UNIDAD N° 1
Ingeniería de Software en el contexto

(*) Solo leer

Introducción a la Ingeniería del Software. ¿Qué es?

Software

En general, podría definirse como un set de programas junto con la documentación que lo acompaña. Es decir está conformado por:

- Diversos programas independientes
- Archivos de configuración que se utilizan para ejecutar estos programas
- Documentación que describe la estructura del sistema
- Documentación para el usuario que explica cómo utilizar el sistema

Clasificación

- **Productos genéricos:** Son sistemas aislados producidos por una organización de desarrollo y que se venden al mercado abierto a cualquier cliente. La especificación es controlada por la organización que desarrolla.

- **Productos personalizados (o hechos a medida):** Son sistemas requeridos por un cliente en particular. La especificación por lo general, es desarrollada y controlada por la organización que compra el software.

La calidad que alcanza un software recae en la personalidad y el intelecto de los miembros del equipo que lo crean. Es decir que si aplicáramos un proceso que ha entregado excelente resultados para un equipo en otro equipo, muy posiblemente no se alcance el resultado de calidad logrado por el primer equipo en cuestión.

Ingeniería de Software

Es una disciplina de la ingeniería que se preocupa de todos los aspectos de la producción de un software; **desde** las primeras etapas de la especificación **hasta** el mantenimiento del sistema después que se pone en operación. Parmas [1987] definió a la ingeniera en software como "Multi-person construcción of multi-version software". En esta definición existen dos frases clave:

> La función del ingeniero
> Los ingenieros aplican teorías, métodos y herramientas de la manera más conveniente siempre tratando de descubrir soluciones a los problemas, teniendo en cuenta que deben trabajar con restricciones financieras y organizacionales, por lo que buscan soluciones considerando estas restricciones.
>
> Disciplinas
> - **Técnicas:** Ayudan a construir el producto. Estas son: requerimientos, análisis, diseño, implementación, prueba.
> - **De Gestión:** Planificación de proyectos, Monitoreo y control.
> - **De Soporte:** Gestión de Configuración, métricas, aseguramiento de la calidad.
>
> ¿Cuál es la diferencia entre ingeniería de software e ingeniería de sistemas?
> La ingeniería en sistemas se ocupa de los aspectos basados en computadoras del desarrollo de sistemas, incluye hardware, software y procesos de ingeniería. La ingeniería de software es parte de este proceso general.

¿Costos de la ingeniería en sistemas?
A grandes rasgos, el 60% de los costos son de desarrollo y el 40% de testing; en programas personalizados, frecuentemente los costos de evolución son mayores que los de desarrollo.

Software vs Manufactura

- El software es menos predecible
- No hay producción en masa, casi ningún producto de software es igual a otro
- No todas las fallas son errores.
- El software no se gasta
- El software no está gobernado por las leyes de la física

Estado Actual y Antecedentes. La Crisis del Software.

La crisis del Software

Este término tiene origen en 1968 por F.L Bauer, quien recalco la dificultad para generar software libre de defectos, fácilmente comprensibles y que sean verificables. Las causas son:

- La introducción de nuevas tecnologías de hardware que generaron la posibilidad de construir software más complejo y de mayor tamaño.
- La complejidad que supone la tarea de programar.
- Los cambios a los que tiene que ser sometido un software para ser continuamente adaptado a las necesidades de los usuarios.

Problemáticas con el desarrollo de software

- La versión final del producto no satisface las necesidades del cliente
- No es fácil extenderlo y/o adaptarlo, es decir agregar funcionalidad puede resultar una tarea sumamente difícil e incluso imposible.
- Mala documentación
- Mala calidad
- Tiempos y costos excedidos a los del presupuesto.

Motivos de productos de software exitosos

- Involucramiento del usuario 15.9 %
- Apoyo de la Gerencia 13.0 %
- Enunciado claro de los requerimientos 9.6 %
- Planeamiento adecuado 8.2 %
- Expectativas realistas 7.7 %
- Hitos intermedios 7.7 %
- RRHH competentes 7.2 %

Causas de productos de software fallidos

- Requerimientos incompletos 13.1 %
- Falta de involucramiento del usuario 12.4 %
- Falta de recursos 10.6 %
- Expectativas poco realistas 9.3 %
- Falta de apoyo de la Gerencia 8.7 %
- Requerimientos cambiantes 8.1 %

Ariane 5 vuelo 501: "... errores de especificación y diseño del sistema de referencia inercial... "

Proyecto de Software

Definición de proyecto de Software

Un proyecto es un esfuerzo temporal que se lleva a cabo para crear un producto, servicio o resultado único. Sus características son:

Únicos

Todos los proyectos por similares que sean, tienen características que lo hacen únicos, por ejemplo, tienen cronogramas diferentes.

Orientado a objetivos

- **No ambiguos:** deben ser claros, dado que son los responsables de guiar el desarrollo del proyecto.
- **Medibles:** a fin de poder determinar el avance sobre el proyecto.
- **Alcanzables:** es decir que los mismos puedan ser realizados por el equipos

Duración limitada en el tiempo

Es decir tienen un principio y un fin. Es decir cuando se alcanzan los objetivos el proyecto termina. El ejemplo más claro de lo que no es un proyecto, es una línea de producción dado que nunca finaliza.

Conjunto de tareas interrelacionadas basadas en esfuerzo y recursos

Esto se debe a la complejidad sistémica de los problemas

Administración de Proyectos

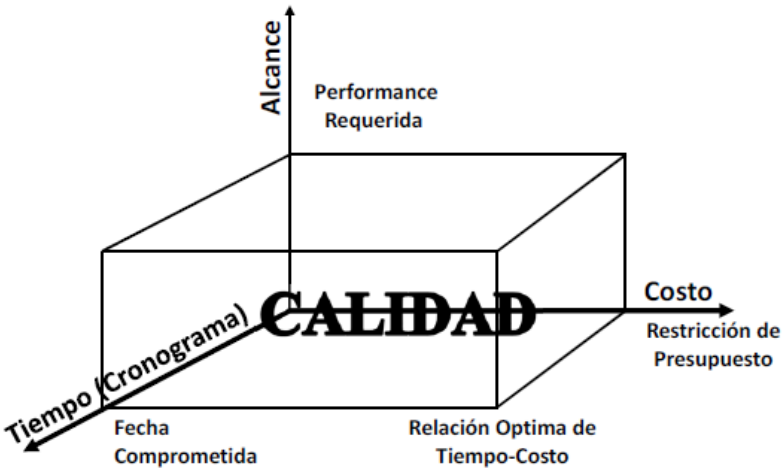
Administrar un proyecto correctamente implica tener el trabajo hecho en tiempo, con el presupuesto acordado, habiendo satisfecho los requerimientos.

Definición formal: "aplicación de conocimientos, habilidades, herramientas y técnicas a las actividades del proyecto para satisfacer los requerimientos del proyecto".

La administración de proyectos incluye:

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

Triple Restricción



Alcance
Son los requerimientos del proyecto, es decir los límites o el ámbito sobre el cual se va a mover el mismo. Esta variable es la que primero se negocia con el cliente.

Tiempo
Hace referencia al calendario, cuáles serán las fechas especificadas para realizar las entregas que determinaran el avance del proyecto.

Costo
Son los recursos que se verán implicados en el desarrollo del proyecto. Esto incluye equipamiento, infraestructura, equipos, salarios, entre otros.

El balance de estos tres factores afecta directamente la calidad del proyecto, ya que un proyecto de alta calidad es aquel que entrega el producto requerido, el servicio o resultado, satisfaciendo los objetivos en el tiempo estipulado y con el presupuesto planificado. Es responsabilidad del Líder de proyecto balancear estos tres objetivos (que a menudo compiten entre ellos y por lo general casi nunca se cumplen simultáneamente).

Errores clásicos en proyectos y organizaciones

Ver Resumen_02.doc – Página 5

El desarrollo de software es una actividad complicada. Un proyecto de software típico puede presentar muchas oportunidades para aprender de los errores originados en la aplicación frecuente de prácticas inefectivas. Estos errores se clasifican en:

- Gente.
- Proceso.
- Producto.
- Tecnología.

Rol del Líder de Proyecto / Equipo

Líder de Proyecto

Para ser un líder uno debe sentirse cómodo con los cambios y entender a la organización. El líder debe tener los **Hard Skills,** que son los conocimientos del producto, herramientas y técnicas, y los **Soft Skills**, que es la capacidad de trabajar con gente y son los más difíciles de conseguir (comunicación, liderazgo, creatividad, organización, motivación, empatía).

Es responsabilidad del líder de proyecto:

- Definir el alcance del proyecto
- Identificar involucrados
- Detallar de tareas (wbs) estimar tiempos y requerimientos
- Identificar recursos y presupuestos
- Identificar y evaluar riesgos
- Preparar planes de contingencia
- Controlar hitos
- Participar en las revisiones de las fases del proyecto
- Administrar el proceso de control de cambios y producir reporte de status.

El Equipo

Grupo de personas comprometidas en alcanzar un conjunto de objetivos de los cuales se sienten mutuamente responsables. Tienen diversos conocimientos y habilidades, trabajan juntos desarrollando sinergia y en general es un grupo pequeño. Tienen sentido de responsabilidad como una unidad.

Stakeholders

Son los interesados del proyecto, incluye el equipo de proyecto, el equipo de dirección, el líder de proyecto y el patrocinador. Este último debe tener jerarquía en la empresa y garantizar recursos. En metodologías agiles el líder del proyecto es el scrum master y el patrocinador el Product Owner.

¿Qué es un plan de proyecto?

El plan de proyecto es el documento en el cual se especifica:

¿Qué es lo que hacemos?
Es decir se especifica el alcance del proyecto el cual guiará el desarrollo del mismo.

¿Cuándo lo hacemos?
Especifica el calendario, las fechas estipuladas para cubrir el alcance del proyecto.

¿Cómo lo hacemos?
Es decir las actividades y tareas que se deberán llevar a cabo para cubrir el alcance del proyecto, en el tiempo especificado.

¿Quién lo va a hacer?
Definición de responsables, los que llevaran a cabo cada una de las tareas descriptas anteriormente en el cómo.

Este documento es de suma importancia que sea correctamente mantenido y actualizado de manera permanente, es decir "no está escrito en piedra" y tiene ciclos de cambios. El mismo funciona como un paraguas o caparazón para el equipo. Si el plan de proyecto no se actualiza, es muy probable que el proyecto tienda al fracaso.

"Un plan es a un proyecto lo que una hoja de ruta a un viaje"

¿Qué implica la planificación de un proyecto?

Definir el Alcance

- **Alcance del Producto**
  Todas las características que pueden incluirse en un producto o servicio. El cumplimiento del alcance del producto se mide contra la Especificación de Requerimientos.

- **Alcance del Proyecto**
  Es todo el trabajo y solo el trabajo que debe hacerse para entregar el producto o servicio con todas las características y funciones especificadas. El cumplimiento del alcance del producto se mide contra el Plan de Proyecto (o el Plan de Desarrollo de Software).

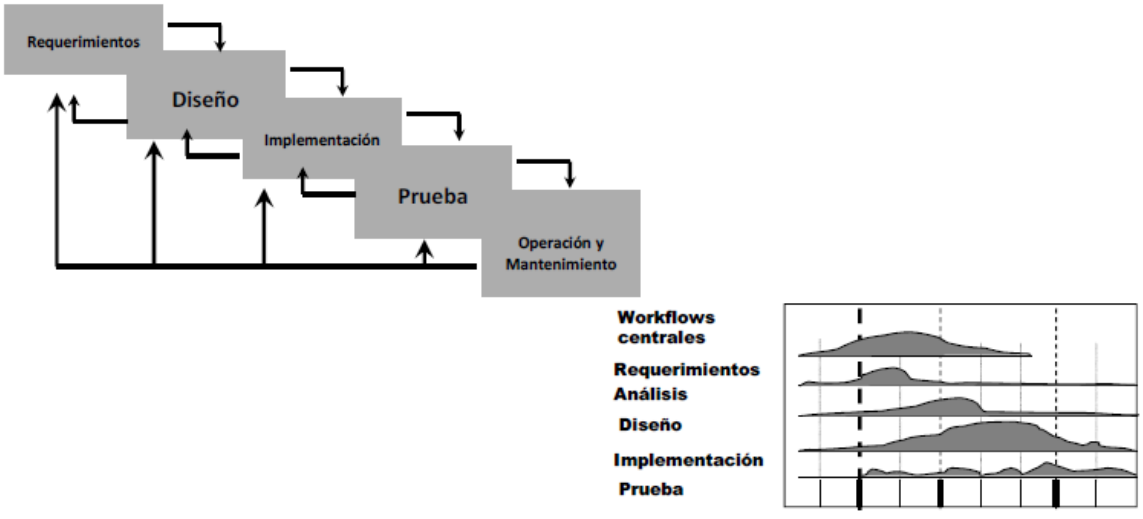
Es importante aclarar que el alcance del proyecto es menor que el alcance del producto, dado que este último sobrevive al proyecto que le da origen.

Definir el Proceso y Ciclo de Vida

El ciclo de vida de un proyecto define:

- Que trabajo técnico debería realizarse en cada fase
- Quien debería estar involucrado en cada fase
- Como controlar y aprobar cada fase
- Como deben generarse los entregables
- Como revisar, verificar y validar el producto.

La mayoría de los ciclos de vida comparten algunas características a saber: Los costos y el personal son bajos al inicio y más altos hacia el final cayendo abruptamente cuando el proyecto termina.



Estimación

Se estima **tamaño, esfuerzo, calendario, costos** y **recursos críticos.** Para estimar primero debemos conocer el dominio en búsqueda de una unidad de peso, de modo que después podamos convertir esa unidad en estimaciones, dejando el juicio experto como último recurso.

Gestión de Riesgos

Ver Resumen_02.doc – Pág. 10

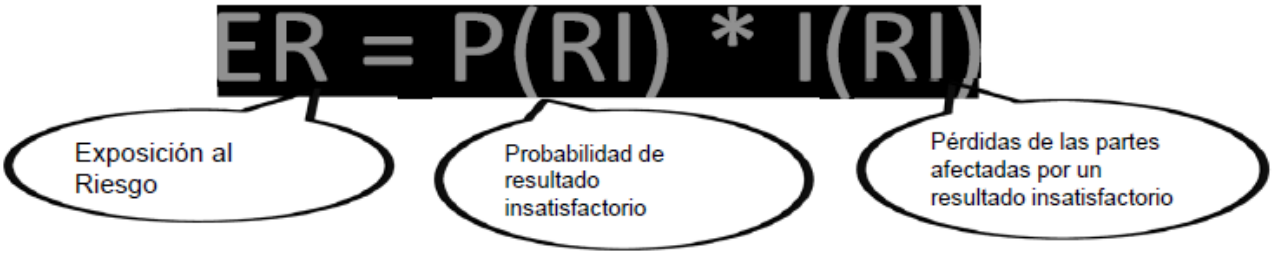
Definición

Un riesgo podría definirse como la probabilidad de que un evento no deseado ocurra en un proyecto, es decir un evento esperando por suceder y que podría llegar a comprometer el éxito del proyecto.

- No existe proyecto sin riesgo
- Los riesgos pueden provocar incrementos en los costos o desbordamiento del proyecto

Clasificación

- Riesgos del proyecto: afectan la calendarización o los recursos del proyecto.
- Riesgos del producto: afectan a la calidad o al rendimiento del software que se está desarrollando.
- Riesgos del negocio: afectan a la organización que desarrolla o suministra el software.



- **La exposición del Riesgo** es la amenaza total del riesgo
  - Exposición de Riesgo = Probabilidad x Impacto
  - **Ejemplo:** Exposición de Riesgo = **0.75 x 4 = 3.0**
  - **Ejemplo:** Exposición de Riesgo = **0.6 x $100,000 = $60,000**

Para medir los riesgos, se utiliza lo que se conoce como **exposición al riesgo**, esto es la amenaza total del riesgo, la cual resulta del producto de la probabilidad de ocurrencia del riesgo y el impacto que generará dicho riesgo si se convierte en problema (medida del daño). Este valor permite organizar y gestionar los riesgos, y se debe hacer foco en reducir la exposición del riesgo.

Riesgos comunes

- Cronogramas y Presupuesto Irreales
- Desarrollo de funciones erróneas
- Desarrollo de interfaces de usuarios erróneas
- Cambios de requerimientos

Identificación y especificación de riesgos
Se hace al recibir los requerimientos, lo que nos permite actuar sobre la fuente. Se identifican y se especifica claramente el riesgo.

Analizar
Determinar la exposición al riesgo, el producto de la probabilidad por el impacto. Una vez calculado dicho valor, se realiza el ordenamiento por mayor exposición. Es importante aclarar que no se manejan más de 10 riesgos semanales en sesiones de control de proyecto.

Planificar
Se realiza el plan, con estrategias de mitigación y contingencia. Cuando se habla de contingencia se hace referencia a actuar reduciendo al mínimo el impacto del riesgo, y mitigación hace referencia a reducir la probabilidad de ocurrencia del riesgo. Además entra en juego el "que pasa sí", el cual implica la existencia de alternativas para escenarios posibles.

Seguimiento y control
Se realiza sobre la ejecución de las acciones de mitigación y contingencia, para realizar el posterior aprendizaje, para conformar luego una Base de Conocimiento de Riesgos. Es importante aclarar que si se replanifica el proyecto, es necesario realizar nuevamente la gestión de riesgos, y acá por ejemplo, es donde entra en juego y es de gran importancia la base de conocimientos de riesgos.

<u>Consideraciones</u>

- La identificación y la gestión de riesgos se deben realizar a través de todas las fases del proyecto.
- Los riesgos e incertidumbre son altos al inicio del proyecto.
- La capacidad de los involucrados para influir en las características finales del producto y en el costo final es más alta al inicio del proyecto.

Definición de Métricas

El proyecto también incluye las métricas, utilizadas para informar, motivar, comparar, entender, evaluar, predecir, y mejorar. **Unidad** es una cantidad particular, definida y adoptada por convención, con la que poder comparar otras cantidades de la misma clase para expresar sus magnitudes respecto a esa cantidad particular [ISO-15939].

- **Métrica directa** es aquella que se puede obtener sin depender de ninguna otra métrica y cuya forma de medir es a través de un método de medición, ejemplo líneas de código de un módulo.

- **Métrica indirecta** es aquella que proviene de una función de cálculo cuyos argumentos son otras métricas directas o indirectas, ejemplo total de horas de programación.

Características de las métricas

- **Validez** está relacionada a la exactitud de la métrica, es decir la proximidad con respecto al valor verdadero.
- **Confiabilidad** a la precisión, hace referencia al repetibilidad o reproductibilidad de la medida.

Las métricas de software tienen 3 dominios:

- **Producto**
- **Proceso**
- **Proyecto**

La consolidación de las métricas de proyecto crea **métricas de proceso** que son públicas para toda la organización. Las métricas se hacen por nivel, hay métricas para el desarrollador, para el equipo y para la organización.

Asignación de Recursos

Programación de Proyectos

Definición de Controles

Monitoreo y Control

Ver Resumen_00.docx – Pág. 4

El control de proyectos se refiere a comparar el progreso con respecto al plan, con el objetivo de verificar si estamos bien encaminados, de no ser asi generar medidas preventivas que nos permitan volver a lo esperado, si ya se han detectado desviaciones deberán tomarse acciones correctivas.

El objetivo del monitoreo es determinar el estado del proyecto:

- **Proyecto bajo control:** se están alcanzando los hitos del proyecto a tiempo con los recursos estimados y con un nivel de calidad esperado. (Se cumplen Estándares y compromisos de planificación)

- **Proyecto fuera de control:** se deberá replanificar y renegociar el plan.

Consideraciones

- Para el seguimiento se realizan reuniones semanales que pueden incluir encuentros con el cliente.

- Se puede incluir un chek-list de fin de fase, revisiones del plan, auditoria y reuniones post mortem.

- Sobre el grafico de Gantt se puede estimar el porcentaje de tareas no finalizadas y que aún no han comenzado, debiendo haberlo hecho.

- Se realiza seguimiento o tracking para minimizar riesgos conocer y aceptar la realidad y tomar acciones correctivas.

- **Seguimiento de proceso:** se realiza un control por hitos o por tareas

- **Seguimiento de producto:** se realiza un control sobre los entregables.

(*) Si el seguimiento realiza la colección de métricas, reportes de estado y actualización del Gantt, a partir de hitos cumplidos y los reportes de estado de las acciones que se están llevando a cabo (cerradas, en proceso, pendientes, etc). También se debe realizar una revisión y actualización del estado de los riesgos y revisar los objetivos fijados de calidad. Mensualmente se recolectan esfuerzo, errores y tamaño del código.

Es importante aclarar que los proyectos se atrasan de a un día por ves. Es decir los proyectos se atrasan por horas y se recuperan con días. Recuperar un proyecto atrasado depende del momento dentro del ciclo de vida en el cual estoy.

Factores para el éxito de un proyecto

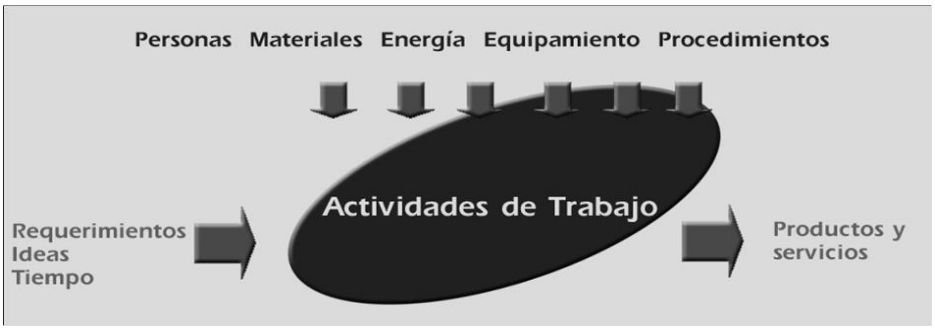
- ✓ Monitoreo & Feedback
- ✓ Misión/objetivos claros
- ✓ Comunicación

Factores para el fracaso de un proyecto

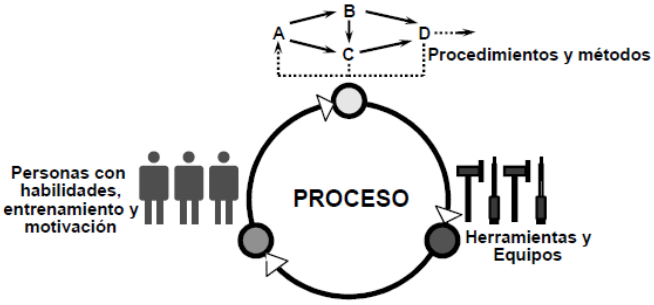
- × Fallas al definir el problema
- × Planificación basada en datos insuficientes
- × Planificación realizada por grupo de planificaciones
- × No hay seguimiento del plan de proyecto
- × Mala planificación de recursos
- × Estimaciones basadas en supuestos sin consultar datos históricos
- × Nadie estaba a cargo

Proceso de desarrollo de Software

Proceso y Proceso de Software



Un **proceso** es una secuencia de pasos ejecutados para un propósito dado (IEEE), mientras que un **proceso de software** se define como un conjunto de actividades, métodos, prácticas y transformaciones que la gente usa para desarrollar o mantener software y sus productos asociados (Sw-CMM).



Consideraciones

- Estas actividades varían dependiendo de la organización y el tipo de sistema que debe desarrollarse, pero deben incluir: productos, roles, responsabilidades y condiciones
- El proceso debe ser explícitamente modelado si va a ser administrado.
- Un proceso de desarrollo se puede aplicar a varios ciclos de vida. La elección del ciclo de vida depende de la estrategia y este debe permitir seleccionar los artefactos a producir, definir actividades y roles, y modelar conceptos.
- Las personas utilizan herramientas y equipos para llevar a cabo los procedimientos que componen el proceso de desarrollo.

Actividades fundamentales

- **Especificación de software:** clientes junto con ingenieros definen el software a producir y sus restricciones.
- **Desarrollo:** diseño y programación del software.
- **Validación:** verificación para asegurar que es lo que el cliente quiere.
- **Evolución:** donde se modifica el software para reflejar los requerimientos cambiantes del cliente y mercado.

Proceso Definido vs. Proceso Empírico

Definido

Un proceso definido asume que podemos repetir el mismo proceso una y otra vez, indefinidamente, y obtener los mismos resultados.

- Estos procesos están inspirados en las líneas de producción.
- La administración y el control provienen de la predictibilidad del proceso.

Empírico

Asume procesos complicados con variables cambiantes, si el proceso se repite, los resultados obtenidos pueden ser diferentes.

- Estos procesos se ajustan de mejor forma a procesos creativos y complejos.
- La administración y el control es por medio de inspecciones frecuentes y adaptaciones.

Ciclos de vida

Definición de ciclo de vida

Son modelos genéricos (no descripciones definitivas) de los procesos de software; es decir, son abstracciones del proceso que se usan para explicar los distintos enfoques del desarrollo de software. En definitiva, un ciclo de vida de software es una representación de un proceso, el cual grafica una descripción del mismo desde una perspectiva particular.

Características

- También se los conoce como modelo de proceso.
- Especifican las fases del proceso y el orden en el que se llevan a cabo (requerimientos, especificación, diseño, etc)
- Es una guía para la administración del proyecto ya que indica el progreso a través de hitos.
- Los modelos de ciclos de vida se han vuelto necesarios debido a que los sistemas son más complejos por el aumento de funcionalidad y la mayor variedad de usuarios.
- Son independientes de los procedimientos de cada actividad del ciclo de vida.

Clasificación de los ciclos de vida

Los ciclos de vida que se presentan a continuación NO son mutuamente excluyentes y con frecuencia se usan en conjunto, sobre todo para sistemas complejos y de gran envergadura. Estos

- **Secuenciales**
  Toma las actividades fundamentales del proceso; especificación, desarrollo, validación y evolución y los representa como fases separadas del proceso: especificación de requerimientos, diseño de software, implementación, pruebas, etc.

  - ✓ Desarrollo dirigido por un plan
  - ✓ Cada etapa genera documentación para realizar un monitoreo constante contra el plan.
  - ✓ Muy útil cuando los requerimientos son claros y es poco probable un cambio drástico durante el desarrollo

- ✗ La documentación puede ser burocrática y excesiva
- ✗ En etapas finales puede ser que haya que hacer re trabajo por cambios en requerimientos o fallas de diseño

- **Iterativos**
  Este enfoque vincula las actividades de especificación, desarrollo y validación. El sistema se desarrolla como una serie de versiones (incrementos) y cada una añade funcionalidades a la versión anterior.

  - ✓ La especificación, desarrollo y validación están entrelazadas en lugar de separadas y aisladas.
  - ✓ Rápida retroalimentación a través de las actividades.
  - ✓ Muy útiles para sistemas de requerimientos cambiantes (Ej.: e-commerce, empresariales, etc)
  - ✓ Más fácil y menos costoso implementar cambios.
  - ✓ Cada iteración genera funcionalidad para el cliente.

  - ✗ Los incrementos progresivos tienden a degradar la estructura del sistema.

- **Recursivos**
  Se inicia con algo en forma completa, como una subrutina que se llama a si misma e inicia nuevamente. Se presenta un prototipo que va mejorando con cada vuelta.

  - ✓ Se generan productos independientes de la implementación, que pueden ser reusables en sistemas de características similares.

  - ✗ Puede ser más costoso en tiempo y dinero readaptarlos para reutilizarlos para los diferentes proyectos.
  - ✗ La tecnología puede ser obsoleta.
  - ✗ Pueden carecer de mantenimiento o documentación.

Modelos de ciclos de vida

Code and Fix
Se desarrolla sin especificaciones o diseño y se lo modifica hasta que el cliente este satisfecho. Los cambios se realizan durante el mantenimiento, lo que es muy caro.

Modelo en Cascada Puro
Se sigue una secuencia de pasos ordenada y se realiza revisión al final de cada fase. Es conducida por la documentación con fases que no se superponen.

- Permite encontrar errores en etapas tempranas.
- Hay exceso de documentación y falta de resultados hasta el final.
- Se utiliza cuando la definición del producto es estable o los requerimientos de calidad dominan a los de costo y tiempo.

Modelo en Cascada con Fases Solapadas
Hay un fuerte grado de solapamiento. La documentación es menor pero es más difícil hacer el seguimiento de progreso.

Modelo de Entrega por Etapas
Se ven signos tangibles de progreso. Es útil cuando el cliente necesita funcionalidad de inmediato y las necesidades se modifican. Debe haber una planificación rigurosa para que funcione.

Modelo en Cascada con Retroalimentación
Es una variación del modelo clásico en que es posible volver a una etapa anterior antes de llegar al final aunque es muy caro hacerlo.

Modelo en Cascada con Subproyectos
Es un modelo secuencial. Se avanza en cascada hasta el diseño arquitectónico, de ahí en más se avanza en partes dividiendo en Subproyectos.

Modelo en espiral
Busca minimizar los riesgos haciendo un análisis de riesgos y buscando alternativas al iniciar cada fase. Al final de cada fase se planifica la siguiente y se evalúa si se sigue adelante.

- Permite evaluar la factibilidad de un nuevo producto.
- En proyectos grandes la identificación de los riesgos puede costar más que el desarrollo.
- Es un modelo adecuado para proyectos con ciclos de vida largos, que puede utilizar equipos diferentes en cada ciclo y muestra el progreso al fin de cada ciclo.

Modelo Evolutivo
Se fija el tiempo o el alcance mientras el otro se va modificando. Es útil cuando los requerimientos no son claros y se realizan entregas tempranas al cliente.

- El problema es que es un modelo recursivo que repite todas las tareas y depende de que los diseñadores desarrollen un sistema fácil de modificar.

Diseño para Cronograma
No se asegura alcanzar la versión final, aunque si una entrega del producto para la fecha definida con los aspectos más importantes. Es útil cuando no hay seguridad sobre las capacidades de programación.

Diseño para Herramientas
Consiste en incluir solo la funcionalidad soportada por las herramientas de software existentes. Se usa en proyectos muy sensibles al tiempo. Puede combinarse con otros ciclos de vida, pero no se podrán implementar todos los requerimientos.

Prototipación Evolutiva
Se da cuando hay cambios rápidos de requerimientos y los clientes no se comprometen con los requisitos. Requiere menos documentación pero no se pueden programar los release. Se desarrolla el concepto del sistema a medida que se avanza.

Elección de un ciclo de vida

La elección depende de muchos factores tales como:

- Riesgos técnicos
- Riesgos de Administración
- Volatilidad de los requerimientos
- Ciclo de tiempo requerido
- Aspectos del cliente
- Tamaño del Equipo

Algunas consideraciones

- Una herramienta para la planificación y el monitoreo de proyectos.
- Un modelo de progreso del proyecto.
- Independiente de los métodos y procedimientos de cada actividad del ciclo de vida.
- Muy abstracto.

Métodos de Desarrollo Ágiles

Objetivo del Enfoque Ágil

El objetivo primordial es construir software de forma rápida para aprovechar las actuales oportunidades y responder ante la amenaza competitiva, convirtiendo la velocidad de entrega en el requerimiento fundamental de los sistemas de software: **"software de entregas rápidas en un entorno cambiante".**

Los desarrollos agiles **se utilizan para entornos con gran variabilidad de requerimientos**, ya que los clientes encuentran imposible predecir como un sistema afectará sus prácticas operacionales o que cambios habrá en el entorno (mercado o políticas de negocio) que pueden dejar el sistema completamente obsoleto.

Los procesos de desarrollo de software rápido se diseñan para producir rápidamente un software útil, el cual no **se desarrolla** como una sola unidad, sino **como una serie de incrementos**, y cada uno de ellos incluye una nueva funcionalidad del sistema.

- Por lo general se crean nuevas versiones (incrementos) del sistema cada 2 o 3 semanas y se pone a disposición del cliente.

- Involucran a los clientes en el proceso de desarrollo para conseguir una rápida retroalimentación sobre los requerimientos cambiantes y minimizan la cantidad de documentación con el uso de comunicaciones informales en lugar de reuniones formales con documentos escritos.

- **Fowler:** "Un compromiso útil entre nada de proceso y demasiado proceso"

Valores del manifiesto Ágil

Individuos e interacciones por sobre procesos y herramientas

En metodologías agiles estoy centrado sobre los individuos y por lo tanto los roles son intercambiables a diferencia de las metodologías tradicionales.

Software funcionando por sobre documentación detallada

Las metodologías agiles entregan software funcionando todo el tiempo. Se documenta todo aquello que agregue valor al producto y este centrado en mi cliente (valor agregado). Si la documentación lo hace, tiene que ir creciendo en cada iteración.

Colaboración por sobre negociación con el cliente

Hay que estar dispuesto al cambio y cerca del cliente para predecirlo. Hay ciertos requerimientos que surgen de la colaboración con el cliente, donde van apareciendo alternativas que generan cambios en el producto. La relación con el cliente puede o no ser 1 a 1 (Ej. en Lyn thinking no).

Responder a cambios por sobre seguir un plan

Los nuevos requerimientos pueden tener un mayor valor que los iniciales. Se reciben cambios de requerimientos, y estos son vistos en su mayoría de buena forma por el equipo.

Los 12 Principios del Manifiesto

1. La prioridad es satisfacer al cliente a través de releases tempranos y frecuentes
2. Recibir cambios de requerimientos, aun en etapas finales
3. Releases frecuentes (2 semanas a un mes)
4. Técnicos y no técnicos trabajando juntos TODO el proyecto
5. Hacer proyectos con individuos motivados
6. El medio de comunicación por excelencia es cara a cara
7. La mejor métrica de progreso es la cantidad de software funcionando
8. El ritmo de desarrollo es sostenible en el tiempo
9. Atención continua a la excelencia técnica
10. Simplicidad - Maximización del trabajo no hecho
11. Las mejores arquitecturas, diseños y requerimientos emergen de equipos auto organizados
12. A intervalos regulares, el equipo evalúa su desempeño y ajusta la manera de trabajar

Definido vs Empírico

En desarrollo de software tenemos dos tipos de procesos, definidos y empíricos. El desarrollo de sistemas tiene tanta complejidad y es tan impredecible que se requiere un modelo empírico de control de proceso.

Proceso Definido

Es aquel que tiene definidas las entradas, los requerimientos que deben alcanzar en esas entradas y las salidas que produce así como algún tipo de verificación sobre esa salida. Si la verificación no se cumple se vuelve accionar sobre la entrada y se vuelve a iterar. Los modelos de mejora como SPICE, ISO, o CMMI, requieren que el proceso sea definido, porque si es definido es repetible, si es repetible es medible, si es medible es mejorable.

Proceso Empírico

Son aquellos que están completamente basados en la experiencia de quien lo ejecuta, donde el proceso esta internalizado en la persona. Hay iteraciones frecuentes y adaptación al proceso.

Proceso ágil

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

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

En el universo ágil hay categorías, XP, TDD, FDD, CM, etc. Scrum está en término medio, el cual representa en 55% del nivel de uso entre las metodologías agiles.

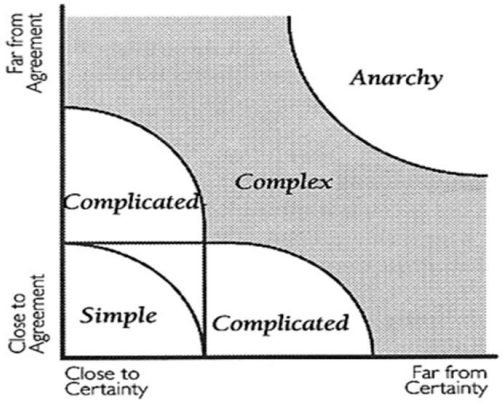
¿Cuándo es aplicable Agile?

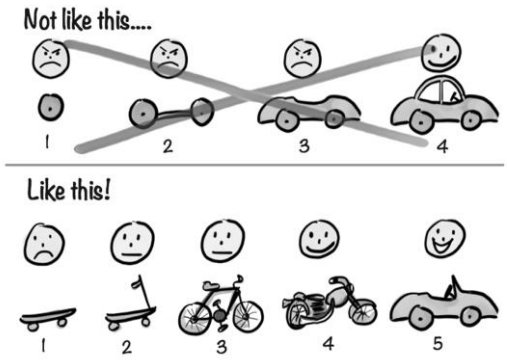
Agile da mejores resultados para problemas que caen dentro de "Complex"

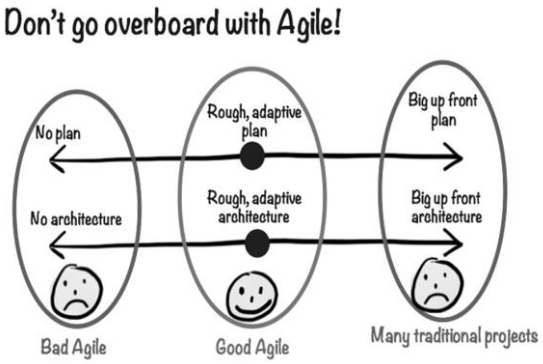
El desarrollo de nuevos productos y Knowledge Work tiende a estar dentro de complex.

Investigación está dentro de Anarchy

La duda sobre donde colocar mantenimiento ¿Simple?







User Stories

User Storie

Una user storie contiene una descripción corta de una funcionalidad valuada por un usuario o cliente de un sistema. Se llaman "stories" porque se supone que Ud. cuenta una historia, es decir lo que se escribe en la tarjeta no es lo más importante, sino la conversaciones entre los clientes y desarrolladores a cerca de la historia.

Los user stories son:

- Una necesidad de usuario
- Una descripción del producto
- Un ítem de planificación
- Token para una conversación
- Mecanismo para diferir una conversación

Estructura de una User Storie

Como **<< rol de usuario >>**
yo puedo **<< actividad >>**
de forma tal que **<< valor de negocio >>**

Rol de usuario: representa quien está realizando la acción o quien recibe el valor de la actividad.

Actividad: representa la acción que realizará el sistema.

Valor de negocio: comunica porque es necesaria la actividad, es decir de qué forma agregar valor al negocio.

Componentes de una User Storie

1. **Tarjeta:** Una descripción de la historia, utilizada para planificar y como recordatorio.

2. **Conversación:** Discusiones acerca de la historia que sirven para desarrollar los detalles de la historia.

3. **Confirmación:** Pruebas que expresan y documentan detalles y que pueden usarse para determinar cuándo una historia está completa.

Detalles de la User Storie

Los detalles de la historia se obtienen de conversaciones entre el dueño del producto y el equipo. Sin embargo, si es necesario más detalle puede proveerse como adjunto en forma de algoritmo, planilla de cálculo, prototipo o lo que sea. Puede obtenerse en el tiempo con discusiones adicionales con los involucrados.

Criterios de aceptación de una US

Los criterios de aceptación son condiciones de satisfacción ubicadas en el sistema. Son las condiciones que deben cumplirse para determinar que la historia fue desarrollada completamente, de lo contrario, los desarrolladores no tendrían un parámetro para definir si la misma fue cumplimentada o no.

Pruebas de aceptación de una US

Expresan detalles resultantes de las conversaciones entre clientes y desarrolladores; suelen usarse para completar detalles de la US y se ven como un proceso de 2 pasos:

- Notas en el dorso de la US.
- Pruebas muy completas utilizadas para demostrar que la historia se ha hecho correctamente y se ha codificado en forma completa.

Las mismas deben escribirse antes que la programación empiece (las escribe el cliente o al menos especifica que pruebas se utilizaran para determinar si la historia ha sido desarrollada correctamente).

¿Qué NO es una User Storie?

Las user stories NO son especificaciones detalladas de requerimientos (como los UC), son expresiones de intensión, "es necesario que haga algo como esto… ".

- No están detalladas al principio del proyecto
- Necesita poco o nulo mantenimiento
- Pueden descartarse después de la implementación
- Junto con el código sirven de entrada a la documentación que se desarrolla incrementalmente después.

Dividiendo User Stories

Las User Stories frecuentemente se derivan de épicas (**epics**) y características (**features**), conceptos vagos y grandes de algo que queremos hacer para un usuario. No hay una rutina definida para dividir User Stories, sólo lineamientos generales:

- Que sea una pieza de valor para el usuario.
- Que tenga un corte vertical a través del sistema.
- Que entre en una iteración.

Investment Themes (Temas de Inversión)

Representan un conjunto de iniciativas o propuestas de valor que conducen la inversión de la empresa en sistemas, productos, aplicaciones y servicios con el objetivo de lograr una diferenciación en el mercado y/o ventajas competitivas.

Los THEMES son una combinación de inversión en:

- Inversión en ofertas de productos existentes, mejoras, soporte y mantenimiento.

- Inversión en nuevos productos y servicios que mejorarán los beneficios y/o ganarán porciones de mercado en el período actual o al corto plazo.

- Inversión a futuro en productos y servicios avanzados.

- Inversión hoy, pero que no dará beneficios hasta dentro de unos años.

- Inversión en reducción (estrategia de retiro) para ofertas existentes que están cerca del final de su vida útil.

Epics

Son iniciativas de desarrollo de gran escala que muestran el valor de los temas de inversión; son requerimientos de alto nivel que se utilizan para coordinar el desarrollo, son estimadas, priorizadas y mantenidas en el backlog; son planificadas antes de la planificación del release y descompuestas en características (features).

- **Epics de negocio:** son funcionales o de experiencia de usuario.
- **Epics arquitectónicas:** usadas para implementar cambios tecnológicos que deben realizarse a elementos significativos.

US vs THEME vs EPIC

- **EPIC** = user storie de gran tamaño
- **THEME** = colección de user stories relacionadas.
- **US:** descripción de funcionalidad deseada, contada desde la perspectiva del cliente.

Spikes

Las spikes son un tipo especial de historias de usuario (invento de XP), usado para quitar el riesgo e incertidumbre de una US y otra faceta del proyecto; se clasifican en técnicas y funcionales y pueden utilizarse para:

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

Spikes técnicas

Usadas para investigar enfoques técnicos en el dominio de la solución (evaluar performance potencial, decisiones de hacer o comprar y evaluar la implementación de cierta tecnología).

Spikes funcionales

Usadas cuando hay cierta incertidumbre respecto de cómo el usuario interactuará con el sistema, usualmente son mejor evaluadas con prototipos para obtener retroalimentación de los usuarios involucrados.

INVEST Model

- **I**ndependent: Calendarizable e implementables en cualquier orden.
- **N**egotiable: el "qué" no el "cómo".
- **V**aluable: Debe tener valor para el cliente.
- **E**stimable: Para ayudar al cliente a armar un ranking basado en costos.
- **S**mall: Deben ser "consumidas" en una iteración.
- **T**esteable: Demostrar que fueron implementadas.

Ejemplos

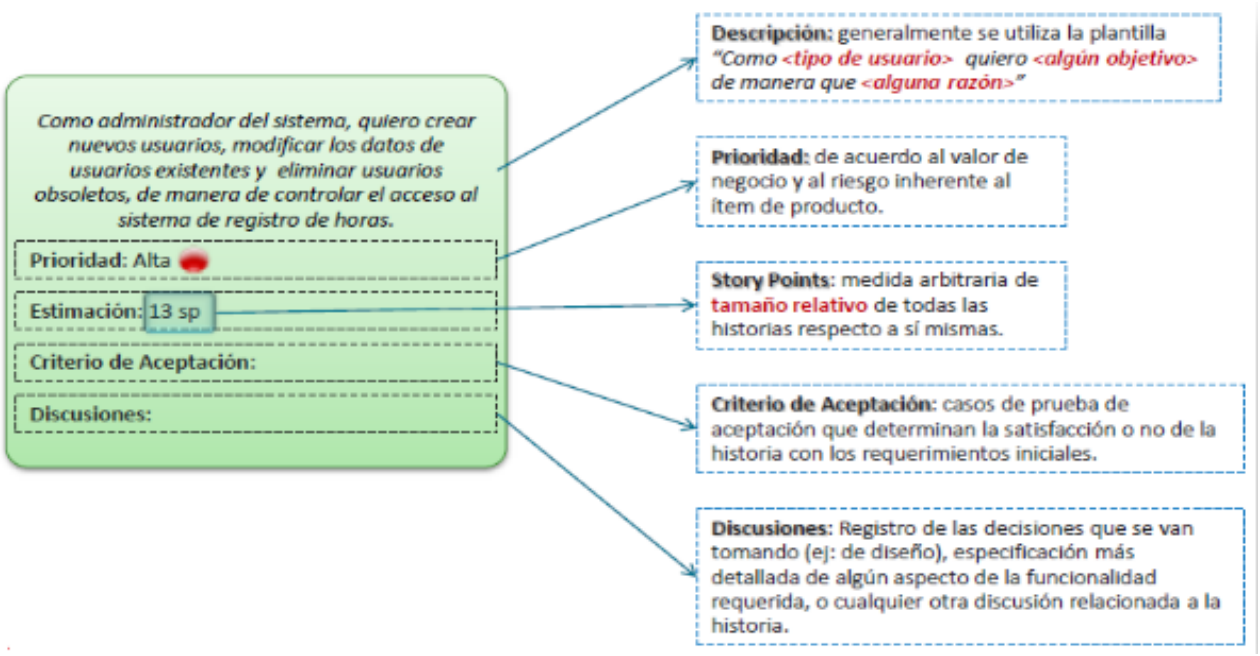
1) User Storie

"Como cliente, quiero poder ver mi consumo de energía diario, así puedo bajar mis costos y usos de energía."

Criterio de aceptación:

- Leer los metros DecaWatt cada 10 segundos y mostrarlos en el portal con incrementos de 15 minutos y mostrarlos en forma local en cada lectura.
- No tendencias multi-días por ahora (otra historia).

2) User Storie



Administración de proyectos ágiles

La responsabilidad principal de los administradores del proyecto de software es dirigir el proyecto, de modo que el software se entregue en tiempo y con el presupuesto planeado, monitoreando el avance en el desarrollo del mismo.

El desarrollo ágil tiene que administrarse de tal modo que busque el mejor uso del tiempo y recursos disponibles, por lo que es necesario un enfoque diferente a la administración tradicional de proyecto, sino que se adapte al desarrollo incremental, como ser Scrum, que más allá de ser un método ágil, su enfoque está en la administración iterativa del desarrollo y no en los enfoques técnicos específicos para la ingeniería de software ágil (como XP, aunque pueden usarse de forma conjunta).

- A cada ciclo se lo llama "sprint", y en él se desarrolla una versión o incremento del sistema. Los mismos son de longitud fija (2 a 4 semanas).

- El punto de partida es el "backlog" del producto, que es la lista de trabajo a realizar en el proyecto. Durante la planificación de cada sprint, se revisa, se asignan prioridades y riesgos, y el cliente define al comienzo de cada sprint si desea o no introducir nuevos requerimientos o tareas.

- En la fase de selección, se trabaja de forma conjunta entre el equipo de desarrollo y el cliente para definir que funcionalidades se van a desarrollar en el sprint.

- Una vez comenzado el sprint, la comunicación entre el cliente y el equipo de desarrollo se hace a través del "scrum master".

- Al finalizar cada sprint, el trabajo se revisa y presenta a los participantes, luego se comienza con el siguiente sprint.

El Scrum master es el facilitador que ordena las daily meetings (reuniones breves y enfocadas donde se comparten avances, problemas y planes para el día siguiente), rastrea el atraso del trabajo a realizar, registra decisiones, mide el progreso del atraso y se comunica con los clientes y administradores fuera del equipo; todo el equipo participa de la toma de decisiones en caso que sea necesario replantear cursos de acción para corregir problemas.

SCRUM

¿Qué es Scrum?

Es un framework que establece lineamientos para gestionar un proyecto de manera ágil, el cual permite crear su propio proceso para crear nuevos productos.

Carácterísticas

- Es simple y puede implementarse en pocos días pero perfeccionarlo lleva tiempo.
- Menos tiempo planeando, definiendo tareas, creando reportes y más tiempo con el equipo investigando las situaciones.
- Supone dominios impredecibles, por lo cual no supone un proceso repetible.
- El control se alcanza con inspecciones frecuentes y los ajustes correspondientes.
- Se debe planear, ejecutar, reflexionar y volver a iniciar.
- Permite trabajar con sistemas funcionando, con tecnologías inestables y el surgimiento de requerimientos.

Cimientos

Empirismo

Concepto filosófico en donde la experiencia es la base para la formación de conocimiento. Los procesos definidos y la planificación detallada en las primeras fases son remplazadas por ciclos de inspección y revisión just in time, y ciclos adaptativos.

Auto Organización

En grupos de trabajo que manejan sus propias cargas de tareas y se organizan entre ellos alrededor de un objetivo claro y tomando en cuenta las restricciones.

Colaboración

Los líderes de Scrum, diseñadores de productos y clientes trabajan en conjunto con los desarrolladores para alcanzar los objetivos. Ellos no los gerencian o dirigen.

Priorización

Trabajar en lo más importante, no perder el tiempo haciendo foco en el trabajo que no tiene y/o agrega valor

Time Boxing

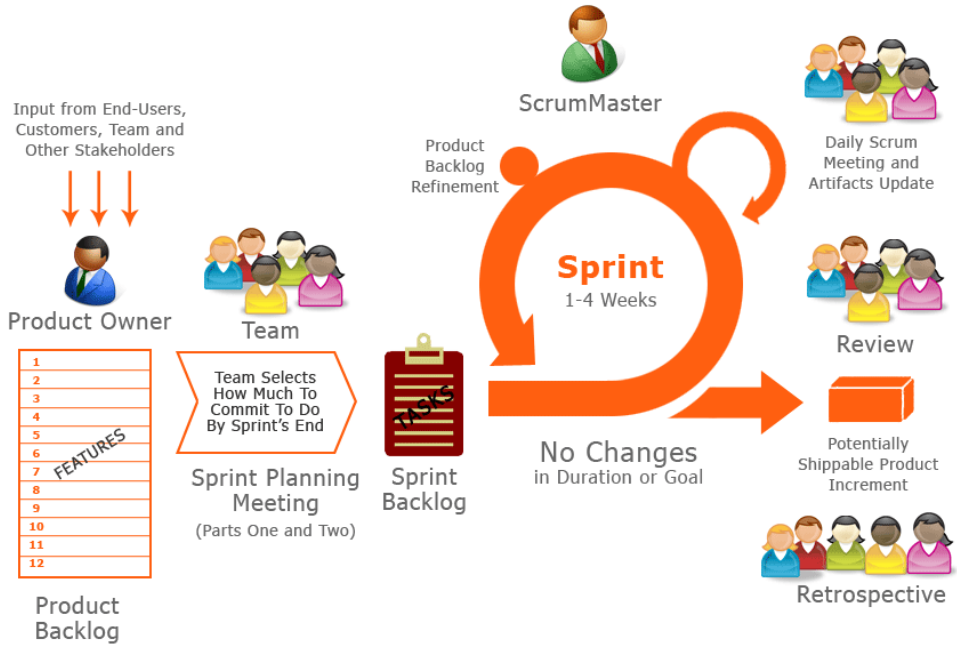
Es una técnica de planificación de proyectos, generalmente de software, en donde el schedule (programación) es dividido en un numero separado de periodos de tiempo (time box), normalmente entre 2 y 6 semanas, los cuales tienen sus propios entregables, fechas y costos. Esta técnica crea el ritmo que guía el desarrollo.

Valores de Scrum

- Compromiso
- Foco
- Abierto a ideas
- Respeto
- Valentía

El ritmo de scrum

Estructura del Sprint



Período fijo de tiempo, sugerido en 30 días. Durante el Sprint no se puede cambiar el alcance, agregar funcionalidad ni modificar las reglas del equipo. Los Sprints producen un incremento usable basado en el producto anterior. No deben existir interrupciones y debe hacerse foco en las tareas.

Durante las Sprint Planning, se decide que funcionalidad se va a implementar, y el equipo decide cómo se hará. En las Daylies, de aproximadamente media hora el Scrum Master pregunta a los asistentes que se completó desde la última reunión, que obstáculos se presentaron y que se hará hasta la próxima reunión. Los Scrum diarios mejoran la comunicación y el conocimiento de todos, elimina otras reuniones, promueve decisiones rápidas y remueve obstáculos.

Durante los Sprint se congelan tres de las cuatro variables del proyecto, que son el tiempo, los costos y la calidad, pero puede variar la funcionalidad siempre y cuando se cumpla con el objetivo del Sprint. Un Sprint puede cancelarse si el objetivo se vuelve obsoleto, si las condiciones técnicas o del mercado cambian, o si el equipo lo decide ya que no se puede alcanzar el objetivo o se encuentra un problema muy grande. Cancelar un Sprint es un costo y está mal organizacionalmente.

Factores claves

- Objetivos declarados claramente y comprensibles.
- Foco en las tareas
    - Una tarea se encuentra "lista" cuando cumple el "check check", el miembro del equipo la marca como "hecha" y el Scrum master verifica la tarea

- En un sprint se congelan 3 de 4 variables de un proyecto:
    - Tiempo: 20 – 30 días.
    - Costo**:** salarios + ambiente.
    - Calidad: generalmente un estándar organizacional.

- Sin cambios
    - No se puede cambiar el alcance.
    - No se puede agregar funcionalidad.
    - No se pueden modificar las reglas del equipo.

- El mismo equipo puede descubrir más trabajo a ser hecho.
- Cambios de requerimientos se colocan en el product backlog y se prioriza nuevamente.
- El equipo puede cambiar funcionalidad siempre y cuando cumpla con el objetivo del sprint.
- El UNICO que puede sacar funcionalidad es el PO.
- El Scrum master es el responsable de encontrar y resolver impedimentos que puedan presentarse en el sprint.

Elementos de un Sprint

- Reuniones de Scrum.
- Se produce un incremento usable y visible**,** basados en un producto anterior.
- Compromiso de los miembros a la asignación.

Tareas obligatorias

- Reuniones de Scrum diarias en donde participan todos los miembros del equipo.
- El Backlog del Sprint debe estar actualizado y con las últimas estimaciones de los desarrolladores.

Cancelación de un Sprint – Muy costoso

- Se puede cancelar un Sprint si las circunstancias hacen que no sea necesario
    - El objetivo se vuelve obsoleto
    - Las condiciones técnicas o de marcado cambian.

- Decidido por el equipo
    - Porque no se puede alcanzar el objetivo
    - Se encuentran en un problema muy grande

Roles de Scrum

Stakeholders y Usuarios

Product Owner

- Controla y gestiona el Backlog
- Una persona, no un comité
- Establece prioridades, es decir nadie puede definirle al equipo una prioridad diferente.
- Conoce en detalle el negocio

Scrum Master

- Responsable de que las prácticas, valores y reglas se realicen
- Nexo entre la gerencia y el equipo
- Responsable de mantener el equipo enfocado
- Dirige los Scrum diarios (Daylies):
    - Hora y lugar fijo
    - Gerentes pueden asistir pero no participan
    - No son reuniones de diseño

- Realiza el seguimiento del avance, comparando el progreso planeado con lo real
- Asegura que se resuelvan los impedimentos y toma decisiones rápido
- Trabaja con la gerencia y el cliente para identificar el PO
- Responsable, junto con el PO y el equipo en producir el Backlog del producto

Equipo

- Alrededor de 7 personas como máximo, en caso de ser más dividirlos en varios equipos trabajando sobre el mismo backlog.
- Autónomos y auto-organizados.
- Se comprometen a entregar un conjunto de ítems del Backlog al final del Sprint.
- No hay roles y todos codifican.
- Libertad de acción, limitados por estándares y políticas organizacionales.
- Scrum es empírico, a veces se alcanza el objetivo reduciendo funcionalidad.

Entregables de Scrum

Product Backlog

Características

- Cola priorizada de funcionalidades técnicas y de negocio, que necesitan ser desarrolladas.
- Debe contener la funcionalidad mínima necesaria para la siguiente iteración.
- Contiene una lista de requerimientos; características, funciones, tecnologías, bugs, etc.
- Todo lo que hay que hacer a lo largo del desarrollo.
- Cada ítem tiene valor para el usuario o el cliente.
- Priorizado por el PO y repriorizada al comienzo de cada Sprint

Origen

- Marketing
- Ventas
- Desarrollo
- Soporte al cliente

Refinamiento
Es como jugar Asteroides, grandes rocas (Epics) se descomponen en rocas más pequeñas (stories) que son lo suficientemente pequeñas para ser desarrolladas y entregadas en un sprint.

Estimaciones del Backlog

- Las estimaciones son iterativas, y comprenden un esfuerzo colaborativo entre las partes.
- Si un ítem no puede ser debidamente estimado, se debe dividir en el Backlog.
- Los estimados sirven de base para la funcionalidad en el Sprint.

Sprint Backlog

El equipo determina que ítems del product backlog van a contemplarse en el desarrollo del siguiente sprint, es decir que debe hacerse para cumplir con el objetivo de dicho Sprint.

- El Product Owner suele asistir
- Se realiza una lista de tareas que tomen de 4 a 16 horas para completarse
- El Sprint Backlog no se modifica durante el sprint
- Si el equipo descubre no se puede alcanzar con todos los ítems del sprint backlog, el Scrum Master y el PO determinan si algún ítem puede ser removido.

Ejemplo

Tareas	L	M	M	J	V
Codificar UI	8	4	8		
Codificar negocio	16	12	10	4	
Testear negocio	8	16	16	11	8
Escribir ayuda online	12				
Escribir la clase foo	8	8	8	8	8
Agregar error logging			8	4	

Producto de software potencialmente listo para producción

Parte del producto desarrollado en un sprint en condiciones de ser usado en el ambiente correspondiente.

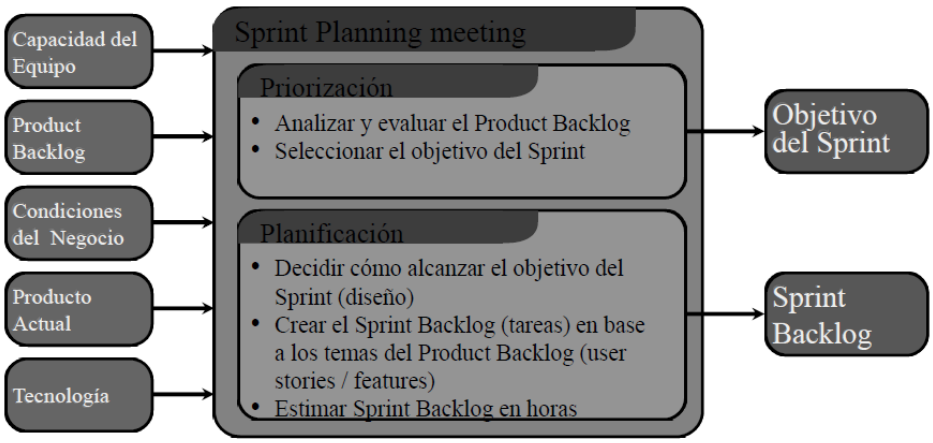
Reuniones de Scrum

Sprint Planning

Se podría hablar de dos reuniones consecutivas, en primer lugar el equipo se reúne con el PO, la gerencia y los usuarios para decidir que funcionalidad se va a implementar en el siguiente sprint, es decir que ítems del product backlog se contemplaran en el siguiente sprint (Sprint Backlog). Y en segundo lugar el equipo se reúne para decidir cómo llevarlo a cabo.

Características

- Inputs: product backlog, último incremento, métricas.
- Duración de 4 horas como máximo.
- Outputs: minuta de planificación; sprint backlog, miembros del team y que se va a entregar.



Daily (Scrum)

Reuniones diarias de corta duración (15 a 30 minutos), en las cuales se da el estado de avance de lo que se está desarrollando en el sprint, siendo el Scrum Master en encargado de guiar el desarrollo de la misma. Cada miembro del equipo responde a las siguientes tres preguntas:

- ¿Qué hice desde la última reunión?
- ¿Qué voy a hacer hasta la próxima reunión?
- ¿Qué obstáculos se presentaron?

Características

- Asiste todo el equipo
- No son para solucionar problemas
- Se trata de compromiso delante de pares (no es status report al scrum master)
- El Scrum Master es el facilitador.
- Fiel reflejo del control en Scrum (seguimiento diario de avance)

Sprint Demo/review

Reunión informativo en donde el equipo presenta el incremento desarrollado a gerentes, clientes, usuarios y el Product Owner, será este último quien aceptará o rechazara dicho incremento.

Características

- Informal, duración máxima de 4 horas
  - 2 Horas de preparación
  - No usar diapositivas
- Todo el equipo participa, se invita a todo el mundo.
- Se reportan problemas encontrados
- Cualquier ítem puede ser agregado, eliminado y re-priorizado del product backlog.
- Se estima el nuevo Sprint y se asignan tareas.

Restropectivas

Reunión que generalmente se realiza al final de cada sprint en la cual se echa un vistazo a lo que funciona y a lo que no desde el punto de vista del equipo principalmente.

Características

- Normalmente de 1 a 4 horas
- Todo el equipo participa, posiblemente clientes y otros interesados.
- Puede realizarse cada 2 sprints si el equipo ya trabaja junto hace tiempo
- Se busca determinar:
    - Malas prácticas: ¿Qué hay que dejar de hacer?
    - Buenas practicas ¿Qué hay que continuar haciendo?
    - Mejoras: ¿Qué hay que comenzar a hacer?

Story time/Grooming (opcional)

El equipo se reúne con el PO para discutir ítems del backlog de alta prioridad, determinar su criterio de aceptación y asignar la priorización a cada nuevo ítem. Esto ocurre inicialmente antes del desarrollo y después iterativamente en cada sprint.

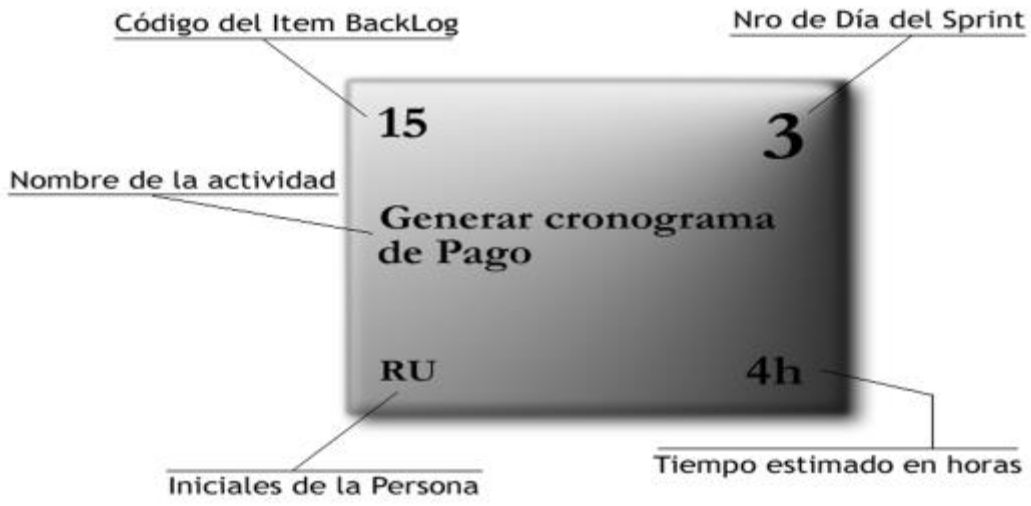
Herramientas de Scrum

Taskboard

Tablero utilizado en Scrum para realizar un seguimiento del trabajo realizado Este está conformado de una serie de secciones:

- **Story:** ítems del product backlog (user stories, epics, invest themes)
- **To Do:** ítems del sprint backlog, es decir funcionalidad a ser contemplada en el sprint (US)
- **In Process:** user stories que están siendo trabajadas por el equipo.
- **To Verify:** user stories finalizadas de las cuales deben evaluarse sus criterios de aceptación.
- **Done:** user stories completadas y aceptadas por el PO.

Tarjetas de User Stories



Gráficos de Backlog

Son gráficos que brindan información relacionada al avance del proyecto, que resultan de gran utilidad para la gerencia a la hora de determinar:

- Progreso del sprint
- Progreso del Release
- Progreso del Producto

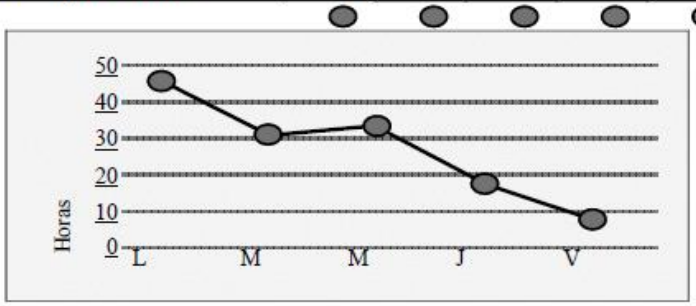
El backlog de trabajo es la cantidad de trabajo que queda por ser realizado, y la tendencia del backlog refleja el trabajo que queda vs el tiempo.

Burndown Charts

Ver 06_Intro_a_scrum_2015.pdf – Página 32

Indica cómo voy avanzando, es descendente, empieza con la fecha de hoy, termina con la fecha de fin de sprint (eje X = tiempo, fecha de hoy, y días hasta fin de sprint; eje Y = horas de trabajo o story points).

Tareas	L	M	M	J	V
Codificar UI	8	4	8		
Codificar Negocio	16	12	10	7	
Testear Negocio	8	16	16	11	8
Escribir ayuda online	12				



Beneficios de Scrum

- Se gestionan los cambios de requerimientos
- Se incorpora la visión de mercado
- Los clientes ven incrementos que refinan los requerimientos en un tiempo razonable
- Mejores relaciones con el cliente

Estimaciones de Software

Definición de Estimación

Una estimación es una predicción que tiene como objetivo predecir la completitud y administrar los riesgos. Se relaciona con los objetivos del negocio, compromisos y control.

Errores en las estimaciones

- Información imprecisa acerca del software a estimar o acerca de la capacidad para realizar el proyecto
- Demasiado caos en el proyecto (mal definido el proyecto)
- Imprecisión generada por el proceso de estimación.
- Una de las fuentes de error más común es omitir actividades necesarias para la estimación del proyecto tales como, requerimientos faltantes, licencias, reuniones, revisiones, etc.

Consideraciones

- Momentos apropiados para estimar:
  - Al inicio del proyecto
  - Luego de la especificación de requerimientos
  - Luego del diseño
- Una de las actividades más complejas en el desarrollo de software, luego de la definición del software.
- Por definición una estimación no es precisa, la mayor cantidad de veces nos equivocamos al estimar.
- Existe un universo de probabilidades asociado a las estimaciones
- 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.
- A mayor diferencia entre lo estimado y lo planeado, mayor es el riesgo.
- Las estimaciones no son compromisos
- Pasos: Estimaciones - WBS - Calendarización

Relación entre estimación y planes

El objetivo de la estimación es la precisión y de los planes es obtener un resultado en particular. Las estimaciones son la base de los planes, pero los planes no tienen que ser lo mismo que lo estimado. A mayor diferencia entre lo estimado y lo planeado mayor riesgo.

¿Por qué estimamos?

Existen dos razones principales por las cuales se llevan adelante las estimaciones:

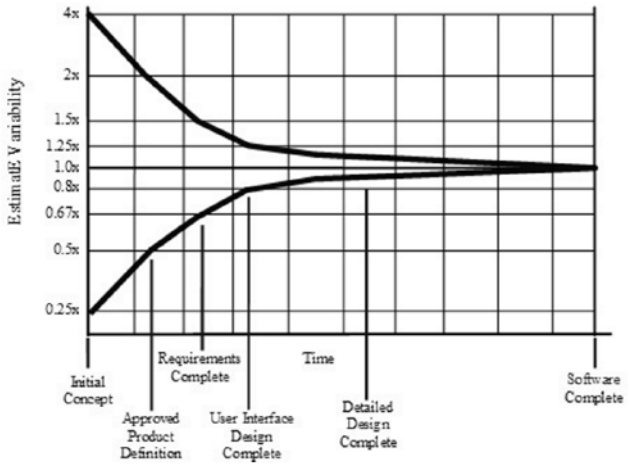
- Para predecir completitud
- Para administrar riesgos

Antes de que el proyecto comience, el líder del proyecto y el equipo de software deben estimar el trabajo que habrá de realizarse, los recursos que se requerirán y el tiempo que transcurrirá desde el principio hasta el final. Las estimaciones requieren:

- Experiencia
- Acceso a buena información histórica (métricas)
- El valor para comprometerse con predicciones cuantitativas cuando la información cualitativa es todo lo que existe.

Siempre se desea saber muy al inicio del proyecto cual será el costo, el tiempo que llevara, entre otros aspectos y como aún no se conoce con precisión cual será el alcance del producto resulta complejo predecir con exactitud estos aspectos, es por ello que a medida que avanza el proyecto, las estimaciones son más certeras, dado que se cuenta con mayor información sobre la cual realizar las estimaciones.

Típicamente la primera estimación difiere hasta un 400%



Al inicio del desarrollo hay una estimación errónea de 2 a 4 veces más, esto se debe a dos aspectos:

1. Al inicio del proyecto tenemos poca información y demasiada incertidumbre, lo que aumentan el riesgo de generar estimaciones erróneas.

2. Las estimaciones son de naturaleza optimista, por eso las estimaciones deben ser recalculadas conforme se avanza con el proyecto.

3. Se suele decir: "el que no sabe estimar, estima muchas veces, y el que no sabe planificar, planifica muchas veces".

Proceso de Estimación

1. **Inicio con la especificación de requerimientos**
   En general un proceso de estimación parte de la especificación de requerimientos, que contiene la funcionalidad que el cliente espera del producto.

2. **Estimación de tamaño**
   Lo primero que hay que estimar es el tamaño de lo que quiere el usuario.

3. **Estimación de esfuerzo**
   Determinado lo que el usuario quiere y su tamaño podemos calcular el esfuerzo, es decir horas hombres lineales para desarrollar el producto.

4. **Estimación de calendario**
   Una vez determinada la cantidad de horas necesarias, en función de las horas de trabajo diarias, se determina el calendario del proyecto.

5. **Estimación de costo y recursos**
   En función de lo anterior, podemos estimar el costo. El costo más significativo de un proyecto es el costo del esfuerzo. El resto, son costos insignificantes, que incluso en ocasiones se desprecia.

En función de esto, es posible saber a cuanto lo podemos vender y tener un precio competitivo. Antiguamente por las condiciones del mercado, donde había clientes cautivos y software propietario no se hacía cálculo de costos, sin embargo con la competencia y globalización el cálculo de costos se volvió una necesidad.

¿Qué estimamos?

En general las empresas utilizan su información historia propia para estimar. Lo que se estima es:

Tamaño

El tamaño es lo que primero se estima y comprende determinar qué tan complejo/grande es el producto a desarrollar.

Características

- Es el valor que más se busca en las estimaciones de software.
- En muchas situaciones, es lo más complejo de estimar.
- Existen diversas técnicas para estimar tamaño, originalmente se tomaban líneas de código sin comentarios (en desuso por múltiples razones conocidas), puntos de función, casos de uso, nro. de páginas web, cantidad de clases, etc.
- En metodologías agiles, el tamaño corresponde a una medida de la cantidad de trabajo necesario para producir una feature/story.

Esfuerzo

Horas personas lineales que necesito para construir el producto con la funcionalidad esperada por el cliente. Estas horas se ven afectadas por:

- Proceso de desarrollo
- Características del proyecto
- Organización y equipo
- Personas: nivel de expertise, nivel de conocimiento de la tecnología, del dominio, etc.

Muchos proyectos estiman el esfuerzo con una lista detallada de tareas. Para estimar inicialmente el esfuerzo se usa como base el tamaño. La mayor influencia en el esfuerzo son el tamaño del software y la productividad del equipo. Cuando no se dispone de datos históricos pueden utilizarse tablas de productividad por tipos de software.

Calendario

Es decir determinar cuánto tiempo me va a llevar construir el producto, se establecen fechas, se busca determinar que tareas podrían realizarse en paralelo para reducir tiempos, etc.

Costo

Determinar derivado de todo lo anterior el monto total en dinero que requerirá el proyecto para llevarse adelante. El costo más significativo es el que se deriva del esfuerzo de los integrantes del equipo, en muchas situaciones se tiende a despreciar los demás costos

Recursos

También se estiman las necesidades de puestos de trabajo, licencias, espacio físico, ventilación, luz, etc. Los recursos deben estimarse antes del costo, pero la mayoría de las empresas ya tienen estos recursos y los costos están amortizados, por lo que no se imputan directamente al proyecto.

¿Cómo mejorar las estimaciones?

La estimación del costo y el esfuerzo nunca será una ciencia exacta. Demasiadas variables (humanas, técnicas, ambientales, políticas, etc.) pueden afectar el costo final del software y el esfuerzo aplicado a desarrollarlo. Sin embargo, la estimación del proyecto de software se puede transformar de una práctica oscura en una serie de pasos sistemáticos que proporcionan estimaciones con riegos aceptables, para ello existen varias opciones:

- **Demorar la estimación hasta más tarde en el proyecto.**
  Esta opción en la mayoría de los casos no es práctica, porque el cliente requiere en inicio del proyecto un conocimiento del costo y el tiempo que llevará el proyecto.

- **Basar las estimaciones en proyecto similares que hayan sido completados.**
  Esta alternativa puede funcionar relativamente bien si el proyecto en curso es muy similar a los previos, el problema es que hay más variables en juego como el equipo, el cliente, etc.

- **Emplear técnicas de descomposición relativamente simples para generar estimaciones de costo y esfuerzo del proyecto.**
  Viable para estimaciones de software.

- **Utilizar uno o más modelos empíricos en la estimación de costo y esfuerzo.**
  Viable para estimaciones de software.

Técnicas fundamentales de estimación

Técnica fundamental - Contar

La técnica fundamental de las estimaciones es contar, pero la pregunta es ¿qué contar?

- **En etapas tempranas**
  Requerimientos, características, casos de uso, user stories, etc.
- **En la mitad del proyecto**
  Puntos de función, pedidos de cambio, páginas web, reportes, pantallas, tablas, etc.
- **Más avanzado el proyecto**
  Defectos, clases, tareas, etc.

Consideraciones

- Contar aquello que esté muy relacionado con el tamaño del software que se está estimando.
- Buscar aquello que esté disponible lo más pronto en el proyecto.
- Entender lo que se cuenta, es decir si se contrasta con datos históricos, asegurarse que se utilizan las mismas presunciones que se utilizaron en el pasado.
- Lo que se cuenta debe requerir poco esfuerzo.

Métodos utilizados

Lo que se recomienda es utilizar distintos métodos de estimación y luego contrastar. Todos los métodos incluyen un "Factor de Ajuste" que permite que el método cierre (calibraciones).

- La calibración se realiza contra la industria, la organización o el mismo proyecto con etapas anteriores.
- De todas las opciones, compararse contra la industria es la más riesgosa pues depende del contexto.
- La estimación no es promesa ni certeza, ya que se asumen muchas variables desconocidas y elementos no tenidos en cuenta (tiempo de reuniones, tiempo de armado del set de pruebas, tiempo para revisiones, etc.).

Basados en la experiencia

Datos Históricos

Se deben recolectar los datos básicos del desarrollo de los proyectos, como el tamaño, esfuerzo, tiempo, defectos, entre otros de modo de ir generando una base de conocimiento que sea de utilidad para futuras estimaciones, con esto se busca una alternativa en la cual el conocimiento de cada individuo se transfiere a la organización y no estamos atados a una única persona.

- Industry Data: datos de otras organizaciones que aportan al mercado y desarrollan productos con algún grado de semejanza y que permite una comparación básica. (En general están es estándares).

- Historical Data: datos de la organización de proyectos que se desarrollaron y ya se cerraron.(Se guardaron datos-estimaciones)

- Project Data: datos del proyecto pero de etapas anteriores a la que se está estimando.(Cada vez que se abre una fase estimarla)

Consideraciones

- Dejar de lado el optimismo de pensar que no se van a tener los problemas que se enfrentaron en el pasado.

- La productividad de una organización es un atributo que no tendrá gran variación entre proyectos.

- Los datos históricos se deben utilizar para evitar discusiones entre desarrolladores y clientes.

- Cuando se recolectan datos históricos, se debe tener en cuenta que se haga de la misma manera en todos los proyectos.
    - Tamaño: ¿cómo se cuenta el código rehusado? (LDC, PF, tablas, CU por comp.**)**
    - Esfuerzo: ¿Qué unidad se utiliza? ¿Cómo se cuenta el tiempo en el proyecto?
    - Tiempo: ¿Cuándo comenzó? ¿Cuándo tenemos aprobado el presupuesto?
    - Defectos: ¿Cuántos defectos se originaron en el diseño?

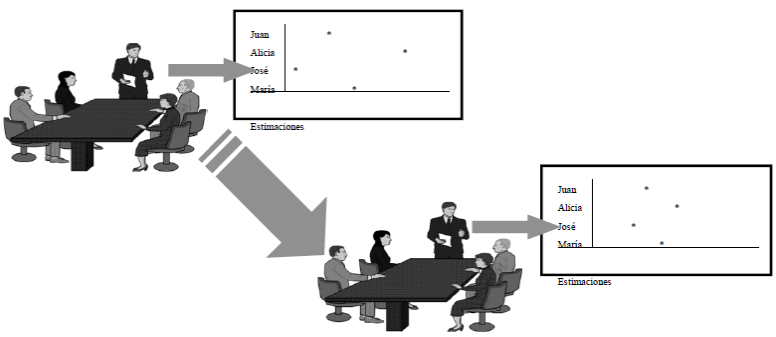
Juicio Experto

Es el método más utilizado en la práctica, se calcula que alrededor del 75% de las organizaciones utilizan este método de estimación, pero el problema se presenta cuando se utiliza como única técnica de estimación el juicio experto puro. Para esta técnica se requiere:

- Juicio Experto puro
  Un experto estudia las especificaciones y hace su estimación. Se basa fundamentalmente de los conocimientos con los disponga el experto, y como se observa a simple vista, si el experto se va de la organización, esta pierde su capacidad de estimación.

  Se debe estructurar el juicio experto de la siguiente manera:
    - Tener una granularidad aceptable sobre lo cual se va a estimar (WBS).
    - Usar el método optimista, pesimista y habitual (o+4h+p)/6.
    - Use un checklist y un criterio definido para asegurar cobertura
        - Para asegurarnos que estamos estimando correctamente
        - Tener cobertura completa de lo que vamos a estimar

- Juicio Experto: Wideband Delphi
  Similar al anterior pero grupal, es decir un grupo de personas se reúne con el objetivo de determinar lo que costará el desarrollo tanto en esfuerzo, como en duración.



1. Se dan las especificaciones a un grupo de expertos.
2. Se les reúne para que discutan tanto el producto como la estimación.
3. Remiten sus estimaciones individuales al coordinador.
4. Cada estimador recibe información sobre su estimación, y las ajenas pero de forma anónima.
5. Se reúnen de nuevo para discutir las estimaciones.
6. Cada uno revisa su propia estimación y la envía al coordinador.
7. Se repite el proceso hasta que la estimación converge de forma razonable.
    a) Si no converge luego de sucesivas iteraciones, se utilizan el enfoque optimista-pesimista-habitual, o bien la desviación estándar entre observaciones.

Analogía

Se compara con situaciones anteriores para poder estimar. Para que funcione debe determinarse de antemano cuales son los factores relevantes a tener en cuenta para buscar proyectos parecidos. Ejemplo (cantidad de casos de uso por complejidad). Cantidad de usuarios, tipos de tecnologías, equipos de trabajo asignado, etc. Es el método que mayor error tiene.

Factores

- Tamaño: ¿Mayor o menor?
- Complejidad: ¿Más complejo de lo usual?
- Usuarios: Si hay más usuarios habrán más complicaciones
- Otros factores:
    - Sistema operativo, entornos (la primera vez más).
    - Hardware, ¿Es la primera vez que se va a utilizar?
    - Personal del proyecto, ¿nuevos en la organización?

Basados exclusivamente en los recursos

En la estimación consiste en ver de cuanto personal y durante cuánto tiempo se dispone de él, haciendo las estimaciones exclusivamente en función de este factor. En la realización: "El trabajo se expande hasta consumir todos los recursos disponibles, independientemente si se alcanzan o no los objetivos."

Basados en la capacidad del cliente

Es un método en lo que se evalúa no es el producto que se está intentando vender, sino a quien se lo vende, se evalúa su capacidad de pago. Es decir en muchas ocasiones se suele cobrar menos a clientes como una estrategia de ingreso al mercado, ganar la confianza del cliente.

Método basado exclusivamente en el mercado

- Lo importante es conseguir el contrato.
- El precio se fija en función de lo que creemos que está dispuesto a pagar el cliente.
- Si se usa en conjunción con otros métodos puede ser aceptable, para ajustar la oferta.
- Peligroso si es el único método utilizado
- Actualmente en desuso dado que se utilizaba con clientes cautivos.

Basados en los componentes del producto o en el proceso de desarrollo

Cuando hablamos de producto, hacemos referencia a módulos y sus respectivas funciones, mientras que por procesos el resultado es la WBS. Dos técnicas:

- **Bottom-up**
  Se descompone el proyecto en las unidades lo menores posibles.
  Se estima cada unidad y se calcula el coste total.

- **Top-Down**
  Se ve todo el proyecto, se descompone en grandes bloques o fases.
  Se estima el coste de cada componente.

Métodos algorítmicos

En base al producto se realiza la ejecución de algunos algoritmos que contemplan algunas características del producto tales como tamaño, complejidad, conocimiento del dominio, etc. (factores multiplicadores), esto genera como resultado una estimación del esfuerzo.

- Que se un cálculo matemático no quiere decir que el resultado sea certero.

Estimaciones basadas en CU

En las estimaciones basadas en CU se tienen en cuenta una serie de factores para determinar el tamaño:

- Complejidad (simple, mediano, complejo, extremo)
- Cantidad y tipos de actores
- Dependencias
- Quien lo va a desarrollar
- Cuestiones de performance
- Si se puede revisar en función de otro CU
- Tecnología
- Dominio

Estimaciones Ágiles

En los equipos Agiles, las features/stories son estimadas usando una medida de tamaño relativo conocida como story points (SP). Existen una serie de consideraciones importantes:

- Son medidas relativas no absolutas.
- No es una medida basada en el tiempo
- Las personas no saben estimar en términos absolutos
- Somos buenos comparando
- Comparar es generalmente más rápido
- Se obtiene una mejor dinámica grupal y pensamiento de equipo más que individual.
- Se emplea mejor el tiempo de análisis de las stories.

Estimando tamaño

La palabra tamaño se refiere a cuán grande o pequeño es algo, por lo que en agile, se define como una medida de la cantidad de trabajo necesaria para producir una feature/story.

El tamaño indica:

- Cuán compleja es una feature/story
- Cuánto trabajo es requerido para hacer o completar una feature/story
- Cuán grande es una feature/story

Y qué hacemos con el tamaño!?????

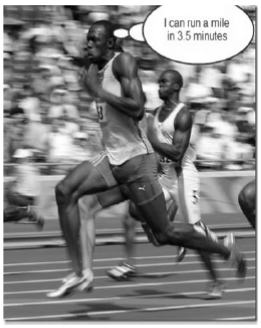
- Tamaño por números: 1 a 10
- Talles de remeras: S, M, L, XL, XXL
- Serie 2^n : 1, 2,4, 8, 16, 32, 64, etc.
- Fibonacci: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, etc.

Una vez elegida la escala no se cambia! Si se cambia cambiamos el metro patrón

Tamaño Vs. Esfuerzo

Las estimaciones basadas en tiempo son más propensas a errores debido a varias razones.

- Habilidades
- Conocimiento
- Asunciones
- Experiencia
- Familiaridad con los dominios de aplicación/negocio



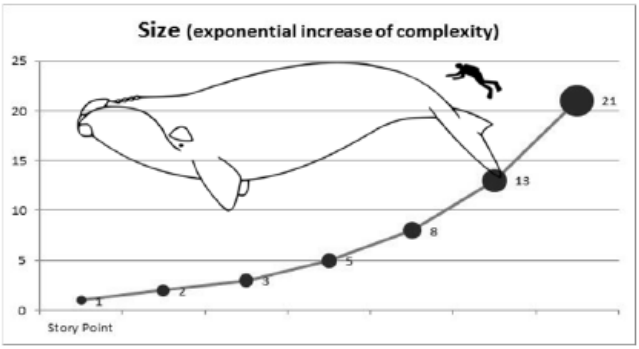
Tamaño *NO ES* esfuerzo

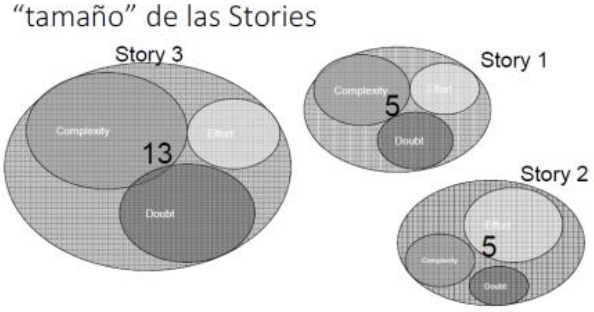
EL "QUIEN" DEFINE ESTA DIFERENCIA

¿Qué es un Story Point?

Es una unidad de medida específica de complejidad, riesgo y esfuerzo propia del equipo, es decir es la unidad que el kilo es a nuestro sistema de medición de peso.

- Story point da la idea del "peso" de cada story
- Decide cuando grande/compleja es
- Dicha complejidad tiende a incrementarse exponencialmente.





Capacity

Es una estimación de cuanto trabajo puede completarse en un período de tiempo basado en la cantidad de tiempo ideal disponible del equipo. Es teórica, y resulta de gran utilidad al momento de estimar, derivando de esta la velocidad.

Como se puede medir

- Esfuerzo (horas hombre)
- Puntos de Historia (Story Points)

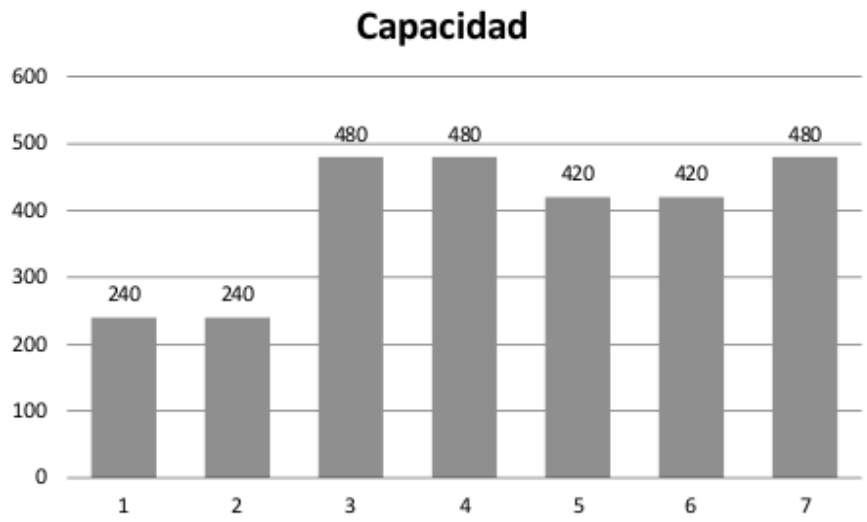
Cálculo de la capacidad

Horas de Trabajo Disponible por día (**WH**) x Días Disponibles Iteración (**DA**) = Capacity

Ejemplo

- Equipo de 8 miembros
- 4 miembros disponibles los 2 primeros sprints
- 1 miembro se casa en sprint 5 y 6
- 6 horas de trabajo

Sprint	1	2	3	4	5	6	7	Total
Horas	240	240	480	480	420	420	480	2760
Puntos de Historia	30	30	45	60	58	52	60	335



Consideraciones adicionales

- Individuos deben calcular capacidad realista
- Aplicar estimaciones honestas a sus tareas
- Considerar una capacidad máxima de 50% - 70%
- Comprender la capacidad a largo plazo con la velocidad y los puntos de historia
- Conocer promedio de finalización de un punto de historia para equipo/individuo

Velocity

La velocidad es una observación empírica de la capacidad del equipo para completar el trabajo por iteración, comprobable entre iteraciones de un equipo dado. Es importante aclarar, que la velocidad no se estima, se calcula. Se lo suele definir como "cuanta velocidad tengo con mi equipo para llevar algo a producción".

En definitiva, la velocidad está determinada por la cantidad de Storie Points que el Product Owner acepto, y se toma en la Sprint Review.

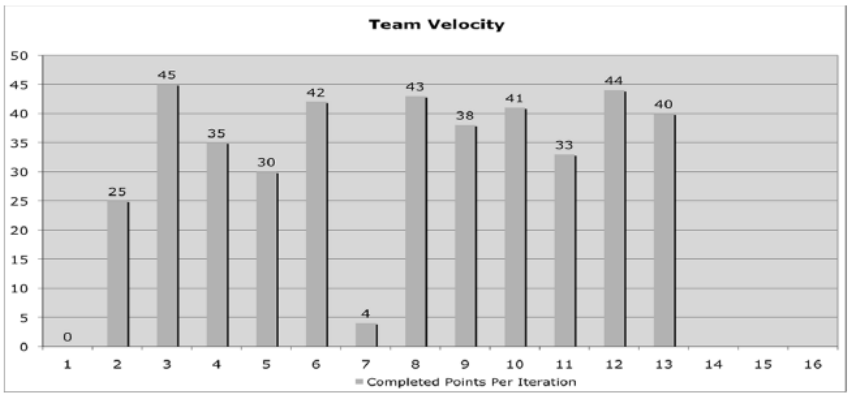
Consideraciones

- Solo cuenta el trabajo completado
- No es una estimación
- No es un objetivo a alcanzar
- No es comparable entre equipos
- No es comparable entre proyectos
- Me sirve para observar empíricamente mi capacidad de trabajo pero no para estimar mis objetivos.
- En Scrum baja la velocidad cuando se corren ciclos de testing y se encuentran bugs y tengo que hacer bugfixing.
- Estas dos métricas se mezclan en el Burn-downchart, acá puedo proyectas a mitad del sprint si llego o no.

Unidad de medida

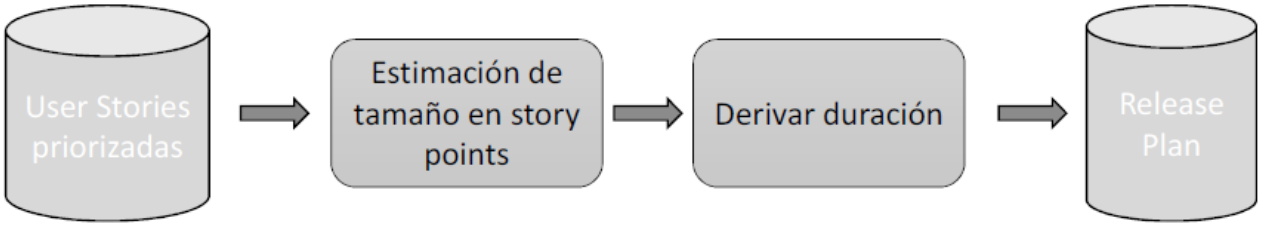
Cómo planea el equipo	Unidad de Medida
Compromiso con las historias	Historias
Tamaño relativo (puntos)	Puntos de Historia
Estimación (horas ideales)	Horas ideales

Ejemplo



¿Cómo se estima la duración de un proyecto?

Si la estimación se realiza utilizando Story Points para medir la complejidad relativa de las User Stories, para determinar la duración de un proyecto se realiza la derivación tomando el número total de story points de sus user stories y dividiéndolas por la velocidad del equipo.



Son las estimaciones en story points separa completamente la estimación de esfuerzo de la estimación de duración.

Poker Estimation

Técnica de estimación popular entre los practicantes de las metodologías Agiles, publicado por Mike Cohn. Resulta de la combinación de distintos métodos de estimación; opinión de experto, analogía y desegregación.

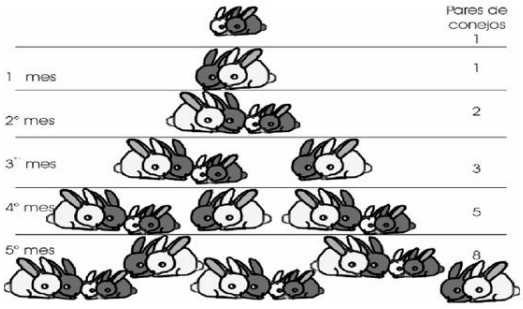
- Asume la idea de que el proceso de estimación debe ser llevado adelante por la gente que más capacitada esta para el mismo, es decir los desarrolladores (programadores, testers, ingenieros de bases de datos, analistas, diseñadores y demás).
    - "Las personas más competentes en resolver una tarea deben ser quienes las estiman"

- El product owner participa en la planificación pero no estima.

- El moderador es frecuentemente el dueño del producto o el analista, de todas formas, el moderador puede ser cualquiera ya que no hay privilegios asociados

Serie de Fibonacci

Esta técnica de estimación está basada en la Serie de Fibonacci, la cual define los valores que se utilizan para realizar la estimación de complejidad relativa de las User Stories, cada valor de la serie se obtiene de la suma de los dos valores anteriores. Se toma como referencia dado que crece exponencialmente al igual que lo hace el software.



Procedimiento

1. Se define una lista de actividades, módulos, casos de uso, user stories, etc.

2. Se acuerda cuál de las anteriores es la más simple.

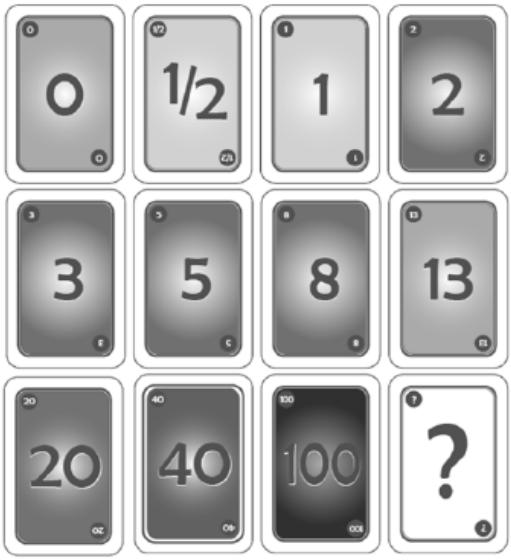
3. Se asigna tamaño/complejidad 1 a la actividad a partir de una analogía con experiencias anteriores.

4. Se ejecuta un wide band delphi para estimar complejidad/tamaño del resto de la lista, comparando con la más simple y solo asignado valores de complejidad de la serie Fibonacci. El mayor valor aceptable es 8, y ya muy posiblemente esa tarea deba ser dividida en tareas más simples.

5. Cada uno expone su valor estimativo para la tarea (juicio experto), esta se lleva a cabo mediante una presentación de cartas con el valor de complejidad, las misma boca abajo se dan vuelta de forma simultánea, Si la estimación difiere, el mayor y el menor estimador explican sus razones, con la idea de conocer sus opiniones.

6. Se estima el esfuerzo requerido para llevar a cabo la tarea Z y aplique el multiplicado a todas las actividades

7. El objetivo es que los estimadores converjan en una única estimación que puede ser usada para la historia de usuario.



- **0**: Quizás ud. no tenga idea de su producto o funcionalidad en este punto.
- **1/2, 1**: funcionalidad pequeña (usualmente cosmética).
- **2-3**: funcionalidad pequeña a mediana. Es lo que queremos. ☺
- **5**: Funcionalidad media. Es lo que queremos ☺
- **8**: Funcionalidad grande, de todas formas lo podemos hacer, pero hay que preguntarse sino se puede partir o dividir en algo más pequeño. No es lo mejor, pero todavía ☺
- **13**: Alguien puede explicar por que no lo podemos dividir?
- **20**: Cuál es la razón de negocio que justifica semejante story y más fuerte aún, por qué no se puede dividir?.
- **40**: no hay forma de hacer esto en un sprint.
- **100**: confirmación de que está algo muy mal. Mejor ni arrancar.

Consideraciones adicionales

- La estimación basada en UC es similar a los PF, para hacer estimación de tamaño independiente de la LDC, se suelen realizar una serie de cálculos contemplando factores de los CU como complejidad, reusabilidad, cantidad de actores: factores que en definitiva afectan al tamaño

- En relación al esfuerzo, además de tener en cuenta el tamaño de la actividad o tarea, es de suma importancia considerar factores relacionados a quien será el responsable de realizar esa tarea; es decir su experiencia, el conocimiento en la tecnología, conocimiento en el proceso de desarrollo, y factores organizacionales como madurez; que en definitiva, determinaran cuanto tiempo le llevará a esa persona realizar la tarea.

- Una vez obtenido el cumulo de horas necesarias, es necesario distribuirlo a lo largo del ciclo de vida del proyecto, es acá donde se busca determinar tareas a realizarse en paralelo para disminuir el calendario, NO el esfuerzo.

Estimación de proyectos pequeños

Para estimar proyectos pequeños lo mejor es utilizar datos históricos propios de la organización, dado que en estos casos los datos históricos de la industria no se adaptan correctamente a estas situaciones.

- Las estimaciones en proyectos pequeños dependen en gran medida de las capacidades de los individuos que hacen el trabajo. El SEI propone el Personal Software Process para proyectos unipersonales o 2 personas, conocidos como "mini proyectos".

Estimación de proyectos de Mantenimiento

Las estimaciones en proyectos de mantenimiento son complejas dado que la mayoría de los ejemplos que existen en datos de la industria son para estimar proyectos desde 0, sumado a esto que las características organizacionales suelen ser diferentes, como por ejemplo el número de personas asignadas a un proyecto de mantenimiento siempre es menor.

- En proyectos de mantenimiento se debe evaluar si la arquitectura actual del sistema podrá soportar una nueva funcionalidad, caso contrario el esfuerzo de mantenimiento incrementará el re-trabajo de la arquitectura

- Puede existir una sobreestimación si se utilizan modelos de estimación calibrados para nuevos proyectos.

- No existe información en el mercado para hacer analogía u otra técnica formal, más allá de que el mantenimiento ocupa el 80% de los costos, no se le da la importancia que debería.

- Generalmente tiene una fecha de entrega fija y un número fijo de personal, otro elemento con el que hay que lidiar al momento de estimar.

Problemas asociados a las estimaciones

- Las estimaciones no son compromisos.
- Las estimaciones no son exactas, debido a que son una predición, por lo cual siempre existirán desviaciones.
- No se documentan las experiencias previas, por lo cual las organizaciones no cuentan con esta base de conocimiento para estimar futuros proyectos.
- Una vez que obtenemos una estimación luego de haber aplicado métodos y técnicas, no es aceptada y es necesaria una modificación, perdiendo la objetividad de las estimaciones.
- Una de las fuentes de error más común en las estimaciones es omitir actividades necesarias para la estimación del proyecto.
    - Requerimientos faltantes.
    - Actividades de desarrollo no contempladas (documentación técnica, participación en revisiones, creación de datos para el testing, mantenimiento de producto en previas versiones, reuniones)
    - Actividades generales. (días por enfermada, licencias, cursos, reuniones de compañía).
- **Solución:** Uso de buffers
    - "Nunca tenga temor que estimaciones creadas por desarrolladores sean demasiado pesimistas, dado que los desarrolladores siempre generan cronogramas demasiado optimistas"

Estrategias de estimación

- Tenga idea del dominio y busque una "unidad de peso", por ejemplo Story Points.
- Use un método para convertir las "unidades de peso" en estimaciones (Poker Planning)
- Use el Juicio experto como último resorte.
- Agile es empírico, inspeccionar y adaptar es mandatorio.