C O C O M O v 2

Modelo de Estimación de Costes para proyectos software

PROFESOR: Francisco Ruiz González
ASIGNATURA: Planificación y Gestión de Sistemas de Información
MAYO 1999

ANTONIO DE LA FUENTE MOYA
Cuarto Curso
Escuela Superior de Informática
Universidad de Castilla-La Mancha
Campus de Ciudad Real

Notas del Profesor sobre este trabajo

- El apéndice B (Calibración del modelo post-arquitectura no está incluida en este documento.
- Algunas fórmulas tienen coeficientes que no coinciden con los utilizados en la actualidad para el modelo COCOMO.

Consultar la página siguiente para información actualizada sobre los dos aspectos anteriores:
http://sunset.usc.edu/research/COCOMOII/cocomo_main.html

ÍNDICE

Página

ANOTACIONES

1. Se pretende dar una introducción suficiente, pero no abusiva (aunque esta consideración siempre resulte subjetiva), para situar a quien lea este trabajo dentro del significado y del mundo donde se utiliza el modelo COCOMO, recordándole algunos conceptos y dejándolo al final de la introducción en condiciones suficientes para comprender lo que significa y la utilidad de COCOMO.

2. Se ha perseguido que después de leer este documento, se comprenda el modelo COCOMO 2.0, y se tenga la suficiente información para poder utilizar el software que acompaña a este documento, y que no es sino una herramienta desarrollada por la Universidad del Sur de California, para aplicar este modelo.

3. Más documentación, sobre manuales, métodos específicos de calibración del modelo, etc podrá encontrarse en las referencias bibliográficas, al final de este documento.

SITUACIÓN

COCOMO (COnstructive COnst MOdel) es una herramienta utilizada para la estimación de algunos parámetros (costes en personas, tiempo, ...) en el diseño y construcción de programas y de la documentación asociada requerida para desarrollarlos, operarlos y mantenerlos (Boehm), es decir, en la aplicación práctica de la Ingeniería del Software.

Este desarrollo de software, y ante los problemas que se encuentran en él, hizo que desde la década de los 70 creciera un interés en el estudio de los problemas que lleva consigo el software, surgiendo conceptos como control de calidad (SQA), metodologías de análisis y diseño, ingeniería del software, etc...

¿Cuáles son algunos de estos problemas que se encuentran en el desarrollo de aplicaciones?:

- Incremento de la complejidad de los sistemas (mecanizar áreas más amplias y complejas)

- Los proyectos nunca terminan en el plazo previsto. Las estimaciones se realizan en un momento en el que no se tiene suficiente información o ésta no se usa correctamente para poder determinar plazos.

- Los problemas más costosos se producen en las fases más tempranas del desarrollo del software.

- Los costes de mantenimiento actualmente son muy elevados debido a la continua evolución de los sistemas y a los errores en su concepción.

- Distinto vocabulario incluso dentro de los integrantes del propio equipo de desarrollo informático.

- Cada diseñador utiliza una "forma de hacer" diferente para definir un sistema de información.

- Riesgo inherente a que la definición de los sistemas la realicen técnicos provisionales.

- Escasa implicación del usuario en las tareas de desarrollo de los nuevos sistemas.

INTRODUCCIÓN A LAS TÉCNICAS DE ESTIMACIÓN DE COSTES

Debido a lo que se conoce como "crisis del software", con problemas como los anteriormente descritos, comienza una preocupación por controlar los costes de desarrollo, así como la distorsión de este desarrollo respecto a la planificación establecida.

"*Para llevar a cabo un buen proyecto de desarrollo de software, debemos comprender el ámbito del trabajo a realizar, los recursos requeridos, las tareas a ejecutar, las referencias a tener en cuenta, el esfuerzo (COSTE) a emplear y la agenda a seguir*." (R. Pressman)

Para intentar dar solución a estos problemas se han introducido en la Ingeniería del Software una serie de técnicas, utilizadas dentro de las tareas de planificación, que ayudan a planificar y controlar el esfuerzo y el tiempo necesario de desarrollo:

- Técnicas de estimación del esfuerzo (coste) de desarrollo. Dentro de las cuales se sitúa COCOMO.

- Técnicas de planificación y seguimiento de proyectos.

El problema de realizar estimaciones es que en el instante en que se requiere dicha estimación no se tiene suficiente información para que ésta tenga la exactitud requerida.

No obstante, el desarrollo de software presenta un comportamiento característico que puede ser analizado y empleado para planificar adecuadamente su desarrollo dentro de unos límites de tiempo y coste razonables. Se necesita por tanto conocer:

- cómo se comportan los trabajos de desarrollo.

- qué factores pueden ser controlados.

- cuáles de éstos son determinantes para el proceso de desarrollo de un proyecto.

Existen algunos trabajos que intentan desde un estudio estadístico de una muestra, elegida de manera representativa, de un conjunto de casos reales, establecer modelos básicamente empíricos, que ayuden a realizar dichas estimaciones con un mayor grado de fiabilidad.

Tradicionalmente existían dos premisas básicas:

- los recursos humanos y el tiempo son variables intercambiables.

- el nivel de productividad, dentro de una organización, es relativamente constante para todos los proyectos.

Hoy en día ambas afirmaciones pueden considerarse erróneas o al menos inexactas, ya que:

- el incremento del número de personas si aumenta su productividad, pero también añade otras tareas como la integración de esas personas dentro del grupo de trabajo que aumentan el global necesario.

- a su vez si el aumento de personas se realiza no al principio del proyecto, sino durante la realización del mismo, hay que añadir el tiempo necesario para poner al día a los nuevos integrantes del equipo.

- por último se ha demostrado que la productividad es, entre otros factores, una función compleja del nivel de dificultad intrínseca de cada proyecto.

Por tanto la relación siguiente solamente es aplicable a proyectos de muy pequeño tamaño.

Esfuerzo **=** Número de líneas de código **/** Productividad

Existen **dos grandes orientaciones de medida del software**:

- **Función**: Tipo de problema que resuelve.
- **Tamaño**: Volumen del software.

Dentro de la primera se sitúa la técnica de los **Puntos de Función**, de A. Albrecht (1979) y de la segunda el modelo **COCOMO** (Constructive Const Model) de Barry Boehm (1981).

La utilización de LDC (líneas de código) o DSI (delivered source instructions) es discutida por algunos autores pues la consideran una medida poco consistente, la cual presenta variaciones difícilmente ponderables:

- longitud
- dificultad
- cantidad de información y
- funcionalidad
- etc.

más aún si se trata de líneas escritas en distintos lenguajes.

Así, algunos investigadores de estos temas han llegado a decir, por ejemplo, que la funcionalidad de una LDC PL1 es casi el doble que la de una COBOL, y que una de un 4GL proporciona entre dos y cuatro veces más funcionalidad que la de un lenguaje de programación convencional.

El modelo **COCOMO,** aún basándose en una estimación de LDC, ha sido ampliamente aceptado por:

- ser un modelo público bien documentado.

- debido a que los datos de entrada que solicita el modelo y sus resultados son mucho más claros y precisos que en otros modelos.

- admite la posibilidad de calibrarse para entornos específicos.

PUNTOS FUNCIÓN.

Realizada por Allan Albercht en 1979 y revisada a continuación en 1983, esta técnica está basada (orientada) en la teoría de la "ciencia del software" desarrollada por Halstead, la cual está orientada al análisis del proceso de construcción de programas y se basa en la medida del número de "unidades sintácticas básicas" (operadores y operandos).

No se fija en el número de LDC sino en su funcionalidad.

La finalidad de la técnica de los puntos función es estimar el tamaño de un producto software y el esfuerzo asociado a su desarrollo, expresado éste en horas trabajadas por punto función, en las etapas previas a su desarrollo.

Los estudios realizados sobre la utilización de este método reflejan la bondad del mismo y la existencia de un elevado grado de correlación entre el número de líneas de código (LDC) y la estimación total de los puntos función.

Etapas del método:

1. Contar las funciones de usuario.
2. Ajustar el modelo en función de la complejidad del proceso.

1. Contar las funciones de usuario. En la etapa primera, se definen cinco tipos de funciones de usuario:

   - Entradas (al slstema): entradas de usuario que proporcionan al sistema diferentes datos orientados a la aplicación.
   - Salidas: salidas de usuario que le proporcionan a éste información sobre la aplicación.
   - Consultas: peticiones de usuario que como resultado obtienen algún tipo de respuesta en forma de salida.
   - Ficheros lógicos internos o archivos maestros: número de archivos lógicos maestros (agrupación lógica de datos).
   - Ficheros o interfaces externos: interfaces legibles (archivos de datos en cinta o disco) utilizados para transmitir información a otros sistemas .

2. Ajustar el modelo en función de la complejidad del proceso. El recuento de las funciones de usuario se realiza tras una clasificación previa de éstas en tres niveles de complejidad:

   - Simple
   - Medio
   - Complejo

Tras esta división de las funciones de usuario según su tipo y complejidad se les aplica un peso, como aparece reflejado en la tabla adjunta, obteniendo el total de los puntos función sin ajustar:

Tipo de función	Nivel de complejidad			Total
	Simple	Medio	Complejo	
Entradas	___ x3= ___	___ x4= ___	___ x6= ___	_____
Salidas	___ x4= ___	___ x5= ___	___ x7= ___	_____
Ficheros lógicos internos	___ x7= ___	___ x10= ___	___ x15= ___	_____
Ficheros externos	___ x5= ___	___ x7= ___	___ x10= ___	_____
Consultas	___ x3= ___	___ x4= ___	___ x6= ___	_____
Total Puntos función sin ajustar CF=_____				

Valoraciones según el nivel de complejidad

La estimación del contador de puntos de función (CF) debe ajustarse valorando la "complejidad del proceso", la cual puede variar dependiendo del entorno de desarrollo y de las características propias de la aplicación.

Esta complejidad puede verse afectada según este método por catorce características, las cuales se evalúan de conformidad a una escala de "grados de influencia" que toma valores enteros comprendidos entre 0 (sin influencia alguna) y 5 (grado de influencia más elevado). Es decir:

Características		GI
C_1	Transmisión de datos	____
C_2	Proceso distribuido	____
C_3	Rendimiento, respuesta	____
C_4	Configuración	____
C_5	Indice de transacciones	____
C_6	Entrada de datos on-line	____
C_7	Eficiencia de usuario	____
C_8	Actualización on-line	____
C_9	Complejidad del proceso	____
C_{10}	Reusabilidad	____
C_{11}	Facilidad de instalación	____
C_{12}	Sencillez en operación	____
C_{13}	Adaptabilidad	____
C_{14}	Flexibilidad	
Total Grados de Influencia GI = ____		

Características de la aplicación

Como resultado obtenemos los puntos de función ajustados.

PF = CF x (0,65 + (0,01 x GI)) PF: Puntos función ajustados
CF: Puntos función sin ajustar GI: Grados de influencia.

Ejemplo:

APLICACIÓN: _____ APL ID: _____

PREPARADO POR: _____ __/__/__ REVISADO POR: _____ __/__/__

NOTAS:

RECUENTO DE FUNCIONES

Tipo de función	Nivel de complejidad			Total
	Simple	Medio	Complejo	
Entradas	. x3= .	. x4= .	. 4 x6= 24 .	. 24 .
Salidas	. x4= .	. 4 x5= 20 .	. x7= .	. 20 .
Ficheros lógicos internos	. x7= .	. x10= .	. 6 x15= 70 .	. 90 .
Ficheros externos	. x5= .	. x7= .	. x10= .	. .
Consultas	. x3= .	. 5 x4= 20 .	. x6= .	. 20 .

Total Puntos función sin ajustar CF = 154

COMPLEJIDAD DEL PROCESO

Características	GI	Características	GI
C_1 Transmisión de datos	. .	C_8 Actualización on-line	. .
C_2 Proceso distribuido	. .	C_9 Complejidad del proceso	. .
C_3 Rendimiento, respuesta	. 4 .	C_{10} Reusabilidad	. .
C_4 Configuración	. .	C_{11} Facilidad de instalación	. .
C_5 Indice de transacciones	. 4 .	C_{12} Sencillez de operación	. .
C_6 Entrada de datos on-line	. .	C_{13} Adaptabilidad	. .
C_7 Eficiencia de usuario	. .	C_{14} Flexibilidad	. 5 .

Total Grados de Influencia GI 13 .

ESCALA DE GRADOS DE INFLUENCIA:

No influye	= 0	Media	= 3
Insignificante	= 1	Significativa	= 4
Moderada	= 2	Fuerte	= 5

FACTOR DE AJUSTE	CP = 0,65 + (0,01) x GI	= 0,78
TOTAL PUNTOS FUNCIÓN	PF = CF x CP	= 120
Nº LINEAS CODIGO	I = [redacted] x PF	= 7.920=8k

esto será para un lenguaje concreto

MODELO COCOMO'81 (Constructive Cost Model, versión de 1981)

Este fue presentado por Barry Boehm en 1981 y se convirtió en el más conocido y referenciado, además del más documentado de todos los modelos de estimación de esfuerzo de las actividades de diseño, codificación, pruebas y mantenimiento.

La versión inicial de COCOMO se obtuvo a partir de la revisión de los modelos de costes existentes, en la cual participaron varios expertos en dirección de proyectos, los cuales poseían además cierta experiencia en la utilización de diferentes modelos de estimación.

Inicialmente se creó un modelo con un único modo de desarrollo, pero posteriormente se vio que la aplicación del modelo a una amplia variedad de entornos implicaba la creación de los tres modos de desarrollo:

- **Orgánico**. Proyectos de no más de 50 KLDC (50.000 LDC), sobre áreas muy específicas y bien conocidas por el equipo participante.
- **Semiempotrado** (semilibre). El nivel de experiencia del equipo de desarrollo se sitúa en niveles intermedios y suelen ser sistemas con interfaces con otros sistemas, siendo su tamaño menor a 300 KLDC.
- **Empotrado** (restringido). Proyectos de gran envergadura, con una exigencia de altos niveles de fiabilidad y en los que participan muchas personas.

Por otra parte Boehm presenta una jerarquía de modelos de estimación según el nivel de detalle empleado en su utilización:

- **Básico**. Modelo que calcula el esfuerzo de desarrollo como función del tamaño estimado del software en LDC. Adecuado para realizar estimaciones de forma rápida, aunque sin gran precisión.
- **Intermedio**. En éste el esfuerzo se calcula como función del tamaño del producto, modificado por la valoración de los atributos directores del coste, los cuales incluyen una valoración subjetiva del producto, del hardware, del personal, etc. Los valores de los diferentes atributos se consideran como términos de impacto agregado al coste total del proyecto.
- **Detallado**. En él la valoración de los atributos tiene en cuenta su influencia en cada una de las fases de desarrollo del proyecto.

Las estimaciones relacionadas con el coste (esfuerzo) se expresan en meses-hombre (tiempo que requeriría una sola persona para desarrollar el sistema), considerando que la dedicación de una persona es de 152 horas al mes.

1. Modelo Básico.

	Orgánico	Semiempotrado	Empotrado
Esfuerzo estimado	$E_D=2,4(KLDC)^{1,05}$ h-m	$E_D=3,0(KLDC)^{1,12}$ h-m	$E_D=3,6(KLDC)^{1,20}$ h-m
Tiempo de desarrollo	$T_D=2,5(E_D)^{0,38}$ m	$T_D=2,5(E_D)^{0,35}$ m	$T_D=2,5(E_D)^{0,32}$ m
Productividad	$PR = LDC / E_D$		
Nº medio de personas	FSP (Full-Time equivalent Software Personel) $P_E = E_D / T_D$ h		
Esfuerzo De Mantenimiento	TCA (Tráfico de cambio anual): porción de instrucciones fuente que sufren algún cambio durante un año, bien sea por adición o por modificación. $E_M = TCA \times E_D$ Y por tanto el valor medio del número de personas a tiempo completo, dedicadas a mantenimiento durante 12 meses sería: $(P_E)_M = E_M / 12$		

h=hombre, m=mes, h-m=hombres-mes

Ejemplo.
Un estudio inicial determina que el tamaño del producto estará alrededor de 32.000 LDC, a partir de las anteriores ecuaciones obtendríamos que las características del proyecto serían:

Por el tamaño del producto a desarrollar vemos que debemos aplicar las ecuaciones asociadas al modo orgánico, obteniendo lo siguiente:

$E_D = 2,4\ (32)^{1,05} = 91$ hombre-mes
$T_D = 2,5\ (91)^{0,38} = 14$ meses
$PR = 32.000 / 91 = 352$ líneas/hombre-mes
$P_E = 91 / 14 = 6,5$ hombres

Tamaño (líneas)	Esfuerzo (hombre-mes)	Productividad (líneas/hombre-mes)	Tiempo (meses)	PE (hombres)
Pequeño 2KS	5,0	400	4,6	1,1
Intermedio 8KS	21,3	376	8,0	2,7
Medio 32KS	91,0	352	14,0	6,5
Grande 128KS	392,0	327	24,0	16,0

Perfiles de proyectos estándares: Modo orgánico.

2. Modelo Intermedio.

Este modelo es una versión ampliada del modelo Básico, en la que se presenta una mayor precisión en las estimaciones, manteniendo prácticamente la misma sencillez del anterior modelo. Esta mayor precisión viene dada por la incorporación de 15 factores que reflejan la influencia de ciertos elementos sobre el coste del software.

El criterio para la elección de estos factores es *generalidad* e *independencia*. Es decir, se eliminaron aquellos factores que solamente eran significativos de un pequeño número de situaciones, así como aquellos otros que mostraban una fuerte correlación con aspectos puntuales del desarrollo del software.

Finalmente estos 15 factores se agruparon en cuatro grandes grupos: Atributos del Producto, del Computador, del Personal y del Proyecto.

Cada uno de estos 15 atributos tiene asociado un factor multiplicador para estimar el efecto de éste sobre el esfuerzo nominal.

	Orgánico	Semiempotrado	Empotrado
Ecuaciones del esfuerzo nominal	$E_N=3,2(KLDC)^{1,05}$	$E_N=3,0(KLDC)^{1,12}$	$E_N=2,8(KLDC)^{1,20}$

Atributos	Valor					
	Muy bajo	Bajo	Nominal	Alto	Muy alto	Extra alto
Atributos del producto						
Fiabilidad	,75	,88	1,00	1,15	1,40	
Tamaño base de datos		,94	1,00	1,08	1,16	
Complejidad	,70	,85	1,00	1,15	1,30	1,65
Atributos del computador						
Restricciones de tiempo de ejecución			1,00	1,11	1,30	1,66
Restricciones de memoria virtual			1,00	1,06	1,21	1,56
Volatilidad de la máquina virtual		,87	1,00	1,15	1,30	
Tiempo de respuesta		,87	1,00	1,07	1,15	
Atributos del personal						
Capacidad de análisis	1,46	1,19	1,00	,86	,71	
Experiencia en la aplicación	1,29	1,13	1,00	,91	,82	
Calidad de los programadores	1,42	1,17	1,00	,86	,70	
Experiencia en la máquina virtual	1,21	1,10	1,00	,90		
Experiencia en el lenguaje	1,14	1,07	1,00	,95		
Atributos del proyecto						
Técnicas actualizadas de programación	1,24	1,10	1,00	,91	,82	
Utilización de herramientas software	1,24	1,10	1,00	,91	,83	
Restricciones de tiempo de desarrollo	1,23	1,08	1,00	1,04	1,10	

Factores multiplicadores del esfuerzo de desarrollo de software

Esfuerzo total: $E_D = E_N * \vartheta f_i$

donde fi se corresponde con los quince factores descritos anteriormente.

Si se observan los valores para el modelo Intermedio, se puede ver que los coeficientes escalares (3'2, 3'0, 2'8) decrecen a medida que se incrementa la complejidad del modo de desarrollo, al contrario que en el modelo Básico.

Esta diferencia indujo a Conte y otros a presentar un conjunto de ecuaciones alternativas, cuyo comportamiento, según sus autores, mejora el de las ecuaciones originales del modelo Intermedio:

	Orgánico	Semiempotrado	Empotrado
Ecuaciones del esfuerzo nominal	$E_N=2,6(KLDC)^{1,08}$	$E_N=2,9(KLDC)^{1,12}$	$E_N=2,9(KLDC)^{1,20}$

3. Modelo Detallado.

Este modelo presenta principalmente dos mejoras respecto al anterior modelo:

+ Los factores correspondientes a los atributos son sensibles a la fase sobre la que se realizan las estimaciones, puesto que aspectos tales como experiencia en la aplicación, utilización de herramientas software, etc, tienen mayor influencia en unas fases que en otras.

+ Establece una jerarquía de tres niveles de productos, de forma que:
  - los aspectos que presentan gran variación a bajo nivel, se consideran a nivel módulo.
  - los que presentan pocas variaciones a nivel de subsistema
  - los restantes se consideran a nivel sistema.

Calibración del modelo.

El modelo puede ajustarse para mejorar la precisión de sus estimaciones, por las características propias de una instalación particular.

Este proceso puede realizarse por diversos procedimientos, no excluyentes:

- calibrar las ecuaciones del esfuerzo nominal de acuerdo al comportamiento del proyectos finalizados.

- consolidar o eliminar algunos de los atributos directores del coste.

- añadir otros atributos que pueden ser más significativos en la instalación en particular.

Veamos cómo podría ser una posible calibración de las ecuaciones propuestas por el modelo Intermedio para su modo orgánico:

1. Calibración de una constante:

$$E_D = \mathbf{c}\,(KLDC)^{1,05} * \vartheta$$

Supongamos que en una instalación se han finalizado n proyectos cuyos tamaños finales fueron $KLDC_1$, ... $KLDC_n$, los factores de ajustes empleados ϑ_1, ... ϑ_n, y el esfuerzo dedicado a cada uno de ellos E_1, ... E_n. En este caso calcular la constante **c** consiste en resolver el sistema de ecuaciones tal que la suma de cuadrados de esas diferencias sea lo más pequeña posible.

$$\begin{aligned} E_1 &= \mathbf{c}\ (KLDC_1)^{1,05}\ \vartheta_1 \\ E_2 &= \mathbf{c}\ (KLDC_2)^{1,05}\ \vartheta_2 \\ &\ldots\ldots\ldots\ldots \\ E_n &= \mathbf{c}\ (KLDC_n)^{1,05}\ \vartheta_n \end{aligned}$$

Así pues, planteamos la suma de cuadrados de las diferencias como:

$$E = \sum_{i=1}^{15} (\mathbf{c}\,(KLDC_i)^{1,05}\,\vartheta_i - E_i)^2$$

Haciendo

$$Q_i = (KLDC_i)^{1,05}\,\vartheta_i$$

y sustituyendo

$$E = \sum_{i=1}^{15} (\mathbf{c}\,Q_i - E_i)^2$$

para minimizar E calculamos su derivada e igualamos a cero:

$$0 = dE/dc = 2 \sum (\mathbf{c}Q_i - E_i)\,Q_i$$

de ahí puede despejarse la constante **c** como:

$$c = \sum E_i\,Q_i \ /\ \sum Q_i^2$$

2. Calibración de dos constantes:

Para realizar esta calibración puede aplicarse la misma técnica de aproximación por mínimos cuadrados, teniendo en cuenta que ahora se desea determinar el término multiplicador **c** y el factor de escala **b** de la ecuación de esfuerzo:

$$E_D = \mathbf{c}\,(KLDC)^{\mathbf{b}} * \vartheta$$

Comencemos reescribiendo la ecuación tomando logaritmos:

$$\ln(\mathbf{c}) + \mathbf{b}\ln(KLDC) = \ln(E_D/\vartheta)$$

En este caso nos interesa minimizar

$$E = 3 (\ln(\mathbf{c}) + \mathbf{b} \ln(KLDC_i) - \ln(E_{i/\vartheta i}))^2$$

Los valores óptimos de ln(**c**) y **b** pueden determinarse resolviendo el siguiente sistema:

$$a_0 \ln(\mathbf{c}) + a_1 \mathbf{b} = d_0$$
$$a_1 \ln(\mathbf{c}) + a_2 \mathbf{b} = d_1$$

donde

a_0 = n (número de proyectos)
a_1 = $3 \ln(KLDC_i)$
a_2 = $3 (\ln(KLDC_i))^2$
d_0 = $3 \ln(E_{i/\vartheta i})$
d_1 = $3 \ln(E_{i/\vartheta i}) \ln(KLDC_i)$

resolviendo obtenemos:

$$\ln(\mathbf{c}) = a_2 d_0 - a_1 d_0 \; / \; a_0 a_2 - a_1^2$$
$$\mathbf{b} = a_0 d_1 - a_1 d_0 \; / \; a_0 a_2 - a_1^2$$

Situación a principios de los 90

Al inicial modelo de estimación de costes COCOMO 81, le siguió una actualización para el lenguaje Ada en 1987, que recibió el nombre de Ada COCOMO. Desde entonces el desarrollo de nuevos ciclos de vida, ocasionados por la evolución del desarrollo del software, ha incrementado la dificultad de estas estimaciones. Y estamos hablando de términos como desarrollo rápido de aplicaciones, aplicaciones no secuenciales, reusabilidad del software, reingeniería, programación orientada a objetos, etc.

COCOMO 2.0

INTRODUCCIÓN .

Actualmente una nueva generación de procesos y productos software está cambiando la forma en que las organizaciones desarrollan software, intentando ser más competitivas. Estas nuevas aproximaciones -procesos software colaborativos, evolucionarios, y centrados en los riesgos; generadores de aplicaciones y lenguajes de cuarta generación; "comercial off-the-shelf" (COTS) y dependientes de la reutilización que se deba hacer del software- llevan a significativos beneficios en términos de calidad software y reducción de coste que supone su desarrollo, reducción de riesgos y del tiempo de ciclo.

De cualquier forma, si bien alguno de los modelos de estimación de coste software existentes poseen iniciativas que tienen en cuenta algunos de estos aspectos, estas nuevas aproximaciones no han sido tenidas en cuenta suficientemente hasta hace pocos años por los nuevos modelos de estimación de costes y planificación. Esto hace que sea difícil a las organizaciones realizar una planificación, análisis y control de proyectos de manera efectiva y utilizando las nuevas aproximaciones.

Estos argumentos han llevado a realizar una nueva formulación del modelo COCOMO, sucesor de los anteriores COCOMO 81 de Boehm (1981) y el COCOMO Ada de Royce (1989).

OBJETIVOS:

Los principales objetivos de COCOMO 2.0 son:

1. El desarrollo de un modelo de estimación de costes y planificación que pudiera ser utilizado en el ciclo de vida de la década de los 90 y posterior al 2000.

2. Desarrollar una base de datos indicativa del coste del software y un soporte de herramientas con la capacidad suficiente para el continuo progreso y perfeccionamiento del modelo.

3. Proporcionar un cuantioso y analítico marco de trabajo, y un conjunto de herramientas y técnicas para la evaluación de los efectos que la tecnología software tiene sobre el coste de los ciclos de vida software y de planificación.

Las **principales capacidades** de COCOMO 2.0 son los ajustes a medida dependiendo del software a desarrollar, involucrando en la estimación del coste a los puntos objeto (object points), puntos función (function points) y líneas de código fuente; utilizando modelizaciones no lineales para atender a la reingeniería y reusabilidad del software, ... y todo esto sobre la base del anterior COCOMO.

Los cuatro elementos principales de la estrategia de COCOMO 2.0 son:

1. Preservar las habilidades y apertura del modelo original, para que continúe siendo un modelo abierto y público (algoritmos, parametrizaciones, etc...)

2. Adaptar la estructura de COCOMO 2.0 a los futuros sectores del mercado software descritos antes.

3. Adaptar las entradas y salidas de los submodelos de COCOMO 2.0 al nivel de información disponible en cada etapa.

4. Permitir a los submodelos de COCOMO 2.0 ajustarse a la medida dependiendo de la estrategia utilizada en cada proyecto particular (atender a las distintas calibraciones de los submodelos que se utilicen para obtener mayor fiabilidad en la estimación global).

Un fundamento importante es el considerar la granularidad del modelo de estimación de coste, teniendo en cuenta la información disponible que sirve de soporte al modelo, entendiendo que en las primeras etapas del proyecto software se conocen muy pocas cosas sobre el tamaño del producto a ser desarrollado, la naturaleza de la plataforma objetivo, la naturaleza del personal involucrado en el proyecto, o los detalles específicos del proceso que se utilizará.

La siguiente figura indica el efecto de las incertidumbres del proyecto tienen sobre la precisión del tamaño del software y la estimación del coste. En los primeros estados, al comienzo, no puede conocerse la naturaleza específica del producto a desarrollar con una aproximación mayor de un factor 4. Según el ciclo de vida avanza, y se realizan decisiones sobre el producto, la naturaleza del producto y su consecuente tamaño son mejor conocidos, y la naturaleza del proceso y sus consecuentes parámetros de coste son mejor conocidos:

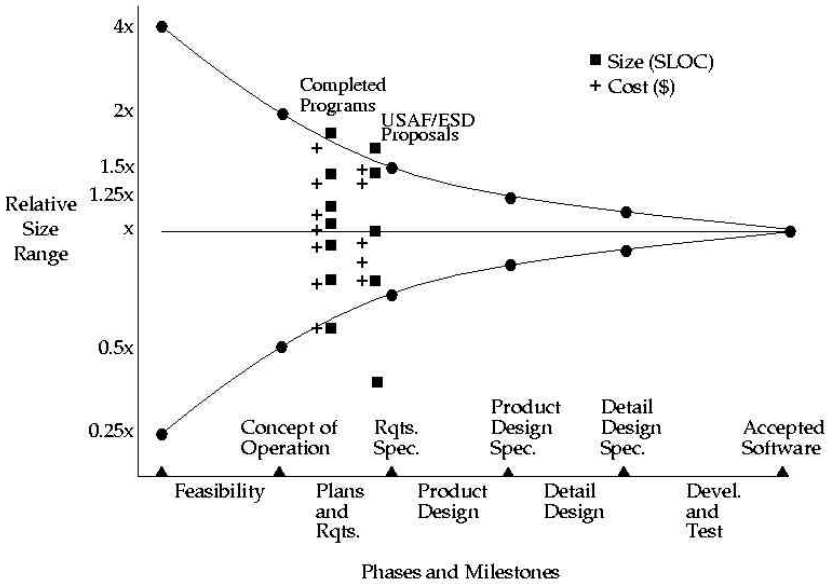

Figure 2. Software Costing and Sizing Accuracy vs. Phase

Figura 2. Precisión de tamaño y coste software dependiendo de la fase en que se encuentre el proceso.

COCOMO 2.0 permite a los proyectos suministrar a los parámetros de coste una información rudamente granulada en los primeros estados del proyecto, para ir incrementandola más detalladamente granulada según se avanza en los estados.

Resumiendo, el modelo COCOMO 2.0 proporciona los siguientes 3 estados (series de modelos) para la estimación de Generadores de Aplicaciones, Integración de Sistemas, e infraestructura de proyectos software:

1. En las primeras fases o ciclos espirales que generalmente incluirán prototipado, se hacen uso de las capacidades del **Modelo de Composición de Aplicaciones**. Este modelo soporta estas fases, y cualquiera otras actividades de prototipado que suceden con posterioridad en el ciclo de vida.

   Incluye pues tareas de prototipado para resolver cuestiones como el diseño de los inferfaces de usuario, la interacción del software con el sistema, el rendimiento o la madured de la tecnología empleada. Asi, los costes de este tipo de esfuerzo son mejor estimados por un modelo de composición de aplicaciones..

   Nos encontramos pues en una situación donde existen aplicaciones muy diversas, tanto que no se manejan como paquetes integrados de soluciones, pero que son lo suficientemente simples para ser rápidamente acopladas y de interoperar sus componentes.

   Como veremos posteriormente, el modelo COCOMO se basa en Puntos Objeto para modelizar la Composición de Aplicaciones, y es una cuenta de las pantallas, informes y módulos desarrollados con lenguajes de tercera generación, cada uno con un peso según la complejidad del parámetro (mínimo, medio, alto). Esto mantiene una equivalencia con el nivel de información de la que generalmente se dispone, sobre esta composición de aplicaciones, durante sus estados de planificación temporal, y el correspondiente nivel de precisión necesaria para estimar el coste software.

2. Las siguientes fases o ciclos espirales que generalmente incluyen la exploración de arquitecturas alternativas o estrategias de desarrollo incrementales. Para soportar estas actividades, COCOMO 2.0 proporciona un temprano modelo llamado **Modelo de Diseño Inicial**. Este nivel de detalle en este modelo es consistente con el nivel general de información disponible y con el nivel general de estimación detallada que es necesaria en esta etapa.

   Se utilizan, como veremos, puntos función y/o instrucciones fuente, y un pequeño número de parámetros de coste (cost drivers).

3. Una vez que el proyecto esta listo para ser desarrollado se debería tener una arquitectura de ciclo de vida, la cual proporcionase más información detallada sobre las entradas de los parámetros de coste, y permitieses mayor precisión en la estimación del coste. Para soportar esta etapa, COCOMO 2.0 proporciona el **Modelo Post-Arquitectura**.

   Incluye este modelo el desarrollo y mantenimiento último del producto software. Se utilizan, como veremos, instrucciones fuente y/o puntos función para la estimación, existiendo modificadores que los operan; un conjunto de 17 factores multiplicativos de evaluación de coste, y un total de 5 modos para dimensionar el proyecto, que sustituyen a los anteriores modos Orgánico, Semiempotrado y Empotrado del COCOMO original.

Un análisis de los datos debería permitir también la calibración de las relaciones entre puntos objeto, puntos función, y líneas de código fuente para varios lenguajes y composición de sistemas, permitiendo mayor flexibilidad al elegir los parámetros que influyen en la clasificación del tamaño.

ESTIMACIÓN DEL ESFUERZO DE DESARROLLO.

COCOMO 2.0 expresa el esfuerzo en terminos de Personas Mes (PM). Todas las ecuaciones del esfuerzo están representadas en el apéndice A. Una persona mes es la cantidad de tiempo que una persona dedica a trabajar sobre el proyecto de desarrollo software durante un mes. El número de personas mes es diferente del tiempo que tomará el proyecto para ser completado; a esto se le llama planificación de desarrollo. Por ejemplo, un proyecto puede estimarse en requerir 50 PM de esfuerzo, pero tener una planificación temporal de 11 meses.

Número nominal de Personas Mes.

La siguiente ecuación es la base de los modelos de Diseño Inicial y Post-Arquitectura. Las entradas son el tamaño del desarrollo software, una constante A, y un factor de escala B. El tamaño se dan en miles de líneas de código fuente (KSLOC). Pudiéndose estimar también utilizando Puntos Función Desajustados (UFP), y convertirlos a SLOC dividiendo por mil.

El factor de escala (o exponencial) B, cuenta la relativa economía, positiva o negativa, de la escala encontrada en proyectos software según cambie el tamaño de éste.

La constante A es usada para capturar los efectos multiplicativos sobre el esfuerzo con proyectos que incrementan su tamaño.

$$PM_{nominal} = A \times (Size)^{B} \qquad \textbf{Ecuación 1}$$

Breakage.

El Modelo COCOMO 2.0 utiliza un porcentaje del código "breakage" (BRAK) para ajustar el tamaño efectivo del producto. El término *Breakage* hace referencia a la volatilidad de los requerimientos de un proyecto. Esto es, el porcentaje de código que se desecha. Por ejemplo, un proyecto formado finalmente por 100.000 instrucciones de las que se han descartado otras 20.000 instrucciones adicionales, entonces BRAK tendrá un valor de 20. Esto debería usarse para ajustar el tamaño efectivo del proyecto a 120.000 instrucciones. Ese factor BRAK no se utiliza en el modelo de Composición de Aplicaciones, donde se espera un cierto grado de interacción del producto, incluida una calibración de los datos.

Ajustando la reusabilidad.

Efectos no lineales: Selby realiza un análisis sobre cerca de 3000 módulos reutilizables, en el Laboratorio de Ingeniería Sofware de la NASA, de cómo influye la

reutilización del código, y como resultado indica que este coste queda expresado por una función no lineal, debido a dos razones principales:

- No se comienza desde el principio. Existe un coste alrededor del 5% debido a la valoración, selección y asimilación que debe hacerse del componente reutilizable.

- Pequeñas modificaciones producen desproporcionadas reacciones en el coste. Esto es debido principalmente a dos factores: el coste de comprender el software que va a ser modificado, y el relativo coste asociado a testear el interface.

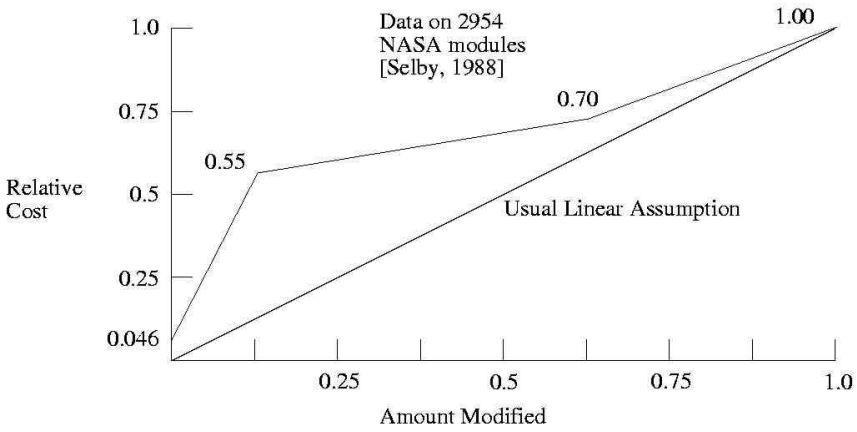

Figure 3. Nonlinear Reuse Effects

Estudios realizados determinan que el 47% del esfuerzo de mantenimiento del software está directamente relacionado con la comprensión del software que se modifica. También se demuestra que si se modifican k de m módulos, el número N que indica los interfaces de módulos que son chequeadas es *N = k * (m-k) + k *x (k-1) / 2.* Relación que se muestra la Figura 4, donde la curva demuestra la mencionada relación no lineal.

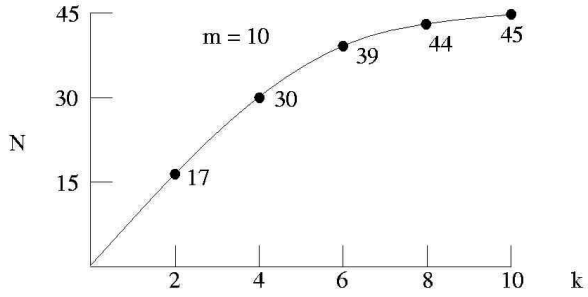

Figure 4. Number of Module Interface Checks vs. Fraction Modified

Siempre teniendo en cuenta que el tamaño que supone este tiempo de comprensión del software y de chequeo del interface puede ser reducido mediante una buena estructura sofware.

Un modelo para tener en cuenta la reusabilidad: el modelo COCOMO 2.0 trata esta reutilización del software usando un modelo de estimación no lineal. Esto incluye una estimación de la cantidad de software a ser adaptado, ASLOC, y tres parámetros según el grado de modificación: porcentaje de diseño modificado (DM), porcentaje de código modificado (CM), y porcentaje del esfuerzo original que supone integrar el software reutilizable (IM).

El incremento que supone la comprensión del software (SU) se obtiene de la siquiente tabla; y expresado cuantitativamente como un porcentaje.

	Muy Bajo	Bajo	Nominal	Alto	Muy Alto
Estructura	Muy baja cohesión, alto acoplamiento, definición poco clara del código.	Moderada baja cohesión, alto acoplamiento.	Razonablemente bien estructurado; algunas áreas resultan débiles.	Alta cohesión, bajo acoplamiento.	Enormemente modular, Información ocultada mediante estrucuras de datos y control.
Claridad de la Aplicación	No existe una clara identidad entre las visiones del programa y de la aplicación.	Existe alguna correlación entre el programa y la aplicación.	Moderada correlación entre el programa y la aplicación.	Buena correlación entre ambos.	Existe una clara relación entre las visiones globales de ambos (programa y aplicación).
Descripción implícita	Código obtuso; la documentación o no existe o resulta oscura u obsoleta.	Existen algunos comentarios en el código y alguna documentación útil.	Un moderado nivel de comentarios en el código, y documentación.	Bien comentado el código, documentación útil, aunque débilmente en algunas áreas.	Código autodescriptivo, documentación actualizada, bien organizada y con un diseño racional.
Incremento de SU en ESLOC	50	40	30	20	10

Tabla 1: Escala según se Incrementa la Compresión del Software (SU).

El otro incremento no lineal sobre la reusabilidad tiene que ver con el grado de Valoración y Asimilación (AA) necesarias para determinar si un cierto módulo software completamente reutilizable es apropiado para la aplicación, e integrar su descripción dentro de la descripción global del producto. La Tabla 2 siguiente proporciona una escala para este porcentaje:

Incremento de AA	Nivel de esfuerzo AA
0	Ninguno.
2	Una básica búsqueda modular y de documentación.
4	Alguna Evaluación y Chequeo modular, documentación.
6	Una considerable Evaluación y Chequeo modular, documentación.
8	Una gran Evaluación y Chequeo modular, documentación.

Tabla 2: Escala según el incremento de Valoración y Asimilación (AA).

Finalmente la ecuación usada para determinar esta reusabilidad es:

$$ESLOC = ASLOC \times \frac{(AA + SU + 0.4 \times DM + 0.3 \times CM + 0.3 \times IM)}{100}$$ **Ecuación 2**

Ajustando la Conversión o Reingeniería.

El modelo COCOMO 2.0 necesita de un refinamiento adicional para estimar el coste de la reingeniería software y conversión. La mayor diferencia viene por la eficiencia de las herramientas automatizadas para la reestructuración del software, que permiten que a un alto porcentaje de código modificado le corresponda un pequeño esfuerzo.

Para realizar esta estimación se incluye un parámetro adicional, AT, que indica el porcentaje de código que es tratado en esta reingeniería mediante translación automática:

$$PM_{nominal} = A \times (Size)^{B} + \left[\frac{ASLOC\left(\frac{AT}{100}\right)}{ATPROD} \right]$$

Ecuación 3

Ajustando Personas Mes.

Los parámetros de coste (cost drivers) son utilizados para capturar características del desarrollo del software que afectan al esfuerzo para completar el proyecto. Estos parámetros de coeste expresan el impacto del parámetro sobre el esfuerzo de desarrollo, en Personas Mes (PM). Este rango va de Muy Bajo a Extra Alto, y tiene un peso asociado según el rango al que pertenezca cada parámetro. Este peso es llamado multiplicador de esfuerzo (EM), siendo la media asignada a cada uno de 1'0 (rango medio o nominal). Este valor será mayor que 1'0 si influye incrementando el esfuerzo, y menor que 1'0 si lo disminuye.

Existen 7 de estos multiplicadores de esfuerzo en el modelo de Diseño Inicial, y 17 en el modelo Post-Arquitectura.

$$PM_{adjusted} = PM_{nominal} \times \left(\prod_{i} EM_{i} \right)$$

Ecuación 4

ESTIMACIÓN DE LA PLANIFICACIÓN TEMPORAL DE DESARROLLO.

La ecuación base inicial para mostrar la planificación para los tres estados de COCOMO 2.0 es:

$$TDEV = \left[3.0 \times (\overline{PM})^{(0.33 + 0.2 \times (B - 1.01))} \right] \times \frac{SCED\%}{100}$$

Ecuación 5

donde TDEV es un calendario temporal expresado en meses, utilizado para determinar que los requerimientos del producto se han completado y que quedan certificados. PM "negado" indica la estimación Personas-Mes excluyendo el multiplicador de esfuerzo SCED. B es la suma de los factores exponentes de escala.

Rangos de salida.

Los tres estados del modelo COCOMO 2.0 permiten que la estimación sea expresada dentro de un rango de valores, que teniendo en cuenta la Figura 2 de "Precisión de tamaño y coste de software según la fase en que se encuentra el proyecto" expresa el resultado E (esfuerzo estimado) como un conjunto de estimaciones pesimitas y optimistas.

Estado	Estimación Optimista	Estimación Pesimista
1	050 E	2'00 E
2	0'67 E	1'50 E
3	0'80 E	1'25 E

Tabla 4.Rangos de salida.

ESCALA DE PRODUCTIVIDAD DEL SOFTWARE.

Los modelos de estimación de coste software a menudo poseen un factor exponencial que cuenta lo favorable o desfavorable que es económicamente un proyecto dependiendo de cómo varie su tamaño. El exponente B de la Ecuación 1 se utiliza para capturar estos efectos.

Si B < 1'0 el proyecto muestra una escala positiva, económicamente favorable, de tal forma que si el tamaño del producto se duplicase, el esfuerzo resultante seria menor del doble. Es decir, la productividad del proyecto se incrementa conforme el tamaño del producto aumenta.

Si B = 1'0, el proyecto está economicamente equilibrado. Este modelo lineal es usado frecuentemente para realizar estimaciones en pequeños proyectos. Usado en el modelo de Composición de Aplicaciones.

Si B > 1'0 el proyecto muestra una escala negativa, economicamente desfavorable, debido generalmente a dos razones principales: crecimiento de las comunicaciones interpersonales que lo desbordan, y crecimiento de la integración de sistemas grandes, que igualmente desborda. En el sentido de que proyectos grandes necesitarán de más personal, y a más personal mayor comunicación que se requiere, no olvidando que incluso aunque un proyecto grande se organiza en pequeños subproyectos que necesitan un esfuerzo menor, se requiere un esfuerzo adicional para diseñar, mantener, integrar y testear todos estos productos por separado y luego globalmente.

Ya el análisis de datos del COCOMO original indicaba que los proyectos mostraban una escala desfavorable económicamente, así existian ya tres clases de modelos de desarrollo software (Orgánico, Semiempotrado, y Empotrado), cada uno con un exponente distinto (B=1'05, B=1'12, B=1'20 respectivamente).

La distinción entre los factores de estos modelos básicamente concierne al entorno: proyectos en modo empotrado tuvieron menos precedentes, requiriendo mayor saturación de comunicaciones así como una integración compleja, son menos flexibles, ...

ESCALANDO LOS PARÁMETROS.

La Ecuación 6 define el cálculo de este exponente B, utilizado ya en la Ecuación 1. La Tabla 19 proporciona una valoración por niveles para estos parámetros de escala del COCOMO 2.0. Cada uno de estos parámetros se divide en un rango según el nivel, que va desde Muy Bajo a Extra Alto. Cada uno de estos rangos tiene un peso (W). Los factores de escala de un proyecto se suman para determinar el exponente de escala B, mediante la siguiente fórmula

$$B = 1'01 + 0'01 * \Sigma W_i \qquad \textbf{Ecuación 6}$$

Por ejemplo si a la escala Extra Alta le asignamos el peso cero, entonces un proyecto con 100KSLOC con todos sus factores de escala en el rango "Extra Alto" tendrán W_i=0, y B=1'01, y un esfuerzo relativo $E=100^{1'01}$ =105 PM. Si por el contrario los factores, todos, están en el rango "Muy Bajo", asignamos un peso de 5 a cada uno, teniendo W_i=25 y B=1'26, y un esfuerzo relativo E=331 PM. Aunque esto representa

una gran variación, el incremento provocado por un cambio en un único parámetro representa tan sólo el 4'7%.

Factor de Escala (W_i)	Muy Bajo (5)	Bajo (4)	Nominal (3)	Alto (3)	Muy Alto (1)	Extra Alto (0)
PREC	Mínima	Poca	Algo	Medianamente familiar	Muy familiar	Altamente familiar
FLEX	Poca o ninguna (muy riguroso)	Ocasional	Alguna	alta	Muy alta	Tendiendo a optima (metas generales)
RESL[a]	Reducción del riesgo entorno al 20%	Idem 40%	Idem 60%	Idem 75%	Idem 90%	Idem 100%
TEAM	Interacciones muy dificiles	Algunas interacciones dificiles	Interacciones para una cooperación básica	Muy cooperativo	Altamente cooperativo	Cooperación óptima
PMAT	peso medio de las respuestas afirmativas en el cuestionario Madured CMM					

Tabla 5. Escala de Factores para los modelos de Diseño Inicial y Post-Arquitectura.

Precedentes (PREC) y Flexibilidad de Desarrollo (FLEX).

Estos dos escalas de factores capturan en gran medida las diferencias entre los modos, del COCOMO original, Orgánico, Semiempotrado y Empotrado. La Tabla 6 reorganiza las características de un proyecto en estos términos. Esta tabla puede ser usada para más de una profundidad de exploración para los factores PREC y FLEX dados por la Tabla 19.

Característica	Muy Baja	Nominal / Alta	Extra Alta
Precedencias			
Objetivos del producto y comprensión organizacional	General	Considerable	Profundo
Experiencia trabajando con sistemas software relacionados	Moderada	Considerable	Extensiva
Desarrollo concurrente asociado a nuevo hardware y nuevos procedimientos operacionales	Extensivo	Moderado	Alguno
Necesidad para la innovación en arquitecturas de proceso de datos , algoritmos	Considerable	Alguno	Minimo
Flexibilidad de desarrollo			
Necesidad de que el sofware se ajuste a los requerimientos preestablecidos	Completo	Considerable	Básico
Necesidad de que el sofware se ajuste a las especificaciones de interface externos	Completo	Considerable	Básico
Prima para un desarrollo completo inicial	Alto	Medio	Bajo

Tabla 6: escala de Factores relacionados con los modos de desarrollo de COCOMO

Resolución de Arquitectura/Riesgos (RESL).

Este factor combina a dos de los factores de escala del modelo Ada COCOMO: "Diseo minucioso mediante examen del Diseño del Producto (PDR)", y "Eliminación de riesgos mediante PDR". La Tabla 7 consolida estos factores del modelo Ada COCOMO para formar una definición más comprensiva de los niveles de valores RESL en COCOMO 2.0. Esta valoración del RESL es una medida de los pesos subjetiva.

Cohesión del Equipo de Trabajo (TEAM).

El factor de escala de Cohesión del Equipo cuenta las fuentes que originan turbulencias y entropia en el proyecto debido a las dificultades de sincronización: usuarios, clientes, desarrolladores, personal encargado del mantenimiento, interfaces, otros. Estas diferencias pueden deberse a diferencias culturales, dificultades para reconocer objetivos, hasta familiaridad para trabajar en equipo. La Tabla 8 proporciona una definición detallada para el conjunto completo de niveles TEAM. La valoración final es la media subjetiva de los pesos.

Madured del Proceso (PMAT)

El procedimiento para determinar PMAT se organiza mediante 18 claves e área de proceso (KPAs) por el Modelo de Capacidad de Madured, del SEI. Este procedimiento que determina PMAT decide el porcentaje de conformidad para cada uno de los KPAs.

	Key Process Areas	Almost Always (>90%)	Frequently (60-90%)	About Half (40-60%)	Occasionally (10-40%)	Rarely If Ever (<10%)	Does Not Apply	Don't Know
1	Requirements Management	❒	❒	❒	❒	❒	❒	❒
2	Software Project Planning	❒	❒	❒	❒	❒	❒	❒
3	Software Project Tracking and Oversight	❒	❒	❒	❒	❒	❒	❒
4	Software Subcontract Management	❒	❒	❒	❒	❒	❒	❒
5	Software Quality Assurance	❒	❒	❒	❒	❒	❒	❒
6	Software Configuration Management	❒	❒	❒	❒	❒	❒	❒
7	Organization Process Focus	❒	❒	❒	❒	❒	❒	❒
8	Organization Process Definition	❒	❒	❒	❒	❒	❒	❒
9	Training Program	❒	❒	❒	❒	❒	❒	❒
10	Integrated Software Management	❒	❒	❒	❒	❒	❒	❒
11	Software Product Engineering	❒	❒	❒	❒	❒	❒	❒
12	Intergroup Coordination	❒	❒	❒	❒	❒	❒	❒
13	Peer Reviews	❒	❒	❒	❒	❒	❒	❒
14	Quantitative Process Management	❒	❒	❒	❒	❒	❒	❒
15	Software Quality Management	❒	❒	❒	❒	❒	❒	❒
16	Defect Prevention	❒	❒	❒	❒	❒	❒	❒
17	Technology Change Management	❒	❒	❒	❒	❒	❒	❒
18	Process Change Management	❒	❒	❒	❒	❒	❒	❒

- Casi siempre: cuando los objetivos son consistentemente alcanzables y bien establecidos en procedimientos operativos estándar.
- Frecuentemente: cuando los objetivos son alcanzables con relativa frecuencia, pero en algunas ocasiones son emitidas bajo circunstancias difíciles (entre el 60% y 90% de las veces).
- Sobre la mitad: cuando los objetivos son alcanzables sobre la mitad de las veces (entre el 405 y 60% de las veces).
- Ocasionalmente: cuando los objetivos son alcanzados algunas veces, pero poco frecuentes (entre el 10% y 40% de las veces).
- Raramente (si acaso): cuando los objetivos raramente son alcanzados (menos del 10% de las veces).
- No se aplica: cuando se tiene el conocimiento requerido sobre el proyecto u organización y el KPA, pero se tiene un sentimiento de que los KPAs circunstancialmente no se pueden aplicar.
- Se desconoce: cuando no se tiene certeza de cómo responderán los KPAs.

Después de que el nivel de KPA es determinado, a cada nivel de conformidad se le asigna un peso o valor y el factor PMAT queda calculado:

$$5 - \left[\sum_{i=1}^{18} \left(\frac{KPA\%_i}{100} \times \frac{5}{18} \right) \right]$$

Ecuación 7

MODELO DE COMPOSICIÓN DE APLICACIÓN.

Este modelo está dirigido a aplicaciones que están demasiado diversificadas como para crear rápidamente una herramienta específica dentro de un dominio. Ejemplos de estos sistemas basados en componentes son los generadores de interfaces de usuario (GUI), gestores de objetos o bases de datos, procesamiento distribuido o de transacciones, manejadores hipermedia, pequeños buscadores de datos, y componentes de un dominio específico tales como domínios financieros, médicos, o paquetes de control de procesos industriales.

Objeto de este estudio son estos sistemas que se caracterizan por llevar asociados esfuerzos de prototipado, uso de ICASE (CASE integrado) para obtener una composición rápida asistida por la computadora, que proporcionan generadores de interfaces gráficos de usuario, herramientas de desarrollo software, y un largo número de componentes de aplicaciones e infraestructuras. En estas áreas se ha demostrado el buen funcionamiento de los Puntos Función para realizar estimaciones sobre un conjunto no trivial de aplicaciones.

Estudios experimentales realizados demuestran que tanto el uso de Puntos Función como de Puntos Objeto dan unas estimaciones muy próximas, si bien el método de los Puntos Objeto resulta más fácil de utilizar.

PROCEDIMIENTO PARA CONTAR PUNTOS OBJETO

Definición de términos:

- NOP: Nuevos Puntos Objeto (cuenta de los Puntos Objeto ajustados para su reutilización).
- srvr: número de servidores de datos (mainframes o equivalentes).
- clnt: número de clientes de datos (estaciones de trabajo personales).
- %reutilización: porcentaje de pantallas, informes y módulos 3GL reutilizados por aplicaciones previas, según su grado de reutilización.

Pasos:

1. Estimar el número de pantallas, informes, y componentes 3GL que comprenderán la aplicación. Asumiendo las definiciones estandar de estos objetos en el entorno ICASE utilizado.

2. Clasificar cada instancia de cada objeto en niveles de complejidad simple, media o alta, dependiendo de los valores según la dimensión. Utilizar el siguiente esquema:

Pantallas				**Informes**			
Número de vistas que contiene	**Total < 4 (<2 srvr <3 clnt)**	**Total < 8 (2-3 srvr 3-5 clnt)**	**Total 8+ (>3srvr >5 clnt)**	**Número de secciones que contiene**	**Total < 4 (<2 srvr <3 clnt)**	**Total < 8 (2-3 srvr 3-5 clnt)**	**Total 8+ (>3 srvr >5 clnt)**
<3	Simple	Simple	Medio	0 o 1	Simple	Simple	Medio
3-7	Simple	Medio	Alto	2 o 3	Simple	Medio	Alto
>8	Medio	Alto	Alto	4+	Medio	Alto	Alto

3. Valorar con un peso (número) como los mostrados en la siguiente tabla, ue reflejan el esfuerzo relativo requerido para implementar una instancia de ese objeto dependiendo del nivel de complejidad:

Tipo de Objeto	Complejidad		
	Simple	**Media**	**Alta**
Pantalla	1	2	3
Informe	2	5	8
Componente 3GL			10

4. Sumar todos los pesos de los objetos instanciados para obtener un único número, que será la cuenta de Puntos Objeto.

5. Estimar el porcentaje de reutilización esperado para realizar el proyecto. Calcular los nuevos Puntos Objeto a ser desarrollados.

$$NOP = \frac{(Object\ Points) \times (100 - \%Reuse)}{100}$$

Ecuación 8

6. Determinar un ratio de productividad PROD=NOP/Personas-mes utilizando el siguiente esquema:

Capacidad y experiencia de los desarrolladores	Muy Baja	Baja	Nominal	Alta	Muy Alta
Capacida y madured de ICASE	Muy Baja	Baja	Nominal	Alta	Muy Alta
PROD	4	7	13	25	50

7. Calcular la estimación de personas-mes:

$$PM = \frac{NOP}{PROD}$$

Ecuación 9

MODELO DE DISEÑO INICIAL (ANTICIPADO).

Se utilizarán Puntos Función Desajustados como medida de tamaño. Este modelo es usado en los primeros estados de un proyecto software, cuando pocas cosas podemos conocer sobre el tamaño del producto a desarrollar, la naturaleza de la plataforma objetivo, la naturaleza del personal involucrado en el proyecto, o los detalles específicos del proceso que se utilizará. Este modelo podría ser empleado en Generadores de Aplicaciones, Integración de Sistemas, o sectores de desarrollo de infraestructuras.

CUENTA DE PUNTOS FUNCIÓN

La aproximación de la estimación del coste mediante Puntos Función está basada en la cantidad de funcionalidades de un proyecto software y en un conjunto de factores individuales del proyecto. Los Puntos Función son estimaciones valiosas ya que están basadas en la información que está disponible al inicio del ciclo de vida del proyecto.

Los Puntos Función miden un proyecto software cuantificando la información asociada a los principales datos externos o control de entrada, salida, o tipos de ficheros.

Pueden identificarse cinco tipos de funciones de usuario según la Tabla 9:

Entrada Externa (Entradas)	Cuenta cada dato de usuario o tipo de entrada de control del usuario que (1) se introduce desde el exterior del sistema y (2) que añade o modifica datos en un fichero lógico interno.
Salida Externa (Salidas)	Cuenta cada dato de usuario o tipo de entrada de control que abandona el sistema hacia el exterior.
Ficheros Lógicos Internos (Ficheros)	Ficheros pasados o compartidos entre sistemas software que deberían contarse como tipos de ficheros de interface externos que están dentro del sistema.
Consultas Externas (Informes)	Cuenta cada combinación de entrada-salida única, donde una entrada causa y genera una salida inmediata, como un tipo de consulta externa.

Tabla 9: Tipos de Funciones de Usuario.

Cada instancia de estos tipos de funciones es clasificado según su nivel de complejidad. Los niveles de complejidad determinan un conjunto de pesos o valores, los cuales son aplicados a su correspondiente cuenta de tipo de función para determinar la cantidad de Puntos Función Desajustados. Esta es la función de medida del tamaño empleada por COCOMO 2.0. El procedimiento usual de Puntos Función incluye una valoración del grado de influencia (DI) de catorce características de la aplicación del proyecto software de acuerdo a una escala, que va desde 0'0 a 0'05 para cada característica. Los 14 ratios son sumados juntos, y añadidos a un nivel base de 0'65 para producir un factor de ajuste de las características generales con un rango que va desde 0'65 hasta 1'35.

Cada una de estas 14 características, como son las funciones distribuidas, rendimiento, y reusabilidad, contribuyen hasta un máximo del 5% sobre el esfuerzo estimado. Esto resulta inconsistente para la experiencia de COCOMO; es por ello que COCOMO 2.0 utiliza Puntos Función Desajustados para realizar una medida del tamaño, y aplica factores de reutilización, parámetros de coste multiplicativos del esfuerzo, y factores de escala que son exponenciales.

PROCEDIMIENTO DE CUENTA DE PUNTOS FUNCIÓN DESAJUSTADOS

El procedimiento usado en COCOMO 2.0 para determinar los Puntos Función Desajustados es el siguiente:

1. Contar las funciones según su tipo: la cuenta de funciones desajustadas debería ser realizada por un técnico basándose en la información recopilada en los documentos de requerimientos y diseño del software. El número de cada uno de los cinco tipos de funciones de usuario (Ficheros Lógicos Internos (ILF), Ficheros de Interface Externos (EIF), Entrada Externa (EI), Salida Externa (EO), y Consultas Externas (EQ)) debe de ser obtenido.

2. Contar las funciones atendiendo al nivel de complejidad: contar y clasificar cada función dentro de los niveles (Bajo, Medio, o Alto) de complejidad dependiendo de los tipos de datos y el número de tipos de ficheros referenciados. Utilizar para ello el siguiente esquema:

Para ILF y EIF				Para EO y EQ				Para EI			
Elementos Registro	Elementos Dato			Tipos de Ficheros	Elementos Dato			Tipos de Ficheros	Elementos Dato		
	1-19	20-50	51+		1-5	6-19	20+		1-4	5-15	16+
1	Bajo	Bajo	Med	0 o 1	Bajo	Bajo	Med	0 o 1	Bajo	Bajo	Med
2-5	Bajo	Med	Alto	2-3	Bajo	Med	Alto	2-3	Bajo	Med	Alto
6+	Med	Alto	Alto	4+	Med	Alto	Alto	3+	Med	Alto	Alto

3. Aplicar los pesos asignados a cada nivel de complejidad: utilizar los pesos del siguiente esquema (los pesos reflejan el valor relativo de cada función para el usuario):

Tipos de Función	Complejidad-Peso		
	Bajo	Medio	Alto
Ficheros Lógicos Internos	7	10	15
Ficheros de Interface Externos	5	7	10
Entradas Externas	3	4	6
Salidas Externas	4	5	7
Consultas Externas	3	4	6

4. Calcular los Puntos Función Desajustados: sumar todos los pesos contados para obtener un único número, número que determina el valor de los Puntos Función Desajustados.

CONVERSIÓN DE PUNTOS FUNCIÓN A LÍNEAS DE CÓDIGO

Para determinar el número nominal de personas mes que nos da la Ecuación 1 para el Modelo de Diseño Inicial, los Puntos Función Desajustados han de convertirse a líneas de código fuente que implementen el lenguaje (ensamblador, lenguaje de alto nivel, lenguaje de cuarta generación, etc).

COCOMO 2.0 realiza esto para los modelos de Diseño Inicial y Post-Arquitectura mediante el uso de tablas para poder transladar estos Puntos Función Desajustados en el equivalente a SLOC:

Lenguaje	SLOC/FP
Ada	71
AI Shell	49
APL	32
Ensamblador	320
Ensamblador (Macro)	213

Lenguaje	SLOC/FP
ANSI/Quick/Turbo Basic	64
Basic-Compilado	91
Basic-Interpretado	128
C	128
C++	29
ANSI Cobol 85	91
Fortran 77	105
Forth	64
Jovial	105
Lisp	64
Modula 2	80
Pascal	91
Prolog	64
Generador de Informes	80
Hoja de Cálculo	6

Tabla 10: Conversión de Puntos Función a Líneas de Código.

PARÁMETROS DE COSTE (*COST DRIVERS*)

El modelo de Diseño Inicial utiliza KSLOC para valorar el tamaño. Los Puntos Función Desajustados son convertidos a su equivalente en SLOC y luego a KSLOC. La aplicación de los factores de escala del proyecto es igual en el modelo de Diseño Inicial y en el modelo Post-Arquitectura. En el modelo de Diseño Inicial un reducido conjunto de parámetros de coste es utilizado. Los parámetros de coste del modelo de Diseño Inicial se obtienen por combinación de los parámetros de coste del modelo Post-Arquitectura de la Tabla 19. La ecuación de esfuerzo es la misma que la dada por la Ecuación 4.

Aproximación global: Ejemplo de Capacidad Personal (PERS).

La siguiente aproximación es utilizada para "mapear" un conjunto completo de parámetros de coste y escalas de ratios de los participantes en el modelo de Diseño Inicial. Esto incluye el uso y combinación de equivalentes numéricos a los niveles de ratio. Más específicamente, a un parámetro de coste del modelo Post-Arquitectura con un ratio de "Muy Bajo" le corresponde un valor numérico de 1, 2 para "Bajo", 3 para "Nominal o medio", 4 para "Alto", 5 para "Muy Alto" y 6 para "Extra Alto". Para los parámetros de coste combinados del modelo de Diseño Inicial, los valores numéricos de los parámetros de coste del modelo Post-Arquitectura, Tabla 11, son sumados y el resultado total es asignado a un ratio de escala (desde Extra Bajo a Extra Alto) del modelo de Diseño Inicial.

Parámetro de Coste Diseño Inicial	Combinación Equivalente Post-Arquitectura
RCPX	RELY, DATA, CPLX, DOCU
RUSE	RUSE
PDIF	TIME, STOR, PVOL
PERS	ACAP, PCAP, PCON
PREX	AEXP, PEXP, LTEX
FCIL	TOOL, SITE
SCED	SCED

Tabla 11: Multiplicadores de esfuezo Diseño Inicial y Post-Arquitectura.

La escala de ratios del modelo de Diseño Inicial siempre tiene un total nominal igual a la suma de los ratios nominales de los elementos del modelo Post-Arquitectura con los que se han formado.

El siguiente ejemplo ilustrará esta aproximación. El parámetro de coste PERS del Diseño Inicial combina los parámetros de coste Post-Arquitectura de capacidad del analista (ACAP), capacidad del programador (PCAP), y continuidad del personal (PCON). Cada uno de estos tiene una escala de ratios desde Muy Bajo (=1) a Muy Alto (=5). Sumando estos ratios numéricos se produce un rango de valores entre 3 y 15. Estos son diseñados sobre una escala, y los niveles de ratio PERS del Diseño Inicial son asignados a ellos, tal y como muestra la Tabla 19.

	Extra Bajo	Muy Bajo	Bajo	Nominal	Alto	Muy Alto	Extra Alto
Suma de rangos ACAP, PCAP, PCON	3, 4	5, 6	7, 8	9	10, 11	12, 13	14, 15
Combinación porcentajes ACAP y PCAP	20%	39%	45%	55%	65%	75%	85%
Personal que anualmente regresa	45%	30%	20%	12%	9%	5%	4%

Tabla 12: Niveles de ratio PERS

El ratio personal PERS con valor 9 corresponde a la suma (3+3+3) de los ratios nominales para ACAP, PCAP y PCON, y su multiplicador de esfuerzo correspondiente es 1'0. Tener en cuenta que de todas formas el ratio nominal PERS de 9 resulta de poder haber sumado otras combinaciones, por ejemplo 1+3+5=9 para ACAP=Muy Bajo, PCAP=Nominal, y PCON=Muy Alto.

Las escalas de ratio y los multiplicadores de esfuerzo para PCAP y los otros parámetros de coste de Diseño Inicial mantienen la consistencia relacional con sus equivalentes en el modelo Post-Arquitectura. Por ejemplo, los niveles de ratio Extra Bajo de PERS (20% combinando ACAP y PCAP, y el 25% de movimiento de personal) representan niveles de ratio medios de ACAP, PCAP, y PCON por encima del 3 o 4.

Manteniendo estas relaciones de consistencia entre los niveles de ratio del Diseño Inicial y Post-Arquitectura se asegura la consistencia en las estimaciones de coste de estos dos modelos (de Diseño Inicial y Post-Arquitectura).

Complejidad y Confianza del Producto.

Estos parámetros de coste del Diseño Inicial combinan los cuatro parámetros de coste Post-Arquitectura (Confianza Software Requerida (RELY), tamaño de la base de datos (DATA), Complejidad del Producto (CPLX), y Documentación asociada a las necesiddes del ciclo de vida (DOCU)). A diferencia de los componentes PERS, los componentes RCPX tienen escalas de ratio con diferente margen. Los rangos de RELY y DOCU van desde Muy Bajo a Muy Alto; los rangos de DATA van desde Bajo a Muy Alto; y los rangos de CPLX van desde Muy Bajo a Extra Alto. Y con un valor numérico entre 5 (Muy Bajo, Bajo, Muy Bajo, Muy Bajo) y 21 (Muy Alto, Muy Alto, Extra Alto, Muy Alto).

La Tabla 19 asigna los ratios RCPX a través de este rango, y asocia escalas de ratio apropiadas a cada uno de los ratios RCPX (desde Extra Bajo a Extra Alto).

	Extra Bajo	Muy Bajo	Bajo	Nominal	Alto	Muy Alto	Extra Alto
Suma de ratios RELY, DATA, CPLX, DOCU	5, 6	7, 8	9 -11	12	13 - 15	16 - 18	19 - 21
Enfasis sobre confiana, documentación	Muy poco	Poco	Algo	Basico	Fuerte	Muy fuerte	Extremo
Complejidad del producto	Muy	Simple	Algo	Moderado	Compl	Muy	Extrema

	Extra Bajo	Muy Bajo	Bajo	Nominal	Alto	Muy Alto	Extra Alto
	Simple				ejo	complejo	damente complejo
Tamaño de la base da datos	Pequeño	Pequeño	Pequeño	Moderado	grande	Muy grande	Muy grande

Tabla 13: Niveles de ratio de RCPX

Reutilización Requerida (RUSE).

Este parámetro de coste del Diseño Inicial es el mismo a su homónimo en el modelo Post-Arquitectura. Un resumen de estos niveles de ratio se ofrece aquí, y en la Tabla 19:

	Muy Bajo	Bajo	Nominal	Alto	Muy Alto	Extra Alto
RUSE		ninguno	A través del proyecto	A través del programa	A través de la línea de producto	A través de múltiples líneas de producto

Tabla 14: Resumen del nivel de ratio de RUSE

Inconvenientes de la Plataforma (PDIF).

Este parámetro de coste del Diseño Inicial combina los tres parámetros de coste Post-Arquitectura (tiempo de ejecución (TIME), almacenamiento principal (STOR), y volatilidad de la plataforma (PVOL)). TIME y STOR tienen rangos desde Nominal a Extra Alto; PVOL rangos desde Bajo a Muy Alto. Y con un valor numérico entre 8 (Nominal, Nominal, Bajo) y 17 (Extra Alto, Extra Alto, Muy Alto).

	Bajo	Nominal	Alto	Muy Alto	Extra Alto
Suma de TIME, STOR, y PVOL	8	9	10 - 12	13 - 15	16, 17
Restricción de almacenamiento y Tiempo	#50%	#50%	65%	80%	90%
Volatilidad de la plataforma	Muy estable	estable	Algo volatil	volatil	Altamente volatil

Tabla 15: Niveles de ratio de PDIF

Experiencia Personal (PREX).

Este parámetro de coste del Diseño Inicial combina los tres parámetros de coste Post-Arquitectura (experiencia en la aplicación (AEXP), experiencia en la plataforma (PEXP), y experiencia en las herramientas y lenguajes (LTEX)). Cada uno de estos con un rango desde Muy Bajo a Muy Alto. Y con un valor numérico entre 3 y 15.

	Extra Bajo	Muy Bajo	Bajo	Nominal	Alto	Muy Alto	Extra Alto
Suma de ratios de AEXP, PEXP, y LTEX	3, 4	5, 6	7, 8	9	10, 11	12, 13	14, 15
Experiencia en Aplicaciones, Plataforma, Lenguaje y Herramientas	#3 meses	5 meses	9 meses	1 año	2 años	4 años	6 años

Tabla 16: Niveles de ratio PREX

Facilidades (FCIL).

Este parámetro de coste del Diseño Inicial combina dos parámetros de coste Post-Arquitectura: uso de herramientas software (TOOL) y desarrollo multisitio o distribuido (SITE). TOOL tiene un rango desde Muy Bajo a Muy Alto; el rango de SITE va desde Muy Bajo a Extra Alto. El valor numérico suma de estos ratios está entre 2 (Muy Bajo, Muy Bajo) y 11 (Muy Alto, Extra Alto).

Planificación (SCED).

Este parámetro de coste de Diseño Inicial es el mismo que su homómino en el modelo Post-Arquitectura. Ver Tabla 19.

	Extra Bajo	Muy Bajo	Bajo	Nominal	Alto	Muy Alto	Extra Alto
Suma ratios de TOOL y SITE	2	3	4, 5	6	7, 8	9, 10	11
Soporte TOOL	Minima	Alguna	Simple colección de herramientas CASE	Herramientas de ciclo de vida básicas	Buena; moderadamente integrada	Fuerte; moderadamente integrada	Fuerte; muy integrada
Condiciones Multisitio	Escaso soporte de complejo desarrollo multisitio	Algún soporte de complejo desarrollo multisitio	Algún soporte de moderado desarrollo multisitio	Soporte básico de moderado desarrollo multisitio	Fuerte soporte de moderado desarrollo multisitio	Fuerte soporte de simple desarrollo multisitio	Muy fuerte de ordenado o simple desarrollo multisitio

Tabla 17: niveles de ratio FCIL.

	Muy Bajo	Bajo	Nominal	Alto	Muy Alto	Extra Alto
SCED	75% del nominal	85%	100%	130%	160%	

Tabla 18: Resumen de nivel de ratio SCED.

MODELO POST-ARQUITECTURA.

Este modelo es el más detallado y es utilizado cuando la arquitectura del ciclo de vida del software ha sido desarrollada. Este modelo es usado en el desarrollo y mantenimiento de productos software como Generadores de Aplicaciones, Integración de Sistemas e Infraestructura.

NOMAS PARA CONTAR LAS LÍNEAS DE CÓDIGO

En COCOMO 2.0 las sentencias lógicas fuente han sido elegidas como el estándar para determinar la línea de código. Definir el concepto de línea de código es una difícil tarea debido a las diferencias conceptuales que suponen para diferentes lenguajes las sentencias ejecutables y declaraciones de datos. El objetivo es medir la cantidad de trabajo intelectual puesto en el desarrollo del programa, pero surgen dificultades cuando intentamos definir medidas consistentes para lenguajes diferentes. Para minimizar estos problemas, la definición que da el Instituto de Ingeniería del Software (SEI) de sentencia lógica fuente es utilizada para definir la medida de líneas de códigos.

La figura 5 de la página siguiente muestra cómo una parte de esta definición es aplicada para soportar el desarrollo del modelo COCOMO 2.0. Cada marca en la columna "Incluye" identifica un tipo de sentencia particular o atributo incluido en la definición, y viceversa para la columna "Exclude". Otras secciones en la definición clarifican los atributos de las sentencias para su uso, reparto, funcionalidad, replicación y estudio de desarrollo. Hay también aclaraciones para sentencias específicas de lenguajes como ADA, C, C++, CMS-2, COBOL, FORTRAN, JOVIAL y Pascal.

Algunos cambios fueron realizados para definir línea de código para apartarse de la definición por defecto. Estos cambios eliminan categorías de software las cuales resultan ser generalmente pequeñas fuentes de esfuerzo del proyecto. El código generado con generadores de código fuente no está incluido, aunque sin embargo será tenido en cuenta para soportar el análisis.

La definición de línea de código de COCOMO 2.0 está calculada directamente por la colección de herramientas métricas automatizadas *Amadeus*, las cuales son usadas para asegurar la uniformidad de la colección de datos dentro de los datos recopilados y del análisis del proyecto de COCOMO 2.0.

Para soportar mejor el análisis de datos, Amadeus tomará automáticamente medidas adicionales que incluyen las líneas fuente totales, comentarios, sentencias ejecutables, declaraciones, estructura, componentes de interfaz, y otros. La herramienta proporcionará varias medidas del tamaño.

Definition Checklist for Source Statements Counts

Definition name: Logical Source Statements Date: _____

(basic definition) Originator: COCOMO 2.0

Measurement unit:	Physical source lines						
	Logical source statements	✔					
Statement type	Definition	✔	Data Array			Includes	Excludes
When a line or statement contains more than one type, classify it as the type with the highest precedence.							
1 Executable	Order of precedence →				1	✔	
2 Nonexecutable							
3 Declarations					2	✔	
4 Compiler directives					3	✔	
5 Comments							
6 On their own lines					4		✔
7 On lines with source code					5		✔
8 Banners and non-blank spacers					6		✔
9 Blank (empty) comments					7		✔
10 Blank lines					8		✔
11							
12							
How produced	Definition	✔	Data array			Includes	Excludes
1 Programmed						✔	
2 Generated with source code generators							✔
3 Converted with automated translators						✔	
4 Copied or reused without change						✔	
5 Modified						✔	
6 Removed							✔
7							
8							
Origin	Definition	✔	Data array			Includes	Excludes
1 New work: no prior existence						✔	
2 Prior work: taken or adapted from							
3 A previous version, build, or release						✔	
4 Commercial, off-the-shelf software (COTS), other than libraries							✔
5 Government furnished software (GFS), other than reuse libraries							✔
6 Another product							✔
7 A vendor-supplied language support library (unmodified)							✔
8 A vendor-supplied operating system or utility (unmodified)							✔
9 A local or modified language support library or operating system							✔
10 Other commercial library							✔
11 A reuse library (software designed for reuse)						✔	
12 Other software component or library						✔	
13							
14							

Figure 5. Definition Checklist

Figura 5

PUNTOS FUNCIÓN

Para la estimación de puntos función del modelo Post-Arquitectura, son válidos los cálculos utilizados para convertir los Puntos Función Desajustados a KSLOC que se utilizan en el modelo de Diseño Inicial. COCOMO 2.0 permite que algunos componentes sean evaluados utilizando puntos función, y otros (en los cuales los punto función no los describen correctamente, tales como cálculos científicos o en tiempo real) en SLOC. Toda medida es expresada en KSLOC y usada en la Ecuación 4. El Apéndice A muestra la ecuación para el modelo Post-Arquitectura.

PARÁMETROS DE COSTE

Existen 17 multiplicadores de esfuerzo utilizados en el modelo Post-Arquitectura para ajustar el esfuerzo nominal, Persona mes, para poder reflejar el producto software bajo desarrollo. Estos multiplicadores son agrupados en cuatro categorias: del producto, de la plataforma, personales, y del proyecto. La figura 19 muestra los diferentes parámetros de coste junto a sus criterios para establecer los rangos. Siempre que una valoración de un parámetro de coste está entre un rango de valores cercanos al valor medio, por ejemplo si un parámetro de coste está entre Alto y Muy Alto, entonces seleccionaremos Alto. Una parte de estos multiplicadores del esfuerzo han sido tratados ya en el modelo de Diseño Inicial.

Factores del Producto:

- Confianza Software Requerida (RELY). Esta es una medida hasta el punto de lo que el software debe realizar mediante las funciones construidas y durante un periodo de tiempo. Si el efecto de un fallo software es únicamente producir pequeños inconvenientes, entonces RELY tienen un valor bajo. Si un fallo llegase a arriesgar la vida humana entonces RELY tomaría un valor muy alto.

- Tamaño de Base de Datos (DATA). Esta medida pretende capturar los efectos que tienen requerimientos de gran cantidad de datos sobre el desarrollo del producto. El porcentaje está determinado por cálculo D/P. El tamaño de la base de datos es una consideración importante debido al esfuerzo requerido para generar todos los test de datos.

$$\frac{D}{P} = \frac{DataBaseSize\,(Bytes)}{ProgramSize\,(SLOC)}$$ **Ecuación 10**

DATA toma un valor bajo si D/P es menor de 10, y toma un valor muy alto si es mayor de 100.

- Complejidad del Producto (CPLX). La Tabla 20 proporciona la nueva escala de ratios de CPLX en el modelo COCOMO 2.0. La complejidad es dividida en cinco áreas: operaciones de control, operaciones computacionales (de cálculo), operaciones que son dependientes de los dispositivos, operaciones de manejo de datos, y operaciones de gestión del interfaz de usuario. Una selección de estás áreas o combinación de ellas caracterizará

el producto, o a un subsistema o parte del producto. La escala de complejidad es una valoración subjetiva media de estas áreas.

- Reutilización Requerida (RUSE). Este parámetro de coste mide el esfuerzo adicional necesario para la construcción de componentes que puedan reutilizarse en el actual o en futuros proyectos. Este esfuerzo es consumido durante la creación de un diseño software más genérico, una documentación más elaborada, y un chequeo más exhaustivo para asegurarse de que los componentes quedan listos para ser utilizados en otras aplicaciones.

	Muy bajo	Bajo	Nominal	Alta	Muy alta	Extra Alta
RELY	Inconvenientes sin importancia	Poco	Moderado	Alta	Riesgos	
DATA		DB bytes/Pgm SLOC<10	10#D/P<100	100#D/P<1000	D/P ô 1000	
CPLX	Mirar tabla sobre la valoración de complejidad según el tipo de modulo					
RUSE		None	Determinado en el proyecto	Determinado en el programa	Determinado en línea	Determinado en multiples líneas
DOCU	A penas necesarios	Algo necesario	Tamaño ajustado a las necesidades del ciclo de vida	Excesivo a las necesidades del ciclo devida	Muy excesivas para las necesidades del ciclo de vida	
TIME			#50% del tiempo usado para la ejecución	70%	85%	95%
STOR			#50% del almacenamiento disponible	70%	85%	95%
PVOL	Grandes cambios cada 12 meses, y cambios pequeños cada mes	Grandes: 6 meses. Pequeños: 2 semanas.	Grandes: 2 meses. Pequeños: 1 semana.	Grandes: 2 semanas. Pequeños: 2 días.		
ACAP	15%	35%	55%	75%	90%	
PCAP	15%	35%	55%	75%	90%	
PCON	45%/año	24%/año	12%año	6%/año	3%/año	
AEXP	#2meses	6 meses	1 año	3 años	6 años	
PEXP	#2meses	6 meses	1 año	3 años	6 años	
LTEX	#2meses	6 meses	1 año	3 años	6 años	
TOOL	Editor, codificar, depurar	CASE hacia delante, hacia atrás y simple, pequeña integración	Herramientas básicas del ciclo de vida, moderada integración	Madured en la utilización de herramientas del ciclo de vida, integración moderada	Idem, integración de procesos, metodos, reutilización	
SITE: Distribución	Internacional	Varias ciudades y varias compañias	Varias ciudades o varias compañias	Misma ciudad o misma zona	Mismo edificio o complejo	Completamente junto
SITE: Comunicación	Telefono y correo	Correo indivicual y fax	Correo electrónico en un canal angosto	Cominicación electrónica utilizando un canal ancho	Idem, ocasional video conferencia	Interactivo Multimedia
SCED	75% del nominal	85%	100%	130%	160%	

Tabla 19. Parámetros de coste multiplicadores del esfuerzo, para el Modelo Post-Arquitectura.

	Muy baja	Baja	Nominal	Alta	Muy alta	Extra alta
Operaciones de control	Lineas simples de código con poco codigo y sin estructuras anidadas: DOs, CASEs, IFTHENELSs. Llamadas a procedimientos simples.	Uso anidado de operadores de programación estructurada de forma directa. Mayoritariamente uso de predicados simples.	Mayoría de los anidamientos de operadores son simples. Algún control entre modulos. Tablas de decisión. Paso de mensajes, incluyendo proceso distribuido a medio nivel	Programación altamente anidada con mucha composición de predicados. Procesos distribuidos. Control en tiempo real simple.	Codigo reentrante y recursivo. Manejo de interrupciones con prioridad. Sincronización de tareas, control en tiempo real complejo.	Múltiple planificación de recursos con prioridades que cambian dinamicamente. Control a nivel de microcódigo. Control distribuido en tiempo real.
Operaciones de computo	Evaluación simple de expresiones: A=B+C*(D-E)	Evaluación de expresiones a nivel medio: SQRT(B^2-4*A*C)	Uso de rutinas estandar, matemáticas y estadísticas. Operaciones básicas sobre matrices o vectores.	Analisis numérico básico: multivariable, interpolacion, ecuaciones diferenciales de primer orden.	Dificil y estructurado análisis numérico: ecuaciones matriciales, ecuaciones diferenciales parciales. Paralelización simple.	Analisis numérico dificil y no estructurado: análisis altamente preciso de ruido, datos estocásticos. Paralelización compleja.
Operaciones dependientes de dispositivos	Lecturas simples, escritura de declaraciones con formas simples.	No es necesario conocimiento sobre procesadores I/O particulares. La I/O se realiza a niivel de GET/PUT.	El proceso de I/O incluye selección de dispositivos, comprobación de estado, y proceso de los errores.	Operaciones de I/O a nivel fisico (traducción de direcciones físicas, posicionamientos en memoria, lecturas, ...).	Rutinas para diagnóstico, de servicio, de enmascaramiento de interrupciones. Manejo de lina de comunicaciones. Sistemas de proceso intensivo.	Codificación con control de tiempos de dispositivos, operaciones de microprogramación. Sistemas de proceso crítico.
Operaciones de manejo de datos	Arrays sencillos en memoria principal. Consultas y actualizaciones simples de la base de datos.	Tramtamiento de ficheros de forma simple, sin cambios en la estrucutra de datos, sin ficheros intermedios, ... Consultas y actualizaciones moderadas de la base de datos.	Entrada desde varios ficheros, y salida única. Cambios simples en la estructura. Consultas y actualizaciones complejas.	Simples señales de activación flujos de datos. Reestructuración de datos compleja.	Coordinación de bases de datos distribuidas. Señales de activación complejas. Optimización de búsquedas.	Altamente acoplados, relacionados dinámicamente y con estructura de objetos. Manejo de lenguaje natural de datos.
Operaciones para gestión del interfaz de usuario	Formularios de entrada simples. Generador de listados.	Uso de GUIs simples.	Uso simple de elementos de interfaz.	Desarrollo de elementos de interfaz. Multimedia y I/O simple con voz.	Moderadamente complejo, 2D/3D, gráficos dinámicos.	Multimedia compleja, realidad virtual.

Tabla 20. Valoración de complejidad según el tipo de modulo

- Documentación relacionada con las necesidades del ciclo de vida (DOCU). Distintos modelos de coste software poseen un parámetro de coste para valorar el nivel de documentación requerida. En COCOMO 2.0, la escala de ratio para el parámetro de coste DOCU es evaluada en términos de la adecuada documentación del proyecto que necesita el ciclo de vida. Esta escala va desde Muy Bajo (se observan algunas necesidades del ciclo de vida) hasta Muy Alto (muchas y grandes necesidades del ciclo de vida).

Factores de la Plataforma:

La plataforma se refiere al complejo formado por el hardware e infraestructura software (antes conocido como máquina virtual) de la máquina objetivo. Los factores han sido revisados para reflejar estos parámetros tal y como se describen. Algunos factores adicionales de la plataforma fueron considerados, tal como distribución, paralelismo, sistemas empotrados, y operaciones en tiempo real. Estas consideraciones han sido acomodadas mediante la expansión de los porcentajes del Modulo de Complejidad en la Ecuación 20.

- Tiempo de Ejecución Necesitado (TIME). Esta es una medida del tiempo de ejecución que es impuesto, por necesidad, por el sistema software. El porcentaje es expresado en términos del tanto por ciento del tiempo de ejecución disponible que se espera que ea consumido por el sistema o subsistema, del total del recurso formado por el tiempo de ejecución. Los rangos van desde Nominal, menos del 50% usado de este recurso, hasta el Extra Alto, 95% de este tiempo.

- Volatilidad de la plataforma (PVOL). El término "plataforma" es utilizado para comprender el complejo conjunto formado por el hardware y software (OS, DBMS, etc) que el producto software utiliza, llama, para realizar sus tareas. Si el software a desarrollar es un sistema operativo entonces la plataforma es el hardware de la computadora. Si es un sistema gestor de bases de datos lo que va a desarrollarse, entonces la plataforma es el hardware y el sistema operativo. Si es un navegador texto de red el objetivo del desarrollo, entonces la plataforma es la red, el hardware de la computadora, el sistema operativo, y los lugares distribuidos donde se localiza la información. La plataforma incluye cualquier compilador o ensamblador utilizado para desarrollar el sistema software. Este rango va desde Bajo, donde existen cambios importantes en la plataforma cada 12 meses, hasta Muy Alta, con cambios importantes cada dos semanas aproximadamente.

Factores Personales:

- Capacidad de los Analistas (ACAP). Los analistas son personas que trabajan sobre los requerimientos, diseñan a un alto nivel y lo detallan. Los principales atributos que deberían de ser considerados en esta categoría son la habilidad y eficiencia, y la habilidad de comunicación y cooperación. Esta escala no debería de expresar el nivel de experiencia del analista, que ya es valorado con AEXP.

- Capacidad del Programador (PCAP). La tendencia actual enfatiza la importancia de analistas altamente capaces, sin embargo el papel creciente de los complejos paquetes COTS, y la influencia de una significativa productividad asociada con la habilidad de los programadores para tratar con estos paquetes COTS, indica una buena tendencia que da mayor importancia a las capacidades del programador.

  La evaluación debería de basase en la capacidad de los programadores para formar grupos más que en su individualidad. Los principales factores que deberían ser considerados para establecer estos niveles son pues la habilidad y eficiencia, y la habilidad de comunicación y cooperación. La

experiencia de los programadores no debería de ser considerada aquí, ya que es valorada con AEXP.

- Experiencia en las Aplicaciones (AEXP). Esta valoración es dependiente del nivel de experiencia que se posee en las aplicaciones por el equipo de desarrollo. Los ratios son definidos en términos de equipos de proyecto equivalentes al nivel de experiencia con ese tipo de aplicación. Un rango de Muy Bajo para una experiencia menor de 2 meses, y Muy Alto si se posee una experiencia igual o superior a 6 años.

- Experiencia en la Plataforma (PEXP). El modelo Post-Arquitectura amplia la influencia que sobre la productividad tiene este parámetro PEXP, reconociendo la importancia de comprender el uso de más plataformas potentes, incluyendo interfaces de usuario gráficos, bases de datos, trabajo con redes, etc...

- Experiencia con Herramientas y Lenguajes (LTEX). Esta es una medida del nivel de experiencia con lenguajes de programación y utilidades software que posee el equipo de desarrollo del sistema o subsistema. El desarrollo del software incluye el uso de herramientas para realizar los requerimientos, y el análisis y diseño de la representación, manejo de la configuración, extracción de la documentación, gestión de librerias, formateo y estilo de programa, chequeo de consistencia, etc... Adicionalmente a la experiencia en programación con un lenguaje específico, el soporte de un conjunto de utilidades o herramientas tiene efectos sobre el tiempo de desarrollo. Un ratio Bajo se da para una experiencia menor de 2 meses, y de Muy Alto para experiencias iguales o mayores a 6 años.

- Continuidad del Personal (PCON). El ratio de la escala para PCON se da en términos del personal que retorna anualmente al proyecto: desde 3%, Muy Alto, al 48%, Muy Bajo.

Factores del Proyecto:

- Uso de herramientas software ITOOL). Las herramientas software han mejorado significativamente desde los años 70, que sirvieron para calibrar el COCOMO. El ratio de rangos van desde las simples herramientas de edición y coficicación, a las que se les asigna un valor asociado de Muy Bajo, hasta las herramientas integradas de gestión del ciclo de vida, asignándoles un valor de Muy Alto.

- Desarrollo Multisitio (SITE). Dan la creciente frecuencia de desarrollos realizados en distintos lugares, e indican como este desarrollo multisitio tiene unos efectos significativos, es por ello que estas características fueron añadidas al COCOMO 2.0. La determinación de estos parámetros de coste incluyen la valoración y un promedio de dos factores: la situación (dentro de una distribución mundial) y el soporte de comuncicaciones (desde el correo y teléfono, hasta los completos medios interactivos de carácter multimedia).

- Planificación de Desarrollo Requerida (SCED). Este ratio mide la planificación temporal a establecer, obligada e impuesta por el equipo de desarrollo del proyecto del software. Los ratios son definidos en términos

del porcentaje de planificación que se alarga o acelera con respecto a la planificación media para un proyecto que necesita una cantidad de esfuerzo determinado. Planificaciones temporales aceleradas tienden a producir más esfuerzo en las últimas fases de desarrollo debido a que más asuntos son dejados para ser determinados posteriormente. Una planificación comprimida un 74% es valorada como un porcentaje Muy Bajo, y un alargamiento del 160% es valorada como un porcentaje Muy Alto.

GLOSARIO

3GL	Lenguaje de Tercera Generación
AA	Porcentaje de esfuerzo de reusabilidad que es debido a la Valoración y Asimilación de ese código reutilizable.
ACAP	Capacidad del Analista
AEXP	Experiencia en la Aplicación
ASLOC	Adaptación de Línea de Código Fuente
BRAK	Breakage. Cantidad de cambios controlados que se permiten en un desarrollo software antes del "cierre" de los requerimientos.
CASE	Ingeniería del Software Asistida por Computador
CM	Porcentaje de código modificado persiguiendo la reutilización
CMM	Modelo de Capacidad de Madured
Cost Drivers	Párametro que influye en el coste del software. Es una característica particular de un desarrollo software que tiene efectos sobre la cantidad del esfuerzo de desarrollo, incrementandolo o decrementandolo.
COTS	"*Commercial Off The Shelf*"
CPLX	Complejidad del Producto
DATA	Tamaño de la Base de Datos.
DBMS	Sistema Gestor de Bases de Datos.
DI	Grado de Influencia.
DM	Porcentaje del diseño que es modificado debido a la reusabilidad.
DOCU	Documentación asociada con las necesidades del ciclo de vida.
ESLOC	Equivalente en Líneas de Código Fuente.
FCIL	Facilidades.
FP	Puntos Función
GUI	Interface Gráfico de Usuario

ICASE	Entorno Software Integrado Asistido por Computador
IM	Porcentaje de integración que se vuelve a hacer persiguiendo la reutilización de código
KSLOC	Miles de Líneas de Código Fuentes
LTEX	Experiencia en Lenguajes y Herramientas
NOP	Nuevos Puntos Objeto
OS	Sistemas Operativos
PCAP	Capacidad del Programador
PCON	Continuidad del Personal
PDIF	Dificultad de la Platadorma
PERS	Capacidad del Personal
PEXP	Experiencia en la Plataforma
PL	Linea de Producto
PM	Persona Mes. Es la cantidad de tiempo que dedica durante un mes una sóla pesona que trabaja sobre un proyecto de desarrollo software
PREX	Experiencia Personal
PROD	Porcentaje de Productividad
PVOL	Volatilidad de la Plataforma
RCPX	Complejidad y Fiabilidad del Producto
RELY	Fiabilidad Software Requerida
RUSE	Reusabilidad Requerida
SCED	Planificación de Desarrollo Requerida
SEI	Instituto de Ingeniería del Software
SITE	Desarrollo Multisitio
SLOC	Líneas de Código Fuente

STOR	Restricción de Almacenamiento Principal
SU	Porcentaje de esfuerzo requerido en la reutilización del código, y que es debido a la comprensión del software
TIME	Restricción de Tiempo de Ejecución
TOOL	Uso de Herramientas Software

Apéndice A: ECUACIONES

COMPOSICIÓN DE APLICACIONES .

Los Puntos Objeto Nuevos son determinados por:

$$NOP = \frac{(Object\ Points) \cdot (100 - \%Reuse)}{100}$$ **Ecuación 11**

El porcentaje de productividad, PROD, es estimado mediante una media subjetiva de experiencia del desarrollador y la capacidad y madured de la herramienta ICASE:

Capacidad y Experiencia del desarrollador	**Muy Bajo**	**Bajo**	**Medio**	**Alto**	**Muy Alto**
Capacidad y Madured de la herramienta ICASE	**Muy Bajo**	**Bajo**	**Medio**	**Alto**	**Muy Alto**
PROD	4	7	13	25	50

Y el esfuerzo estimado resulta:

$$PM = \frac{NOP}{PROD}$$ **Ecuación 12**

El esfuerzo estimado resulta:

$$PM = \prod_{i=1}^{7} \left(EM_i \right) \cdot A \cdot [Size]^{\left(1.01 + 0.01 \sum_{j=1}^{5} SF_j \right) + \left(\frac{ASLOC \cdot \left(\frac{AT}{100} \right)}{ATPROD} \right)}$$

Ecuación 13

donde

$$Size = KASLOC + \begin{cases} KASLOC \cdot \frac{100 - AT}{100} \cdot \frac{ADAPT + 30}{100}, where \;\; ADAPT > 0 \\ KASLOC \cdot 0.05, where \;\; ADAPT = 0 \end{cases}$$

y

$$SLOC = UFP\ Count \cdot \frac{Lines\ of\ Source\ Code}{UFP}$$

Símbolo	Descripción
A	Constante, provisionalmente con un valor 3'0
ADAPT	Porcentaje de componentes adaptados (representa el esfuerzo requerido para comprender el software)
AT	Porcentaje de componentes que son automáticamente transladados
EM	Multiplicadores del esfuerzo: RCPX, RUSE, PDIF, PERS, PREX, FCIL, SCDE
KASLOC	Tamaño del componente adaptado expresado en miles de líneas de código fuente adaptadas
KNSLOC	Tamaño del componente expresado en miles de nuevas líneas de código fuente
PM	Estimación de esfuerzo dada en Personas Mes
SF	Factores de escala: PREC, FLEX, RESL, TEAM, PMAT
SLOC	Tamaño de un componente expresado in líneas de código fuente
UFP	Puntos Función Desajustados
UFP Count	Tamaño del componente expresado en Puntos Función Desajustados

El esfuerzo estimado resulta:

$$PM = \prod_{i=1}^{17}\left(EM_i\right)\cdot A\cdot\left[\left(\frac{1+BRAK}{100}\right)\cdot Size\right]^{\left(1.01+0.01\sum_{j=1}^{5}SF_j\right)}+\left(\frac{ASLOC\cdot\left(\frac{AT}{100}\right)}{ATPROD}\right)$$

Ecuación 14

donde

$$Size = KNSLOC + \left[KASLOC\cdot\left(\frac{100-AT}{100}\right)\cdot\frac{(AA+SU+0.4\cdot DM+0.3\cdot CM+0.3\cdot IM)}{100}\right]$$

Símbolo	Descripción
A	Constante, provisionalmente con un valor de 3'0
AA	Evaluación y Asimilación
ADAPT	Porcenta de componentes adaptados (representa el esfuerzo requerido en comprender el software)
AT	Porcentaje de componentes que son automáticamente transladados
ATPROD	Productividad de la translación automática
BRAK	Breakage: porcentaje de código "tirado" debido a la volatilidad de los requerimientos
CM	Porcentaje de código modificado
DM	Porcentaje de diseño modificado
EM	Multiplicadores del esfuerzo: RELY, DATA, CPLX, RUSE, DOCU, TIME, STOR, PVOL, ACAP, PCAP, PCON, AEXP, PEXP, LTEX, TOOL, SITE, SCED
IM	Porcentaje de integración y testeo del código modificado
KASLOC	Tamaño del componente adaptado expresado en miles de líneas de código fuente adaptadas
KNSLOC	Tamaño del componente expresado en miles de nuevas líneas de código fuente
PM	Estimación de esfuerzo dada en Personas Mes
SF	Factores de escala: PREC, FLEX, RESL, TEAM, PMAT
SU	Comprensión del software (cero si DM=0 y CM=0)

Determina el tiempo de desarrollo (TDEV) con un esfuerzo estimado, PM "negado", que excluye el efecto del multiplicador de esfuerzo SCED:

$$TDEV = \left[3.0 \times (\overline{PM})^{(0.33 + 0.2 \times (B - 1.01))} \right] \cdot \frac{SCED\%}{100}$$ **Ecuación 15**

donde

$$B = 1.01 + 0.01 \sum_{j=1}^{5} SF_j$$

Símbolo	Descripción
SCED%	Porcentaje de compresión/expansión del multiplicador de esfuerzo SCED
PM "negado"	Personas Mes de esfuerzo estimado para los modelos de Diseño Inicial y Post-Arquitectura (excluyendo el efecto del multiplicador de esfuerzo SCED)
SF	Factores de escala: PREC, FLEX, RESL, TEAM, PMAT
TDEV	Tiempo de desarrollo

ESTADO ACTUAL DEL MODELO .

COCOMO 2.O es actualmente una colección de datos, calibraciones y experiencias, donde se aportan datos de distintos proyectos que posteriormente se validan para determinar la consistencia con el modelo.

Han sido desarrolladas algunas herramientas, tanto experimentales para uso de investigadores del modelo, como comerciales.

BIBLIOGRAFÍA

- "Ingeniería del software"
  R. Fairley. Ed. McGraw-Hill

- "Ingeniería del software. Un enfoque práctico"
  R. Pressman. Ed. McGraw-Hill

- Documentos y notas de Pascual Gonzalez, Escuela Superior de Informática, Universidad de Castilla-La Mancha, Campus de Albacete.

- Direcciones URLs:

  + Universidad del Sur de California
    http://sunset.use.edu/COCOMOII/cocomo.html
    http://sunset.use.edu/techRpts/Reports.html

  + Softstars Systems
    http://www.softsartsystem.com

  +Varios
    http://si.di.fc.ul.pt/pbd/Transparentes
    http://www.geocities.com/ResearchTriangle/Facility/2917