



TEMA 7:
COCOMO II



7.1. Antecedentes de COCOMO II.

7.1.1. ¿Qué es COCOMO II?

7.1.1.1. Introducción.

El modelo original COCOMO se publicó por primera vez en 1981 por Barry Boehm y reflejaba las prácticas en desarrollo de software de aquel momento. En la década y media siguiente las técnicas de desarrollo software cambiaron drásticamente. Estos cambios incluyen el gasto de tanto esfuerzo en diseñar y gestionar el proceso de desarrollo software como en la creación del producto software, un giro total desde los mainframe que trabajan con procesos batch nocturnos hacia los sistemas en tiempo real y un énfasis creciente en la reutilización de software ya existente y en la construcción de nuevos sistemas que utilizan componentes software a medida.

Estos y otros cambios hicieron que la aplicación del modelo COCOMO original empezara a resultar problemática. La solución al problema era reinventar el modelo para aplicarlo a los 90. Después de muchos años de esfuerzo combinado entre USC-CSE¹, IRUS y UC Irvine² y las Organizaciones Afiliadas al Proyecto COCOMO II, el resultado es COCOMO II, un modelo de estimación de coste que refleja los cambios en la práctica de desarrollo de software profesional que ha surgido a partir de los años 70. Este nuevo y mejorado COCOMO resultará de gran ayuda para los estimadores profesionales de coste software.

Por tanto, COCOMO II es un modelo que permite estimar el coste, esfuerzo y tiempo cuando se planifica una nueva actividad de desarrollo software. Está asociado a los ciclos de vida modernos. El modelo original COCOMO ha tenido mucho éxito pero no puede emplearse con las prácticas de desarrollo software más recientes tan bien como con las prácticas tradicionales. COCOMO II apunta hacia los proyectos software de los 90 y de la primera década del 2000, y continuará evolucionando durante los próximos años.

En resumen, los objetivos a la hora de la creación del modelo COCOMO II fueron:

- Desarrollar un modelo de estimación de tiempo y de coste del software de acuerdo con los ciclos de vida utilizados en los 90 y en la primera década del 2000.
- Desarrollar bases de datos con costes de software y herramientas de soporte para la mejora continua del modelo.

¹ Universidad del sur de California. Centro para la Ingeniería del Software

² Unidad de investigación de software Irvine. Universidad Irvine de California



- Proporcionar un marco analítico cuantitativo y un conjunto de herramientas y técnicas para la evaluación de los efectos de la mejora tecnológica del software en costes y tiempo del ciclo de vida software.

Estos objetivos apoyan las necesidades primarias expresadas por los usuarios de la estimación de costes del software. En orden de prioridades, estas necesidades eran: el apoyo de la planificación de proyectos, la previsión de personal del proyecto, la preparación del proyecto, la replanificación, el seguimiento del proyecto, la negociación del contrato, la evaluación de la propuesta, la nivelación de recursos, exploración de conceptos, la evaluación del diseño y decisiones referentes a la oferta/demanda. Para cada una de estas necesidades COCOMO II proporcionará un apoyo más moderno que sus predecesores, el COCOMO original y Ada COCOMO.

Los cuatro elementos principales de la estrategia que ha seguido COCOMO II son:

- Preservar la apertura del COCOMO original.
- Desarrollar COCOMO II de forma que sea compatible con el futuro mercado del software
- Ajustar las entradas y salidas de los submodelos de COCOMO II al nivel de información disponible.
- Permitir que los submodelos de COCOMO II se ajusten a las estrategias de proceso particulares de cada proyecto.

COCOMO II sigue los principios de apertura usados en el COCOMO original. De esta manera todos sus algoritmos y relaciones están disponibles públicamente.

También todos sus interfaces están diseñadas para ser públicas, bien definidas y parametrizadas para que los preprocesos complementarios (modelos de analogía basados en casos de otras medidas de estimación), post-procesos (planificación de proyectos y herramientas de control, modelos dinámicos de proyectos, analizadores de riesgo) y paquetes de alto nivel (paquetes de gestión de proyectos, ayudas de negociación de producto) puedan combinarse estrechamente con COCOMO II.

Para apoyar a los distintos sectores del mercado software, COCOMO II proporciona una familia de modelos de estimación de coste software cada vez más detallado y tiene en cuenta las necesidades de cada sector y el tipo de información disponible para sostener la estimación del coste software. Esta familia de modelos está compuesta por tres submodelos cada uno de los cuales ofrece mayor fidelidad a medida que uno avanza en la planificación del proyecto y en el proceso de diseño. Estos tres submodelos se denominan:

- El modelo de Composición de Aplicaciones.
Indicado para proyectos construidos con herramientas modernas de construcción de interfaces gráficos para usuario.



- El modelo de Diseño anticipado.
Este modelo puede utilizarse para obtener estimaciones aproximadas del coste de un proyecto antes de que esté determinada por completo su arquitectura. Utiliza un pequeño conjunto de drivers de coste nuevo y nuevas ecuaciones de estimación. Está basado en Punto de Función sin ajustar o KSLOC (Miles de Líneas de Código Fuente).
- El modelo Post-Arquitectura.
Este es el modelo COCOMO II más detallado. Se utiliza una vez que se ha desarrollado por completo la arquitectura del proyecto. Tiene nuevos drivers de coste, nuevas reglas para el recuento de líneas y nuevas ecuaciones.

La primera implementación de COCOMO II se presentó al público general a mediados de 1997. USC COCOMOII.1997 se calibró con 83 puntos de datos (proyectos de desarrollo software históricos), utilizando un enfoque de media ponderada del 10% para combinar datos empíricos con la opinión del experto y calibrar los parámetros del modelo. USC COCOMOII.1998.0 beta se creó en Octubre de 1998. Esta versión se calibró con 161 puntos de datos y utilizando por primera vez un enfoque bayesiano para realizar la calibración del modelo. USC COCOMO II.1999.0 se ha publicado a mediados de 1999 y USC-COCOMO II.2000.0 será presentado en el año 2000. Mientras cada versión nueva de la herramienta USC COCOMO II fue mejorando en cuanto a amigabilidad con el usuario, las calibraciones del modelo de los años 1998, 1999 y 2000 son la misma. Es decir, no se han añadido nuevos puntos de datos a la base de datos utilizada para calibrar las versiones de la herramienta del año 1999 y del año 2000 aparte de aquellos que aparecieron en la calibración de la base de datos del año 1998. En las tres implementaciones aparecen los mismos 161 puntos de datos a los que hacíamos referencia anteriormente. Se prevé que la versión USC-COCOMO II.2001.0 ya incluirá una nueva calibración.

Por último, la experiencia ha demostrado que si una organización calibra la constante multiplicativa en COCOMO II para sus propios datos empíricos, la precisión del modelo puede aumentar significativamente por encima de los resultados de calibración genéricos obtenidos con las versiones mencionadas anteriormente.

7.1.2. Predecesores de COCOMO II.

7.1.2.1. Evolución de COCOMO 81 a COCOMO II y comparativa con sus predecesores.

Es importante recalcar los cambios experimentados por COCOMO II con respecto a COCOMO 81 ya que reflejan cómo ha madurado la tecnología de la Ingeniería del software durante las dos décadas pasadas. Por ejemplo, cuando se publicó el primer modelo COCOMO 81 los programadores estaban sometidos a tareas batch. El giro total que se experimentó afectó a su productividad. Por lo tanto un parámetro como TURN que reflejaba la espera media de un programador para recibir de vuelta su trabajo ahora no tiene sentido porque la mayoría de los programadores tienen acceso instantáneo a los recursos computacionales a través de su estación de trabajo. Este parámetro ha desaparecido en el nuevo modelo.



Las tablas 7.1 y 7.2 recalcan los principales cambios hechos a la versión original COCOMO 81 cuando se desarrolló COCOMO II y un resumen de las principales diferencias entre ambos. La tabla 7.2 presenta una comparativa por modelos más detallada de COCOMO II y además incluye el modelo AdaCOCOMO. De ambas tablas podemos deducir:

- COCOMO II se dirige a las siguientes tres fases del ciclo de vida en espiral: desarrollo de aplicaciones, diseño anticipado y Post-Arquitectura.
- Los tres modos del exponente se han reemplazado por cinco factores de escala.
- Se han añadido los siguientes drivers de coste a COCOMO II: DOCU, RUSE, PVOL, PEXP, LTEX, PCON y SITE.
- Se han eliminado los siguientes drivers de coste del modelo original: VIRT, TURN, VEXP, LEXP, Y MODP.
- Los valores de aquellos drivers de coste que se han conservado del modelo original fueron modificados considerablemente para reflejar las calibraciones actualizadas.

Las otras diferencias principales se refieren a efectos relacionados con el tamaño, que incluyen reutilización, reingeniería y cambios en efectos de escala.



	COCOMO 81	COCOMO II
ESTRUCTURA DEL MODELO	Un modelo único que asume que se ha comenzado con unos requisitos asignados para el software	Tres modelos que asumen que se progresa a lo largo de un desarrollo de tipo espiral para consolidar los requisitos y la arquitectura, y reducir el riesgo.
FORMA MATEMÁTICA DE LA ECUACIÓN DEL ESFUERZO	Esfuerzo = A (q) (SIZE) ^{Exponente}	Esfuerzo = A (q) (SIZE) ^{Exponente}
EXPONENTE	Constante fija seleccionada como una función de modo: <ul style="list-style-type: none"> ▪ Orgánico = 1.05 ▪ Semi-libre = 1.12 ▪ Rígido = 1.20 	Variable establecida en función de una medida de cinco factores de escala: <ul style="list-style-type: none"> ▪ PREC Precedencia ▪ FLEX Flexibilidad de desarrollo ▪ RESL Resolución de Arquitectura / Riesgos ▪ TEAM Cohesión del equipo ▪ PMAT Madurez del proceso
MEDIDA	Líneas de código fuente (con extensiones para puntos de función)	Puntos objeto, Puntos de función ó líneas de código fuente.
DRIVERS DE COSTE (c _i)	15 drivers, cada uno de los cuales debe ser estimado: <ul style="list-style-type: none"> ▪ RELY Fiabilidad ▪ DATA Tamaño Base de datos ▪ CPLX Complejidad ▪ TIME Restricción tiempo de ejecución ▪ STOR Restricción de almacenamiento principal ▪ VIRT Volatilidad máquina virtual ▪ TURN Tiempo de respuesta ▪ ACAP Capacidad del analista ▪ PCAP Capacidad programador ▪ AEXP Experiencia aplicaciones ▪ VEXP Experiencia máquina virtual ▪ LEXP Experiencia lenguaje ▪ TOOL Uso de herramientas software ▪ MODP Uso de Técnicas modernas de programación ▪ SCED Planificación requerida 	17 drivers, cada uno de los cuales debe ser estimado: <ul style="list-style-type: none"> ▪ RELY Fiabilidad ▪ DATA Tamaño Base de datos ▪ CPLX Complejidad ▪ RUSE Reutilización requerida ▪ DOCU Documentación ▪ TIME Restricción tiempo de ejecución ▪ STOR Restricción de almacenamiento principal ▪ PVOL Volatilidad plataforma ▪ ACAP Capacidad del analista ▪ PCAP Capacidad programador ▪ AEXP Experiencia aplicaciones ▪ PEXP Experiencia plataforma ▪ LTEX Experiencia lenguaje y herramienta ▪ PCON Continuidad del personal ▪ TOOL Uso de herramientas software ▪ SITE Desarrollo Multi-lugar ▪ SCED Planificación requerida
OTRAS DIFERENCIAS DEL MODELO	Modelo basado en: <ul style="list-style-type: none"> ▪ Fórmula de reutilización lineal ▪ Asunción de requisitos razonablemente estables 	Tiene muchas otras mejoras que incluyen: <ul style="list-style-type: none"> ▪ Fórmula de reutilización No-lineal ▪ Modelo de reutilización que considera esfuerzo necesario para entender y asimilar ▪ Medidas de rotura que se usan para abordar la volatilidad de requisitos ▪ Características de autocalibración

Tabla 7.1. Comparativa entre COCOMO 81 Y COCOMO II



7.1.2.2. COMPARATIVA CON SUS PREDECESORES

	COCOMO 81	AdaCOCOMO	COCOMO II Modelo Composición de aplicaciones	COCOMO II Modelo Diseño Anticipado	COCOMO II Modelo Post-Arquitectura
MEDIDA	Instrucciones fuente entregadas (DSI) ó Líneas de Código fuente (SLOC)	DSI ó SLOC	Puntos Objeto	Puntos de función (FP) y lenguaje ó SLOC	Puntos de función (FP) y lenguaje ó SLOC
REUTILIZACION	SLOC Equivalente= lineal f(DM,CM,IM)	SLOC Equivalente= lineal f(DM,CM,IM)	Implícito en el modelo	%reutilización sin modificar: SR %reutilización modificada: no lineal f(AA,SU,DM,CM,IM)	SLOC equivalente= no lineal f(AA,SU,DM,CM,IM)
ROTURA	Medida de Volatilidad de los requisitos (RVOL)	Medida RVOL	Implícito en el modelo	Rotura % (BRAK)	Rotura % (BRAK)
MANTENIMIENTO	Tráfico anual de cambio (ACT) =%añadido + %modificado	ACT	Modelo de reutilización de Puntos Objeto	Modelo de reutilización	Modelo de reutilización
Factor B en $MN_{nom} = A (Size)^B$	<ul style="list-style-type: none"> ▪ Orgánico = 1.05 ▪ Semi-libre = 1.12 ▪ Rígido = 1.20 	Rígido: 1.04-1.24 dependiendo del grado de : <ul style="list-style-type: none"> ▪ Eliminación anticipada del riesgo ▪ Arquitectura solida ▪ Requisitos estables ▪ Madurez del proceso Ada 	1.0	1.01-1.21 dependiendo del grado de: <ul style="list-style-type: none"> ▪ Precedencia ▪ Conformidad ▪ Arquitectura anticipada, resolución de riesgos. ▪ Cohesión del equipo ▪ Madurez del proceso 	1.01-1.21 dependiendo del grado de: <ul style="list-style-type: none"> ▪ Precedencia ▪ Conformidad ▪ Arquitectura anticipada, resolución de riesgos. ▪ Cohesión del equipo ▪ Madurez del proceso
DRIVERS DE COSTE DE PRODUCTO	RELY, DATA, CPLX	RELY*, DATA*, CPLX*	Ninguno	RCPX* ^T , RUSE* ^T	RELY, DATA, DOCU* ^T , CPLX* ^T , RUSE* ^T
DRIVERS DE COSTE DE LA PLATAFORMA	TIME, STOR, VIRT, TURN	TIME, STOR, VMVH, VMVT, TURN	Ninguno	Dificultad de la plataforma: PDIF* ^T	TIME, STOR, PVOL(=VIRT)
DRIVERS DE COSTE DEL PERSONAL	ACAP, AEXP, PCAP, VEXP, LEXP	ACAP*, AEXP, PCAP*, VEXP, LEXP*	Ninguno	Capacidad y experiencia del personal: PERS* ^T , PREX* ^T	ACAP*, AEXP*, PCAP*, PEXP* ^T , LTEX* ^T , PCON* ^T
DRIVERS DE COSTE DEL PROYECTO	MODP, TOOL, SCED	MODP*, TOOL*, SCED, SECU	Ninguno	SCED* ^T , FCIL* ^T	TOOL* ^T , SCED, SITE* ^T

Tabla 7.2. Comparativa entre COCOMO II y sus predecesores.

* Multiplicadores diferentes

^T Medida de escala diferente



7.2. CONTEXTO DE UTILIZACIÓN DE LOS MODELOS DE COSTE DE COCOMO II.

7.2.1. SITUACIÓN DEL SW EN EL MERCADO FUTURO

“Nos estamos convirtiendo en una compañía de software”, es una frase cada vez más repetida en organizaciones tan diversas como las finanzas, transporte, aeroespacial, electrónica y las empresas industriales. La ventaja competitiva depende cada vez más del desarrollo de productos inteligentes y a medida, y de la habilidad de desarrollar y adaptar más rápidamente que los competidores estos productos y servicios.

La notable reducción de los costes del hardware y la comodidad de las soluciones software han influido indirectamente en los costes del desarrollo de sistemas. Esta situación hace que sean aún más importantes los cálculos coste-beneficio, la selección de los componentes adecuados para la construcción y evolución del ciclo de vida de un sistema y el convencimiento de la escéptica dirección financiera de la ventaja comercial de las inversiones en software. También resalta la necesidad de productos coexistentes, la determinación del proceso y la habilidad de dirigir análisis trazados entre el software y los costes del ciclo de vida del sistema, tiempos de ciclo, funciones y calidades.

Al mismo tiempo, una nueva generación de procesos software y productos están cambiando la manera en que las organizaciones desarrollan software: Enfoque evolutivo, riesgo controlado, procesos software colaborativos, enfoque de desarrollo software de trayectoria rápida, lenguajes de 4ª generación, enfoque dirigido a la reutilización software. Todas estas prácticas mejoran la calidad del software y reducen el riesgo, el coste y el tiempo de ciclo.

Sin embargo, aunque alguno de los ya existentes modelos de coste tienen iniciativas que se dirigen a aspectos de estos problemas, estos nuevos acercamientos no se han emparejado lo suficiente a los nuevos modelos complementarios para estimación de software y planificación. Esto dificulta a las organizaciones la realización de planes efectivos, análisis y control de proyectos usando los nuevos enfoques.

Estas preocupaciones han llevado a la formulación de una nueva versión del modelo de coste constructivo COCOMO para la estimación del esfuerzo, del tiempo y del coste software. El COCOMO original y su especializado sucesor Ada COCOMO fueron razonablemente bien equiparados con los tipos de proyectos software que ellos modelizaban: Gran extensión de clientes, software construido para la especificación. Aunque Ada COCOMO añadió la capacidad de estimar costes y tiempos para el desarrollo incremental de software, COCOMO encontró una creciente dificultad para estimar los costes de software comercial, de software orientado a objetos, de software desarrollado mediante modelos en espiral ó modelos de desarrollo evolutivo, ó de software desarrollado en gran parte mediante utilidades de composición de aplicaciones comerciales a medida.

La figura 7.1 resume el modelo de mercado de las futuras prácticas de software que usamos para guiar el desarrollo de COCOMO II. Incluye en la plataforma superior un sector dedicado a la programación para



usuarios finales a la que pertenecerán alrededor de 55 millones de personas en Estados Unidos cerca del año 2005. A un nivel más bajo, un sector de infraestructura, al que pertenecerán aproximadamente 0.75 millones de personas y 3 sectores intermedios que incluyen el desarrollo de generadores de aplicaciones y ayudas para la composición (0.6 millones), desarrollo de sistemas mediante la composición de aplicaciones (0.7 millones) y sistemas de integración a gran escala y/o sistemas de software embebidos (0.7 millones).

Programación de usuario final (55,000,000)		
Generador de Aplicaciones y Ayudas a la Composición (600,000)	Composición de Aplicación (700,000)	Integración de Sistema (700,000)
Infraestructura (750,000)		

Figura 7.1. Modelo de mercado de futuras prácticas de software.

La programación para usuarios finales estará dirigida por la creciente capacidad de la computadora de leer y escribir y por la presión competitiva para la creación de soluciones de proceso de información rápidas, flexibles y manejables por el usuario. Estas tendencias empujarán al mercado del software a tener usuarios que desarrollen ellos mismos la mayoría de aplicaciones que procesan información mediante generadores de aplicaciones.

Algunos ejemplos de generadores de aplicaciones son las hojas de cálculo, los sistemas de queries extendidas y los sistemas de inventarios de planificación especializados. Ellos facilitarán el que los usuarios sean capaces de determinar la aplicación que procese la información que ellos deseen mediante la opción de dominios familiares ó reglas simples. Cada empresa de las compañías Fortune100 para pequeños negocios y el departamento de defensa de Estados Unidos estarán involucrados en este sector.

Los productos típicos del sector de la infraestructura estarán en las áreas de Sistemas Operativos, Sistemas gestores de bases de datos, Sistemas de gestión de interfaz con el usuario y Sistemas de redes. Cada vez más, el sector de la infraestructura se dirigirá hacia soluciones “middleware” para problemas genéricos tales como el proceso distribuido y el proceso transaccional. Empresas representativas del sector de la infraestructura son Microsoft, NeXT, Oracle, SyBase, Novell y los principales vendedores de computadoras.

En contraste con los programadores para usuarios finales que suelen conocer bien el dominio de sus aplicaciones y relativamente poca informática, los desarrolladores de infraestructura saben mucho de



informática y relativamente poco de aplicaciones. La línea de sus productos tendrá muchos componentes reutilizables pero el avance de la tecnología (nuevos procesadores, memoria, comunicaciones, displays y tecnología multimedia) les exigirá que construyan muchos componentes y utilidades desde el principio.

Las personas que pertenecen a los tres sectores intermedios de la figura 5.2, necesitarán conocer una buena parte de informática y de software especializado para infraestructura, y también uno ó más dominios de aplicación. Esto supondrá un gran reto.

El sector de los Generadores de Aplicaciones creará numerosos paquetes de utilidades para la programación de usuarios. Firmas típicas que operan en este sector son Microsoft, Lotus, Novell, Borland y vendedores de sistemas para la ingeniería, la manufacturación y la planificación asistida por ordenador. Su línea de productos tendrá muchos componentes reutilizables, pero requerirá en gran parte el desarrollo de utilidades a partir de cero. El sector de la Composición de aplicaciones trata con aplicaciones demasiado diversificadas como para ser utilizadas en soluciones empaquetadas, pero que son los suficientemente sencillas como para ser rápidamente compuestas a partir de componentes interoperables. Componentes típicos serán Desarrolladores de interfaces gráficos para usuarios, Bases de datos ó Gestores de objetos, middleware para procesos distribuidos ó procesos transaccionales, manejadores hipermedia, Buscadores de datos sencillos y Componentes de dominio específico tales como paquetes de procesos de control médicos, financieros ó industriales.

La mayoría de las grandes empresas tendrán grupos para la creación de aplicaciones como estas pero muchas de las empresas especializadas en software se comprometerán a proporcionar aplicaciones compuestas. Dichas empresas van desde las grandes y versátiles como Andersen Consulting y EDP hasta pequeñas empresas especializadas en áreas tales como el soporte a la decisión ó proceso transaccional, ó en dominios de aplicaciones tales como las finanzas ó la manufacturación.

El sector de la Integración de sistemas trata con sistemas a gran escala, altamente embebidos ó sistemas sin precedentes. Parte de estos sistemas pueden desarrollarse utilizando utilidades de composición de aplicaciones, pero su demanda requiere una significativa cantidad de sistemas de ingeniería up-front y de desarrollo tradicional de software. Las empresas aeroespaciales trabajan dentro de este sector, así como empresas de integración de sistemas especializados tales como EDS y Andersen Consulting, importantes empresas en desarrollo de productos software-intensivos y servicios, empresas de telecomunicaciones, automoción, finanzas y productos electrónicos, y empresas que desarrollan sistemas de información corporativa a gran escala ó sistemas de soporte a la fabricación.

7.2.2. COCOMO II. MODELOS PARA LOS SECTORES DEL MERCADO DEL SW.



Programación de usuario final		
Generador de Aplicaciones y Ayudas a la Composición	Composición de Aplicación	Integración de Sistema
Infraestructura		

Figura 7.2. Modelo de mercado de futuras prácticas de software

El sector de la programación de usuarios finales de la figura 7.2 no necesita un modelo COCOMO II. El desarrollo de sus aplicaciones lleva de días a horas, así pues, una estimación simple basada en actividad, será suficiente.

Programación de usuario final		
Generador de Aplicaciones y Ayudas a la Composición	Composición de Aplicación	Integración de Sistema
Infraestructura		

Figura 7.3. Modelo de mercado de futuras prácticas de software

El modelo COCOMO II para el sector de la composición de aplicaciones (figura 7.3) está basado en Puntos Objeto. Los Puntos Objeto son un conjunto de pantallas, informes y módulos de lenguajes de 3ª generación desarrollados en la aplicación, cada uno ponderado mediante un factor de complejidad de tres niveles (simple, medio y complejo). Esto se corresponde con el nivel de información que normalmente se conoce de un producto de Composición de aplicaciones durante sus fases de planificación y el nivel correspondiente de exactitud que se necesita para estimar el coste software (dichas aplicaciones se desarrollan usualmente por un equipo pequeño en semanas ó meses).



Programación de usuario final		
Generador de Aplicaciones y Ayudas a la Composición	Composición de Aplicación	Integración de Sistema
Infraestructura		

Figura 7.4. Modelo de mercado de futuras prácticas de software

La capacidad de COCOMO II para la estimación del desarrollo de un Generador de Aplicación, Integración de un Sistema ó Infraestructura (figura 7.4) está basada en una mezcla hecha a medida del modelo de Composición de Aplicaciones (para tiempos de prototipados tempranos ó anticipados) y dos modelos de estimación cada vez más detallados para porciones del ciclo de vida, Diseño Anticipado y Post-Arquitectura.

Con respecto a la estrategia de proceso, los proyectos de Generador de Aplicaciones, Integración de Software e Infraestructura involucrarán a una mezcla de los tres principales Modelos de proceso. Los drivers apropiados dependerán de los drivers del mercado del proyecto y del grado de conocimiento del producto.

La razón para proporcionar esta mezcla hecha a medida se basa en tres premisas fundamentales:

- 1°. A diferencia de la situación del COCOMO inicial a finales de los 70, en los que había un único y preferido modelo de ciclo software, los proyectos de software actuales y futuros ajustan sus procesos a los particulares drivers de proceso. Estos drivers de proceso incluyen COTS ó disponibilidad de software reutilizable, grados de composición de arquitecturas y requisitos, ventana de mercado, y fiabilidad necesaria.
- 2°. La granularidad que usa el modelo de estimación de coste software debe ser consistente con la información disponible para dar soporte a la estimación del coste software. En las primeras fases de un proyecto software se conoce muy poco del tamaño del producto que se desarrolla, la naturaleza de la plataforma designada, la naturaleza del personal involucrado en el proyecto ó los datos concretos del proceso que se utilizará..
- 3°. Dada la situación de las premisas 1 y 2, COCOMO II permite que los proyectos proporcionen información sobre los drivers de coste, de grano grueso en las primeras etapas del proyecto y progresivamente información de grano fino en las etapas posteriores. Así pues COCOMO II no produce valores de puntos de coste y esfuerzo software, sino un rango de valores asociado al grado de definición de las entradas estimadas.



7.2.3. ¿CUÁNDO UTILIZAR CADA MODELO DE COSTE COCOMO II?

7.2.3.1. UTILIZACIÓN DEL MODELO DE COMPOSICIÓN DE APLICACIONES

Las primeras fases o ciclos en espiral utilizados en proyectos software de Generador de Aplicaciones, Integración de Sistema e Infraestructura implicarán generalmente prototipado y utilizarán las utilidades del Módulo de Composición de Aplicaciones. El Modulo de Composición de Aplicaciones COCOMO II, soporta estas fases y cualquier otra actividad de prototipado que se realice más adelante en el ciclo de vida.

Este modelo se dirige a aplicaciones que están demasiado diversificadas para crearse rápidamente en una herramienta de dominio específico, (como una hoja de cálculo) y que todavía no se conocen suficientemente como para ser compuestas a partir de componentes interoperables. Ejemplos de estos sistemas basados en componentes son los creadores de interfaces gráficas para usuario, bases de datos ó gestores de objetos, middleware para proceso distribuido ó transaccional, manejadores hipermedia, buscadores de datos pequeños y componentes de dominio específico tales como paquetes de control de procesos financieros, médicos ó industriales.

Dado que el Modelo de Composición de Aplicaciones incluye esfuerzos de prototipado para resolver asuntos potenciales de alto riesgo tales como interfaces de usuario, interacción software/sistema, ejecución ó grado de madurez tecnológica, los costes de este tipo de esfuerzo se estiman mejor mediante dicho modelo.

7.2.3.2. UTILIZACIÓN DEL MODELO DE DISEÑO ANTICIPADO

Hemos visto que las primeras fases o ciclos en espiral utilizados en proyectos software de Generador de Aplicaciones, Integración de Sistema e Infraestructura se ajustaban mejor al modelo de Composición de Aplicaciones.

Las siguientes fases ó ciclos espirales normalmente incluirán la exploración de arquitecturas alternativas ó estratégicas de desarrollo incremental. Para sostener estas actividades COCOMO II proporciona un modelo de estimación anticipado : el Modelo de Diseño Anticipado. El nivel de detalle de este modelo puede ser consistente con el nivel general de información disponible y con el nivel general de aproximación de la estimación requerida en esta etapa.

El Diseño Anticipado incluye la exploración de arquitecturas de software/sistema alternativas y conceptos de operación. En esta fase no se sabe lo suficiente como para dar soporte a la estimación de grano fino. La correspondiente capacidad de COCOMO II incluye el uso de Puntos de Función y un conjunto de siete drivers de coste de grano grueso (por ejemplo, dos drivers de coste para capacidad del personal y



experiencia del personal en lugar de los seis drivers de coste del Modelo Post-Arquitectura que cubren varios aspectos de capacidad del personal, continuidad y experiencia).

El modelo de Diseño Anticipado usa Puntos de Función No Ajustados como métrica de medida. Este modelo se utiliza en las primeras etapas de un proyecto software, cuando se conoce muy poco sobre el tamaño del producto que se va a desarrollar, la naturaleza de la plataforma objetivo, la naturaleza del personal involucrado en el proyecto ó especificaciones detalladas del proceso que se va a usar. Este modelo puede aplicarse a cada uno de los sectores de desarrollo de Generador de Aplicaciones, Integración de sistemas ó Infraestructura. (ver apartado 5.2.1, Situación del software en el mercado futuro).

7.2.3.3. UTILIZACIÓN DEL MODELO POST-ARQUITECTURA

Hemos visto que las primeras fases o ciclos en espiral usados en proyectos software de Generador de Aplicaciones, Integración de Sistema e Infraestructura se ajustaban mejor al modelo de Composición de Aplicaciones y que las siguientes fases ó ciclos espirales normalmente serán sostenidas por el modelo de Diseño Anticipado. Una vez que el proyecto está listo para desarrollar y sostener un sistema especializado, debe haber una arquitectura de ciclo de vida que proporcione información más precisa de los drivers de coste de entradas y permita cálculos de coste más exactos. Para apoyar esta etapa COCOMO II proporciona el Modelo Post-Arquitectura.

El Modelo Post-Arquitectura incluye el actual desarrollo y mantenimiento de un producto software. Esta fase avanza rentablemente si se desarrolla una arquitectura de ciclo de vida software válida con respecto a la misión del sistema, al concepto de operación y al riesgo, y establecido como marca de trabajo del producto. El modelo correspondiente de COCOMO II tiene aproximadamente la misma granularidad que los anteriores modelos COCOMO y AdaCOCOMO. Utiliza instrucciones fuente y/ó Puntos de Función para medir , con modificadores para reutilización y objetos software; un conjunto de 17 drivers de coste multiplicativos; y un conjunto de 5 factores que determinan el exponente de escala del proyecto. Estos factores sustituyen los modos de desarrollo (Orgánico, Semilibre y Rígido) del modelo original COCOMO y refina los 4 factores de exponente-escala en Ada COCOMO.

7.3. MODELO DE COSTE DE COMPOSICIÓN DE APLICACIONES

7.3.1. INTRODUCCIÓN A LOS PUNTOS OBJETO

Los Puntos Objeto son el recuento de pantallas, informes y módulos de lenguajes de 3ª generación desarrollados en la aplicación, cada uno ponderado mediante un factor de complejidad de tres niveles (simple, medio y complejo) .



La estimación de Puntos Objeto es un enfoque relativamente nuevo de medida del software, pero encaja bien en las prácticas del sector de la Composición de Aplicaciones. También encaja muy bien en los esfuerzos de prototipado asociados, basados en el uso de herramientas ICASE que proporcionan constructores de interfaces gráficas de usuario, herramientas de desarrollo software; y en general, infraestructura que puede componerse y componentes de aplicación. En estas áreas se ha comparado bien con la estimación de Puntos de Función en un conjunto de aplicaciones no triviales (pero todavía limitadas).

Uno de los estudios comparativos entre la estimación de Puntos de Función y Puntos Objeto analizó una muestra de 19 proyectos software sobre inversión bancaria de una misma organización, desarrollado utilizando las utilidades de Composición de Aplicaciones ICASE y con un rango de esfuerzo de 4.7 a 71.9 MM (Meses-Persona). El estudio descubrió que el enfoque de Puntos Objeto justificó un 73% de la varianza (R²) en meses-persona ajustados por la reutilización, comparada con el 76% para los Puntos de Función.

Un experimento posterior diseñado estadísticamente incluyó cuatro gestores de proyecto experimentados utilizando Puntos de Función y Puntos Objeto para estimar el esfuerzo requerido en dos proyectos completos (3.5 y 6 Meses-persona actuales), basado en las descripciones del proyecto disponibles al principio para dichos proyectos. El experimento descubrió que los Puntos de Función y los Puntos Objeto dan resultados comparablemente precisos (un poco más precisos con Puntos Objeto pero estadísticamente insignificantes). Desde el punto de vista de la utilización, el tiempo medio para producir un valor de Punto Objeto era de alrededor del 47% del correspondiente tiempo medio para producir un valor de Puntos de Función. También los gestores consideraron el método de Puntos Objeto más fácil de usar (ambos resultados fueron estadísticamente significantes).

De esta manera aunque estos resultados no están todavía extendidos, su ajuste al desarrollo software de Composición de Aplicaciones parece justificar la selección de Puntos Objeto como el punto de partida para el modelo de estimación de Composición de Aplicaciones.

7.3.2. PROCEDIMIENTO DE OBTENCIÓN DE PUNTOS OBJETO

La definición de los términos utilizados en los Puntos de Objetos es la siguiente:

NOP : Nuevos Puntos Objeto. (Cantidad de Puntos Objeto ajustados por la reutilización).

Srvr : Número de tablas de datos del servidor (mainframe ó equivalente) usadas junto con la pantalla o el informe.

Clnt : Número de tablas de datos del cliente (estación de trabajo personal) usadas junto con la pantalla o el informe.



%reuse : Porcentaje de pantallas, informes y módulos 3GL reutilizados a partir de aplicaciones anteriores prorrateadas por grado de reutilización.

Los ratios de productividad se basan en los datos del proyecto del año 1 y año 2. En el año 1 la herramienta CASE estaba construyéndose y los desarrolladores no la conocían. La productividad media de NOP/Mes-persona en los 12 proyectos del año 1 se asocia con los niveles bajos de desarrollador y madurez y capacidad ICASE. En los siete proyectos del año 2, ambos, herramientas CASE y capacidades de los desarrolladores se consideran más maduras. La productividad media era 25 NOP/Meses-persona, que se corresponde con los niveles altos de madurez ICASE y del desarrollador.

Hemos de destacar que el uso del término “objeto” en Puntos Objeto define pantallas, informes y módulos 3GL como objetos. Esto puede tener relación ó no con otras definiciones de objetos como aquellas características de posesión tales como, por ejemplo, pertenencia a una clase, herencia, encapsulación, paso de mensajes y así sucesivamente. Las reglas de recuento para “objetos” de esa naturaleza, cuando se usan en lenguajes como C++, se discutirán en el Capítulo del Modelo Post-Arquitectura.

1. Hacer el recuento de objetos: Estimar el número de pantallas, informes y componentes de las que consta esta aplicación. Suponer las definiciones estándar de estos objetos en el entorno ICASE correspondiente.
2. Clasificar cada instancia de objeto dentro de niveles de complejidad simple, media y difícil dependiendo de los valores de las dimensiones de la característica. Usar la tabla 7.3:

Para Pantallas			Para Informes				
Nº de vistas que contiene	# y fuente de tablas de datos			Nº de secciones que contiene	# y fuente de tablas de datos		
	Total<4 (<2 srvr <3 clnt)	Total<8 (2/3 srvr 3-5 clnt)	Total 8+ (>3 srvr >5 clnt)		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 ó 1	Simple	Simple	Medio
3 -7	Simple	Medio	Difícil	2 ó 3	Simple	Medio	Difícil
>8	Medio	Difícil	Difícil	4+	Medio	Difícil	Difícil

Tabla 7.3. Complejidad asociada a las instancias de objetos



3. Pesar el número de cada celda usando la tabla 7.4. El peso refleja el esfuerzo relativo que se requiere para implementar una instancia de ese nivel de complejidad:

Tipo de Objeto	Complejidad-Peso		
	Simple	Medio	Difícil
Pantalla	1	2	3
Informe	2	5	8
Componente 3 GL			10

Tabla 7.4. Pesos asociados a los niveles de complejidad

4. Determinar Puntos Objeto: Suma todas las instancias de objeto pesadas para conseguir un número. El recuento de Puntos Objeto.
5. Estimar el porcentaje de reutilización que se espera lograr en este proyecto. Calcular los nuevos Puntos Objeto a desarrollar.

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

6. Determinar un ratio de productividad PROD = NOP/Meses-persona a partir de la tabla 7.5:

Experiencia y capacidad de los desarrolladores	Muy Bajo	Bajo	Nominal	Alto	Muy Alto
ICASE madurez y capacidad	Muy Bajo	Bajo	Nominal	Alto	Muy Alto
PROD	4	7	13	25	50

Tabla 7.5 Ratio de productividad PROD



7. Calcular el valor Meses-persona estimado según la ecuación:

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

7.4. EL MODELO DE DISEÑO ANTICIPADO Y POST-ARQUITECTURA.

Los modelos de Diseño Anticipado y Post-Arquitectura se basan en la misma filosofía a la hora de proporcionar una estimación. Como hemos indicado ya su principal diferencia se produce en la cantidad y detalle de la información que se utiliza para obtener la estimación en cada uno de ellos. La fórmula básica para obtener una estimación de esfuerzo con ambos modelos es:

$$MM_{NOMINAL} = A \times (Size)^B$$

Esta ecuación calcula el esfuerzo nominal para un proyecto de un tamaño dado expresado en Meses-persona(MM).

Las entradas son la medida del desarrollo del software, una constante, A, y un factor de escala, B. La medida está en unidades de líneas de código fuente (KSLOC). Esto se deriva de la medida de módulos software que constituirán el programa de aplicación, puede estimarse también a partir de Puntos de Función sin ajustar convirtiendo a SLOC y luego dividiendo por 1000.

El factor de escala (exponencial B), explica el ahorro ó gasto relativo de escala encontrado en proyectos software de distintos tamaños. La constante A, se usa para cortar los efectos multiplicativos de esfuerzo en proyectos de tamaño incremental.

A continuación desarrollamos la fórmula completa del modelo de Diseño Anticipado y se describen sus componentes.

7.4.1. CONSTANTE A.

Como ya hemos visto anteriormente la constante A, se usa para capturar los efectos multiplicativos de esfuerzo en proyectos de tamaño incremental. Provisionalmente se le ha estimado un valor de 2.45 .

7.4.2. VARIABLE SIZE.

Dónde:



$$\underline{\text{Size}} = \text{Size} \times \left[1 + \frac{\text{BRAK}}{100} \right]$$

COCOMO II utiliza un porcentaje de Rotura BRAK para ajustar el tamaño eficaz del producto. La rotura refleja la volatilidad de los requisitos en un proyecto. Es el porcentaje de código desperdiciado debido a la volatilidad de los requisitos. El factor BRAK no se usa en el Modelo de Composición de Aplicaciones donde se espera un cierto grado de iteración en el producto y se incluye en la calibración de datos.

Por ejemplo, un proyecto que parte de 100.000 instrucciones (Size= 100.000) pero desecha el equivalente a 20.000 instrucciones tiene un valor de BRAK de 20. Esto debe usarse para ajustar el tamaño efectivo del proyecto a 120.000 instrucciones para una estimación COCOMO II. (Size = 100.000 x [1 + (20/100)])

El tamaño de una aplicación se mide en unidades de líneas de código fuente (KSLOC). Al igual que en la versión inicial del COCOMO, este valor se deriva de la medida de módulos software que constituirán el programa de aplicación, sin embargo, en la nueva versión COCOMO II puede estimarse también a partir de Puntos de Función sin ajustar convirtiendo a SLOC y luego dividiendo por 1000.

Si se opta por utilizar directamente el valor del número de líneas de código, la meta es medir la cantidad de trabajo intelectual que se emplea en el desarrollo del programa, pero las dificultades aparecen al intentar definir medidas consistentes en diferentes lenguajes.

Si se opta por utilizar los Puntos de Función sin Ajustar para determinar el tamaño del proyecto, éstos deben convertirse en líneas de código fuente en el lenguaje de implementación (ensamblador, lenguajes de alto nivel, lenguajes de cuarta generación, etc...) para evaluar la relativamente concisa implementación por Puntos de Función (ver tabla 7.6). COCOMO II realiza esto tanto en el Modelo de Diseño Anticipado como en el de Post-Arquitectura usando tablas que traducen Puntos de Función sin ajustar al equivalente SLOC.



Lenguaje	SLOC / UFP
Ada	71
Al Shell	49
APL	32
Assembly	320
Assembly (Macro)	213
ANSI/Quick/Turbo Basic	64
Basic - Compiled	91
Basic Interpreted	128
C	128
C++	29
Visual Basic	32
ANSI Cobol 85	91
Fortran 77	105
Forth	64
Jovial	105
Lisp	64
Modula 2	80
Pascal	91
Prolog	64
Report Generator	80
Spreadsheet	6

Tabla 7.6. Conversión de Puntos de Función a Líneas de código

A continuación sólo nos quedaría por hacer la conversión a KSLOC mediante la operación SLOC/1000.

Por ejemplo, si al aplicar el procedimiento de cálculo para puntos de función sin ajustar se obtiene un resultado de 165 UNFP (Puntos de Función sin Ajustar) y el proyecto va a desarrollarse en el lenguaje de programación C++:

165 UNFP x 29 = 4785 SLOC (Líneas de Código Fuente)

Haciendo la conversión mencionada anteriormente:

4785/1000= 4.785 KSLOC (Miles de Líneas de Código Fuente).



7.4.3. Variable B (ahorro y gasto software de escala).

$$B = 0.91 + 0.01 \times \sum_{j=1}^5 SF_j$$

Los modelos de estimación de coste del software a menudo tienen un factor exponencial para considerar los gastos y ahorros relativos de escala encontrados en proyectos software de distinto tamaño. El exponente B se usa para capturar estos efectos. El valor de B es calculado en la ecuación anterior.

Si $B < 1.0$. El proyecto presenta ahorros de escala. Si el tamaño del producto se dobla, el esfuerzo del proyecto es menor que el doble. La productividad del proyecto aumenta a medida que aumenta el tamaño del producto. Pueden lograrse algunos ahorros de escala del proyecto con herramientas de proyecto específicas (por ejemplo, simulaciones, testbeds) pero normalmente es difícil lograrlo. Para proyectos pequeños, fijar costes de salida tales como herramientas a medida y normas de montaje, e informes administrativos, son a menudo una fuente de ahorro de escala.

Si $B = 1.0$. Los ahorros y gastos de escala están equilibrados. Este modelo lineal se usa a menudo para la estimación de coste de proyectos pequeños. Se usa para el modelo COCOMO II : Composición de Aplicaciones.

Si $B > 1.0$. El proyecto presenta gastos de escala. Esto se debe normalmente a dos factores principales: El crecimiento del gasto en comunicaciones y el gasto en crecimiento de la integración de un gran sistema. Los proyectos más grandes tendrán más personal y por lo tanto más vías de comunicación interpersonales produciendo gasto. Integrar un producto pequeño como parte de uno más grande requiere no sólo el esfuerzo de desarrollar el producto pequeño sino también el gasto adicional en esfuerzo para diseñar, mantener, integrar y probar sus interfaces con el resto del producto.

El exponente B se obtiene mediante los denominados drivers de escala. La selección de drivers de escala se basa en la razón de que ellos son un recurso significativo de variación exponencial en un esfuerzo ó variación de la productividad del proyecto. Cada driver de escala tiene un rango de niveles de valores desde Muy Bajo hasta Extra Alto (tabla 7.7). Cada nivel de valores tiene un peso, SF, y el valor específico del peso se llama factor de escala. Un factor de escala de un proyecto, SF_j (ver tabla 7.8), se calcula sumando todos los factores y se usa para determinar el exponente de escala, B.



Factores de Escala (SF _f)	Muy Bajo	Bajo	Nominal	Alto	Muy Alto	Extra Alto
PREC	Completamente sin precedentes	Prácticamente sin precedentes	Casi sin precedentes	Algo familiar	Muy familiar	Completamente familiar
FLEX	Riguroso	Relajación ocasional	Algo de relajación	Conformidad general	Algo de conformidad	Metas generales
RESL*	Poco (20%)	Algo (40%)	A menudo (60%)	Generalmente (75%)	En su mayor parte (90%)	Por completo (100%)
TEAM	Interacciones muy difíciles	Algo de dificultad en las interacciones	Interacciones básicamente cooperativas	Bastante cooperativo	Altamente cooperativo	Completas interacciones
PMAT	Peso medio de respuestas "Sí" para el cuestionario de Madurez CMM					

Tabla 7.7. Factores de escala para el Modelo de COCOMO II de Diseño Anticipado .

*% de módulos de interfaz significativos especificados, % de riesgos significativos eliminados.

Factores de Escala (Wi)	Muy Bajo	Bajo	Nominal	Alto	Muy Alto	Extra Alto
PREC	6.20	4.96	3.72	2.48	1.24	0.00
FLEX	5.07	4.05	3.04	2.03	1.01	0.00
RESL*	7.07	5.65	4.24	2.83	1.41	0.00
TEAM	5.48	4.38	3.29	2.19	1.10	0.00
PMAT	7.80	6.24	4.68	3.12	1.56	0.00

Tabla 7.8. Valores de los Factores de escala para el Modelo de COCOMO II de Diseño Anticipado pertenecientes a la versión USC-COCOMOII.1999.0

Por ejemplo, si tenemos factores de escala con valores Muy Bajo, entonces un proyecto 100 KSLOC tendrá $SF_j = 31,62$, $B = 1.2325$ y un esfuerzo relativo $E = E = 100^{1.2325} = 291$ PM.

Los valores actuales de los factores de escala están reflejados en la tabla 7.8.



(PREC) (FLEX). Precedencia y Flexibilidad de desarrollo.

Estos dos factores de escala capturan en gran parte las diferencias entre los modos Orgánico, Semilibre y Rígido del modelo original COCOMO. La tabla 7.9 se organiza para trazar su rango de proyecto dentro de las escalas de Precedencia y Flexibilidad de desarrollo.

Características	Muy Bajo	Nominal/Alto	Extra Alto
Precedencia			
Comprensión organizacional	General	Considerable	Profundo
Experiencia en trabajo con sistemas Sw relacionados	Moderado	Considerable	Extenso
Desarrollo concurrente de nuevo Hw asociado y procedimientos operacionales	Extenso	Moderado	Algo
Necesidad de arquitecturas de proceso de datos innovativas, algoritmos	Considerable	Algo	Mínimo
Flexibilidad de Desarrollo			
Necesidad de conformidad del Sw con requisitos preestablecidos	Completo	Considerable	Básico
Necesidad de conformidad del Sw con especificaciones de interfaz externas	Completo	Considerable	Básico
Prioridad en finalización anticipada	Alto	Medio	Bajo

Tabla 7.9. Factores de escala relacionados con Modos de desarrollo COCOMO

(RESL) Arquitectura/Resolución de Riesgos

Este factor combina dos factores de medida de AdaCOCOMO “Minuciosidad del diseño por revisión del diseño del producto (PDR)” y “Eliminación de riesgos por PDR”. La tabla 7.10 consolida las medidas de AdaCOCOMO para formar una definición más comprensiva de los niveles de medida RESL de COCOMO II. La medida de RESL es la media pesada subjetiva de las características de la lista.



Características	Muy Bajo	Bajo	Nominal	Alto	Muy Alto	Extra Alto
El plan de gestión de riesgos identifica todos los items de riesgos críticos, establece hitos para resolverlos mediante PDR	Ninguno	Poco	Algo	Generalmente	A menudo	Completamente
Horario, presupuesto e hitos internos con PDR compatible con el Plan de gestión de riesgos	Ninguno	Poco	Algo	Generalmente	A menudo	Completamente
Tanto por ciento de horario desarrollado dedicado a establecer la arquitectura dados los objetivos generales del producto	5	10	17	25	33	40
Porcentaje de arquitectos Sw de alto nivel requeridos, disponibles para el proyecto	20	40	60	80	100	120
Herramientas de soporte disponibles para resolver items de riesgo, desarrollar y verificar garantías de la arquitectura	Ninguno	Poco	Algo	Bueno	Fuerte	Completo
Nivel de incertidumbre en drivers de arquitectura claves: misión, interfaz de usuario, COTS, Hw, tecnología, ejecución	Extremo	Significativo	Considerable	Algo	Poco	Muy Poco
Número y criticalidad de items de riesgo	>10 Crítico	5-10 Crítico	2-4 Crítico	1 Crítico	>5 No Crítico	<5 No Crítico

Tabla 7.10. RESL Componentes de medida

(TEAM). Cohesión del Equipo

El factor de escala de Cohesión del Equipo explica los recursos de turbulencia y entropía del proyecto debido a dificultades en la sincronización de los implicados en el proyecto, usuarios, clientes, desarrolladores, los que lo mantienen, etc... Estas dificultades pueden aparecer por las diferentes culturas y objetivos de los implicados; dificultades en conciliar objetivos y la falta de experiencia y de familiaridad de los implicados en trabajar como un equipo. La tabla 7.11 proporciona una información



detallada para los niveles de medida completos TEAM. La medida final es la media pesada subjetiva de las características de la lista.

Características	Muy Bajo	Bajo	Nominal	Alto	Muy Alto	Extra Alto
Consistencia de objetivos y culturas	Poco	Algo	Básico	Considerable	Fuerte	Completo
Habilidad y servicialidad para acomodar objetivos de otros grupos	Poco	Algo	Básico	Considerable	Fuerte	Completo
Experiencia de los Desarrollados en operar como un equipo	Nada	Poco	Poco	Básico	Considerable	Extensa
Para lograr visión compartida y compromisos	Nada	Poco	Poco	Básico	Considerable	Extensa

Tabla 7.11. TEAM Componentes de medida

(PMAT). Madurez del proceso

El procedimiento para determinar PMAT se obtiene a través del Modelo de Madurez de Capacidad del Instituto de Ingeniería del Software (CMM). El periodo de tiempo para medir la madurez del proceso es el momento en el que el proyecto comienza. Hay dos formas de medir la madurez del proceso. La primera toma el resultado de una evaluación organizada basada en el CMM.

Nivel de Madurez Global

- Nivel 1 CMM (Mitad inferior)
- Nivel 1 CMM (Mitad superior)
- Nivel 2 CMM
- Nivel 3 CMM
- Nivel 4 CMM
- Nivel 5 CMM

Áreas de Proceso Principales

La segunda está organizada en base a 18 áreas de proceso principales (KPA's) en el modelo de Madurez de Capacidad SEI. El procedimiento para determinar PMAT es decidir el porcentaje de conformidad para



cada uno de los KPA's. Si el proyecto ha sufrido una valoración CMM reciente entonces se usa el porcentaje de conformidad para la KPA global (basada en datos de valoración de la conformidad práctica principal). Si no se ha hecho una valoración entonces se usan los niveles de conformidad para las metas de los KPA's (con la escala Likert de abajo) para poner el nivel de conformidad. El nivel basado en meta (objetivo) de conformidad se determina haciendo una media basada en juicio de las metas de cada área de proceso principal (ver tabla 7.12.).



Áreas de Proceso Clave	Casi siempre (>90%)	Frecuente- mente (60-90%)	En la Mitad (40-60%)	Ocasional- mente (10-40%)	En Pocas Ocasiones (<10%)	No se Aplica	No se Conoce
1 Gestión de requisitos	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
2 Planificación de proyectos Software	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
3 Seguimiento de proyecto Software	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
4 Gestión de subcontrato Software	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
5 Aseguramiento de la calidad Software	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
6 Gestión de la configuración Software	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
7 Focos de proceso de organización	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
8 Definición de proceso de organización	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
9 Programa de formación	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
10 Gestión del Software integrado	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
11 Ingeniería de producto Software	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
12 Coordinación intergrupos	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
13 Informes detallados	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
14 Gestión de proceso cuantitativo	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
15 Gestión de calidad Software	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
16 Prevención de defectos	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
17 Gestión de cambio de tecnología	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
18 Gestión de cambio de proceso	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>



Tabla 7.12. Porcentaje de conformidad para las áreas de proceso principales.

- Verificar casi siempre, cuando las metas se logran de forma consistente y se establecen bien en procedimientos que operan bajo la norma (alrededor del 90% del tiempo).
- Verificar frecuentemente, cuando las metas se logran relativamente a menudo pero algunas veces se pasan por alto bajo circunstancias difíciles (alrededor del 60% al 90% del tiempo).
- Verificar sobre la metas, cuando las metas se logran alrededor de la mitad del tiempo (alrededor del 40% al 60% del tiempo).
- Verificar ocasionalmente, cuando las metas se logran a veces pero menos a menudo (entre el 10% y 3l 40% del tiempo).
- Casi nunca verificar, si las metas no se alcanzan casi nunca (menos del 10% del tiempo).
- No aplicar verificación, cuando se tiene conocimiento requerido para tu proyecto u organización y el KPA pero siente que el KPA no se adapta a tus circunstancias.
- No sabe verificar cuando no sabes cómo responder para el KPA.

Después de que el nivel de conformidad del KPA se determina, se pesa cada nivel de conformidad y se calcula un factor PMAT. Inicialmente todos los KPA's tendrán el mismo peso.

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

Por ejemplo, el valor del factor PMAT para un caso en el que aplicando la tabla de Áreas de Proceso Clave: (Tabla 7.12.), obtengamos los siguientes resultados:

1.Gestión de requisitos: 90%

2.Planificación de Proyectos Software: 50%

3.Seguimiento del proyecto software: 60%

4.Gestión de subcontrato software: 30%

5.Aseguramiento de la calidad software: 60%



-
- 6. Gestión de la configuración software: 80%
 - 7. Focos de proceso de organización: 50%
 - 8. Definición de proceso de organización: 45%
 - 9. Programa de formación: 80%
 - 10. Gestión del software integrado: 60%
 - 11. Ingeniería de producto software: 40%
 - 12. Coordinación intergrupos: 60%
 - 13. Informes detallados: 80%
 - 14. Gestión de proceso cuantitativo: 10%
 - 15. Gestión de calidad software: 25%
 - 16. Prevención de defectos: 20%
 - 17. Gestión de cambio de tecnología: 60%
 - 18. Gestión de cambio de proceso: 45%
- será: $PMAT = 5 - (9.45 \times (5/18)) = 2.375$
-

7.44. AJUSTE MEDIANTE DRIVERS DE COSTE

Los drivers de coste se usan para capturar características del desarrollo del software que afectan al esfuerzo para completar el proyecto. Los drivers de coste tienen un nivel de medida que expresa el impacto del driver en el esfuerzo de desarrollo. Estos valores pueden ir desde Extra Bajo hasta Extra Alto. Para el propósito del análisis cuantitativo, cada nivel de medida de cada driver de coste tiene un peso asociado. El peso se llama multiplicador de esfuerzo (EM). La medida asignada a un driver de coste es 1.0 y el nivel de medida asociado con ese peso se llama nominal. Si un nivel de medida produce más esfuerzo de desarrollo de software, entonces su correspondiente EM está por encima de 1.0. Recíprocamente si el nivel de medida reduce el esfuerzo entonces el correspondiente EM es menor que 1.0. La selección de multiplicadores de esfuerzo se basa en una fuerte razón que explicaría una fuente significativa de esfuerzo de proyecto ó variación de la productividad independientemente.

Los EM se usan para ajustar el esfuerzo Meses-persona nominal. Hay 7 multiplicadores de esfuerzo para el Modelo de Diseño Anticipado y 17 para el modelo de Post-Arquitectura .

$$MM = A \times (\text{Size})^B \times \prod \underline{EM}_i$$

Los drivers de coste del Diseño Anticipado se obtienen combinando los drivers de coste del modelo Post-Arquitectura de la tabla 7.13. Siempre que una evaluación de un driver de coste está entre niveles de ratio,



hay que redondear al valor más próximo al nominal. Por ejemplo, si un valor de un driver de coste está entre Muy Bajo y Bajo, entonces seleccionar Bajo.

Drivers de coste del Diseño Anticipado	Drivers de coste combinados, homólogos del 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 7.13. Multiplicadores de esfuerzo del Diseño Anticipado y Post-Arquitectura

7.4.4.1. OBTENCIÓN DE DRIVERS DE COSTE PARA MODELO DE DISEÑO ANTICIPADO

(RCPX). Fiabilidad del Producto y Complejidad

Este driver de coste del Diseño Anticipado combina los 4 drivers de coste: Fiabilidad Software (RELY); Tamaño de la Base de Datos (DATA), Complejidad del Producto (CPLX), y Documentos que necesita el Ciclo de Vida (DOCU).

La tabla 7.14 proporciona los niveles de medida de RCPX

	Extra Bajo	Muy Bajo	Bajo	Nominal	Alto	Muy Alto	Extra Alto
Énfasis en la fiabilidad, documentación	Muy Poco	Poco	Algo	Básico	Fuerte	Muy Fuerte	Extremo
Complejidad del producto	Muy Simple	Simple	Algo	Moderado	Complejo	Muy Complejo	Extremadamente Complejo
Medida de la Base de Datos	Pequeño	Pequeño	Pequeño	Moderado	Grande	Muy Grande	Muy Grande



Tabla 7.14. Niveles de medida RCPX

(RUSE) Reutilización Requerida

Este driver de coste del modelo de Diseño Anticipado (tabla 7.15) es el mismo que su homólogo de Post-Arquitectura.

	Muy Bajo	Bajo	Nominal	Alto	Muy Alto	Extra Alto
RUSE		Nada	A lo largo del programa	A lo largo del proyecto	A lo largo de la línea de producto	A lo largo de múltiples líneas de producto

Tabla 7.15. Resumen del nivel de medida RUSE

(PDIF) Dificultad de la Plataforma

Este driver de coste del Diseño Anticipado combina los 3 drivers de coste de Post-Arquitectura siguientes: Tiempo de Ejecución (TIME), Restricciones de Almacenamiento (STOR) y Volatilidad de la Plataforma (PVOL). La tabla 7.16 proporciona una ayuda para obtener los niveles de medida PDIF.

	Bajo	Nominal	Alto	Muy Alto	Extra Alto
Restricciones de tiempo y de almacenamiento	≤ 50%	>50%	65%	80%	90%
Volatilidad de la Plataforma	Muy Estable	Estable	Volátil	Volátil	Volátil

Tabla 7.16. Niveles de medida PDIF

(PREX) Experiencia Personal

Este driver de coste de Diseño Anticipado combina los 3 drivers de coste de Post-Arquitectura siguientes: Experiencia (AEXP), Experiencia en la Plataforma (PEXP) y Experiencia en el Lenguaje y Herramientas (LTEX). La tabla 7.17 asigna valores PREX en el rango correspondiente.



	Extra Bajo	Muy Bajo	Bajo	Nominal	Alto	Muy Alto	Extra Alto
Experiencia en aplicaciones, plataforma, lenguaje y herramienta	≤ 3 Meses	5 Meses	9 Meses	1 Año	2 Años	4 Años	6 Años

Tabla 7.17. Niveles de medida PREX

(FCIL) Facilidades

Este driver de coste de Diseño Anticipado combina los 2 drivers de coste de Post-Arquitectura siguientes: Uso de Herramienta Software (TOOL) y Desarrollo MultiLugar (SITE). La tabla 7.18 de la asigna valores FCIL.

	Extra Bajo	Muy Bajo	Bajo	Nominal	Alto	Muy Alto	Extra Alto
Soporte de TOOL	Mínimo	Algo	Herramienta CASE simple	Herramientas de ciclo de vida básicas	Bueno; moderado	Fuerte; moderado	Fuerte; Bien integrado
Condiciones Multilugar	Soporte débil de desarrollo multilugar complejo	Algo de soporte de desarrollo multilugar complejo	Algo de soporte de desarrollo multilugar moderadamente complejo	Soporte básico de desarrollo multilugar moderadamente complejo	Fuerte soporte de desarrollo multilugar moderadamente complejo	Fuerte soporte de desarrollo multilingue simple	Soporte muy fuerte de desarrollo multilingue simple

Tabla 7.18. Niveles de medida FCIL

(SCED) Planificación Temporal

El driver de coste de Diseño Anticipado (tabla 7.19.) es el mismo que su homólogo de Post-Arquitectura.

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

Tabla 7.19. Resumen de nivel de medida SCED



En la tabla 7.20. se reflejan los valores actualizados que toman los drivers de coste para el modelo de Diseño Anticipado:

	Extra Bajo	Muy Bajo	Bajo	Nominal	Alto	Muy Alto	Extra Alto
RCPX	0.73	0.81	0.98	1.00	1.30	1.74	2.38
RUSE	--	--	0.95	1.00	1.07	1.15	1.24
PDIF	--	--	0.87	1.00	1.29	1.81	2.61
PERS	2.12	1.62	1.26	1.00	0.83	0.63	0.50
PREX	1.59	1.33	1.12	1.00	0.87	0.71	0.62
FCIL	1.43	1.30	1.10	1.00	0.87	0.73	0.62
SCED	--	1.43	1.14	1.00	1.00	1.00	--

Tabla 7.20. Multiplicadores de esfuerzo actualizados para el modelo de Diseño Anticipado pertenecientes a la versión USC-COCOMOIL1999.0.

Por ejemplo, si al analizar los drivers de coste para nuestro proyecto obtenemos los siguientes resultados:

	MB	B	N	A	MA	EA
RCPX				x		
RUSE			x			
PDIF			x			
PERS			x			
PREX				x		
FCIL				x		
SCED			x			

EB=Extra Bajo
 MB=Muy Bajo
 B=Bajo
 N=Nominal
 A=Alto
 MA=Muy Alto
 EA=Extra Alto



Aplicando los valores de la tabla 7.20 obtendremos:

$$\prod_{i=1}^7 EM_i = 1.30 \times 1 \times 1 \times 1 \times 1 \times 0.87 \times 0.87 \times 1.0 = 0.984$$

i=1

En los drivers de coste como por ejemplo FCIL (ver tabla 7.18) se evalúan varias características, en este caso Soporte de TOOL y Condiciones multilugar. El resultado Alto se ha obtenido de hacer la media subjetiva entre Soporte de TOOL= Alto y Condiciones Multilugar = Muy Alto. Esta media es Alto porque como hemos visto anteriormente siempre que una evaluación de un driver de coste está entre dos niveles de ratio, hay que redondear al nivel más próximo al nominal. Si la evaluación hubiese estado entre Extra Bajo y Bajo la media subjetiva hubiese sido Muy Bajo.

7.4.4.2. OBTENCIÓN DE DRIVERS DE COSTE PARA MODELO DE DISEÑO POST-ARQUITECTURA

(RELY). Fiabilidad Requerida de Software

Esta es la medida de hasta qué punto el software debe realizar su función esperada durante un periodo de tiempo. Si el efecto de un fracaso es sólo una molestia ligera entonces RELY es Bajo. Si un fallo arriesgase vidas humanas entonces RELY es Muy Alto. (tabla 7.21)

	Muy Bajo	Bajo	Nominal	Alto	Muy Alto	Extra Alto
RELY	Inconveniente ligero	Bajo, pérdidas fácilmente recuperables	Moderado, pérdidas recuperables	Alta pérdida financiera	Riesgo de vidas humanas	

Tabla 7.21. Niveles de medida RELY

(DATA). Medida del Volumen de Datos

Esta medida intenta capturar lo que afecta en el desarrollo del producto unos requerimientos de datos grandes. La medida se determina calculando D/P. La razón por la que es importante considerar el tamaño de la Base de Datos es por el esfuerzo necesario para generar datos de prueba que se usarán para ejecutar el programa.

$$\frac{D}{P} = \frac{\text{DataBaseSize (Bytes)}}{\text{ProgramSize (SLOC)}}$$



DATA se valora como Bajo si D/P es menor que 10 y Muy Alto si es mayor que 1000 (tabla 7.22).

	Muy Bajo	Bajo	Nominal	Alto	Muy Alto	Extra Alto
DATA		DB bytes/ Pgm.SLOC < 10	$10 \leq D/P < 100$	$100 \leq D/P < 1000$	$D/P \geq 1000$	

Tabla 7.23. Niveles de medida DATA

(CPLX). Complejidad del Producto

La Tabla 7.24 proporciona la nueva escala de valores CPLX de COCOMO II. La complejidad se decide en 5 áreas: Funcionamiento de control, Funcionamiento computacional, Funcionamiento de Dispositivos dependientes, Funcionamiento del sector de datos y Funcionamiento del Gestor de Interfaz de Usuario. Se seleccionará el área ó combinación de áreas que caracterizan al producto ó a un subsistema del producto. La medida de complejidad es la media subjetiva de estas áreas.



	Operaciones de control	Operaciones computacionales	Operaciones dependientes del dispositivo	Operaciones de gestión de datos	Operaciones de gestión de interfaz de usuario
Muy Bajo	Código straight line con unos pocos operadores estructurados de programación no anidados: DOs, CASEs, IF THEN ELSEs. Composición de módulos simple via llamadas a procedimientos ó simples scripts	Evaluación de expresiones simples: p.e. $A=B+C*(D-E)$	Declaraciones simples de lectura, escritura con formatos simples	Arrays simples en memoria principal. Actualizaciones, COTS-DB queries simples.	Forma de entrada simple, generadores de informes
Bajo	Anidamiento sencillo de operadores de programación estructurados. Mayoría de predicados simples	Evaluación de expresiones de nivel moderado: p.e. $D=\text{SQRT}(B**2-4.*A*C)$	No necesario conocimiento de características del procesador particular ó del dispositivo de I/O. I/O hecha a nivel GET/PUT.	Ficheros unicos sin cambio en la estructura de datos, sin ediciones, sin ficheros intermedios. COTS- DB moderadamente complejos, queries, actualizaciones.	Uso de constructores simples de interfaces gráficos de usuarios
Nominal	Mayoría de anidamiento simple. Llamadas ó paso de mensajes simples que incluye procesamiento distribuido para soportar middleware	Uso de rutinas matemáticas y estadísticas estandar. Operaciones de matriz/vector básicas	El procesamiento de I/O incluye selección de dispositivo, estado de validación y procesamiento de errores.	Entrada de múltiples ficheros y salida de un único fichero. Cambios estructurales simples, ediciones simples. Queries, actualizaciones y COTS-DB complejas.	Uso simple de conjunto widget
Alto	Operadores de programación estructurados altamente anidados compuesto de muchos predicados. Control de pilas y colas. Proceso homogéneo y distribuido.	Análisis numérico básico; Interpolación multivariada, ecuaciones diferenciales ordinarias. Truncamiento básico,	Operaciones de I/O a nivel físico (traducciones de direcciones de almacenamiento físicas; búsquedas, lecturas, etc.) Overlap de I/O optimizada.	Encadenamientos simples activados por contenidos de hileras de datos. Reestructuración de datos compleja	Extensión y desarrollo de conjunto widget . Voz simple de I/O, multimedia.
Muy Alto	Codificación reentrante y recursiva. Manejo de interrupciones con prioridades fijadas. Sincronización de tareas, retornos de llamadas complejas, proceso distribuido heterogéneo. Control en tiempo real de un único procesador	Análisis numérico difícil pero estructurado: ecuaciones de matrices, ecuaciones diferenciales parciales. Paralelización simple.	Rutinas para diagnóstico de interrupciones, dar servicio enmascaramiento. Manejo de línea de comunicación. Sistemas embebidos de rendimiento intensivo.	Coordinación de bases de datos distribuidas. Encadenamientos complejos. Optimización de la búsqueda.	Moderadamente complejo 2D/3D, gráficos dinámicos multimedia
Extra Alto	Múltiples recursos de planificación con cambio dinámico de prioridades. Control a nivel de microcódigo. Control distribuido en tiempo real.	Análisis numérico difícil y sin estructurar: análisis de ruido altamente aproximado, datos estocásticos. Paralelización compleja.	Dispositivo dependiente temporalmente de la codificación, operaciones microprogramadas.	Fuerte acoplamiento, estructuras de objetos y relacionales dinámicas. Gestión de datos del lenguaje natural.	Multimedia compleja, realidad virtual

Tabla 7.24. Medidas de complejidad del módulo versus Tipo de módulo



(RUSE). Reutilización Requerida

Este driver de coste explica el esfuerzo adicional necesario para construir componentes pensados para ser reutilizados en proyectos presentes ó futuros (tabla 7.25). Este esfuerzo se consume en crear un diseño más genérico del software, documentación más detallada y pruebas más extensas, para asegurar que los componentes están listos para utilizar en otras aplicaciones.

	Muy Bajo	Bajo	Nominal	Alto	Muy Alto	Extra Alto
RUSE		Nada	A lo largo del programa	A lo largo del proyecto	A lo largo de la línea de producto	A lo largo de las líneas de producto múltiples

Tabla 7.25. Niveles de medida RUSE

(DOCU). Documentación Asociada a las Necesidades del Ciclo de Vida

Varios modelos de coste software tienen un driver de coste para el nivel de documentación requerida. En COCOMO II la escala de medida para el driver de coste se evalúa en términos de la adecuación de la documentación del proyecto a las necesidades de su ciclo de vida (tabla 7.26). La escala de valores va desde Muy Bajo (muchas necesidades del ciclo de vida sin cubrir) hasta Muy Alto (excesiva para las necesidades del ciclo de vida).

	Muy Bajo	Bajo	Nominal	Alto	Muy Alto	Extra Alto
DOCU	Muchas necesidades del ciclo de vida sin cubrir	Algunas necesidades del ciclo de vida sin cubrir	Necesidades correctas al ciclo de vida	Excesivas necesidades para el ciclo de vida	Necesidades muy elevadas para el ciclo de vida	

Tabla 7.26. Niveles de medida DOCU

Factores de Plataforma

La plataforma se refiere a la complejidad del objetivo-máquina del hardware y la infraestructura software (anteriormente llamada máquina virtual). Los factores deben ser revisados para reflejar esto tal y como se describe en esta sección. Se consideraron algunos factores de la plataforma adicionales, tales como, distribución, paralelismo, rigidez y operaciones en tiempo real.



(TIME). Restricción del Tiempo de Ejecución

Esta es una medida de la restricción del tiempo de ejecución impuesta en un sistema software. Las medidas se expresan en términos de porcentaje de tiempo de ejecución disponible que se espera que sea usado por el subsistema ó sistema que consume el recurso de tiempo de ejecución. Los valores van desde nominal, menos del 50% de recursos de tiempo de ejecución utilizados, hasta Extra Alto, 95% del recurso de tiempo de ejecución consumido (tabla 7.27).

	Muy Bajo	Bajo	Nominal	Alto	Muy Alto	Extra Alto
TIME			≤ 50% del uso de tiempo de ejecución disponible	70%	85%	95%

Tabla 7.27. Niveles de medida TIME

(STOR). Restricción de Almacenamiento Principal

Esta medida representa el grado de restricción de almacenamiento principal impuesto a un sistema ó subsistema software. Dado el notable aumento en el tiempo de ejecución disponible del procesador y de almacenamiento principal, uno puede cuestionar si estas variables de restricción son todavía pertinentes. Los valores van desde nominal, menos que el 50%, a Extra Alto, 95% (tabla 7.28).

	Muy Bajo	Bajo	Nominal	Alto	Muy Alto	Extra Alto
STOR			≤ 50% del uso del almacenamiento disponible	70%	85%	95%

Tabla 7.28. Niveles de medida STOR

(PVOL). Volatilidad de la Plataforma

“Plataforma” se usa aquí para significar la complejidad del Hardware y Software (OS, DBMS, etc.) que el producto software necesita para realizar sus tareas. Si el software a desarrollar es un sistema operativo, entonces la plataforma es el hardware del ordenador. Si se desarrolla un Gestor de Base de Datos, entonces la plataforma es el hardware y el sistema operativo. Si se desarrolla un browser de texto de red, entonces la plataforma es la red, el hardware del ordenador, el sistema operativo y los repositorios de información distribuidos. La plataforma incluye cualquier compilador ó ensamblador que soporta el



desarrollo del sistema software. Los valores van desde Bajo donde cada 12 meses hay un cambio importante, hasta Muy Alto, donde hay un cambio importante cada 2 semanas (tabla 7.29).

Muy Bajo	Bajo	Nominal	Alto	Muy Alto	Extra Alto
PVOL	Mayor cambio cada 12 meses Menor cambio cada mes	Mayor: 6 meses Menor: 2 semanas	Mayor: 2 meses Menor: 1 semana	Mayor: 2 semanas Menor: 2 días	

Tabla 7.29. Niveles de medida PVOL

Factores de Personal

(ACAP). Habilidad del Analista

Los analistas son personal que trabaja en los requisitos de diseño de alto nivel y en diseño detallado. Los atributos principales que deben considerarse en esta medida son la habilidad de análisis y diseño, la eficiencia y minuciosidad y la habilidad para comunicar y cooperar. La medida no debe considerar el nivel de experiencia del analista, eso se mide con AEXP. Los analistas que caen en el 90avo percentil se evalúan como Muy Alto (tabla 7.30).

	Muy Bajo	Bajo	Nominal	Alto	Muy Alto	Extra Alto
ACAP	15º percentil	35º percentil	55º percentil	75º percentil	90º percentil	

Tabla 7.30. Niveles de medida ACAP

(PCAP). Habilidad del Programador

Las tendencias actuales continúan dando énfasis a la capacidad de los analistas. Sin embargo el creciente papel de los paquetes complejos COTS y la relevante influencia asociada a la capacidad de los programadores para tratar con esos paquetes COTS, indica una tendencia también a darle mayor importancia a la capacidad del programador.

La evaluación debe basarse en la capacidad de los programadores como un equipo, más que individualmente. La habilidad del programador no debe considerarse aquí, eso se mide con AEXP. Unos



valores Muy Bajos del equipo de programadores es el 15avo percentil y Muy Alto en el 90avo percentil (tabla 7.31).

	Muy Bajo	Bajo	Nominal	Alto	Muy Alto	Extra Alto
PCAP	15° percentil	35° percentil	55° percentil	75° percentil	90° percentil	

Tabla 7.31. Niveles de medida PCAP

(AEXP). Experiencia en las Aplicaciones

Esta medida depende del nivel de experiencia en aplicaciones del equipo de proyecto al desarrollar sistemas ó subsistemas software. Los valores se definen en términos de nivel de experiencia del equipo de proyecto en este tipo de aplicaciones. Un valor muy bajo para experiencia en aplicaciones es menor que 2 meses. Un valor muy alto es por experiencia de 6 años ó más (tabla 7.32).

	Muy Bajo	Bajo	Nominal	Alto	Muy Alto	Extra Alto
AEXP	≤ 2 meses	6 meses	1 año	3 años	6 años	

Tabla 7.32. Niveles de medida AEXP

(PEXP). Experiencia en la Plataforma

El modelo post-Arquitectura amplía la influencia en productividad de PEXP, reconociendo la importancia de entender el uso de plataformas más poderosas, incluyendo más interfaces gráficos de usuario, redes y capacidades de middleware distribuido (tabla 7.33).

	Muy Bajo	Bajo	Nominal	Alto	Muy Alto	Extra Alto
PEXP	≤ 2 meses	6 meses	1 año	3 años	6 años	



Tabla 7.33. Niveles de medida PEXP

(LTEX). Experiencia en la Herramienta y en el Lenguaje

Esta es una medida del nivel de experiencia en el lenguaje de programación y en la herramienta software del equipo de proyecto que desarrolla el sistema ó subsistema software. El desarrollo de software incluye el uso de herramientas que realizan representación de requisitos y diseño, análisis, gestión de la configuración, origen de los documentos, gestión de librería, estilo de programa y estructura, verificación de consistencia, etc...Además de la experiencia programando en un lenguaje específico, las herramientas que dan soporte también influyen en el tiempo de desarrollo. Tener una experiencia de menos de 2 meses dá un valor Bajo. Si es de 6 ó más años el valor es Muy Alto (tabla 7.34).

	Muy Bajo	Bajo	Nominal	Alto	Muy Alto	Extra Alto
LTEX	≤ 2 meses	6 meses	1 año	3 años	6 años	

Tabla 7.35. Niveles de medida LTEX

(PCON). Continuidad del Personal

La escala de valores de PCON se mide en términos del movimiento de personal del proyecto anualmente: desde 3%, Muy Ato, hasta el 48%, Muy Bajo (tabla 7.36).

	Muy Bajo	Bajo	Nominal	Alto	Muy Alto	Extra Alto
PCON	48% /año	24% /año	12% /año	6% /año	3% /año	

Tabla 7.36. Niveles de medida PCON

Factores del Proyecto

(TOOL). Uso de Herramientas Software



Las herramientas software han mejorado significativamente desde los proyectos de los 70 usados para calibrar COCOMO. Los valores para la herramienta van desde edición y código simple, Muy Bajo, hasta herramientas integradas de gestión del ciclo de vida, Muy Alto (tabla 7.37).

	Muy Bajo	Bajo	Nominal	Alto	Muy Alto	Extra Alto
TOOL	Editar, Código, Depuración	Simple, frontend, backend CASE, integración pequeña	Herramientas de ciclo de vida básicas	Fuerte, herramientas de ciclo de vida maduras, moderada-mente integrada	Fuerte, maduro, herramientas de ciclo de vida proactivas, bien integrado con los procesos, métodos, reutilización	

Tabla 7.37. Niveles de medida TIME

(SITE). Desarrollo Multilugar

Dada la frecuencia creciente de desarrollos multilugar e indicaciones de que los efectos del desarrollo multilugar son significantes. Se ha añadido en COCOMO II el driver de coste SITE. Determinar la medida del driver de coste incluye el cálculo y la medida de 2 factores: Localización del lugar (desde totalmente localizado hasta distribución internacional) y soporte de comunicación (desde correo de superficie y algún acceso telefónico hasta multimedia totalmente interactivo) (tabla 7.38).

	Muy Bajo	Bajo	Nominal	Alto	Muy Alto	Extra Alto
SITE: Localización	Internacional	Multi-ciudad y Multi-compañía	Multi-ciudad o Multi-compañía	Misma ciudad ó área metrop.	Mismo edificio ó complejo	Completamente localizado
SITE: Comunicaciones	Algo de teléfono, mail	Teléfono individual FAX	Banda estrecha, email	Comunicación electrónica de banda ancha	Comunicación electrónica de banda ancha, video conferencia ocasional	Multimedia interactiva

Tabla 7.38. Niveles de medida SITE

(SCED). Calendario de Desarrollo Requerido

Este valor mide las restricciones de horario impuestas al equipo de proyecto que desarrolla el software. Los valores se definen en términos de porcentaje de aceleración ó alargamiento sobre el calendario



respecto de un calendario nominal para un proyecto que requiere una cantidad de esfuerzo dada. Los calendarios acelerados tienden a producir más esfuerzo en las fases más tardías de desarrollo porque se dejan por solucionar más problemas debido a la falta de tiempo para resolverlos más pronto. Una compresión del calendario del 74% se evalúa como Muy Bajo. Un alargamiento del calendario produce más esfuerzo en las fases más tempranas del desarrollo, donde hay más tiempo para planificar, hacer especificaciones y validar minuciosamente. Un alargamiento del 160% se valora como Muy Alto (tabla 7.39).

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

Tabla 7.39. Niveles de medida TIME

La tabla 7.40 muestra un resumen de los valores correspondientes a los drivers de coste estudiados para el modelo Post-Arquitectura. La tabla 7.41 muestra el valor de los multiplicadores para los drivers correspondientes.



	Bajo	Nominal	Alto	Muy Alto	Extra Alto
RELY	Bajo, pérdidas fácilmente recuperables	Moderado, pérdidas recuperables	Alta pérdida financiera	Riesgo de vidas humanas	
DATA	DB bytes/ Pgm SLOC < 10	$10 \leq D/P < 100$	$100 \leq D/P < 1000$	$D/P \geq 1000$	
CPLX	Ver Tabla 7.24.				
RUSE	Nada	A lo largo del proyecto	A lo largo del programa	A lo largo de la línea de producto	A lo largo de las líneas de producto múltiples
DOCU	Algunas necesidades del ciclo de vida sin cubrir	Necesidades correctas al ciclo de vida	Excesivas necesidades para el ciclo de vida	Necesidades muy elevadas para el ciclo de vida	
TIME		$\leq 50\%$ del uso de tiempo de ejecución disponible	70%	85%	95%
STOR		$\leq 50\%$ del uso del almacenamiento disponible	70%	85%	95%
PVOL	Mayor cambio cada 12 meses Menor cambio cada mes	Mayor: 6 meses Menor: 2 semanas	Mayor: 2 meses Menor: 1 semana	Mayor: 2 semanas Menor: 2 días	
ACAP	35º percentil	55º percentil	75º percentil	90º percentil	
PCAP	35º percentil	55º percentil	75º percentil	90º percentil	
PCON	24% /año	12% /año	6% /año	3% /año	
AEXP	6 meses	1 año	3 años	6 años	
PEXP	6 meses	1 año	3 años	6 años	
LTEX	6 meses	1 año	3 años	6 años	
TOOL	Simple, frontend, backend CASE, integración pequeña	Herramientas de ciclo de vida básicas moderadamente integradas	Fuerte, herramientas de ciclo de vida maduras, moderadamente integrada	Fuerte, maduro, herramientas de ciclo de vida proactivas, bien integrado con los procesos, métodos, reutilización	
SITE: Localización	Multi-ciudad y Multi-compañía	Multi-ciudad o Multi-compañía	Misma ciudad ó área metrop.	Mismo edificio ó complejo	Completamente localizado
SITE: Comunicaciones	Teléfono individual, FAX	Banda estrecha, email	Comunicación electrónica de banda ancha	Comunicación electrónica de banda ancha, video conferencia ocasional	Multimedia interactiva
SCED	85%	100%	130%	160%	

Tabla 7.40. Resumen del nivel de medida de los drivers de coste Post-Arquitectura



	Muy Bajo	Bajo	Nominal	Alto	Muy Alto	Extra Alto
RELY	0.82	0.92	1.00	1.10	1.26	--
DATA	--	0.90	1.00	1.14	1.28	--
CPLX	0.73	0.87	1.00	1.17	1.34	1.74
RUSE	--	0.95	1.00	1.07	1.15	1.24
DOCU	0.81	0.91	1.00	1.11	1.23	--
TIME	--	-	1.00	1.11	1.29	1.63
STOR	--	--	1.00	1.05	1.17	1.46
PVOL	--	0.87	1.00	1.15	1.30	--
ACAP	1.42	1.19	1.00	0.85	0.71	--
PCAP	1.34	1.15	1.00	0.88	0.76	--
PCON	1.29	1.12	1.00	0.90	0.81	--
AEXP	1.22	1.10	1.00	0.88	0.81	--
PEXP	1.19	1.09	1.00	0.91	0.85	--
LTEX	1.20	1.09	1.00	0.91	0.84	--
TOOL	1.17	1.09	1.00	0.90	0.78	--
SITE	1.22	1.09	1.00	0.93	0.86	0.80
SCED	1.43	1.14	1.00	1.00	1.00	--

Tabla 7.41. Multiplicadores de esfuerzo actualizados para el modelo Post-Arquitectura pertenecientes a la versión USC-COCOMOII.1999.0



Por ejemplo, si al analizar los drivers de coste para nuestro proyecto obtenemos los siguientes resultados:

	MB	B	N	A	MA	EA
RELY			x			
DATA				x		
CPLX			x			
RUSE			x			
DOCU			x			
TIME				x		
STOR			x			
PVOL			x			
ACAP		x				
PCAP			x			
PCON			x			
AEXP			x			
PEXP			x			
LTEX				x		
TOOL				x		
SITE				x		
SCED			x			

EB=Extra Bajo
 MB=Muy Bajo
 B=Bajo
 N=Nominal
 A=Alto
 MA=Muy Alto
 EA=Extra Alto

17 Aplicando los valores de la tabla 7.41 obtendremos:

$$\prod_{i=1}^{17} EM_i = 1 \times 1.14 \times 1 \times 1 \times 1 \times 1 \times 1.05 \times 1 \times 1.19 \times 1 \times 1 \times 1 \times 1 \times 1 \times 0.91 \times 0.90 \times 1.09 \times 0.93 \times 1 = 1.085$$

i=1

Nota: En los drivers de coste como por ejemplo CPLX o SITE se evalúan varias características, en este último caso caso Comunicaciones y Localización (ver tabla 7.38). El resultado Alto se ha obtenido de hacer la media subjetiva entre Comunicaciones= Alto y Localización = Muy Alto. Esta media es Alto porque como hemos visto anteriormente siempre que una evaluación de un driver de coste está entre dos niveles de ratio, hay que redondear al nivel nominal. Si la evaluación hubiese estado entre Extra Bajo y Bajo la media subjetiva hubiese sido Muy Bajo.



7.4.5. VALORES DE TIEMPO DE DESARROLLO.

La versión inicial de COCOMO II proporciona una capacidad de estimación de tiempo simplemente similar a las de COCOMO. La ecuación siguiente es la ecuación inicial de tiempos base para las tres etapas de COCOMO II es:

$$TDEV = [3.67 \times PM^{(0.28+0.2X(B-1.01))}] \times \frac{SCED\%}{100}$$

Donde TDEV es el tiempo en meses desde la determinación de una línea base de requisitos del producto hasta que se completa una actividad de aceptación que certifica que el producto satisface los requisitos. PM, es la estimación de meses-persona, excluyendo el estimador de esfuerzo SCED, B, es la suma de los factores de escala del proyecto y SCED % es el porcentaje de compresión/expansión en el multiplicador de esfuerzo SCED, (ver tabla 7.39).

A medida que COCOMO II evoluciona tendremos un modelo de estimación del tiempo más extenso que refleja las diferentes clases de modelo de proceso que puede usar un proyecto; los efectos de software COTS y reutilizable, y los efectos de las capacidades de composición de las aplicaciones.

Por ejemplo, si tomamos un proyecto en el que hemos obtenido un resultado de 26 MM, B=0.75 y no existen restricciones de tiempo, al aplicar la ecuación anterior obtenemos:

$$TDEV = [3.67 \times 26^{(0.28+0.2 \times (0.75-1.01))}]$$

RANGOS DE SALIDA

Varios usuarios de COCOMO han expresado una preferencia por los rangos de estimación en lugar de cálculo de puntos como salidas de COCOMO. Las tres etapas del modelo COCOMO II permiten la estimación de rangos probables de valores de salida usando distintas relaciones de exactitud de coste y medida. Una vez que se calcula el valor de esfuerzo más probable, E, del modelo elegido: Composición de Aplicaciones, Diseño anticipado o Post-Arquitectura, se calculan un conjunto de estimaciones optimistas y pesimistas que representan una desviación estándar alrededor del valor más probable, de la siguiente forma:



	Estimación Optimista	Estimación Pesimista
Composición de Aplicaciones	0.50E	2.0 E
Diseño Anticipado	0.67 E	1.5 E
Post-Arquitectura	0.80 E	1.25 E

Tabla 7.42. Rango de salida estimado

El rango de valores de esfuerzo puede utilizarse en la ecuación de tiempo para determinar un rango de valores de tiempo.

Basándonos en el ejemplo anterior en el que calculamos el tiempo de desarrollo, si queremos estimar los rangos probables de valores de salida:

$MM = [0.80 (26) + 1.25(26)]$ y los rangos de TDEV se obtendrán sustituyendo ambos valores en la ecuación de tiempo correspondiente.

Veamos un ejemplo añadiendo una restricción de tiempo:

Supongamos que en un proyecto hemos aplicado la fórmula del cálculo del tiempo (con SCED=1 por ser la primera estimación) y hemos obtenido que el tiempo estimado de duración del proyecto es de 4 meses. Se nos ha impuesto una restricción de tiempo, de forma que debemos ajustarnos a 3 meses que es el tiempo máximo que tenemos. Esto representa un 25% menos de tiempo del que obtenemos en la primera estimación, es decir un 75% del nominal, lo que se corresponde con un valor de SCED= Muy Bajo según la tabla 7.47. Ahora tenemos que volver a aplicar la fórmula del esfuerzo ajustado, modificando el valor de SCED que sería 1.43. (tabla 7.41) para obtener un nuevo valor del esfuerzo, que será mayor que el inicial.

Una vez estimado el esfuerzo y la duración total de un proyecto, se puede calcular el esfuerzo y duración distribuido por fases y subfases del proyecto con las tablas del modelo COCOMO 81, el cual supone un modelo de proceso en cascada (secuencial y progresivo). Si el modelo de proceso no es en cascada, no pueden aplicarse estas distribuciones de fases.

7.4.6. TRATAMIENTO DE LA REUTILIZACIÓN EN LA ESTIMACIÓN



En los modelos de Diseño Anticipado y Post-Arquitectura se pueden incluir consideraciones especiales cuando se prevee reutilización del código que compondrá la aplicación que estamos estimando.

La inclusión de características de reutilización conlleva describir el parámetro Size como sigue:

$$\text{Size} = \text{KSLOC} + \text{KASLOC} \times \left[\frac{100 - \text{AT}}{100} \right] \times (\text{AAM})$$

Dónde la variable KSLOC ha sido explicada anteriormente, y representa el número de líneas de código a desarrollar desde cero; la variable KASLOC representa miles de líneas de código fuente adaptadas; el valor AT representa el porcentaje de traducción automatizada y por último la variable AAM representa un Multiplicador de Ajuste para la Adaptación:

$$\text{AAM [ESLOC]} = \frac{\text{ASLOC [AA+AAF(1+0.02(SU)(UNFM))]}}{100} \quad \text{AAF} \leq 0.5$$

$$\text{AAM [ESLOC]} = \frac{\text{ASLOC [AA+AAF+(SU)(UNFM)]}}{100} \quad \text{AAF} > 0.5$$

$$\text{AAF} = 0.4(\text{DM}) + 0.3(\text{CM}) + 0.3(\text{IM})$$

El tratamiento que hace COCOMO II del software reutilizado utiliza un modelo de estimación no lineal. Esto implica que hay que estimar la cantidad de software que se va a adaptar, ASLOC y tres parámetros de grado de modificación: El porcentaje de diseño modificado (DM), el porcentaje de código modificado (CM) y el porcentaje de esfuerzo inicial de integración requerido para la integración del software reutilizado (IM).

El cálculo de ESLOC se basa en una cantidad intermedia, el Factor de Ajuste de Adaptación (AAF). Las cantidades de adaptación DM, CM, IM se usan para calcular AAF, donde:

DM : Porcentaje de diseño modificado. El porcentaje de diseño de software que es modificado para adaptarlo a los nuevos objetivos y al entorno. (Esto es necesariamente una cantidad subjetiva).

CM : Porcentaje de código modificado. El porcentaje de código software adaptado que es modificado para adaptarlo a los nuevos objetivos y al entorno.

IM : Porcentaje de integración requerida para software modificado. El porcentaje de esfuerzo requerido para integrar el software adaptado dentro de la totalidad del producto y comprobar el



producto resultante comparado con la cantidad de esfuerzo normal de integración y pruebas para software de un tamaño similar.

Si no hay DM ó CM (el componente se usa sin modificaciones) entonces no es necesario SU. Se aplica SU si el código es modificado.

El incremento de comprensión del software (SU) se obtiene de la tabla 7.43. SU se expresa cuantitativamente como un porcentaje. Si el software se tasa muy alto en estructura, claridad y descriptividad del mismo, la comprensión del software y la penalización por comprobar el interfaz es del 10%. Si el software se tasa muy bajo en estos factores, es del 50%. SU se determina tomando el promedio subjetivo de las tres categorías.



	Muy Bajo	Bajo	Normal	Alto	Muy Alto
Estructura	Muy baja cohesión alta , código "spaghetti"	Cohesión moderadamente baja, Alto acoplamiento	Razonadamente bien estructurado; algunas áreas débiles	Alta cohesión, Bajo acoplamiento	Fuerte modularidad, información oculta en estructuras de datos/control
Claridad de Aplicación	No ajuste entre programa y aplicación	Alguna correlación entre programa y aplicación	Correlación moderada entre programa y aplicación	Buena correlación entre programa y aplicación	Claro ajuste entre programa y aplicación vistas-globales
Descriptividad propia	Código confuso; documentación perdida, confusa u obsoleta	Algún comentario de código y cabeceras; alguna documentación útil	Nivel moderado en comentarios de código, cabeceras, documentaciones	Buen comentario en código y cabeceras ; documentación útil; algunas áreas débiles	Código descriptivo por si mismo; documentación actualizada; bien-organizada, con diseño racional
Incremento de SU a ESLOC	50	40	30	20	10

Tabla 7.43. Escala de valoración para el incremento de compresión del software (SU)

El otro incremento de reutilización no lineal trata con grados de valoración y asimilación (AA) necesarios para determinar si un módulo software completamente reutilizado es apropiado para la aplicación e integrar una descripción dentro de la del producto completo. La tabla 7.44. proporciona la escala de medida y los valores para el incremento de valoración y asimilación. AA es un porcentaje.

Incremento AA	Nivel de Esfuerzo AA
0	Ninguno
2	Módulo Básico de búsqueda y documentación.
4	Algún módulo de Test y Evaluación (T&E), documentación.
6	Módulo considerable de T&E, documentación.
8	Módulo Extensivo de T&E, documentación

Tabla 7.44. Escala de valoración para el incremento de la valoración y asimilación (AA)



La cantidad de esfuerzo necesario para modificar el software existente es una función no sólo de la cantidad de modificación (AAF) y la comprensibilidad del software existente (SU) sino también de lo relativamente desconocido que es el software para el programador (UNFM). El parámetro UNFM se aplica multiplicativamente al incremento de esfuerzo en comprensión del software. Si el programador trabaja con el Software todos los días, el multiplicador 0.0 no añadirá incremento de comprensión del software. Si el programador no ha visto nunca antes el software, el multiplicador 1.0 añadirá el mayor incremento de esfuerzo de comprensión del software. Los valores de UNFM están en la tabla 7.45.

Incremento UNFM	Nivel de Desconocimiento
0.0	Completamente Conocido
0.2	Bastante Conocido
0.4	Algo Conocido
0.6	Considerablemente Desconocido
0.8	Bastante Desconocido
1.0	Completamente Desconocido

Tabla 7.45. Escala de valoración para el Desconocimiento del Programador

Supongamos que en el desarrollo de una aplicación se va a reutilizar un programa gestor de nóminas previo con ASLOC= 800. El cambio en el diseño de este programa va a afectar a unos pocos módulos ($DM = 10\%$), y el porcentaje de código modificado va a ser del 10 % ($CM = 10\%$). Si la integración de una cantidad de software similar en el desarrollo inicial supuso un esfuerzo de 0.3 PM y ahora ha sido de 0.35 MM, el esfuerzo se ha incrementado en 0.05 MM y eso es el 16% del esfuerzo inicial (0.3 MM). Por lo que el porcentaje de modificación del esfuerzo inicial de integración requerido para la integración de software reutilizado es de $IM = 16\%$ (0.16). Por lo tanto:

$$AAF = (0.4 \times 0.1) + (0.3 \times 0.1) + (0.3 \times 0.16) = 0.04 + 0.03 + 0.048 @ 0.118$$

El componente (programa gestor de nóminas) ha de ser modificado para poder aplicarlo al nuevo desarrollo, por eso se aplica SU (ver tabla 7.43):

Estructura. No hay que modificar demasiados módulos debido a que está razonablemente bien estructurado: Normal.

Claridad de aplicación. Buena correlación entre programa y aplicación: Alta.

Descriptividad propia. Bien documentados el código y las cabeceras, pero escasa información sobre la relación entre submódulos: Alta.

$$SU = (30 + 20 + 20) / 2 = 35$$

Se ha elegido una persona especialista en el lenguaje de programación en el que está desarrollado el componente gestor de nóminas y que ya lo conoce muy bien. Por lo que $UNFM = 0.0$



$AA = 0$ (ninguno, ver tabla 7.44.)

Al no haber traducción automática de código, $AT = 0$

Como $AAF < 0.5$ aplicamos la expresión:

$$AAM(ESLOC) = 800 \times [0 + 0.118 (1 + 0.02 (0.35)(0.0))] / 100 = 894,4$$

Vamos a suponer que $KSLOC = 7.360$ y que el programa gestor de nóminas tiene 800 Líneas de código fuente, $KASLOC = 0.8$) :

$$\text{Size} = 7.360 + 0.8 \times [(100 - 0) / 100] \times 894.4 = 8075.52$$

7.5. MANTENIMIENTO DE LAS APLICACIONES

COCOMO II utiliza el Modelo de reutilización cuando la cantidad de código fuente base añadido ó modificado es menor ó igual que el 20% del nuevo código que se está desarrollando. Código base es código fuente que ya existe y que se cambia para usarlo en un proyecto actual. Para el mantenimiento de proyectos que incluyen más del 20% de cambio en el código base existente (relativo al nuevo código que se está desarrollando) COCOMO II utiliza el Modelo de mantenimiento, donde el tamaño (Size_M), se obtiene de forma relativamente distinta, con lo que la fórmula de estimación del esfuerzo de mantenimiento resultante es la siguiente.

$$MM_{\text{Mantenimiento}} = A \times (\text{Size}_M)^B \times \prod EM_i$$

Un tamaño de mantenimiento inicial se puede obtener de dos formas:

1. Si el tamaño de código base se conoce y el porcentaje de cambio también, entonces:

$$(\text{Size})_M = [(\text{BaseCodeSize}) \times \text{MCF}] \times \text{MAF}$$

$$\text{MCF} = \frac{\text{SizeAdded} + \text{SizeModified}}{\text{BaseCodeSize}}$$

$$\text{MAF} = 1 + \left[\frac{\text{SU}}{100} \times \text{UNFM} \right]$$



El porcentaje de código base modificado se llama Factor de Cambio (MCF). El MCF es similar a la tasa de cambio anual de COCOMO 81 sólo que pueden usarse periodos de mantenimiento superior a un año.

2. Si se conoce la fracción de código añadido ó modificado del código base durante el periodo de mantenimiento se utiliza la ecuación siguiente donde:

$$(\text{Size})_M = (\text{SizeAdded} + \text{SizeModified}) \times \text{MAF}$$

$$\text{MAF} = 1 + \left[\frac{\text{SU}}{100} \times \text{UNFM} \right]$$

El tamaño puede referirse a miles de líneas de código fuente (KSLOC), Puntos de Función ó Puntos Objeto. Cuando se usan Puntos de Función ó Puntos Objeto es mejor estimar MCF en términos de fracción de la aplicación global que está siendo modificada, en lugar de la fracción de entradas, salidas, pantallas, informes, etc... afectados por los cambios. La experiencia ha demostrado que contar los items afectados puede llevar a estimaciones importantes por encima, así como cambios relativamente pequeños pueden afectar a un relativo gran número de items.

El cálculo del tamaño de mantenimiento inicial (descrito anteriormente) se ajusta con el Factor de Ajuste del Mantenimiento (MAF). COCOMO 81 utiliza diferentes multiplicadores para los efectos de la Fiabilidad Requerida (RELY) y para las Prácticas de Programación Modernas (MODP) en el mantenimiento, en contra del esfuerzo de desarrollo. COCOMO II, en cambio, utilizó los factores de Comprensión del Software (SU) y de Desconocimiento del Programador (UNFM) de su Modelo de Reutilización para modelar los efectos del software comprensiblemente suficiente ó escasamente estructurado por el esfuerzo de mantenimiento.

Vamos a ver el caso del mantenimiento de un proyecto cuyo tamaño es de 8000 SLOC. Van a mantenerse 4000 SLOC del proyecto inicial y se van añadir 2500 SLOC nuevas .

Calculando SU (ver tabla 7.43.):

Estructura: Se han utilizado funciones independientes que son llamadas desde distintos módulos y que a la vez se comunican entre sí: Alta.

Claridad de aplicación: Claro ajuste entre programa y aplicación: MuyAlta.

Descriptividad propia: Bien documentado el código y las cabeceras: Alto.

$$SU = (20 + 10 + 20) / 2 = 25$$



Se ha elegido a una persona que últimamente ha trabajado bastante con el lenguaje en el que se ha realizado la aplicación. $UNFM = 0.2$.

Así tenemos que:

$$MAF = 1 + [(25/100) \times 0.2] = 1.05$$

$$MCF = (2500 + 4000) / 8000 = 0.8125$$

$$(Size)_M = [(8000) \times 0.8125] \times 1.05 = 6825 \text{ SLOC} = 6.825 \text{ KSLOC}$$
