

# Tema 2. Estimación de costes

- ✓ Introducción
- ✓ Productividad
- ✓ Técnicas de estimación
- ✓ Modelo algorítmico de costes
- ✓ Duración y personal del proyecto

# Bibliografía

Captítulo 26. Software cost estimation. Software Engineering Sommerville 7ª edición.

Capítulo 18. Métricas del software. Ingeniería del software. 4ª edición. Roger S. Pressman.

# Introducción

## ✓ ¿Qué es la estimación de costes?

Consiste en predecir los **recursos** (monetarios, temporales, humanos, materiales, ...) necesarios para llevar a cabo el proceso de desarrollo del software.

## ✓ Cuestiones fundamentales:

- ¿Cuánto esfuerzo es necesario para completar una actividad?
- ¿Cuánto tiempo se necesita para completar una actividad?
- ¿Cuál es el coste total de una actividad?

# Componentes de coste

- ✓ Costes hardware y software
- ✓ Costes de viajes y aprendizaje
- ✓ Costes de **esfuerzo** (factor dominante casi siempre)
  - sueldo ingenieros del proyecto
  - gastos seguros y seguridad social
- ✓ Otros costes:
  - costes de alquiler, calefacción y luz
  - costes de redes y comunicaciones
  - costes de recursos compartidos (p.e. librería, personal del restaurante, etc.)

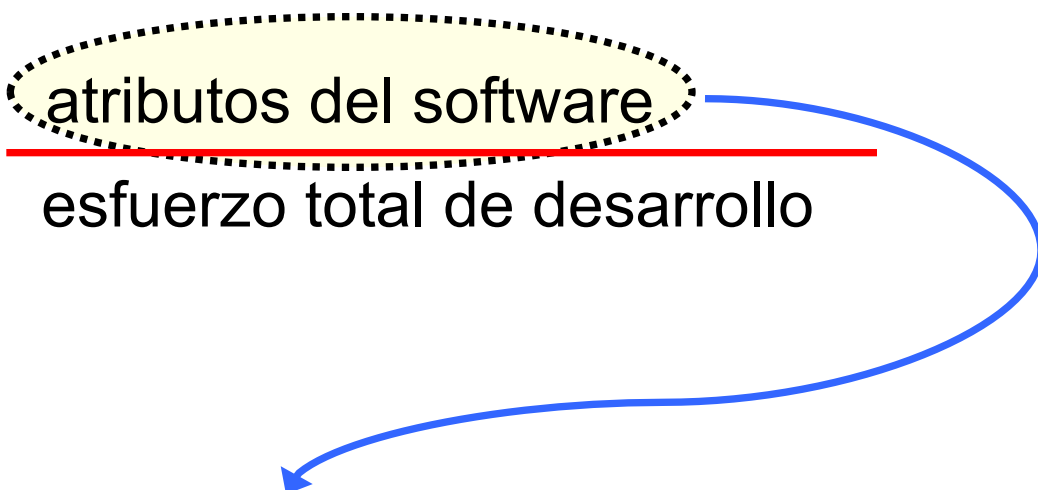
# Factores de coste

Factor	Description
Market opportunity	A development organisation may quote a low price because it wishes to move into a new segment of the software market. Accepting a low profit on one project may give the opportunity of more profit later. The experience gained may allow new products to be developed.
Cost estimate uncertainty	If an organisation is unsure of its cost estimate, it may increase its price by some contingency over and above its normal profit.
Contractual terms	A customer may be willing to allow the developer to retain ownership of the source code and reuse it in other projects. The price charged may then be less than if the software source code is handed over to the customer.
Requirements volatility	If the requirements are likely to change, an organisation may lower its price to win a contract. After the contract is awarded, high prices may be charged for changes to the requirements.
Financial health	Developers in financial difficulty may lower their price to gain a contract. It is better to make a small profit or break even than to go out of business.

# Productividad (I)

- ✓ La productividad de un programador es una medida de la "velocidad" a la que los ingenieros implicados en el desarrollo del software producen dicho **software** y su **documentación** asociada
- ✓ Es necesario estimar la productividad:
  - para realizar las estimaciones necesarias en el proyecto
  - para evaluar si un proceso o mejoras en la tecnología son efectivas.

# Productividad (II)

$$\text{Productividad} = \frac{\text{atributos del software}}{\text{esfuerzo total de desarrollo}}$$


## ✓ Tipos de medidas:

- Relacionadas con el tamaño (líneas de código)
- Relacionadas con la funcionalidad (puntos de función, puntos de objeto)

# Líneas de código

- ✓ ¿Qué es una línea de código?
  - Es una medida propuesta inicialmente cuando los programas se escribían en tarjetas, con una línea por tarjeta
  - Actualmente los lenguajes permiten escribir varias sentencias en una línea, o una misma sentencia en varias líneas
- ✓ Se debe decidir qué programas deberían contarse como parte del sistema
- ✓ Asumen una relación lineal entre el tamaño y el volumen de documentación



# Comparaciones de productividad

- ✓ Cuanto mayor sea la expresividad del lenguaje, más baja será su productividad aparente.
- ✓ Cuánto más líneas de código emplee el programador, mayor será su productividad

	Analysis	Design	Coding	Testing	Documentation
Assembly code	3 weeks	5 weeks	8 weeks	10 weeks	2 weeks
High-level language	3 weeks	5 weeks	8 weeks	6 weeks	2 weeks
	Size	Effort	Productivity		
Assembly code	5000 lines	28 weeks	714 lines/month		
High-level language	1500 lines	20 weeks	300 lines/month		

Comparar la productividad utilizando lenguajes diferentes de programación puede llevar a conclusiones erróneas respecto a la productividad de los programadores

# Puntos de función

- ✓ Basados en una combinación de características del programa
  - entradas y salidas externas
  - interacciones de usuario
  - interfaces externas
  - ficheros usados por el sistema
- ✓ Se asocia un peso con cada uno de ellos
- ✓ Los puntos de función se calculan multiplicando cada factor por su peso y sumando todos ellos

# Cálculo de los puntos de función(I)

Análisis del dominio de inf. y desarrollo de contadores

Establecer contadores para el dominio de entrada e interfaces del sistema

Establecer pesos y evaluar la complejidad

Asignar niveles de complejidad o pesos para cada contador

Evaluación de factores globales

Considerar factores externos,  $F_i$  como reutilización, concurrencia, SO, ...

Calcular los puntos de función

Puntos de función =  $\Sigma(\text{contador} \times \text{peso}) \times C$   
donde:  $C = (0.65 + 0.01 \times N)$  (Multiplicador de complejidad)

$N = \Sigma F_i$  (Grado de influencia)

# Cálculo de los puntos de función(II)

<u>Parámetro de medida</u>	<u>Contador</u>	<u>Factor de peso</u>				
		<u>Simp. Med. Compl.</u>				
N° entradas usuario	<input type="text"/>	× 3	4	6	=	<input type="text"/>
N° salidas usuario	<input type="text"/>	× 4	5	7	=	<input type="text"/>
N° peticiones usuario	<input type="text"/>	× 3	4	6	=	<input type="text"/>
N° ficheros	<input type="text"/>	× 3	4	6	=	<input type="text"/>
N° interfaces externas	<input type="text"/>	× 7	10	15	=	<input type="text"/>
Total contadores						<input type="text"/>
Multiplicador de complejidad						<input type="text"/>
Puntos de función						<input type="text"/>

# Ventajas de los puntos de función

- ✓ Son independientes del lenguaje de programación
- ✓ Pueden calcularse a partir de la especificación
- ✓ Usa información del dominio del problema
- ✓ Resulta más fácil a la hora de reusar componentes
- ✓ Se encamina a aproximaciones orientadas a objetos

# Puntos de función y estimación

- ✓ Los puntos de función (FP) pueden usarse para estimar el **número de líneas de código** (LOC) dependiendo del número medio de LCDs por PF para un lenguaje dado (AVC)
  - $LOC = AVC * \text{número de puntos de función}$
  - AVC es un factor dependiente del lenguaje que varía desde 200-300 para lenguaje ensamblador hasta 2-40 para un lenguaje 4GL
- ✓ Los puntos de función son muy **subjetivos**.  
Dependen del estimador.

# Puntos de objeto

- ✓ Los puntos de objeto son una medida alternativa relacionada con la funcionalidad cuando se utilizan lenguajes 4GLs o similares para el desarrollo
- ✓ Los puntos de objeto **NO** son clases de objetos
- ✓ El número de puntos de objeto en un programa es una estimación ponderada de:
  - El número de pantallas que son visualizadas por separado
  - El número de informes que se producen por el sistema
  - El número de módulos 3GL que deben desarrollarse para complementar el código 4GL

# Estimación de puntos de objeto

- ✓ Son mas fáciles de estimar a partir de una especificación que los puntos de función, ya que solamente consideran pantallas, informes y módulos 3GL
- ✓ Por lo tanto pueden estimarse en fases tempranas del proceso de desarrollo. En estas etapas resulta muy difícil estimar el número de líneas de código de un sistema



# Factores que afectan la product.

- ✓ Experiencia en el dominio de la aplicación
- ✓ Calidad del proceso
- ✓ Tamaño del proyecto
- ✓ Tecnología de soporte
- ✓ Entorno de trabajo

# Calidad y productividad

- ✓ Todas las métricas basadas en volumen/unidad de tiempo son engañosas debido a que no tienen en cuenta la calidad
- ✓ La productividad puede incrementarse generalmente a costa de la calidad
- ✓ No está claro cómo la productividad y las métricas de calidad están relacionadas
- ✓ Las métricas de productividad deberían usarse únicamente como guía

# Técnicas de estimación (I)

- ✓ No existe una forma simple de obtener estimaciones exactas del esfuerzo requerido para desarrollar un sistema software
  - Las estimaciones iniciales se basan en información incompleta en la definición de requerim. del usuario
  - El software puede tener que ejecutarse sobre ordenadores no usuales o usar nuevas tecnologías
  - Puede desconocerse a la gente que interviene en el proy.

# Técnicas de estimación (II)

- ✓ Modelado algorítmico de costes
- ✓ Juicio experto
- ✓ Estimación por analogía
- ✓ Ley de Parkinson
- ✓ *Pricing to win*

# Modelado algorítmico de costes

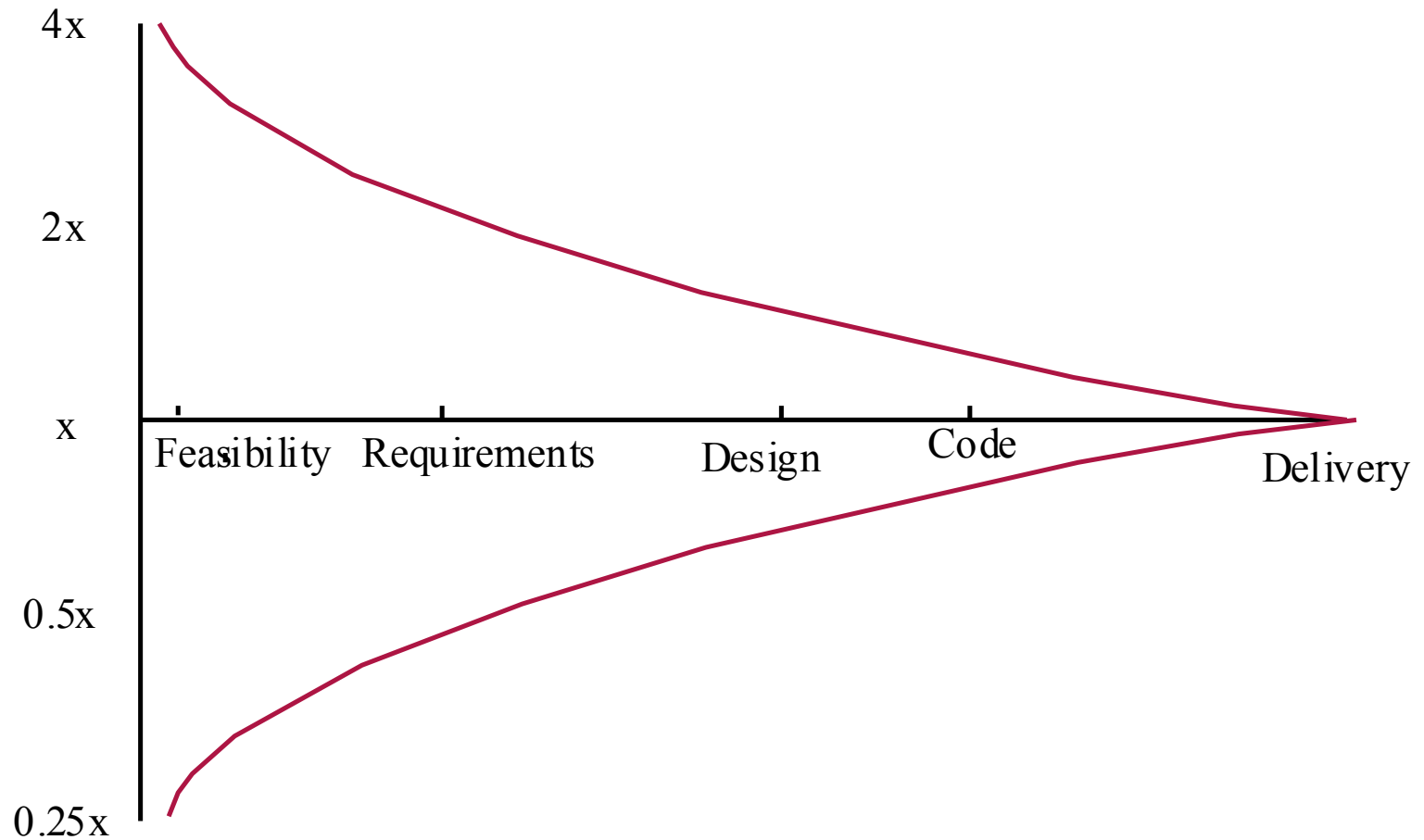
- ✓ Es una aproximación que utiliza fórmulas obtenidas a partir de información histórica. Suele basarse en el tamaño del software
- ✓ La mayoría de modelos tienen una componente exponencial (los costes no crecen normalmente de forma lineal con el tamaño del proyecto)

○  $\text{Esfuerzo} = A \times \text{Tamaño}^B \times M$

# Exactitud de la estimación

- ✓ El tamaño de un sistema software puede conocerse con exactitud solamente cuando está terminado
- ✓ Algunos factores que influyen en el tamaño final son:
  - Uso de COTs y componentes
  - Lenguaje de programación utilizado
  - Distribución del sistema
- ✓ La estimación del tamaño se realiza de forma más exacta a medida que el desarrollo del sistema progresa

# Estimar la incertidumbre



# Juicio experto

- ✓ Uno o más expertos, tanto en desarrollo de software como en el dominio de la aplicación usan su experiencia para predecir los costes de software. Se realizan iteraciones hasta que se alcanza un consenso
- ✓ **Ventajas:** Método de estimación relativamente barato. Puede ser bastante exacto si los expertos tienen experiencia directa en sistemas similares
- ✓ **Desventajas:** ¡Muy impreciso si no se dispone de los expertos adecuados!



# Estimación por analogía

- ✓ El coste de un proyecto se calcula por comparación con proyectos similares en el mismo dominio de aplicación
- ✓ **Ventajas:** Bastante preciso si se disponen de datos de proyectos previos
- ✓ **Inconvenientes:** Imposible de realizar sin no se han abordado proyectos comparables. Necesita un mantenimiento sistemático de una base de datos

# Ley de Parkinson

- ✓ Los costes del proyecto están en función de los recursos disponibles, utilizando todo el tiempo permitido.
- ✓ **Ventajas:** No realiza presupuestos "abultados"
- ✓ **Inconvenientes:** El sistema normalmente no termina

# Pricing to win

- ✓ El coste del proyecto está en función de lo que el cliente está dispuesto a pagar
- ✓ **Ventajas:** La empresa desarrolladora consigue el contrato
- ✓ **Inconvenientes:** La probabilidad de que el cliente obtenga el sistema que quiere es pequeña. Los costes no reflejan realmente el trabajo requerido

# Estimación ascend. y descend.

- ✓ Cualquiera de estas aproximaciones puede utilizarse de forma ascendente o descendente
- ✓ Descendente
  - Comienza a nivel de sistema y evalúa la totalidad de funcionalidades y cómo éstas se subdividen en subsistemas
- ✓ Ascendente
  - Comienza a nivel de componentes y estima el esfuerzo requerido para cada componente. Dichos esfuerzos se añaden a la estimación final

# Estimación descendente

- ✓ Se puede usar sin conocer la arquitectura ni los componentes que formarán parte del sistema
- ✓ Tiene en cuenta costes tales como integración, gestión de configuraciones y documentación
- ✓ Puede infra-estimar costes relacionados con la resolución de problemas técnicos de bajo nivel difíciles de resolver

# Estimación ascendente

- ✓ Se puede usar cuando la arquitectura del sistema es conocida y los componentes han sido identificados
- ✓ Proporciona estimaciones bastante exactas si el sistema se ha diseñado con detalle
- ✓ Puede infra-estimar costes a nivel de sistema, tales como integración y documentación

# Comparando métodos

- ✓ Cada método tiene sus ventajas e inconvenientes
- ✓ La estimación debería basarse en varios métodos
- ✓ Si el resultado de aplicar varios de ellos difiere mucho, es que no se dispone de suficiente información
- ✓ Muchas veces el método *Pricing to win* es el único aplicable

# Modelo COCOMO

- ✓ COnstructive COst MOdel
- ✓ Es un modelo empírico basado en la experiencia con proyectos (grandes)
- ✓ Es un método bien documentado, cuya primera versión se publicó en 1981
- ✓ La última versión, COCOMO 2, tiene en cuenta diferentes aproximaciones de desarrollo, reutilización, etc.



# COCOMO 81

Project complexity	Formula	Description
Simple	$PM = 2.4 (KDSI)^{1.05} \times M$	Well-understood applications developed by small teams.
Moderate	$PM = 3.0 (KDSI)^{1.12} \times M$	More complex projects where team members may have limited experience of related systems.
Embedded	$PM = 3.6 (KDSI)^{1.20} \times M$	Complex projects where the software is part of a strongly coupled complex of hardware, software, regulations and operational procedures.

# Niveles COCOMO 2

- ✓ COCOMO 2 es un modelo de tres niveles que permite estimaciones cada vez más detalladas y que pueden realizarse a la vez que progresa el desarrollo del proyecto
- ✓ Nivel inicial de prototipado
  - Estimaciones realizadas con puntos de objeto y una fórmula simple para el cálculo del esfuerzo
- ✓ Nivel inicial de diseño
  - Estimaciones realizadas con puntos de función convertidas en líneas de código
- ✓ Nivel post-arquitectura
  - Estimaciones basadas en líneas de código fuente

# Nivel inicial prototipado

- ✓ Soporta proyectos con prototipado y proyectos que hacen uso intensivo de la reutilización
- ✓ Basado en estimaciones estándar de la productividad del desarrollador en puntos-objeto/mes
- ✓ Tiene en cuenta el uso de herramientas CASE
- ✓ La fórmula es:
  - $PM = ( NOP \times (1 - \%reuse/100) ) / PROD$
  - PM es el esfuerzo en personas-mes, NOP es el número de puntos de objeto, y PROD es la productividad

# Productiv. de puntos de objeto

- ✓ La productividad (PROD) depende de:
  - La experiencia y capacidad del desarrollador
  - Las capacidades de la herramienta CASE utilizada

Developer's experience and capability	Very low	Low	Nominal	High	Very high
ICASE maturity and capability	Very low	Low	Nominal	High	Very high
PROD (NOP/month)	4	7	13	25	50

# Nivel inicial de diseño

- ✓ Las estimaciones pueden hacerse después de que los requerimientos hayan sido establecidos
- ✓ Basado en las fórmulas estándar para métodos algorítmicos
  - $PM = A \times \text{Tamaño}^B \times M + PM_m$  en donde
  - $M = PERS \times RCPX \times RUSE \times PDIF \times PREX \times FCIL \times SCED$
  - $PM_m = (ASLOC \times (AT/100)) / ATPROD$
  - $A = 2.5$  según la calibración inicial, **Tamaño** se da en KLOC, **B** varía desde 1.1 hasta 1.24 dependiendo de la novedad del proyecto, la flexibilidad del desarrollo, la gestión de riesgos, y la madurez del proceso

# Multiplicadores (**M**)

- ✓ Los multiplicadores reflejan la capacidad de los desarrolladores, requerim. no funcionales, la familiaridad con la plataforma de desarrollo, etc.
  - RCPX - fiabilidad de producto y complejidad
  - RUSE - reutilización requerida
  - PDIF - dificultad de la plataforma
  - PREX - experiencia del personal
  - PERS - capacidad del personal
  - SCED - agenda requerida
  - FCIL - facilidades de soporte de grupo
- ✓ **PM** refleja la cantidad de código generada automáticamente

# Nivel post-arquitectura

- ✓ Usa la misma fórmula que la estimación inicial de diseño
  - $PM = A \times \text{Tamaño}^B \times M + PM_m$
- ✓ Se ajusta la estimación de **tamaño** para que tenga en cuenta
  - La volatilidad de los requerimientos
  - Grado de posible reutilización
  - $ESLOC = ASLOC \times (AA + SU + 0.4DM + 0.3CM + 0.3IM)/100$ 
    - ✗ **ESLOC** es el número de líneas de código nuevo. **ASLOC** es el número de líneas de código reusable que debe modificarse, **DM** es el porcentaje de diseño modificado, **CM** es el porcentaje de código que se modifica, **IM** es el porcentaje del esfuerzo original de integración del software reusado.
    - ✗ **SU** es un factor basado en la interpretación del coste del software, **AA** es un factor que refleja los costes de evaluación iniciales para decidir si el software puede reutilizarse.

# El término exponente (**B**)

- ✓ Depende de 5 factores de escala. La suma de dichos factores se divide por 100 y se añade a 1.01
- ✓ Ejemplo
  - Antecedentes - proyecto nuevo - 4
  - Flexibilidad desarrollo - no implicación cliente - Muy alto - 1
  - Arquitectura/resolución riesgos - No análisis de riesgos - Muy bajo - 5
  - Cohesión del grupo - nuevo grupo - nominal - 3
  - Madurez proceso - algún control - nominal - 3
- ✓ El factor de escala es 1.17



# Factores de escala de exponente

(B)

Scale factor	Explanation
Precedentedness	Reflects the previous experience of the organisation with this type of project. Very low means no previous experience, Extra high means that the organisation is completely familiar with this application domain.
Development flexibility	Reflects the degree of flexibility in the development process. Very low means a prescribed process is used; Extra high means that the client only sets general goals.
Architecture/risk resolution	Reflects the extent of risk analysis carried out. Very low means little analysis, Extra high means a complete a thorough risk analysis.
Team cohesion	Reflects how well the development team know each other and work together. Very low means very difficult interactions, Extra high means an integrated and effective team with no communication problems.
Process maturity	Reflects the process maturity of the organisation. The computation of this value depends on the CMM Maturity Questionnaire but an estimate can be achieved by subtracting the CMM process maturity level from 5.

# Multiplicadores (**M**)

- ✓ Atributos del producto
- ✓ Atributos del ordenador
- ✓ Atributos del personal
- ✓ Atributos del proyecto

# Conductores de coste del proy.

Product attributes			
RELY	Required system reliability	DATA	Size of database used
CPLX	Complexity of system modules	RUSE	Required percentage of reusable components
DOCU	Extent of documentation required		
Computer attributes			
TIME	Execution time constraints	STOR	Memory constraints
PVOL	Volatility of development platform		
Personnel attributes			
ACAP	Capability of project analysts	PCAP	Programmer capability
PCON	Personnel continuity	AEXP	Analyst experience in project domain
PEXP	Programmer experience in project domain	LTEX	Language and tool experience
Project attributes			
TOOL	Use of software tools	SITE	Extent of multi-site working and quality of site communications
SCED	Development schedule compression		

# Efectos de los conduct. de coste

Exponent value	1.17
System size (including factors for reuse and requirements volatility)	128, 000 DSI
<b>Initial COCOMO estimate without cost drivers</b>	<b>730 person-months</b>
Reliability	Very high, multiplier = 1.39
Complexity	Very high, multiplier = 1.3
Memory constraint	High, multiplier = 1.21
Tool use	Low, multiplier = 1.12
Schedule	Accelerated, multiplier = 1.29
<b>Adjusted COCOMO estimate</b>	<b>2306 person-months</b>
Reliability	Very low, multiplier = 0.75
Complexity	Very low, multiplier = 0.75
Memory constraint	None, multiplier = 1
Tool use	Very high, multiplier = 0.72
Schedule	Normal, multiplier = 1
<b>Adjusted COCOMO estimate</b>	<b>295 person-months</b>

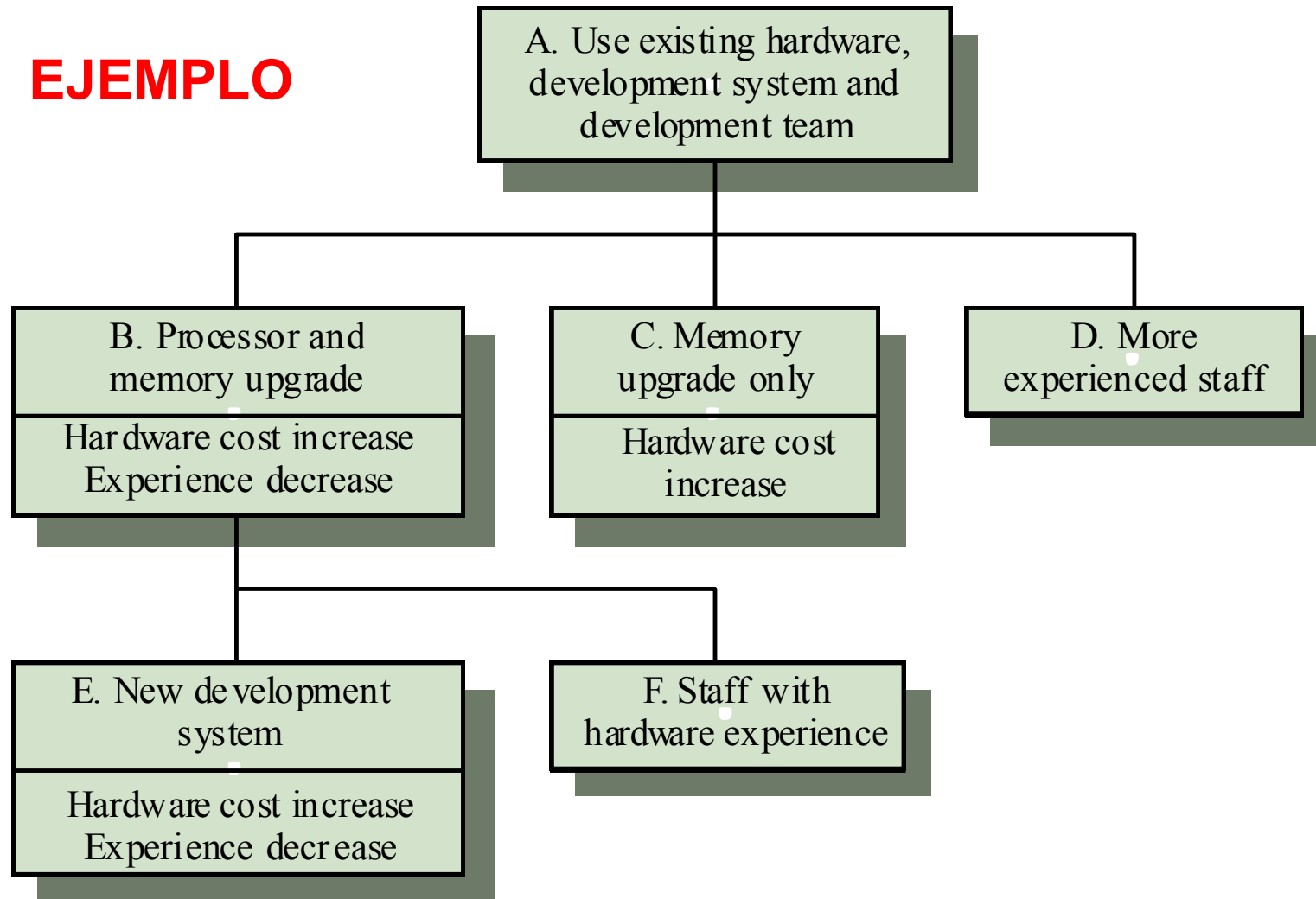
(M)

# Planificación del proyecto

- ✓ Los modelos algorítmicos de costes proporcionan una base para la planificación del proyecto en tanto que permiten comparar estrategias alternativas
- ✓ Ejemplo: desarrollo de un sistema espacial empotrado
  - Debe ser fiable
  - Debe tener un peso mínimo (número de chips)
  - Multiplicadores de fiabilidad y restricciones del ordenador  $> 1$
- ✓ Componentes de coste
  - Hardware destino
  - Plataforma de desarrollo
  - Esfuerzo requerido

# Opciones de gestión

## EJEMPLO



# Gestión de opciones de coste

## EJEMPLO

Option	RELY	STOR	TIME	TOOLS	LTEX	Total effort	Software cost	Hardware cost	Total cost
A	1.39	1.06	1.11	0.86	1	63	949393	100000	1049393
B	1.39	1	1	1.12	1.22	88	1313550	120000	1402025
C	1.39	1	1.11	0.86	1	60	895653	105000	1000653
D	1.39	1.06	1.11	0.86	0.84	51	769008	100000	897490
E	1.39	1	1	0.72	1.22	56	844425	220000	1044159
F	1.39	1	1	1.12	0.84	57	851180	120000	1002706

# Selección de opciones

- ✓ Opción D (usa más personal con experiencia) parece la mejor alternativa
  - Sin embargo tiene un alto riesgo asociado ya que personal con experiencia puede ser difícil de encontrar
- ✓ Opción C (actualización memoria) tiene un menor ahorro de costes, pero un riesgo muy bajo

En conjunto, el modelo revela la importancia del personal con experiencia en el desarrollo del software



# Duración y personal del proyecto

- ✓ Además de la estimación del esfuerzo, se debe estimar el tiempo requerido para terminar el proyecto, así como el personal necesario
- ✓ La duración del proyecto puede estimarse mediante la fórmula de COCOMO 2
  - $TDEV = 3 \times (PM)^{(0.33+0.2*(B-1.01))}$
  - PM es el esfuerzo.

La duración es independiente del número de gente que trabaje en el proyecto (depende del esfuerzo total invertido en el proyecto)

# Puntos clave

- ✓ Es necesario estimar costes: (esfuerzo, tiempo de desarrollo y número de recursos)
- ✓ La productividad es un factor a tener en cuenta a la hora de realizar estimaciones
- ✓ Existen varias técnicas de estimación de costes. La estimación algorítmica de costes es difícil al necesitar una estimación previa de atributos del producto terminado
- ✓ Los modelos de estimación algorítmicos suponen una opción de análisis cuantitativo
- ✓ El tiempo necesario para completar un proyecto no es proporcional al número de personas que trabajan en el mismo