Ingeniería en Software

Resumen Primer Parcial

UNIDAD 1 - INGENIERÍA EN SOFTWARE

- Crear soluciones cost-effective
- A problemas prácticos (concretos)
- Aplicando conocimiento científico / codificado
- Construyendo software que sirva al hombre

BODY OF KNOWLEDGE

Describe el conocimiento relevante para la disciplina.

SWEBoK

Áreas de conocimiento: Software Requirements, Software Design, Software Construction, Software Testing, Software Maintenance, Software Configuration Management, Software Quality, etc.

PROBLEMAS HABITUALES EN EL D&M DE SW

Problemas

- Se terminan con mucho atraso (**tiempo**)
- Se exceden en el costo estimado (**costo**)
- El producto final del proyecto no cumple con las expectativas del cliente (**alcance**)
- El producto final no cumple con los reqs de calidad mínimos (**calidad**)

Classic Mistakes

Problemas recurrentes en todos los proyectos de D&M de software. Suelen tener un gran impacto negativo en el desarrollo. Dada la alta probabilidad de que se cometan estos errores durante un proyecto, es importante conocerlos, junto con sus causas y consecuencias para tratar de evitar que sucedan o reaccionar de la mejor manera posible cuando, casi inevitablemente, sucedan.

1. Overly optimistic schedules (*cronogramas muy optimistas)*
2. Unrealistic expectations (*expectativas poco realistas)*
3. Excessive Multitasking (*multitasking excesivo)*
4. Shortchanged QA *(escatimar en aseguramiento de calidad)*
5. Wishful Thinking *(ilusiones)*

Causas

- No hay **administración y control** de proyectos
- El éxito depende de los **gurúes**
- Se **confunde** la solución con los requerimientos
- Las **estimaciones son empíricas** y no hay historia sobre proyectos realizados
- La calidad es una noción netamente **subjetiva** que no se puede medir
- No se consideran los **riesgos**
- Se asumen compromisos **imposibles** de cumplir
- Las actividades de **mantenimiento** van degradando el software
- Se comienzan los proyectos con requerimientos **no claros** ni estables

Reacciones

- Buscar al **gurú** que nos saque del problema
    - Dependencia
- **Agregar gente** a un proyecto atrasado (Lo atrasa aún más - *ley de Brook*)
    - Cada recurso nuevo requiere training y explicación de la situación del desarrollo. Esto hace que la nueva persona no sea productiva por un tiempo, mientras que le quita tiempo a los otros recursos.
    - Más gente trabajando en un proyecto significa que se pierde (exponencialmente) más tiempo en comunicación para la coordinación entre ellos.
    - Los proyectos SW no son fáciles de dividir para trabajar en paralelo.
- Recurrir a una **bala de plata**
    - Nuevo paradigma / herramienta / lenguaje
- **Menos documentación**
- **Menos testing**
- menos (cualquier actividad que no sea codificar)

DIAGRAMA CAUSA-EFECTO

Representación de varios elementos (causas) de un sistema que pueden contribuir a un problema (efecto).

- Útil para la recolección de datos
- Efectivo para estudiar procesos y situaciones

Usado para identificar las posibles **causas** de un problema específico.

UNIDAD 2 - MODELOS DE CALIDAD DE SOFTWARE

Costo de calidad: prevención y medición

Costo de no calidad

- costos visibles: fallas internas y externas
- costos invisibles
    - Baja motivación y desgaste de los equipos de trabajo
    - Re-trabajo constante ($$)
    - Imagen negativa ante el cliente

Los requerimientos de calidad deben establecerse **al inicio** junto con los reqs. del proyecto. Además se discute en cada paso del proyecto.

QUÉ ES LA CALIDAD

- Cumplir con los requerimientos de alguna persona
- Adecuación al uso
- Ausencia de deficiencias
- La totalidad de aspectos y características de un producto o servicio que se sustentan en su capacidad de cumplir las necesidades especificadas o implícitas

VISIONES DE LA CALIDAD

Cada interesado tiene una visión según su rol.

- **VISIÓN TRASCENDENTAL**
    - La calidad es algo que se puede reconocer pero no se puede definir (*ej: audi vs peugeot)*
- **VISIÓN DEL USUARIO**
    - La calidad es adecuación al propósito (*¿cuándo necesitaría el audi?)*
- **VISIÓN DE LA MANUFACTURA**
    - La calidad es conformidad con la especificación, cumplir con los requerimientos
        - Minimizar el retrabajo
        - Minimizar el desperdicio
- **VISIÓN DEL PRODUCTO**
    - La calidad está vinculada a las características inherentes del producto *(ej RAM)*
- **VISIÓN BASADA EN EL VALOR**
    - La calidad depende de la cantidad de **dinero** que el usuario está dispuesto a pagar por el producto

CALIDAD DEL PRODUCTO SOFTWARE (VISIÓN PRODUCTO - ISO 25010)

Adecuación Funcional

La capacidad para proporcionar funciones que **satisfacen las necesidades** declaradas e implícitas cuando el producto se usa en condiciones especificadas.

- **Completitud Funcional:** grado en el cual las funcionalidades **cubren** las tareas y objetivos del usuario.
- **Corrección Funcional**: proveer **resultados correctos** con el nivel de precisión requerido (las funcionalidades las cumple correctamente).
- **Pertinencia Funcional:** proporcionar un conjunto apropiado de funciones. (no hace cosas **innecesarias**).

Eficiencia de Desempeño

Desempeño relativo a la cantidad de **recursos utilizados** bajo determinadas condiciones.

- **Comportamiento temporal**: Los **tiempos de respuesta** y procesamiento y los **ratios de throughput** cuando lleva a cabo sus funciones (bajo condiciones determinadas) en relación con un benchmark establecido.
- **Utilización de recursos**: Las **cantidades y tipos** de recursos utilizados cuando lleva a cabo su función (bajo condiciones determinadas).
- **Capacidad**: Grado en que los **límites máximos** de un parámetro cumplen con los requisitos. *(ej: no alocar 500MB para un string)*

Compatibilidad

Capacidad de **dos o más sistemas** o componentes para **intercambiar información** y/o llevar a cabo sus funciones requeridas cuando comparten el mismo entorno hardware o software.

- **Coexistencia**: Capacidad del producto para coexistir con otro software independiente, en un entorno común, **compartiendo recursos** comunes sin detrimento.
- **Interoperabilidad**: Capacidad de dos o más sistemas o componentes para **intercambiar información** y utilizar la información intercambiada.

Usabilidad

Capacidad para ser **entendido, aprendido, usado** y **resultar atractivo** para el usuario (cuando se usa bajo determinadas condiciones).

- **Capacidad para reconocer su adecuación**: permite al **usuario** entender si el software es adecuado para sus necesidades.
- **Capacidad de aprendizaje**: permite al usuario **aprender** su aplicación.
- **Capacidad para ser usado**: permite al usuario operarlo y controlarlo con **facilidad**.
- **Protección contra errores de usuario**: proteger a los usuarios de hacer errores.
- **Estética de la interfaz de usuario**: Capacidad de la interfaz de usuario de agradar y satisfacer la **interacción** con el usuario.
- **Accesibilidad**: permite que sea utilizado por usuarios con determinadas características y **discapacidades**.

Fiabilidad

Capacidad para **desempeñar las funciones** especificadas (bajo unas condiciones y periodo de tiempo determinados).

- **Madurez:** satisfacer las necesidades de **fiabilidad** en condiciones normales *(ej: un producto cuando arranca tiene 80 errores, con el tiempo bajan. Un error no lo muestra como Victoria sino que lo muestra bien).*
- **Disponibilidad**: estar **operativo** y accesible para su uso cuando se **requiere**.
- **Tolerancia a fallos:** operar según lo previsto en presencia de fallos (***fail safe***).
- **Capacidad de recuperación:** recuperar los datos afectados y **restablecer el estado** deseado en caso de fallo.

Portabilidad

Capacidad de ser **transferido** de forma efectiva y eficiente de un entorno a otro.

- **Adaptabilidad**: permite ser adaptado de forma efectiva y eficiente a diferentes entornos.
- **Capacidad para ser instalado:** Facilidad con la que se puede instalar y/o desinstalar.
- **Capacidad para ser reemplazado:** ser utilizado en lugar de otro producto software con el mismo propósito y en el mismo entorno (*Ej: al instalar Chrome te importa todo solo)*.

CALIDAD DEL PROCESO (VISIÓN MANUFACTURA)

Visión de manufactura y de producto son las más medibles.

Premisa: manufactura de calidad => producto de calidad

MODELOS DE CALIDAD DE PROCESO

- CMMI
- ITIL
- ISO 15504 (SPICE)

CMMI (Capability Maturity Model Integrated)

Permite ir trabajando el el proceso de modo que se va volviendo cada vez más maduro.

Modelo para la mejora y evaluación de los procesos de: desarrollo, mantenimiento y operación de software.

- Determina la madurez de un proceso (5 niveles de madurez)
- Organiza el esfuerzo para mejorar el proceso describiendo un **camino incremental** de mejora

No es un proceso para desarrollar SW (**No es una metodología**): dice qué, no cómo ni quién. Por ejemplo, dice que “todo proyecto debe tener un plan”. Esto permite comparar para ver en qué nivel de madurez se encuentra el proyecto (si tiene o no un plan x ej).

Habla de dos cosas del proceso

- Capacidades
- Madurez

Proceso

Actividad para desarrollar y mantener software y sus productos asociados.

- Procedimientos y métodos
- Personas
    - habilidades, capacitación,motivación
- Herramientas
    - IDEs, herramientas de SCM, etc

Madurez

Capacidad organizacional de cumplir sistemáticamente con los objetivos.

Es el grado en que un proceso está:

- Proceso definido y documentado
- Administrado y controlado
- Medido y es efectivo

Proceso Maduro	Proceso Inmaduro
Soportado por la gerencia	La gerencia dice soportarlo...
Definido, documentado, conocido, y practicado	Improvisado sobre la marcha
Existe infraestructura adecuada para soportarlo	Aunque esté definido no se sigue rigurosamente
Adecuadamente medido	No hay entrenamiento formal ni herramientas para sustentarlo
Adecuadamente controlado	Presupuestos y plazos son generalmente excedidos por estimaciones no realistas
Presupuestos y plazos realistas	Es una organización "reactiva"
Riesgo conocido y controlado	
Proactivo	
Es como respirar… institucionalizado	

Capacidad de un Proceso

Habilidad inherente de un proceso de SW para **producir los resultados** planificados.
El CMMI se enfoca a que la organización produzca productos de alta calidad en forma **consistente** y **predecible**.

ITIL

Conjunto de **buenas prácticas** utilizadas para la gestión de servicios de tecnologías de información: "**Provisión y Soporte** de servicios de IT es lo más importante"

ISO 15504 (SPICE)

Estándar internacional de **evaluación** y determinación de la **capacidad** y **mejora continua** de procesos de ingeniería de SW.
Modela procesos para: gestionar, controlar, guiar y monitorear el desarrollo del software.

WHEN GOOD ENOUGH SOFTWARE IS BEST

Rápido - Barato - Bueno

- Balance: costo, tiempo, gente, funcionalidades, calidad
    - Elegido por el cliente / marketing. Mostrarle los trade offs de tocar cada uno.
- Ej: desarrollar un sistema completo, el cliente se las arregla con Excel (good enough).
- Tarde nunca es mejor
    - Cuando ya se está usando un software A, cuesta cambiarlo por otro B por más que sea mucho mejor. Conviene lanzar B antes, con menos funcionalidades/ más bugs, pero good enough.

UNIDAD 3 - SOFTWARE ENGINEERING APPROACHES

Gestionando en un contexto cambiante

PROYECTO

Esfuerzo temporal emprendido para crear un producto o servicio único para lograr un objetivo. Comprende una sucesión de fases en las que se coordinan recursos limitados.

- **Objetivo**: guía al proyecto
  - SMART
  - El **alcance** es la lista de funcionalidades que voy a tener que hacer para que ese software satisfaga el objetivo que dije.
- **Temporal**: tiene principio y fin
- **Único**: cada proyecto es diferente al otro

Project Management (disciplina): planificar, organizar, obtener y controlar recursos. Utiliza herramientas y técnicas para lograr que el proyecto alcance sus objetivos en tiempo y forma.

Un proyecto no es BAU (Business as usual).

Un proyecto software es intangible, no se degrada con el tiempo y engañosamente fácil de modificar.

DIMENSIONES DE UN PROYECTO SW

No se pueden cumplir todas a la vez.

- Funcionalidades
- Calidad
- Costo
- Tiempo
- Personas

Roles de las dimensiones

Lo importante no es ver qué dimensión tiene que rol, sino discutir qué prioridad tiene cada dimensión para saber cómo se pueden ajustar.

Según la mirada, una dimensión puede tener distintos roles.

- **Driver**: **objetivo vital** para el proyecto. Poca/nada flexibilidad

    - Funcionalidad
- **Constraint:** Factor limitante que el PM **no puede ajustar**. Poca/nada flexibilidad.
- **Grado de libertad:** Todo lo que no es constraint ni driver. Flexible.

Se puede usar un diagrama de Kiviat que el eje indique la flexibilidad de cada dimensión.

STANDING ON PRINCIPLE

- No dejarse convencer por el cliente de hacer un mal trabajo
- Buscar soluciones a las necesidades reales del usuario
    - Sus requerimientos pueden ser solucionados de otra forma
- No dejar que el cliente intervenga en los detalles técnicos/ de implementación
- Dar la verdadera información a los stakeholders
- Avisar si hay sobrecarga de tareas. Permitir (como PM) que te lo digan.
- Las estimaciones iniciales tienen un 80% de error. Avisarlo.

ROLES EN UN PROYECTO - STAKEHOLDERS

Stakeholders: Todos los involucrados al proyecto, pueden ser:

- **Sponsor**
    - Owner
    - Es el que tiene la **necesidad** que empuja que el proyecto salga adelante
- **Usuario Campeón**:
    - **Experto** en el dominio
    - Tiene que asegurar su disponibilidad y su **capacidad** para su función.
- **Usuarios Directos**
    - Los que interactúan directamente con el sistema
- **Usuarios Indirectos**
    - Usan el sistema pero no necesariamente lo operan (*Ej:* El área de Branding tiene que ver los íconos con la paleta de colores)

Tienen poder de decisión. Es importante tenerlos identificados.

CINEFYN

Modelo que compara características de cinco dominios de complejidad. Ayuda a reconocer los dominios de trabajo.

Entendiendo en qué dominio (contexto operativo) estamos podemos:

- determinar qué tipo de framework conviene utilizar
- tomar las decisiones adecuadas.

DOMINIOS CINEFYN

Simple y **Complicado:** gestión basada en hechos

- Relaciones de causa y efecto perceptibles.
- Respuestas correctas: en función de los hechos.

Complejo y **Caótico:** gestión basada en patrones

- No hay relación inmediatamente aparente entre causa y efecto
- El camino a seguir se determina según los **patrones emergentes**.

Simple

- Observar, categorizar (el hecho), responder
- Existe una **Best Practice** o una regla para el hecho.

Complicado

- Observar, analizar, responder
- Existen múltiples respuestas correctas, **Buenas Prácticas**.
- Para responder se requiere análisis o experiencia, para ver qué good practice aplicar.

Complejo

- Experimentar, observar, responder
- Existen **Emerging Practices**
- Se exploran soluciones (adaptativas). No es predecible el resultado.
- No hay experiencia en la industria.

Caótico

- Actuar, observar, responder
- No hay tiempo de armar una prueba concepto, se actúa directamente.
- Cualquier **Acción** es la respuesta apropiada
- Se actúa para transformar el dominio en complejo

METODOLOGÍAS ÁGILES

(No es aplicable a todos los problemas)

Principios

1. Mayor prioridad: **satisfacer al cliente**
2. **Aceptar** que los requisitos **cambien**
3. **Entregar** software funcional **frecuentemente**
4. Responsables de negocios, diseñadores y desarrolladores deben **trabajar juntos** día a día durante el proyecto
5. **Individuos motivados**
6. **Conversaciones cara a cara**
7. **Software funcionando**
8. Desarrollo sostenible
9. Atención continua a la excelencia técnica y al buen diseño
10. Simplicidad
11. Equipos auto-organizados
12. Ajuste de comportamiento para ser más efectivo

SCRUM FRAMEWORK

Marco metodológico de trabajo para la administración de proyectos. Enfatiza el **trabajo en equipo**.

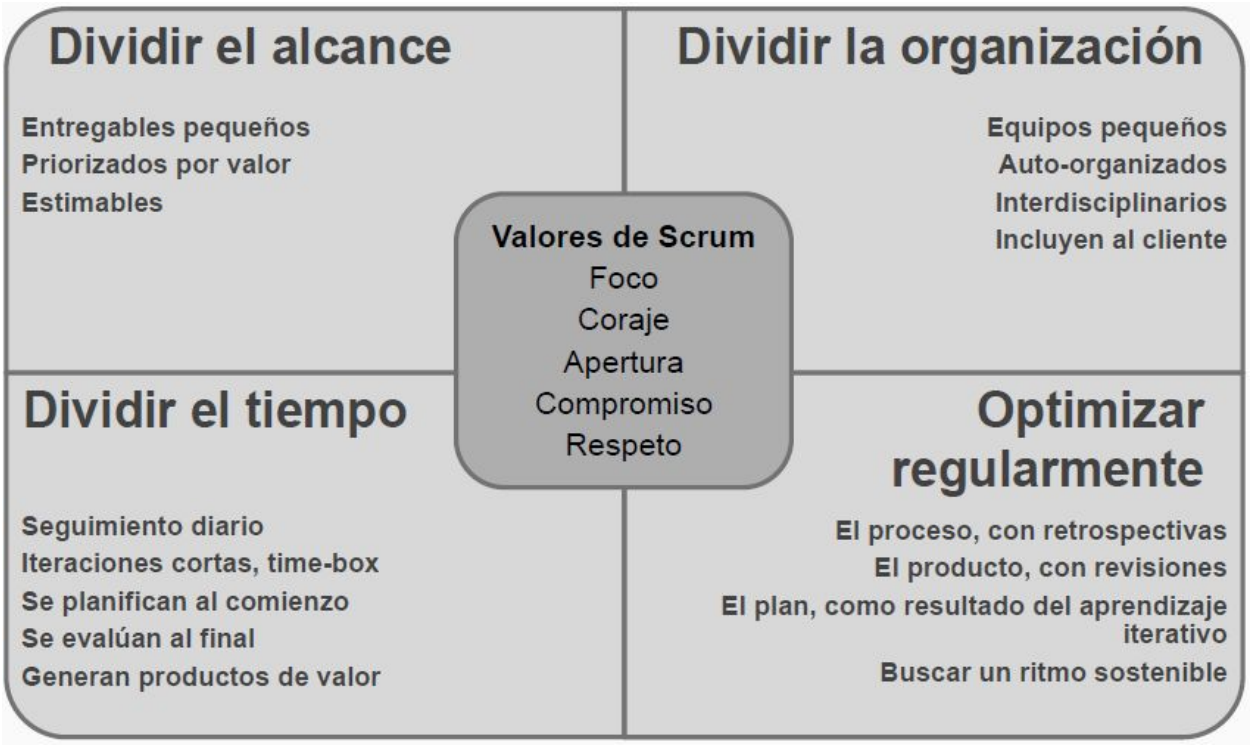
Pensado para construir productos de forma **incremental** hacia un **objetivo bien definido** en Sprints. Cada incremento es una versión mejorada del producto que alcanza criterios de aceptación y el nivel de calidad requerido.

Comenzar con lo que puede ser visto o conocido (MVP, **software funcionando**). Luego, seguir el avance y modificar lo necesario.

Pilares

Transparencia en el ciclo *Inspección<-->Adaptación*.
Este ciclo es del dominio caótico.

Valores



Roles

Una misma persona no puede cumplir todos los roles (podría darse, en partes).

Scrum Master

- Dueño del **proceso**
- Miembro activo del equipo
- Colabora con el equipo, **facilita** las ceremonias
- Elimina impedimentos
- Se asegura que **Scrum** se aplique **correctamente**
- **Cuida al equipo**: lo protege de pedidos e interrupciones externos
- Asegura la calidad de los **entregables**
- NO: asignar tareas el dev team, determinar qué entregar en el siguiente sprint, tomar decisiones en nombre del producto, cuidar el valor del producto

Idealmente desaparecería el Scrum master, con equipos autogestionados

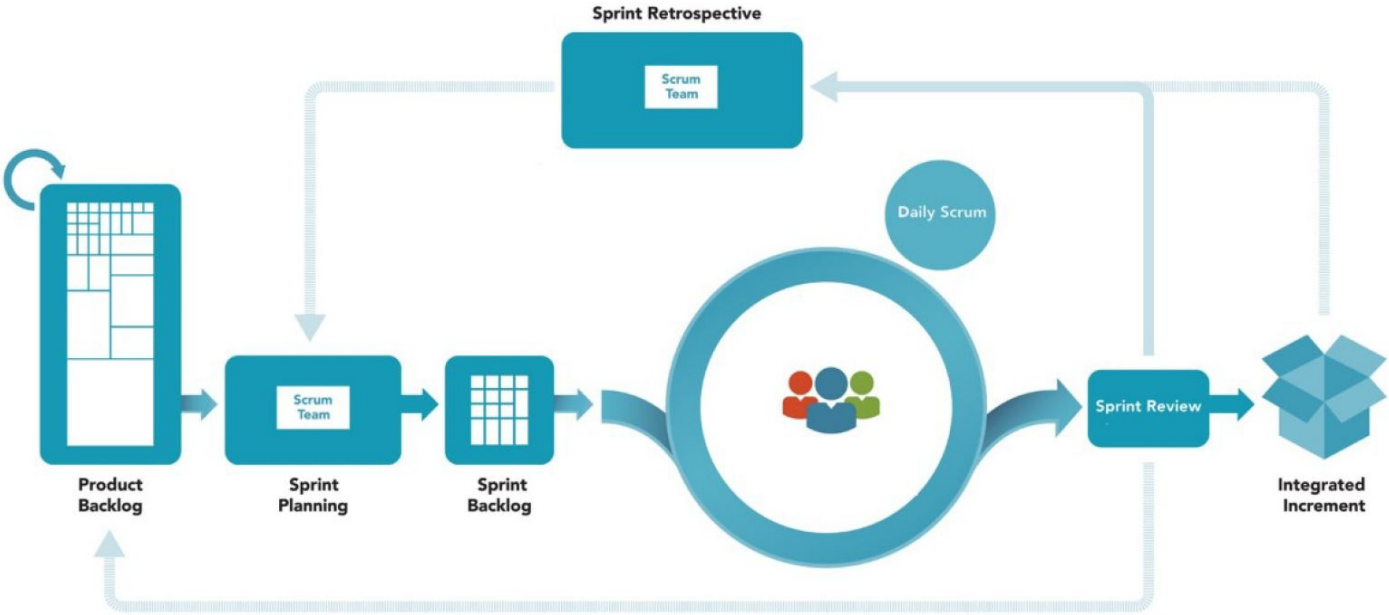
Product Owner

- Dueño de la definición de **terminado**
- Representa al **cliente,** stakeholders y sponsors.
- Miembro activo del equipo
- Dueño del **backlog**. Define las **prioridades**.
- Define el objetivo de cada sprint, establece el plan de entregas
- Optimiza y cuida el valor del **producto** (administra el ROI)
- NO: asignar tareas al dev team, determinar qué entregar en el siguiente sprint, asegurar la calidad de los entregables, facilitar las ceremonias

Equipo

- Dueño de los procesos de **producción**
- Auto organizado
- Define los roles internos del Scrum Team
- Asegurar la calidad de los **entregables**
- Define colaborativamente cómo transformar el Product Backlog en un **incremento de funcionalidad** al final de la siguiente iteración
- NO: cuidar el valor del producto, priorizar el backlog

Proceso



Product backlog: lista (ordenada según prioridades) de los requerimientos para un determinado aplicativo. Se va arreglando cada tanto.

- Ordenado por prioridad de negocio y estimación, no por valor de negocio
- Mantenerse actualizado con los nuevos requerimientos

No contiene todas las historias desde un principio. No todas las historias tienen que estar desarrolladas ni estimadas

Sprint Planning: De todo lo que está en el backlog, elijo qué voy a desarrollar en el próximo sprint.

- Participantes: Scrum Master, Dev Team, Product Owner
- Se estima el esfuerzo de las historias priorizadas (Planning Poker) y se auto asignan
- Se generan las tareas

No se prioriza el backlog, no se refinan las historias.

Sprint backlog: lo que voy a implementar en ese sprint en particular.

No se modifica al durante el Sprint. El PO no define qué historias se comprometen en el sprint (lo hace el equipo). El equipo no puede tomar historias según su especialidad (tienen que tomarlas indistintamente)

Sprint: iteración (entre 1 y 4 semanas) en la cual voy a implementar un incremento de producto (mi sprint backlog).

Daily meeting: ocurre al principio del día. Todo el equipo responde: qué hice ayer, qué voy a hacer hoy y qué problemas encuentro para encontrar mi objetivo (bloqueo).

- Participantes: Scrum Master, Dev Team, Product Owner, Stakeholders
- Se sincronizan las tareas, se detectan impedimentos
- Se revisa el Burndown chart.

No es catarsis, no se resuelven problemas. No se refinan historias. No se realiza en cualquier momento del día.

Sprint review: reviso si cumplí con los objetivos del Sprint, con el PO (y los stakeholders opcionalmente).

- El **Product Owner,** valida y aprueba que haya un incremento.
- El equipo tiene feedback temprano

No se revisa si las estimaciones fueron correctas (se hace en la retro).

Product increment: La salida del sprint. La suma de todos los elementos de sprint.

Existe un siempre al final de cada sprint. No necesariamente significa que va a ingresar código a producción. Podría ser x ej corrimos una prueba, probamos hacer algo (aprendimos, lo aprendido va a al backlog). Si tiene errores, también va a al backlog.

No es el conjunto de funcionalidades puesta en producción. No incluye al producto testeado.

Sprint retrospective: revisar la forma de trabajo del último sprint para la mejora continua. Qué salió bien/mal en el sprint, por qué, qué se podría hacer para mejorar el próximo.

- Participantes: SM, Dev Team, PO (?)
- Se obtienen las lecciones y acciones para mejorar el siguiente sprint

No se resuelven problemas, no se buscan culpables. No se planifica el próximo sprint. No se analiza la performance de los integrantes.

Refinamiento de las historias (reestimación de las historias del product backlog): se realiza de manera espontánea, cada vez que sea necesario.

KANBAN FRAMEWORK

Conceptos básicos

- Administración visual del flujo de trabajo
- Limita el trabajo en progreso de tareas en forma simultánea
- Realiza mediciones y optimiza el flujo de trabajo

Objetivo principal

Detectar cuellos de botella que representan estancamiento para avanzar. Evitar abrumación. Mejorar comunicación, evitando duplicación y retrabajo.

Aplicar cuando

- Proyectos inestables - domino **caótico** y **complejo**
- La prioridad de los requerimientos cambia constantemente
- Nos abrumamos de tareas
- Perdemos el foco cambiando de tarea
- Hay demasiado trabajo con poca productividad
- Hay problemas de comunicación en el equipo provocando esfuerzos duplicados, defectos, retrabajos, etc.

Métricas

- **Touch time:** tiempo neto de trabajo sobre una tarea determinada
- **Lead Time:** tiempo entre el momento de pedido y el momento de su entrega
- **Cycle Time:** tiempo entre el inicio y el final del proceso (para un ítem de trabajo)

LEAN FRAMEWORK

Conceptos básicos

- Mirada de manufactura
- Hincapié en la eliminación de residuos o de elementos que añaden no valor al trabajo
- Mejora continua
- Busca:
    - disminuir los tiempos del ciclo
    - disminuir costos

Filosofía

- Dinámica de mejora continua
    - resolución de problemas
    - aprendizaje organizacional continuo
- Respeto, desafío, selección y desarrollo de las personas
- Añadir valor y eliminar desperdicios
    - optimización de recursos
    - flujo de valor a cada cliente
    - afecta a los procesos
- Propósito a largo plazo

Desperdicios en proyectos software

- Entregar funcionalidad que el usuario no quiere/no aporta valor
- Corregir un bug
- Tiempo muerto

UNIDAD 4 - ESTIMACIONES DE PROYECTOS SOFTWARE

INTRODUCCIÓN

La primera pregunta al estimar tiene que ser ¿Cual es el **tamaño** de lo que tenemos que construir? Si no tenés el tamaño no podés decir cuánto te va a llevar.
El resultado de los métodos de estimación es el esfuerzo.

Requerimientos → Tamaño → Esfuerzo → Duración

Esfuerzo: horas hombre. La cantidad de tiempo que le lleva a una persona trabajando continuamente en esa tarea para completarla completamente. **Esfuerzo!=Progreso** (el ejemplo del viaje a la costa en 4 horas vs 10 horas)

Duración: tiempo calendarizado.

Tamaño → esfuerzo: La unidad de tamaño (m2, líneas de código, módulos) tiene una traducción a esfuerzo. La conversión depende de la **tecnología** y **metodología** que esté usando (la complejidad).

Esfuerzo→ duración: No es algo lineal. Las tareas tienen que poder ser paralelizables. También puede haber dependencias entre las tareas, o entre las personas (una misma persona tiene que hacer dos tareas). Esto hace que no sea una simple multiplicación.

Refinando en la metodología se puede reducir la duración.

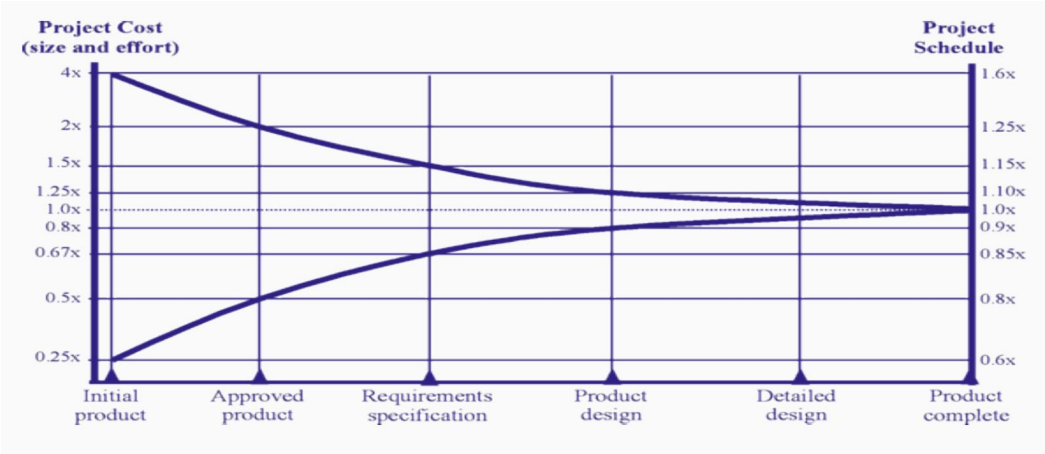
POR QUÉ FALLAN LAS ESTIMACIONES

- **Optimismo**
- Estimaciones **informales** (“estomacales”)
- No hay **historia**
- Mala definición del **alcance**
- Novedad / Falta de **Experiencia**
- **Complejidad** del problema a resolver
- No se estima el **tamaño**
- Porque la estimación fue buena pero cuando empieza el proyecto:
    - Mala **administración** de los requerimientos
    - No hay seguimiento y **control**

- Se confunde **progreso** con esfuerzo

NIVEL DE INCERTIDUMBRE

El “cono de la incertidumbre” muestra niveles de incertidumbre (probable) de las estimaciones en cada etapa del proyecto, respecto al costo (tamaño y esfuerzo) y a la duración.



No asumir que solo por el paso del tiempo y de las fases de un proyecto se avanza con menos incertidumbre en las estimaciones.

TIPS PARA ESTIMAR

- Diferenciar entre **estimaciones**, **objetivos** y **compromisos**
  - Los objetivos: (ver lo de grados de libertad)
- Asociar las estimaciones a un % de **confiabilidad** y con un **rango**
- Siempre presentar junto con la estimación los **supuestos** que se tuvieron en cuenta para llegar a la misma.
  - Si mi estimación **falla** puedo ver **por qué**
    - estaban **mal los supuestos**
    - **método incorrecto** de estimar.
- **Ley de parkinson**: toda tarea se expande hasta ocupar el tiempo que tiene asignado.
- Considerar todas las **actividades** relacionadas al desarrollo de software

ESTIMACIONES CREÍBLES

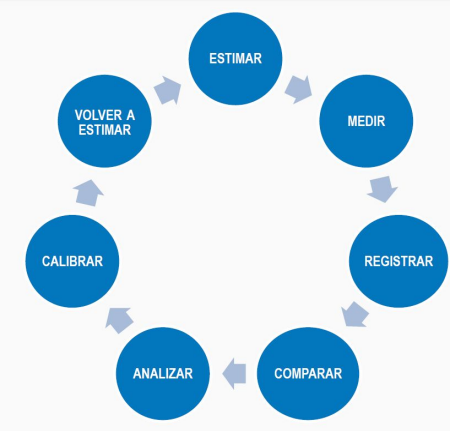
- Las estimaciones las hacen las **personas**, no herramientas ni modelos
  - Se necesitan **juicios** razonables y **compromisos** con los objetivos organizacionales que no pueden delegar a modelos automáticos.

- Las estimaciones se basan en **comparaciones**
    - Se necesita **historia** de proyectos pasados
- Si necesitamos pedir **prórrogas** cae nuestra **credibilidad**.
- Para una estimación correcta no se usa un solo método.

Para ser creíbles: que nuestro cliente sepa que tenemos un método de estimación, mostrarselo (que sea una **caja blanca**). En la medida que cumplamos con eso, ganamos confiabilidad por parte del cliente. Necesitamos también **nosotros confiar** en nuestro método.

CICLO DE ESTIMACIÓN

Para lograr un método de estimación **confiable**, lo ideal es seguir un ciclo para refinar el proceso de estimación, mejorarlo con el tiempo. (También sirve para armar un método desde 0).



Estimar: Comenzar por estimar el **tamaño** para derivar el esfuerzo y costo.
Tenemos que saber bien cuáles son las entradas (m2, líneas de código, módulos).

Medir: Mientras evoluciona el proyecto, medir el tamaño, el esfuerzo y el costo. Cuál fue el **verdadero valor** que nos dio lo que habíamos estimado.

Registrar: **anoto** las mediciones de forma clara.

Analizar: **Razones** de desvíos, supuestos que variaron, temas no contemplados

Calibrar: Ajustar cada una de las variables y parámetros que intervienen en el proceso de medición. “en realidad no tuvimos en cuenta que requería un entorno cloud”, entonces la próxima sabemos que si se necesita un entorno cloud lo vamos a tener en cuenta para la estimación.

Volver a estimar: el mismo proyecto (ahora con más info). Nuevos proyectos, con la calibración. Independientemente del método que usemos, vamos a tener un método bueno; **nosotros vamos a creer** en los métodos.

MÉTODOS RUDIMENTARIOS DE ESTIMACIÓN

Basados principalmente en la experiencia e intuición

Juicio de Expertos

Basado en experiencia e intuición

Pert

Un juicio experto más elaborado. Trabaja con el más probable, el pesimista y el optimista.

Wideband Delphi

Estimación estomacal pero consensuada. Siguiendo un proceso definido se va refinando la estimación consensuada por los expertos.

Planning Poker

Componentes

- **Product Backlog** priorizado
- **Dev Team** (todos los que trabajan en el producto durante el sprint)
- **Moderador**
- **PO**

Procedimiento

1. Para la elección de los rangos, se tiene una tarea modelo (“tarea pivote”) de referencia, que todos masomenos conocen.
2. Elegir User Story del **Product Backlog** (empezando por la de más alta prioridad).
    a. El PO explica el US, se hacen preguntas.
3. Todos eligen un **número para la estimación**.
    a. Si hay **discordancia**, se averiguan **por qué** en los outliers. Se vuelve a elegir un número para la estimación hasta que todos tengan el mismo número.
4. Se repite el procedimiento en el Sprint backlog (para cada tarea de cada historia del sprint backlog)

El **tamaño** se mide en **Story Points**.
Participan todos, da **sentido de pertenencia** a las tareas.
No está bueno para proyectos sin historia xq no vas a tener con qué comparar. Tampoco para un proyecto corto.

MÉTODOS PARAMÉTRICOS DE ESTIMACIÓN

Tienen inputs llamados “parámetros” que nosotros procesamos, le agregamos complejidad y con eso definimos el tamaño de lo que tenemos que estimar.

Function points

Entrada: **componentes** según sean:

- Transaccionales
    - External Inputs (EI)
    - External Outputs (EO)
    - External Enquiries (EQ)
- De datos
    - Internal Logical Files (ILF)
    - External Interface Files (EIF)

Salida**: function points**

Proceso

1. Determinar los **componentes** del producto
2. **Categorizar** dichos componentes
3. Asignar **niveles de complejidad**
    a. según la **cantidad de dato**s que hay que ingresar
4. Calcular **Function Points** NO ajustados (UFP)
5. Calcular el **factor de ajuste** de valor (VAF)
6. Obtener los **Function Points netos** (NFP)

Es independiente a las tecnologías a utilizar.

Para el ajuste, se consideran factores del **entorno**, cada uno según el grado de influencia:

- Data communications
- Performance
- Heavily used configuration
- Transaction rate
- Complex processing
- Reusability
- Installation ease
- Operations ease

- Online data entry
- End user efficiency
- Online update
- Multiple sites
- Facilitate change
- Distributed functions

Lleva gran cantidad de tiempo. Es difícil de aprender y perfeccionar.

Use case points

Entrada: **Casos de uso**
Salida: **Use case points** (o esfuerzo: UCP * CF)

Factores ambientales: bajan la estimación
Factores tecnológicos: suben la estimación

El número de Use Case Points de un **producto** depende de:

- La **cantidad** de casos de uso
- La **complejidad** de dichos casos, según su cantidad de **transacciones**
- Los **actores** que intervienen
- Factores **técnicos** y **ambientales**

Proceso

1. Clasificar **actores**
2. Clasificar **casos de uso** (según complejidad)
3. Hallar
    a. Peso no ajustado de los **actores** (UAW)
    b. Peso no ajustado de **casos de uso** (UUCW)
    c. **Use Case Points** no ajustados (UUCP)
    d. Factores de **ajuste** (Technical Complexity y Environmental)
    e. Use **Case Points netos** (UCP)

Ojo: Pueden cambiar mucho las estimaciones xq ante mayor detalle del use case, más transacciones (más use case points). Hay estandarizar el nivel de detalle.
La **especificación** debe **incluir** casos de uso bien definidos.

Story points

No es un método sino una unidad de medición que se usa en los métodos.

Velocidad del equipo: Cuántos story points meto en el sprint, depende de la velocidad de mi equipo.

En un sprint meto por ejemplo una tarea de 5, una de 8 y una de 2 story points,¿

Requerimientos no funcionales: considerarlos. Por ejemplo para cuántos alumnos armar el SIGA.

Definición de terminado: tener en cuenta testing? aceptación del usuario? en producción?

Object points

Entrada: **Objetos**
Salida: **Object Points (ajustados)**

Objetos: pantallas, reportes y módulos (y su complejidad)

- Son objetos concretos con una tecnología concreta que ya conozco
- No está relacionado necesariamente con objetos de OOP
- El método original contempla solo estos tres tipos de objetos. Se pueden otros (stored procedures, clases, scripts SQL, etc ...)
- Usado para proyectos de mantenimiento, donde ya conozco los componentes que voy a estar utilizando

Proceso

1. Se asigna a cada **componente** un **peso** de acuerdo a su clasificación por **complejidad** en simple, medio o difícil.

	Number and sources of data tables		
Number of Views Contained	Total < 4	Total < 8	Total 8+
<3	simple	simple	medium
3-7	simple	medium	difficult
8+	medium	difficult	difficult

2. Se le brinda un **peso** a cada **nivel de complejidad** (teniendo directa proporción con el **esfuerzo** que requiere la implementación de c/u)

	Weight		
Type	**Simple**	**Medium**	**Difficult**
Screen	1	2	3
Report	2	5	7
Modules	10	10	10

3. Sumando la cantidad de objetos de cada complejidad (usando sus pesos según la tabla) da como **resultado** los **Object Points**
4. Considera como factor de **ajuste** el **porcentaje de reuso** de código

TIMEBOX DEVELOPMENT

Forma de establecer las duraciones, teniendo en cuenta la **ley de Parkinson**.

- Urgencia/**presión** de deadline (**motivación**) para los devs
- Enfoque en los **MVPs**
- **Redefinición** del producto
- Usuario: menos funciones pero menor tiempo

Riesgos:

- Aplicarlo en productos en los que no se puede aplicar
- **Sacrificar calidad** en lugar de funciones

Cuándo aplicar:

- Requerimientos bien detallados
- Preferentemente no aplicarlo en la etapa de análisis

Requerimientos:

- Priorizar
- Que no se sobrepase el deadline

Fijo el tiempo, no descuido la calidad y ajusto las otras dimensiones (con la funcionalidad principalmente).

Ejemplo (Oscar): se hacían ciclos de pruebas y era un proceso caótico. De todos los issues, el usuario ordena por prioritario. Estuvo disponible. “Bueno para la semana que viene metemos esto”, etc. Fueron metiendo todos los issues en una semana cada uno. Se probaban las features, el usuario lo aprobaba y se iban al siguiente. Así se estabilizó el producto para pasarlo a producción.