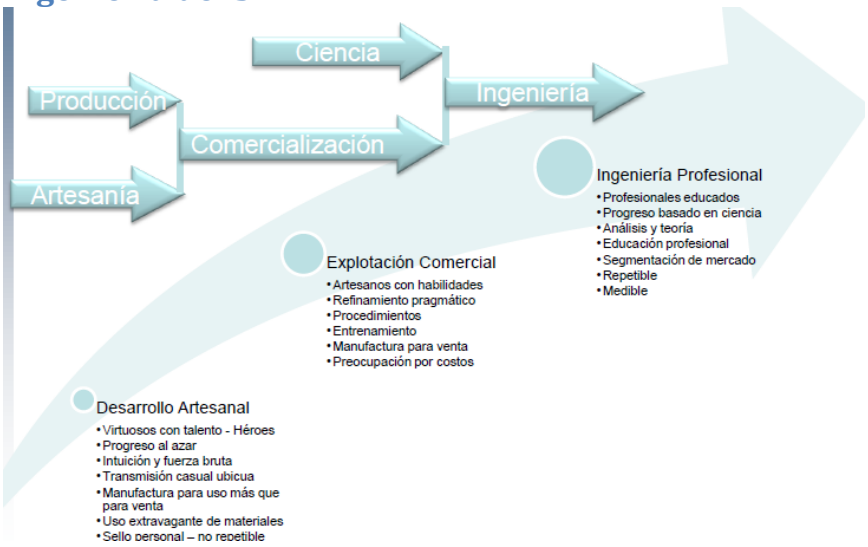
Ingeniería en SW: 1er parcial 2016

Unidad 1: Ingeniería del SW



Diferencias científico-ingeniero

•Un científico construye para aprender, un ingeniero aprende para construir
•Un ingeniero hace por 1 centavo lo que cualquiera hace por un peso
•El ingeniero trabaja con restricciones que un científico no tiene mucho en cuenta, por ejemplo look & feel, costos, tiempos, presupuesto, normas legales, de industria, de seguridad, etc

Software

-Engañosamente fácil de cambiar (causas y efectos)
-Es un producto mental, no físico: no se rige por las leyes de la física
-Se desarrolla, no se fabrica:
- •Mecanismo de replicación
- •A medida vs. Componentes
- •La calidad no es en la producción sino el desarrollo

-Envejecimiento diferente: no se desgasta por el uso

Ingeniería de SW

-SEI(1990): **Ingeniería:** aplicación sistemática del conocimiento científico en la creación y construcción de soluciones ("cost-effective") para resolver problemas prácticos al servicio del hombre. **Ingeniería del SW**: parte de la ingeniería que aplica los principios de las ciencias de la computación y la matemática para alcanzar soluciones ("cost-effective") a problemas de software.
–Fairley(1985): Disciplina tecnológica y de administración que se ocupa de la producción sistemática & mantenimiento de productos de software que son desarrollados en tiempo y costo estimados.
–IEEE: Aplicación de un proceso sistemático, cuantificable y disciplinado a la creación, desarrollo, la operación, y el mantenimiento de software.
En la Ing. en SW usamos técnicas, métodos, herramientas y paradigmas para solucionar problemas y obtener productos de alta calidad, siendo más eficientes y productivos.

Problemas habituales en el Desarrollo y Mantenimiento de SW

–Los proyectos se terminan con mucho atraso o no se terminan
–Se suspenden
–Se postergan
–Se cancelan
–Aparición de “Proyecto Fase II” o “Reingeniería de ...”
–Los proyectos se exceden en el costo estimado
–El producto final no cumple con las expectativas del cliente: no hace lo que se supone debería hacer
–El producto final no cumple con los requerimientos de calidad mínimos esperados: además de lo funcional, temas relacionados con mantenibilidad, robustez, escalabilidad, etc....

Causas

–No hay administración y control de proyectos: No se puede controlar lo que no se mide
–El éxito depende de los “gurúes”
–Se confunde la solución con los requerimientos. La tecnología no es la solución en si misma
–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. Existe una tendencia a ser optimista (simplificando, subestimando,...)

Reacciones:

–Buscar al “gurú” que nos saque del problema.
–Agregar gente a un proyecto que está atrasado
–Recortar/eliminar cualquier actividad que genere documentación
–Recortar/eliminar la etapa de Testing
–Recortar/eliminar cualquier actividad que no sea codificar
–Asumir definiciones en lugar del usuario
–Recurrir a una “bala de plata”: Un nuevo paradigma / Una nueva herramienta / Un nuevo lenguaje
–Justificar con frases como: “el usuario no sabe lo que quiere”, “Ya casi está, está en un 90%”, etc.

Análisis de la situación

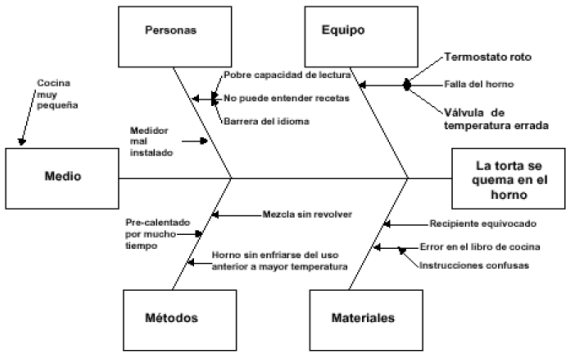
Hay que reconocer que tenemos un problema, buscar un sponsor y armar equipo multidisciplinario que comparta la problemática (y se vea afectado) y participe con su visión en la identificación de causas

Técnica Diagrama Causa-Efecto (Espinazo / Fishbone / Ishikawa):

•Permite identificar en detalle todas las posibles causas relacionadas a un problema determinado con el objetivo de descubrir sus raíces
•Concentra la atención en las causas y no en los síntomas
•Brinda mucha visibilidad en un solo gráfico, reuniendo todas las ideas para el diagnóstico desde diferentes puntos de vista

Ejemplo:

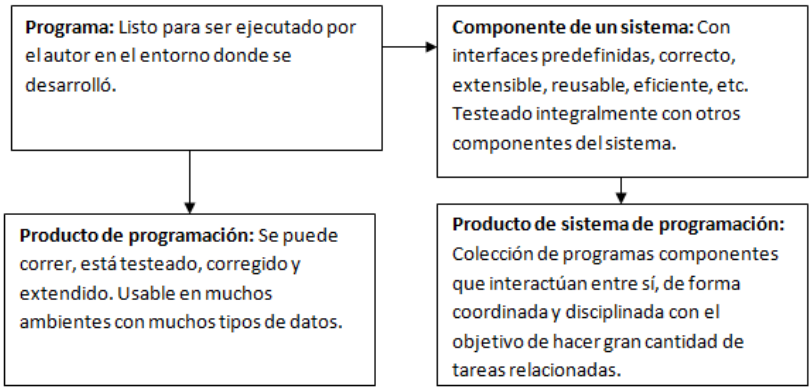
- **Diagrama Causa Efecto – Ejemplo:**



Apunte: The Mythical Man-Month Cap 1: The Tar Pit/El pozo de alquitrán

La construcción de grandes sistemas de computación se asemeja a un pozo de alquitrán (tar pit) en donde se hunden, al igual que lo hacían los grandes animales prehistóricos, los equipos de desarrollo de software, que intentan alcanzar los objetivos, la planificación y el presupuesto.

Evolución del producto de Programación de Sistema



Diversión en programación:

1. La diversión de crear cosas.
2. El placer de crear cosas que son útiles para otros.
3. La magia de combinar objetos con partes móviles que interactúan entre sí y ver cómo funcionan juntos.
4. El placer de aprender algo, a veces teórico, a veces práctico, a veces ambos, que surge de la naturaleza no repetitiva de las tareas.
5. El hecho de crear cosas de la nada, en un medio abstracto e ideal, producto de la imaginación, pero con resultados tangibles.

Padecimientos en programación:

1. Hay que hacerlo de forma perfecta, sino, no funciona. Ajustar lo necesario para alcanzar esa perfección es lo más difícil en el aprendizaje de la programación.
2. Dependencia de los demás y de los programas de otras personas, que frecuentemente están mal diseñados, pobremente implementados, incompletos (sin código fuente ni casos de prueba) y pobremente documentados.
3. Se tarda mucho tiempo en depurar y los defectos son difíciles de encontrar a medida que pasa el tiempo.
4. El producto se vuelve obsoleto muy rápido (cambio en requerimientos, tecnología, contexto, etc.)

Apunte: Classic Mistakes

Classic mistake es un error que se comete en muchos proyectos por mucha gente con lo cual está claramente identificado que es un error y tiene consecuencias negativas conocidas, por eso es importante tenerlos en cuenta.

Evitar estos errores no asegura obtener un desarrollo más rápido, pero cometer alguno asegura obtener un desarrollo más lento.

Gente

- Baja motivación
- Personal inadecuado: *"Las personas no están preparadas adecuadamente para el proyecto."*
- Problemas con los empleados: "*No se gestiona correctamente el impacto de los empleados problemáticos en el resto del equipo ni en el proyecto"*
- Actitudes heroicas: *"Surgen héroes que son los que se echan a la espalda el proyecto ignorando al resto. Pensar que se puede llegar aunque sea imposible"*
- Agregar más gente a un proyecto atrasado: *"lo que se consigue es que el proyecto se retrase aún más debido a la curva de aprendizaje y a la dedicación del personal actual a formarlos"*
- Oficinas ruidosas y muy llenas: *" Baja la productividad y entorpece el desarrollo del proyecto"*
- Fricciones entre desarrolladores y los clientes
- Expectativas poco realistas
- Falta de patrocinio eficaz: *"puede forzar a aceptar fechas límite poco realistas o a hacer cambios que perjudiquen el proyecto.*
- Falta de compromiso de stakeholders
- Falta de participación del usuario
- Anteponer la política por sobre los objetivos del proyecto
- Pensamiento ilusorio: *"Actuar pensando lo que nos gustaría que ocurriese en lugar de pensar en lo que es más probable que suceda. Por ej creer que no se va a llegar a la fecha estimada pero si hay suerte, todos trabajan duro y nada sale mal, entonces si llegarán, o por ejemplo no mostrar los últimos cambios al prototipo por creer que se sabe lo que el cliente quiere"*

Procesos

- Planificación demasiado optimista
- Gestión de riesgos insuficiente
- Fallo en los proveedores
- Planificación insuficiente
- Abandono de la planificación bajo presión*: "pasar a modo code & fix"*
- Desperdiciar el tiempo antes del kickoff del proyecto
- Acortar las tareas de alto nivel: *"Generalmente las tareas que no generan código pero que su reducción tiene un impacto importante en las fases posteriores"*
- Diseño inadecuado
- Reducción de las tareas de control de la calidad
- Control insuficiente de gestión: "*Pocos controles que permitan detectar los problemas a tiempo y realizar acciones correctivas"*
- Se prepara el producto para su liberación demasiado pronto o demasiadas veces: *"Se realizan las tareas de pruebas de rendimiento, impresión de la documentación final, etc. demasiado pronto o con demasiada frecuencia"*
- Omitir tareas al estimar
- Planificar pensando en que recuperaremos el retraso acumulado más tarde: *"Si en el primer hito de entrega nos retrasamos 1 mes, volvemos a planificar pensando que ese tiempo lo recuperaremos más tarde"*

- Code-Like-Hell programming: Programar a cualquier precio: "*Se piensa que codificar a cualquier precio es un método más rápido de desarrollo"*

Producto

- Añadir requisitos no solicitados por el cliente: *"sobrevalorados, gold-plating"*
- Corrupción del alcance*: "Añadir, cambiar, modificar continuamente los requisitos del proyecto sin importar en qué fase del desarrollo estemos"*
- Los desarrolladores añaden características no solicitadas: *"Durante la codificación se añaden características que no se solicitaron por parte del cliente."*
- Negociaciones de tira y afloja: *"Se concede un retraso en la planificación (asumiendo un error en la misma) pero a cambio se añaden nuevos requisitos (cometiendo un nuevo error)"*
- Desarrollo enfocado a la investigación: *"Cuando el proyecto debe desarrollar algoritmos o métodos que nunca antes se han utilizado en lugar de desarrollo de software estamos en una investigación y las estimaciones de las investigaciones no están tan claras ni definidas."*

Tecnología

- Síndrome de la bala de plata: "C*onfiar demasiado en que una nueva tecnología, metodología, lenguaje, plataforma, etc. resolverá los problemas de planificación"*
- Sobreestimación del ahorro por la utilización de nuevas herramientas o métodos: *"Pensar que una nueva tecnología o nueva herramienta incrementarán la productividad. Pero obtener el máximo rendimiento de una nueva herramienta o método lleva un tiempo entre que aprendemos a usarlo (curva de aprendizaje) y lo usamos de la forma correcta."*
- Cambiar herramientas en medio de un proyecto
- Falta de control automático de código fuente

Apunte Classic Mistakes 2008

A los errores anteriores se suman:

- Confundir estimaciones con objetivos: *"Se pacta un objetivo con el cliente: tener el sistema funcionando en una fecha, y después se acaba concluyendo que el desarrollo tardará justamente lo que se ha pactado… sin seguir ninguna metodología de estimación).*
- Abuso de la multi-tarea: *"Llevar varios asuntos al mismo tiempo a costa de nuestra productividad real. En el ámbito de una empresa multi-proyecto, los recursos típicamente se asignan a varios proyectos a la vez, con el coste añadido que supone volver a "enfocarse" al ir saltando entre proyectos"*
- Suponer que el desarrollo global tiene un impacto insignificante sobre el esfuerzo total: *"Cuanto mayores sean las diferencias entre los sitios en términos de husos horarios, las culturas de empresa, y las culturas nacionales, mayor será el esfuerzo"*
- Visión del proyecto poco clara: *"Contribuye a cambios en la dirección del proyecto, incluyendo los requerimientos detallados; los planes no están alineados con las prioridades del proyecto y puede no cumplirse con el cronograma"*
- Confiar más en el mapa que en el terreno: "*Un mapa no deja de ser una herramienta, no podemos ensimismarnos con nuestras planificaciones y no comprobar lo que está pasando realmente en nuestro proyecto"*
- Externalizar para reducir costes: *"Tener en cuenta los problemas de coordinación, comunicación, y hasta a veces, cultura, que supone llevar parte de los desarrollos a otras empresas y otros países, generalmente termina resultando en costos más altos y cronogramas más largos"*
- Dejar que el equipo se oscurezca*: "Dejar que el equipo trabaje con poca supervisión y poca visibilidad en el progreso, puede provocar que se retrasen y no avisen a tiempo por ejemplo (Nota: este nuevo error reemplaza al anterior "Control insuficiente de gestión")*

Unidad 2: Modelos de calidad

Visiones de la calidad

- Visión Trascendental: se puede reconocer, pero no definir. Es un conocimiento generalizado, relacionado a la cultura general, subjetivo (Ej: “es tal marca”)
- Visión del Usuario: cuanto se adecúa a su propósito. “Fit for use”: Qué tanto se adapta a las necesidades del usuario. Distintos usuarios pueden valorar distintos atributos, lo puedo saber relevando.
- Visión de la Manufactura: conformidad con la especificación. “Meet requirements”. No asegura el éxito pero lo impulsa. Es objetiva y cuantificable. Enfocado en el proceso de la manufactura. Cuánto mejor esté armado el proceso, será más repetible, se minimiza el “desperdicio” y hay menos “re-trabajo”, menos costo.
- Visión del Producto: vinculada a las características inherentes de este. Es objetiva, se fija si tiene más atributos que otro o más cantidad de un atributo que otro. Más atributos, más costo, más calidad.
- Visión Basada en el Valor: cuanto está dispuesto a pagar el usuario por lo que quiere. Calidad contra Valor (Ej: Windows: Pago, pero con licencias Vs Linux: Free)

Costo de Calidad: Compuesto por los costos de calidad (Prevención y Medición) y costos de no calidad (Fallas internas: en desarrollo y testing por ejemplo, y fallas externas: impacta en los clientes)

Definición de calidad

Calidad: Grado en que un sistema, componente o proceso cumplen con los requerimientos. Grado en que un sistema, componente o proceso cumplen con las necesidades o expectativas del cliente (IEEE).

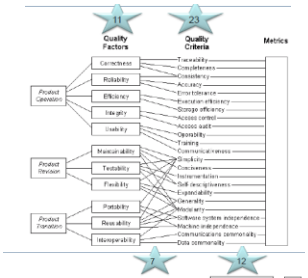
Calidad de SW: Grado en que el proceso software posee una combinación deseada de atributos (IEEE).

Calidad de producto >> ISO 9126

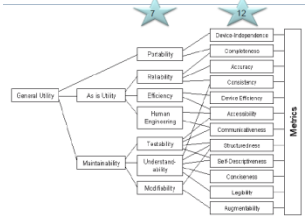
Calidad de proceso >> CMMI

Modelos de calidad

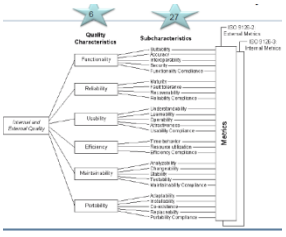
a) Modelo de McCall (1977): 3 grupos (Product Operation, Product Revision y Product Transition), con 11 Quality factors y 23 Quality Criteria, dan las métricas.



b) Modelo de Barry Boehm (1978): empieza por la utilidad general y se va abriendo en subcategorías, como Mantenibilidad, Para lo que es útil, portabilidad, etc. Para obtener las métricas.



c) ISO-9126: Expone características internas y externas de la calidad, en 6 grandes grupos, que se subdividen en 27 categorías para dar las métricas externas e internas. Es el estándar internacional para medir calidad de software.
Características:



- Funcionalidad: ¿Las funciones satisfacen las necesidades explicitas e implícitas?
    - Cumplimiento de la funcionalidad, precisión, interoperabilidad (interacción con otros sistemas), idoneidad y seguridad.
- Confiabilidad: ¿Puede mantener el nivel de rendimiento bajo ciertas condiciones y en cierto tiempo?
    - Madurez, tolerancia a fallos, capacidad de recuperación ante fallas.
- Usabilidad: ¿El software es fácil de usar y de aprender?
    - Facilidad de aprendizaje, facilidad para entenderlo, para atraer al usuario, para operarlo (cuánto tarda el usuario en empezar a usarlo).
- Eficiencia: ¿Es rápido y minimalista en el uso de recursos?
    - Comportamiento en el tiempo (medible) respecto de los recursos disponibles (memoria, disco, etc.)
- Mantenibilidad: ¿Es fácil de modificar y verificar? Algo es más o menos mantenible según cuánto tiempo lleve modificarlo.
    - Facilidad de análisis (esfuerzo para detectar la raíz de una falla), facilidad de cambio y de pruebas, estabilidad.
- Portabilidad: ¿es fácil de transferir entre ambientes?
    - Facilidad de instalación, intercambiabilidad, coexistencia y adaptabilidad.

•Para tener éxito hay que usar sabiamente más de una visión. Ej: usar la visión del usuario para tomar los requerimientos, la del producto para hallar los atributos de producto que satisfacen esos requerimientos, y la de la manufactura para construir el producto.
•Un buen proceso de desarrollo de software debe contemplar las distintas visiones a través de las actividades de SQA y teniendo en cuenta los costos asociados

Fundamentos de la calidad

1. Como ingenieros de SW se espera que compartamos un compromiso orientado al software de calidad como parte de nuestra cultura.
2. La calidad es una de las dimensiones sobre las que tenemos que trabajar en proyectos de software (junto con la funcionalidad, los costos y el calendario)
3. La calidad termina siendo una ecuación de costo y beneficio (trade-off). ¿Siempre?
4. Existe un costo de calidad (prevención y medición) contra un costo de no calidad (fallas que afectan al cliente y al equipo de desarrollo)
5. Inicialmente se establecen junto con los requisitos del proyecto, los requisitos de calidad. Más allá de esto, la calidad se discute en cada paso del proyecto: arquitectura / diseño / pruebas / aceptación.

•Los modelos de calidad permiten convertir la calidad del software en algo tangible y medible.
•Estos modelos, una vez adaptados y calibrados, nos permiten medir la calidad desde los requerimientos (características externas) pasando por el diseño y la codificación (características internas)
•Son empíricos y perfectibles, el ISO 9126 es más abarcativo y estandarizado para medir calidad de producto.

CMMI

•No es un proceso para desarrollar Software, ni una descripción de procesos, ni un ciclo de vida.
•Es un conjunto de modelos para evaluar la madurez de un proceso y organizar el esfuerzo para mejorarlo, describiendo un camino incremental de mejora. Nos ayuda a:
–Saber dónde estamos
–Tener un mapa que nos indique hacia dónde ir

•Cubren prácticas de planificación, ingeniería y gestión de desarrollo y mantenimiento de SW.
•Contiene: Modelos CMMI DEV, CMMI IPPD, etc, Métodos de evaluación (SCAMPI A, B, y C) y Entrenamientos
•Es un framework que define los elementos clave de un proceso efectivo
•Una guía para evolucionar desde actividades inmaduras y ad-hoc hasta procesos maduros y disciplinados
•Nos dice QUÉ hacer, pero no CÓMO ni QUIÉN.
Madurez es el grado en que un proceso está:
•definido y documentado
•administrado y controlado
•medido y es efectivo

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

Representaciones del CMMI

a) Representación Continua: máxima flexibilidad para enfocarse en una Process Area especifica de acuerdo a los objetivos del negocio.
b) Representación por Niveles: da una hoja de ruta a implementar: grupo de áreas de proceso, secuencia de implementación.

Hay que analizar cuál es la representación que más se adapta a los objetivos de la organización

Level	*Continuous Representation Capability Levels*	*Staged Representation Maturity Levels*
Level 0	Incomplete	
Level 1	Performed	Initial
Level 2	Managed	Managed
Level 3	Defined	Defined
Level 4		Quantitatively Managed
Level 5		Optimizing

Estructura

•Requeridos: Si *no se cumplen, no cumple con la procesaria*
–Specific Goals
–Generic Goals
•Esperados*: Prácticas alternativas para cumplir los objetivos*
–Specific Practices
–Generic Práctices
•Informativos
–Subpractices, ejemplos de work products

CMM-I Nivel 1: Inicial: Todas las empresas lo son.

- Procesos ad-hoc y caóticos, sin entorno estable que soporte procesos.

- Éxito basado en heroísmo, retrasos en las entregas, sobre-compromisos y no pueden repetir éxitos anteriores. Si tienen éxito, exceden presupuesto y plazos de entrega.
- Tienden a abandonar el plan en casos de crisis.

CMM-I Nivel 2: Administrado. Repetible

- Los procesos se ejecutan de acuerdo a las políticas, se monitorean, controlan y revisan. Se evalúan por su adherencia al proceso definido y se mantiene la disciplina.
- Se llevan a cabo por recursos adecuados para producir las salidas de forma controlada.
- Se involucra a los stakeholders relevantes y se establecen compromisos con los usuarios relevantes.
- Los trabajos y servicios satisfacen los estándares, procedimientos y procesos.
- Puede haber distintas prácticas para distintos grupos.

CMM-I Nivel 3: Definido: Zoom de Procesos.

- Procesos bien descriptos y documentados, mas aun que en nivel 2. Prácticas definidas a nivel de toda la organización.
- Están definidos con sus entradas, criterios de entrada, procesos, salidas, criterios de salida y verificaciones (ETVX+)

CMM-I Nivel 4: Cuantitativamente Administrado.

- Se establecen objetivos cuantitativos para manejar la calidad y la performance de los procesos y se utilizan como criterios para administrarlos. Estos se basan en las necesidades de los usuarios.
- Calidad y performance medida sobre bases estadísticas.
- Aumentan predictibilidad.
- Tomar decisiones en base a medidas de los proyectos.

CMM-I Nivel 5: Optimizado.

- La organización mejora continuamente sus procesos basada en un entendimiento cuantitativo de los objetivos del negocio y necesidades de performance.
- Mejoras de proceso y tecnológicas son la base de la mejora continua.
- En el nivel 4 se busca entender y controlar los procesos. Aquí en el 5 mejorarlos para mejorar la performance de la organización. “Mejora Continua”.

Beneficios Tangibles: Costo, Agenda, Productividad, Calidad, Satisfacción del Cliente, ROI

Beneficios Intangibles: mejora moral del equipo, mejora la calidad de vida en el trabajo, la comunicación y calidad percibida por el cliente. Menos sobresfuerzo y menor rotación de personal.

Apunte: Cuando Un Software Lo Suficientemente Bueno Es Mejor

Los clientes pretenden siempre “optimizar” los 3 parámetros básicos “Rápido, Bueno y Barato”, incluso cuando es claramente imposible. Prefieren rápidas entregas por sobre los defectos y hasta prefieren usar herramientas menos útiles o adecuadas “pero que son más baratas”.

Suponiendo que tenemos un producto no del todo perfecto y hasta poco performante, pero que satisface lo necesario, es probable que el cliente no quiera cambiar por un nuevo producto mejor ya que es “lo suficientemente bueno” y el cambio quizás traiga otros problemas.

Al ofrecer un nuevo producto, es el cliente el que decide el equilibro justo entre los 5 parámetros fundamentales del desarrollo de software (costo, tiempo, personas, funcionalidad y calidad), sabiendo que con lo vertiginoso que es el mercado, puede cambiar la prioridad.

Parámetros de un proyecto SW: Tenemos X número de personas, en Y tiempo, con un costo de Z pesos, con P unidades de funcionalidad y Q bugs por function point. Pero X,Y,Z,P y Q no suele ser aceptados por el cliente, que además requiere la mitad de tiempo o el doble de funcionalidad. Hay que mostrarles la situación real y hacerlos racionalizar, con cada parámetro uno por uno, los

cambios y los efectos. Si el usuario quiere el software en 0,5 unidades de tiempo, entonces debemos presentar una contra propuesta que muestre el impacto que ese cambio tendrá en los otros parámetros. Puede que renegociar no sea tan importante en un proyecto de tres meses, pero en un proyecto que se extiende por un año o dos, es casi inevitable en el turbulento mundo de los negocios de hoy en día.

Unidad 3: Composición de un Plan de Desarrollo y Mantenimiento de Software

Proyecto

Esfuerzo temporal llevado a cabo por personas para crear un producto o servicio único para lograr un objetivo.

- Tiene un objetivo o beneficio que guía el proyecto
- Temporal: porque tiene principio y fin
- Único: No es repetible, cada proyecto es diferente al otro
- Posee Recursos limitados
- Consta de una sucesión de actividades o fases en las que se coordinan los distintos recursos

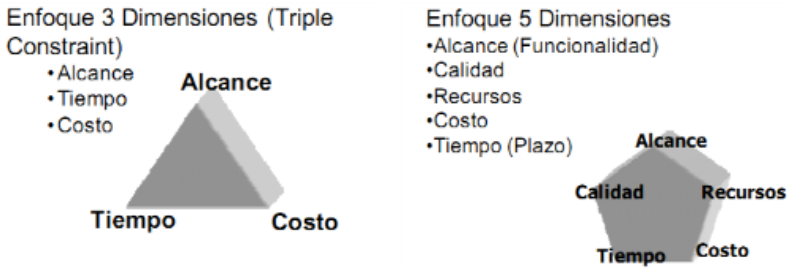
Project Management

La administración de proyectos es una disciplina que consiste en planificar, organizar, obtener y controlar recursos, utilizando herramientas y técnicas para logar que el proyecto logre sus objetivos en tiempo y forma.

PMI (Project Management Institute)→ publica el PMBOK (PM Body Of Knowledge) donde aparecen las áreas que comprende la administración de proyectos, comunes a la mayoría de los proyectos.

Dimensiones de un proyecto

2 enfoques:



- Driver: Empuja el proyecto para lograr los objetivos. Tengo cierto nivel de control sobre esa dimensión. Ej: Salgo con cierta funcionalidad.
- Restricción: Hay que cumplirlo sío sí. No es negociable, no tengo control sobre esta dimensión.Generalmente es algo externo. Ej: Tiempo de entrega para cierto evento, o si la cantidad del equipo es fija será restricción.
- Grados de libertad: respecto de las demás dimensiones, depende el proyecto cuáles se pueden mover para negociar. Ej: Si no puedo mover costos ni tiempo, achicar funcionalidad. O si la funcionalidad tiene que estar, mover $ y recursos.

Composición de Plan de Proyecto

–Objetivos (de negocio / de proyecto)
–Alcance: límites y supuestos

–Restricciones
–Riesgos del proyecto
–Calendario: WBS, Gantt y Recursos asignados a las tareas
–Estimaciones: Técnicas, supuestos, historia y Resultados de la estimación
–Recursos: humanos, otros
–Estructura y roles
–Procedimientos de tracking y control
–Sponsor: Autoridad que lleva adelante el proyecto
–Stakeholders: Todos los involucrados en el proyecto
–Usuarios: directos (interactúa directamente son el sistema), indirectos (Hace uso del sistema pero no lo opera), key-user (usuario con conocimiento, para relevar requerimientos) y champion-user(es el referente, experto en el dominio del problema).

•Objetivo: Describe los resultados de negocio esperados a los cuales contribuye la concreción del proyecto. El éxito del proyecto deberá medirse en el grado de cumplimiento de dichos objetivos.
Los objetivos deben ser precisos (de lo contrario no se podrá corroborar si se alcanzaron).

- Recordar no confundir la solución con el objetivo del proyecto: La solución es el curso de acción para alcanzar el objetivo. El objetivo no es la forma en que vamos a hacer algo sino el resultado que esperamos
- Debe ser SMART: específico, medible, alcanzable, realista y acotado en el tiempo.

•Alcance: Define los límites del sistema, lo que incluye. Si es necesario, también se aclara lo que no incluye.
Hay un alcance de proyecto "incluye estas actividades" y un alcance de producto "incluye estas funcionalidades".

Riesgos

- Un **riesgo** es un problema que todavía no llegó, un evento negativo que impacta en alguno de los objetivos del proyecto.
- Un **problema** es un riesgo que se manifestó, un riesgo al que le llegó la hora

Es la medida de la probabilidad y la consecuencia de no lograr un objetivo del proyecto, es parte de toda actividad y nunca puede ser eliminado por completo
–No es malo en sí mismo, es esencial para progresar: si no existiesen los riesgos siempre haríamos lo mismo y los errores son parte esencial del aprendizaje
Riesgo (según SEI): es la posibilidad de sufrir una pérdida. En un proyecto de desarrollo, esa pérdida puede verse de la siguiente forma:

- •Disminución de la calidad del producto final
- •Incremento en el costo estimado
- •Finalización fuera de la fecha establecida
- •Falla

–Los riesgos tratan sobre eventos posibles del futuro que se caracterizan por:

- •Probabilidad de ocurrencia
- •Impacto negativo si ocurren

–La exposición al riesgo se mide:

- •Probabilidad de ocurrencia * Impacto negativo

Gestión de riesgos

–Involucra todas las tareas relacionadas con la identificación, resolución y comunicación de los riesgos
–Se basa en tomar decisiones bajo niveles de incertidumbre

–No involucra decisiones futuras: incluye todas las decisiones presentes que tienen incidencia en el futuro
–No es una actividad aislada, debe acompañar a todo el ciclo de vida de desarrollo de SW.
El ciclo de gestión de riesgos comprende: identificarlos, analizarlos, planificar, seguimiento y control. Todo esto acompañado con la comunicación. Depende del proyecto, ir viendo en qué momento volver a empezar el ciclo.

1. Identificación: Se utilizan varias técnicas entre brainstorming, taxonomías, repaso de experiencia, etc. Deben documentarse. Para enunciarlos se suele utilizar la notación de Glutch: "Dado que *"causa fuente"* entonces posiblemente *"consecuencia local"* y entonces *"consecuencia global"*.
2. Análisis: Convertir la información de riesgos que se identificó en información que permita tomar decisiones. Cada riesgo debe estar lo suficientemente claro para permitir decidir acerca de él. Esta actividad permite concentrarse en los riesgos más críticos. Para ello se debe: estimar probabilidad e impacto, estudiar causas y acciones correctivas, identificar causas comunes e identificar tiempos de ocurrencia.
3. Planificación: La información de riesgos se transforma en decisiones y acciones. La priorización se hace en función del grado de exposición y de la urgencia que demande la acción correctiva. Un plan de acción puede tener la siguiente forma:
   - Aceptar el riesgo sin tomar acciones, tomar la decisión de manera consciente, aceptando las consecuencias derivadas de su posible ocurrencia.
   - Evitar el riesgo: atacar la fuente, por ejemplo cambiando el diseño del producto final.
   - Transferir: trasladar a un tercero todo o parte del impacto negativo de una amenaza. La transferencia de un riesgo le confiere a una tercera persona la responsabilidad de su gestión, no lo elimina. Ej: Seguros
   - Mitigar: Planes para reducir la probabilidad de ocurrencia o reducir el impacto.
   - Plan de contingencia: establecer el evento "trigger" que dispare el despliegue el plan de contingencia.
4. Seguimiento: Monitorear que las acciones que fueron definidas en el plan se ejecuten, aplicar las métricas sobre presupuesto, calendario y consideraciones técnicas, informar las desviaciones respecto de los objetivos e identificar nuevos riesgos.
5. Control: Realizar las correcciones de las desviaciones producidas sobre el proceso de riesgos. Control de procesos de riesgos (SQA)
6. Comunicación: Feedback sobre las actividades sobre los riesgos. Para poder ser analizados y administrados, los riesgos deben ser comunicados a los niveles adecuados de la organización.

Calendarización de proyectos

Armar el plan: Descomponer en fases/etapas, con tareas/subtareas/actividades. Determinar las relaciones de precedencia en la descomposición anterior, asignar los recursos a las tareas y revisar el plan resultante con el objetivo de asegurar el cumplimiento de los plazos y resolver los conflictos con las asignaciones de recursos.

WBS

Método para representar en forma jerárquica los componentes de un proceso o un producto. No determina dependencias, solo jerarquía y tiene que tener todas las tareas o entregables para cumplir con el alcance.

Definición de WBS:

–Definir el nodo raíz de la jerarquía (nombre del proyecto)
–Dividirlo en los componentes principales que lo conforman
–Continuar con la apertura de los componentes principales en otras sub-partes
–Finalizar el proceso de división cuando se llegue a un nivel donde se pueda trabajar con el nivel de granularidad alcanzado (hasta que podamos para cada tarea estimar su esfuerzo, su duración y asignarle recursos)

Tipos de WBS:

- A nivel conceptual: WBS por proceso (análisis, desarrollo, construcción, testing, implementación), por producto (muestra la funcionalidades) y WBS híbrida (ciclo de vida en el alto nivel con detalle de las características de producto en el bajo nivel).
- Por formato: Diagrama jerárquico (árbol) o tabla indentada.

Dependencias:

•Definir las dependencias de todas las tareas sin olvidar las dependencias con terceras partes (otros proyectos relacionados)
•Asegurar que cada tarea tenga un entregable preciso (permite definir cuando la tarea finalizó) y a su vez los entregables necesarios para otras tareas sucesoras
•Recordar “Entry Task Verification eXit”
•Incluir milestones (hitos / puntos de revisión)
•Todo proyecto debe tener puntos de revisión periódicos
•Identificar el camino crítico
•Secuencia de tareas cuyo atraso provoca atrasos en la fecha de fin del proyecto
•Cuidado con las tareas no críticas que tienen un límite a partir del cual se transforman en críticas

Recursos:

•Conocer los recursos con los que se cuenta y cuál es su disponibilidad real para asignarlos, acordar disponibilidad y confirmar asignación
•Resolver conflictos de sobreasignación, ya que cuando se hace la descomposición y se definen las dependencias, la duración de las tareas se realiza teniendo en cuenta la dedicación completa de los recursos y en la práctica no es real.

Podemos resolverlo (siempre que la tarea lo permita):

–Alargando la duración de la tarea
–Asignando recursos para su ejecución
–Alterando el orden de la misma

•No olvidar la asignación de los roles cubiertos por el usuario: el usuario “campeón” suele ser un recurso con poca disponibilidad.
•Recomendación: evitar caer en la “dedicación completa”: No hay recurso que esté disponible el 100% del tiempo, tener en cuenta vacaciones, enfermedades, emergencias, training, etc...
•Consideraciones al armar el equipo de trabajo:
–La complementación técnica y social de los participantes
–La cantidad de gente
–La posibilidad de crecimiento futuro del equipo
–El skill de las personas (fortalezas y debilidades)
•Todos los recursos tienen que tener un backup asignado, debidamente asignado y notificado.
–Tener especial cuidado con los que están asignados a las tareas críticas.

- Un buen equipo comparte el objetivo y se compromete:

–Tiene una identidad propia
–Compromiso con el equipo
–Confianza mutua
–Comunicación efectiva
–Tiene un sentido de autonomía
–Tiene un sentido de empowerment

Planificación adaptativa

Base de las metodologías agiles: Plantea la naturaleza cambiante de la realidad y los sistemas, tratando de acomodarse a esos cambios rápidamente.

• Se adaptan fácilmente a los cambios de alcance y a los cambios de prioridades.
• Planifica a corto plazo, por lo que se hace difícil determinar el fin del proyecto.
• Se planifica el trabajo en un esquema iterativo: En cada iteración se define qué funcionalidades se completarán.
• Es complejo utilizarlas en proyectos grandes o de misión critica.
• Requieren un nivel de gestión y conocimiento de situación a nivel personal y un involucramiento constante del usuario /cliente.

Ej: RUP, Scrum, eXtreme Programming.

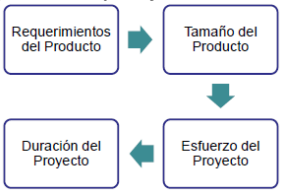
Estimaciones

Qué estimamos? Tiempo, recursos, esfuerzo (días/personas), costo y tamaño (cuán grande o pequeño es lo que deseo construir). Al inicio, la estimación es con más incertidumbre.

Pasos: tamaño >> esfuerzo >> cronograma

¿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
-Hay que diferenciar estimación, compromiso y objetivo

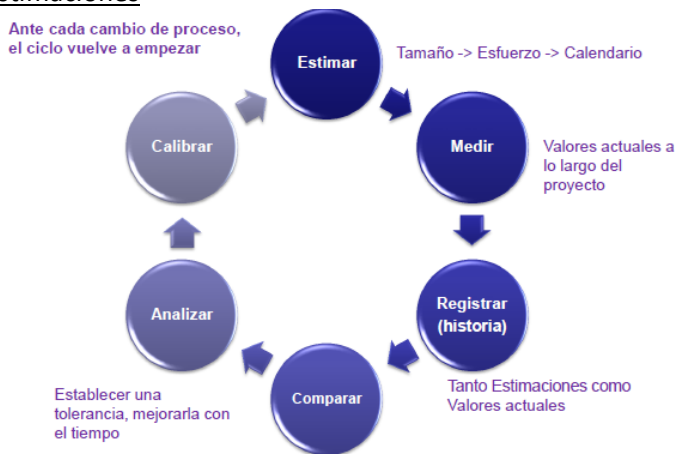


Bases de estimaciones creíbles:

1. Las estimaciones las hacen personas, no herramientas ni modelos. Se necesitan juicios razonables y compromisos con los objetivos organizacionales que no se pueden delegar a modelos automáticos.
2. Las estimaciones se basan en comparaciones. Se buscan similitudes y diferencias con proyectos previos

3. Antes de estimar, se necesitan conocimientos basados en información histórica de proyectos pasados para hacer las comparaciones sobre bases firmes
4. Un método creíble de estimación debe ser de "caja blanca"
5. Las estimaciones pueden mejorar con la cooperación de todos para juntar datos históricos

Ciclo dorado de las estimaciones



Tips para estimar

-Seleccionar o desarrollar un método y aplicarlo (ciclo de estimaciones)
-Tener claros los conceptos: estimación, objetivo, compromiso
-Asociar a las estimaciones un % de confiabilidad, indicando nivel o expresarlas como rango
-Siempre presentar junto con la estimación, los supuestos que se tuvieron en cuenta.
-Tener presente: La Ley de Parkinson y El síndrome del estudiante
-Considerar todas las actividades relacionadas al desarrollo de sw, no solamente codificación y testing (Análisis, diseño, actividades de SCM, etc.)
-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)
-Recolectar datos históricos para tener como referencia
-Considerar que puede haber enfoques para estimar "Top-Down" y "Bottom-Up"

Métodos para estimar

Métodos no paramétricos

- **Juicio Experto:**
  - Depende de la persona que haga la estimación
  - Se basa en la experiencia personal (netamente "estomacal")
  - Por lo general se recurre a analogías.
- **Pert:** También conocido como Clark
  - Se basa en el juicio experto.
  - Incluye factores optimistas y pesimistas en su estimación.
  - Estimación = (Optimista + 4 x Medio + Pesimista) / 6
  - Se puede usar para estimar tamaño y esfuerzo
- **Wideband Delphi** : *"sirve para empezar"*
  - Es el juicio experto en grupos
  - Participan todos los involucrados
  - Ventajas: fácil de implementar y da sentido de propiedad sobre la estimación
  - Se puede utilizar en etapas tempranas del proyecto
  - Se recomienda utilizarlo en proyectos poco conocidos en donde no se cuenta con historia

Métodos paramétricos

- **Function Points:**
  - Miden el tamaño del SW en base a la funcionalidad capturada en los requerimientos
  - Usa el modelo lógico o conceptual para trabajar
  - En general son aceptados en la industria a pesar de su complejidad y antigüedad
  - Hay muchas heurísticas en el mercado basadas en PFs
  - Los PFs son independientes de la tecnología
  - Requieren un análisis de requerimientos avanzado

1) Identificar los elementos del sistema:
   a) Archivos lógicos internos (ALI o ILF): Grupo de datos lógicos de información o de control, identificables por el usuario, mantenidos a través de procesos elementales de la aplicación dentro de los límites de la misma. *Archivos internos del sistema, tablas*
   b) Archivo de interface externa (AIE o EIF): Grupo de datos lógicos de información o de control, identificables por el usuario, referenciados por la aplicación pero mantenidos a través de procesos elementales de una aplicación diferente. Los EIF no son mantenidos por la aplicación. *Archivos de otra aplicación que el sistema consume*
   c) Entradas externas (EE o EI): Proceso elemental de la aplicación que procesa datos o información de control que entran desde el exterior del sistema. Ingresan al sistema. Los datos procesados mantienen uno o más ILFs. La información de control puede o no ser mantenida en un ILF. *Pantallas o formularios donde se agrega o cambia información.*
   d) Salidas externas (SE o EO): Proceso elemental de la aplicación que envía datos o información de control fuera de los límites del sistema. *Resultado de una entrada, implica una modificación en el sistema.* Puede hacer update de un ILF. Dicho procesamiento de información debe contener al menos una fórmula matemática o cálculo, o crear datos derivados. *Pantalla/reportes de un proceso del sistema*
   Una EO puede también mantener uno o más ILFs y/o alterar el comportamiento del sistema.
   e) Consultas Externas (CE o EQ): Proceso elemental de la aplicación que envía datos o información de control fuera de los límites del sistema. *"Solo obtener información, no se modifica nada".* NO posee fórmulas matemáticas ni cálculos. Y no crea datos derivados. No mantiene ILFs durante su procesamiento, ni altera el comportamiento del sistema. *Consultas*
2) Calcular los FP sin ajustar: Tomando los datos del Paso 1, se realizan cálculos, se suma y obtienen los puntos de función "sin ajustar" (UFP), que son independientes de la tecnología.

Function Type	Low	Average	High
External Input	x3	x4	x6
External Input	x4	x5	x7
Logical Internal File	x7	x10	x15
External Interface File	x5	x7	x10
External Inquiry	x3	x4	x6

	1-5 Data element types	6-19 Data element types	20+ Data element types
0-1 File types referenced	Low	Low	Average
2-3 File types referenced	Low	Average	High
4+ File types referenced	Average	High	High

3) Calcular el technical complexity factor (TCF, o CGS: Características generales del sistema): TCF = 0.65 + (sum de factores) / 100

Hay 14 factores de complejidad técnica. C/U se evalua según el grado de influencia (0-5).
–Data communications
–Performance
–Heavily used configuration
–Transaction rate
–Online data entry
–End user efficiency
–Online update
–Complex processing
–Reusability
–Installation ease
–Operations ease
–Multiple sites
–Facilitate change
–Distributed functions

4) Calcular puntos de función: Puntos de Función Netos: PFN = UFP * TCF.
Y luego se busca el equivalente en el estándar, convirtiendo FP a LOC, 1 FP equivale a:

	Mínimo	Media	Máximo
Java	40 LOC	55 LOC	80 LOC
C++	40 LOC	55 LOC	80 LOC
Cobol	65 LOC	107 LOC	150 LOC
SQL	7 LOC	13 LOC	15 LOC

- **Use case points:**
  - El número de Use Case Points depende de :
    –Cantidad y complejidad de los casos de uso
    –Cantidad y complejidad de los actores intervinientes en el sistema
    –Factores técnicos y ambientales
  - 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.
  - Basado en el método de Function Points
  - Elementos del sistema
    –Casos de Uso
    –Transacciones de Casos de Uso
    –Actores

1) Clasificar actores
   a) Simples: Sistemas externos, muy predecibles, todo sistema con interfaz de aplicación bien definida. Peso: 1
   b) Promedio: Dispositivos de HW Requieren más esfuerzo controlarlos, son más propensos a errores. Y personas interactuando a través de una interfaz basada en texto. Peso: 2
   c) Complejos: Humanos, impredecibles y difíciles de controlar. Personas que interactúan a través de una GUI. Peso: 3

   Se cuenta el número de actores en cada categoría, multiplicando ese número por el correspondiente peso y se obtiene el UAW (Unadjusted Actor Weigth).

2) Clasificar casos de uso
   a) Simple: 3 o más transacciones. Peso: 5
   b) Medio: 4 a 7 transacciones. Peso: 10
   c) Complejo: más de 7 transacciones. Peso: 15

Se cuenta el número de casos de uso en cada categoría, multiplicando ese número por el correspondiente peso y se obtiene el UUCW (Unadjusted Use Case Weigth).

- UAW + UUCW = UUPC Unadjusted use case points.

3) Factores de ajuste: asignar un peso a los factores técnicos o del entorno que pueden influir en el costo de desarrollar el software. A Cada factor se le asigna un valor entre 0 y 5 según la influencia.
   Se suman los distintos Pesos * Valor Asignado para ambos y se obtienen T-Factor y E-Factor respectivamente.
   -Trabajadores part-time y Lenguaje de programación restan

Factores técnicos

Factor	Descripción	Peso
T1	Sistema distribuido	2
T2	Respuesta Rendimiento Performance	2
T3	Eficiencia del usuario final	1
T4	Procesamiento interno complejo	1
T5	Código reusable	1
T6	Fácil de instalar	0.5
T7	Fácil de usar	0.5
T8	Portable	2
T9	Fácil de cambiar	1
T10	Concurrente	1
T11	Incluye características de seguridad	1
T12	Provee acceso a terceras partes	1
T13	Se requieren entrenamiento especial	1

Factores del entorno

Factor	Descripción	Peso
T1	Familiar con RUP	1,5
T2	Experiencia en la aplicación	0,5
T3	Experiencia en objetos	1
T4	Buena capacidad de análisis	0,5
T5	Motivación	1
T6	Requerimientos estables	2
T7	Trabajadores part - time	-1
T8	Lenguaje de programación	-1

4) Calculo de los UCP
   - Factor Técnico De La Complejidad: TCF= 0,6 + (0,01 * T-Factor)
   - Factor Ambiental De La Complejidad: EF= 1,4+(– 0,03 * E-Factor)
   - UCP = UUCP * T-Factor * E-Factor

•Object Points:

- No está relacionado necesariamente con programación orientada a objetos.
- Se asigna a cada componente un peso, de acuerdo a la clasificación por su complejidad.
- Considera como factor de ajuste el porcentaje de reutilización de código.

1) Identificar elementos del sistema: Pantallas, Reportes y Modulos 3GL (Lenguajes de 3ra generación). El método original contempla solo esos 3 tipos, las implementaciones ad hoc del método consideran otros tipos de componentes como Clases Java, Script SQL, etc.
2) Calcular complejidad: Multiplicando por el peso según la complejidad de los componentes (Simple, Medio, Complejo) y se obtienen los Object Points brutos OPB

Object Type	Peso - Complejidad		
	Simple	Medio	Complejo
Screen	1	2	3
Report	2	5	8
3GL Component			10

3) Calcular porcentaje de reusabiliad
   - OP= [OPB * (100 - % Reutilización)] : 100

   (Si porcentaje de reutilización es 20%, se multiplica por 0.8)
   Luego se debe transformar el esfuerzo en personas por mes:
   - NOP (new object points) = OP (100-%reusabilidad)/100
   - –Esfuerzo = NOP/PROD

   PROD (Nivel de productividad) se obtiene de la sig. Tabla:

	Muy Baja	Baja	Media	Alta	Muy Alta
PROD	4	7	13	25	50

•CoCoMo:

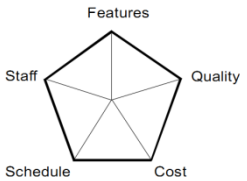
- Modelo empírico basado en recolección de datos de varios proyectos grandes
- Está muy bien documentado, es de dominio público y hay varias herramientas comerciales
- Toma en cuenta el proyecto, el producto, el HW y la experiencia de los RRHH

- Existe una nueva versión, CoCoMO II

No hay un método mejor que otro. En el caso de FP y UCP son parecidos. UCP es menos sentible a cambios, por ejemplo agregando campos a una tabla no se ve el cambio porque contempla transacciones, con FP se pueden ver esos cambios al estimar.
Además podemos tener casos de uso en distintos niveles y puede dificultar ver cierta complejidad.
UCP se puede usar en etapas más tempranas porque no necesita tanto detalles como FP. Se pueden combinar.
OP es más útil para proyectos de mantenimiento, donde puedo aplicar el porcentaje de reutilización.

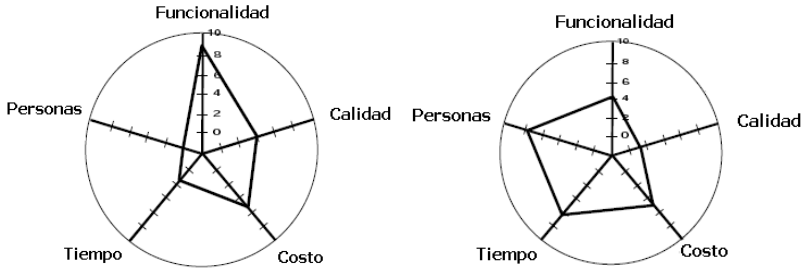
Apunte: Standing on principle – Apoyarse en los principios

Nunca hay que dejar que un jefe o cliente te convenza de hacer un mal trabajo. Si bien hay que escuchar lo que dice y pide el cliente, no hay que caer en solo escuchar su voz. Hay que mirar más allá del requerimiento del cliente y encontrar las verdaderas soluciones para la necesidad real.
Cuando sea posible, mirar más allá del interés del cliente y ver si es posible generalizar alguna funcionalidad para cubrir una necesidad mayor y poder reutilizar en un futuro. Pensar en diseñar con visión de reusabilidad como una inversión que puede replicarse en múltiples proyectos futuros por el bien de la compañía.
Hay que saber ver dónde termina el conocimiento técnico el cliente, y no dejarse intimidar por sus ideas que puedan derivar en malas decisiones de diseño. Los desarrolladores deben basar su relación con los clientes en respeto y confianza mutua. Hay que hacerles ver el impacto real de cualquier modificación de requerimientos sobre la marcha.
También es importante ser honesto con los clientes, ya que para ellos es más valiosa la información real y precisa, aunque sean malas noticias. Si no se pude cumplir algo, sea cual fuese la razón, hay que hablar sobre ello, hacerles saber lo antes posible para que juntos puedan hallat una alternativa.
Tampoco decir mentiras o “lo que el manager quiera oír”. Hay que explicarle los métodos de estimación en que nos basamos, ya que es más difícil que te discutan una estimación basada en un proceso analítico y cuantitativo, que un número sacado de la galera.
Si no se puede proveer una estimación precisa porque falta información, proponer una entrega luego de una exploración inicial que permita estimar. Hacerle notar que una estimación tan temprana puede tener un 80% de error. Si realmente quieren un cronograma acelerado, entonces negociar para obtener un equipo más grande, menos funcionalidad o entrega por fases.
También se puede re-estimar suponiendo otro tamaño del proyecto, otros recursos u otros factores para ver cómo nos podemos acercar a la meta del manager y realizar una contraoferta para mostrarle realmente qué parte de las funcionalidades pueden estar en el plazo que espera.
Las cinco dimensiones del Proyecto de Software: funcionalidad, calidad, costo, cronograma y recursos. Cada una de esas dimensiones cumple un rol en el proyecto y no puede ser el mismo para las 5 al mismo tiempo:



a) Un driver es un objetivo clave para el proyecto. Para un producto que tiene que estar a tiempo para cubrir un nicho de mercado, el tiempo es driver.
b) Una Constraint o restricción es un factor que no está bajo el control del líder del proyecto. Ej: si el tamaño del equipo es fijo, entonces las personas son una restricción.
c) Si no es un Driver ni una restricción es un grado de libertad sobre el que el líder de proyecto puede balancear y ajustar hasta cierto punto para alcanzar los objetivos. Ej: en algunos sistemas de información internos, los drivers son la funcionalidad y la calidad, y las personas son restricción, entonces el costo y el cronograma tienen un grado de libertad.

Las cinco no pueden ser drivers, ni tampoco todas pueden ser restricciones. Este proceso de negociación ayuda a definir las reglas y los límites del proyecto. Una manera de clasificar cada dimensión en una de las tres categorías es pensar en la flexibilidad que tiene el líder respecto a esa dimensión. Una restricción no le da flexibilidad al líder, un driver le da poca flexibilidad, y una con grado de libertad tiene una flexibilidad mayor que las otras cuatro. Un diagrama de Kiviat ayuda a visualizar la dependencia entre relaciones respecto a las otras 4 y sus grados de libertad. Si movemos un eje para reducir una latitud en una dimensión, habrá que ajustar otras dimensiones para compensar: nada llega sin un precio. Otra forma es renegociar cuando todo cambia, si un nuevo requerimiento DEBE ser incorporado, hay que consultar que se debería reajustar para acomodar este pedido.



Apunte: Stop Promising Miracles –Dejar de prometer milagros

WideBand Delphi: Se especifica un problema, y se obtiene una lista de tareas, calidad, procesos y estimaciones anónimas de cada participante de la reunión. Sigue 6 pasos:

1) Planificación: se define el problema y se subdivide en partes más pequeñas. Se le entrega información a cada participante, incluyendo el moderador, proyect manager y otros estimadores.
2) KickOff Meeting: Se hace una reunión inicial de una hora para poner al tanto a los participantes del problema, se revisa y discute buscando problemas de estimación. Se definen las unidades.
3) Preparación Individual: cada participante individualmente da una lista de tareas que deben realizarse para llegar al objetivo. Deben definirse claramente, para que el moderador pueda unificar toda la lista y debe incluir las "tareas soporte" como control de calidad. También especificar las suposiciones.
4) Reunión de Estimación: El moderador junta todas las estimaciones individuales anónimas y arma un gráfico. Cada estimador lee su lista de tareas inicial y dice todas sus suposiciones contestando todas las preguntas que surgen. Como resultado irán apareciendo una lista común de tareas. Luego privadamente cada estimador irá modificando su lista de tareas de acuerdo a su percepción de la reunión previa. Luego el moderador toma estas nuevas estimaciones y las refleja sobre el gráfico anterior. Se repite el proceso hasta que se hayan completado 4 rondas o se haya convergido lo suficiente a un acuerdo común.
5) Assembling Tasks: El moderador toma los listados finales y los combina en una lista final de tareas y tiempos, eliminando duplicados y razonando los casos especiales.
6) Revisión de Resultados: El equipo de estimación revisa el resultado sumarizado y da su aprobación a la salida final. Los participantes deben estar seguros de que la lista final esta lo más completa posible.

La estimación esta completa cuando todo el criterio está satisfecho: las tareas están ensambladas, tenemos una lista sumarizada de supuestos, y los estimadores tienen consenso de que sus estimaciones individuales están sintetizadas en una lista de tareas estándar de

rango aceptable. Deben verse claramente el mejor caso, el peor caso y el planificado (promedio).

Apunte: Artículo: Timebox Development – Desarrollo Timebox

Timebox Development ayuda a darle al equipo de trabajo un sentido de urgencia y a que se enfoquen en las características más importantes. Permite desarrollar todo cuanto se quiera, pero debe dejar como Contraint inamovible el factor tiempo.

Características:

- Enfatiza la prioridad del calendario: lo deja absolutamente fijo, por sobre toda otra consideración.
- Elude el problema de 90-90: evita "90% terminado" eterno. Más que construir una primera versión completa, ayuda a sacar una versión básica rápidamente y luego, con esa experiencia, construir una segunda versión mucho más completa.
- Ayuda a clarificar prioridades: ya que los proyectos gastan mucho tiempo en características que hacen una pequeña diferencia a la utilidad del producto.
- Controla el "Feature Creep": ayuda a reducir los tiempos de los ciclos, y de este modo evitar ser afectados por cambios de condiciones del mercado.
- Ayuda a motivar desarrolladores y usuarios finales: el sentido de urgencia fomenta el "creer que lo que se hace es importante", cosa que a la gente le gusta escuchar.

Uso: Se desarrollan primero lo esencial y más importante y luego los pequeños detalles cuando se pueda. Es importante tener un prototipo funcional primero que vaya evolucionando hacia el sistema final. No puede extenderse un Timebox por más de 60-120 días y es crítico que no se extienda de la fecha final o "deadline". Luego de la construcción se puede optar por: aceptar el resultado y ponerlo en producción, rechazarlo por fallas de construcción y empezar uno nuevo o Rechazarlo por no cumplir las necesidades de la organización.

Tipos de desarrollo donde se puede aplicar: Lista priorizada de características, calendario realista, correcto tipo de proyecto (más apropiado para proyectos pequeños y de lenguajes de desarrollo veloz), requerimientos no volátiles y con usuarios finales involucrados.

Equipo de Proyecto: de 1 a 5 personas. Serán juzgadas si crean un sistema que es aceptado, si es exitoso, serán recompensados y se celebrará.

Variaciones: se puede usar para muchos tipos de actividades, desde la construcción de software hasta la documentación y prototipos.

Riesgos:

- Usarlo para productos inadecuados: por ejemplo, no es recomendable para el plan de proyecto o tareas de "arranque" de la cadena alimenticia, donde no es conveniente escatimar tiempo.
- Sacrificar calidad en lugar de características: si el cliente no quiere cortar características en lugar de calidad, no usar Timebox. Una vez que la calidad empieza a perderse, el calendario también lo hará.

A veces se interactúa con JAD, Herramientas CASE, Prototipos y Entregas Evolutivas, entre otros.

Se sabe que Timebox produce productividad de cerca de 80 puntos de función por mes, comparado con entre 15 y 25 de otras metodologías.

Claves para ser exitoso con Timebox:

- Usarlo solo en proyectos que no se puedan extender de la fecha limite (de 60 a 120 días)
- Mantener al equipo motivado y consciente de la importancia del proyecto.
- En caso de necesitarlo, cortar características pero nunca correr la fecha límite. Nunca.

Apunte: The Mythical Man-Month Cap 2: The mythical Man-Month/El mítico hombre-mes

Muchos más proyectos han fracasado por la falta de tiempo que por el resto de las causas.

1) Técnicas de estimación pobremente desarrolladas, reflejan la asunción implícita de que todo irá bien cuando en realidad no es así.
2) Técnicas de estimación que confunden esfuerzo con progreso, escondiendo la asunción que los hombres y los meses son intercambiables.
3) Inseguridad en las estimaciones
4) El avance del plan es pobremente monitoreado.
5) Cuando se descubre un retraso en el plan, la reacción es agregar gente.

Todos los programadores son optimistas. Entonces la primera falsa suposición en la que se basa todo plan de desarrollo de sistemas es que todo irá bien, que cada tarea tomará el tiempo que fue estimado para ella. El programador construye desde entes ideales: conceptos y representaciones muy flexibles de los mismos. Como el medio es tan flexible, esperamos pocas dificultades en la implementación, he aquí la razón de nuestro constante optimismo. Como nuestras ideas tienen fallas, tenemos bugs, por lo tanto nuestro optimismo es injustificado.

En una sola tarea, la suposición de que todo irá bien tiene un efecto probabilístico en la planificación. Un gran esfuerzo de programación consiste en muchas tareas, algunas encadenadas de punta a punta. La probabilidad de que cada una vaya bien se vuelve demasiado chica.

<u>El mes/hombre</u>

La segunda falla radica en la única unidad de esfuerzo usada para estimar y planificar: el mes/hombre. El costo del producto varía en función de la cantidad de personas y la cantidad de hombres. El progreso no. El hecho que mes/hombre es una unidad para medir el tamaño de un trabajo es un mito peligroso y engañoso. Implica que hombres y meses son intercambiables. Los hombres y los meses son intercambiables solamente cuando una tarea puede ser dividida entre muchos hombres sin comunicación entre ellos. Cuando una tarea no puede ser dividida debido a restricciones secuenciales, la aplicación de más esfuerzo no tiene efecto en el plan. La gestación de un bebe tomas nueve meses, no importa cuántas mujeres se asignen. Muchas tareas del desarrollo de software tienen esta característica por la naturaleza secuencial de la depuración.

En las tareas que se pueden dividir pero requieren comunicación entre las subtareas, el esfuerzo dedicado a la comunicación debe ser agregado a la cantidad de trabajo a realizar. La carga que se agrega debido a la comunicación está conformada por dos partes, capacitación e intercomunicación. Cada trabajador debe ser capacitado en la tecnología, las metas del esfuerzo, la estrategia y el plan de trabajo. Esta capacitación no puede ser dividida, por lo tanto esta parte del esfuerzo agregado varía linealmente con el número de trabajadores. La intercomunicación es aun peor. Si cada parte de la tarea debe coordinar con cada una de las otras partes, el esfuerzo se incrementa.

Como el desarrollo de software es inherentemente un esfuerzo sistemático el esfuerzo dedicado a la comunicación es grande, y rápidamente domina la disminución en el tiempo de las tareas individuales provocada por la división. Agregar más gente alarga el tiempo del proyecto.

<u>Prueba de sistemas</u>

No hay partes de la planificación que se ven tan afectadas por restricciones de secuencialidad como la depuración y la prueba del sistema. El tiempo requerido depende de la cantidad de errores encontrados, en consecuencia las pruebas son la parte que más errores tiene en su planificación.

El error de no asignar suficiente tiempo para las pruebas es un desastre muy peculiar. Como el atraso surge al final del cronograma, nadie está al tanto de esta demora hasta casi llegada la fecha de entrega. Un retraso en este punto tiene usualmente severas repercusiones psicológicas y financieras. El proyecto está lleno de personal y su costo por día es altísimo. Más serio aun, el

software se usa para soportar otras tareas del negocio y los costos secundarios de demorarlas son muy altos, ya que estamos casi al momento de la entrega. Efectivamente, estos costos secundarios pueden superar a los otros. Por esto es muy importante asignar suficiente tiempo para las pruebas de sistema en el cronograma original.

Estimación débil

Realizar planes falsos para satisfacer las fechas deseadas por el jefe es mucho más común en nuestra rama de la ingeniería. Es muy difícil hacer una defensa fuerte creíble de una estimación que no está derivada de ningún método cuantitativo. Se necesitan dos soluciones: desarrollar y publicitar estadísticas de productividad, estadísticas de incidencias, reglas de estimación.

Hasta que la estimación tenga una base más sólida, cada manager tendrá que defender su estimación con la seguridad que sus pobres métodos son mejores que estimaciones derivadas de deseos.

En resumen: “Agregar más gente al proyecto, lo hace terminar mas tarde.”

Esta es la desmitificación del mes/hombre. El número de meses de un proyecto depende de sus restricciones de secuencialidad. El número máximo de personas depende del número de tareas independientes. De estas dos cantidades podemos deducir usar menos hombres y más meses (el único riesgo de estos es la obsolescencia). Uno no puede lograr cronogramas que funcionen usando más gente y menos meses. Muchos más proyectos han fracasado por la falta de tiempo que por el resto de las causas.