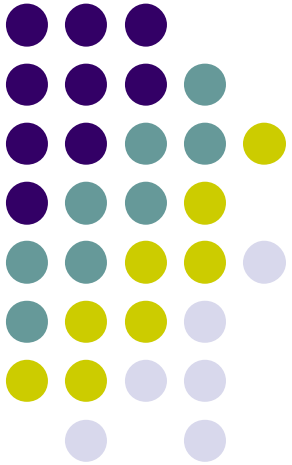
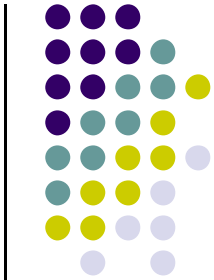
Gestión y Planificación de Proyectos

Gestión de Proyectos

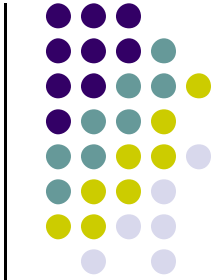


Material bibliográfico

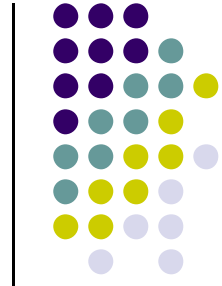


- “Desarrollo y gestión de proyectos informáticos”. Steve McConnell. McGraw-Hill.
  - Contiene bibliografía comentada al final de cada tema (e.g., Estimación, Motivación, Equipos de trabajo y Aumento de productividad).
- “Ingeniería del software. Un enfoque práctico”. Roger S. Pressman. 7ª edición. McGraw-Hill.
- “Software engineering”. Ian Sommerville. 9ª edición. Addison-Wesley.
- “Ingeniería del software. Aspectos de gestión. Tomo 1: Conceptos básicos, teoría, ejercicios y herramientas”. Román López-Cortijo y García y Antonio de Amescua Seco. Instituto Ibérico de la Industria del Software (www.iiis.es).
- “La calidad del software y su medida”. Jesús Mª Minguet Melián y Juan F. Hernández Ballesteros. Editorial Centro de Estudios Ramón Areces.
- “Calidad de sistemas informáticos”. Mario G. Piattini Velthius, Félix O. García Rubio e Ismael Caballero Muñoz-Reja. Ra-Ma.
- “Project management práctico. Técnicas, herramientas y documentos”. J. Eduardo Caamaño. Ed. Círculo rojo-Docencia. (www.pmpractico.com)
- “Interfaces, técnicas y prácticas. MÉTRICA versión 3”. Ministerio de las Administraciones Públicas: http://www.csi.map.es/csi/metrica3/.
- International Function Point Users’ Group (IFPUG): http://www.ifpug.org.

Índice

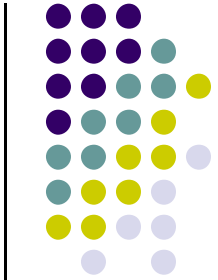


- Introducción.
- Ciclo de vida de la gestión de proyectos.
- Dimensiones de un proyecto software.
- Pilares fundamentales.



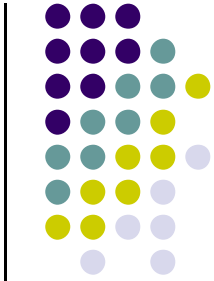
Introducción

Introducción (I)



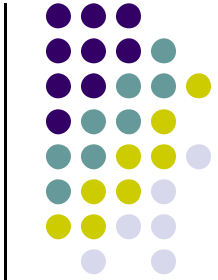
- Factor de calidad en el ámbito del software:
  - Requisito fundamental de ISO 9000 (i.e. de sus guías de aplicación al desarrollo software).
  - La planificación y seguimiento de proyectos son dos áreas de proceso del nivel 2 de CMMI, en donde se busca sólo la repetitividad.
- Problema:
  - Resulta duro aprender sobre temas de calidad trabajando todo el día para sacar adelante los proyectos.
    - Círculo vicioso: Parar para planificar, no parar para acabar.
- Solución:
  - Controlar los proyectos (planificación) y
  - Controlar las planificaciones (seguimiento).

Introducción (II)



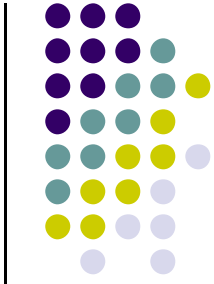
- Crisis del software:
  - Durante años el desarrollo de software se ha considerado un “arte”.
  - El director (o jefe de proyecto) lo dirigía más bajo consideraciones técnicas que de gestión.
- Problemas originados:
  - Desviaciones en costes y tiempos inaceptables.
  - Calidad baja o muy baja en sistemas vitales.
  - Incremento en la complejidad de los sistemas.
- Conclusiones:
  - La ingeniería del software sólo no es la solución si no se considera la gestión, planificación y seguimiento de proyectos.
  - Necesidad de una metodología para organizar, planificar y controlar los proyectos software.

Introducción (III)



- Dirigir un proyecto software requiere, fundamentalmente y entre otras cosas:
  - Procesos de ingeniería del software y de gestión bien definidos y aplicados.
  - Metodologías, técnicas y herramientas adecuadas y conjuntadas.
  - Buena especificación de requisitos por parte del usuario.
  - Planificación adecuada con calendarios realistas.
  - Presupuestos correctos, ajustados y proporcionados.
  - Personal formado, motivado y adecuado organizado en buenos y equilibrados equipos.
  - Control/coordinación.
  - Buena comunicación/información.
  - Gestión de riesgos.

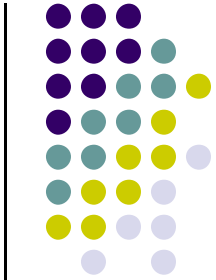
Introducción (IV)



- Si no hay procesos repetibles, que emplee toda la organización, no existe una base para la mejora continua, tanto de la calidad como de la productividad.

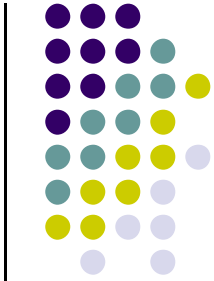
- Organización inmadura:
  - No hay un proceso definido y riguroso.
  - No hay estimaciones realistas, por lo que la planificación no se cumple.
  - Si se impone un calendario, la calidad y la funcionalidad se resienten para lograrlo.

- Organización madura:
  - Hay habilidad generalizada para gestionar los procesos de desarrollo software.
  - El proceso es conocido y todo se desarrolla de acuerdo con lo planificado.
  - Los procesos son consistentes (responden a lo esperado).
  - Las responsabilidades y papeles están claramente definidos.
  - La planificación está basada en datos históricos y es realista.



Ciclo de vida de la gestión de proyectos

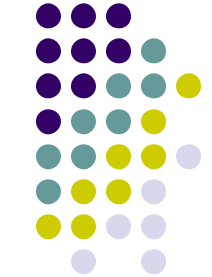
Definiciones



- Proyecto:
  - Acción iniciada por la empresa,
  - en la que recursos humanos, financieros y materiales se organizan de una nueva forma
  - para acometer un trabajo único,
  - en el que dadas unas especificaciones y dentro de unas limitaciones presupuestarias,
  - se intenta conseguir un cambio beneficioso definido por unos objetivos cualitativos y/o cuantitativos.

- Gestión de proyectos:
  - Sistema de procedimientos, prácticas, tecnologías y conocimientos que
  - facilitan la organización, gestión, dirección, planificación y control necesarios para que el proyecto termine con éxito.

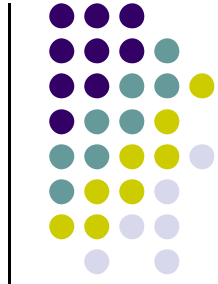
Ciclo de vida (I)



- La forma en que se aplican los procesos de gestión cambia conforme el proyecto avanza.
- Existen diferentes modelos de ciclo de vida con variación en el número de estados propuestos.
- Ejemplo:

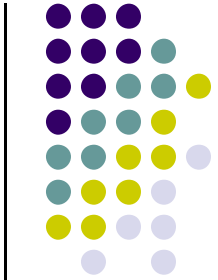
ETAPA	**NOMBRE**	**OBJETIVO DE GESTIÓN**
Germinación	Propuesta e iniciación	- Definición del proyecto - Alcance y objetivos - Diseño funcional - Viabilidad - Estimación inicial - Decisión de continuar o no

Ciclo de vida (II)



ETAPA	**NOMBRE**	**OBJETIVO DE GESTIÓN**
Crecimiento	Diseño y validación	- Diseño del sistema - Producto base - Estimación - Planes y recursos - Validación del diseño
Madurez	Ejecución y control	- Formación y comunicación - Planificación detallada y diseño - Control estimación - Asignación de trabajo - Seguimiento del progreso - Terminación de las previsiones - Control y recuperación
Muerte	Finalización y cierre	- Terminación del trabajo - Uso del producto - Consecución de beneficios - Disolución/recompensa del equipo - Auditoría y revisión - Registro histórico de los datos

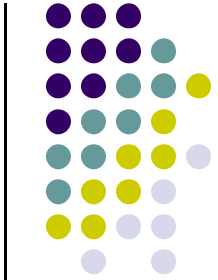
Definición del proyecto



- El contenido del informe de definición de un proyecto debería considerar los siguientes apartados principales:
  - Definición: Qué se pretende en términos de resultados finales.
  - Justificación: Si el esfuerzo de desarrollo justifica la inversión.
  - Objetivos y metas: Servirán para determinar si el proyecto concluye con éxito.
  - Factores críticos de éxito: Aquellos que harán que el proyecto concluya con éxito o fracase.
  - Riesgos: Identificar las posibles fuentes de problemas.
  - Alcance del proyecto: Fija las fronteras del proyecto.
  - Estructura del proyecto: Se subdividirá si la complejidad lo exige.
    - Cada subproyecto debe incluir título, objetivo, alcance, producto final, hitos, riesgos/dependencias y responsables.
  - Fecha prevista de finalización.
  - Fases, actividades, tareas e hitos.
  - Organización: Estructura organizativa y responsabilidades.
  - Sistema de control: Forma de revisar el trabajo en el proyecto.
  - Supuestos, dependencias y restricciones: Supuestos sobre el entorno de desarrollo, dependencias con otros proyectos, etc.
  - Presupuesto y gestión del presupuesto: Presupuesto disponible, forma de gestionarlo y responsable.
  - Proyectos relacionados.

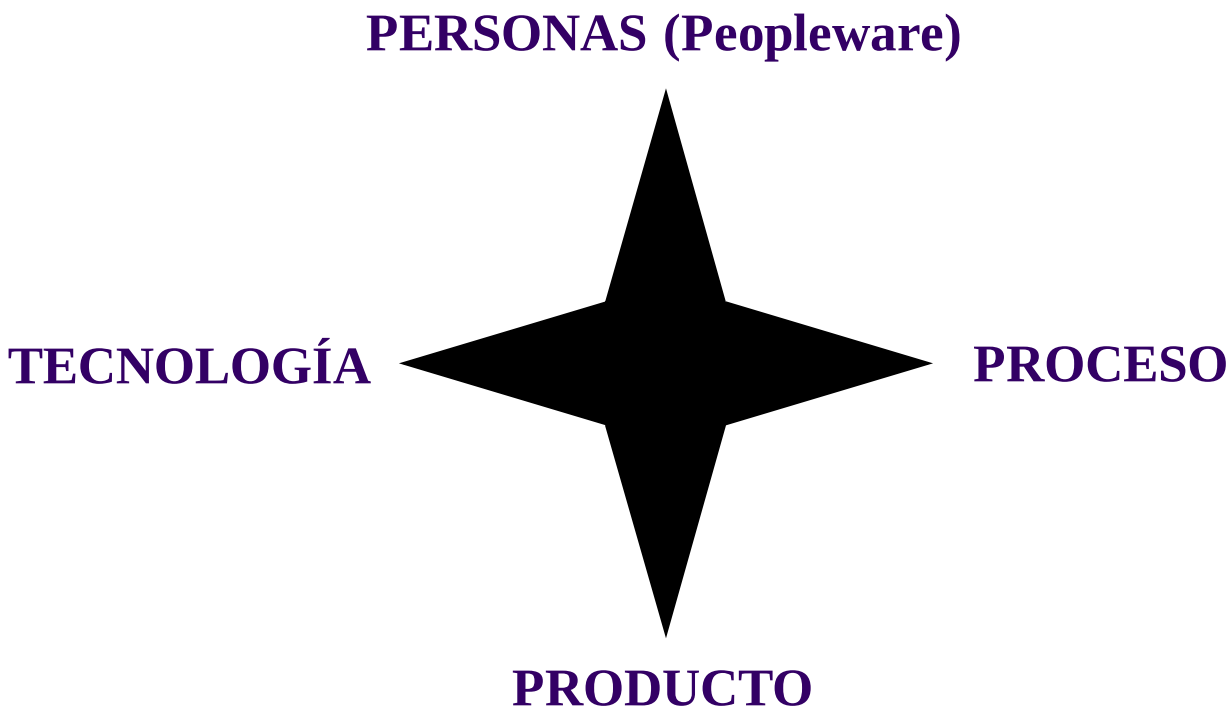
Finalización del proyecto

- Hay que asegurarse de que el trabajo se ha completado a tiempo y de forma eficiente. Para ello se debe finalizar formalmente el proyecto:
  - Finalización del trabajo:
    - Planificar y controlar los trabajos: Listas de trabajos a realizar y realizados.
    - Establecer reuniones de control.
    - Disolver el equipo y cerrar contratos con terceros.
  - Transferencia del producto a los usuarios:
    - Planificar la transición.
    - Aceptación del producto por los usuarios.
    - Formar a los usuarios.
    - Garantizar el mantenimiento del producto y niveles de servicio.
  - Obtención de beneficios:
    - Establecer una medida de referencia.
    - Calcular variaciones entre lo medido y lo establecido.
    - Tomar acciones correctoras para eliminar las variaciones.
  - Disolución del equipo:
    - Planificar la disolución del equipo y después devolver los recursos a su origen.
    - Mantener una reunión de final de proyecto.
    - Repartir premios, “castigos”, evaluaciones y otras consideraciones.
  - Revisiones post-finalización:
    - Configuración final.
    - Comparar costes y beneficios finales para retroalimentar el proceso (históricos).
    - Registrar lo logros técnicos del proyecto para realimentar el proceso de desarrollo.
    - Revisar éxitos y fracasos del proyecto para el futuro.



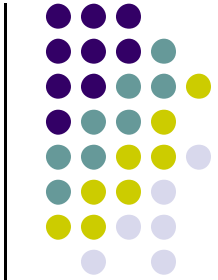
Dimensiones de un proyecto software

4 dimensiones



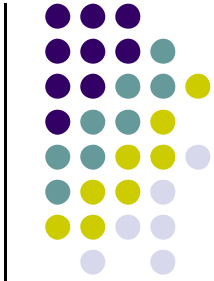
- Si se potencian estas cuatro dimensiones:
  - Se maximiza la velocidad de desarrollo.
  - Se aumenta la calidad.
  - Además, la planificación será más completa, creativa, efectiva y satisfará mejor a todos.
- A continuación se considerarán brevemente …

Personas



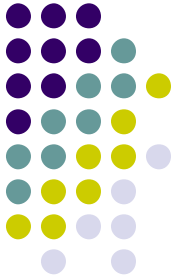
- Todos los aspectos relacionados con las personas tienen un gran impacto en la productividad del software y en su calidad.
- La tecnología exclusivamente no es la respuesta:
  - Los métodos más efectivos son aquellos que sacan partido al potencial humano de los desarrolladores.
- La mejora en la productividad conlleva ocuparse de temas relacionados con el personal:
  - Motivación.
  - Equipo de trabajo.
  - Selección de personal.
  - Formación.
  - Gestión de personal.
- Barry Boehm en su libro “Software engineering economics” (1981) propone:
  - Máximo talento: Usar poco y buen personal.
  - Trabajo adecuado: Explotar habilidades y motivar a la gente.
  - Progresión profesional: Actualización vs. encajonamiento y monotonicidad.
  - Equilibrado del equipo: Seleccionar a gente que se complemente y armonice con los demás.
  - Eliminar inadaptación: Reemplazar a los miembros problemáticos del equipo lo antes posible.

Proceso



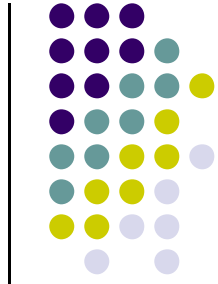
- Para la mejora de esta dimensión se debe considerar la aplicación de:
  - Metodologías de desarrollo:
    - Metodologías.
    - Estándares.
    - Procedimientos.
    - Técnicas.
    - Herramientas CASE.
  - Gestión y planificación de proyectos.
- Myers en 1992 y Gibbs en 1994 ponen como ejemplos de mejora en el proceso de desarrollo a:
  - Hughes Aircraft.
  - Lockheed.
  - Motorola.
  - Nasa.
  - Xerox.
- Dichas empresas:
  - Han reducido sus plazos de salida al mercado a la mitad.
  - Han reducido costes y errores (y por ende mantenimiento) en un factor de 3 a 10.

Producto



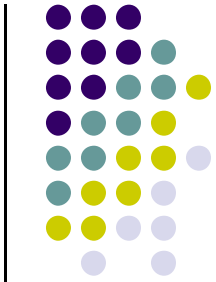
- Ocuparse del tamaño y características del producto software da oportunidades de reducir la planificación:
  - Productos grandes: Más tiempo; Productos pequeños: Menos tiempo.
  - Más prestaciones: Más análisis, diseño,… → más tiempo y costes.
- Intentar aplicar las reglas siguientes (sin tomarlas al pie de la letra):
  - 80/20: Desarrollar el 80% del producto empleando el 20% del tiempo.
  - 3x3: Desarrollar el producto en 3 meses trabajando 3 personas.
- El esfuerzo para construir software se incrementa desproporcionadamente más rápido que el tamaño del software.
  - La reducción del tamaño mejorará la velocidad del desarrollo desproporcionadamente:
    - Reducir a la mitad el tamaño de un programa intermedio normalmente supone una reducción de al menos dos tercios del esfuerzo.
- Objetivos ambiciosos de rendimiento, robustez y fiabilidad suponen más tiempo y más costes. Esto lleva a:
  - Desarrollar sólo las prestaciones esenciales.
  - Desarrollar el producto por etapas, incrementalmente.
  - Emplear lenguajes de alto nivel.
  - Emplear herramientas CASE, técnicas y metodologías adecuadas.

Tecnología



- Se deben emplear herramientas efectivas en cada fase del proceso software:
  - Herramientas CASE de análisis y diseño.
  - Herramientas de pruebas.
  - L4G.
  - Etc.

- Esto es, por ejemplo:
  - Requisite Pro vs. documentos (en el mejor de los casos).
  - Erwin, Bpwin vs. scripts o programar.
  - Reutilización vs. repetición.
  - Delphi vs. ensamblador.
  - Etc.



Pilares fundamentales

Datos reales

- Lederer y Prasad (en 1992) Gibbs (en 1994) y Standish Group (en 1994) identificaron que:
  - Alrededor de dos tercios de todos los proyectos superan **ampliamente** sus estimaciones.
- Capers Jones (en 1994) indicó que:
  - Los grandes proyectos se retrasan en su fecha de entrega entre un 25% y un 50% de su duración.
  - El retraso medio se incrementa con el tamaño del proyecto.
- Symons (en 1991) y McConnell (en 1996) propusieron:
  - Desarrollo rápido (vs. desarrollo lento y típico con largas planificaciones).
  - En definitiva, proyectos pequeños.

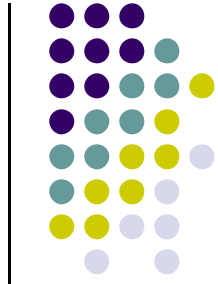
En un proyecto se debe evitar …

- Jefe:
  - “Quiero desarrollar un producto preparado para el cambio. Deseo tener en cuenta la calidad y futuras prestaciones, controlar su planificación y tener una fecha de entrega predecible.”

- Mente del técnico:
  - “La prioridad real es poner el producto a la venta lo antes posible. Pero…
    - ¿Qué producto?: Da igual, ya vamos retrasados.
    - ¿Usabilidad?: No hay tiempo.
    - ¿Rendimiento?: Puede esperar.
    - ¿Facilidad del mantenimiento?: Para la próxima.
    - ¿Verificación?: El producto es para ayer; hay que acabar como sea.”

- En definitiva, un proyecto debe comenzar (**y hacerlo formalmente**) cuando se sepa lo que se quiere abordar a través de ese proyecto (i.e., lo que hay que hacer).
  - No cumplir esto suele ser origen de proyectos “bola de nieve”: productos inicialmente “muy sencillos” (i.e., proyectos pequeños) cuya funcionalidad se modificó y amplió sin control, igual que su planificación.

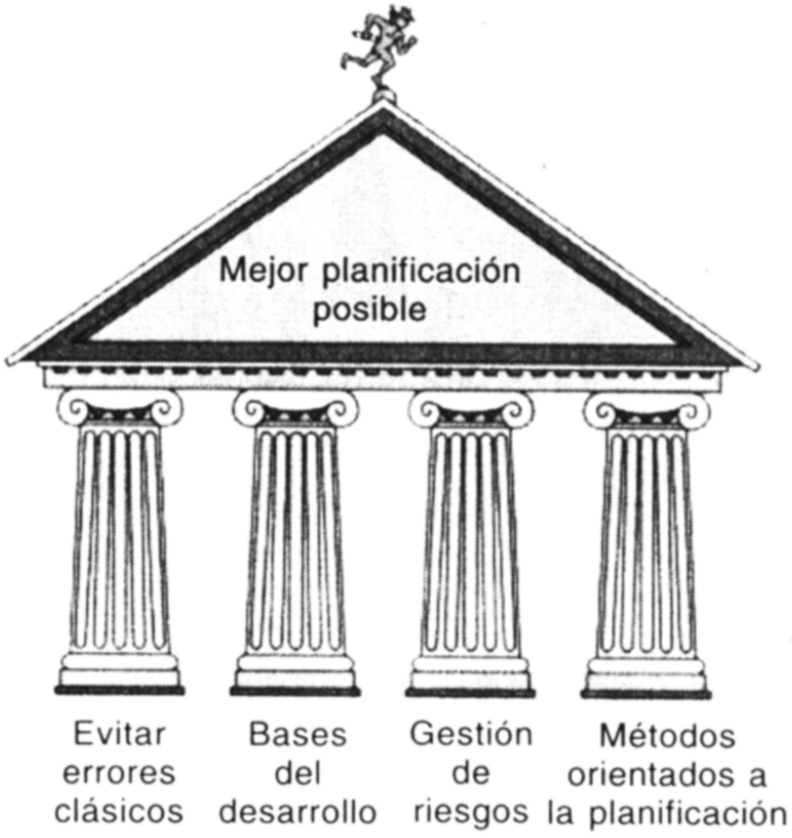
El camino a seguir es …



- Gestión y planificación de proyectos.
- Es un camino poco transitado, pero el más concurrido es el que actualmente redunda en:
  - Costes y retrasos masivos, incluso con proyectos cancelados.
  - Baja calidad.
  - Muchos cambios de personal y fricciones entre personas.
  - Etc.
- Los beneficios, contrastados, que principalmente aporta son:
  - Controlar tiempos y costes del proyecto.
  - Proyectos de alta calidad.
  - Incrementar productividad al conocer capacidades y sobrecargas.
  - Reducir sensación de incompetencia.
  - Mejora la imagen externa y seriedad en el trabajo.
- Es un remedio a muchos males (no a todos) pero requiere:
  - Tiempo (de 3 a 4 años).
  - Esfuerzo (de la dirección y de los trabajadores).
  - Formación.
  - Herramientas de soporte.
- Pero no hay otra solución. Como dijo McConnell en 1996:
  - Las soluciones simples tienden a funcionar sólo con problemas sencillos, y el desarrollo de software no lo es.

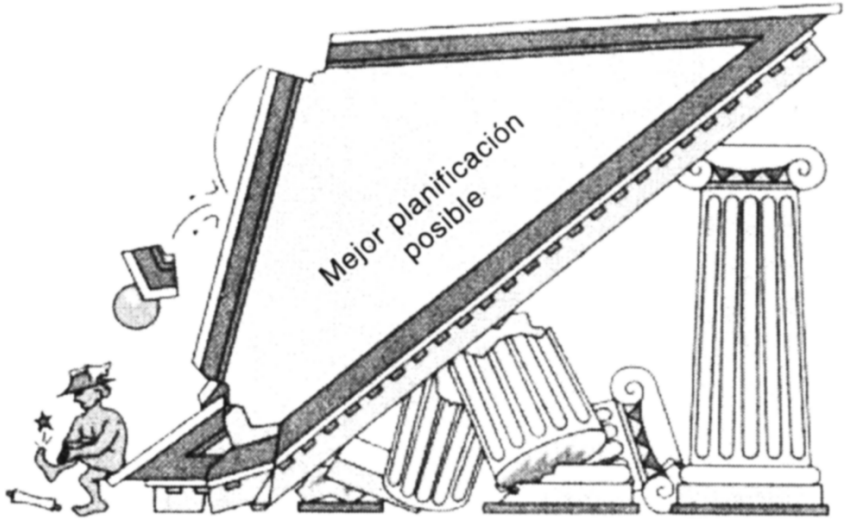
Pilares fundamentales (I)

- Se puede obtener un desarrollo rápido, económico y de calidad basándose en los cuatro pilares que McConnell propone en su libro “Desarrollo y gestión de proyectos informáticos”:



Pilares fundamentales (II)

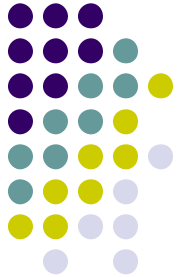
- Pero cuidado, porque ni los métodos orientados a la planificación más avanzados s el mejor plan posible:



Métodos orientados a la planificación

- Cometer errores clásicos, no actuar según los buenos principios de desarrollo o no gestionar los riesgos provocará retrasos y costes asociados.
  - Estos tres pilares son la mayor parte de lo necesario para obtener el mejor plan posible. La planificación sola no sirve para nada.

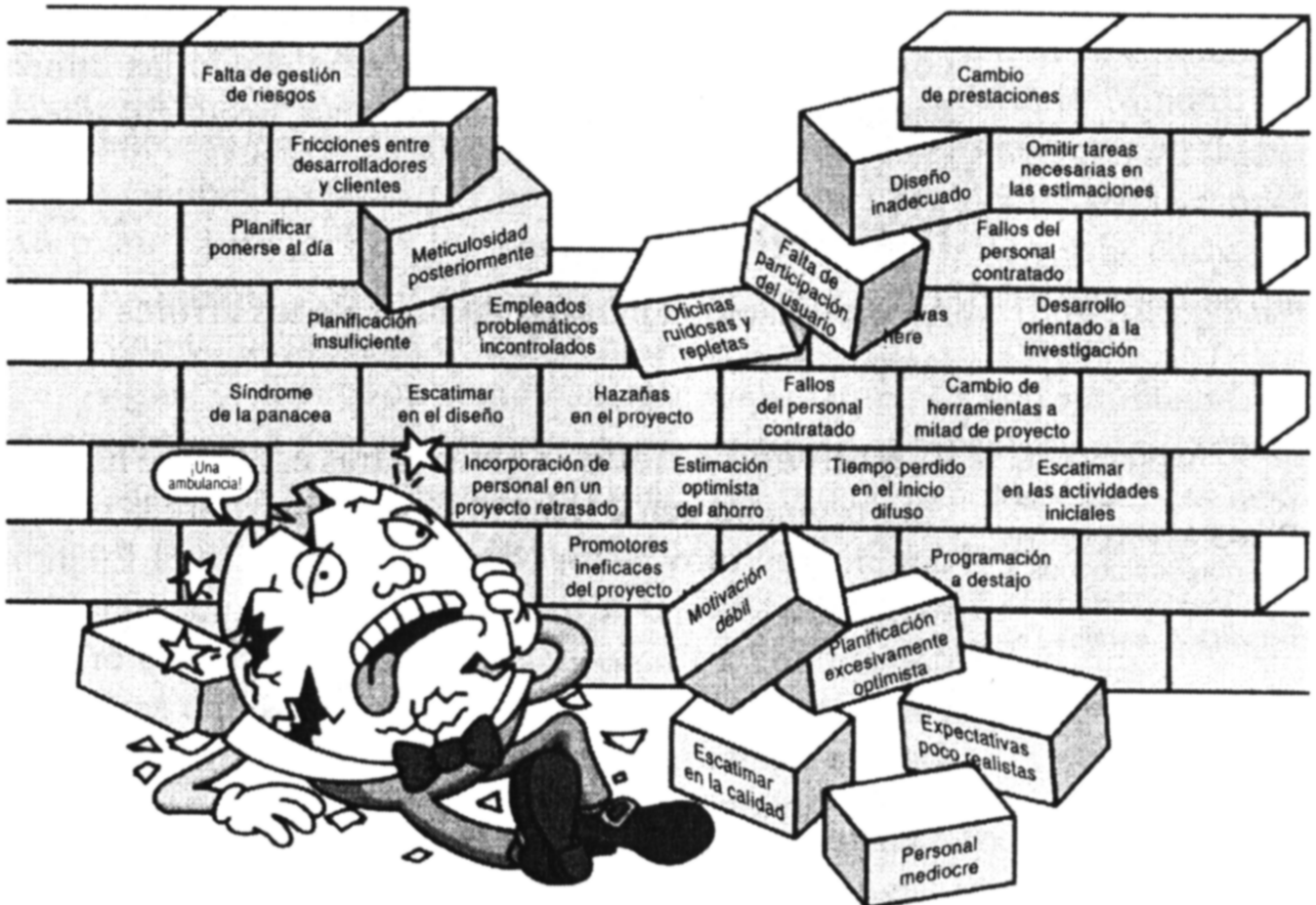
Necesidad de los 3 primeros pilares



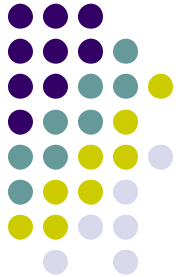
- Pilar 1 – Evitar errores clásicos:
  - El error clásico en desarrollo de software de descuidar la calidad del proyecto en sus fases iniciales supone desperdiciar tiempo y dinero más adelante corrigiendo defectos, justo lo más caro:
    - Retraso en tiempo y aumento en costes.
- Pilar 2 – Bases del desarrollo:
  - Si no se analiza, diseña y codifica por ese orden y de forma adecuada, el sistema fallará cuando cambie la concepción del producto durante el desarrollo:
    - Retraso en tiempo y aumento en costes.
- Pilar 3 – Gestión de riesgos:
  - Si no se controlan los riesgos se puede descubrir muy tarde que un subcontratista , por ejemplo, se ha retrasado:
    - Retraso en tiempo y aumento en costes.
- A continuación se abordan estos pilares …

Pilar 1: Evitar errores clásicos (I)

- Todo influye en el desarrollo de software y un error puede dar al traste con to



Pilar 1: Evitar errores clásicos (II)



- Errores clásicos relacionados con las personas:
  - Motivación débil: Recompensas inadecuadas, no reconocimiento, horas extras, ...
  - Personal mediocre: Contrataciones inadecuadas, personal problemático, …
  - Hitos: Sólo fijarse en que se cumplan los hitos y no cómo va el proyecto día a día.
    - Hay personas que se rigen exclusivamente por hitos: ¡¡ CUIDADO !!
  - Añadir más personal a un proyecto retrasado.
  - Oficinas repletas y ruidosas: Se requiere silencio y privacidad.
  - Fricciones entre clientes/usuarios y desarrolladores.
  - Expectativas poco realistas: Los directivos suelen planificar de forma muy optimista.
    - Esto al final se percibe como un plan muy largo y malo.
  - Falta de participación de los implicados.
  - Falta de participación del usuario: Cuando no se implica al usuario desde el principio, no se entienden ni se consideran sus requisitos.
    - Esto originará retrasos y decepciones en el equipo de desarrollo.
  - Ilusiones y confianza en la bondad de destino: Se cree que el destino deparará algo mejor de lo que se piensa.
    - Generalmente es justo lo contrario.

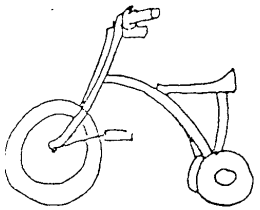
Pilar 1: Evitar errores clásicos (III)

- Errores clásicos relacionados con el proceso:
  - Inicio difuso.
  - Escatimar en las actividades iniciales: Análisis y diseño principalmente.
  - Control insuficiente de la dirección.
  - Programación a destajo, sin métodos técnicas o estándares.
  - Omitir tareas necesarias en la estimación.
  - Fallos en las subcontrataciones:
    - Los subcontratistas suelen entregar el trabajo tarde, deficiente y sin cumplir con las especificaciones (así lo estudió Boehm en 1989).
  - Planificación excesivamente optimista:
    - Optimista sí, pero los retos suelen no cumplirse. Se acaba minando la moral y la productividad.
  - Planificación insuficiente.
  - Gestión de riesgos insuficiente.
  - Abandono de la planificación bajo las presiones:
    - El fallo no es abandonar el plan. Es no crear un plan alternativo.
  - Hacer una gestión de requisitos poco cuidadosa:
    - Los problemas derivados de la comunicación al establecer los requisitos son uno de los mayores problemas como se puede ver irónicamente a continuación:

Pilar 1: Evitar errores clásicos (IV)

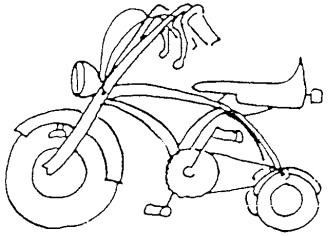
- Problemas de comunicación entre usuarios y desarrolladores:

Lo que se quiere:

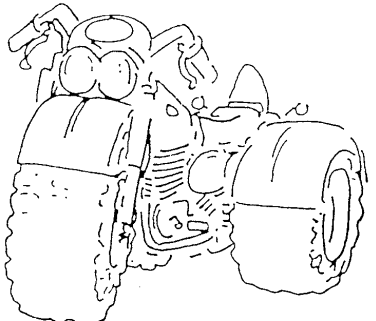


Distorsión

Lo que se diseña:

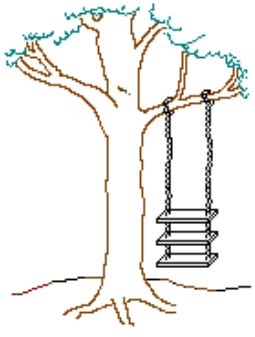


Lo que se desarrolla:

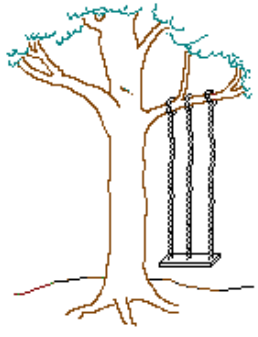


Pilar 1: Evitar errores clásicos (V)

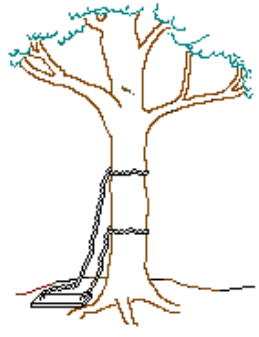
- Problemas de comunicación entre miembros del equipo de desarrollo:



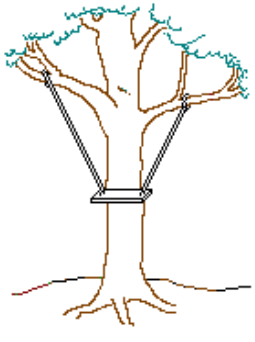
1. Lo que el director desea.



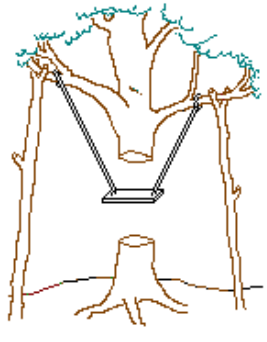
2. Como lo define el director de proyecto.



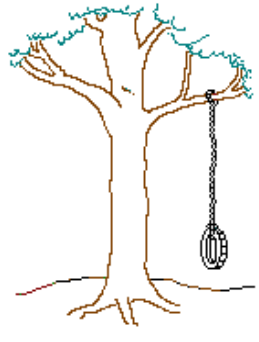
3. Como se diseña el Sistema.



4. Como lo desarrolla el programador.



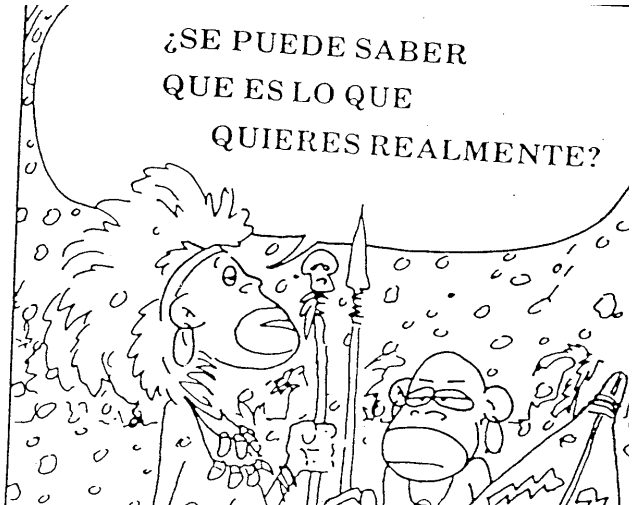
5. Como se ha realizado la instalación.



6. Lo que el usuario quería.

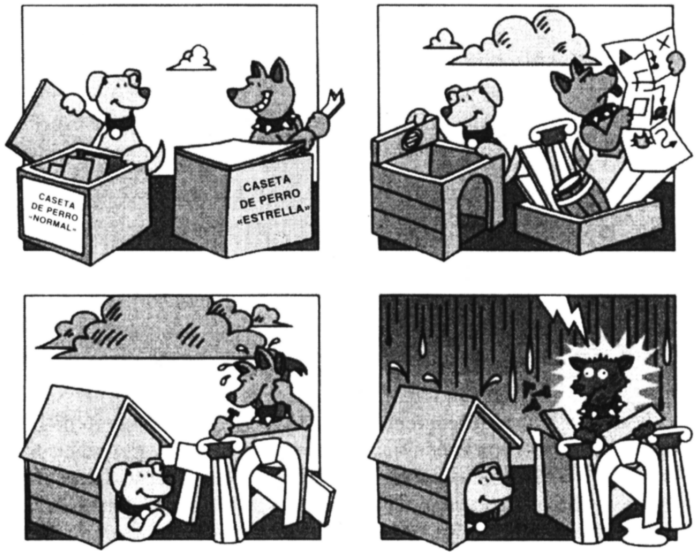
Pilar 1: Evitar errores clásicos (VI)

- Cada requisito una interpretación:

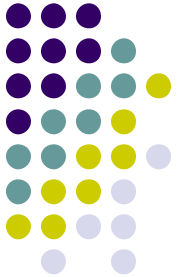


Pilar 1: Evitar errores clásicos (VII)

- Errores clásicos relacionados con el producto:
  - Desarrolladores meticulosos y fascinados por la tecnología.
  - Desarrollo orientado a la investigación:
    - No intentar sobrepasar los límites de la ingeniería en más de dos áreas a la vez. El riesgo de fallo es demasiado alto.
  - Cambios en las prestaciones:
    - Según Capers Jones en 1994, como media hay un 20% de cambios en los requisitos a lo largo del ciclo de vida de un proyecto.
    - Este porcentaje conlleva un 25% de aumento en el plan.
  - Exceso de requisitos: Sólo los absolutamente necesarios:

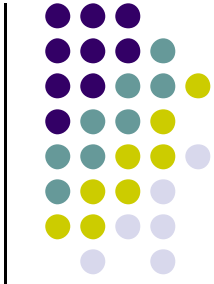


Pilar 1: Evitar errores clásicos (VIII)



- Errores clásicos relacionados con la tecnología:
  - Síndrome de la panacea ante una tecnología.
  - Sobreestimación de las ventajas del empleo de nuevas herramientas o métodos:
    - Hay curvas de aprendizaje, las mejoras serán lentas, …
  - Cambiar de herramientas o filosofía de desarrollo a mitad del proyecto.
    - El ejemplo es claro si, por ejemplo, se pretende aplicar OO cuando el análisis se ha efectuado según el paradigma estructurado.

Pilar 2: Bases del desarrollo (I)

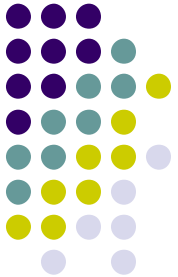


- Se puede planificar bien, pero no considerar los conceptos básicos del desarrollo de software supondrá no alcanzar las metas planificadas.
  - Los métodos fundamentales de ingeniería del software deben emplearse porque reducen el coste y el tiempo de comercialización.

- Se ha observado (Burlton, 1992: “Managing a RAD project: Critical factors for success”) que implantar la disciplina de la ingeniería del software antes que la de gestión de proyectos es un fracaso:
  - Por eso a continuación se aborda primeramente la gestión de proyectos y luego el desarrollo de software propiamente dicho.

- Con respecto a la gestión, ésta controla la planificación, el coste y el producto y sus fundamentos consisten en:
  - Determinar el tamaño del producto.
  - Asignar los recursos apropiados a un producto de ese tamaño.
  - Crear un plan para poner en operación esos recursos.
  - Controlar el plan dirigiendo los recursos para impedir desvíos.

Pilar 2: Bases del desarrollo (II)



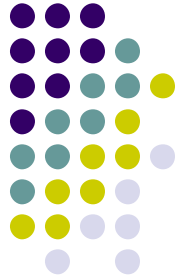
- Los proyectos bien ejecutados pasan por tres etapas básicas para crear una planificación software:
  - Estimación del tamaño del producto.
  - Estimación del esfuerzo necesario para un producto de ese tamaño.
  - Realización de una planificación, basándose en la estimación del esfuerzo.

- Estimación y planificación son las bases del desarrollo.
- Una mala estimación reduce eficiencia en el desarrollo.
  - Una estimación correcta es necesaria para una planificación efectiva y un desarrollo eficiente.
- Para ello hay:
  - COCOMO.
  - Puntos (de) función o Function Points.
  - Etc.

Pilar 2: Bases del desarrollo (III)

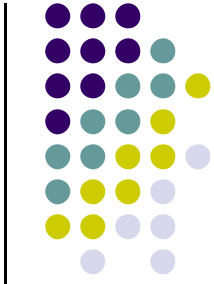
- Hetzel, en 1993, revisó un grupo de los “mejores proyectos” seleccionados por las propias empresas:
  - Se encontró que los mejores se caracterizaban por una fuerte planificación anticipada para definir las tareas y programaciones.
- Pero la planificación no basta:
  - Debe hacerse un seguimiento del proyecto para comprobar que se está cumpliendo lo previsto en objetivos de planificación, coste y calidad.
- En los “mejores proyectos”, Hetzel descubrió que se había hecho una estricta medición y seguimiento del estado del proyecto.
- Capers Jones, en 1995:
  - “El control del proceso software es tan malo que muchos desastres software no se conocen hasta el mismo día del despliegue esperado”.
- El SEI y Kitson y Masters, en 1993:
  - “El 75% necesitan mejorar la supervisión y el seguimiento del proyecto”.
- Baumert, en 1995:
  - “En las evaluaciones de las empresas, los principales problemas aparecen en las áreas de planificación, seguimiento y supervisión del proyecto”.
- Se necesita el seguimiento:
  - Reuniones periódicas.
  - Informes periódicos sobre el estado del proyecto.
  - Revisiones de hitos.
  - Informes de presupuestos.

Pilar 2: Bases del desarrollo (IV)



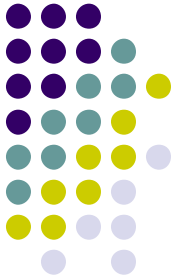
- Medidas o métricas e históricos:
  - Una forma de progresar, a largo plazo, es recoger datos métricos para analizar la calidad y productividad del software.
  - Se deben recoger datos de:
    - Tamaño de los programas (líneas de código: LOC).
    - Planificación.
    - Tiempo.
    - Costes.
  - Recoger más datos puede suponer mucho tiempo.
  - Con los anteriores se tienen las bases para la planificación de proyectos futuros, que siempre es mejor que el instinto para responder a preguntas del estilo:
    - ¿Se puede desarrollar este producto en 3 meses?
      - Nunca hemos desarrollado un producto de ese tamaño en menos de 11 meses y el tiempo medio es de 13 meses.

Pilar 2: Bases del desarrollo (V)



- Con respecto al desarrollo software propiamente dicho:
  - La peor decisión es ahorrar tiempo reduciendo el poco tiempo que ya se le dedica al análisis, al diseño, a las revisiones de código, a la planificación y a las pruebas.
  - Todo funcionará mejor si no se cometen los errores en el primer paso.
  - Si un software tiene demasiados errores, se emplea más tiempo corrigiéndolo que escribiéndolo.
    - Gartner Group estimó que en 1995 el 95% de los costes fueron a mantenimiento.
  - Capers Jones:
    - IBM fue la primera compañía en descubrir que la calidad y la planificación del software estaban relacionadas.
    - También descubrieron que los productos con el menor número de defectos eran los que tenían una mejor planificación.
    - El número de defectos (baja calidad) que se dan cuando los programadores lanzan productos bajo una excesiva presión en la planificación es hasta 4 veces mayor.

Pilar 2: Bases del desarrollo (VI)



- Conclusión: Seguir “las instrucciones de uso”, como en cualquier otro ámbito.

- Por ejemplo:
  - Objetivo: Pintar la caseta del perro.
  - Pasos: Lectura de las instrucciones de la lata de pintura:
    - Preparar superficie.
    - Con superficie seca, aplicar capa fina a una temperatura entre 15 y 30ºC. Dejar secar 2 horas.
    - Aplicar 2ª capa fina y dejar 24 horas antes de utilizarla.
  - Si hay prisa, no se siguen las instrucciones y se va al paso 3 directamente y sin dejar secar.
    - Si la caseta es de madera, no está muy sucia y el tiempo es bueno, puede que todo vaya bien. Después de unos pocos meses la pintura se agrietará y habrá que volver a pintar: más costes y más tiempo.
  - ¿Qué pasaría si fuese un boeing 747 y no se siguiesen las instrucciones?
    - Seguridad deficiente y eficiencia nula.
    - Además, una capa de pintura de un 747 son de 200 a 400 kilos.
    - No preparar la superficie bien haría que el viento, a la velocidad de crucero, y la lluvia atacasen la pintura.

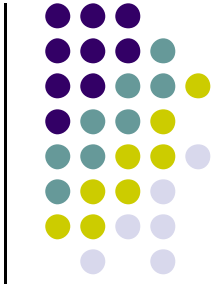
Pilar 3: Gestión de riesgos (I)

- Los proyectos de software incluyen un amplio rango de riesgos:
  - Cambios en los requisitos de usuario.
  - Falta de experiencia en la gestión.
  - Mala estimación de la planificación.
  - Personal contratado poco fiable y efectivo.
  - Problemas con la tecnología, con el desarrollo y/o con los proveedores.
  - Etc.
- Probabilidades en el mundo real:
  - De finalizar un proyecto complejo en el tiempo estimado: **Tiende a 0**.
  - De cancelar un proyecto complejo: **Tiende a 0.5**.
- Peat Marwick, en 1988:
  - De 600 empresas, el 35% ha tenido al menos un proyecto que se le fue de las manos.
  - Ejemplo:
    - Allstate comenzó a automatizar en 1982 sus actividades administrativas. Planificaron 5 años y 8 millones.
    - 6 años más tarde y 15 millones después Allstate estableció una nueva fecha límite y estimó 100 millones de dólares.
- Tom Gilb: “Si no controlas los riesgos, ellos te controlarán a ti”.

Pilar 3: Gestión de riesgos (II)

- La función de la gestión de riesgos es:
  - Identificar, estudiar y eliminar las fuentes de riesgo antes de que empiecen a amenazar la finalización satisfactoria de un proyecto.
- Según Pressman, se pueden controlar los riesgos a varios niveles (si se está en los niveles 1, 2 o 3 se ha perdido la batalla de la planificación):
  - Nivel 1: Control de crisis.
    - Controlar los riesgos sólo cuando se han convertido ya en problemas: Actuar de bombero apagando el fuego.
  - Nivel 2: Arreglar cada error.
    - Detectar y reaccionar rápidamente ante cualquier riesgo, pero sólo después de haberse producido.
  - Nivel 3: Mitigación de riesgos.
    - Planificar con antelación el tiempo que se necesitaría para cubrir riesgos en el caso de que ocurran, pero no intentar eliminarlos inicialmente.
  - Nivel 4: Prevención.
    - Crear y llevar a cabo un plan como parte del proyecto software para identificar riesgos y evitar que se conviertan en problemas.
  - Nivel 5: Eliminación de las causas principales.
    - Identificar y eliminar los factores que puedan hacer posible la presencia de algún tipo de riesgo.

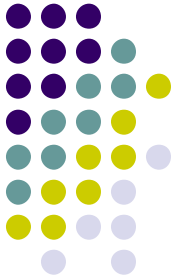
Pilar 3: Gestión de riesgos (III)



- Según Boehm, la gestión de riesgos se compone de:
  - Estimación de riesgos:
    - Identificación de riesgos: Genera una lista de riesgos capaces de romper la planificación del proyecto.
    - Análisis de riesgos: Mide la probabilidad y el impacto de cada riesgo, y los niveles de riesgo de los métodos alternativos.
    - Priorización de riesgos: Genera una lista de riesgos ordenados por su impacto.
  - Control de riesgos:
    - Planificación de la gestión de riesgos: Genera un plan para tratar cada riesgo significativo. También asegura que los planes para la gestión de riesgos de cada uno de ellos son consistentes entre sí y con el plan del proyecto.
    - Resolución de riesgos: Es la ejecución del plan para resolver cada uno de los riesgos significativos.
    - Monitorización de riesgos: Es la actividad del progreso de la monitorización dirigido a la resolución de cada elemento del riesgo.

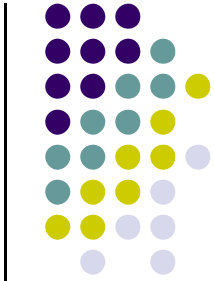
- A continuación se verán brevemente estas fases …

Pilar 3: Gestión de riesgos (IV)



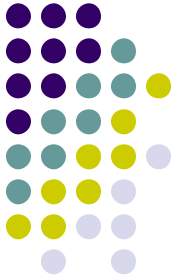
- Identificación de riesgos:
  - Los riesgos se parecen mucho a los errores clásicos ya mencionados.
  - La diferencia es:
    - Los errores clásicos se cometen con mayor frecuencia que los riesgos.
    - Los riesgos son menos comunes o pueden ser únicos para un proyecto determinado.
  - Steve McConnell recopila en su libro “Desarrollo y gestión de proyectos informáticos” los riesgos más habituales en planificación y los riesgos potenciales de la planificación.

Pilar 3: Gestión de riesgos (V)



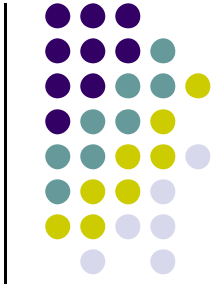
- Análisis de riesgos:
  - Una vez identificados, el paso siguiente es analizar cada riesgo para determinar su impacto.
  - Para cada riesgo, determinar su exposición al riesgo:
    - ER = probabilidad de pérdida no esperada x magnitud de pérdida, donde riesgo = pérdida no esperada.
  - Las probabilidades y las magnitudes de pérdida hay que estimarlas, sin pretender ser exactos.
  - Si sólo se quieren los riesgos de planificación, las pérdidas se expresan en unidades de tiempo.
    - Si se suman todas las ER, se obtiene el retraso total del proyecto y se podría emplear para ajustar la planificación con un margen de retraso.
  - Steve McConnell presenta ejemplos de estimación de riesgos en su libro ya mencionado.

Pilar 3: Gestión de riesgos (VI)



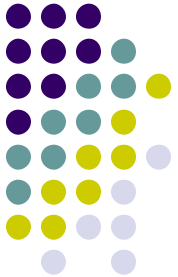
- Priorización de riesgos:
  - Los proyectos suelen gastar el 80% de su presupuesto en arreglar el 20% de sus problemas.
  - Conclusión:
    - Centrase, fundamentalmente, en ese 20%.
  - Se pueden ordenar los riesgos por su ER y así saber cuáles controlar para compensar esfuerzo y reducción en tiempo de los riesgos.
  - Steve McConnell presenta ejemplos de priorización de riesgos en su libro ya mencionado donde se aprecia:
    - Mayor utilidad al controlar unos riesgos que otros.
    - Necesidad de controlar riesgos especialmente “dolorosos”.

Pilar 3: Gestión de riesgos (VII)



- Planificación de la gestión de riesgos:
  - El plan de gestión de riesgos puede ser tan sencillo como un párrafo para cada riesgo describiendo:
    - Quién.
    - Qué.
    - Cuándo.
    - Cómo.

Pilar 3: Gestión de riesgos (VIII)



- Resolución de riesgos:
  - Depende del riesgo específico que se esté tratando.
  - Algunas posibles acciones son:
    - Evitar el riesgo: No realizar actividades arriesgadas.
    - Trasladar el riesgo de una parte del sistema a otra: Expertos en una materia supervisan a los noveles y que el riesgo no esté en el camino crítico de la planificación.
    - Informarse del riesgo: Conocerlo mejor y ver si es o no posible.
    - Asumir el riesgo: Si las consecuencias son pequeñas y el esfuerzo grande.
    - Comunicar: Comunicar el posible impacto de un riesgo en caso de ocurrir.
    - Controlar el riesgo: Planificar acciones en caso de ocurrir.
    - Recordar el riesgo: Para proyectos futuros.

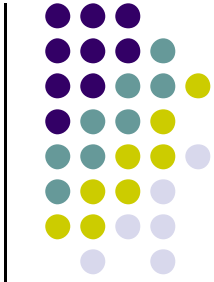
Pilar 3: Gestión de riesgos (IX)

- Monitorización de riesgos:
  - Los riesgos aparecen y desaparecen en el desarrollo del proyecto.
  - Por ello se necesita una monitorización de riesgos que permita:
    - Comprobar cómo progresa el control de un riesgo.
    - Identificar cómo aparecen los nuevos riesgos.

Pilar 4: Métodos orientados a la planificación

- Uno de los aspectos más importantes (si no el más) para llevar a cabo una buena planificación es la estimación.
  - Más adelante se verán métodos orientados a realizar la planificación de un proyecto, pero ahora se abordará la estimación.
    - Es el primer paso para poder efectuar un plan de proyecto.
- Cualquier proyecto debe justificarse, bien por ahorro en costes de operación, expansión del negocio, etc.
  - Debe existir una valoración de la viabilidad técnica y la inversión detallada, con el coste y los plazos manejados.
- Es decir, es necesaria la estimación del nuevo proyecto.
- Sin embargo, la credibilidad de la estimación en los proyectos suele ser baja. En un estudio de Capers Jones de 1994:
  - 5% cumplen presupuesto.
  - 10% lo exceden en un 10%.
  - 20% lo exceden en un 25%.
  - 30% lo exceden en un 50%.
  - 35% lo exceden en un 100% o más.
- Una buena regla sería que los beneficios esperados superasen en al menos 5 veces los costes previstos.

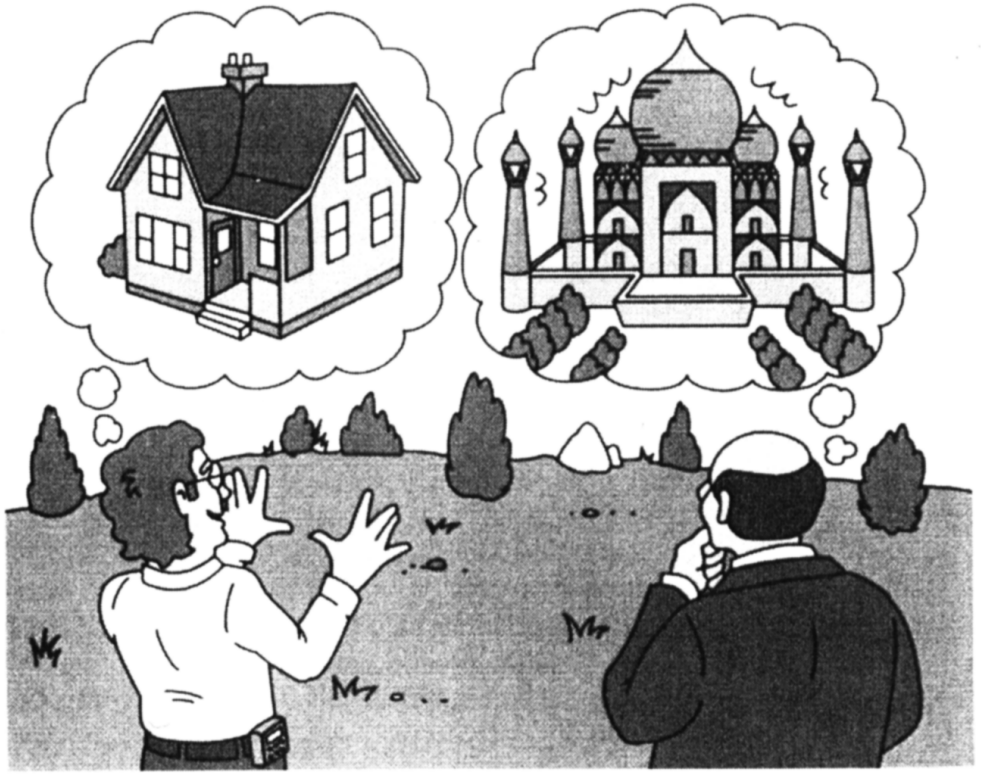
Estimación (I)



- Estimación es la actividad que permite obtener respuesta a:
  - ¿Cuánto costará?
  - ¿Cuánto tiempo llevará hacerlo?
- Las dificultades de una estimación son, entre otras:
  - No hay un modelo de cómo hacerla en todas las organizaciones.
  - Se necesitan diferentes estimaciones para las diferentes personas que hay en un proyecto (alta dirección, dirección del proyecto y cada recurso).
  - La utilidad de una estimación depende de la etapa de desarrollo en que se esté:
    - Se precisa de diferente grado de exactitud a medida que avanza el proyecto.
  - Se suele hacer muy superficialmente y bajo presión.
  - Hay muchos cambios y la información de partida es muy vaga a veces.
  - Los rápidos cambios en las tecnologías de información son problemas para la estabilidad de un proceso de estimación.
  - Escasez de históricos y experiencia en estimar proyectos.
  - Tendencia a subestimar, ignorando aspectos no lineales como coordinación y gestión.
- Es considerada por muchos la actividad más difícil en gestión de proyectos, porque …

Estimación (II)

- Como indica Steve McConnell, es muy difícil saber si es posible construir el producto que el cliente desea en el tiempo esperado hasta que se comprenda perfectamente lo que quiere:



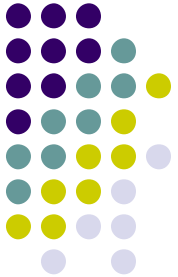
«¿Un año para construir una casa aquí? No hay ningún problema.»

«Bien. Vamos a empezar. Me corre prisa.»

Estimación (III)

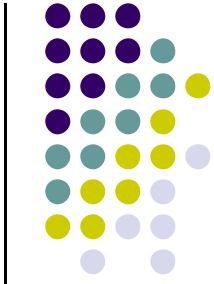
- Ciertamente es una tarea muy difícil.
- Pero es una tarea imprescindible en planificación.
- El proceso para planificar un desarrollo consta de tres pasos básicos fundamentales:
  - Estimar el tamaño del producto:
    - Es el paso más difícil con diferencia.
      - Se puede hacer, por ejemplo, a través del número de líneas de código (LOC) o por puntos función.
  - Estimar el esfuerzo asociado a un producto de dicho tamaño:
    - Si hay una buena estimación del tamaño y un histórico de la organización en proyectos similares, es relativamente fácil.
  - Estimar la planificación:
    - Estimados tamaño y esfuerzo, esta estimación es fácil.

Estimación (IV)



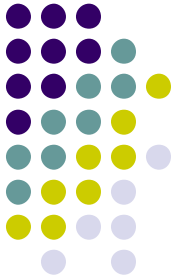
- Estimación del tamaño:
  - Se puede estimar el tamaño de un proyecto de varias formas:
    - Utilizar un enfoque algorítmico (e.g., puntos de función) para estimar el tamaño del programa a partir de sus prestaciones.
    - Utilizar un software de estimación, a partir de la descripción de las prestaciones del programa (pantallas, informes, archivos, consultas, tablas de la BD, ...).
    - Emplear experiencias en proyectos similares.

Estimación (V)



- Estimación del esfuerzo:
  - Obtenida la estimación del tamaño se puede derivar la estimación del esfuerzo.
  - Esta estimación ayudará a:
    - Saber cuántas personas hay que incorporar al proyecto.
    - Estimar la planificación.
  - Para convertir la estimación del tamaño en la del esfuerzo existen varias posibilidades:
    - Emplear software de estimación.
    - Emplear tablas de conversión tamaño-esfuerzo.
    - Emplear históricos.
    - Emplear métodos algorítmicos de aproximación como el COCOMO de Boehm.

Estimación (VI)



- Estimación de la planificación:
  - Una regla muy usada y con pocas variantes para calcular la estimación de la planificación a partir de la estimación del esfuerzo es:
    - Planificación (meses) = 3*personas-mes estimadas$^{1/3}$
  - Ejemplo:
    - Si se ha estimado que serían necesarias 65 personas-mes para un proyecto, la ecuación establece que la planificación óptima es de 12 meses (=3*65$^{1/3}$). Esto implica un tamaño óptimo del equipo de 65 personas-mes/12 meses: un equipo con 5 o 6 miembros.

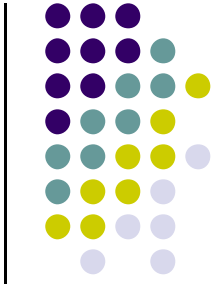
Estimación (VII)

- Relación entre tamaño y esfuerzo:
  - Un aspecto a tener en cuenta a lo largo del ciclo de vida y de las sucesivas estimaciones es la relación existente entre el tamaño del software y el esfuerzo (y la duración).
  - Debido al incremento de esfuerzo de intercomunicación entre las personas que componen el equipo de desarrollo a medida que éste crece, no existe una relación lineal entre tamaño y esfuerzo, ni entre tamaño y duración y tampoco entre esfuerzo y duración.
  - El ejemplo claro se puede ver al hilo de la fórmula anterior o al manejar el modelo COCOMO.

Estimación (VIII)

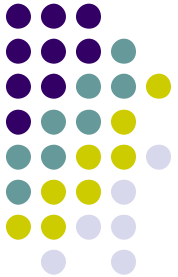
- Estimación dentro del desarrollo:
  - Se escalona a lo largo de varios puntos (referencias) dentro del ciclo de vida:
    - Referencia 0: Cuando se realiza la oferta.
      - Se procede a una primera estimación para dar un “presupuesto inicial”.
    - Referencia 1: Al principio de la fase de especificación.
      - Se elabora una nueva estimación, que constituye el “presupuesto de desarrollo (versión inicial)”, considerando la información aportada por el cliente y las técnicas elegidas en la oferta para desarrollar el producto.
    - Referencia 2: Al concluir la fase de especificación.
      - Se afina el presupuesto previo con las precisiones aportadas por el contenido funcional y operacional del producto y la naturaleza organizativa del proyecto. Es el “presupuesto de desarrollo (versión final)”.
    - Referencia 3: Al concluir la fase de diseño de arquitectura.
      - Constituye el “presupuesto final”. Tras definir la arquitectura se tiene mucho más conocimiento para hacer ya una estimación “definitiva”.

Juicio de expertos



- Es una técnica de estimación que se suele emplear bastante.
- Según Barbara Kitchenham, su nombre es un eufemismo de “adivinación basada en la experiencia personal”.
- Existen técnicas específicas que intentan sistematizar y mejorar la opinión de las distintas personas involucradas (expertos).
- Se suele manejar la opinión de más de un experto para una mayor fiabilidad.
  - Ejemplo: Calcular la media de las estimaciones dadas por los expertos.
    - Riesgo: La estimación así obtenida puede resentirse de valores extremos.
  - Ejemplo: Efectuar una reunión ininterrumpida hasta llegar a un consenso.
    - Riesgo: Las personas más influyentes, por su capacidad o poder de persuasión, determinan el resultado final.
- La técnica más conocida es Delphi:
  - Creada en los años cuarenta en EEUU.
  - Difundida en el ámbito del software por Barry Boehm.
  - La mecánica de la técnica es la siguiente:

Delphi



- Un coordinador proporciona a cada experto una especificación del proyecto considerado y un impreso para expresar su estimación.
- Todos los expertos rellenan el impreso de forma anónima.
  - Pueden hacer preguntas al coordinador sobre el proyecto.
  - No pueden intercambiar opiniones entre ellos.
- El coordinador ofrece a cada experto el valor medio de todas las estimaciones para que la compare con la suya.
  - Se pide realizar una nueva estimación (anónima), indicando las posibles razones de la misma.
- Se repite el proceso de recogida de estimaciones hasta llegar a un consenso en la estimación.
  - No se realizan reuniones en grupo durante todo el proceso.

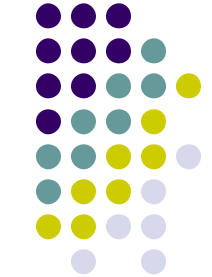
- Boehm propuso una variación, llamada Delphi de banda ancha, que da mejores resultados. Consiste en lo siguiente :

Delphi de banda ancha

- Un coordinador proporciona a cada experto una especificación del proyecto considerado y un impreso para expresar su estimación.
- El coordinador reúne a los expertos para que intercambien puntos de vista sobre el proyecto.
- Todos los expertos rellenan el impreso de forma anónima.
- El coordinador ofrece a cada experto el valor medio de todas las estimaciones para que la compare con la suya.
  - Se pide realizar una nueva estimación (anónima), sin indicar las posibles razones de la misma.
- El coordinador convoca una reunión de grupo para que los expertos discutan las razones de las diferencias entre sus estimaciones.
- Se repite el proceso de recogida de estimaciones hasta llegar a un consenso.

- Ventajas:
  - Permite considerar diferencias entre experiencias pasadas y el proyecto actual difíciles de evaluar sin recurrir a personas experimentadas.
- Inconvenientes:
  - Los consultados pueden expresar una estimación subjetiva e inexperta.

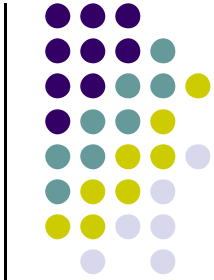
Estimación por analogía



- Consiste en una variante “formal” de la técnica de juicio de expertos:
  - Se compara el proyecto que se va a desarrollar con uno o más proyectos ya terminados de los que existen datos reales recopilados a posteriori.
  - En función de las similitudes y diferencias con dichos proyectos se deduce la estimación para el nuevo desarrollo.
- Las personas involucradas no sólo trabajan aquí con su experiencia acumulada (juicio de expertos), sino que también disponen de datos de proyectos acabados relativamente similares al que hay que estimar.
- Así, por comparación, se pueden evaluar las diferencias entre el nuevo proyecto y los antiguos y extrapolar su estimación.
- Esta técnica mejora sus resultados según se disponga de más datos de proyectos terminados.

- Ventajas:
  - Se considera la experiencia real (no sólo la subjetiva) en los proyectos.
- Inconvenientes:
  - Es difícil conocer el grado de similitud real del proyecto que se estima con los elegidos.

Estimación por descomposición



- También se denomina estimación por jerarquía de componentes o analytic hierarchy process.
- Consiste en descomponer el producto que se va a desarrollar en sus componentes (o el proyecto en tareas sencillas) hasta un nivel de detalle tal que permita estimar directamente cada una de dichas unidades elementales.
- La estimación total del proyecto será el resultado de sumar de forma bottom-up las estimaciones parciales de todas las unidades.
- Para poder aplicar esta técnica con una mínima eficacia se necesita disponer de un diagrama de descomposición.
  - Lo más habitual es contar con una WBS, que suele crearse en la planificación del proyecto.

- Ventajas:
  - Muy intuitiva y permite comprender mejor la tarea (o el producto) a desarrollar.
- Inconvenientes:
  - Olvidos de actividades (o subproductos) que, por tanto, no se estimarán.
  - Inclusión de actividades que no son propias del proyecto.

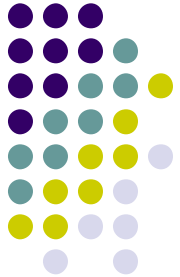
Modelo COCOMO (I)

- COnstructive COst MOdel fue publicado por Barry W. Boehm en 1981.
- Es un modelo de estimación basado en ecuaciones no lineales obtenidas mediante técnicas de regresión sobre un histórico de proyectos ya terminados.
- El método considera tres variables:
  - Tamaño del software a desarrollar (medido en KLOC): Dato a estimar (en miles).
  - Esfuerzo del equipo de desarrollo (expresado en personas-mes).
  - Duración del proyecto (expresada en meses).
- Se distinguen tres modos debidos a las tres categorías de proyectos considerados:
  - Proyectos orgánicos (“organic projects”).
  - Proyectos medianos (“semidetached projects”).
  - Proyectos complejos (“embedded projects”).
- A cada categoría de proyecto se asocian dos ecuaciones en la forma:
  - Esfuerzo = A * (Tamaño ^{B}).
  - Duración = C * (Esfuerzo ^{D}).
  - Cada categoría de proyecto posee sus propias constantes A, B, C y D.
- Se pueden aplicar tres “versiones de estimación” o modelos según el estado en que se encuentre el desarrollo del proyecto:
  - Básico.
  - Intermedio.
  - Detallado.

Modelo COCOMO (II)

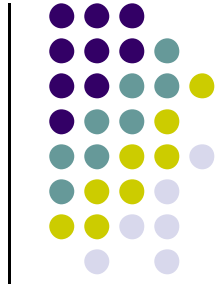
- La entrada al modelo COCOMO es el tamaño del software expresado en miles de líneas de código fuente.
- Para contarlas/estimarlas deben seguirse las mismas reglas predefinidas para todos los proyectos: Hay que establecer un estándar.
- Un conjunto de dichas reglas podría ser:
  - Contar toda línea escrita nueva o modificada.
  - No contar, salvo que tengan un carácter definitivo, las líneas para la instrumentación del código (e.g., para las pruebas).
  - Contar las líneas de los programas de prueba tan sólo si se desarrollan con el mismo nivel de calidad exigido al producto.
  - Contar las líneas correspondientes a los procedimientos de comandos del sistema operativo y JCLs (Job Control Language).
  - No considerar los comentarios (el modelo ya los considera siempre que no sobrepasen un 10% del tamaño global).
  - Contar como una línea cada ocurrencia de macro, copy o include.
  - Contar sólo una vez el código generado por las macros, copy o include.
  - Si se utilizan varios lenguajes de programación para un mismo proyecto, se deben fijar las estimaciones refiriéndose a un solo lenguaje y corregir el resto mediante ratios con respecto a éste.

Modelo COCOMO (III)



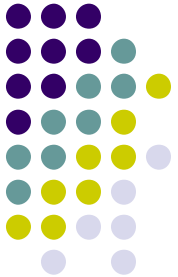
- Para cada uno de los tipos de proyecto identificados en COCOMO difiere la productividad y, por ende, las ecuaciones propuestas para calcular esfuerzo y duración:
  - Las constantes A, B, C y D son propias de cada categoría de proyecto.
- Los **proyectos orgánicos** se caracterizan por:
  - Dominio de la aplicación perfectamente conocido por todos los miembros del equipo.
  - Pocas restricciones operacionales, de interfaz o de rendimiento.
  - Entorno de desarrollo estable, sin material nuevo, ni procedimientos operacionales novedosos.
  - Pocas innovaciones en la arquitectura del software o en los algoritmos para el tratamiento.
  - Organización del proyecto simple.
- Los **proyectos medianos** se caracterizan por:
  - Equipos formados por personas expertas e inexpertas en el dominio de la aplicación.
  - Restricciones operacionales, de interfaces o de rendimiento severas para ciertos aspectos y fáciles para otros.
  - Entorno de desarrollo inestable.
  - Organización del proyecto de complejidad mediana.
  - Esta categoría incluye los proyectos que no se pueden encuadrar en las otras dos.
- Los **proyectos complejos** se caracterizan por:
  - Grandes restricciones operacionales, de interfaces o de rendimiento.
  - Suelen ser sistemas hardware/software complejos con influencia clara de la seguridad o tiempo real.
  - La especificación y/o arquitectura son nuevos o complejos.
  - Organización del proyecto de complejidad alta.

Modelo COCOMO (IV)



- Las constantes A, B, C y D son las siguientes:
- Proyectos orgánicos:
  - A: 2.4 (Para COCOMO Intermedio: 3.2).
  - B: 1.05.
  - C: 2.5.
  - D: 0.38.
- Proyectos medianos:
  - A: 3.
  - B: 1.12.
  - C: 2.5.
  - D: 0.35.
- Proyectos complejos:
  - A: 3.6 (Para COCOMO Intermedio: 2.8).
  - B: 1.2.
  - C: 2.5.
  - D: 0.32.
- Obsérvese que los valores de C y D son muy similares a los de la fórmula genérica que se vio para estimar la duración a partir del esfuerzo en la introducción del apartado de estimación.

Modelo COCOMO (V)



- COCOMO Básico:
  - Permite una primera estimación durante la fase de análisis previo, cuando la mayoría de los factores que incidirán en el proyecto se desconocen.
  - Sólo parte de las KLOC previstas para dar una estimación en cuanto al orden de magnitud del esfuerzo.
- COCOMO Intermedio:
  - Se aplica cuando el proyecto ha sido dividido en subsistemas.
  - Los valores de la constante A se ven afectados.
  - Permite considerar las características más significativas del proyecto mediante la aplicación de factores correctores relativos a atributos del producto software y de recursos (materiales, métodos y personal) que influyen significativamente sobre la productividad y que conciernen a:
    - Exigencias particulares que debe cumplir el software.
    - El entorno (plataforma) empleado durante el desarrollo.
    - La aptitud y competencia de los equipos de desarrollo.
    - El contexto técnico y organizativo del proyecto.
  - Para considerar estos aspectos, Boehm identifica y cuantifica 15 factores que permiten corregir el esfuerzo estimado:
    - Esfuerzo = A * (Tamaño ^{B}) * m; siendo m = Π f_j (j=1…15).
  - Ejemplos de dichos factores:
    - Experiencia en el lenguaje de programación y Experiencia en el dominio de aplicación.
- COCOMO Detallado:
  - Distribuye el esfuerzo corregido entre las fases del proyecto y sus actividades principales con base en unas tablas de distribución proporcionadas por este modelo.

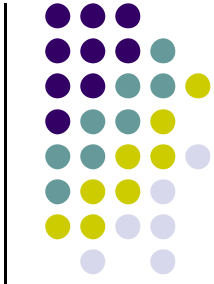
Modelo COCOMO (VI)

- Los supuestos que COCOMO asume son:
  - La base de la estimación es el número de instrucciones de código fuente (excluyendo comentarios y en miles) que será necesario escribir para desarrollar el proyecto.
    - En C y PASCAL una aproximación sería el número de ";".
    - Más de una línea física para una sentencia es una línea de código.
    - OJO: No son lo mismo para ensamblador que para Java.
  - Uso de un modelo de desarrollo en Cascada.
    - Si el proyecto bajo estudio se ejecuta usando otro modelo, los porcentajes de distribución deben reinterpretarse o no deben ser considerados.
  - La estimación del esfuerzo obtenida mediante COCOMO abarca desde la fase de diseño hasta las pruebas internas inclusive.
    - Las actividades de especificación se deberán estimar por separado.
  - El modelo de referencia de Boehm considera 152 horas de trabajo efectivo/mes:
    - 19 días/mes * 8 horas/día.
  - Presupone que se hace una correcta planificación y gestión, se sigue una metodología, etc.
    - Si no es así las estimaciones sufrirán desviaciones importantes (escasa utilidad).
  - Los factores correctores son independientes entre sí (no influyen unos en otros), lo cual puede ser cuando menos discutible.

Modelo COCOMO II

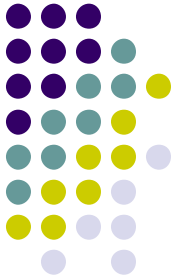
- Boehm y un equipo de colaboradores, fundamentalmente en la University of Southern California, actualizaron y mejoraron el modelo COCOMO a mediados de los 90: El resultado es el nuevo modelo COCOMO II (http://sunset.usc.edu/research/COCOMOII/).
- All igual que su predecesor, COCOMO II es en realidad una jerarquía de modelos de estimación que se emplean en función de la fase de desarrollo en la que se desea estimar:
  - Modelo de composición de aplicación:
    - Se emplea durante las primeras etapas, cuando es fundamental la elaboración de prototipos, la consideración de la interacción del software y el sistema, la valoración del desempeño y la evaluación de la madurez de la tecnología.
    - La estimación de tamaño se basa en puntos objeto (similar al concepto de puntos de función).
  - Modelo de diseño temprano:
    - Se emplea una vez que se han estabilizado los requisitos y se ha establecido la arquitectura básica del software.
    - La estimación de tamaño se basa en puntos de función no ajustados.
  - Modelo Post-arquitectura:
    - Se emplea durante la construcción del software.
    - Se basa en convertir los puntos de función no ajustados a KLOC para aplicar un modelo parecido al COCOMO original donde los factores correctores se han reformado bastante.

Puntos de función (I)



- El método de estimación por puntos de función se denomina FPA:
  - Function Point Analysis (Análisis de Puntos de Función).
- Este método no se basa en las LOC, sino en los puntos de función:
  - Puntos de función es una métrica sintética del tamaño del programa que cuantifica la funcionalidad que hay que entregar al usuario.
- Existen diferentes métodos para contabilizar puntos de función.
  - La propuesta inicial fue realizada por Allan J. Albrecht en 1979.
- Los puntos de función son más fáciles de determinar a partir de la especificación de requisitos que las LOC y dan una medida más exacta sobre el tamaño.
  - Todas las variedades de FPA se apoyan en datos que implican preferentemente la existencia de una especificación de requisitos relativamente formalizada.
    - Se requiere de un mínimo conocimiento de las funcionalidades y entidades que intervienen.
- Aquí se describirá brevemente un método basado en el “1984 IBM Method”, que es la base de:
  - El método actual de IBM.
  - El método del International Function Point Users’ Group.

Puntos de función (II)



- El número de puntos de función en un sistema/programa se basa en el número y la complejidad de:
  - Entradas externas:
    - Grupos lógicos de datos o mandatos de control de un usuario final o cualquier otro programa que entran en la aplicación y añaden o cambian información en un grupo lógico de datos interno. Una entrada es única si difiere en su formato o arranca procesos diferentes.
  - Salidas externas:
    - Grupos lógicos de datos o mandatos de control a un usuario final o a cualquier otro programa que salen de la aplicación. Una salida es única si difiere en su formato o si es generada por procesos lógicos diferentes.
  - Consultas externas:
    - Son entradas que generan una salida simple e inmediata. Son consecuencia de una búsqueda y no una actualización de un grupo lógico de datos internos.
    - Se identificará la complejidad de la parte correspondiente a las entradas externas y de la parte correspondiente a las salidas externas, seleccionándose la más compleja de las dos para signar su complejidad.
  - Archivos lógicos internos o grupos lógicos de datos internos:
    - Grupos lógicos de datos o información de control interna que se generan, son usados y mantiene la aplicación. No deben incluirse aquellos grupos lógicos de datos que no sean accesibles a través de entradas, salidas, ficheros de interfaz o consultas.
  - Archivos de interfaz externos o grupos lógicos de datos de interfaz:
    - Grupos lógicos de datos compartidos con otra aplicación, recibidos o enviados a ella. Los grupos lógicos internos que son a su vez interfaz deben contarse en ambos grupos.

Puntos de función (III)

- Para calcular el número de puntos de función en un sistema/programa se toma el elemento y se multiplica por el peso indicado en la tabla:

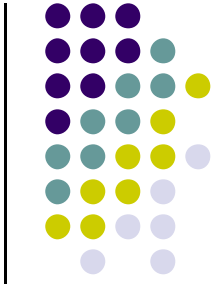
Parámetro	Complejidad baja	Complejidad media	Complejidad alta
Entradas	x 3	x 4	x 6
Salidas	x 4	x 5	x 7
Consultas	x 3	x 4	x 6
Archivos lógicos internos	x 7	x 10	x 15
Archivos de interfaz externos	x 5	x 7	x 10

- La suma de estos números da el total de los puntos de función no ajustados.
- Después se calcula un factor de ajuste de complejidad basado en la influencia que tienen 14 factores sobre el proyecto, valorados en una escala de 0 a 5.
  - Factor de ajuste = (0.01 x Σ factores de complejidad) + 0.65.
- Ejemplos de factores de complejidad:
  - Comunicaciones de datos.
  - Entrada on-line de datos.
  - Complejidad del procesamiento.
- Se multiplican ambos valores para obtener la suma total de los puntos de función.
  - El intervalo de este factor de ajuste es de 0.65 a 1.35 (→ ±35% de variación).

Puntos de función (IV)

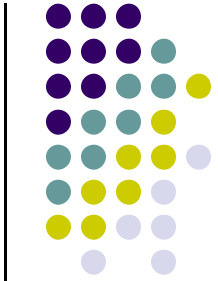
- Las características más destacables de esta técnica son:
  - Independencia del entorno tecnológico de desarrollo.
  - Independencia de la metodología que vaya a ser empleada.
  - Independencia de la experiencia y del estilo de programación.
  - Fácilmente entendible por el usuario.
- El resultado de esta técnica se expresa en puntos de función, que posteriormente han de ser pasados a una métrica de esfuerzo.
  - Esta "traducción" sí tiene en cuenta la experiencia y el estilo de programación, la aplicación de una u otra metodología y la tecnología.
- Este cálculo de días de esfuerzo por punto de función debe basarse en históricos, debiendo actualizarse el valor de conversión con posterioridad a la finalización de cada proyecto.
  - Hay incluso aproximaciones (e.g., Capers Jones) que establecen una equivalencia entre puntos de función y LOC en ciertos lenguajes.
- Con el número de puntos de función se puede comparar el tamaño y la planificación de proyectos anteriores y se podría estimar una planificación a partir de éstos.
- Otra posibilidad es emplear el método de estimación de primer orden de Jones, que se verá más adelante.

Puntos de función (V)



- Las variantes más destacables de esta técnica son:
  - En 1985 se crea una nueva versión como parte del modelo de estimación de la empresa de Capers Jones.
    - Cuentan con un factor más de complejidad pero no requieren catalogar cada uno de los parámetros de los puntos de función no ajustados según su complejidad.
  - En 1986 dicha empresa crea otra variedad especialmente adaptada a aplicaciones de tiempo-real y de software de sistemas.
    - Esta variedad se denomina puntos de características y añade la contabilización y la evaluación de los algoritmos que se deben emplear en el sistema a construir.
  - El método MARK II es una evolución que contempla el sistema como una colección de transacciones lógicas compuestas por componentes de entrada, de proceso y de salida.
    - Estas transacciones lógicas se corresponden exactamente con las funciones del sistema y es necesario conocer:
      - Las entidades que intervienen, los tipos de datos de entrada y los de salida.
      - Si se trata de una función por lotes o en línea.
      - Si se van a emplear lenguajes 3GL o 4GL.

Estimación de primer orden de Jones



- Es un método desarrollado por Capers Jones.
- Si se tiene la suma total de todos los puntos de función, este método permite realizar a partir de ellos un cálculo aproximado de la planificación (duración del proyecto en meses).
- Consiste en tomar el total de los puntos de función y elevarlo a la potencia apropiada seleccionada de la tabla siguiente:

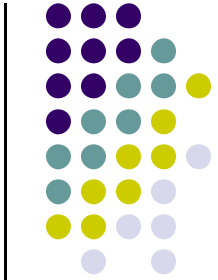
Clase de software	Mejor caso	Media	Peor caso
Sistemas	0.43	0.45	0.48
Gestión	0.41	0.43	0.46
“Prêt-à-porter”	0.39	0.42	0.45

- Esta tabla de potencias surge del análisis de su base de datos de miles de proyectos.
- En la tabla se cruza el tipo de software desarrollado con el tipo de empresa (media, muy organizada y poco organizada).
- No es la mejor forma de estimar la planificación, pero da una primera aproximación (mucho mejor que hacerlo a ojo).

Críticas a los modelos de estimación

- Los modelos de estimación no son la panacea:
  - Los que dependen de LOC suponen calcular de alguna manera ese dato.
    - La adivinación o deducción por expertos de este valor es muy compleja.
  - La aplicación de los puntos de función presenta problemas como la duplicación de la influencia de factores correctores si se parte de ellos para calcular LOC, como entrada por ejemplo al modelo COCOMO.
  - Todos los modelos han surgido del análisis estadístico de datos.
    - El problema es que la cantidad y representatividad de los proyectos no es todo lo amplia que sería de desear.
  - Según diversos estudios, los modelos pierden bastante precisión al utilizarse en entornos distintos a los que los originaron.
    - La solución es adaptar dichos modelos a cada empresa, pero esto puede implicar tanto esfuerzo como crear un nuevo modelo.
  - Los factores correctores siempre son difíciles de cuantificar y se consideran independientes, aunque no necesariamente lo son.
    - En muchos casos, aunque hay guías para aplicar de manera uniforme y consistente los modelos, existe un alto grado de subjetividad.
  - Los modelos tienen un cierto margen de error que, en algunos casos, es aceptable (máximo ± 25% de error sobre el valor final en un 75% de los casos) aunque, lamentablemente, en otros llega a cifras muy malas (en algunos estudios desde el 85% al 700% de error).

Recomendaciones



- La estimación se debe basar en el desarrollo de modelos y procedimientos de estimación apropiados y adaptados a las características de cada organización:
  - Esto significa que hay que recoger datos de los proyectos y crear procedimientos de estimación que puedan modificarse según se analizan dichos datos.
- En general, no es bueno ceñirse a un solo método de estimación.
- El enfoque recomendado consiste en:
  - Las primeras estimaciones (para negociar contratos disponiendo de poca información) deberían apoyarse en el juicio de expertos con técnica Delphi, preferentemente basadas en analogías con datos fiables de proyectos propios.
    - Recomendación: Un grupo más o menos permanente de expertos dedicados a la estimación.
    - Ventaja: Estimaciones más homogéneas, que reflejan menos prejuicios o preferencias personales, y que las personas acumulan experiencia y formación.
  - Una vez que existen especificaciones medianamente detalladas y se puede medir su tamaño, hay que evaluar cada caso:
    - Si el proyecto no es parecido a los asiduos en la organización o se desea contrastar las ecuaciones de estimación, pueden ser necesarias la estimación a través de expertos y la analogía.
    - Sin embargo, lo más normal es aplicar modelos o ecuaciones de estimación.
  - Conviene aplicar modelos o ecuaciones locales, es decir, no tomar sin más modelos desarrollados en otros entornos u organizaciones.
    - Es preferible adaptarlos o crear ecuaciones propias.
    - Lo ideal es trabajar con modelos cuyos parámetros se puedan medir con facilidad en nuestra organización e, incluso, con ecuaciones diferentes y adaptadas a cada fase del desarrollo.