Primer Parcial – Ingeniería de Software - 1C2022

Unidad 1: *"Introducción a la Ingeniería de Software"*

¿Qué es la Ingeniería de Software?

- La aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento de software.
- Crear soluciones cost-effective.
- Resolver problemas prácticos (concretos) al servicio del hombre.
- Aplica conocimiento científico / codificado.

Body of Knowledge

- "**Cuerpo de Conocimiento**" es un requisito para calificar cualquier área de la ingeniería como una profesión y describe el conocimiento relevante para una disciplina.
- **SWEBoK – "Software Engineering Body of Knowledge"**
    - Describe el conocimiento que existe de la Ingeniería de Software.
    - Conformado por 15 Áreas de Conocimiento que sintetizan conceptos básicos y referencian a información más detallada.
    - **Áreas de conocimiento**: Software Requirements, Software Design, Software Construction, Software Testing, Software Maintenance, Software Configuration Management, Software Quality, entre otros.

Problemas Habituales en el D&M de Software

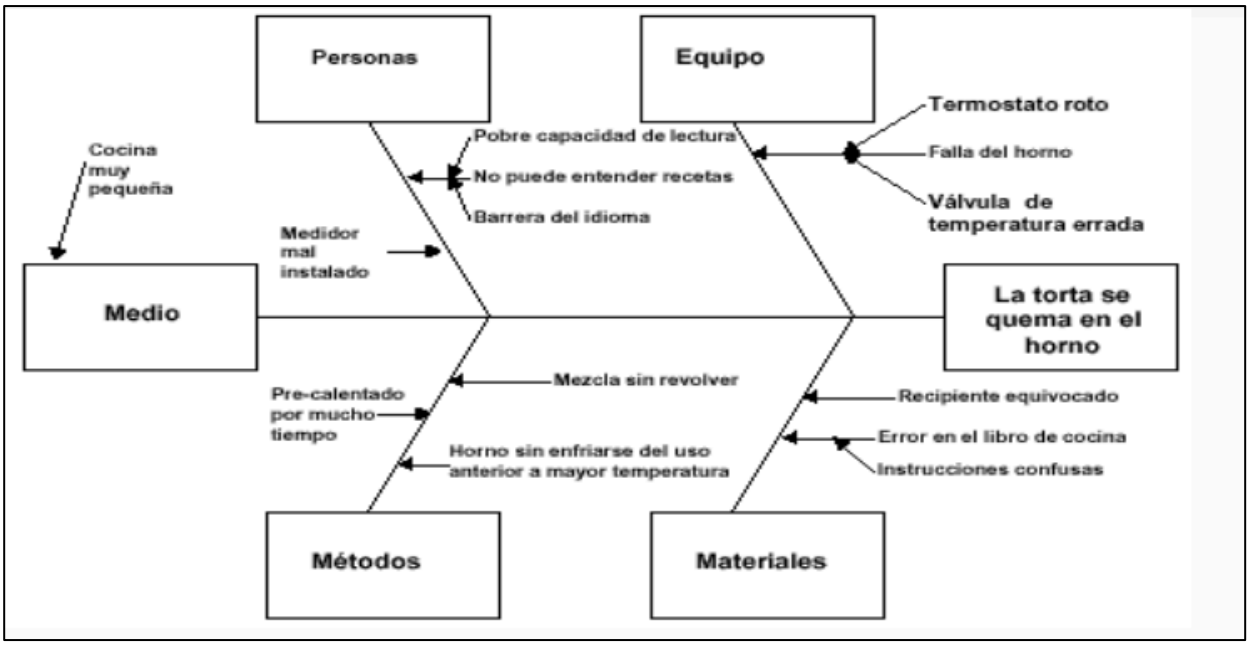
- **Problemas Habituales**:
    - **Tiempo**: los proyectos terminan con mucho atraso.
    - **Costo**: los proyectos se exceden del costo estimado.
    - **Alcance**: el producto final no cumple las expectativas del cliente.
    - **Calidad**: el producto final no cumple con los requisitos de calidad mínimos esperados.

- **Causas Comunes**:
    - **Administración y Control**: no hay administración y control de proyectos.
    - **Gurúes**: el éxito depende de los gurúes.
    - **Confusión**: se confunde la solución con los requerimientos.
    - **Tiempo**: las estimaciones son empíricas y no hay historia sobre proyectos realizados.
    - **Calidad**: la calidad es una noción netamente subjetiva que no se puede medir.
    - **Riesgos**: no se consideran los riesgos.
    - **Compromisos**: se asumen compromisos imposibles de cumplir.
    - **Mantenimiento**: las actividades de mantenimiento van degradando el software.
    - **Requerimientos**: se comienzan proyectos con requerimientos no claros ni estables.
- **Reacciones mas Frecuentes**:
    - **Dependencia**: buscar al gurú que nos saque del problema.
    - **Ley de Brook – Agregar Gente**:
        - La ley dice que agregar más gente a un proyecto atrasado lo atrasa aún más.
        - Cada recurso humano nuevo requiere training y explicación de lo que se esta haciendo. Nueva persona no es productiva hasta cierto tiempo.
        - Mas gente indica más tiempo perdido en reuniones de coordinación y comunicación.
        - Proyectos de SW no son fáciles de dividir para trabajar en paralelo.
    - **Bala de Plata**: recurrir a una “bala de plata” que no salve el problema. Nuevo paradigma / nueva herramienta / nuevo lenguaje.
    - **Asumir Definiciones**: asumir definiciones en lugar del usuario.
    - **Reducir / Eliminar la documentación**.
    - **Reducir / Eliminar el testing**.
    - **Reducir / Eliminar cualquier actividad que no sea codificar**.
- Además, se realiza un **mal análisis de la situación**, lo que lleva a padecer los mismos problemas y cometer los mismos errores ya que no se logra entender el problema.

Diagrama Causa – Efecto / Diagrama de Ishikawa / Espina de Pescado

- Representación de varios elementos (**causas**) de un sistema que pueden contribuir al problema (**efecto**).
- Es útil para la recolección de datos y es efectivo para estudiar los procesos y situaciones.
- Un diagrama causa efecto bien preparado es un vehículo para ayudar a los equipos de mejora continua a tener una concepción común de un problema complejo.

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



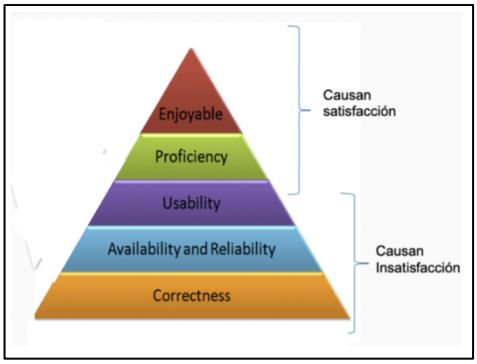
Paper – Classic Mistakes

- **Classic Mistakes**: errores que se cometen tan seguido y por tanta gente que las consecuencias de cometerlos deberían ser predecibles y los mismos errores deberían ser evitables.
- Problemas recurrentes en todos los proyectos de D&M de software.
- Suelen tener un gran impacto negativo en el desarrollo.
- Conocerlos (sus **causas y consecuencias**) para evitar que sucedan o reaccionar de la mejor manera posible ante su ocurrencia.
- **Classic Mistakes**:
    - Cronogramas muy optimistas – Overly optimistic schedules.
    - Expectativas poco realistas – Unrealisitc Expectations.
    - Multitasking Excesivo – Excessive Multitasking.
    - Escatimar en QA – Shortchanged QA.
    - Ilusiones – Whisful Thinking.

Unidad 2: *"Calidad de Software"*

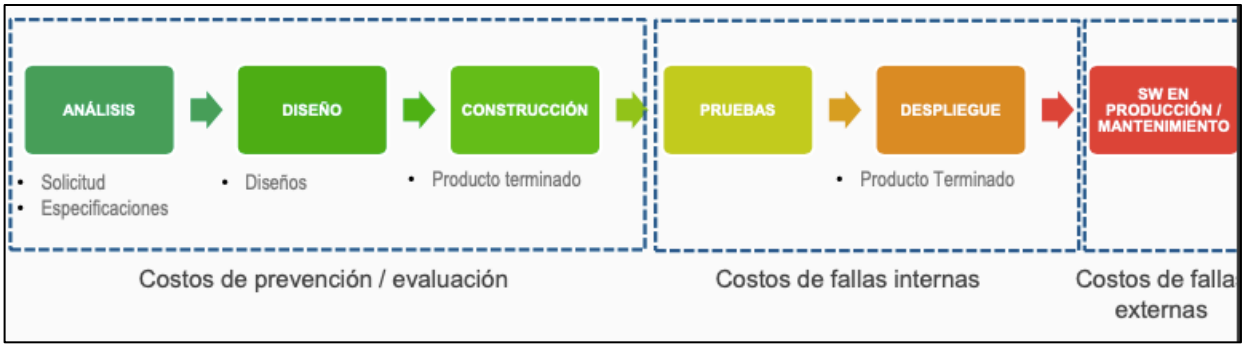
¿Qué es la Calidad?

- Existen múltiples definiciones sobre que es la calidad. Se considera una dimensión sobre la que trabajar.
- Cumplir con los requerimientos (requisitos del usuario) de manera correcta.
- Adecuación al uso.
- Ausencia de deficiencias. Agregado de valor.
- La totalidad de aspectos y características de un producto o servicio que se sustentan en su capacidad de cumplir las necesidades especificadas o implícitas.
- **Pirámide de Jerarquía de Factores de Calidad**: Insatisfacción (deben cubrirse) – Satisfacción (agregan)



Fundamentos de Calidad – Costo de la NO Calidad

- Los requerimientos de calidad deben establecerse al inicio junto con los requerimientos del proyecto. Además, se discuten en cada paso del proyecto.
- **Costo de Calidad**: prevención y medición.
- **Costo de No Calidad**:
  - **Visibles**: fallas internas y externas que afectan al cliente.
  - **No Visibles**:
    - **Duplicación de Esfuerzos**: baja motivación de los equipos de trabajo.
    - **Mayor Costo**: re-trabajo es constante.
    - **Desgaste del equipo de trabajo**.
    - **Imagen negativa ante el cliente**.
- Diagrama:



Visiones de la Calidad

- Cada interesado tiene una visión según su rol. La calidad puede ser percibida desde cinco perspectivas.
    - **Visión Trascendental**: la calidad es algo que se puede reconocer, pero no se puede definir. (*Ejemplo: Audi vs Renault*).
    - **Visión del Usuario**: la calidad es adecuación al propósito. (*Ejemplo: ¿Cuándo necesito un Audi?*).
    - **Visión de la Manufactura**: la calidad es conformidad con la especificación. Cumplir con los requerimientos.
        - Minimizar el re-trabajo.
        - Minimizar el desperdicio.
    - **Visión del Producto**: la calidad esta vinculada a las características inherentes del producto. (*Ejemplo: Dodge RAM*).
    - **Visión Basada en el Valor**: la calidad depende de la cantidad de dinero que el usuario esté dispuesto a apagar por el producto.

Calidad del Producto de Software (Visión del Producto – ISO 25010)

- **ISO 25000**: familia de normas que tiene por objetivo la creación de un marco de trabajo común para evaluar productos de software.
- **Diagrama Calidad del Producto de Software**:

Adecuación Funcional

- La capacidad para proporcionar funciones que satisfacen las necesidades declaradas e implícitas, cuando el producto se usa en las condiciones especificadas.
    - **Completitud Funcional**: grado en el cual las funcionalidades cubren todas las tareas y los objetivos del usuario.
    - **Corrección Funcional**: capacidad del producto para proveer resultados correctos con el nivel de precisión requerido. (*Funcionalidades las cumple correctamente*).
    - **Pertinencia Funcional**: capacidad del producto para proporcionar un conjunto apropiado de funciones. (*No hace cosas innecesarias*).

Eficiencia de Desempeño

- Representa el desempeño relativo a la cantidad de recursos utilizados bajo determinadas condiciones.
    - **Comportamiento Temporal**: tiempos de respuesta y procesamiento del sistema en comparación a un conjunto de datos de pruebas preestablecido.
    - **Utilización de Recursos**: cantidades y tipos de recursos utilizados cuando lleva a cabo su función.
    - **Capacidad**: grado en que los límites máximos de un parámetro cumplen con los requisitos. (*Ejemplo: no alocar 500MB para un String*).

Compatibilidad

- Capacidad de dos o más sistemas o componentes para intercambiar información y/0 llevar a cabo sus funciones requeridas cuando comparten el mismo entorno hardware o software.
    - **Coexistencia**: capacidad para coexistir con otro software independiente, en un entorno común, compartiendo recursos comunes sin detrimento.
    - **Interoperabilidad**: capacidad de dos o más sistemas para intercambiar información y utilizarla.

Usabilidad

- Capacidad del producto 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 del Usuario**: capacidad de agrandar y satisfacer la interacción con el usuario.
- **Accesibilidad**: capacidad que permite ser utilizado por usuarios con determinadas características y discapacidades.

Fiabilidad

- Capacidad de un sistema o componente para desempeñar las funciones especificadas, cuando se una bajo unas condiciones y periodo de tiempo determinados.
  - **Madurez**: capacidad de satisfacer las necesidades de fiabilidad en condiciones normales. (*Ejemplo: producto disminuye errores con el paso del tiempo*).
  - **Disponibilidad**: capacidad de estar operativo y accesible para su uso cuando se requiere.
  - **Tolerancia a Fallos**: capacidad para operar según lo previsto en presencia de fallos de hardware o software. (*Ejemplo: Fail-Safe*).
  - **Capacidad de Recuperación**: capacidad de recuperar los datos afectados y restablecer el estado deseado del sistema en caso de falla.

Seguridad

- Capacidad de protección de la información y los datos de manera que personas o sistemas no autorizados no puedan leerlos o modificarlos.
  - **Confidencialidad**: capacidad de protección contra acceso de datos e información no autorizados.
  - **Integridad**: capacidad para prevenir accesos o modificaciones no autorizados a datos o programas.
  - **No Repudio**: capacidad de demostrar las acciones o eventos que han tenido lugar, de manera que no puedan ser repudiados posteriormente.
  - **Responsabilidad**: capacidad de rastrear de forma inequívoca las acciones de una entidad.
  - **Autenticidad**: capacidad de demostrar la identidad de un sujeto o recurso.

Mantenibilidad

- Característica que representa la capacidad del producto para ser modificado efectiva y eficientemente, debido a necesidades evolutivas, correctivas o perfectivas.
    - **Modularidad**: capacidad que permite que un cambio en un componente tenga un impacto mínimo en los demás.
    - **Reusabilidad**: capacidad que permite que sea utilizado en más de un sistema de software.
    - **Analizabilidad**: facilidad que se puede evaluar el impacto de un determinado cambio sobre el resto del software, o identificar las partes a modificar.
    - **Capacidad para ser Modificado**: capacidad que permite ser modificado de forma efectiva y eficiente sin introducir defectos o degradar el desempeño.
    - **Capacidad para ser Probado**: facilidad con la que se pueden establecer criterios de prueba para un sistema y con la que se pueden llevar a cabo pruebas para determinar si cumplen.

Portabilidad

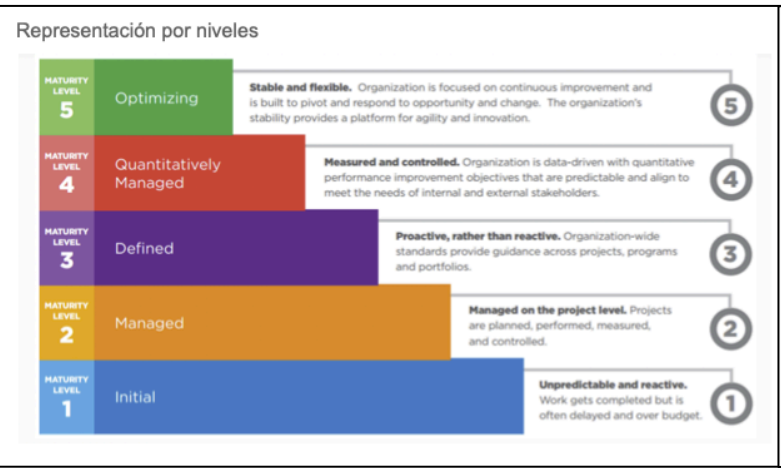
- Capacidad del producto de ser transferido de forma efectiva y eficiente de un entorno de hardware, software, operacional o de utilización a otro.
    - **Adaptabilidad**: capacidad del producto que le permite ser adaptado de forma efectiva y eficiente a diferentes entornos determinados.
    - **Capacidad para ser Instalado**: facilidad con la que se puede instalar y/o desinstalar de forma exitosa.
    - **Capacidad para ser Reemplazado**: capacidad para ser utilizado en lugar de otro producto determinado con el mismo propósito y en el mismo entorno.

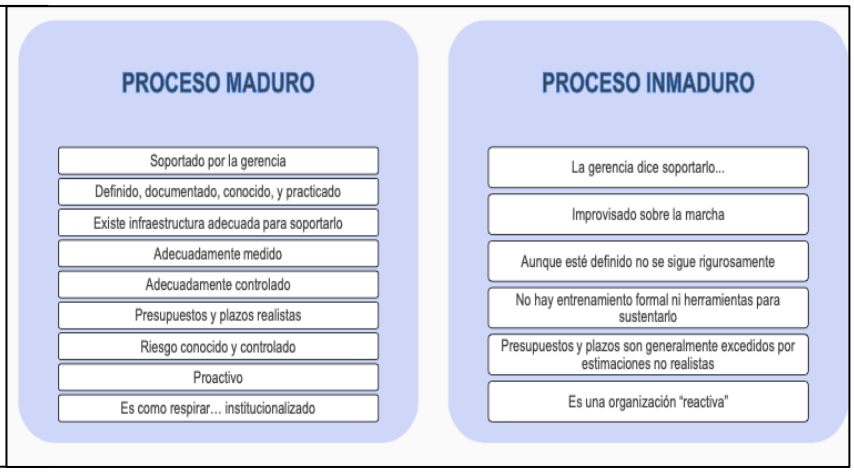
Calidad de Proceso

- **Proceso**: conjunto de actividades que la gente usa para desarrollar y mantener software y sus productos asociados.
- **Madurez**: capacidad organizacional, no individual, de cumplir sistemáticamente los objetivos.
- **Capacidad de un Proceso**: habilidad inherente de un proceso para producir resultados planificados.
- Para evaluar la calidad de un proceso, lo hacemos en función de ver cuál es el proceso que intento seguir, y ver realmente si ocurre. Tratar de buscar contra que me voy a comparar.
- Existen 3 modelos de calidad de proceso: CMMI, ITIL y SPICE -> Framework que me ayuda a seguir sus pasos y posiblemente ser exitoso.

CMMI (Capability Maturity Model Integrated)

- Modelo para la mejora y evaluación de los procesos de desarrollo, mantenimiento y operación de software. Es un **framework de evaluación y trabajo. No una metodología**.
- **Determina la madurez de un proceso y organiza el esfuerzo para mejorarlo describiendo un camino incremental de mejora**.
- Evalúa los puntos clave de acuerdo con un puntaje de 5 niveles de madurez. Se asciende de nivel si se cumplen con los requisitos. (*1. Caos – 2. Prácticas de Gestión – 3. Definidos – 4. Avanzado / Manageado – 5. Optimizado*)
- Habla de dos cosas del proceso: **Capacidades y Madurez**.
- Diagramas:





ITIL

- Conjunto de buenas prácticas utilizadas para la gestión de servicios de tecnologías de información.
- Provisión y Soporte de servicios IT es lo más importante.

SPICE (ISO 15504)

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

Paper – "When Good Enough Software Is Best"

- **Balance entre costo, tiempo, recursos, funcionalidades y calidad (5 ejes fundamentales)**. Esto debe ser determinado por el cliente y nosotros debemos limitarnos a exhibir y mostrar ventajas y desventajas de cada uno. (Ejemplo: calculadora software vs calculadora de mano).
- **Tarde nunca es mejor**. Cuando ya se está usando un software es muy difícil cambiarlo por otro por más que este sea mucho mejor. Conviene lanzar software mejor antes con menor cantidad de funcionalidades pero que sea "**Good enough**".
- **Rápido, bueno y barato nunca pueden ir juntas**.

Unidad 3: "*Software Engineering Approaches*"

Proyecto

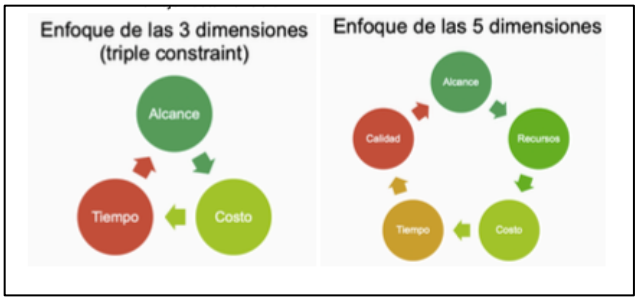
- Es un esfuerzo temporal emprendido para crear un producto o servicio único para lograr un objetivo. Se caracteriza por el contexto y variables únicas de cada proyecto.
- **Tareas no rutinarias. Tienen objetivo y tiempo finito (inicio y fin).**
- **Características**:
    - Tiene un **objetivo** o beneficio a obtener que guía el proyecto.
    - **Temporal**: porque tiene principio y fin.
    - **Único**: no es recurrente, cada proyecto es diferente al otro. Siempre contexto y variables son distintas.
    - Posee **recursos limitados**.
    - Consta de **sucesión de actividades** o fases en las que se coordinan los distintos recursos.

Project Management

- Es una disciplina que consiste en planificar, organizar, obtener y controlar recursos, utilizando herramientas y técnicas para lograr que el proyecto logre sus objetivos en tiempo y forma.

Dimensiones de un Proyecto de Software

- No se puede cumplir con todas a la vez. Determinar prioridades en la dimensión para saber cómo se pueden ajustar entre sí.
- Existen dos enfoques de 3 y 5 dimensiones:
    - **Alcance**
    - **Costo**
    - **Tiempo**
    - Calidad
    - Recursos



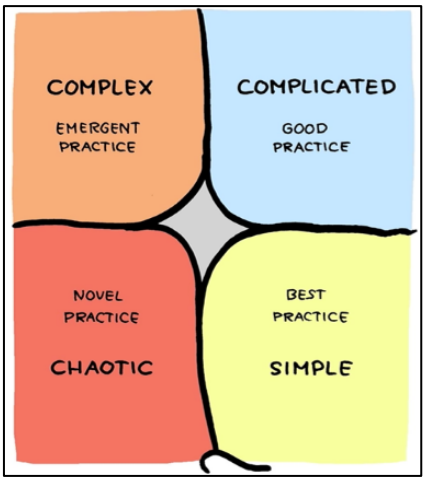
- Las **dimensiones** pueden ser consideradas como:
    - **Driver**: objetivo vital a lograr. NO tiene flexibilidad.
    - **Restricción**: no está bajo nuestro control directo y también NO tiene flexibilidad.
    - **Grado de Libertad**: es la libertad para fijar objetivos. Existe flexibilidad para manejar esta variable.

Roles Principales de un Proyecto

- **Stakeholders**:
  - Son todos los involucrados por el proyecto.
  - Tienen poder de decisión con capacidad de influir en la marcha del proyecto.
- **Sponsor**:
  - Es el "owner" (dueño) del proyecto. Quien pone el dinero y toma decisiones principales.
  - Es quien tiene la autoridad para llevar adelante el proyecto.
- **Usuario Campeón**:
  - Experto en el dominio del problema del proyecto. Es el que sabe todo. Es muy demandado.
  - Asegurar su capacidad para la función y su disponibilidad.
- **Usuarios Directos**:
  - Los que interactúan directamente con el sistema. Quienes usan al sistema.
- **Usuarios Indirectos**:
  - Hacen uso del sistema, pero no necesariamente son los que lo operan. (*Ejemplo: área de finanzas que se beneficia con el reporte que se genera*).

CYNEFIN

- Modelo que compara características de cinco dominios de complejidad.
- Ayuda a reconocer los dominios de trabajo.
- Entendiendo en que dominio (contexto operativo) estamos, podemos:
  - Determinar qué tipo de framework de trabajo conviene utilizar.
  - Tomas las decisiones más adecuadas.



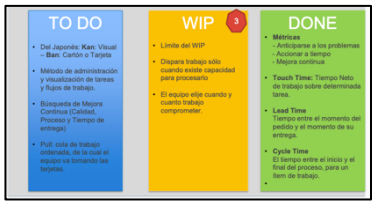
- **Contexto simple y complicado**: universo ordenado, relaciones de causa y efecto son perceptibles y las respuestas correctas se pueden determinar en función de los hechos.
- **Contexto complejo y caótico**: universo desordenado, no hay una relación inmediata aparente entre causa y efecto, y el camino a seguir se determina según patrones emergentes.
- **Universo Ordenado -> Gestión** basada en **hechos**. **Universo Desordenado** -> **Gestión** basada en **patrones**.

- **Contexto Simple:**
    - Situación es estable y existen reglas probadas para aplicar.
    - Relación entre causa y efecto es clara: si se realiza X, se espera Y.
    - En este dominio se establecen los hechos, se categorizan y se responde con una regla o se aplica una **mejor practica**.
- **Contexto Complicado:**
    - Relación entre causa y efecto requiere análisis o experiencia, ya que existen múltiples respuestas correctas.
    - **Buenas prácticas** son métodos que se pueden aplicar según decisión de un experto. Se requiere juicio refinado y con experiencia.
    - Se evalúan los hechos, se analizan y se aplica una **buena práctica**.
- **Contexto Complejo:**
    - Relación causa y efecto solo puede ser decida en retrospectiva. **No hay respuestas correctas. Se deben explorar soluciones adaptadas y no es predecible el resultado.**
    - No hay experiencia en la industria.
    - Existen **Emerging Practices**.
    - Experimentar, observar y responder.
- **Contexto Caótico:**
    - Actuar, observar y responder.
    - Relación entre causa y efecto es incierta. Los eventos son muy confusos para esperar una respuesta basada en el conocimiento.
    - **Cualquier acción es la respuesta apropiada. Nos encontramos en crisis y lo importante es actuar.**
    - Se actúa para **transformar el dominio en complejo.**

Enfoques para encarar un Proyecto de Software

KANBAN (Tablero Visual)

- **Es una administración visual de flujo de trabajo que ayuda a realizar más con menos estrés**. Busca mejora continua (calidad, proceso y tiempo de entrega).
- **Objetivo**: detectar cuellos de botella que representan estancamiento para avanzar. **Evita abrumacion, mejora comunicación y evita duplicación y retrabajo.**
- **Conceptos Básicos**:
    - Visualizar el flujo de trabajo.
    - Limitar el trabajo en progreso, es decir, trabajo de forma simultánea.
    - Realizar mediciones y optimizar el flujo.
- **¿Cuándo uso Kanban?**
    - Los proyectos sean inestables, tenemos muchos cambios. (Proyectos Caóticos o Complejos)
    - Los requerimientos estén cambiando constantemente de prioridad.
    - Entornos de resolución de incidencias.
- **Tablero** (Columnas)
    - TO-DO
    - WIP
    - DONE
    - (No columna) PULL: cola de trabajo ordenada de donde se toman las tareas.

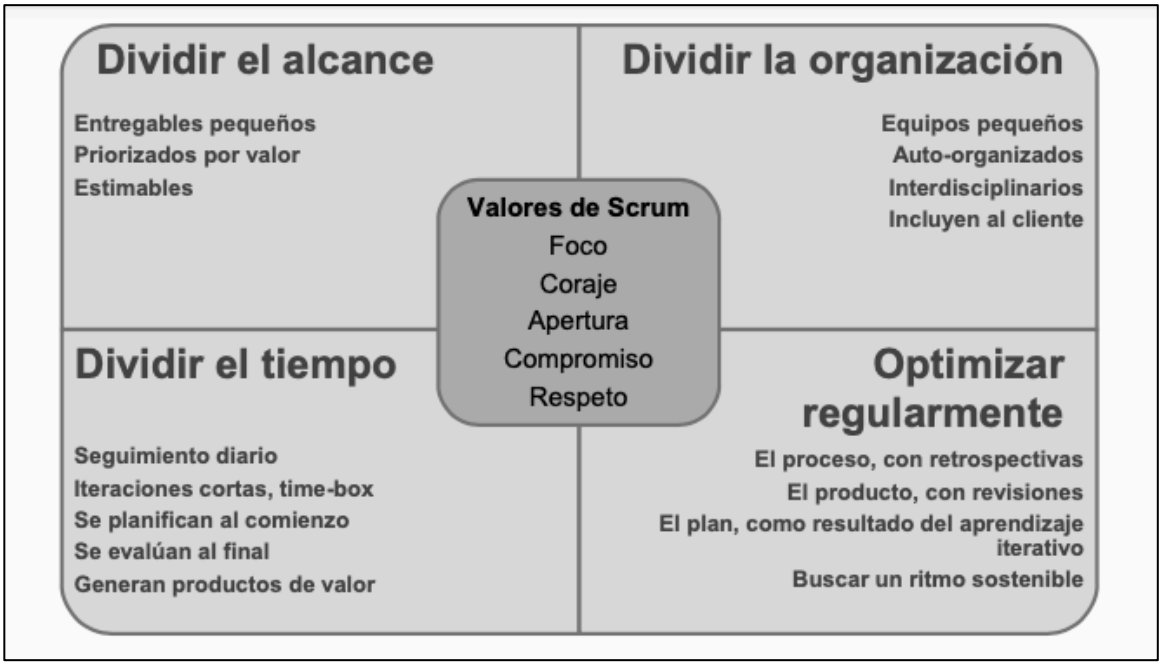


- **Limitar WIP**: consiste en acordar la cantidad de ítems que pueden trabajarse en paralelo por cada etapa del proceso. El límite marca la diferencia entre “Kanban” y una lista normal. El trabajo solo se dispara si existe capacidad para realizarlo, equipo elije cuando y que trabajo hacer.
- **Métricas**:
    - **Touch Time**: tiempo neto de trabajo sobre una tarea determinada.
    - **Lead Time**: tiempo entre momento de pedido y momento de su entrega.
    - **Cycle Time**: tiempo entre el inicio y el final del proceso (para un ítem de trabajo).

SCRUM (Framework de Trabajo)

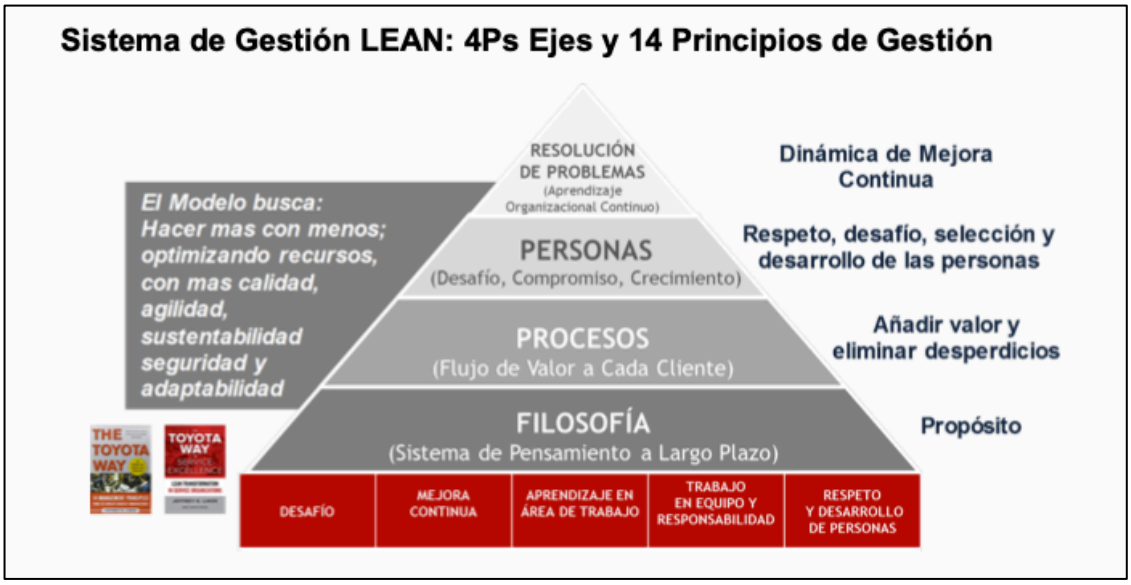
- **Framework de trabajo** para la administración de proyectos que **enfatiza el trabajo en equipo,** dentro de un **proceso iterativo hacia un objetivo bien definido.** Está basado en el principio de agilidad.
- Pensado para construir productos de forma incremental, en serie de tiempos denominados Sprints.
    - **Sprint**: periodo fijo de tiempo (1 a 4 semanas) en donde el equipo construye y entrega un incremento del producto.
    - Cada incremento es una versión mejorada del producto que alcanza criterios de aceptación y nivel de calidad requerido.
- **Roles**:
    - **Scrum Master**: dueño del proceso, colabora con el equipo eliminando impedimentos. Cuida el proceso y se asegura que Scrum se aplique correctamente. Protege y cuida al equipo.
    - **Product Owner**: dueño de la definición de "terminado". Representa al cliente y usuarios. Es dueño del backlog y define las prioridades. Cuida el valor del producto y establece plan preliminar de entregas. Mira desde la perspectiva del negocio.
    - **Equipo**: dueño del proceso de producción. Es pequeño (5 a 9 personas), interdisciplinario y auto-organizado. Responsable de construir el producto y de manera colaborativa define como transformar el backlog en un incremento de funcionalidad.
- **Ceremonias:**
    - **Sprint Planning:** de todo lo que está en el backlog, se eligen que van a desarrollar en el próximo sprint. Sucede una vez al inicio de cada sprint. Se asignan las tareas y las user-stories comprometidas.
    - **Daily Meeting:** reunión diaria que ocurre al comenzar el día. El equipo intenta remover los impedimentos.
    - **Sprint Review:** ocurre una vez al finalizar cada sprint. Se revisa si se cumplieron los objetivos del Sprint. Es con el Product Owner. REVISAN EL PRODUCTO.
    - **Retrospective:** ocurre una vez al finalizar cada sprint. Se revisa las lecciones aprendidas y acciones para mejorar el siguiente sprint. REFINAN EL SPRINT.
    - **Product Backlog Refinement:** reestimación de las historias del producto backlog. Se realiza de manera espontánea, cada vez que sea necesario.

- **Elementos:**
    - **Product Backlog:** lista ordenada de los requerimientos para un determinado aplicativo.
    - **Sprint Backlog:** lo que se va a implementar en ese sprint en particular. No se modifica durante el Sprint.
    - **Product Shippeable:** incremento del producto que se trabaja a lo largo del Sprint y que es entregado al final del mismo.
- **DoD (Definition of Done)**: se debe definir para poder comparar al final del sprint si la tarea fue completada de manera correcta. Es la "definición de listo o terminado". Cada incremento es una versión mejorada del producto que alcanza un nuevo DoD.
- **Valores de Scrum**:



LEAN

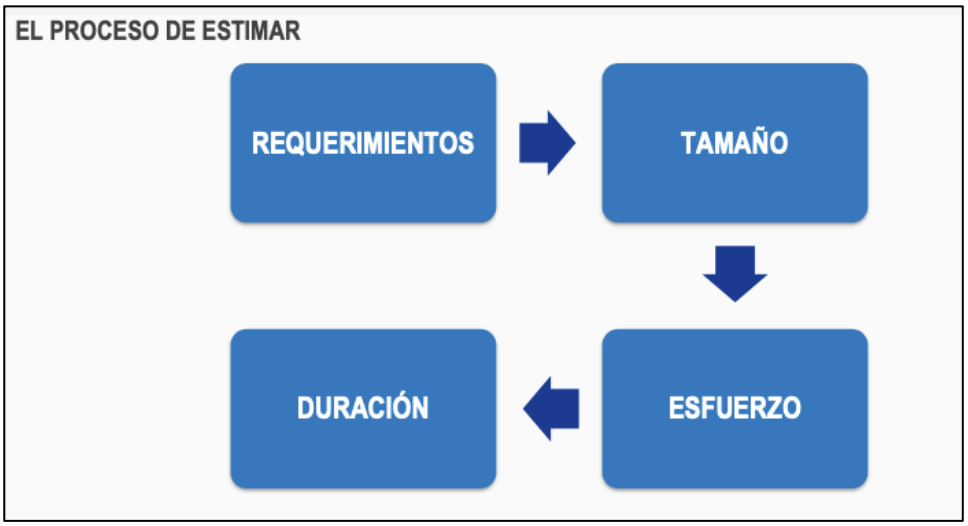
- **Filosofía** (serie de principios) y enfoque que hace hincapié en **la eliminación de residuos o de no valor añadido al trabajo**, **a través de** un enfoque en la **mejora continua para agilizar las operaciones**.
- **Lean se centra en ofreces una mayor calidad, reducir el tiempo de ciclo y reducir los costos**.
- Sistema de Gestión que busca llegar a procesos “ultra-livianos”.
- **Filosofía**:
    - Dinámica de mejora continua: resolución de problemas, aprendizaje continuo.
    - Respeto, desafío, selección y desarrollo de personas.
    - Añadir valor y eliminar desperdicios: optimizar recursos, valor a cada cliente y afecta procesos.
    - Propósito a largo plazo.



- Desperdicios en Proyectos de Software:
    - Entregar funcionalidad que el usuario no quiere / no aporta valor.
    - Corregir un Bug.
    - Tiempo muerto.

Unidad 4: *"Estimaciones de Software"*

Estimaciones

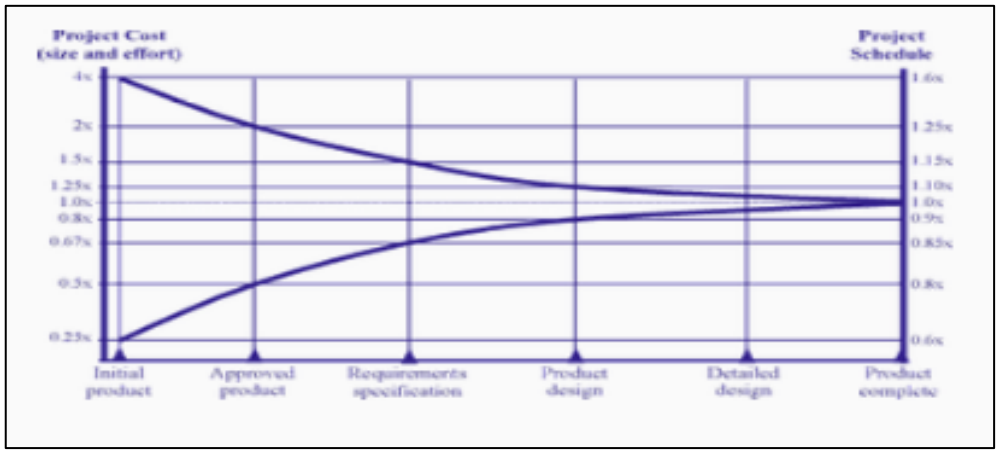
- Primer Pregunta -> ¿Cuál es el **tamaño** de lo que tenemos que construir?
- No estimo tareas ni dependencias. Estimo variables con incertidumbre. **El tamaño es la primera variable de estimación**, luego el resto viene por consecuencia del tamaño.
- **Tamaño**: dimensión del objeto a construir.
- **Esfuerzo**: horas hombre. La cantidad de tiempo que le lleva a una persona completar una tarea.
- **Duración**: tiempo calendarizado.
- **Requerimientos -> Tamaño -> Esfuerzo -> Costo -> Duración**. Estimar el **TAMAÑO** el resto es **CONSECUENCIA**.
- 

Fallos en las Estimaciones

- Optimismo.
- Estimaciones informales. (Estimaciones de mierda)
- No hay historia.
- Mala definición de alcance.
- Novedad / Falta de Experiencia.
- Complejidad del problema a resolver.
- No se estima el tamaño.
- Estimación buena al comienzo, pero luego el proyecto es mal administrado, no hay seguimiento y control y/o se confunde progreso con esfuerzo.

Incertidumbre

- El día que tenemos certeza de la estimación es el día que terminamos el proyecto.
- **Cuanto más refinada es la definición, más exacta será la estimación.**
- No asumir que solo por el paso del tiempo y de las fases de un proyecto se avanza con menos incertidumbre en las estimaciones.
- El "**cono de la incertidumbre**" **muestra niveles de incertidumbre** (probable) de las estimaciones en **cada etapa del proyecto, respecto del costo y duración** (tamaño y esfuerzo).
- Para achicar el espacio del medio, se debe forzar la eliminación de variabilidades. Dar certidumbre al proyecto.
-



Tips para Estimar

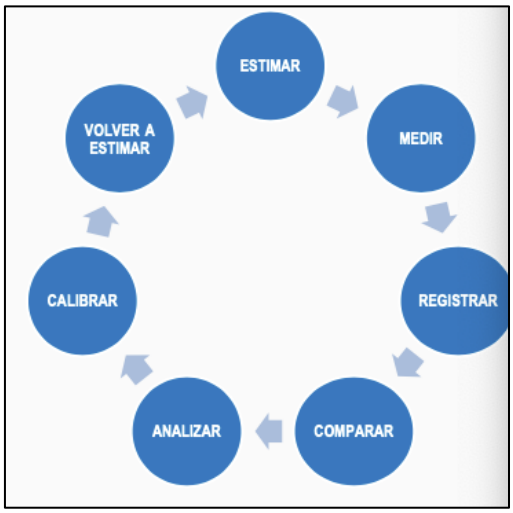
- **Ley de Parkinson**: "Toda tarea se expande hasta ocupar el tiempo que tiene asignado".
- Diferenciar entre estimaciones, objetivos y compromisos.
- Asociar a las estimaciones un % de confiabilidad. Presentarlas como rangos en lugar de un único valor.
- Siempre presentar los supuestos que se tuvieron en cuenta para arribar a la estimación.
- Considerar todas las actividades relacionadas al desarrollo.
- Recolectar datos históricos para tener como referencia.
- No asumir que solo por el paso del tiempo y de las fases de un proyecto se avanza con menor incertidumbre en las estimaciones. (Cono de la incertidumbre).

Estimaciones Creíbles

- Las estimaciones las hacen las personas, no herramientas ni modelos. Se necesitan juicios razonables.
- Estimaciones se basan en comparaciones. Se necesita historia de proyectos ya realizados.

Ciclo de Estimación

- Para lograr estimación confiable, lo ideal es seguir un ciclo para refinar el proceso de estimación, mejorarlo con el tiempo.
    - **Estimar**: comenzar por estimar el **tamaño** para derivar el esfuerzo y el costo.
    - **Medir**: a medida que evoluciona el proyecto, **medir el tamaño, esfuerzo y costo incurrido.**
    - **Registrar**: anotar / registrar las mediciones tomadas sobre las tres variables.
    - **Analizar**: realizar un análisis sobre las razones de desvíos, supuestos que variaron y temas no contemplados.
    - **Calibrar**: ajustar cada una de las variables y parámetros que intervinieron en el proceso de estimación. Reducir el desvío.
    - **Volver a Estimar**: estimar nuevamente en base a los ajustes realizados.



Métodos de Estimación

Métodos Rudimentarios

- Los métodos rudimentarios estiman esfuerzo a través de horas hombre. (Hs / H).
- **Juicio Experto**
    - Basado en la experiencia e intuición. Depende de la persona que haga la estimación.
    - Por lo general se recurren a analogías. Es muy subjetivo por los ojos del experto.
    - Es rápida y bajo costo. Se estiman las Hs de manera “pura”.
    - La persona debe tener conocimientos de la tarea para poder realizarla.
    - Problema: no es colaborativa
- **Clark (Pert)**
    - Basado en la experiencia e intuición. Conjunto de personas realizan la estimación como en Juicio Experto. Incluye factores optimista, probable y pesimista en su estimación-
    - Es rápida y de bajo costo. Estima Hs “pura” en completar la tarea.
    - Persona que ejecuta debe tener todos los conocimientos para hacer la tarea.
    - Problema: no es colaborativa.

- **Wideband Delphi**
    - Basada en intuición y experiencia. Es el juicio experto, pero en grupos consensuados por los expertos. Da lugar a la participación de todos los involucrados.
    - Se puede utilizar en etapas tempranas del proyecto. Se recomienda utilizarlo en proyectos poco conocidos donde no se cuenta con historia.
    - Ventajas: fácil de implementar y da sentido de propiedad sobre la estimación.
- **Planning Póker**
    - Basada en intuición y experiencia. Es el juicio experto, pero en grupos consensuados por los expertos. Da lugar a la participación de todos los involucrados.
    - Es mejor en iteraciones posteriores una vez que se conoce al equipo y al producto sobre el que trata el proyecto.
    - Se estiman tamaños de tareas relativos entre sí.
    - Se define una tarea pivote, y se le asigna un tamaño en story points que sigan una escala. Teniendo en cuenta esa tarea se le asigna un tamaño a las demás.

Métodos Paramétricos de Estimación

- Los métodos paramétricos estiman el tamaño del proyecto.
- **Functional Point**
    - Estima el tamaño en base a reglas y otorga *function points* a cada funcionalidad del sistema que se debe implementar.
    - Comparto sistemas midiendo la misma unidad que son funcionalidades.
    - Requiere un análisis de los requerimientos avanzado ya que es importante poder capturar todas las funcionalidades.
- **Use Case Point**
    - Basado en el método de *functional points*, estima el tamaño mediante la unidad de medida dada por los casos de uso que presenta el sistema.
    - Tiene en cuenta factores externos del entorno. Factores Ambientales (-) y Tecnológicos (+).
    - El método requiere que sea posible contar el número de transacciones en cada caso de uso. Una transacción es un evento que ocurre entre el actor y el sistema.
    - Numero de *use case points* dado por: casos de uso, cantidad de transacciones, actores y factores.

- **Story Points**
    - No es un método por sí mismo, sino que es una unidad de medición que se usa en los métodos.
    - Velocidad del equipo: cuantos story points meto en el sprint, depende de la velocidad.
    - Requerimientos No Funcionales: hay que considerarlos. Ejemplo cant. de usuarios posibles.
    - DoD: determinar el *definition of done* antes de arrancar.
- **Object Points**
    - Sistema es un conjunto de componentes (objetos) y cuenta cuantos componentes deben ser modificados. Muy utilizada en etapas de mantenimiento.
    - Analiza los componentes que se modifican. Es interesante porque el sistema ya está construido.
    - Los objetos contemplan: pantallas, reportes y módulos. No es necesariamente los objetos de OOP.
    - Se asigna a cada componente un peso de acuerdo con su clasificación por complejidad.
    - Considera como factor de ajuste el porcentaje de reúso del código.

Paper – "Timebox Development"

- Trata de un framework para llevar adelante proyectos, que solo es aplicable por periodos cortos de tiempo de 1 a 3 meses. Aplicar cuando los requerimientos son bien detallados.
- Fijo en el tiempo, no descuido la calidad y ajusto otras dimensiones (calidad contra funcionalidad principalmente).
- Se enfoca en mantener al equipo motivado y con el sentido de urgencia adecuado para poder cerrar un MVP o realizar la última parte de un proyecto más largo.
- Una vez concluido con el TBD se premia al equipo.