cada una de las iteraciones del proceso de desarrollo de software, que es un proceso iterativo e incremental
- Posee una duración fija de un mes o menos, que se establece al inicio del proyecto
- La duración permitir la inclusión de requisitos de alta prioridad que pudieran surgir mientras se lleva a cabo un **Sprint**
- Cada **Sprint** se inicia con una **Reunión de Planificación**, que establecerá el trabajo a realizarse en el tiempo que dura el **Sprint**, teniendo en cuenta la velocidad de desarrollo
- Los ítems del **Backlog** que se incluyen en el **Sprint** conforman el **Backlog del Sprint**
- Cada **Sprint** debe tener un **objetivo** claro (**Sprint Goal**).
- Al término de un **Sprint** se hace una **Reunión de Revisión** para evaluar lo realizado y el cumplimiento del **Objetivo del Sprint.**
- Si durante el **Sprint** se detecta que no se puede cumplir con el **Objetivo**, el **Scrum Master** se debe reunir de inmediato con el Dueño **del Producto** y el **Equipo** para ver qué ítem del **Backlog del Sprint** se puede quitar o disminuir su alcance o profundidad.
- Cuando el **Equipo** descubre que no podrá cumplir con sus compromisos en el **Sprint**, debe solicitar una **Terminación Anormal del Sprint** y se debe convocar a una reunión de planificación de un nuevo **Sprint.**
- En todo proyecto existen cuatro restricciones: tiempo, costo, calidad y funcionalidad a entregar; un **Sprint** prácticamente fija las tres primeras
- El tiempo, son 30 días; el costo, el del salario del equipo, el equipamiento y los consultores y herramientas que se pudieran precisar; la calidad se debe establecer antes de iniciar el **Sprint**; la funcionalidad se puede manejar siempre y cuando se cumple con el **Objetivo del Sprint**
- **Sprints** más cortos: permiten retroalimentación más rápida, más entregas de producto, menos desvíos por trabajar en lo incorrecto.
- **Sprints** más largos: permiten que el **Equipo** se acomode, aprenda de sus errores y corrija el rumbo.

Para medir la velocidad de desarrollo:

- usar "el tiempo que se hizo ayer", que consiste en basarse en la historia del propio proyecto de los **Sprints** anteriores
- calcular el "factor de ajuste" o "factor de dedicación" como (velocidad de desarrollo / días-hombre disponibles) y en donde la velocidad de desarrollo son los puntos planificados al inicio del **Sprint** sólo de las historias terminadas en un 100%
- no tener en cuenta eventos especiales que pudieran haber afectado gravemente la velocidad

Release sprint

- al término de cada **sprint** se debe entregar un incremento de producto funcionando
- Cada cierta cantidad de **sprints** se completa una cantidad de funcionalidad que compone un release que sí puede ponerse en producción.
- Para poner en producción un release, normalmente se requieren actividades extra, por lo que en muchos casos se define un "**sprint de release**".
- Un **sprint de release** puede ser un síntoma de debilidad de **Scrum**, por lo que se debe revisar cómo incluir

estas actividades de modo de no necesitarlo.

- En este caso, el **Dueño de Producto** es quien decide cuándo el producto está lo suficiente maduro como para pasar a producción.

Quede en la página 60

comparaciones

Tema	Scrum	Kanban
Tiempos	son fijos	son fijos opcionalmente
Ítems	se agregan sólo al comienzo del Sprint	se agregan a medida que haya capacidad disponible
Estimación	es obligatoria	es opcional
WIP	limitado indirectamente por la velocidad de desarrollo	limitado explícitamente
División en tareas	es obligatoria	es opcional
Planificación	basada en la velocidad de desarrollo del Equipo	basada en el tiempo de entrega

---

13/09/2021

trabajo práctico en trello

1 crear cuenta en trello

2 crear dos tarjetas 1.1 bpm que haría el soft 1.2 crear tarjeta que describa el alcance

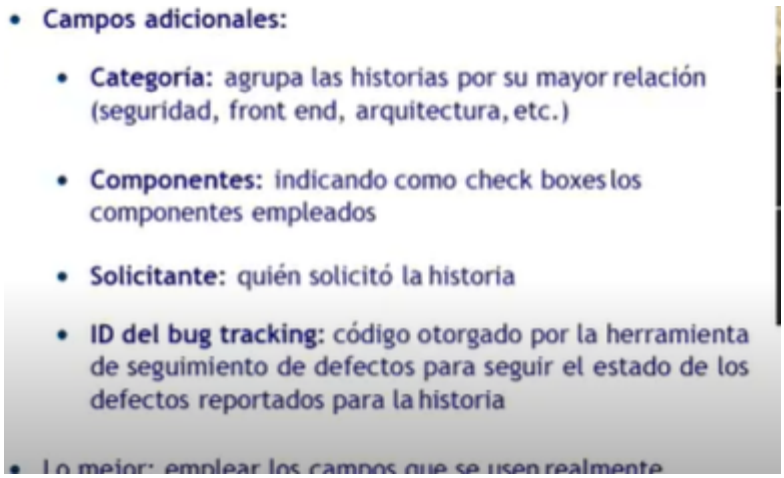
3 alcance

-

https://app.diagrams.net/ BPM

dejar las tareas más relevantes en primer lugar

(para las tarjeta de trello)



04/10/2021

https://drive.google.com/drive/folders/1UcJD7yItSNszG36103OMaIjuMrAoPIOd

https://trello.com/b/XR8X79AR/trabajo-practico

que es story map? 42, 43, 44

-

selección de evento

reserva

////////////////////////////////////

de la teoria hasta donde llegamos a ver entra en el parcial: sobre todo estructura de scrum y todo lo de scrum

son 20 de v o f

5 de multiple choice que tiene que ver con el tp

//////

quedarse con 5 funcionalidades de cada coso para definir todo

solo necesito 5 tarjetas de cada funcionalidad general del sistema