

Tema 4

Estimación del Coste

- 4.1 Introducción**
- 4.2 Ley de Parkinson**
- 4.3 Price-to-Win**
- 4.4 (Wideband) Delphi**
- 4.5 Planning Poker**
- 4.6 CBR**
- 4.7 Clasificación automática**
- 4.8 COCOMO 81**
- 4.9 COCOMO II**
- 4.10 Coste del Producto**

Pablo.Bermejo@uclm.es

4.1 Introducción

4.2 Ley de Parkinson

4.3 Price-to-Win

4.4 (Wideband) Delph

4.5 Planning Poker

4.6 CBR

4.7 Clasificación automática

4.8 COCOMO 81

4.9 COCOMO II

4.10 Coste del Producto

Pablo.Bermejo@uclm.es

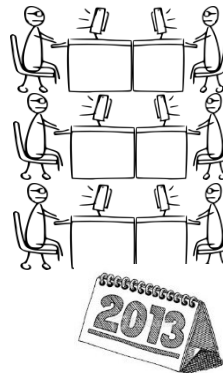
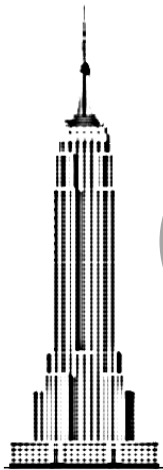
4.1 Introducción

- En la literatura podemos encontrar **diferentes métricas de medición del coste**:
 - **Esfuerzo**: medido en person-months (PM): cuántos meses tardaría una persona en realizar todo el trabajo necesario.
 - **Coste monetario o costos**: dinero necesario para llevar a cabo la implementación del software.
 - **Tiempo**: dado un equipo, cuánto tiempo se tardará en implementar el software.
- Normalmente, el método utilizado para estimar el coste ofrece mecanismos (fórmulas) para convertir esas unidades entre sí.

4.1 Introducción

- Podréis encontrar textos en los que se utiliza el Tamaño del Software (en longitud, funcionalidad o complejidad) como indicación del coste.
 - No es completamente correcto: el **tamaño** es la variable que más afecta al coste (esfuerzo, dinero, tiempo), pero **no es el único factor**.
 - Como ya vimos, el proceso más lógico a seguir en las fases de Inicio y Planificación es estimar:

Tamaño → Esfuerzo → Tiempo → Costos 



Estimación del Coste

COSTE

4.1 Introducción

Tamaño → Esfuerzo → Tiempo → Costos

- El factor que más influye al calcular el **esfuerzo** es el tamaño del software, pero también:
 - Herramientas CASE y reutilización de código.
 - Requisitos no funcionales: calidad, idioma, restricciones de velocidad,...
 - Productividad media estimada para una persona
- El factor que más influye al calcular el tiempo (**cronograma**) es el esfuerzo o el tamaño, pero otros a tener en cuenta:
 - Productividad histórica de cada miembro del equipo.
 - Requisitos no funcionales.
 - **Capacidad de gestión del PM** (project manager).
- El factor que más influye en los **costos** del proyecto es el personal (sueldo+impuestos indirectos como Seg. Social, bajas,...), pero también hay que planificar bien el uso de:
 - Equipo ofimático
 - Redes y comunicaciones
 - Viajes y dietas
 - Productividad exigida

4.1 Introducción

- Así, vemos que la productividad del personal influye en todas las estimaciones. Pero...

- ¿A más LOC eres mejor trabajador?



- ¿La productividad del equipo depende del mejor trabajador?



- ¿Ser productivo en un proyecto se puede generalizar?

No. La productividad de un empleado no puede derivarse solo de la información histórica. Ya que hay varios factores que afectan a la productividad:

- experiencia en la tecnología a utilizar,
 - metodología de desarrollo software: pasar de plan-driven a change-driven
 - tamaño del software (a mayor tamaño, mayor equipo y tiempo gastado en comunicaciones),
 - CASE y reutilización,
 - el ambiente de trabajo.

4.1 Introducción

- Existe una amplia variedad de modelos para la estimación del Coste. Su taxonomía más generalmente aceptada es la siguiente:

Algorítmicos	Analogía	Juicio de Expertos	Otros
<i>COCOMO 81</i>	Razonamiento Basado en Casos (CBR)	(Wideband) Delphi	Ley de Parkinson
<i>COCOMO II (2000)</i>	Clasificación automática	<i>Planning Poker</i>	Price-to-Win

- Además, en algunos casos se añade también las estimaciones “Top-Down” y “Bottom-Up”, sin embargo estos pueden considerarse una forma de aplicar modelos de la tabla, más que un modelo en sí mismo.
- Los más complejos (en términos de cantidad de parámetros) son los algorítmicos.
- En Cursiva los métodos diseñados específicamente para desarrollos software.*

- 4.1 Introducción**
- 4.2 Ley de Parkinson**
- 4.3 Price-to-Win
- 4.4 (Wideband) Delphi
- 4.5 Planning Poker
- 4.6 CBR
- 4.7 Clasificación automática
- 4.8 COCOMO 81
- 4.9 COCOMO II
- 4.10 Coste del Producto

Pablo.Bermejo@uclm.es

4.2 Ley de Parkinson

‘Work expands so as to fill the time available for its completion’

[C.N. Parkinson 1955]

- Lo cual insinúa que...
 - si tenemos que programar 100 LOC y nos dan 2 semanas, tardaremos 2 semanas en hacerlo.
 - si tenemos que programar 5 FP en Java en 2 días (lo normal serían 5), ¡lo conseguiremos!
- Esta ley (opti/pesi)mista nos recuerda que hay que ser realista en el esfuerzo que se puede pedir al equipo.
- Como consecuencia de esta “ley natural”, si el cliente nos indica un *deadline* demasiado temprano, tendremos que ser capaces de desechar funcionalidad poco importante.

4.2 Ley de Parkinson

- **Al disponer de poco tiempo** y equipos pequeños, se aplica esta ley junto a una metodología ágil:
 - Minimizando planificaciones y documentación.
 - Decidiendo requisitos a implementar en cada iteración que aporten el mayor valor.
 - Ej: *Planning Game* en cada iteración de *Extreme Programming*.
- **Si disponemos de mucho tiempo**, la ley de Parkinson nos recuerda que conviene hacer un esfuerzo extra en:
 - **Controlar** la productividad del equipo.
 - **NO** repartir el volumen de forma **balanceada** en el cronograma, sino de forma REALISTA. Si sobra tiempo, ¡mejor!

- 4.1 Introducción**
- 4.2 Ley de Parkinson**
- 4.3 Price-to-Win**
- 4.4 (Wideband) Delphi
- 4.5 Planning Poker
- 4.6 CBR
- 4.7 Clasificación automática
- 4.8 COCOMO 81
- 4.9 COCOMO II
- 4.10 Coste del Producto

Pablo.Bermejo@uclm.es

4.3 Price-to-Win

- De manera informal, se refiere al hecho de que el esfuerzo o costos estimados depende del dinero que tenga el cliente.
- Pero en realidad es un proceso más ético: se captura una especificación poco detallada de los requisitos y el PM pone un precio. Si se firma el contrato, entonces se gestiona el **'poco dinero' retocando el alcance**:
 - El PM realiza una planificación de gestión de riesgos.
 - Comienzan las negociaciones para añadir requisitos de forma detallada.
 - El PM indica el aumento en el precio a cambio de aceptar nuevos requisitos (bottom-up)
 - El PM indica el aumento en el precio a cambio de controlar los riesgos que puedan aparecer con los nuevos requisitos.
 - El cliente selecciona que (sub)funcionalidad y seguridad ante riesgos quiere obtener del proyecto. Es raro que decida abortar el proyecto conforme va cambiando el precio (el contrato inicial tuvo que ser competitivo, ya que pidió varios presupuestos).
- Si el cliente pide un sistema con muchos requisitos pero no tiene mucho dinero, obtendrá un producto peor de lo que quiere.
- La estimación realizada para cada escenario no se basa en parámetros de información histórica, por lo que dependerá de la habilidad del PM.

4.3 Price-to-Win

Price-To-Win Flow & Responsibilities

Momentos posibles para
firmar el contrato inicial.
¡Corre, no sabes a quién más
le han pedido presupuesto!

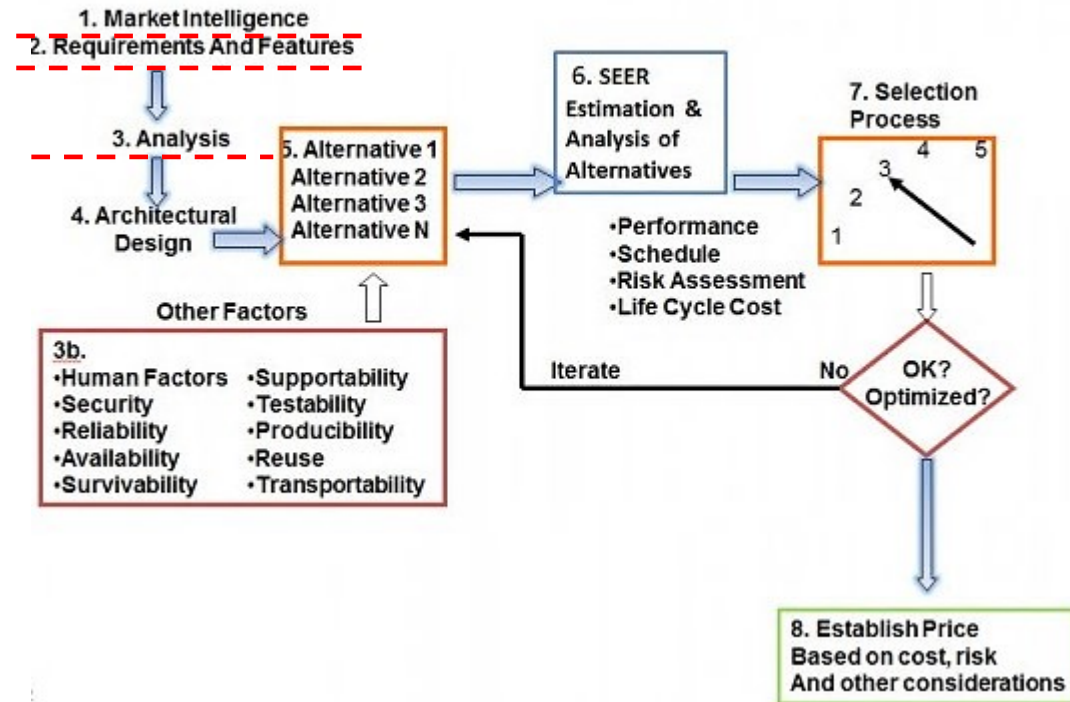


Diagrama obtenido de:

<http://www.galorath.com/wp/price-to-win-what-it-is-and-when-estimation-gets-involved.php>

4.3 Price-to-Win

- Es muy criticado por los usuarios de métodos de estimación Algorítmicos. [Boehm,81] expresa su confianza en que su primera versión de COCOMO acabe con el uso de price-to-win, y lo critica con el siguiente ejemplo:

'I know the cost model estimated \$2million for this job, and none of your experts believe we can do it for less than a million and a half. But I also know that the customer has only \$1million budgeted for this software contract, so that's what we're gonna bid. Now go and fix up the cost estimate and make it look credible.'

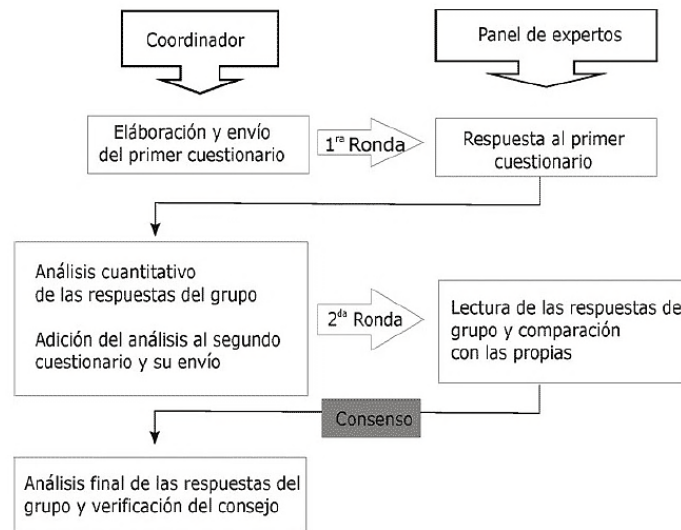
- 4.1 Introducción**
- 4.2 Ley de Parkinson**
- 4.3 Price-to-Win**
- 4.4 (Wideband) Delphi**
- 4.5 Planning Poker
- 4.6 CBR
- 4.7 Clasificación automática
- 4.8 COCOMO 81
- 4.9 COCOMO II
- 4.10 Coste del Producto

Pablo.Bermejo@uclm.es

4.4 (Wideband) Delphi

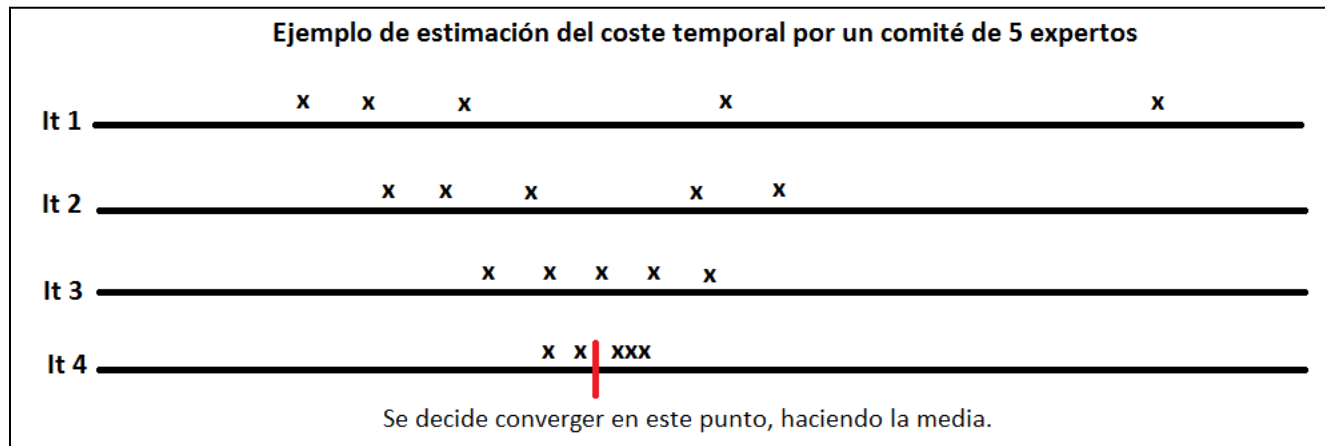
- El método **Delphi** es el método estándar en la Comisión de Expertos. Consiste en varias iteraciones, dirigidas por un *facilitador o coordinador*, en las que cada experto:
 - Da su aproximación anónima (utilizando cualquier método de estimación oficial, o siguiendo asunciones propias).
 - Todos reciben las previsiones del resto, y se opina en grupo

El proceso converge cuando hay una diferencia mínima entre todos o cuando esto no ocurre tras 2 ó 3 iteraciones.



4.4 (Wideband) Delphi

- Es el mismo autor de COCOMO 81 y II quien, antes de publicar la primera versión de COCOMO, propone una variación de Delphi.
- **Wideband Delphi** supone un mayor intercambio de ideas entre los expertos:
 - Antes de la primera iteración, se discute el método sugerido de estimación del esfuerzo.
 - Se discute en grupo las tareas cuyo esfuerzo calculado varía más.
 - Se realizan tantas iteraciones como se consideren oportunas.



- Una discusión muy interesante sobre Delphi:
<http://www.westknollwoodprecinct.com/the-delphi-technique.html>

- 4.1 Introducción**
- 4.2 Ley de Parkinson**
- 4.3 Price-to-Win**
- 4.4 (Wideband) Delphi**
- 4.5 Planning Poker**
- 4.6 CBR
- 4.7 Clasificación automática
- 4.8 COCOMO 81
- 4.9 COCOMO II
- 4.10 Coste del Producto

Pablo.Bermejo@uclm.es

4.5 Planning Poker

- Es una variación de Wideband Delphi y el método de estimación del tamaño Puntos Historia.
- Las estimaciones se realizan principalmente con una baraja de **series de fibonacci** (modificadas para que no parezcan relaciones matemáticas) ya que el esfuerzo no es lineal con el tamaño del software.
- Cada estimación es para el siguiente incremento en la funcionalidad del producto. Las **estimaciones más extremas** son justificadas antes de comenzar una iteración.
- Se da poco tiempo para cada estimación.
- La estimación del encargado de programar una historia de usuario tiene un **mayor peso** al realizar la media final.
- En Scrum se denomina **Scrum Poker**.



- 4.1 Introducción**
- 4.2 Ley de Parkinson**
- 4.3 Price-to-Win**
- 4.4 (Wideband) Delphi**
- 4.5 Planning Poker**
- 4.6 CBR**
- 4.7 Clasificación automática
- 4.8 COCOMO 81
- 4.9 COCOMO II
- 4.10 Coste del Producto

Pablo.Bermejo@uclm.es

2.6 Case-Based Reasoning (CBR)

- El **Razonamiento Basado en Casos** consiste en comparar, matemáticamente, la similitud de proyectos pasados con el que queremos estimar.
- Es bastante útil cuando nuestra empresa dispone de **información histórica** de proyectos similares: las variables más importantes son el tamaño del equipo y el tamaño estimado.
- También existen repositorios públicos de proyectos (<http://www.isbsg.org/>) (*una de las propuestas del Trabajo 2*).
- Es necesario ser coherentes y ser capaces de reconocer que no disponemos de **información de suficiente calidad** para ejecutar un proceso de CBR.

2.6 Case-Based Reasoning (CBR)

Ejemplo 1: Razonamiento basado en Tamaño y caso más parecido

Proyecto ID	Categoría	Lenguaje	Tamaño del Equipo	Tamaño [Real o Estimado] del Software	ESFUERZO
1	Tiempo-Real	C++	10	200	1000
2	Simulador	C++	9	175	950
NUEVO	Tiempo-Real	C++	10	150	?

- 4 variables comparables: Categoría, Lenguaje, T. Equipo y T. Software
 - El proyecto 1 y el NUEVO se parecen sobre un 90%.
 - El proyecto 2 y el NUEVO se parecen sobre un 50%
- Hacemos regla de tres entre el Proyecto 1 y el Nuevo usando el Tamaño del Software como parámetro principal.

$$\text{Esfuerzo}_{\text{NUEVO}} = \frac{150 \times 1000}{200} = 750$$

2.6 Case-Based Reasoning (CBR)

Ejemplo 2: Razonamiento basado en Tamaño y media de todos los casos

Proyecto ID	Categoría	Lenguaje	Tamaño del Equipo	Tamaño [Real o Estimado] del Software	ESFUERZO
1	Tiempo-Real	C++	10	200	1000
2	Simulador	C++	9	175	950
NUEVO	Tiempo-Real	C++	10	150	?

- 4 variables comparables: Categoría, Lenguaje, T. Equipo y T. Software
 - El proyecto 1 y el NUEVO se parecen sobre un 90%.
 - El proyecto 2 y el NUEVO se parecen sobre un 50%

$$\text{Esfuerzo}_{\text{NUEVO}} = \frac{\left(\frac{150 \times 1000}{200} + \frac{150 \times 950}{175} \right)}{2} \approx 782$$

2.6 Case-Based Reasoning (CBR)

Ejemplo 3: Razonamiento basado Temaño y media de todos los casos ponderados por similitud.

Proyecto ID	Categoría	Lenguaje	Tamaño del Equipo	Tamaño [Real o Estimado] del Software	ESFUERZO
1	Tiempo-Real	C++	10	200	1000
2	Simulador	C++	9	175	950
NUEVO	Tiempo-Real	C++	10	150	?

- 4 variables comparables: Categoría, Lenguaje, T. Equipo y T. Software
 - El proyecto 1 y el NUEVO se parecen sobre un 90%.
 - El proyecto 2 y el NUEVO se parecen sobre un 50%
 - 90+50=140.

$$\text{Esfuerzo}_{\text{NUEVO}} = \frac{150 \times 1000}{200} \times \frac{9}{14} + \frac{150 \times 950}{175} \times \frac{5}{14} \approx 773$$

Ejemplo adaptado de [Software Cost Metrics. B.H. Far.]

2.6 Case-Based Reasoning (CBR)

Ejercicio 1: En la base de datos de tu empresa se ha encontrado información de 2 proyectos similares al nuevo proyecto. Utilizando CBR, estima el coste en esfuerzo necesario para implementar el nuevo software.

Haremos un Juicio o Comité de Expertos con el método Delphi con 2 iteraciones, y nos quedaremos con la media de las estimaciones.

Proyecto ID	Categoría	Lenguaje	Tamaño del Equipo	Tamaño Funcional [Real o Estimado] del Software	ESFUERZO
1	OO	Java	8	100	800
2	Empotrado	C	12	70	900
NUEVO	C++	C++	10	90	?

- 4.1 Introducción**
- 4.2 Ley de Parkinson**
- 4.3 Price-to-Win**
- 4.4 (Wideband) Delphi**
- 4.5 Planning Poker**
- 4.6 CBR**
- 4.7 Clasificación automática**
- 4.8 COCOMO 81
- 4.9 COCOMO II
- 4.10 Coste del Producto

Pablo.Bermejo@uclm.es

4.7 Clasificación automática

- A partir de un repositorio de proyectos descritos por decenas o cientos de variables, resulta más costoso realizar CBR.
- Sin embargo, el Aprendizaje Automático (Machine Learning) y, en concreto, la **Clasificación Automática**, permite construir automáticamente clasificadores a partir de la base de datos de entrenamiento, y predecir el valor de la variable clase; en este caso, el Esfuerzo. Si la variable a predecir es numérica, hablamos de **Regresión**.
- Además del aritmético, como en CBR, los clasificadores pueden pertenecer a diferentes paradigmas, lo cual enriquece el tipo de comparación:
 - Espacial: k-NN
 - Funcional o vectorial: Support Vector Machine
 - Probabilístico: Naive Bayes, TAN
 - Árboles de decisión: CART, c4.5

4.7 Clasificación automática

- Normalmente los clasificadores darán una predicción, pero muchos son de caja negra y no pueden explicar su proceso de inferencia (c4.5 o NB sí pueden).
- Otra gran ventaja del aprendizaje automático es la existencia de un abanico amplísimo para la **selección automática** de las variables más relevantes:
 - Proyectos descritos por decenas o cientos de variables.
 - Según el ámbito o el clasificador, algunas variables pueden ser importantes y en otros casos ya no.
 - Selección voraz, con aleatoriedad guiada, hacia delante, hacia atrás,...
- No es un método muy extendido en la empresa privada debido a que el ingeniero del software no suele estar formado en Inteligencia Artificial o en concreto Aprendizaje Automático. En entornos de investigación, hay multitud de artículos relevantes.
- Lectura de interés: *[An Investigation of Machine Learning Based Prediction Systems. Mair et al. 1999]*

- 4.1 Introducción**
- 4.2 Ley de Parkinson**
- 4.3 Price-to-Win**
- 4.4 (Wideband) Delphi**
- 4.5 Planning Poker**
- 4.6 CBR**
- 4.7 Clasificación automática**
- 4.8 COCOMO 81**
- 4.9 COCOMO II
- 4.10 Coste del Producto


Pablo.Bermejo@uclm.es

4.8 COCOMO 81

- Todos los **modelos algorítmicos** se basan en el ajuste de fórmulas de regresión (no lineal), a partir de información histórica de proyectos.
 - La variable que más afecta al valor predicho es el Tamaño del Software. Éste se medirá en LOC o FPs.
 - Hay otros factores que se tienen en cuenta: equipo, dificultad, lenguaje, tecnología,...
 - Prácticamente todos siguen la forma:

$$\textit{Esfuerzo} = a \times \textit{Tamaño}^b \times m$$



a, b y m dependen del nivel de detalle con que se utilice COCOMO.

b: entre 1 y 1.5. ¿Qué indica que B potencie al Tamaño? 

m: calculado a partir de los factores (controladores de coste) que se quieran tener en cuenta.

- Veremos COCOMO 81 y COCOMO II.

4.8 COCOMO 81

- **Constructive Cost Model** (Bohem, 1981): conocido por COCOMO o, para no confundirlo con su segunda versión, COCOMO 81. Sus fórmulas han sido ajustadas a partir de información de 63 proyectos ejecutados de 1964 a 1979, principalmente en FORTRAN y Ensamblador. 
- Asume un ciclo de vida del proyecto de Cascada.
- Su **unidad de tamaño** en KDSI: miles de instrucciones fuente entregadas. La definición que da COCOMO 81 sería compatible con NCLOC:
 - No se cuentan los comentarios
 - Si 2 instrucciones están en una misma línea, se cuentan como 1.
- Su **unidad de Esfuerzo** es el man-months (MM) o person-months (PM); estimando 1MM=152horas/mes.
- Distingue 3 **niveles** de detalle en la **estimación del esfuerzo para el DISEÑO, DESARROLLO y TESTS**: 
 - Básico: aconsejable para las primeras negociaciones con el cliente.
 - Intermedio: es necesario disponer de una buena especificación de requisitos.
 - Avanzado: cuando tenemos el sistema diseñado. Justo antes de comenzar a programar.

4.8 COCOMO 81

- Distingue **3 niveles** de dificultad del **desarrollo**:
 - Orgánico: Equipos pequeños, donde todos los miembros están familiarizados con la categoría del software a desarrollar. No es necesaria mucha comunicación, hay una alta productividad y el código a desarrollar es bajo (<50 KDSI).
 - Semi-encajado (*semi-attached*): se considera semi-encajado si comparte la mitad de las características del un desarrollo orgánico y la mitad de empotrado; o, en general, si el equipo no tiene mucha experiencia con la categoría del software a desarrollar. Tamaño <300 KDSI.
 - Empotrado: desarrollo de una software con fuertes restricciones técnicas (ej: controlador de aviación), los requisitos iniciales no se pueden cambiar, y hace falta un equipo amplio, lo cual deriva en poca productividad, pero es necesario por la dificultad del proyecto.
- Así, tenemos 3 niveles de estimación del esfuerzo x 3 niveles de dificultad en el desarrollo = 9 conjuntos de valores para {a,b,m} en

$$\text{Esfuerzo} = a \times \text{Tamaño}^b \times m$$

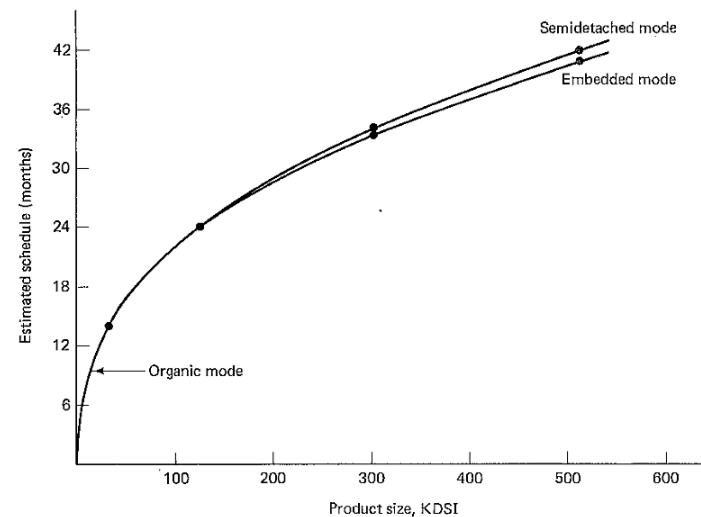
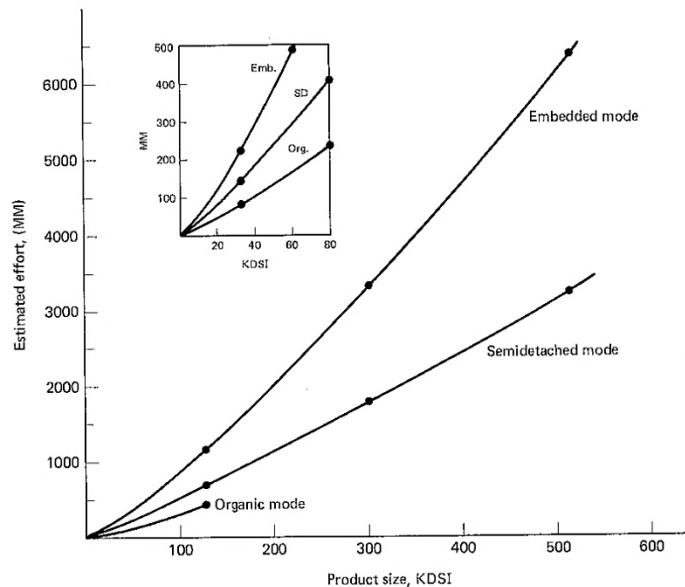
4.8 COCOMO 81 - Básico

- El modelo **COCOMO 81 Básico** no tiene en cuenta factores a parte del tamaño, así que $m=1$, y queda:

$$Esfuerzo(MM) = a \times KDSI^b$$

$$Tiempo(meses) = 2.5 \times MM^c$$

Desarrollo	a	b	c
Orgánico	2.4	1.05	0.38
Semi-encajonado	3.0	1.12	0.35
Empotrado	3.6	1.20	0.31




Gráficas obtenidas de [Software Engineering Economics. Boehm. 1981]

4.8 COCOMO 81 - Intermedio

- COCOMO 81 Intermedio** añade a la estimación Básica información de 15 factores, denominados **atributos controladores del coste** (*cost drivers*) y repartidos en 4 categorías. Su contribución al Esfuerzo depende de su valoración:

Categoría del controlador	Controlador	Aportación al esfuerzo según Valoración del Controlador					
		Muy bajo	Bajo	Nominal	Alto	Muy Alto	Extra Alto
Producto	RELY – Required Software Reliability	.75	.88	1.00	1.15	1.40	-
	DATA – Data Base Size	-	.94	1.00	1.08	1.16	-
	CPLX – Product Complexity	0.70	0.85	1.00	1.15	1.30	1.65
Máquina	TIME – Execution Time Constraint	-	-	1.00	1.11	1.30	1.66
	STORE – Main Storage Constraint	-	-	1.00	1.06	1.21	1.56
	VIRT – Virtual Machine Volatility	-	0.87	1.00	1.15	1.30	-
	TURN – Computer Turnaround Time	-	0.87	1.00	1.07	1.15	-
Personal	ACAP – Analyst Capability	1.46	1.19	1.00	0.6	0.71	-
	AEXP – Applications Experience	1.29	1.13	1.00	0.91	0.82	-
	PCAP – Programmer Capability	1.42	1.17	1.00	0.86	0.70	-
	VEXP – Virtual Machine Experience	1.21	1.10	1.00	0.90	-	-
	LEXP – Programming Language Experience	1.14	1.07	1.00	0.95	-	-
Proyecto	MODP – Modern Programming Practices	1.24	1.10	1.00	0.91	0.82	-
	TOOL – Use of Software Tools	1.24	1.10	1.00	0.91	0.83	-
	SCED – Required Development Schedule	1.23	1.08	1.00	1.04	1.10	-

- ¿Qué factores tienen más influencia en el esfuerzo, además del Tamaño, según COCOMO 81 Intermedio?  ¡ATENCIÓN A LA HORA DE NEGOCIAR REQUISITOS!
- En la Tabla 8-3 de [Boehm, 1981] tenéis una descripción detallada para valorar los controladores.

4.8 COCOMO 81 - Intermedio

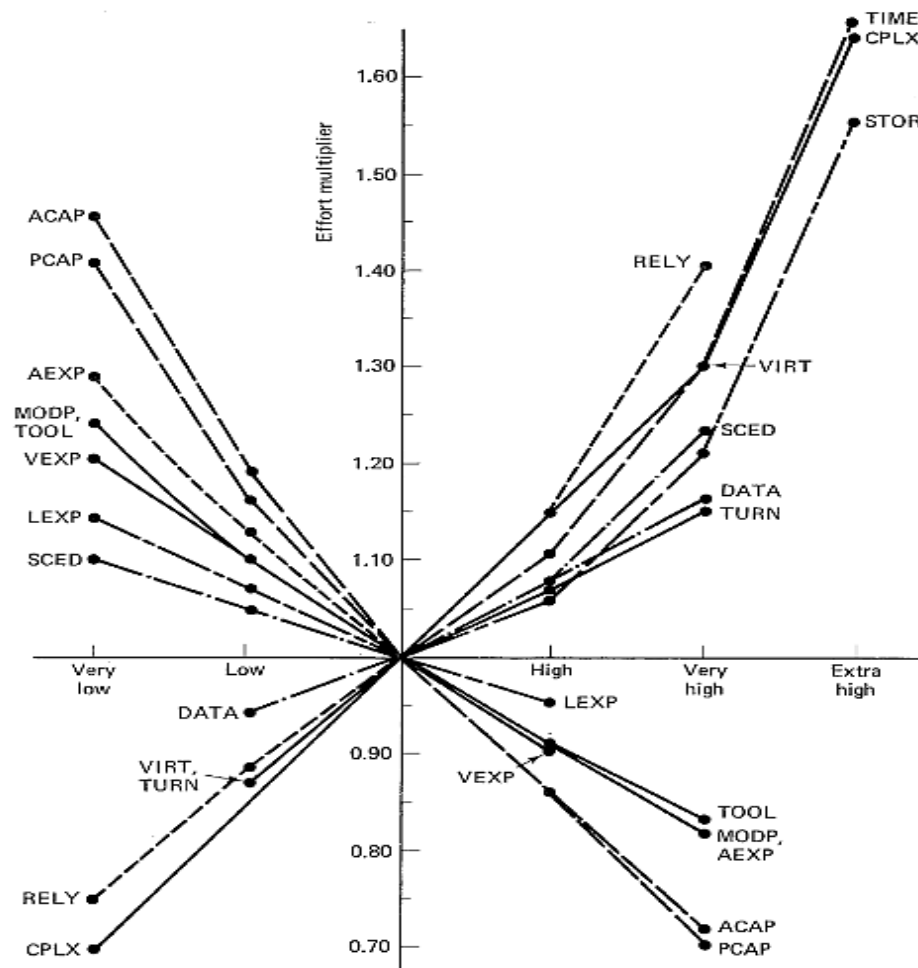


FIGURE 8-2 Intermediate COCOMO effort multipliers

El Factor de Ajuste del Esfuerzo (EAF) es el producto del valor dado a todos los controladores del coste

Un proyecto en el que la habilidad de los programadores es Muy Alta, varía el esfuerzo respecto a otra con habilidad Baja en un (%)...

(manteniendo el resto de factores constantes)

Un software en el que se maneja una base de datos Muy Alta, varía el esfuerzo respecto a una base de datos Baja en un (%)...

(manteniendo el resto de factores constantes)

4.8 COCOMO 81 - Intermedio

$$EAF = \prod_{i=1}^{15} \text{controlador}_i$$

$$\text{Esfuerzo}_{\text{nominal}}(\text{MM}) = a \times \text{KDSI}^b$$

$$\text{Esfuerzo}(\text{MM}) = \text{Esfuerzo}_{\text{nominal}} \times EAF$$

$$\text{Tiempo}(\text{meses}) = 2.5 \times \text{MM}^c$$

Desarrollo	a	b	c
Orgánico	3.3	1.05	0.38
Semi-encajonado	3.0	1.12	0.35
Empotrado	2.8	1.20	0.32

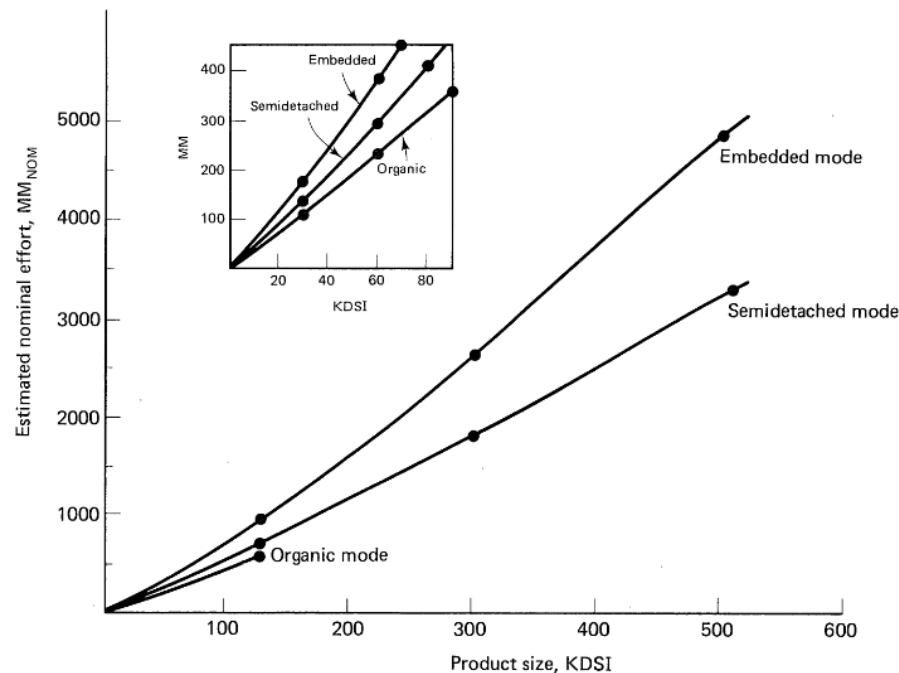


FIGURE 8-1 Intermediate COCOMO: Nominal effort estimates

Gráfica obtenida de [Software Engineering Economics. Boehm. 1981]

4.8 COCOMO 81 - Avanzado

- **COCOMO 81 Avanzado o Detallado** considera que la **influencia de los controladores de coste es diferente según la fase**, con lo cual añade una nueva dimensión a las tablas, la Fase de desarrollo:
 - Requisitos y Planificación (RP)
 - Diseño Detallado (DD)
 - Código y Test (CUT)
 - Integración y Tests (IT)
- Estos valores vendrán en cualquier software que implemente COCOMO 81. Pueden consultarse en las Tablas 23-2 y 23-3 de [Boehm,81].
- Las fórmulas de estimación de esfuerzo y tiempo son las mismas que en el nivel Intermedio.

4.8 COCOMO 81 – distribución del esfuerzo

- La cantidad media de personal a contratar es **FSP** (Full-time-equivalente Software Personnel) = Esfuerzo/Tiempo.
- Pero la **distribución del personal depende de la fase** del proyecto y el tipo de desarrollo.
- Ejemplo para COCOMO Básico con un tamaño de desarrollo medio (32KDSI)

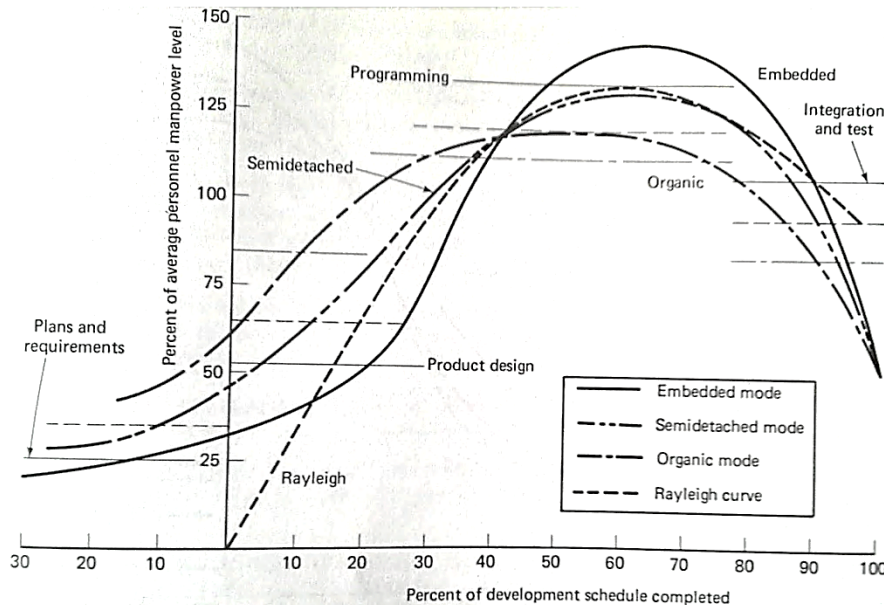


FIGURE 6-8 Basic COCOMO personnel distribution: Medium (32 KDSI) projects

Gráfica obtenida de [Software Engineering Economics. Boehm. 1981]

Una buena aproximación a estas curvas, que no tiene en cuenta el tipo de desarrollo, es la distribución de Rayleigh, sobretodo para el modo semi-encajado:

$$FSP_{RayleighBasico} = 13,300 \frac{t}{3600} e^{-t^2/7200}$$

siendo t el % del tiempo de ejecución para el cual se calcula FSP

4.8 COCOMO 81 – distribución del esfuerzo

% del Esfuerzo y del Tiempo estimados según la fase del proyecto:

TABLE 6-8 Phase Distribution of Effort and Schedule: All Modes

Effort distribution		Size				
		Small 2 KDSI	Inter- mediate 8 KDSI	Medium 32 KDSI	Large 128 KDSI	Very Large 512 KDSI
Mode	Phase					
Organic	Plans and requirements (%)	6	6	6	6	
	Product design	16	16	16	16	
	Programming	68	65	62	59	
	Detailed design	26	25	24	23	
	Code and unit test	42	40	38	36	
	Integration and test	16	19	22	25	
Semidetached	Plans and requirements (%)	7	7	7	7	7
	Product design	17	17	17	17	17
	Programming	64	61	58	55	52
	Detailed design	27	26	25	24	23
	Code and unit test	37	35	33	31	29
	Integration and test	19	22	25	28	31
Embedded	Plans and requirements (%)	8	8	8	8	8
	Product design	18	18	18	18	18
	Programming	60	57	54	51	48
	Detailed design	28	27	26	25	24
	Code and unit test	32	30	28	26	24
	Integration and test	22	25	28	31	34
Schedule distribution		2 KDSI	8 KDSI	32 KDSI	128 KDSI	512 KDSI
Organic	Plans and requirements (%)	10	11	12	13	
	Product design	19	19	19	19	
	Programming	63	59	55	51	
	Integration and test	18	22	26	30	
Semidetached	Plans and requirements (%)	16	18	20	22	24
	Product design	24	25	26	27	28
	Programming	56	52	48	44	40
	Integration and test	20	23	26	29	32
Embedded	Plans and requirements (%)	24	28	32	36	40
	Product design	30	32	34	36	38
	Programming	48	44	40	36	32
	Integration and test	22	24	26	28	30

[Software Engineering Economics. Boehm. 1981]

4.8 COCOMO 81

- Motivos para No querer a COCOMO 81:
 - Asume ciclo de vida de cascada, mientras que ahora se tiende más a entregas de funcionalidad incremental.
 - Asume uso de lenguajes imperativos como FORTRAN y C.
 - No tiene en cuenta la reutilización de código y uso de herramientas de generación automática de código.
- Motivos para querer a COCOMO 81:
 - Fue el primer gran método algorítmico de estimación del esfuerzo.
 - Aún es usado y referenciado, aunque sea de forma comparativa.

*No haremos ejercicios ni prácticas con COCOMO 81, sino con COCOMO II, de más actualidad.
Si estáis interesados en ejemplos prácticos, podéis pedir prestado en la biblioteca el libro
Software Engineering Economics. Boehm. 1981.*

- 4.1 Introducción**
- 4.2 Ley de Parkinson**
- 4.3 Price-to-Win**
- 4.4 (Wideband) Delphi**
- 4.5 Planning Poker**
- 4.6 CBR**
- 4.7 Clasificación automática**
- 4.8 COCOMO 81**
- 4.9 COCOMO II**
- 4.10 Coste del Producto**

Pablo.Bermejo@uclm.es

4.9 COCOMO II

- En 2000 se publica [*Software Cost Estimation With **COCOMO II**. Boehm et al*], presentando una nueva versión de COCOMO que supone un **desarrollo en espiral**. También se conoce como COCOMO 2000.
- Los parámetros de sus fórmulas se ajustan a partir de un historial de 161 proyectos, y están calibrados para tamaños mayores de 2 KSLOC,
- Define **3 niveles de estimación**, según la fase de desarrollo y, por lo tanto, el nivel de detalle del que se dispone:
 - Construcción de Prototipos (*Application-composition* en el original): conveniente para estimación del esfuerzo inicial, en la que se construyen prototipos para resolver riesgos en decisiones tomadas sobre interfaces, ventanas, métodos de interacción,... Estos prototipos son contruidos a base de scripts, código reutilizado y llamadas a bases de datos.
 - Diseño Inicial o Temprano (*Early Design*): aconsejado cuando ya han quedado claros los requisitos, pero aún no se dispone de una arquitectura final.
 - Postarquitectura (*Post-architecture*): es la estimación con más detalle para calcular el esfuerzo en el desarrollo del software.
- Al igual que en COCOMO 81, las **estimaciones de** esfuerzo y tiempo son del **diseño y desarrollo** del software: no se tiene en cuenta la planificación y captura de requisitos.

4.9 COCOMO II

- Junto al nivel de Postarquitectura, se puede realizar la estimación del esfuerzo de **Reutilización de código**, si procede, con una metodología también propuesta en COCOMO II.
- Dependiendo del nivel de estimación, se utilizan las siguientes unidades de tamaño del software:
 - Construcción de Prototipos: **Puntos de Aplicación**. Son los Object Points , pero se les cambia el nombre para evitar confusión con la programación OO.
 - Diseño Inicial: **Puntos Función NO AJUSTADOS** (UFP), siguiendo la metodología FPA. Y luego pide convertir **UFP a KSLOC**.
 - Post arquitectura: KSLOC (o UFP primero)
- KSLOC: Se refieren a Declaraciones Lógicas de código, primero descritas en el SEI (Software Engineering Institute). Páginas 67 a 71 en *[Software Size Measurement: A Framework for Counting Source Statements]* <http://www.sei.cmu.edu/reports/92tr020.pdf,y> modificadas ligeramente en el texto de referencia de COCOMO II (Tabla 2.53). En <http://sunset.usc.edu/research/CODECOUNT/> podemos descargar un programa que calcula el tamaño de un software aportando diferentes métricas, entre las que se encuentran las KSLOC definidas por SEI con las modificaciones necesarias para COCOMO II.
- El Esfuerzo se mide en **person-months** (PM), que equivale al man-months (MM) de COCOMO 81.

4.9 COCOMO II - Prototipos

- El nivel de **Construcción de Prototipos** (Application-Composition) nos indica el esfuerzo a partir de **prototipos que enseñaremos al cliente** para afinar los requisitos. Por ello los AP (=Object Points) son adecuados en este caso ya que tienen en cuenta el aspecto de las interfaces e informes.
- Como vimos en el Tema 2, al tamaño AP calculado hay que restarle el % de módulos o código reutilizado, aunque **no se tiene en cuenta el esfuerzo en integrar** dicho código.
- Se requiere conocer la productividad $PROD=(AP/mes)$ media del equipo. Aunque nos proponen la siguiente aproximación:

Experiencia y capacidad	Muy baja	Baja	Nominal	Alta	Muy Alta
Uso de CASE	Muy baja	Baja	Nominal	Alta	Muy Alta
PROD	4	7	13	25	50

$$Esfuerzo_{ConstrucciónPrototipos}(PM) = \frac{AP \times (1 - \frac{\%reuso}{100})}{PROD}$$

4.9 COCOMO II – Diseño Inicial

- El nivel de estimación **Diseño Inicial** nos permite, a partir de los requisitos funcionales capturados, poder hacer varias estimaciones de esfuerzo según subconjuntos de estos requisitos. Así, podemos decidir (o negociar con el cliente) la funcionalidad final antes de diseñar el software e implementarlo.

$$Esfuerzo_{DiseñoInicial}(PM) = 2.94 \times KSLOC^{0.91 + (0.01 \times \sum_{j=1}^5 SF_j)} \times \prod_{i=1}^7 EM_i$$

$$Tiempo_{DiseñoInicial}(\text{meses}) = 3.67 \times PM^{0.28 + (0.002 \times \sum_{j=1}^5 SF_j)}$$

- KSLOC es la conversión de UFP a KSLOC según aproximaciones facilitadas (ver página siguiente).
- La relación no lineal del tamaño con el esfuerzo (b) varía según la suma de 5 **factores de escalado exponencial** (SF).
- Los controladores del coste ahora se llaman **multiplicadores del esfuerzo** (EM) y son 7.

4.9 COCOMO II – Diseño Inicial

- Equivalencia de SLOCs por 1 UFP.

Table 2.4 Default UFP to SLOC Conversion Ratios

Language	SLOC / UFP	Language	SLOC / UFP
Access	38	Jovial	107
Ada 83	71	Lisp	64
Ada 95	49	Machine Code	640
AI Shell	49	Modula 2	80
APL	32	Pascal	91
Assembly—Basic	320	PERL	27
Assembly—Macro	213	PowerBuilder	16
Basic—ANSI	64	Prolog	64
Basic—Compiled	91	Query—Default	13
Basic—Visual	32	Report Generator	80
C	128	Second Generation Language	107
C++	55	Simulation—Default	46
Cobol (ANSI 85)	91	Spreadsheet	6
Database—Default	40	Third Generation Language	80
Fifth Generation Language	4	Unix Shell Scripts	107
First Generation Language	320	USR_1	1
Forth	64	USR_2	1
Fortran 77	107	USR_3	1
Fortran 95	71	USR_4	1
Fourth Generation Language	20	USR_5	1
High Level Language	64	Visual Basic 5.0	29
HTML 3.0	15	Visual C++	34
Java	53		

- Tabla obtenida de [Software Cost Estimation With COCOMO II. Boehm et al. 2000]

4.9 COCOMO II – Diseño Inicial

Los Factores de Escalado SF son 5:

PRE	Precedentes	Similaridad con proyectos pasados.
FLEX	Flexibilidad en el Desarrollo	Flexibilidad en la implementación final de los requisitos.
RESL	Resolución de Arquitectura y Riesgos	Nivel de esfuerzo y capacidad en la gestión de riesgos y arquitectura
TEAM	Cohesión del Equipo	Similaridad en la cultura y objetivos de los interesados. Capacidad para mejorarla.
PMAT	Madurez del Proceso	Madurez según el CMM. Nivel de 1 a 5 o medido según Tabla 2.16 en [Boehm et al., 2000]

Table 2.10 Scale Factor Values SF_i for COCOMO II Models

Scale Factors	Very Low	Low	Nominal	High	Very High	Extra High
PREC	thoroughly unprecedented	largely unprecedented	somewhat unprecedented	generally familiar	largely familiar	thoroughly familiar
SF_i	6.20	4.96	3.72	2.48	1.24	0.00
FLEX	rigorous	occasional relaxation	some relaxation	general conformity	some conformity	general goals
SF_i	5.07	4.05	3.04	2.03	1.01	0.00
RESL	little (20%)	some (40%)	often (60%)	generally (75%)	mostly (90%)	full (100%)
SF_i	7.07	5.65	4.24	2.83	1.41	0.00
TEAM	very difficult interactions	some difficult interactions	basically cooperative interactions	largely cooperative	highly cooperative	seamless interactions
SF_i	5.48	4.38	3.29	2.19	1.10	0.00
----- The estimated Equivalent Process Maturity Level (EPML) or -----						
PMATSW-CMM	SW-CMM	SW-CMM	SW-CMM	SW-CMM	SW-CMM	SW-CMM
Level 1	Level 1	Level 2	Level 3	Level 4	Level 5	
Lower	Upper					
SF_i	7.80	6.24	4.68	3.12	1.56	0.00

Tabla de asignación rápida para los SF [Software Cost Estimation With COCOMO II. Boehm et al. 2000]

4.9 COCOMO II – Diseño Inicial

- El Diseño Inicial tiene **7 multiplicadores del esfuerzo** (EM) (efecto multiplicador de los controladores del coste), que en sí son 7 agregaciones de los 17 controladores del nivel Postarquitectura.
- A cada uno de los 17 controladores se les otorga una etiqueta Muy Bajo (1), Bajo (2), Nominal (3), Alto(4), Muy alto(5).
- Para obtener el valor de los 7 controladores del Diseño Inicial, se puede hacer de 2 formas:
 - Sumar los valores de los 16 controladores de Postarquitectura
 - Otorgar directamente una de estas etiquetas: Extra Bajo, Muy Bajo, Bajo, Nominal, Grande, Muy Grande, Extra Grande. Esta es la más lógica ya que si se dispone del detalle para los 17 controladores, haríamos una estimación Postarquitectura., y luego obtener el efecto multiplicador de la tabla de la página siguiente.

4.9 COCOMO II – Diseño Inicial

Valor de los 7 EM de Diseño Inicial

Controlador Diseño Inicial	Suma las etiquetas de...	XB	MB	Bajo	Nomin	Grande	MG	XG
RCPX	RELY, DATA, CPLX, DOCU	5-6	7-8	9-11	12	13-15	16-18	19-21
		0.49	0.60	0.83	1.0	1.33	1.91	2.72
RUSE	RUSE	-	-	3	4	5	6	7
		-	-	0.95	1.0	1.07	1.15	1.24
PDIF	TIME, STOR, PVOL	-	-	9	10-12	13-15	-	-
		-	-	1.0	1.0	1.0	-	-
PERS	ACAP, PCAP, PCON	3-4	5-6	7-8	9	10-11	12-13	14-15
		2.12	1.62	1.26	1.0	0.83	0.63	0.50
PREX	APEX, PLEX, LTEX	3-4	5-6	7-8	9	10-11	12-13	14-15
		1.59	1.33	1.12	1.0	0.87	0.74	0.62
FCIL	TOOL, SITE	2	3	4-5	6	7-8	9-10	11
		1.43	1.30	1.10	1.0	0.87	0.73	0.62
SCED	SCED	-	2	3	4	5	6	-
		-	1.43	1.14	1.0	1.0	1.0	-

4.9 COCOMO II – Postarquitectura

- El nivel de estimación **Postarquitectura** utiliza las mismas fórmulas que el Diseño Inicial, solo cambia el número de controladores del coste.
- Ahora son 17 multiplicadores del esfuerzo (nombrados en la tabla anterior y siguiente).
- Los factores escalares son los mismos y se evalúan igual que en Diseño Inicial.

$$Esfuerzo_{Postarq}(PM) = 2.94 \times KSLOC^{0.91+(0.01 \times \sum_{j=1}^5 SF_j)} \times \prod_{i=1}^{17} EM_i$$

$$Tiempo_{Postarq}(\text{meses}) = 3.67 \times PM^{0.28+(0.002 \times \sum_{j=1}^5 SF_j)}$$

- Si no se tiene en cuenta el EM de SCED, entonces el Esfuerzo y Tiempo calculados se dice que son Nominales.

4.9 COCOMO II – Postarquitectura

Categoría		MB	Bajo	Nom	Alto	MA	XA	Descripción
Producto	<i>RELY</i>	0.82	0.92	1.0	1.10	1.26	-	Fiabilidad necesaria. Coste del fallo.
	<i>DATA</i>	-	0.90	1.0	1.14	1.28	-	Esfuerzo necesario para generar y mantener datos de test.
	<i>CPLX</i>	0.73	0.87	1.0	1.17	1.34	1.74	Complejidad de las interfaces, cálculos y manejo de datos.
	<i>RUSE</i>	-	0.95	1.0	1.07	1.15	1.24	Esfuerzo derivado de la reutilización de código.
	<i>DOCU</i>	0.81	0.91	1.0	1.11	1.23		Esfuerzo en la documentación.
Plataforma	<i>TIME</i>	-	-	1.0	1.11	1.29	1.63	Restricciones en el tiempo de ejecución.
	<i>STOR</i>	-	-	1.0	1.05	1.17	1.46	Restricciones en uso de memoria principal.
	<i>PVOL</i>	-	0.87	1.0	1.15	1.30	-	Rasa de llamadas del software a los elementos con los que actúa.
Personal	<i>ACAP</i>	1.42	1.19	1.0	0.85	0.71	-	Habilidad del Analista. No tener en cuenta la experiencia.
	<i>PCAP</i>	1.34	1.15	1.0	0.88	0.76	-	Habilidad del equipo de programadores. No tener en cuenta la exper.
	<i>PCON</i>	1.29	1.12	1.0	0.90	0.81	-	Continuidad anual del personal en el equipo.
	<i>APEX</i>	1.22	1.10	1.0	0.88	0.81	-	Experiencia del equipo en este tipo de aplicación.
	<i>PLEX</i>	1.19	1.09	1.0	0.91	0.85	-	Experiencia en la/s plataforma/s a utilizar.
	<i>LTEX</i>	1.20	1.09	1.0	0.91	0.84	-	Experiencia en el lenguaje y herramientas de programación.
Proyecto	<i>TOOL</i>	1.17	1.09	1.0	0.90	0.78	-	Uso de herramientas que facilitan la programación y gestión.
	<i>SITE</i>	1.22	1.09	1.0	0.93	0.86	0.80	Equipo trabaja en la misma zona.
	<i>SCED</i>	1.43	1.14	1.0	1.0	1.0	-	Estrechamiento exigido del cronograma (para así valorar escenarios)

En cursiva los controladores que también aparecen en COCOMO 81.

Hay tablas guía en [Boehm et al, 2000. Cap 2] para seleccionar la etiqueta para cada controlador de coste.

4.9 COCOMO II – Postarquitectura

- Es muy común **reutilizar o adaptar código**. Por lo tanto hay que calcular el esfuerzo invertido en entender, modificar y pegar dicho código:
 - Se mide en líneas de código equivalentes (**EKSLOC**)
 - Se suma al tamaño del software al calcular $Esfuerzo_{Postarq}$.
- $$Esfuerzo_{Postarq}(PM) = 2.94 \times (KSLOC + EKSLOC)^{0.91 + (0.01 \times \sum_{j=1}^5 SF_j)} \times \prod_{i=1}^{17} EM_i$$
- Al esfuerzo final se puede añadir el esfuerzo por usar código traducido automáticamente de un lenguaje a otro: PM_{auto} , pero no lo veremos.
- EKSLOC se deriva del código que hay que nos estamos planteando ajustar (**AKSLOC**). Para calcular EKSLOC primero hay que calcular 4 factores:
 - **AAF** (*Adaptation Adjustment Factor*): $AAF = 0.4DM + 0.3CM + 0.3IM$
 - DM: % del **diseño** del software a reutilizar par ajustarnos a los requisitos del proyecto actual.
 - CM: % del **código** (% de AKSLOC) que hay que modificar.
 - IM: % del esfuerzo requerido para la **integración** y tests, comparado con el esfuerzo esperando en software de tamaño similar.
 - **SU** (*Software Understanding effort*): de 10 a 50. Esfuerzo para entender el código reutilizable.
 - **UNFM** (*Unfamiliarity*): de 0 a 1. Es la familiaridad del equipo con ese tipo de software.
 - **AA** (*Assessment&Assimilation*): de 0 a 8: indica el esfuerzo en decidir si conviene reutilizar el código.

4.9 COCOMO II – Postarquitectura

- Así, el esfuerzo que tendremos que hacer para adaptar un código de AKSLOC será el equivalente a implementar EKSLOC, cifra que se calcula con la siguiente fórmula:

$$EKSLOC = AKSLOC \times \left(1 - \frac{AT}{100}\right) \times AAM$$
$$AAM = \begin{cases} \frac{AA + AAF(1 + (0.02 \times SU \times UNFM))}{100}, & \text{si } AAF \leq 50 \\ \frac{AA + AAF + (SU \times UNFM)}{100}, & \text{si } AAF > 50 \end{cases}$$

AT: % de AKSLOC automáticamente generado.

- Para evitar la subjetividad, en [Boehm,2000] se proporcionan tablas guía (2.5 a 2.7) para decidir los valores de SU, UNFM y AA.
- Si el código no es adaptado, sino **reutilizado** (sin cambio alguno):
 - DM y CM son 0.
 - SU y UNFM no se utilizan.
 - Es el caso de los COTS (Commercial off-the-shelf): código adquirido y reutilizado como caja negra.

4.9 COCOMO II – Distribución y mantenimiento

- Al igual que en COCOMO 81, se estima el reparto del esfuerzo según la fase y, en COCOMO II, también según el ciclo de vida
 - Para Cascada, la distribución es como COCOMO 81 – semi-encajado.
 - Para un desarrollo en espiral siguiendo la metodología RUP:

Fase RUP	%Esfuerzo	%Tiempo
Iniciación	5	10
Elaboración	20	30
Construcción	65	50
Transición	10	10

- La fórmula del esfuerzo también puede aplicarse para calcular, en vez de el coste del desarrollo, el coste de un **mantenimiento**. Cuando el tamaño original es conocido se puede aplicar esta fórmula:

$$Tamaño_{mantenimiento} = (Tamaño_{añadido} + Tamaño_{modificado}) \times \left(1 + \frac{SU \times UNFM}{100}\right)$$

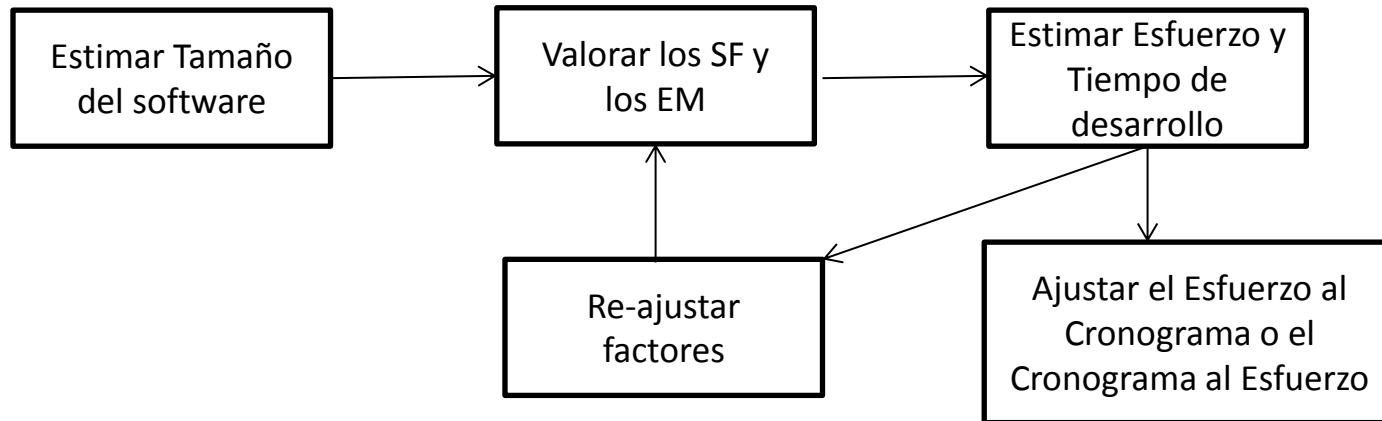
- Se descartan los controladores SCED y RUSE. RELY cambia a: 1.23,1.1,1,0.99,1.07,-.

4.9 COCOMO II

- El mayor problema de COCOMO II es también su mayor ventaja: la gran cantidad de **parámetros** a evaluar. Esto nos proporciona un modelo para evaluar, utilizando los controladores de coste, diferentes **planificaciones para evitar Riesgos y Reducir Costos**.
- Así, no es un problema que las metodologías que presupone sean solo o Waterfall o su propia en espiral MBASE/RUP. Ya que por lo general no se usará para estimar el coste, sino para comparar el coste de un proyecto pasado con otro similar cambiando los parámetros de ajuste (SFi y EMi).
- La evaluación de muchos están rodeados de **incertidumbre** y subjetividad.
- Calibración: se supone que la empresa debe ajustar estos factores a sus propios **valores históricos**... no es común que las empresas tengan tantos datos históricos, al menos de forma limpia y organizada.
- Así, aunque es el modelo más completo, también es el más *costoso (valga la redundancia!)*. Además, frente a la gran evolución y demanda que están teniendo las metodologías de **desarrollo ágil**, los modelos algorítmicos van perdiendo fuerza...

4.9 COCOMO II

Esquema de estimación del coste usando COCOMO II



4.9 COCOMO II

Ejercicio 2. Retorno de la Inversión.

Nuestra empresa quiere saber si el desarrollo de un software para gestionar el mantenimiento del inventario merece la pena, así que nos piden realizar un análisis del Retorno de la Inversión; sabiendo que el uso del software reducirá un 40% de los 600 000 € anuales gastados actualmente en la gestión.

Se calcula que el software medirá 150 KSLOC, y su desarrollo costará 300 000 €.

Anualmente habrá que actualizar un 15% del software, el cual estará muy bien estructurado, y será actualizado en gran parte por el mismo equipo.

Los controladores de coste son nominales para desarrollo y mantenimiento; y el factor exponencial es 1.10.

El coste por empleado en la empresa es 2000€/PM. ¿Cuánto tiempo pasará hasta recibir beneficios?

4.9 COCOMO II

	MB	Bajo	Nom	Alto	MA	XA
RELY	0.82	0.92	1.0	1.10	1.26	-
DATA	-	0.90	1.0	1.14	1.28	-
CPLX	0.73	0.87	1.0	1.17	1.34	1.74
RUSE	-	0.95	1.0	1.07	1.15	1.24
DOCU	0.81	0.91	1.0	1.11	1.23	
TIME	-	-	1.0	1.11	1.29	1.63
STOR	-	-	1.0	1.05	1.17	1.46
PVOL	-	0.87	1.0	1.15	1.30	-
ACAP	1.42	1.19	1.0	0.85	0.71	-
PCAP	1.34	1.15	1.0	0.88	0.76	-
PCON	1.29	1.12	1.0	0.90	0.81	-
APEX	1.22	1.10	1.0	0.88	0.81	-
PLEX	1.19	1.09	1.0	0.91	0.85	-
LTEX	1.20	1.09	1.0	0.91	0.84	-
TOOL	1.17	1.09	1.0	0.90	0.78	-
SITE	1.22	1.09	1.0	0.93	0.86	0.80
SCED	1.43	1.14	1.0	1.0	1.0	-

Ejercicio 3. Ajustando el coste.

El coste de un desarrollo software es estimado en nivel de post-arquitectura con los valores de los controladores marcados en la tabla.

Como gestor de proyecto, te piden que reduzcas el coste monetario del proyecto un 15%, pero tu poder de maniobra está muy limitado, solo puedes actuar sobre: selección de los programadores y esfuerzo en la documentación. También te dejan ensanchar el cronograma.

4.9 COCOMO II

Ejercicio 4. Gestión de Riesgos.

Un desarrollo software está estimado en nivel de post-arquitectura en 10 KSLOC. El exponente del tamaño se ha calculado $E=1.1$ y los controladores de coste son nominales. 1PM le cuesta a la empresa 3000€. Se han detectado 2 riesgos:

- a) Es posible que los requisitos evolucionen de forma que el tamaño del producto aumente un 20%.*
- b) Nuestra empresa prevé reducir su personal el año que viene, así hay incertidumbre sobre la continuidad de todos los miembros del equipo.*

Estima las reservas de dinero necesarias para soportar el aumento en el presupuesto si estos riesgos finalmente se cumplen.

4.9 COCOMO II

Ejercicio 5: Desarrollar o Reutilizar

El director de Portfolio indica al PM de un proyecto que parte del código a desarrollar podría reutilizarse de un proyecto pasado. En concreto, es un módulo de gestión de 40 KSLOC.

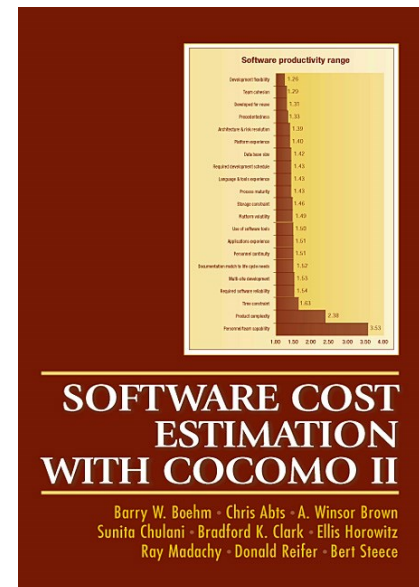
Preguntando a los analistas y programadores de nuestro equipo, descubrimos que hará falta refactorizar y rediseñar parte del código; el cual además está mal documentado, y en el equipo no hay nadie familiarizado con este módulo.

Haremos comité de expertos. Cada alumno debe decidir si merece la pena reutilizar dicho código, dando una estimación de EKSLOC.

COCOMO II

- La portada del libro de COCOMO II se dedica a mostrar la aportación principal según Boehm: los **rangos de productividad**. A partir de los rangos de valores que pueden tomar los factores de escalado (SF) exponencial y los multiplicadores del esfuerzo (EM), se obtienen los siguientes rangos de en qué rango porcentual pueden afectar cada uno de estos parámetros al esfuerzo:

EM o SF	Productivity Range
Development flexibility	1.26
Team cohesion	1.29
Developed for reuse	1.31
Precedentedness	1.33
Architecture and risk resolution	1.39
Platform experience	1.4
Data base size	1.42
Required development schedule	1.43
Language and tools experience	1.43
Process maturity	1.43
Storage constraint	1.46
Platform volatility	1.49
Use of software tools	1.5
Applications experience	1.51
Personnel continuity	1.51
Documentation match to life cycle needs	1.52
Multisite development	1.53
Required software reliability	1.54
Time constraint	1.63
Product complexity	2.38
Personnel/team capability	3.53



$$PR_{SFi} = \frac{100^{0.91 + (0.01 \times SFi_{max})}}{100^{0.91}}$$

$$PR_{EMi} = \frac{EMi_{max}}{EMi_{min}}$$

$$PR_{personnel \text{ and } team \text{ capability}} = \frac{ACAP_{max} \times PCAP_{max}}{ACAP_{min} \times PCAP_{min}}$$

Estimación del Coste

4.9 COCOMO II

- El CSSE (Center for Systems and Software Engineering) es un supergrupo de los autores de COCOMO II, encargados de mantenerlo y desarrollar extensiones de éste. Además de proporcionar software gratuito.

<http://csse.usc.edu/>

- Algunas de las extensiones son:
 - **COCOTS**: complemento de COCOMO II para medir con más detalle el esfuerzo, costo y cronograma de integrar COTS en nuestro desarrollo software.
 - **COPSEMO**: modelo para medir más acertadamente la distribución del esfuerzo y el coste en el cronograma según las fases de proyectos pequeños, para los cuales las cifras de COCOMO II están mal ajustados.
 - **Expert COCOMO**: extiende COCOMO II ayudando a identificar, cuantificar y priorizar riesgos, cambiando el valor de los controladores de coste según cada escenario considerado.

- 4.1 Introducción**
- 4.2 Ley de Parkinson**
- 4.3 Price-to-Win**
- 4.4 (Wideband) Delphi**
- 4.5 Planning Poker**
- 4.6 CBR**
- 4.7 Clasificación automática**
- 4.8 COCOMO 81**
- 4.9 COCOMO II**
- 4.10 Coste del Producto**

Pablo.Bermejo@uclm.es

4.10 Coste del Producto

- Hemos visto que **Tamaño** → **Esfuerzo** → **Tiempo** → **Costos** siendo los costos (coste monetario) del personal lo que más incrementa el Coste del Proyecto, al cual hay que usar infraestructuras, dietas,...
- Si el producto se ha desarrollado para su venta, podríamos pensar que $\text{Coste del Producto} = \text{Coste del Proyecto} + \text{Beneficios Que Queremos Obtener}$
Sin embargo, hay **otros factores** que pueden aumentar o disminuir el **Coste del Producto**:
 - *Oportunidad de Mercado*: bajar el beneficio deseado porque somos nuevos en el mercado.
 - *Términos contractuales*: bajamos el precio a cambio de retener los derechos sobre el código fuente para su reutilización.
 - *Volatilidad de los requisitos*: si aparecen nuevos requisitos durante el desarrollo, su inclusión es muy costosa y eso se ve influenciado en el Coste del Proyecto.
 - *Estado financiero*: si el desarrollador necesita dinero, puede bajar sus beneficios o Coste del Proyecto (sueldo del equipo) para ganar el contrato.
 - *Estrategia*: podemos indicar un coste que nos haga perder dinero ($C. \text{Producto} < C. \text{Proyecto}$), pero el cliente nos asegura que a cambio nos encargará un producto grande.

Conclusiones

- Estimar el Coste del Proyecto es estimar el Esfuerzo y el Tiempo necesario.
- El factor que más influye en el Coste es el Tamaño del Software.
- Existen varios modelos de estimación del Coste, lo ideal es utilizar al menos 2 modelos y comparar las diferencias.
- COCOMO II es el método más complejo debido a su gran cantidad de factores:
 - cuya valoración es subjetiva y rodeada de incertidumbre.
 - pero gracias a la identificación de los controladores de coste, podemos realizar análisis de varios escenarios para la toma de decisiones:
 - Gestión de Riesgos
 - Ajuste del coste monetario.
 - Retorno de la inversión
 - Desarrollar vs Reutilizar
- Las metodologías de desarrollo ágil prefieren la estimación semanal, o por Sprints, de las historias de usuario a desarrollar en ese espacio corto de tiempo. Y la estimación global previa se suele hacer por comisión de Expertos.

¿Te acuerdas de...?

- 1) Trabajas en una empresa orientada a proyectos. ¿Qué harías/dirías si estimas que tu proyecto es corto pero el director de Programa te da 2 años?
- 2) Para la estimación del coste de un proyecto, te dan una base de datos histórica de la empresa con 300 proyectos, cada uno descrito con 100 variables. ¿Qué metodología de estimación del coste piensas que es más oportuna?
- 3) ¿Qué consecuencia tiene que el Tamaño esté exponenciado en la estimación algorítmica del esfuerzo? ¿Y que los controladores de coste multipliquen?
- 4) ¿Qué controlador del coste en COCOMO II tiene potencialmente una mayor influencia a la hora de incrementar o disminuir el coste? ¿Qué te indica esto?