TEMA 2 METRICAS Y MODELOS DE ESTIMACION DEL SOFTWARE

2.1 *Introducción a las métricas del software*

El término "Métricas del Software" comprende muchas actividades, todas ellas relacionadas de alguna manera con la idea de mejorar la calidad del software. Cada una de esas actividades ha evolucionado por sus propios medios dentro de un dominio más amplio que las abarca a todas: la Ingeniería del Software [Fenton - Pfleeger, 1996]

Entre estas actividades se pueden incluir las siguientes:

- Medidas y modelos de estimación de costes y esfuerzo
- Medidas y modelos de productividad
- Recogida de datos
- Medidas y modelos de calidad
- Modelos de fiabilidad
- Modelos y evaluación del rendimiento
- Métricas de la estructura y la complejidad
- Modelos de madurez
- Gestión a través de las métricas
- Evaluación de métodos y herramientas

Nuestro objetivo es revisar las métricas asociadas a estas actividades presentándolas, en la medida de lo posible, en dos grandes grupos: métricas basadas en el código y métricas basadas en el diseño y en las especificaciones. Asimismo, y por ser especialmente relevante para los casos de estudio de esta tesis, se hace mención a una serie de métricas específicas para programas COBOL. Y para finalizar con las métricas más actuales, se estudian algunas de las más relevantes especialmente diseñadas para programación orientada a objetos.

Por último, para completar el estudio del estado actual de las métricas del software y del control de calidad, se presentan en este capítulo los marcos más importantes de medida, así como los principales estándares asociados.

2.1.1 *Métricas basadas en el código*

Los primeros intentos que se realizaron para medir la calidad del software condujeron al desarrollo de una serie de métricas basadas casi exclusivamente en el código de los programas. De esta primera época destacaremos las Líneas de Código, la Ecuación de Putnam, Software Science, la Complejidad Ciclomática, las Métricas Híbridas y los Modelos Cocomo. Estas métricas de la primera etapa se estudian a continuación.

2.1.1.1 Líneas de Código

Aunque el tamaño de una aplicación software se puede medir utilizando unidades de medida muy diversas (número de módulos de los programas, número de páginas de los listados del código "fuente", número de rutinas, etc.), el código de los programas ha sido, originalmente, la principal fuente de medida del software y casi todas las métricas de esta primera etapa de intentos de medir el software se basan exclusivamente en el código. Así, entre las primeras métricas que se utilizaron para predecir la fiabilidad y la complejidad de las aplicaciones se encuentran las líneas de código (LOC: Lines Of Code) Está métrica, por su simplicidad y por la dificultad que representa definir qué es una línea de código, ha sido criticada severamente por diferentes autores [McCabe 1976, DeMarco 1982] En efecto, al intentar usar LOC como medida, surge de inmediato la duda sobre qué es lo que se debe de considerar como una línea de código. Es necesario decidir, por ejemplo, si una línea de código es la línea escrita en un lenguaje de alto nivel, o por el contrario es una línea de código máquina, ya que hay que tener en cuenta que una línea de código escrita en un lenguaje de alto nivel se puede convertir en múltiples líneas de código máquina. Si consideramos que el código máquina es el que ejecuta el ordenador se podría

argüir que éstas son las LOC a considerar. Sin embargo, lo que escribe el programador son las líneas en el lenguaje de alto nivel, y no tiene por que enfrentarse a la dificultad del código máquina.

En 1981, Boehm propone [Boehm, 1981] el uso de líneas de código "fuente" expresadas en miles (KLOC: Kilo-Lines Of Code) y en 1983, Basili y Hutchens [Basili - Hutchens, 1983] sugieren que LOC debe de considerarse como una métrica de base, contra la que se deben de comparar las demás métricas, por lo que sería de esperar que cualquier métrica efectiva se comportase mejor que LOC, y que, en cualquier caso, LOC ofreciese una "hipótesis nula" para evaluaciones empíricas de las diferentes métricas del software.

No obstante, para que la aplicación de las métricas basadas en el código sea efectiva, es necesario definir claramente el concepto de línea de código. Según Fenton y Pfleeger, la definición más extendida es la que establece Hewlett-Packard: "cualquier sentencia del programa, excluyendo las líneas de comentarios y las líneas en blanco" [Fenton - Pfleeger, 1996] Esta definición permite contar las líneas de código de una forma fácil y de una manera uniforme, independientemente del lenguaje de programación empleado.

2.1.1.2 La Densidad de Defectos

La métrica más comúnmente utilizada para medir la calidad del software es la densidad de defectos, que se expresa como:

$$\frac{\text{Número de defectos descubiertos}}{\text{Tamaño del código}}$$

Donde el tamaño normalmente se expresa en miles de líneas de código.

Aunque esta métrica correctamente utilizada puede constituir un indicador útil de la calidad del software, no puede considerarse como una medida de la calidad

en un sentido estricto. De hecho, existen una serie de problemas conocidos y bien documentados sobre esta métrica, destacando en particular los siguientes:

- Es más bien un indicador de la severidad de las pruebas que de la calidad del software

- No existe un consenso generalizado de lo que es un defecto. Puede considerarse como defecto un fallo del software descubierto durante las pruebas (que potencialmente puede convertirse en un fallo en tiempo de operación) o un fallo descubierto durante la ejecución de la aplicación. En algunos casos se considera un defecto el encontrado después de que la aplicación ha superado las pruebas y está siendo utilizada por el usuario final, en otros casos un defecto se refiere a cualquier tipo de fallo conocido, en otros a un error descubierto en una fase en concreto del ciclo de vida del software, etc. La terminología varía mucho dependiendo del tipo de organización que la emplee; normalmente los términos tasa de fallos, densidad de fallos, ratio de errores se utilizan indistintamente.

- El tamaño se utiliza solo como una medida subrogada del tiempo. Por ejemplo, para fallos en tiempo de operación, la ratio de errores se debería de basar en el tiempo medio entre fallos, proporcionando una medida precisa de la fiabilidad del software, que es la interesante desde el punto de vista del usuario final.

- No existe un consenso sobre cómo medir el software de un modo consistente y comparable. Aún empleando Líneas de Código (o kilo-líneas de código) para el mismo lenguaje de programación, las desviaciones en las reglas empleadas para contar pueden originar variaciones en un factor de 1 a 5.

A pesar de estos problemas conocidos, la densidad de defectos se ha convertido en un estándar "de facto" en las empresas a la hora de medir la calidad del software, aunque, por razones obvias, no suelen publicarse los resultados, aunque la densidad de defectos sea relativamente baja.

Uno de los informes más reveladores [Daskalantonakis, 1992] afirma que el objetivo de calidad Sigma Seis de Motorola, es tener "no más de 3.4 defectos por millón de unidades de salida en un proyecto". Esto implica una densidad de defectos excepcionalmente baja de 0,0034 defectos por cada mil líneas de código. El informe parece sugerir que, en 1990, la densidad de defectos media se encontraba entre 1 y 6 defectos por cada mil líneas, con una fuerte tendencia a disminuir.

2.1.1.3 La Ecuación de Putnam

Otra de las métricas basadas en el código fue propuesta por Putnam, quien por medio de un estudio de datos, desarrolla un modelo para la estimación del tamaño y el esfuerzo necesario para el desarrollo de productos software [Putnam, 1978] Putnam define una ecuación para estimar el tamaño (T) del software, expresado en líneas de código:

$$T = C * K^{1/3} * t_d^{4/3}$$

Donde C es una constante que mide el nivel tecnológico de la empresa, K representa el esfuerzo total de desarrollo, expresado en personas / año, incluyendo el esfuerzo debido al mantenimiento, y t_d es el tiempo de desarrollo expresado en años.

2.1.1.4 Software Science

Otra de las métricas de este período y quizá la más conocida, junto con la Complejidad Ciclomática de McCabe, es Software Science, postulada por Maurice Halstead. Software Science es una métrica basada en el código que intenta definir un

conjunto de leyes que Halstead afirma que son análogas a las leyes naturales, y que está relacionada con la termodinámica, la psicología cognitiva, la teoría de la información y la compilación inversa [Halstead, 1977]

Propone Halstead un conjunto de métricas que se obtienen a partir de diversas medidas, tales como el número de operadores distintos que aparecen en el programa, el número de apariciones de dichos operadores, el número de operandos y el número de apariciones de esos operandos, obteniendo como resultado la longitud del programa, su vocabulario, su volumen, su nivel, su dificultad (inverso del nivel) y el esfuerzo requerido para desarrollarlo. Todas las medidas son efectuadas sobre el código de los programas, bien porque éste ya exista o bien porque se haya estimado.

Así, un programa P estaría formado por un conjunto de "testigos", clasificados bien como operadores, bien como operandos. Si tenemos que:

n_1 = Número de operadores únicos
n_2 = Número de operandos únicos
N_1 = Número de total veces que aparecen los operadores
N_2 = Número de total veces que aparecen los operandos

Entonces, la longitud de P vendría dada por:

$$N = N_1 + N_2$$

Y el vocabulario de P por:

$$n = n1 + n2$$

De acuerdo con la teoría de Halstead, se podría estimar N como $n_1(\log n_1) + n_2(\log n_2)$ y el esfuerzo necesario para generar P sería:

$$E = n_1 N_2 N \log n / 2n_2$$

Igualmente, el tiempo necesario para desarrollar P se calcularía como:

$$T = E / 18 \text{ segundos}$$

Diversos autores realizaron investigaciones empíricas sobre Software Science, entre otros Ottenstein [Ottenstein, 1976] quién informó de la aplicación con éxito de Software Science al problema de detectar el plagio entre los estudiantes. Funami y el propio Halstead aplican la métrica para determinar la incidencia de los errores del software [Funami - Halstead, 1976] y Love y Bowman estudian la calidad del software [Love - Bowman, 1976]

Por desgracia, trabajos posteriores no muestran resultados tan favorables, y es de nuevo Love, en colaboración con Fitzsimmons [Fitzsimmons - Love, 1978], cuando al intentar aplicar la métrica a dos conjuntos de datos experimentales, publicados por separado por Gould [Gould, 1975] y por Weissman [Weissman, 1974], descubrieron que las correlaciones encontradas no fueron significativas en ambos casos. Otro estudio debido a Bowen [Bowen, 1978] encontró una modesta correlación entre la métrica y el número de errores encontrado en 75 módulos. Y, lo que es más preocupante, encontró que Software Science se comportaba peor que la tan criticada LOC. Otro estudio más amplio sobre 400 módulos, llevado a cabo por Basili y Philips, asimismo concluye que Software Science no predice el tiempo de desarrollo de modo significativamente mejor que LOC [Basili - Philips, 1981] Además, a raíz de los nuevos descubrimientos, los estudios iniciales favorables a la métrica han sido cuestionados tanto en el campo estadístico como en el experimental.

No obstante, Software Science ha despertado un gran interés por ser el primer intento de dar un marco coherente dentro del cual se pueda medir el software. Posiblemente, el legado más importante de Software Science sea el proporcionar este marco contra el cual medir y hacer interpretaciones.

2.1.1.5 Complejidad Ciclomática

Otra métrica de esta primera etapa es presentada por McCabe como alternativa a Software Science [McCabe, 1976] McCabe postula su Complejidad Ciclomática, v(G), con dos objetivos fundamentales: predecir el esfuerzo necesario para probar el software y para descubrir el modo de hacer particiones apropiadas del mismo y en segundo lugar predecir la complejidad de los módulos resultantes de la partición.

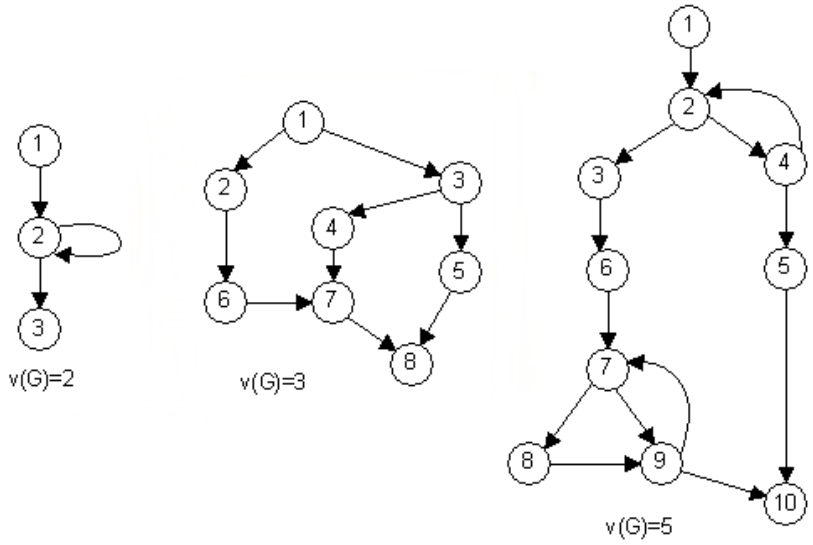


Fig. A.1 Grafos de Complejidad Ciclomática

McCabe representa el software por medio de grafos dirigidos, cuyas aristas representan el flujo de control y cuyos nodos son las instrucciones. La hipótesis de McCabe es que la complejidad del software está relacionada con la complejidad del flujo de control, y que, por tanto, cuantos más bucles, selecciones y bifurcaciones tenga el software más complejo será de desarrollar y comprender. De un modo práctico, v(G) se puede calcular fácilmente contando el número de bucles del grafo y añadiendo uno, lo que evita la posibilidad de tener una complejidad cero.

Como aplicación práctica de la métrica sugiere McCabe un valor de v(G)=10 como límite superior de la complejidad de un módulo, aunque admite que en determinados casos (por ejemplo en grandes estructuras CASE) este límite se podría sobrepasar.

Al igual que Software Science, la idea de McCabe provoca un gran interés y causa que se lleven a cabo investigaciones empíricas de la métrica, tal vez por su simplicidad de cálculo. Los resultados obtenidos fueron muy dispares. Henry y Kafura encontraron una correlación significativa entre v(G) y los cambios realizados en el software cuando estudiaban el sistema operativo Unix [Henry – Kafura, 1981] Sin embargo, la correlación descubierta por Bowen entre la métrica y los errores del software es tan poco significativa (r = -0.09) que no merece ser tenida en cuenta [Bowen, 1978] Basili y Perricone estudian la correlación con la densidad de los errores [Basili - Perricone, 1984]; Gafney revisa su correlación con el esfuerzo necesario de programación [Gafney, 1979]; Kitchenham lo hace con el número absoluto de errores [Kitchenham, 1981], etc., haciendo de la Complejidad Ciclomática una de las métricas más ampliamente validadas.

Además, se realizaron diferentes estudios empíricos para relacionar la métrica con la tendencia al error, con la capacidad de mantener y entender el software y con el esfuerzo de desarrollo, obteniéndose conclusiones muy dispares. La más desconcertante fue la alta correlación observada entre v(G) y LOC, y el mejor rendimiento de LOC en un número significativo de ocasiones [Basili - Hutchens, 1983], [Curtis, 1979], [Kitchenham, 1981], [Paige 1980], [Wang - Dunsmore 1984]

2.1.1.6 Métricas Híbridas

Las Métricas Híbridas surgen como combinación de los mejores aspectos de las métricas existentes. Por ejemplo, la combinación de Software Science con una extensión de la Complejidad Ciclomática basada en el nivel de anidamiento del software, debida a Harrison y Magel, que consideran que una métrica por si sola no es suficiente [Harrison - Magel, 1981] y que los resultados obtenidos por la

combinación de las dos métricas son "intuitivamente más satisfactorios". Sin embargo, no ofrecen una posterior validación de estas observaciones. De un modo similar, Oviedo combina el flujo de control con el flujo de datos en una métrica única para medir la complejidad del software [Oviedo, 1980] En cualquier caso, estas métricas híbridas aún necesitan una validación empírica, que hasta ahora no se ha llevado a cabo.

2.1.1.7 Los Modelos Cocomo

Los modelos Cocomo (COnstructive COst MOdel), de Barry W. Boehm, son una de las métricas basadas en líneas de código más ampliamente difundidas y estudiadas. Estos modelos, utilizados para medir el esfuerzo y el tiempo de desarrollo de las aplicaciones informáticas, fueron postulados por Boehm [Boehm, 1981] tras el estudio detallado de 63 proyectos incluidos en tres categorías: proyectos de negocios, proyectos científicos y proyectos de sistemas. Entre ellos había proyectos de diferente complejidad así como proyectos que habían sido desarrollados para rodar en entornos diferentes, desde Micros a Grandes Sistemas. También escogió Boehm proyectos escritos en distintos lenguajes, contemplando Cobol, Pascal, Jovial y Ensamblador. Presenta Boehm tres modelos diferentes: El Cocomo Básico, el Cocomo Intermedio y el Cocomo Detallado.

De estos modelos, el Cocomo Básico, permite dar una primera estimación durante la fase de Análisis Previo, cuando la mayoría de los factores que incidirán en el Proyecto son todavía desconocidos. Para ello, este modelo se apoya en una ecuación para estimar el número de Meses/Hombre (MH) necesarios para desarrollar un proyecto software, basándose en el número de miles de instrucciones "fuente" (KINST) que tendrá el producto software terminado:

$$MH = 2.4(KINST)^{1.05}$$

También presenta el modelo una ecuación para estimar el tiempo de desarrollo (TDES) de un proyecto, expresado en meses:

$$TDES = 2.5(MH)^{0.38}$$

Estas ecuaciones, así como las demás ecuaciones presentadas por Boehm, están obtenidas heurísticamente y son fruto de la experimentación con una gran variedad de proyectos; no obstante es posible establecer principios matemáticos de base a partir de distribuciones estadísticas, del tipo de la distribución de Rayleigh, para justificarlas.

El segundo modelo de Boehm, el Cocomo Intermedio, constituye una mejora sobre el modelo Básico, ya que tiene en cuenta otros aspectos que intervienen en la estimación del esfuerzo, como pueden ser la fiabilidad necesaria, el tamaño de las bases de datos, el riesgo que se corre frente a un fallo, etc. Es por tanto un modelo más fiable en cuanto a las estimaciones que presenta, posibilitando su utilización en el resto de fases del Proyecto. Se aconseja emplear este modelo cuando ya se ha avanzado en el desarrollo del proyecto, y por tanto se tiene más información sobre el mismo. Normalmente en la Fase de Análisis Funcional pueden ya realizarse estimaciones con este modelo.

La ecuación para el cálculo de Meses/Hombre de Cocomo Intermedio cambia su coeficiente para adaptar las estimaciones, aunque manteniendo siempre la misma estructura, y la ecuación para el tiempo de desarrollo permanece igual que en el modelo Básico:

$$MH = 3.2(KINST)^{1.05}$$
$$TDES = 2.5(MH)^{0.38}$$

Además del cambio en las ecuaciones, como ya se ha mencionado anteriormente, el Cocomo Intermedio contribuye al cálculo del esfuerzo necesario para el desarrollo

de un Proyecto y del tiempo de desarrollo, incorporando una serie de Atributos, en un total de 15, que actúan como modificadores o coeficientes correctores de los resultados previamente obtenidos. Así, además del número de instrucciones "fuente", necesarias para la obtención de las estimaciones en el Modelo Básico, el Modelo Intermedio toma en consideración aspectos tales como la fiabilidad requerida del Sistema a desarrollar, la capacidad de los analistas y programadores y su experiencia en Aplicaciones, el tiempo de desarrollo requerido, etc. Cada atributo, dependiendo de su rango, proporciona un multiplicador (o coeficiente) que hará variar positiva o negativamente el nivel de esfuerzo y de tiempo requeridos para el desarrollo del Proyecto.

Finalmente, el Cocomo Detallado representa una mejora del Cocomo Intermedio ya que, por una parte, permite utilizar, por cada atributo de coste, multiplicadores del esfuerzo sensibles a la fase, con lo que se puede determinar el esfuerzo necesario para completar cada fase, y por otra parte, si los valores de los atributos de los diferentes componentes son prácticamente iguales, entonces se pueden agrupar para introducirlos una sola vez, simplificando así la entrada de datos.

Además, Boehm especifica tres variantes para Cocomo: El modo Orgánico, para proyectos de pequeño tamaño y poca complejidad; el modo Embebido, para proyectos de mucha envergadura y el modo Híbrido, que representa un compromiso entre los dos anteriores.

El problema que presentan los modelos de estimación de costes basados en Cocomo viene dado, entre otros factores, por la necesidad de saber de antemano y con precisión el tamaño que tendrá el Proyecto cuando haya sido terminado. Además, este dato es el origen de todos los cálculos, y con él y con los distintos coeficientes correctores se obtienen todos los resultados de Cocomo, por lo que si el tamaño del proyecto se estima inadecuadamente, los resultados obtenidos serán muy imprecisos. Esta necesidad de tener que estimar el tamaño del software ha originado críticas al método, aunque éste se encuentra completamente aceptado.

Posteriormente, Granja y Barranco estudian los modelos Cocomo y realizan una serie de modificaciones [Granja - Barranco, 1977], obteniendo una versión modificada aplicable a la previsión de costes de mantenimiento, introduciendo un nuevo parámetro, el índice de mantenibilidad, el cual influye directamente en el coste del mantenimiento.

Todos estos modelos se estudian con detalle más adelante en este mismo capítulo.

2.1.2 Métricas basadas en el diseño y en las especificaciones

A pesar del enorme interés que las métricas basadas en el código han despertado en la comunidad científica, ninguna de ellas ha obtenido resultados óptimos. Por el contrario, éstos han sido muy variables dependiendo del objetivo de la medición, y la mayoría de las veces no se han comportado mejor que LOC, siendo incluso en ocasiones superados por ésta.

Todo ello, combinado con el hecho de que estas métricas no se pueden aplicar hasta que el código ha sido desarrollado (lo que se produce bastante tarde en el Ciclo de Vida del Software), hace que sea deseable enfocar nuestra atención hacia otro tipo de métricas basadas en el diseño y en las especificaciones del software, en las que actualmente se centra el interés de los investigadores. Fundamentalmente, la gran ventaja que aportan estas métricas es el hecho de establecer pronto las correcciones que se deben de introducir en el software, con el fin de evitar problemas posteriores de más difícil solución, rebajando así notablemente los costes de desarrollo. Como ya fue expuesto por Boehm, el coste de detección y corrección de un error aumenta a medida que se avanza en el ciclo de vida del software [Boehm, 1981], volviéndose de gran importancia la detección temprana de los errores para mantener el proyecto bajo control.

No obstante, este tipo de métricas del diseño presentan, al igual que las anteriores basadas en código, algunos inconvenientes que es importante conocer para determinar sus limitaciones. Hay, en primer lugar, una falta de notación formal de uso común por todos los analistas [Shepperd, 1993]; aunque existen diversas técnicas y herramientas para facilitar el diseño estructurado, la tendencia a la hora de diseñar una aplicación suele ser el utilizar técnicas "propias". Más aún, con demasiada frecuencia se emplea el lenguaje natural para detallar especificaciones de proceso que posiblemente se pudieran plasmar de un modo más preciso usando, por ejemplo, una tabla de decisión o un árbol de decisión. Esta tendencia a la *informalidad* a la hora de definir los procesos constituye uno de los principales escollos con los que se encuentran las métricas basadas en el diseño del software. Esto nos lleva al segundo inconveniente de este tipo de métricas: las dificultades que presenta normalmente su validación.

Con el fin de paliar en cierta medida estos inconvenientes, se han definido una serie de características que las métricas deben de cumplir para realizar su objetivo. Destacaremos aquí las consideradas como fundamentales por la mayoría de los autores. En primer lugar, una métrica debe de permitir la extracción automática de los datos que necesita para ser utilizada; esto es obvio debido al gran número de datos que es necesario manejar habitualmente, por lo que se debe de disponer del software adecuado para la captura automática de datos. Además, una métrica debe de poder ser repetible, y ante los mismos datos de entrada deberá de entregar los mismos resultados. Lo que inevitablemente nos conduce al tercer requisito: las métricas deben de ser objetivas, y sus resultados deben de poder interpretarse sin lugar a dudas ni a ambigüedades.

Entre las métricas del diseño a destacar, cabe mencionar las propuestas por Alexander [Alexander, 1964] y posteriormente recogidas y adaptadas por Stevens en su Metodología de Diseño Funcional [Stevens, 1974] relativas a criterios de evaluación del diseño. Estas heurísticas afirman que se debe de "maximizar la cohesión de los módulos que forman un programa y minimizar su acoplamiento". Se

denomina cohesión el propósito o función única de un módulo y acoplamiento su grado de independencia con los demás módulos. Por tanto, según Stevens, cuanto mayor sea la cohesión de un módulo mejor será la calidad del diseño, y por ello, cada módulo deberá de representar una única función bien definida. El acoplamiento puede considerarse como un corolario de la cohesión, ya que si un módulo representa una única función, su grado de independencia será muy alto y por tanto su acoplamiento será mínimo.

2.1.2.1 Flujo de Información

La métrica IF fue propuesta en 1981 por Henry y Kafura [Henry - Kafura, 1981] y está basada en la idea de que la complejidad de un Sistema se puede medir en términos de las conexiones de un módulo con su entorno, definidas por el número de flujos de información que pasan a través de ese módulo.

Así, un módulo A influye en otro módulo B por medio de la información que le proporciona, teniendo de este modo el módulo A la capacidad de hacer variar el comportamiento del módulo B, dependiendo de esa información proporcionada. Esta transmisión de la información puede hacerse de dos modos, bien sea por medio del pase de parámetros desde el módulo A al módulo B, o bien utilizando estructuras comunes para el almacenamiento de la información, donde el módulo A dejaría la información y el módulo B la recogería.

Según los autores, la complejidad de cada módulo se puede calcular basándose en su fan-in y a su fan-out. El fan-in de un módulo se define como el número de flujos locales que terminan en un módulo más el número de estructuras de datos de las que se recibe la información. El fan-out de un módulo se define como el número de flujos locales que salen de un módulo más el número de estructuras de datos que son actualizadas por ese módulo. La complejidad de cada módulo se define entonces como:

Longitud del módulo x (fan-in x fan-out)^2

Una de las causas por las que IF despertó un gran interés fue que su idea de fondo concordaba con las normas de diseño del software, que indicaban que se debe de minimizar la complejidad de las interfases entre un componente del diseño y su entorno [Alexander, 1964] [Parnas, 1972] [Myers, 1979]

Posteriormente en 1991, Shepperd e Ince estudian la forma en que el fan-out y el tamaño de los módulos (es decir, las decisiones relativas al modo de partir un Sistema en módulos, y la manera en que esos módulos interaccionan con los demás) tienen una importancia significativa en la tendencia al error del Sistema, a su facilidad de mantenimiento, a la capacidad de reutilización de los módulos, etc. [Shepperd - Ince, 1991], y algo más tarde, en 1983, presentan modificaciones a la métrica. Afirman [Shepperd, 1993] que se debe de suprimir la longitud del módulo en el cálculo de la complejidad. Si una interfase es compleja, entonces no se puede trabajar de un modo aislado en un componente del diseño, lo cual perjudica notablemente las acciones de mantenimiento. Esto provoca características no deseables en el software, y por tanto es preferible medir la complejidad de las interfases entre módulos a medir la complejidad del propio módulo, lo cual es equivalente a medir el IF si se suprime la longitud del módulo de la fórmula de Henry y Kafura.

Otra simplificación que proponen Ince y Shepperd consiste en tratar por igual los flujos enviados a un módulo (o recibidos por un módulo) por medio de estructuras comunes o por medio de pase de parámetros, no contar los flujos duplicados y modelizar la reutilización de los módulos, contando una sola vez la interfase de los módulos que se reutilizan, aunque el módulo reutilizado sí contribuye al fan-in y al fan-out del módulo que lo llama. De esta manera, se reduce la complejidad de la interfase de los módulos reutilizados de un programa y no se penaliza la reutilización, que es una característica deseable del diseño de un sistema software.

2.1.2.2 Puntos-Función

Otra de las métricas basadas en el diseño es la debida a Albretch [Albretch, 1979] y conocida como el Método de Puntos-Función. Albretch sugiere que podría estimarse el tamaño de un Sistema Software teniendo en cuenta las diferentes funciones que el software deberá de realizar, y posteriormente asignando un valor, en puntos, a cada una de esas funciones.

En general, el método consiste en contar las funciones que debe realizar el software, ajustarlas según su complejidad y hacer una estimación basada en la suma de los pesos de los Puntos-Función resultantes.

Así, propone Albretch contar, a partir del diseño, las siguientes funciones:

- Número de entradas externas
- Número de salidas externas
- Número de ficheros lógicos internos
- Número de ficheros de interfase
- Número de consultas externas

El número total de funciones de cada clase se ajusta posteriormente según su complejidad, multiplicándolo por un valor determinado, que varía entre 3 y 15 dependiendo de si la función es simple, tiene una complejidad promedio o es una función compleja. A continuación se sumarían los resultados y se obtendría el total de Puntos-Función iniciales, PF.

En la tabla siguiente se muestran los coeficientes multiplicadores de Albretch según la complejidad de las funciones.

Contar, a partir del diseño:	Simple	Promedio	Compleja
Entradas externas	3	4	6
Salidas externas	4	5	7
Ficheros lógicos internos	7	10	15
Ficheros de interfase	5	7	10
Consultas externas	3	4	6

Tabla A.1 Coeficientes de ajuste de la complejidad de las funciones

Seguidamente, es necesario calcular la complejidad del proyecto, con respecto a 14 características que pueden o no estar presentes en el mismo, y que se valoran de 0 a 6, dependiendo de su influencia en el proyecto. Un valor cero indica que la característica no está presente o no tiene influencia y un valor 6 indica una fuerte influencia.

Estas características son las siguientes:

- Comunicación de datos
- Funciones distribuidas
- Rendimiento
- Configuración
- Ratio de transacciones
- Entrada de datos “on-line”
- Eficiencia del usuario final
- Actualización “on-line”
- Complejidad de procesamiento
- Reutilización del software
- Facilidad de instalación

- Facilidad de operación
- Lugares múltiples
- Facilidad de cambios

Sumando los resultados parciales, se obtiene la complejidad total del proyecto C_p, que se ajusta según la siguiente fórmula, obteniéndose así la complejidad ajustada del proyecto, C_{pa}.

$$C_{pa} = 0,65 + (0,01C_p)$$

Finalmente, se calculan los Puntos-Función ajustados, PF_a, como:

$$PF_a = PF \times C_{pa}$$

Estos Puntos-Función ajustados los utiliza Albretch para estimar el tamaño del proyecto, proporcionando una estimación de las líneas de código a partir de los puntos función, y dependiendo del lenguaje de desarrollo empleado. Presenta además un Modelo General Simplificado, para cualquier lenguaje:

$$Nº \text{ de Instrucciones (NINST)} = 66PF_a$$

Algunos de los inconvenientes del método vienen dados por la arbitrariedad de los coeficientes de ajuste utilizados por Albretch, así como a la dificultad para identificar las funciones del software a partir de las especificaciones, ya que éste es un proceso bastante subjetivo.

Este método se estudia con todo detalle más adelante en este mismo tema.

2.1.2.3 La Métrica "Bang"

Otra de las métricas orientadas al diseño es la métrica "Bang", propuesta por DeMarco en 1982, y que clasifica el software en dos categorías: el software "fuerte en funciones", como podrían ser los sistemas basados en la robótica, y el software "fuerte en datos", como serían los sistemas que traten con la recuperación y procesamiento de la información [DeMarco, 1982] La métrica Bang se deriva de las especificaciones estructuradas del diseño, haciendo uso de técnicas y notaciones como por ejemplo los Diagramas de Flujo de Datos, los Diagramas Entidad-Relación, los Diagramas de Transición de Estados, y de herramientas de Administración de Datos, como los Diccionarios de Datos, entre otras.

2.1.2.4 Métricas específicas para programas COBOL

El lenguaje de programación Cobol ha sido el lenguaje más empleado en todos los Centros de Proceso de Datos que utilizan grandes sistemas para realizar sus desarrollos informáticos. Este lenguaje sigue siendo todavía el más usado a nivel internacional y es el lenguaje en el que se encuentra implementada la mayoría de los sistemas que se mantienen en la actualidad [Piattini *et al*, 1998] Por eso no es de extrañar que diversos autores hayan definido métodos para el control del mantenimiento de aplicaciones Cobol. Entre otros, se encuentran los trabajos de Rossi [Rossi, 1991], y Natale [Natale, 1995]

Para controlar la mantenibilidad de los programas Cobol, Rossi identifica una serie de atributos del software y propone unos determinados valores para algunos de ellos. Los atributos propuestos y sus valores son los siguientes:

1. Número total de líneas de código
2. Número total de líneas de procedimientos (Líneas de la PROCEDURE DIVISION)
3. Número de líneas de código ejecutables
4. Número de líneas de código ejecutables "destructivas", tales como ENTRY y ALTER. No se deberían de utilizar.

5. Código muerto (código que nunca se ejecuta) Se debería de eliminar.
6. Bloques recursivos de instrucciones PERFORM y subrutinas. No se deberían de utilizar.
7. Densidad de instrucciones condicionales (número de instrucciones que comprueban una o más condiciones) No debería de exceder del 15%.
8. Densidad de instrucciones de modificación del flujo de control (número de instrucciones de modificación del flujo de control dividido entre el número total de sentencias ejecutables) Debería de estar entre el 5% y el 15%.
9. Densidad de instrucciones PERFORM (número de instrucciones PERFORM dividido entre el número de líneas de código) Entre un 5% y un 12%.
10. Número de subrutinas llamadas vía PERFORM. Un 5% del número de líneas de PROCEDURE DIVISION.
11. Densidad de comentarios (líneas de comentario / número total de instrucciones) Entre un 10% y un 20%.
12. Corrección de la alineación de las instrucciones. Grado de cumplimiento de los estándares de la organización. Cuanto más alto, mejor.

Por su parte, Natale propone llevar a cabo dos niveles de control del mantenimiento: un control de la dimensión y la complejidad del código y un control de la estructura y distribución de dicha complejidad dentro de cada programa. Para el primer nivel de control, identifica Natale una serie de elementos que deben de ser controlados, tales como el número de líneas de código, el número de instrucciones ALTER y de instrucciones GO TO, que debería de ser cero, el número de instrucciones PERFORM, el número de IF anidados, el número de nodos y de arcos, el número de instrucciones LABEL y el número de instrucciones MULTI / ENTRY / EXIT.

2.1.2.5 Métricas para el Diseño Orientado a Objetos

En estos últimos años, los sistemas orientados a objetos han causado mucho interés en un gran número de áreas de la Informática. La primera aparición del concepto de "objeto" como un constructor de programación fue en Simula, un lenguaje para programar simulaciones por ordenador [Birtwistle, 1973] Posteriormente, el concepto de objeto se extendió al desarrollo de aplicaciones y al prototipado, siendo este el enfoque propuesto por el lenguaje Smaltalk [Goldberg – Robson, 1983], refinando el concepto de "clase de objeto" introducido en Simula.

Desde hace varios años, se ha estado utilizando la encapsulación de procedimientos, macros y librerías con el fin de conseguir la reutilización del software. Las técnicas de orientación a objetos consiguen una reutilización aún mayor encapsulando programas y datos. Técnicas y mecanismos tales como la instanciación, las clases, la herencia y el polimorfismo están relacionadas con los paradigmas de empaquetamiento de objetos, de tal manera que se puedan reutilizar convenientemente sin tener que modificarlas para resolver nuevos problemas [Nierstrasz, 1989]

En un diseño orientado a objetos, éstos se agrupan en clases, las cuales a su vez están formadas por los atributos (o propiedades del objeto) y por los métodos (operaciones que se pueden realizar sobre los objetos) De esta manera, el conjunto de atributos y métodos describen una clase determinada. Además, una clase puede heredar propiedades de otras clases y un objeto de una clase puede enviar (o recibir) información de otros objetos, que actúe como "disparador" de los métodos del objeto que la recibe. Por ejemplo, la clase *Tractor* puede heredar todas o algunas de las propiedades de la clase *Vehículo*, y el método *SalirDelHangar* de un objeto de clase *Avión* puede provocar la ejecución de los métodos *AbrirPuertaHangar*, *EsperarSalidaHangar* y *CerrarPuertaHangar*, de la clase *Hangar.*

Como se puede apreciar, la problemática para la construcción de programas orientados a objetos es algo diferente del diseño y desarrollo de programas "semánticos", aunque ambos presenten elementos comunes, ya que, por ejemplo, la encapsulación de datos y procedimientos se ha utilizado desde siempre para la reutilización del software, como se ha mencionado anteriormente. Por tanto, aunque algunas métricas "convencionales" se puedan emplear en programación orientada a objetos, es conveniente disponer de métricas propias, adecuadas a las nuevas características del diseño orientado a objetos.

De todas las métricas propuestas para analizar la programación orientada a objetos, posiblemente las más difundidas sean las métricas conocidas como "conjunto de métricas CK", presentadas por Chidamber y Kemerer en 1994 [Chidamber - Kemerer, 1994]

Son las siguientes:

1. WMC Weighted Methods per Class (Métodos Ponderados por Clase)

Se define WMC como el sumatorio de las complejidades de los métodos de una clase. Por tanto, si la complejidad de todos los métodos es uno, WMC representará el número de métodos de la clase. A mayor número de métodos en una clase, mayor impacto potencial (frente a modificaciones, por ejemplo) en los hijos, que heredan las propiedades y métodos de la clase, y mayor esfuerzo de desarrollo y mantenimiento de la clase.

2. DIT Depth of Inheritance Tree (Profundidad del Arbol de Herencia)

Se define como la longitud máxima desde un determinado nodo a la raíz del árbol de herencia. Cuanta más profundidad, mayor será la complejidad del diseño y más difícil será predecir el comportamiento de una clase. No obstante, una gran

profundidad en el árbol de herencia puede entenderse como una gran capacidad de reutilización potencial de los métodos que se heredan.

3. NOC Number of Children (Número de Hijos)

Es el número de subclases inmediatamente subordinadas a una clase en la jerarquía de herencia. Cuanto mayor sea el número de hijos, mayor será la posibilidad de reutilización, aunque también, al estar más especializado, será mayor la posibilidad de desaprovechar el tratamiento que proporciona cada hijo. Por otra parte, cuantos más hijos tenga una clase, más difícil será de suprimir, en el caso de tener que hacerlo.

4. CBO Coupling Between Object classes (Acoplamiento Entre Clases de Objetos)

Dos objetos están acoplados si y sólo si al menos uno de ellos actúa sobre el otro. Se dice que un objeto actúa sobre otro si existe algún método de dicho objeto que realiza alguna acción sobre un método o sobre alguna propiedad del otro. Al igual que en el diseño convencional [Stevens, 1974], las heurísticas de acoplamiento son las mismas y se cumplen en ambos casos; es decir, un acoplamiento excesivo entre clases va en detrimento del diseño modular, dificulta el mantenimiento e impide la reutilización, por lo que su medida puede ayudarnos a determinar la complejidad de las pruebas [Piattini *et al*, 1998]

5. RFC Response For a Class (Respuesta Para una Clase)

La Respuesta para una Clase se define como el número de métodos de una clase determinada que se pueden invocar como respuesta a un mensaje de un objeto de esa clase. Cuantos más métodos se puedan invocar, más compleja será la clase, y por tanto su mantenimiento será más difícil.

6. LCOM Lack of Cohesion in Methods (Falta de Cohesión en los Métodos)

Se define la Falta de Cohesión en los Métodos como el número de pares de métodos que no acceden a atributos comunes. La cohesión es una característica deseable, ya que favorece la encapsulación, y su carencia incrementa la complejidad y la probabilidad consiguiente de cometer errores durante el desarrollo. Además, esta carencia de cohesión es un indicador de que las clases probablemente se deberían de subdividir en subclases. Así como en el diseño convencional se debe de maximizar la cohesión y minimizar el acoplamiento entre los módulos de un programa, en programación orientada a objetos esto se debe de hacer entre las distintas clases, para facilitar su desarrollo y posterior mantenimiento.

Además de Chidamber y Kemerer, son muchos los autores que han propuesto métricas para sistemas orientados a objetos. Podemos destacar aquí las presentadas por Li y Henry (1993), Sharble y Cohen (1993), por Lorenz y Kidd (1994), y más recientemente por Henderson-Sellers, en 1996. [Li - Henry, 1993], [Sharble - Cohen, 1993], [Lorenz - Kidd, 1994], [Henderson-Sellers, 1996]

2.1.3 *Marcos de medida y estándares principales*

Antes de iniciar un proyecto de medición de cualquier clase, y en general antes de iniciar cualquier proyecto, es necesario establecer claramente cuáles son los objetivos que se espera alcanzar, de qué medios se dispone para alcanzarlos y cuáles son los beneficios esperados. Muchos proyectos de medición han fracasado por no tener claramente especificados sus objetivos y por no disponer de un marco de trabajo concreto para desarrollarlos.

En este punto abordaremos dos propuestas para paliar este problema; la primera proporciona una aproximación rigurosa a la medición orientada a objetivos, y la segunda facilita un medio para comprobar el nivel de madurez de una organización en la mejora de los procesos de desarrollo de software.

2.1.3.1 Goal-Question-Metrics (GQM)

En 1998, Vic Basili y sus colegas de la Universidad de Maryland, desarrollaron un método para seleccionar las métricas más adecuadas a los objetivos que se esperaba alcanzar con su utilización [Basili – Rombach, 1998] Dado el carácter intuitivo de este método, ha conseguido una difusión y utilización muy amplia y una muy buena aceptación en el entorno de la Ingeniería del Software. La idea principal del método es muy sencilla y se concreta en tres pasos:

1. Definir los objetivos (goals) a alcanzar en términos de propósito, perspectiva y entorno.
2. Refinar los objetivos previamente definidos por medio de preguntas (questions)
3. Deducir las métricas y datos (y los medios para conseguirlos) necesarios para contestar a las preguntas anteriores.

En la figura siguiente se muestra cómo a partir de un objetivo único se obtienen varias métricas que pueden ser empleadas para alcanzarlo.

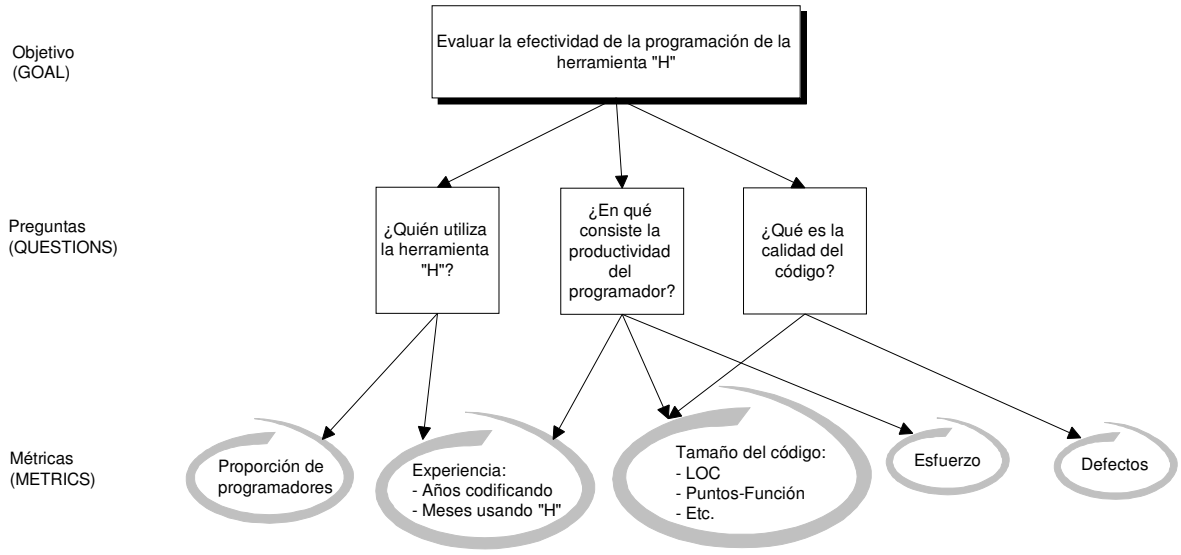


Fig. A.2 Deducción de métricas a partir de objetivos y preguntas

En la figura se indica que el objetivo general es evaluar la efectividad del uso de un estándar de codificación (programación orientada a objetos, por ejemplo), y para ello es necesario responder a varias preguntas. En primer lugar, es importante saber quién utiliza ese estándar de programación, de modo que se pueda comparar la productividad de los programadores que la utilizan y la de los que no la utilizan. Asimismo, es posible que se quiera comparar la calidad del código obtenido por los programadores que utilizan el estándar con la calidad de los que no lo utilizan. Para ello, es necesario hacerse preguntas sobre productividad y calidad.

Una vez que se han identificado las preguntas relativas a los objetivos, se deberán de analizar para averiguar qué es lo que se debe de medir para contestar esas preguntas. Por ejemplo, para saber quién utiliza el estándar, es necesario saber la proporción de programadores que lo usan. Sin embargo, también es importante conocer la experiencia de esos programadores con el uso del estándar, con el entorno de desarrollo, con el lenguaje de programación, así como cualquier otro factor que ayude a valorar la efectividad del estándar.

La pregunta sobre productividad implica definir qué es productividad, normalmente una medida del esfuerzo dividida entre una medida del tamaño del software de algún tipo. Como se muestra en el ejemplo, la métrica empleada podría ser Líneas de Código, Puntos-Función, o cualquier otra métrica adecuada. De una manera semejante, la calidad del software se podría medir contando el número de errores por miles de líneas de código, por ejemplo.

De esta manera, sólo se emplearán las métricas adecuadas para alcanzar el objetivo planteado. Nótese que, en algunos casos, se pueden necesitar varias métricas para contestar a una única pregunta, o una sola métrica puede responder a varias preguntas. El objetivo proporciona la causa para recoger los datos necesarios para las métricas y las preguntas indican cómo se van a usar esos datos.

En la tabla siguiente se muestra un ejemplo del uso de GQM por AT&T para determinar qué métricas son apropiadas para validar sus procesos de revisión de la calidad del código desarrollado [Barnard – Price, 1994]

Goal	Questions	Metrics
Plan	¿Cuánto cuesta el proceso de revisión?	Esfuerzo medio por miles de líneas de código (KLDC)
	¿Cuántos días tarda el proceso de revisión?	Esfuerzo medio KLDC de código. Total de KLDC revisadas.
Monitorización y control	¿Cómo es la calidad del software revisado?	Media de defectos encontrados por KLDC. Media de la tasa de revisión. Media de la tasa de preparación.
	¿En qué medida el equipo de programación cumple los estándares?	Media de la tasa de revisión. Media de la tasa de preparación. Media de líneas de código revisadas. Porcentaje de revisiones.
	¿Cuál es el estado actual del proceso de revisión?	Total de KLDC revisadas.
Mejoras	¿Cuán efectivo es el proceso de revisión?	Eficiencia de la corrección de errores. Media de errores detectados por KLDC. Media de la tasa de revisión. Media de la tasa de preparación. Media de líneas de código revisadas.
	¿Cuál es la productividad del proceso de revisión?	Esfuerzo medio por error encontrado. Media de la tasa de revisión. Media de la tasa de preparación. Media de líneas de código revisadas.

Tabla A.2 Ejemplo del uso de GQM por AT&T

Además de GQM, existen otros métodos aplicables para definir objetivos medibles, de entre los cuales los más conocidos son:

- QDF (Quality Function Deployment Approach)

Es una técnica que ha evolucionado a partir de los principios de Gestión de la Calidad Total (Total Quality Management, TQM), que intentan obtener indicadores desde el punto de vista de los usuarios. El método QDF utiliza matrices simples

(conocidas como “Casa de la Calidad”) con pesos de acuerdo con el criterio de los usuarios.

- SQM (Software Quality Metrics)

2.1.3.2 Capability Maturity Model (CMM)

La “mejora de procesos” es un término para indicar una tendencia, iniciada con los trabajos del Software Engineering Institute en Carnegie Mellon, que se mueve alrededor de la idea de que la calidad final del software está fuertemente relacionada con la mejora de los procesos de desarrollo.

El Modelo de la Madurez de la Capacidad, CMM, fue inicialmente desarrollado por Humphrey [Humphrey, 1989] por encargo del Departamento de Defensa de los Estados Unidos (DOD), como consecuencia de los problemas experimentados para la obtención de su software. Para garantizar la calidad del software, el DOD necesitaba un modo de evaluar la idoneidad de los posibles suministradores.

CMM es un modelo de cinco niveles que muestra la madurez de los procesos de desarrollo de software en las organizaciones, basado en los conceptos de la Gestión de la Calidad Total (TQM) mencionados en el punto anterior. Utilizando cuestionarios, entrevistas y recogida de evidencias, se pueden encuadrar las organizaciones en uno de los cinco niveles de madurez de CMM, basándose principalmente en el rigor de sus procesos de desarrollo.

Con excepción del nivel 1, cada nivel se caracteriza por un conjunto de “áreas de procesos clave”. Por ejemplo, para el nivel 2, las áreas clave son: gestión de requisitos, planificación de proyectos, seguimiento de proyectos, gestión de la subcontratación, control de calidad y gestión de la configuración, y para el nivel 5 son: prevención de defectos, gestión de cambios de la tecnología y gestión del cambio de procesos.

Idealmente, se espera que una empresa se encuentre, como mínimo, en el nivel 3 para optar por un contrato del DOD. Esta motivación comercial ha sido la razón principal del auge de CMM. No obstante, pocas empresas en Estados Unidos han conseguido superar el nivel 3 y la mayoría de ellas se encuentran aún en el nivel 1, y sólo hace unos pocos años se han catalogado algunas como nivel 5. La más conocida es el área de IBM encargada del desarrollo del software del programa de la lanzadera espacial de la NASA [Keller, 1992]

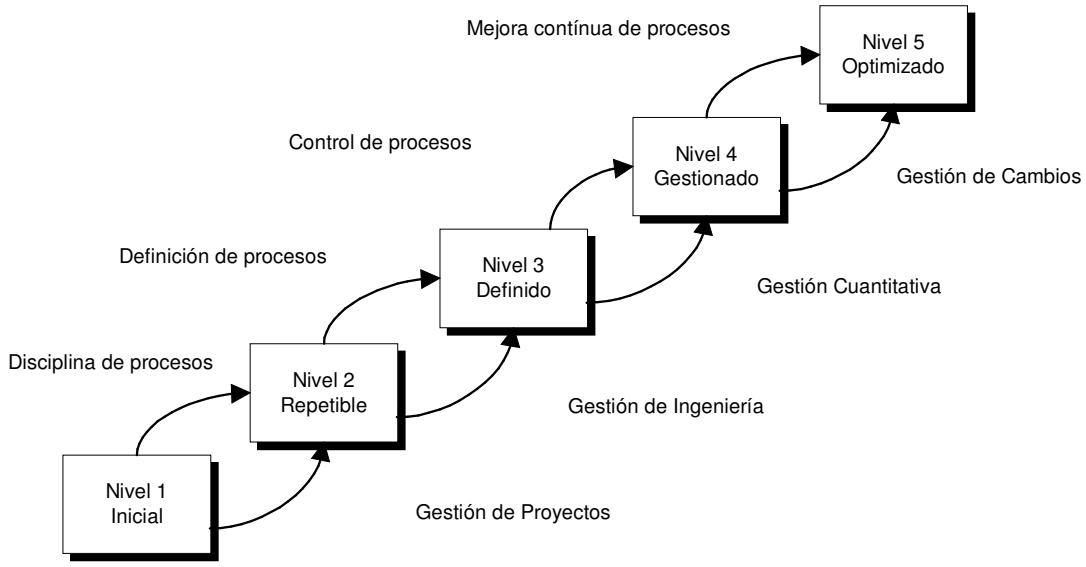


Fig. A.3 El Modelo de la Madurez de la Capacidad, CMM.

El modelo CMM ha conseguido un gran impacto a escala internacional, el cual ha originado un aumento significativo de la concienciación sobre la necesidad de utilización de métricas del software. La razón de ello es la importancia de las métricas en las áreas de procesos clave del modelo.

Las métricas del nivel 1 proporcionan una línea base de comparación a medida que se intenta mejorar los procesos y los productos. En el nivel 2, las medidas se enfocan en la gestión de proyectos, mientras que en el nivel 3 se miden los productos

intermedios y finales obtenidos durante el proceso de desarrollo del software. Las métricas del nivel 4 obtienen las características del proceso de desarrollo en si mismo, para permitir el control de las actividades individuales de los procesos. Por último, un proceso en el nivel 5 es lo suficiente maduro y está lo suficientemente gestionado como para permitir mediciones que proporcionen una retroalimentación para realizar cambios dinámicos del proceso durante el desarrollo de un proyecto en particular.

En la tabla siguiente se muestran los tipos de métricas sugeridas para cada nivel de madurez, donde la selección depende de la cantidad de información visible y disponible para un nivel de madurez dado.

Nivel de Madurez	**Características**	**Tipo de Métricas a Utilizar**
5. Optimizado	Retroalimentación para mejoras dinámicas del proceso	Procesos y feedback para cambiar los procesos
4. Gestionado	Proceso medido	Procesos más feedback para control
3. Definido	Proceso definido e institucionalizado	Producto
2. Repetible	Proceso dependiente de los individuos	Gestión de Proyectos
1. Inicial	Ad hoc	Línea Base

Tabla A.3 Métricas según el nivel de madurez

A pesar de su aceptación internacional, CMM no está libre de críticas. La principal tiene que ver con la validez de los cinco niveles en si mismos. No existe de momento ninguna prueba convincente de que las empresas con un nivel de madurez alto produzcan un software de más calidad. También existen críticas sobre el cuestionario utilizado para catalogar a las organizaciones [Bollinger – McGowan, 1991]

Un proyecto europeo (fundado bajo el programa ESPRIT) fuertemente relacionado con CMM es Bootstrap [Woda – Schynoll, 1992] Este método es también un marco para asesorar sobre la madurez de los procesos software. La diferencia con CMM estriba en que son los proyectos individuales y no las organizaciones los que pueden ser evaluados y calificados con cualquier número real entre 1 y 5. Así, por ejemplo, un determinado departamento puede estar valorado con un 2,6 indicando que supera el nivel 2 de CMM pero que no llega aún al nivel 3.

Otro desarrollo más reciente en el área de mejora de procesos es SPICE (Software Process Improvement and Capability dEtermination), un proyecto internacional [ISO/IEC, 1993] cuyo objetivo es desarrollar un estándar para el asesoramiento en los procesos software, construido con las mejores características de CMM, Bootstrap e ISO 9003, descrito más adelante.

2.1.3.3 Estándares principales

Literalmente, existen cientos de estándares nacionales e internacionales que tienen una relación directa o indirecta con el control de calidad del software. Una crítica bastante generalizada sobre todos ellos es que son subjetivos por naturaleza y que se concentran casi exclusivamente en el desarrollo de procesos en vez de en los productos [Fenton et al, 1993]

A pesar de esas críticas, existen algunos estándares genéricos sobre el control de calidad del software que tienen un impacto importante en las actividades de las métricas del software para el control de calidad, por lo que se citan a continuación.

- **Series ISO 9000 y TickIT**

Tanto en Europa como más recientemente en Japón, el estándar de calidad que más difundido está basado en el estándar internacional ISO 9001 [ISO 9001, 1987], un estándar general para manufacturación, que especifica un conjunto de 20 requisitos

para sistemas de gestión de calidad, política, organización, responsabilidades y revisiones, además de definir los controles que deben de ser aplicados a las actividades del ciclo de vida para conseguir productos de calidad.

ISO 9001 no es específico para ningún sector del mercado en particular. La parte referida al software se recoge en ISO 9003 [ISO 9001, 1990] Este estándar es asimismo la base de TickIT, una iniciativa que está patrocinada por el Departamento de Comercio e Industria del Reino Unido [TickIT, 1992] Las empresas que lo deseen tienen que solicitar una revisión para que se certifique que cumplen el estándar, revisión que deben de pasar cada 3 años. La mayoría de las empresas de IT más importantes del Reino Unido ya disponen de esa certificación.

Además, los distintos países tienen sus propios estándares basados en ISO 9000. Por ejemplo, en el Reino Unido el equivalente es la serie de estándares BS5750, mientras que en la Comunidad Económica Europea el equivalente de ISO 9001 es EN 29001.

- **ISO 9126 Evaluación de productos software: Características de la calidad y guías para su utilización.**

Es el primer estándar internacional que intenta definir un marco para evaluar la calidad del software [ISO 9126, 1991] El estándar define la calidad como "la totalidad de las características y elementos de un producto software que le confieren su capacidad de satisfacer los requisitos impuestos o implícitos".

Con una fuerte influencia del enfoque de SQM ya descrito anteriormente, ISO 9126 afirma que la calidad del software de debe de evaluar a través de seis características: funcionalidad, fiabilidad, eficiencia, capacidad de uso, capacidad de mantenimiento y portabilidad. Cada una de esas características se definen como "un conjunto de atributos que tienen que ver con" los aspectos relevantes del software, y que se pueden refinar a través de múltiples niveles de subcaracterísticas.

Así, por ejemplo, la fiabilidad se define como:

"Un conjunto de atributos que tienen que ver con la capacidad del software de mantener su nivel de rendimiento bajo condiciones dadas por un período de tiempo definido"

Y la portabilidad como:

"Un conjunto de atributos que tienen que ver con la capacidad del software de ser trasladado de un entorno a otro"

También se proporcionan ejemplos de posibles definiciones de las subcaracterísticas, pero se incluyen en el Anexo A, que no forma parte del estándar, y se dejan totalmente sin definir los atributos al segundo nivel de refinamiento, por lo que algunas personas han argüido que el estándar ISO 9126 no proporciona un marco conceptual dentro del cual las distintas partes implicadas (usuarios, vendedores y agencias de regulación), puedan hacer medidas comparativas desde sus distintos puntos de vista de la calidad del software. Además, las definiciones de atributos, tales como la fiabilidad, difieren de la de otros estándares bien establecidos.

De cualquier manera, ISO 9126 constituye un paso importante en el desarrollo de la medida de la calidad del software.

- **IEEE 1061: Metodología de las métricas de la calidad del software.**

El estándar IEEE 1061 [IEEE, 1989] [IEEE, 1992] se completó en el año 1992, y no indica ninguna métrica en concreto del producto software, aunque dispone de un apéndice que describe el enfoque de SQM. Más bien proporciona una metodología

para establecer los requisitos de calidad y para identificar, analizar y validar las métricas de la calidad del software.

2.1.4 Consideraciones finales sobre las métricas actuales

Después de haber analizado algunas de las métricas más relevantes del código y del diseño, así como las últimas métricas para programación orientada a objetos, podemos concluir que de momento no se ha explotado todo el potencial que presentan, aunque están en marcha diversos estudios encaminados en esta dirección. No debemos ocultar el hecho de que las métricas basadas en el diseño y en las especificaciones presentan problemas, tales como no definir claramente lo que se va a medir, y que trabajan con frecuencia con aspectos tales como la complejidad y la calidad de los sistemas, difícilmente medibles y estimables, por lo que adolecen de falta de precisión en algunos casos. A pesar de ello, estas métricas, correctamente empleadas, proporcionan mayores beneficios que las métricas basadas en código, ya que son capaces de detectar problemas durante las primeras fases del desarrollo y son capaces de trabajar con aspectos tales como las funciones y la estructura de los sistemas, los cuales, al ser más abstractos, proporcionan un mayor abanico de aplicabilidad.

Sea cual sea la métrica elegida, hay que tener en cuenta que su aplicación va a ser costosa, ya que es necesario invertir un esfuerzo en definirla, implementarla, utilizarla y analizar sus resultados, procesos todos estos que son consumidores de tiempo y de recursos. Por eso, en algunos sectores empresariales ha surgido, en ocasiones, la controversia sobre si es conveniente utilizarlas o no.

En nuestra opinión, una métrica bien elegida e implementada proporciona muchos más beneficios que costes origina, por lo que no debería de haber ninguna duda sobre la conveniencia o no de implantar un sistema de métricas del software en la empresa. Sobre todo considerando que, hoy en día, tanto el gobierno como la industria en general se han dado cuenta de la cada vez mayor dependencia de la

Informática, lo que hace totalmente necesario medir las características básicas de los sistemas software que afectan a los requisitos de calidad, tales como la funcionalidad, fiabilidad, costes y calendario de los desarrollos informáticos [ISO/IEC, 1998]

Para finalizar, simplemente recordar que no es posible controlar lo que no se puede medir, y que además, en palabras de Schulmeyer, lo que no se puede medir tampoco se puede estimar.

2.1.5 Bibliografía

[Albretch - Gaffney, 1983] Albretch, A. J. and Gaffney, J. R. *Software function, source lines of code, and development effort prediction: a software science validation*. IEEE Transaction on Software Engineering, 9(6): 639-48.

[Basili - Hutchens, 1983] Basili, V. R. and Hutchens, D. H. *An empirical study of a syntactic complexity family*. IEEE Transactions on Software Engineering, 9(6): 664-72. 1983.

[Behrens, 1983] Behrens, C. A. *Measuring the productivity of computer systems development activities with function-points*. IEEE Transactions on Software Engineering, 9(6): 649-58. 1983.

[Boehm, 1981] B. W. Boehm. *Software Engineering Economics*. Englewood Cliffs, NJ Prentice-Hall, Inc., 1981.

[Chidamber - Kemerer, 1994] Chindamber, R. S. and Kemerer, C. F. *A metrics suite for object oriented design*. IEEE Transactions on Software Engineering, vol. 20, nº 6, June. 1994.

[DeMarco, 1982] Thomas DeMarco. *Controlling Software Projects: Management, Measurement, Estimation.* Englewood Cliffs, NJ. Prentice Hall. 1982.

[DOD, 1988] U. S. Dept. Of Defense, DOD-STD-2168. *Defense System Software Quality Program*. Washington, DC: NAVMAT 09Y. 1988.

[Fenton - Pfleeger, 1996] Fenton, N. E. and Pfleeger, S. L. *Software Metrics. A rigorous & practical approach*. International Thomson Computer Press. 1996.

[Halstead, 1977] Halstead, M. *Elements of Software Science*. Elsevier, New York. North-Nolland, 1977.

[Henry - Kafura, 1981] Henry S. and D. Kafura. *Software Structure Metrics Based on Information Flow.* IEEE Transactions on Software Engineering. 7(5): 510-18. Los Alamitos, CA. IEEE Computer Society. 1981.

[McCabe, 1976] T. J. McCabe. *A Complexity Measure.* IEEE Transactions on Software Engineering. SE1(3): 312-27. 1976.

[Natale, 1995] Natale, D. *Qualità e quantità nei sistemi software. Teoria de esperienze.*Ed. FrancoAngeli. Milán. 1995.

[Oviedo, 1980] Oviedo, E. *Control flow, data flow and program complexity*. Proceedings COMPSAC 80, 146-152. 1980.

[Piattini *et al*, 1998] Piattini, M., Villalba, J., Ruiz, F., Fernández, I., Polo, M., Bastanchury, T., Martínez, M. A. *Mantenimiento del Software. Conceptos, métodos, herramientas y outsourcing*. RA-MA Editorial. Julio, 1998.

[Putnam, 1978] Putnam, L. *A General Empirical Solution to the Macro Software Sizing and Estimating Problem*. IEEE Transactions on Software Engineering, vol. 4, nº 4. 1978.

[Rossi *et al*., 1991] Rossi, P., Antonini, P. y Lanza, T. *Experience of Software Maintainability in SIP*. Proceedings of the International Conference on Data Engineering. IEEE. 1996.

[Schulmeyer - McManus, 1996] G. Gordon Schulmeyer & James I. McManus. *Handbook of Software Quality Assurance*. Thompson Computer Press. 1996.

[Shepperd, 1993] Shepperd, M. J. *Software Engineering Metrics Volume I: Measures and Validations*. McGraw-Hill International, 1983.

TEMA 2 METRICAS Y MODELOS DE ESTIMACION DEL SOFTWARE

2.2 ***Introducción***

En este tema se estudiarán diversos métodos de estimación de tiempos y costes de desarrollo de Aplicaciones Software, así como una herramienta que implementa algunos de ellos, facilitando de este modo la tarea, siempre complicada e ingrata, de predecir, incluso cuando se dispone de muy pocos elementos de juicio, cuanto va a durar un desarrollo software.

2.2.1 *¿Para qué se necesita estimar costes?*

Evidentemente, "*la mejor estimación es no hacer estimación ninguna. Así no se corre el riesgo de que la estimación falle*". Afortunadamente, esta aseveración no suele convencer a las personas que contratan el desarrollo de un Sistema y por eso es necesario prever con antelación suficiente cual va a ser el coste de nuestros proyectos, conociendo su duración previsible y los recursos necesarios para llevarlos a cabo. Dejando aparte el chiste fácil, hoy en día resulta de todo punto impensable acometer el desarrollo de un Proyecto serio sin una estimación previa "decente".

Por desgracia, estimar los costes de un Proyecto supone intentar anticiparse a los acontecimientos y predecir, normalmente con pocos elementos de juicio, el comportamiento del Proyecto a lo largo del tiempo. Como se puede suponer, no se trata de una tarea fácil, pero sí es una labor totalmente imprescindible para la buena marcha del negocio, y es por eso que desde hace ya bastantes años un gran número de investigadores se han dedicado a intentar encontrar una fórmula que permita aproximar, en mayor o menor medida, las estimaciones a la posterior realidad.

No existen por tanto soluciones mágicas, ni los resultados de las estimaciones se pueden tomar como dogmas de fe. Sin embargo, si se toman las debidas precauciones, las sucesivas estimaciones de un Proyecto deben de proporcionar una vara de medir bastante buena para la toma de decisiones relativas al Proyecto y para conocer, con las salvedades oportunas, cómo se va a comportar dicho Proyecto a medida que se avance en su desarrollo.

2.2.2 *Los modelos COCOMO: Una herramienta de estimación*

Los modelos **COCOMO** (Constructive Cost Model) de Barry W. Boehm, en sus dos versiones iniciales, Cocomo Básico y Cocomo Intermedio, están basados en los estudios realizados por Boehm, y recogidos en su libro Software Engineering Economics [Boehm, 1981]

De los dos modelos, el Cocomo Básico, permite dar una primera estimación durante la fase de Análisis Previo, cuando la mayoría de los factores que van a influir en el proyecto son todavía desconocidos, apoyándose tan solo en el número de líneas "fuente" que se estima tendrá el proyecto una vez desarrollado. Existen tres variantes para el Cocomo Básico: El modo Orgánico, para proyectos de pequeño tamaño y poca complejidad, el modo Embebido, para proyectos de mucha envergadura y el modo Híbrido, que representa un compromiso entre los dos anteriores.

El modelo Cocomo Intermedio, toma además en consideración aspectos tales como la capacidad de los programadores y analistas, la seguridad y la fiabilidad requerida, el software que se va a producir, el tipo de herramientas utilizadas en su desarrollo y la eficacia de las mismas, etc., siendo por tanto un modelo bastante fiable en cuanto a las estimaciones que presenta, posibilitando su utilización en el resto de fases del proyecto.

Es importante señalar que los resultados de los modelos son estimaciones, y por tanto se pueden producir desviaciones con respecto al coste real del proyecto. Por tanto se debe entender que son una herramienta de estimación, y que sus predicciones, aunque apoyadas en fundamentos matemáticos, no dejan de ser estimaciones, por lo que sus resultados se entenderán como tales y deberán de ser manejados con el consiguiente cuidado.

A pesar de ello, los modelos entregan una aproximación de primera mano bastante fiable sobre la duración de un proyecto y por consiguiente sobre sus costes asociados, permitiendo de este modo concebir una idea bastante aceptable de los mismos en una fase muy temprana del desarrollo, cuando el conocimiento sobre el proyecto es tan escaso que resulta bastante difícil de estimar.

A continuación se dan una serie de definiciones y supuestos que Boehm estima se deben cumplir para que el comportamiento de los modelos Cocomo Básico y Cocomo Intermedio sea el adecuado, y cuya comprensión es, por tanto, imprescindible para su correcta utilización.

2.3 ***Definiciones y supuestos***

En los puntos que se detallan a continuación, se muestran las definiciones y supuestos que son necesarios para realizar de modo fiable las estimaciones. Es importante que la totalidad de los supuestos se verifiquen para que la estimación resulte aceptable.

2.1.1 *Base de la estimación*

Los modelos Cocomo utilizan, para la estimación de un Proyecto, el número de instrucciones "fuente" que será necesario escribir para desarrollarlo. Generalmente, se excluyen todas aquellas instrucciones que no pertenezcan estrictamente al código que va a ser ejecutado, tales como las necesarias para las pruebas y puesta a punto del software. No obstante, si se supone que el rigor de las pruebas ha de ser muy alto, las instrucciones necesarias para llevarlas a cabo se deberán de incluir en el número total de instrucciones previsto. Por "instrucciones fuente" se entienden todas aquellas instrucciones desarrolladas por el Equipo de Proyecto, y que deberán ser posteriormente convertidas en código ejecutable por el ordenador. Se excluye por tanto toda clase de comentarios existentes en los programas, así como los programas de apoyo (Utilidades) existentes, por ser siempre constantes, aunque se incluye el número de líneas de J.C.L. (Job Control Language - Lenguaje de Control de Tareas) necesarias para la ejecución y control de los programas que forman el Proyecto.

2.1.2 *Período de desarrollo cubierto por la estimación*

Los costes obtenidos como resultado de la estimación abarcan desde la fase de Diseño del producto software hasta la fase de Implantación y Pruebas, por lo que los

costes y tiempos de las demás fases del Proyecto se deberán de estimar por separado. Además, cuando se utiliza el Modelo Cocomo Básico, se obtiene también una estimación de costes de la fase inicial de Planes y Requerimientos.

2.1.3 *Unidades de medida*

Se ha evitado dar una estimación en euros (o en cualquier otra moneda) dada la fluctuación que la moneda puede presentar a lo largo del tiempo. Por tanto, las estimaciones se proporcionan en una unidad mucho más estable: la ratio Meses / hombre.

Un Mes / hombre equivale a 152 horas de tiempo laborable real. Este número de horas ha sido calculado en base a la experiencia práctica, teniendo en cuenta períodos de vacaciones, promedio de ausencias por enfermedad, permisos, etc. Resulta muy fácil obtener la equivalencia de Meses / hombre a otras unidades. En la tabla que se muestra a continuación se facilitan las equivalencias más corrientes.

A Horas / hombre	Multiplicar por 152
A Días / hombre	Multiplicar por 19
A Años / hombre	Dividir entre 12

Por otra parte, y dado que los proyectos evolucionan en el tiempo, para convertir Meses / hombre a la moneda local puede ser conveniente aplicar una ratio diferente para cada una de las Fases del Proyecto, para tener en cuenta la posible inflación en el momento del desarrollo de cada Fase y la posible diferencia de sueldo de las personas encargadas de llevarla a cabo. De esta manera se dispondrá de una estimación más ajustada.

2.1.4 *Buena dirección del Proyecto*

Otro de los supuestos que se deben de cumplir para que las estimaciones sean fiables es que la dirección del Proyecto sea "buena". Es decir, que se haga una correcta planificación y gestión del proyecto, que se apliquen los correspondientes Estándares

de Desarrollo de Proyectos, que se siga una Metodología de Desarrollo, y finalmente, que se cumplan los requisitos de Control de Calidad. Si este *entorno de calidad* de desarrollo no se alcanza, las estimaciones sufrirán desviaciones importantes, por lo que serán de escasa utilidad.

2.1.5 *Pocos cambios en los requerimientos*

El proyecto deberá de tener pocos cambios en sus requerimientos. Esto solamente se logrará como consecuencia inmediata del cumplimiento de los puntos del apartado anterior, es decir, gozando el Proyecto de una buena dirección. Lógicamente, si los cambios en los requerimientos no se controlan y éstos se disparan, la duración del Proyecto se verá tanto más afectada cuanto mayores y más tardíos sean los cambios.

El impacto de un cambio en el Proyecto crece exponencialmente dependiendo del grado de avance del mismo, haciéndose máximo cuando éste se encuentra ya en un estado de operatividad. Por tanto, es de vital importancia minimizar los cambios en los requerimientos para que las estimaciones previas se aproximen lo más posible a la realidad.

2.2 *El Modelo COCOMO BASICO*

El modelo Cocomo Básico se apoya en una ecuación para estimar el número de Meses / hombre (MH) necesarios para desarrollar la mayoría de los proyectos de software, en base al número de miles de instrucciones 'fuente' (KNINST) que tendrá el producto software terminado.

$$MH = 2.4\ (KNINST)^{1.05}$$

También presenta una ecuación para estimar el tiempo de desarrollo (TDES) de un proyecto, expresado en meses:

$$TDES = 2.5\ (MH)^{0.38}$$

Estas ecuaciones, así como las demás ecuaciones que se verán a lo largo de este capítulo, están obtenidas heurísticamente, y son fruto de la experimentación durante largos períodos de tiempo y con una gran cantidad de proyectos. No obstante es posible establecer principios matemáticos de base a partir de distribuciones estadísticas del tipo de la distribución de Rayleigh, por ejemplo.

2.2.1 *Estándares*

Para el Modelo Cocomo Básico se han estimado los siguientes estándares para relacionar el tamaño del proyecto con el número de líneas necesarias para desarrollarlo.

Proyecto pequeño	2.000 instrucciones
Proyecto intermedio	8.000 instrucciones
Proyecto mediano	32.000 instrucciones
Proyecto grande	128.000 instrucciones o mas

Además, Cocomo Básico proporciona los siguientes valores de esfuerzo y tiempo para los proyectos de tamaño pequeño, intermedio, mediano y grande:

Tamaño del Producto	Esfuerzo (Meses Hombre)	Productividad (Inst./MH)	Tiempo (Meses)	Promedio de personas
Pequeño	5,0	400	4,6	1,1
Intermedio	21,3	376	8,0	2,7
Mediano	91,0	352	14,0	6,5
Grande	302,0	327	24,0	16,0

Tabla 2.1 Esfuerzo y tiempo según el tamaño del Proyecto

Asimismo, la duración, en %, de cada una de las Fases de un Proyecto, para el Cocomo Básico, Modo Orgánico, es la siguiente:

	Tamaño del Proyecto			
	Pequeño	Intermedio	Mediano	Grande
	2K	8K	32K	128K
Esfuerzo				
Planes y Requerimientos	6%	6%	6%	6%
Diseño del Producto	16	16	16	16
Programación	68	65	62	59
Diseño Detallado	26	25	24	23
Codificación y Pruebas	42	40	38	36
Integración y Pruebas	16	19	22	25
Total	100%	100%	100%	100%
Tiempo				
Planes y Requerimientos	10%	11%	12%	13%
Diseño del Producto	19	19	19	19
Programación	63	59	55	51
Implantación y Pruebas	18	22	26	30
Total	100%	100%	100%	100%

Tabla 2.2 Porcentaje de esfuerzo por Fase

Se tendrá en cuenta que la Fase de Planes y Requerimientos no forma parte del total, estimándose aparte.

Estos valores fueron obtenidos por Boehm tras el estudio detallado de 63 proyectos incluidos en tres categorías: proyectos de negocios, proyectos científicos y proyectos de sistemas. Entre estos 63 proyectos los había de diferentes modos, desde embebidos a orgánicos (ver Modos del Modelo Cocomo, en el punto siguiente) y que habían sido desarrollados para rodar en ordenadores distintos, desde Micros a Grandes Sistemas. También escogió Boehm proyectos escritos en distintos lenguajes, contemplando Cobol, Pascal, Jovial y Ensamblador.

Nótese que los 5 Meses / hombre que Cocomo Básico estima para un proyecto pequeño incluyen documentación, enseñanza a usuarios, pruebas, cambios, etc. Es por tanto posible desarrollar un programa "personal" del mismo tamaño, sin las consideraciones anteriores, con un ratio Meses / hombre mucho menor. No obstante, como ya indicó Brooks en 1.975, se ha de tener en cuenta que "*un producto software necesita por lo menos tres veces más esfuerzo para ser completado que un programa 'personal' del mismo tamaño*".

Por ejemplo se tendrá en cuenta que los 5 Meses / hombre necesarios para un proyecto pequeño incluyen tanto los tiempos de diseño y desarrollo, como las pruebas, (individuales, de integración, de control de calidad, etc.) los tiempos necesarios para depuración y puesta a punto del proyecto, el desarrollo de toda la documentación (manuales del usuario, manuales técnicos, etc.) y el tiempo de las pruebas en "paralelo" si existen. Es decir, todo el tiempo necesario para entregar el proyecto totalmente depurado y funcionando con los niveles de calidad establecidos y cumpliendo, por supuesto, con los requisitos previamente acordados con el usuario.

Si se tienen en consideración todos estos factores, el esfuerzo indicado de 5 Meses / hombre para un proyecto de 2.000 instrucciones no parecerá exagerado.

2.2.2 *Distintos Modos del Modelo Cocomo Básico*

El Modelo Cocomo Básico dispone de tres modos para las estimaciones de proyectos. Estos tres modos difieren únicamente en las fórmulas empleadas y dependen del tipo de proyecto que se va a desarrollar y de sus características principales, así como de las peculiaridades de las personas que van a trabajar en él.

2.2.2.1 Cocomo Básico: Modo Orgánico

El primer modo que presenta el modelo Cocomo Básico es el Modo Orgánico. Las ecuaciones para este modo, tanto para el esfuerzo (Meses / hombre) como para el tiempo de desarrollo son las siguientes:

$$MH = 2.4\ (KNINST)^{1.05}$$

$$TDES = 2.5\ (MH)^{0.38}$$

Este modo está aconsejado cuando se presentan las siguientes características en el proyecto y en el equipo de trabajo:

a) El tamaño del proyecto es pequeño - mediano. (Menor o igual a 50.000 instrucciones)

b) El Equipo de trabajo es relativamente pequeño (3-5 personas).

c) El entorno de trabajo es conocido y estable.

d) El Equipo dispone de gran experiencia en todos los Sistemas relacionados con el que se está desarrollando así como un alto conocimiento de la influencia del Sistema a desarrollar en los objetivos de la Organización, y además existe poca o nula interacción del Sistema a desarrollar con los demás Sistemas.

Todo esto implica que habrá poca "sobrecarga" debido a las comunicaciones, ya que el Equipo tiene amplios conocimientos no solo sobre el Sistema a desarrollar, sino

sobre toda el área de influencia del mismo, así como un fuerte conocimiento de la Organización y de sus objetivos. Por lo tanto la sobrecarga debida a comunicaciones será, como queda dicho, muy escasa o prácticamente nula, ya que no es necesario preguntar nada a nadie. Como consecuencia inmediata de lo anterior se desprende que la productividad del Equipo será muy alta, y que realizar cambios en las especificaciones (si fuese necesario) sería relativamente fácil dado el bajo impacto en el resto de los Sistemas que ya están funcionando en la Organización, y por tanto dichos cambios serán fácilmente negociables y asumibles por el Equipo de Trabajo.

2.2.2.2 Cocomo Básico: Modo Híbrido

Este modo constituye un nivel intermedio entre el Modo Orgánico anteriormente descrito y el Modo Embebido que se verá a continuación. Por ello, tanto las características del Sistema a desarrollar como del Equipo de trabajo también serán intermedias, teniéndose así una mezcla de las características de los otros dos modos. Las ecuaciones son las siguientes:

$$MH = 3.0\ (KNINST)^{1.12}$$

$$TDES = 2.5\ (MH)^{0.35}$$

Por su parte, para utilizar este modo, los Proyectos deberán de cumplir los siguientes requisitos:

a) Proyectos de un tamaño hasta 300.000 instrucciones.

b) Nivel medio de experiencia del Equipo con los Sistemas relacionados con el que se está desarrollando.

c) Mezcla en el Equipo de gente con experiencia y gente sin experiencia.

d) Además, la experiencia puede tenerse en algunos aspectos del Sistema pero no en otros.

Como ejemplo típico de Proyecto de estas características podríamos citar un Sistema Transaccional, con algunas interfases rígidas (y por tanto con características no modificables, o al menos no fácilmente modificables), como pueden ser interfases

con hardware de terminales, por ejemplo, y otras interfases más flexibles, como por ejemplo interfases con la consola del operador, informes, etc. Las primeras presentan un protocolo rígido, que hay que seguir estrictamente, por lo que nuestro Sistema no podrá permitirse desviaciones del estándar en la programación de dichas interfases, mientras que las segundas son más flexibles, y admitirán cambios en la programación y en la estructura de un modo más sencillo.

2.2.2.3 Cocomo Básico: Modo Embebido

El tercer modo que presenta el modelo Cocomo Básico es el que trata con los Sistemas más complejos, de mayor duración y más rígidos, por lo que sus ecuaciones se adaptan para actuar en consecuencia. Son las siguientes:

$$MH = 3.6\ (KNINST)^{1.20}$$

$$TDES = 2.5\ (MH)^{0.32}$$

En los tipos de Sistemas susceptibles de estimación con el Modo Embebido, se presentan las siguientes características más destacables:

a) Proyectos de cualquier tamaño, tanto grandes como medianos o pequeños.

b) Se impone la necesidad de operar con restricciones, debido fundamentalmente a la propia rigidez del Sistema a desarrollar y de sus interfases.

c) El Sistema funcionará dentro de un entorno fuertemente acoplado de procedimientos hardware, software, regulaciones y procedimientos operativos.

d) El coste de cambios de entorno es tan elevado que se considerará que dicho entorno es prácticamente inmodificable.

Como consecuencia, el Sistema deberá de cumplir *rigurosamente* las especificaciones, y no se podrán negociar alternativas de desarrollo de modo fácil. Por todo ello, la productividad será baja y existirá mucha sobrecarga debida a las

comunicaciones de los miembros del Equipo con el resto de la Organización y entre sí, ya que las consultas con especialistas serán muy frecuentes.

Por otra parte, en este tipo de proyectos, una vez completado el diseño del Sistema será necesario disponer de un gran número de programadores trabajando en paralelo, para evitar que el Proyecto se alargue en demasía, ya que de ser así surgirán nuevos cambios en las especificaciones, lo que inevitablemente alargará aún más el Proyecto, y este finalmente estará obsoleto cuando se entregue.

2.2.3 *La Distribución de Rayleigh aplicada al Modo Orgánico*

La media del nivel de esfuerzo estimado (en Meses / hombre), para proyectos que se evalúen con el Modo Orgánico, sigue una curva característica, (Ver Ola de Rayleigh, punto 2.9.2) tal y como se muestra en la siguiente figura.

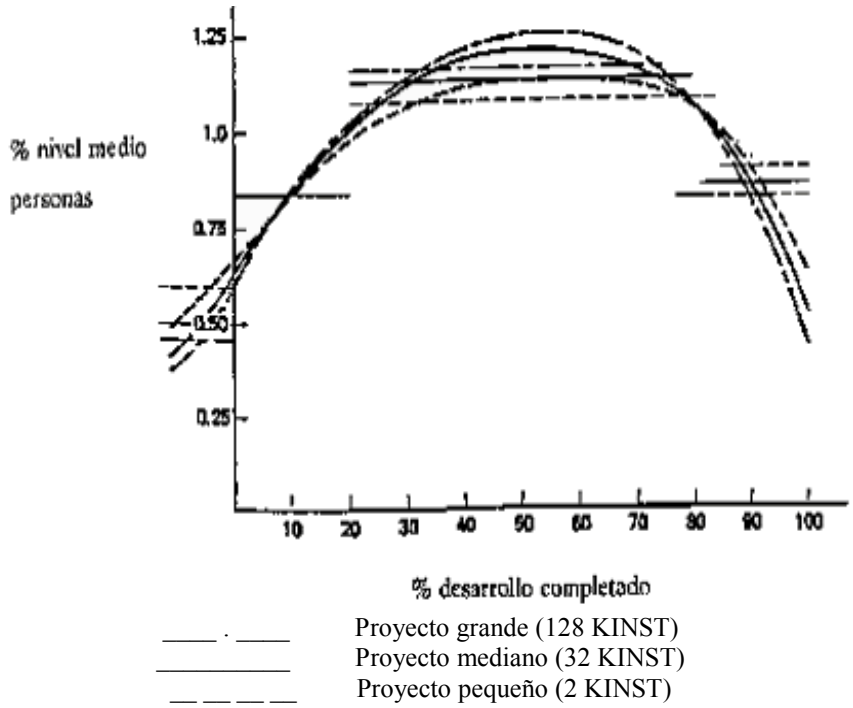


Fig. 2.1 Nivel de esfuerzo para el Modelo Cocomo Básico, Modo Orgánico

Por otra parte, es bien conocido que aumentar el número de personas que trabajan en un proyecto no siempre conduce a una más rápida finalización del mismo. Es más,

añadir personas a un proyecto atrasado, de acuerdo con la Ley de Brooks, normalmente suele retrasarlo más aún.

Debemos por tanto ser capaces de calcular el número óptimo de personas necesarias, trabajando a tiempo completo, para desarrollar el proyecto. Afortunadamente, existe una función matemática que permite calcular este número óptimo de personas a partir de los Meses / hombre de esfuerzo estimados. Esta función es la Función de Distribución de Rayleigh, la cual expresada en términos de variables de Cocomo, se representa por la fórmula:

$$PTC = MH(t\frac{t}{t_p^2})e^{-(\frac{t^2}{2t_p^2})}$$

donde:

PTC Número de Personas a Tiempo Completo, óptimas para el desarrollo del Proyecto

MH meses / hombre

t tiempo, en meses

t_p mes en que el Proyecto alcanza su nivel de esfuerzo más alto (tiempo de pico)

Tal y como ha sido demostrado por Norden [Norden, 1958; Norden, 1970] la Distribución de Rayleigh proporciona una buena estimación a la distribución del esfuerzo para distintos tipos de actividades. Esta función se muestra a continuación, para un Proyecto mediano (32 KINST) con 91 meses / hombre de esfuerzo requerido y un nivel de esfuerzo máximo en el séptimo mes (t_p=7)

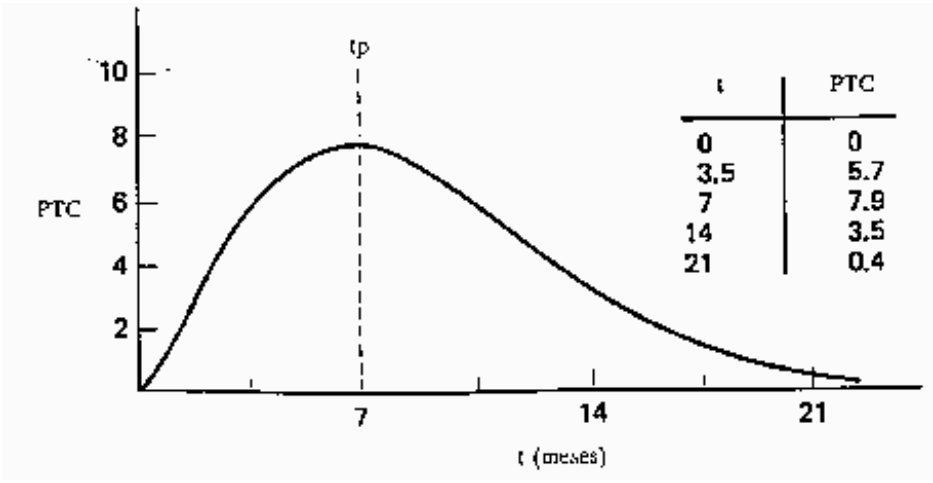


Fig. 2.2 Esfuerzo para un Proyecto de 32000 instrucciones (32 KINST)

2.2.4 *Representaciones gráficas del Modelo Cocomo Básico*

Si representamos de modo gráfico las ecuaciones del modelo Cocomo Básico en sus tres modos de desarrollo, Orgánico, Híbrido y Embebido, se obtienen las siguientes gráficas que permiten obtener una información más visual del esfuerzo y tiempo necesarios para la consecución del Proyecto.

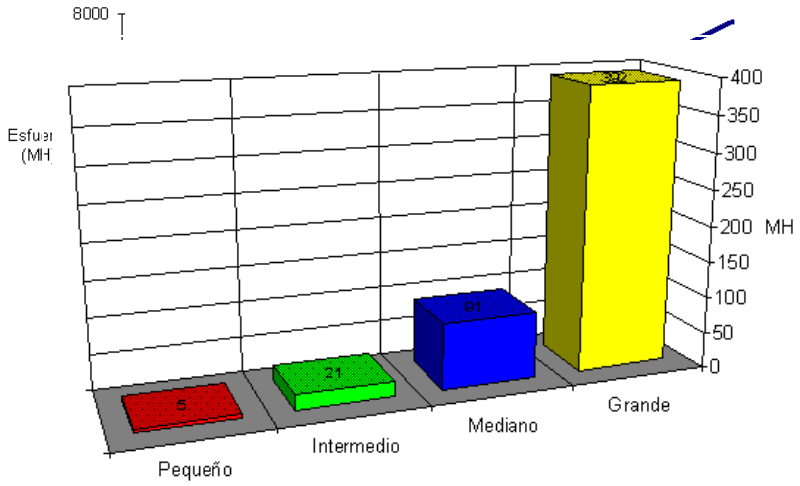
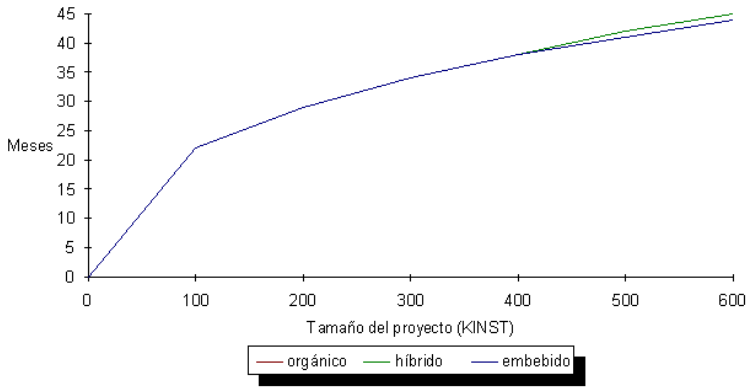


Fig. 2.3 Nivel de esfuerzo según el tamaño del Proyecto



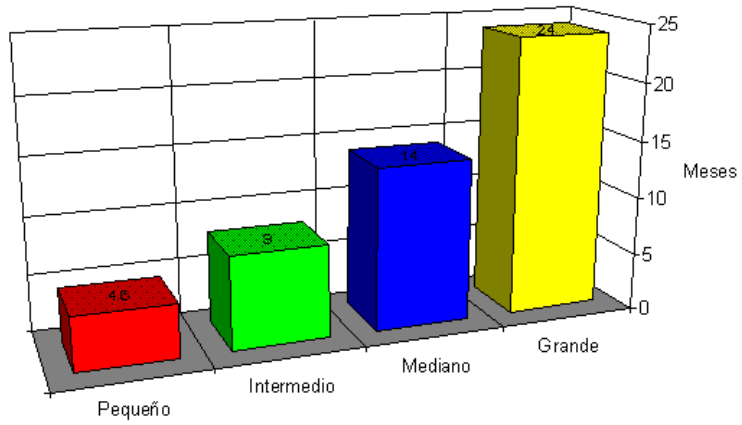


Fig. 2.4 Tiempo de desarrollo según el tamaño del Proyecto

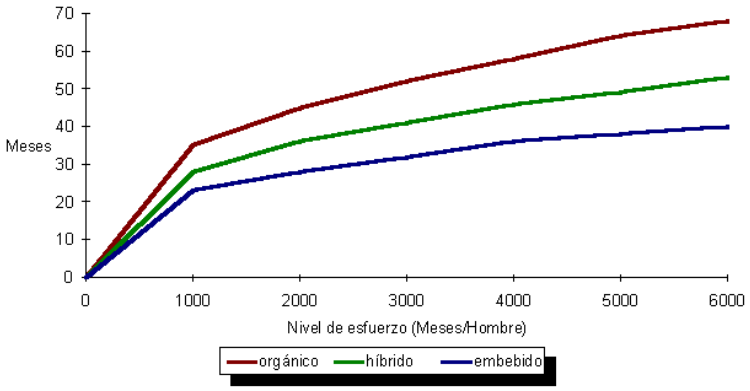


Fig. 2.5 Tiempo de desarrollo según el nivel de esfuerzo

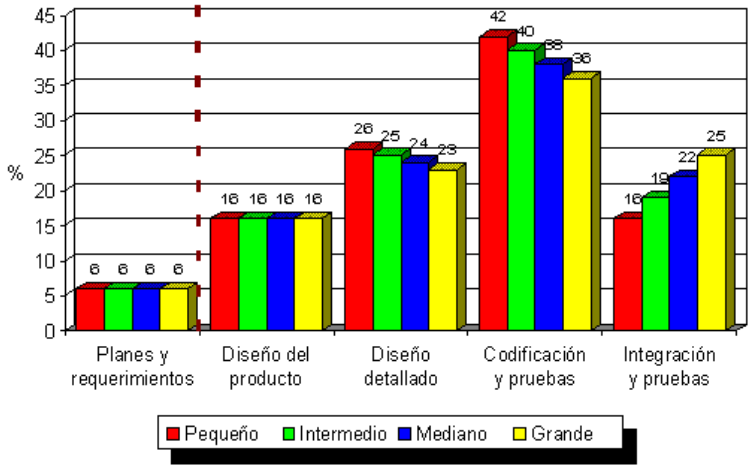


Fig. 2.6 Porcentaje de esfuerzo por Fase, según el tamaño del Proyecto

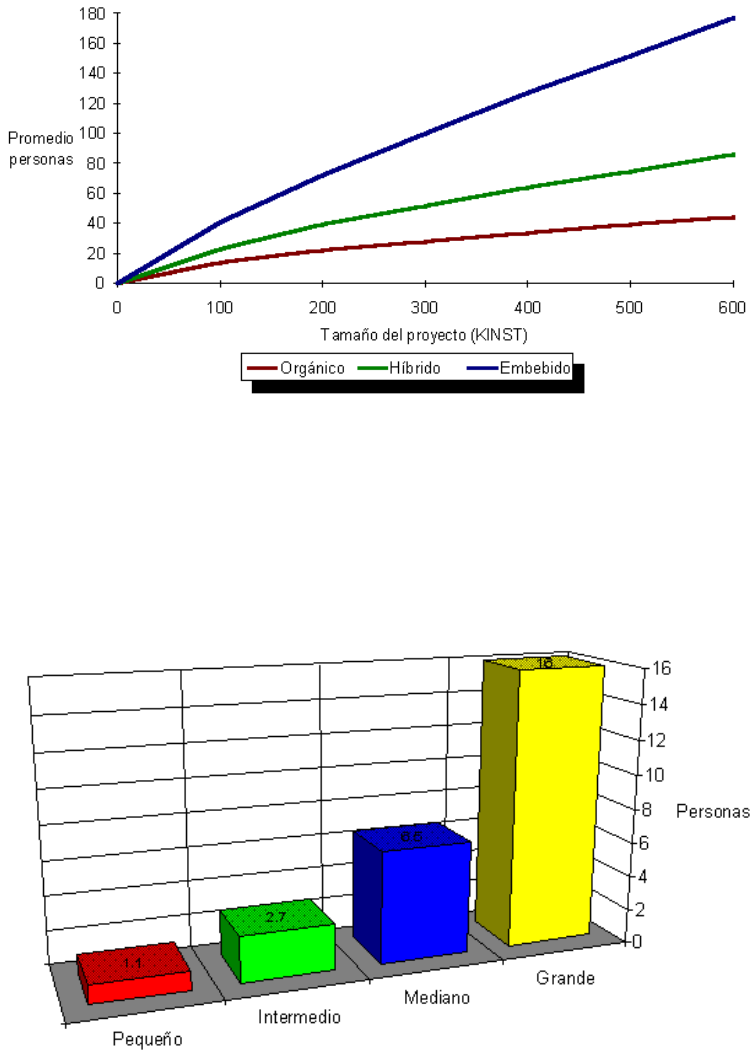


Fig. 2.7 Promedio de personas según el tamaño del Proyecto

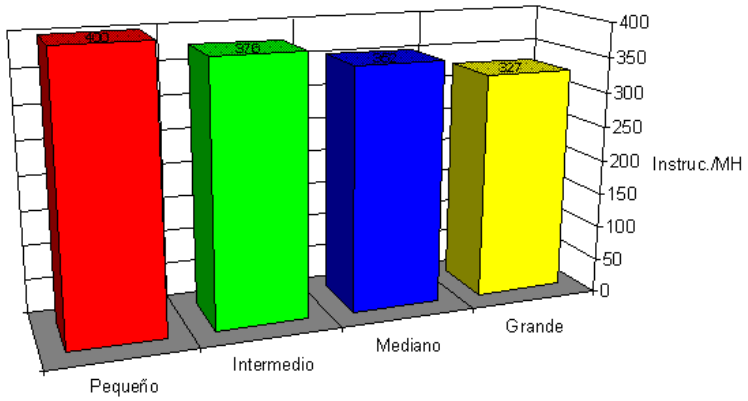


Fig. 2.8 Productividad según el tamaño del Proyecto

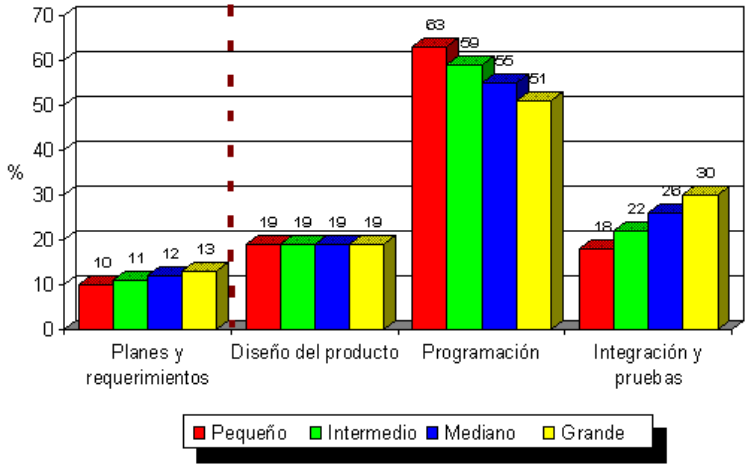


Fig. 2.9 Porcentaje de tiempo por Fase, según el tamaño del Proyecto

2.3 *Estimación de Costes durante el Análisis Previo: Un ejemplo*

A modo de ejemplo, se estima en este apartado un supuesto Proyecto de Contabilidad, para el que se ha calculado un tamaño de 28.000 líneas, y para el que se ha elegido el Modelo Básico, Modo Orgánico por realizarse la estimación durante la fase de Análisis Previo, donde la mayoría de los factores que inciden en el Proyecto son todavía desconocidos, y por darse las características de Proyecto y Equipo válidas para el Modo Orgánico.

Así, tenemos:

PROYECTO:	**Contabilidad**
Número de líneas estimado:	**28.000**
Modelo Utilizado:	**Cocomo Básico, Modo Orgánico**

Las fórmulas que el modelo Cocomo Básico proporciona para el cálculo del esfuerzo (expresado en Meses / hombre) y del tiempo de desarrollo (en meses) son las siguientes:

$$MH = 2.4\ (KNINST)^{1.05}$$
$$TDES = 2.5\ (MH)^{0.38}$$

Después de introducir el número de líneas, expresado en miles, los resultados obtenidos para la totalidad del Proyecto son los siguientes:

Total Proyecto:

Meses / hombre	79.38	==> 79
Productividad (Instrucciones / Mes-hombre)	352.72	==> 353
Tiempo (Meses)	13.18	==> 13
Promedio personas full time	6.02	==> 6

Además, el modelo proporciona una estimación de la Fase de Planes y Requerimientos, (cuyos resultados no están englobados en los totales del Proyecto

anteriores) así como un desglose de dichos resultados totales repartidos entre las fases de Diseño del Sistema, Programación e Implementación y Pruebas. La distribución del esfuerzo (en Meses / hombre) y del tiempo (en meses) en la fase previa de Planes y Requerimientos y en el resto de fases del Proyecto, es la siguiente:

Planes y Requerimientos: (No incluido en el total del proyecto)

Meses / hombre	4.76 ==> 5
Meses	1.58 ==> 2
Promedio personas full time	3.01 ==> 3

Diseño del Producto:

Meses / hombre	12.70
Meses	2.50
Promedio personas full time	5.07

Programación (total):

Meses / hombre	49.45
Meses	7.30
Promedio personas full time	6.77

Programación: Diseño

Meses / hombre	18.91
Meses	2.79
Promedio personas full time	6.77

Programación: Codificación y pruebas

Meses / hombre	30.54
Meses	4.51
Promedio personas full time	6.77

Implantación y Pruebas:

Meses / hombre	17.23
Meses	3.38
Promedio personas full time	5.11

De los resultados anteriores se desprende que será necesario un tiempo de 2 meses con 3 personas a tiempo completo para la fase previa de Planes y Requerimientos, mientras que para el desarrollo total del Proyecto propiamente dicho se estiman 13 meses, con 6 personas trabajando full time. Para la obtención de los resultados anteriores, el Modelo Cocomo utiliza los coeficientes correctores C y C1 cuyos valores se obtienen de la Tabla 2.2.

Estos coeficientes correctores actúan sobre los Meses / hombre obtenidos y sobre el tiempo de desarrollo estimado del software, de acuerdo con las siguientes fórmulas:

$$MH = \frac{MH * C}{100} \qquad TDES = \frac{TDES * C1}{100}$$

Y el número de personas que se van a necesitar a tiempo completo para el desarrollo del software será el número de meses-hombre necesarios dividido entre el tiempo de desarrollo.

$$\text{Pr}omedioPersonas = \frac{MH}{TDES}$$

Interpolación de valores:

El valor de C1 utilizado en la Fase de Planes y Requerimientos es aproximado. Para obtener el valor exacto sería necesario calcular C1 teniendo en cuenta los tramos mostrados en la Tabla 2.2. Por ejemplo, el valor indicado de C1 para un proyecto de 32K instrucciones es de 12 y para un proyecto de 128K es de 13. Si se quiere obtener el valor de C1 para un proyecto de 80K, aplicando la ecuación de la recta por dos puntos tendríamos que:

$$\frac{x - x0}{x1 - x0} = \frac{y - y0}{y1 - y0}$$

donde: x0 = 32 x1 = 128 y0 = 12 y1 = 13

Entonces, para x = 80 se tiene que y = 12,5

En la Fase de Programación (Diseño) sucede lo mismo con los coeficientes correctores del desglose. El valor mostrado en la Tabla 2.6 es 26/68 y corresponde a un proyecto de 2K. No obstante, si analizamos de nuevo la Tabla 2.2 tenemos que los valores para los cuatro tamaños de proyecto proporcionados son 26/28, 25/65, 24/62 y 23/59, y los resultados de los cocientes son, respectivamente, 0.3823529, 0.3846153, 0.3870967 y 0.3898305, por lo que si no se quiere interpolar el valor exacto, es perfectamente válido tomar 0.38 en todos los casos. De la misma manera, en la Fase de Programación (Pruebas) se puede tomar 0.61 como valor aproximado para todos los cálculos.

2.4 *El Modelo COCOMO INTERMEDIO*

El siguiente Modelo de Estimación que aporta COCOMO es el Modelo Cocomo Intermedio. Este modelo puede utilizarse cuando ya se ha avanzado en el desarrollo del proyecto, y por tanto se tiene más información sobre el mismo. Normalmente en la Fase de Análisis Funcional pueden ya realizarse estimaciones empleando este modelo.

Las ecuaciones de Boehm para Cocomo Intermedio son las siguientes:

Modo Orgánico:

$$MH = 3.2\ (KNINST)^{1.05}$$
$$TDES = 2.5\ (MH)^{0.38}$$

Modo Híbrido:

$$MH = 3.0\ (KNINST)^{1.12}$$
$$TDES = 2.5\ (MH)^{0.35}$$

Modo Embebido:

$$MH = 2.8\ (KNINST)^{1.20}$$
$$TDES = 2.5\ (MH)^{0.32}$$

Además del cambio en las ecuaciones anteriores, el Cocomo Intermedio contribuye al cálculo del esfuerzo necesario para el desarrollo de un Proyecto y del tiempo asociado a dicho desarrollo incorporando una serie de Atributos, en un total de 15, que actúan como modificadores o coeficientes correctores de los resultados previamente obtenidos. Así, además del número de instrucciones "fuente" necesarios para la obtención de las estimaciones en el modelo Cocomo Básico, el Modelo Cocomo Intermedio dispone de los Atributos que a se indican en el punto siguiente. Se tendrá en cuenta que cada Atributo proporciona un multiplicador (o coeficiente) que hará variar positiva o negativamente el nivel de esfuerzo y de tiempo requeridos para el desarrollo del Proyecto.

A continuación se detallan los diferentes Atributos y sus correspondientes coeficientes, así como la Escala de Valores para estimar la clase o tipo de atributo, que se deberán de tener en consideración para llevar a cabo una estimación de costes utilizando el Modelo Cocomo Intermedio.

2.4.1 *Atributos del Modelo Cocomo Intermedio*

Como se citaba anteriormente, para determinar la estimación de costes con el Modelo Cocomo Intermedio, se incorporan 15 variables (o Atributos) que van a incidir de modo importante en la duración del Proyecto, y por lo tanto en los costes del mismo. Estos Atributos se pueden agrupar en cuatro categorías y son los siguientes:

2.4.1.1 Atributos del Producto

FIAB	Fiabilidad requerida del Sistema a desarrollar
TABD	Tamaño de la Base de Datos a utilizar
COMP	Complejidad del Producto

2.4.1.2 Atributos del Ordenador

TIEM	Tiempo de ejecución
ALMA	Almacenamiento principal
VIRT	Volatilidad virtual del ordenador [1]
TORD	Tiempo de ordenador necesario

2.4.1.3 Atributos Personales

CAPA	Capacidad de los analistas
EXPA	Experiencia en Aplicaciones
CAPP	Capacidad de los programadores
EXPV	Experiencia virtual con el ordenador
EXPL	Experiencia con el lenguaje de programación

[1] Conjunto de Hardware y Software necesario para realizar un trabajo dado

2.4.1.4 Atributos del Proyecto

TADP	Uso de técnicas avanzadas de programación
UHER	Uso de herramientas software
TDES	Tiempo de desarrollo requerido

2.4.2 *Coeficientes para el cálculo del esfuerzo*

Cada uno de los Atributos expuestos en el punto anterior nos proporciona un multiplicador que hará variar de modo positivo o negativo el esfuerzo requerido (en Meses / hombre) para el desarrollo del Proyecto al utilizar el Modelo Cocomo Intermedio. Estos multiplicadores, o coeficientes, están obtenidos a través de la experiencia lograda en situaciones similares de desarrollo de proyectos, y se muestran en la tabla 2.3 que figura a continuación.

C O E F I C I E N T E S

Atributos	Muy Bajo	Bajo	Nominal	Alto	Muy Alto	Extra Alto
Del Producto:						
FIAB	0.75	0.88	1.00	1.15	1.40	
TABD		0.94	1.00	1.08	1.16	
COMP	0.70	0.85	1.00	1.15	1.30	1.65
Del Ordenador:						
TIEM			1.00	1.11	1.30	1.66
ALMA			1.00	1.06	1.21	1.56
VIRT		0.87	1.00	1.15	1.30	
TORD		0.87	1.00	1.07	1.15	
Personales:						
CAPA	1.46	1.19	1.00	0.86	0.71	
EXPA	1.29	1.13	1.00	0.91	0.82	
CAPP	1.42	1.17	1.00	0.86	0.70	
EXPV	1.21	1.10	1.00	0.90		
EXPL	1.14	1.07	1.00	0.95		
Del Proyecto:						
TADP	1.24	1.10	1.00	0.91	0.82	
UHER	1.24	1.10	1.00	0.91	0.83	
TDES	1.23	1.08	1.00	1.04	1.10	

Tabla 2.3 Coeficientes según la clase de atributos

2.4.3 *Estimación de la Escala de Valores de los distintos Atributos*

Para tener una idea de una Escala de Valores que puede ser utilizada por el Equipo de Proyecto a la hora de determinar el tipo o la clase de atributo, se sugiere tener en cuenta las consideraciones especificadas en las tablas 2.4 y 2.5 que se detallan en las páginas siguientes.

Atributos	Muy Bajo	Bajo	Nominal
Del Producto:			
FIAB	Efecto: Ligera inconveniencia	Pérdidas fácilmente recuperables	Moderado: Pérdidas recuperables
TABD		{Bytes BD} OVER NINST < 10	10 < {Bytes BD} over NINST < 100
COMP	Ver Tabla 2.5		
Del Ordenador:			
TIEM			<= 50% uso del tiempo de ejecución disponible
ALMA			<= 50% uso del almacenamiento disponible
VIRT		Cambios mayores cada 12 meses; cambios menores cada mes	Cambios mayores cada 6 meses; cambios menores cada 2 semanas
TORD		Interactivo	Promedio de tiempo < 4 horas

Tabla 2.4 Escala de Valores del coste del software

Atributos	Alto	Muy Alto	Extra Alto
Del Producto:			
FIAB	Altas pérdidas financieras	Riesgo de vidas humanas	
TABD	100<={Bytes BD} OVER NINST < 1000	{Bytes BD} OVER NINST >= 1000	
COMP	Ver Tabla 2.5		
Del Ordenador:			
TIEM	70% uso del tiempo de ejecución disponible	85% uso del tiempo de ejecución disponible	95% uso del tiempo de ejecución disponible
ALMA	70% uso del almacenamiento disponible	85% uso del almacenamiento disponible	95% uso del almacenamiento disponible
VIRT	Cambios mayores cada 2 meses; cambios menores cada semana	Cambios mayores cada 2 semanas; cambios menores cada 2 días	
TORD	De 4 a 12 horas	Mas de 12 horas	

Tabla 2.4 Escala de valores del coste del software (Cont.)

Atributos	Muy Bajo	Bajo	Nominal
Personales:			
CAPA	15 percentil (1)	35 percentil	55 percentil
EXPA	<= 4 meses	1 año	3 años
CAPP	15 percentil	35 percentil	55 percentil
EXPV	<= 1 mes	4 meses	1 año
EXPL	<= 1 mes	4 meses	1 año
Del Proyecto:			
TADP	No se usan	Se comienzan a usar	Algún uso moderado
UHER	Microprocesadores	Minis	Midis, Maxis, PC's
TDES	75% del nominal	85% del nominal	100% del nominal

(1) Criterio de evaluación del Equipo: habilidad y eficiencia en programación y en comunicarse y cooperar.

Tabla 2.4 Escala de valores del coste del software (Cont.)

Atributos	Alto	Muy Alto	Extra Alto
Del Producto:			
CAPA	75 percentil (1)	90 percentil	
EXPA	6 años	12 años	
CAPP	75 percentil	90 percentil	
EXPV	3 años		
EXPL	3 años		
Del Proyecto:			
TADP	Uso general	Uso rutinario	
UHER	Fuertes herramientas de 'test'	Herramientas de diseño, documentación, gestión, etc.	
TDES	130% del nominal	160% del nominal	

(1) Criterio de evaluación del Equipo: habilidad y eficiencia en programación y en comunicarse y cooperar.

Tabla 2.4 Escala de valores del coste del software (Cont.)

Escala	Operaciones de Control	Operaciones de Cálculo
Muy Baja	Código simple con pocos operadores de Programación Estructurada no anidados. Ej.; DO's, CASE's, IF-THEN-ELSE's.	Evaluación de expresiones simples. Ej.: A=B+C*(D+E)
Baja	Operadores de Programación Estructurada anidados. Gran mayoría de instrucciones simples.	Evaluación de expresiones de nivel moderado. Ej.: D=SQR(B**2-4*A)
Nominal	Anidamiento simple en la mayoría de los casos. Algún control entre módulos. Uso de Tablas de Decisión.	Uso de matemáticas estándar y rutinas estadísticas. Operaciones básicas con matrices y vectores.
Alta	Operadores de Programación estructurada fuertemente anidados. Control de Colas. Considerable control entre módulos.	Análisis numérico básico. Interpolación de variaciones múltiples. Ecuaciones diferenciales básicas.
Muy Alta	Codificación recursiva. Manipulación de interrupciones de prioridad fija.	Ecuaciones matriciales. Ecuaciones diferenciales parciales.
Extra Alta	Distribución de recursos múltiples con cambios dinámicos de prioridad. Control a nivel de microcódigo.	Análisis de precisión de datos estocásticos y con ruido.

Tabla 2.5 Complejidad del Software: criterios de evaluación de módulos, según su clase.

Escala	Operaciones con Dispositivos	Operaciones de Gestión
Muy Baja	Leer y escribir con formatos simples.	Tablas sencillas residentes en memoria principal.
Baja	Conocimiento no necesario de un procesador o dispositivo de I/O en particular.	Descripción sencilla de Ficheros, sin cambios en la estructura de los datos y sin Ficheros intermedios ni edición de Ficheros.
Nominal	Procesos I/O, incluyendo selección de dispositivos, chequeo de 'status' y proceso de errores.	Entradas multifichero y salidas monofichero. Cambios estructurales simples. Ediciones simples.
Alta	Operaciones a nivel I/O físico (cambios de direcciones de almacenamiento físico: SEEK's, READ's, etc.)	Subrutinas de propósito especial, activadas por el contenido de un flujo de datos. Reestructuración compleja de los datos a nivel de registro.
Muy Alta	Rutinas para diagnóstico de interrupciones; masking. Manipulación de comunicaciones.	Rutina paramétrica generalizada para estructuración de Ficheros. Construcción de Ficheros, proceso de comandos, etc.
Extra Alta	Codificación de dispositivos con dependencia del tiempo. Operaciones microprogramadas.	Estructuras relacionales dinámicas, altamente acopladas. Gestión de datos en lenguaje natural.

Tabla 2.5 Complejidad del Software: criterios de evaluación de módulos, según su clase (Cont.)

2.5 Representaciones gráficas del Modelo Cocomo Intermedio

Representando gráficamente las ecuaciones de Boehm para el modelo Cocomo Intermedio, Modo Orgánico, podemos observar cómodamente los distintos valores de esfuerzo y tiempo necesarios para el desarrollo de un Proyecto. A continuación se muestran algunas gráficas a tal efecto.

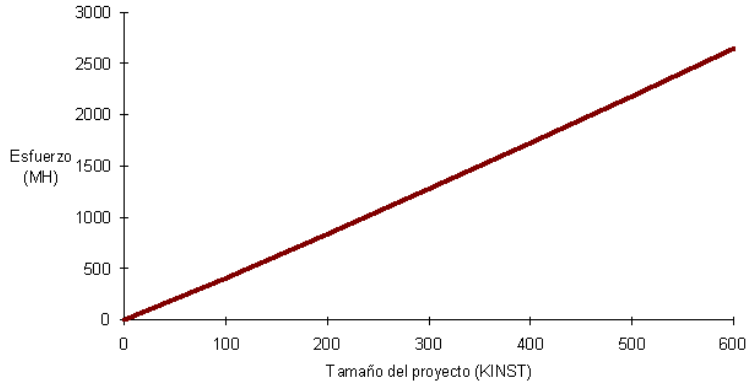


Fig. 2.10 Nivel de esfuerzo según el tamaño del Proyecto

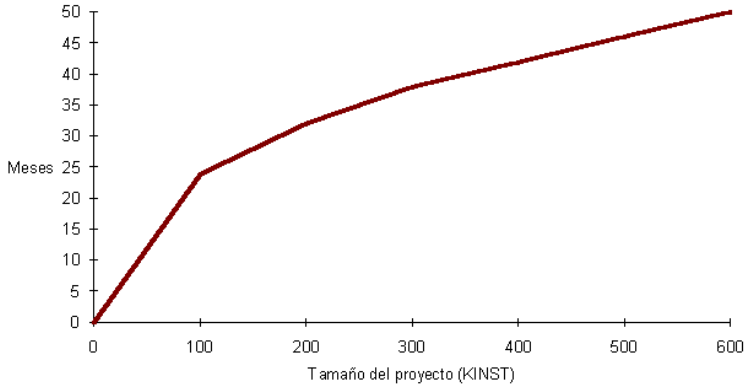


Fig. 2.11 Tiempo de desarrollo según el tamaño del Proyecto

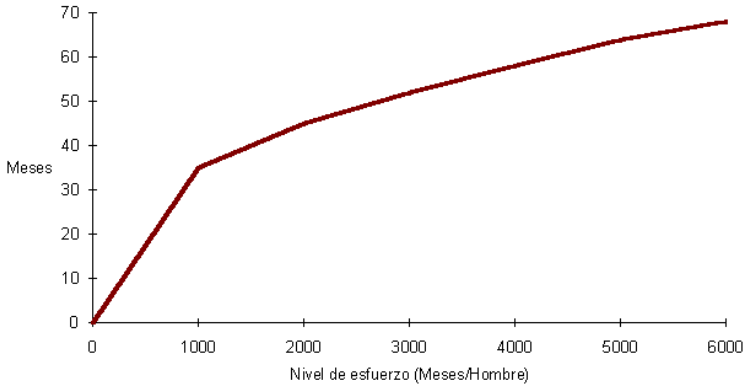


Fig. 2.12 Tiempo de desarrollo según el nivel de esfuerzo

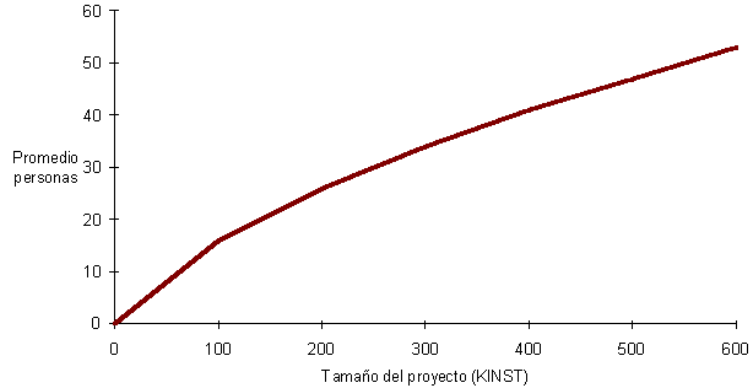


Fig. 2.13 Promedio de personas según el tamaño del Proyecto

2.6 *Estimación de Costes durante el Análisis Funcional: Ejemplos*

Con supuestos similares a los de la estimación realizada con el Cocomo Básico, se estima de nuevo el Proyecto de Contabilidad durante la fase de Análisis Funcional. Se supone ahora que el Proyecto consta de dos partes: una primera de Programación de Interfases, para la que se estiman 7.000 líneas de código, y una segunda parte de Programación de la Contabilidad propiamente dicha, con 21.000 líneas fuente, para la que se va a aprovechar software existente, calculándose modificar un 25% del diseño original, un 45% del código y un 35% de la integración. Así, se tiene:

PROYECTO:	**Contabilidad**
Número de líneas estimado:	**28.000**
Modelo Utilizado:	**Cocomo Intermedio, Modo Orgánico**

Estimación Previa:

Las fórmulas que el modelo Cocomo Intermedio proporciona para el cálculo del esfuerzo, (expresado en Meses / hombre), del tiempo de desarrollo (Meses), de la productividad (Instrucciones por MH) y de las personas necesarias a tiempo completo son las siguientes:

$$MH = 3.2\ (KNINST)^{1.05}$$
$$TDES = 2.5\ (MH)^{0.38}$$
$$\text{Instrucciones / MH} = NTINS / MH \text{ (nominales)}$$
$$PTC = MH / TDES$$

Así, aplicando las fórmulas anteriores sobre el número total de líneas estimado (28000) expresado en miles en el primer caso, se tiene la siguiente estimación previa:

Nº Total de Instrucciones (NTINS)	28000
Nº Total de Instrucciones en miles (KNTINS)	28
Meses / hombre nominales	105
Instrucciones (nominales) /MH	266
Tiempo de Desarrollo en meses (TDES)	15
Personas a tiempo completo (PTC)	7

***Estimación Ajustada*:**

El siguiente paso es calcular el Factor de Ajuste de Adaptación (FAA) de ambos Componentes. Este factor se calcula como:

$$FAA = \frac{0.4 \text{ x } \%\text{Diseño Modificado} + 0.3 \text{ x } \%\text{ Código} + 0.3 \text{ x } \%\text{ Integración}}{100}$$

Lo que proporciona un FAA de 1 para la Programación de Interfases, ya que no va a reutilizar software existente y no necesita un Factor de Ajuste de Adaptación, y un FAA de 0.34 para la Programación de la Contabilidad, (que sí va a reutilizar software) el cual va a reducir su número de instrucciones a 7140. A continuación se eligen los valores de los Atributos que van a incidir en la estimación ajustada. En este caso los valores elegidos fueron:

Coeficientes de Ajuste

Componente: Programación de Interfases

	FIAB	Alto		CAPA	Alto
Producto	TABD	Bajo		EXPA	Nominal
	COMP	Alto	Personales	CAPP	Nominal
				EXPV	Bajo
				EXPL	Nominal
	TIEM	Alto			
Ordenador	ALMA	Nominal		TADP	Alto
	VIRT	Nominal	Proyecto	UHER	Alto
	TORD	Bajo		TDES	Nominal

Componente: Programación de la Contabilidad

	FIAB	Bajo		CAPA	Alto
Producto	TABD	Bajo		EXPA	Bajo
	COMP	Muy Alto	Personales	CAPP	Alto
				EXPV	Bajo
				EXPL	Bajo
	TIEM	Alto			
Ordenador	ALMA	Nominal		TADP	Alto
	VIRT	Nominal	Proyecto	UHER	Alto
	TORD	Bajo		TDES	Nominal

Estos valores, según la Tabla 2.3, proporcionan los coeficientes de ajuste que se muestran a continuación, para cada uno de los componentes de la Aplicación.

	Programación Interfases	Programación Contabilidad
NINST	7000	21000
FAA	1.00	0.34
FIAB	1.15	0.88
TABD	0.94	0.94
COMP	1.15	1.30
TIEMP	1.11	1.11
ALMA	1.00	1.00
VIRT	1.00	1.00
TORD	0.87	0.87
CAPA	0.86	0.86
EXPA	1.00	1.13
CAPP	1.00	0.86
EXPV	1.10	1.10
EXPL	1.00	1.07
TADP	0.91	0.91
UHER	0.91	0.91
TDES	1.00	1.00
FAE	0.94	0.85

Tabla 2.6 Coeficientes correctores por componente y atributo para el Proyecto de Contabilidad

Donde FAE es el Factor de Ajuste de Esfuerzo, calculado como el producto de los 15 coeficientes anteriores, que va a reducir los Meses / hombre nominales de cada Componente.

Finalmente, y tras aplicar los coeficientes de ajuste, y volver a aplicar las fórmulas del Modelo Cocomo Intermedio, los resultados finales de la estimación fueron:

ESTIMACION FINAL

	Componente	N° Inst.	FAE	MHnom	MHajust	N° Inst/MH
1.	Prog.Interfases	7.000	0,94	25	23	304
2.	Prog.Contabldad	7.140	0,84	25	21	340
	Totales:	14.140			44	321

ESTIMACION FINAL AJUSTADA

N° Total de Instrucciones	14.140
Meses-Hombre ajustados	44
N° Instrucciones / Mes-Hombre ajustado	321
N° Personas Full-time	4
Meses	11

En esta estimación, los coeficientes correctores elegidos para cada uno de los componentes que intervienen, han provocado unos resultados diferentes de los realizados con el modelo Cocomo Básico. Han influido favorablemente la alta Capacidad de Analistas y Programadores y el Tiempo de Ordenador, entre otros, y negativamente la Fiabilidad requerida del producto, la Complejidad de ambos componentes, el Tiempo de Ejecución necesario, etc. Es de notar que globalmente han producido un Factor de Ajuste de Esfuerzo (FAE) de 0.94 en la Programación de Interfases y de 0.85 en la Programación de la Contabilidad.

Ejemplo 2:

Veamos ahora, a modo de ejercicio, otra estimación utilizando el Modelo Cocomo Intermedio y realizando los cálculos a mano, con los siguientes supuestos:

Se desea estimar el esfuerzo necesario y tiempo de desarrollo de una Aplicación de Control de Procesos de una Refinería, utilizando el Modelo Cocomo Intermedio, Modo Orgánico. Para facilitar la estimación global de la Aplicación, se ha descompuesto el software en tres componentes independientes, cuyas estimaciones individuales se indican a continuación:

Componente 1: **Operaciones de Control de Procesos**

Tamaño estimado: 7000 Instrucciones
Adaptación Software existente: Ninguna

Coeficientes de Ajuste:

FIAB	1.15	TABD	0.94	COMP	1.15
TIEM	1.11	ALMA	1.00	VIRT	1.00
TORD	0.87	CAPA	0.86	EXPA	1.00
CAPP	1.00	EXPV	1.00	EXPL	1.00
TADP	0.91	UHER	1.00	TDES	1.00

Componente 2: **Funciones del Sistema Operativo**

Tamaño estimado: 5000 Instrucciones
Adaptación Software existente: Se modificará el 15% del Diseño, el 30% del Código y el 60% de la Integración

Coeficientes de Ajuste:

FIAB	1.15	TABD	0.94	COMP	1.30
TIEM	1.11	ALMA	1.00	VIRT	1.00
TORD	0.87	CAPA	0.86	EXPA	1.13
CAPP	0.86	EXPV	1.10	EXPL	1.07
TADP	0.91	UHER	1.00	TDES	1.00

Componente 3: **Funciones de Entrada / Salida**

Tamaño estimado: 10000 Instrucciones

Adaptación Software existente: Se modificará el 30% del Diseño, el 60% del Código y el 80% de la Integración

Coeficientes de Ajuste:

FIAB	1.00	TABD	0.94	COMP	1.00
TIEM	1.00	ALMA	1.00	VIRT	1.00
TORD	0.87	CAPA	1.00	EXPA	1.00
CAPP	1.00	EXPV	1.10	EXPL	1.00
TADP	0.91	UHER	1.00	TDES	1.00

Se desea conocer:

1. Estimación Previa: Esfuerzo necesario (Meses-Hombre), Tiempo de Desarrollo (meses), Productividad (Instrucciones / mes-hombre) y Personas trabajando a tiempo completo.

2. Estimación final ajustada Los mismos parámetros anteriores.

Nota: En todos los cálculos se utilizarán dos decimales, con redondeo.

Solución:

1. Estimación Previa:

Nº Total Instrucciones: 7000 + 5000 + 10000 = 22000

Idem. en miles = 22

Meses-Hombre = $3.2 (22)^{1.05} = 82.16$

Tiempo de Desarrollo = $2.5 (82.16)^{0.38} = 13.35$ meses

Productividad = 22000 / 82.16 = 267.77 ==>> 268 Instrucciones / MH

Personas a tiempo completo = 82.16 / 13.35 = 6.15 ==>> 6 personas

2. Estimación final ajustada:

Componente 1:

Nº Instrucciones (NINST) = 7000

Factor Ajuste Adaptación (FAA) = 1 Dado que no hay adaptación del software

NINST ajustadas = NINST * FAA = 7000

NINST (miles) = 7

Factor Ajuste Esfuerzo (FAE) = 0.94 (Producto de los 15 coeficientes de ajuste)

Meses-Hombre nominales = $3.2 (7)^{1.05} = 24.67$

Meses-Hombre ajustados = 24.67 * FAE = 24.67 * 0.94 = 23.19

Componente 2:

Nº Instrucciones (NINST) = 5000

Factor Ajuste Adaptación (FAA) = 0.15 * 40 / 100 + 0.30 * 30 / 100 + 0.60 * 30 / 100 = 0.33

NINST ajustadas = NINST * FAA = 5000 * 0.33 = 1650

NINST (miles) = 1.65

Factor Ajuste Esfuerzo (FAE) = 1.21 (Producto de los 15 coeficientes de ajuste)

Meses-Hombre nominales = $3.2 (1.65)^{1.05} = 5.41$

Meses-Hombre ajustados = 5.41 * FAE = 5.41 * 1.21 = 6.55

Componente 3:

Nº Instrucciones (NINST) = 10000

Factor Ajuste Adaptación (FAA) = 0.30 * 40 / 100 + 0.60 * 30 / 100 + 0.80 * 30 / 100 = 0.54

NINST ajustadas = NINST * FAA = 10000 * 0.54 = 5400

NINST (miles) = 5.4

Factor Ajuste Esfuerzo (FAE) = 0.82 (Producto de los 15 coeficientes de ajuste)

Meses-Hombre nominales = $3.2 (5.4)^{1.05} = 18.78$

Meses-Hombre ajustados = 18.78 * FAE = 18.78 * 0.82 = 15.40

Estimación Final del Proyecto

Total Meses-Hombre ajustados
de los tres componentes = 23.19 + 6.55 + 15.4 = 45.14

Nº Total Instrucciones ajustadas = 7000 + 1650 + 5400 = 14050

Tiempo de Desarrollo = $2.5 (45.14)^{0.38} = 10.62$ meses

Productividad = 14050 / 45.14 = 311.25 ==>> 311 Instrucciones / MH

Personas a tiempo completo = 45.14 / 10.62 = 4.25 ==>> 4 personas

2.7 *Validez de la Estimación de Costes*

El problema que presentan los modelos de estimación de costes basados en Cocomo viene dado, entre otros factores, por la necesidad de saber de antemano y con precisión el tamaño que tendrá el Proyecto cuando haya sido terminado. Además, este dato es el origen de todos los cálculos, y con él y con los distintos coeficientes correctores se estiman todos los resultados de Cocomo.

Por ello cabe preguntarse cuál es la precisión de la estimación de costes en la práctica. El propio Boehm reconoce que su modelo es modestamente preciso: tan solo un 20% del 68% en la Fase de Programación. Por tanto, a la hora de estimar costes, se deberán de tener en cuenta las siguientes consideraciones:

1 Se deben de utilizar otros Modelos de Estimación conjuntamente con Cocomo. Si existen grandes diferencias, se volverá a estimar reajustando los coeficientes correctores.

2 Se deben de hacer varias estimaciones a lo largo del Proyecto, cuando se disponga de información más precisa.

3 Se tendrá en cuenta que los costes varían a lo largo del Proyecto porque existen toda una serie de factores que inciden sobre ellos, los cuales se deben de considerar. Entre otros podemos destacar los siguientes:

   a) Los valores de entrada son imprecisos, ya que son a su vez estimados.

   b) Los Modelos de Estimación son solo aproximaciones. Al notar sus debilidades, hay que compensarlos.

c) Algunos Proyectos son únicos. No encajan en los Modelos.

d) Los Proyectos sufren cambios: duración, recursos, requerimientos, etc.

e) Los Modelos se calibran con Proyectos que pueden no ser representativos del Proyecto que se está estimando.

Por todo lo expuesto se puede concluir que la estimación de costes en general y la de desarrollo de Proyectos en particular, como por otra parte es bien sabido, no es una disciplina exacta. Se ha que tener en cuenta que se está intentando predecir el comportamiento de un desarrollo que está fuertemente condicionado por demasiados factores que no son del todo controlables. Las Aplicaciones son una cosa *viva*, y como tal se comportan. Por tanto, hay que tomar los resultados de la estimación con cuidado y con reservas, sin olvidarse nunca que dichos resultados son exactamente eso: *estimaciones*, con todo lo que eso implica. Lo mejor, por tanto, sería que cada Organización, conociendo perfectamente su área de trabajo y sus características particulares, tanto de sus Proyectos como de sus personas, obtenga de modo heurístico sus propios coeficientes de esfuerzo, para corregir de esa manera los resultados obtenidos de la estimación, con unos factores correctores propios y obtenidos de su propia experiencia laboral (Ver nota a continuación).

Para finalizar este estudio de los modelos Cocomo, es conveniente recordar las palabras de Boehm: "*Los modelos simplemente sirven de ayuda y no están para tomar las decisiones por ti*".

Nota: Ajuste de Coeficientes del Modelo Cocomo Básico

En el año 1997, se creó en la Facultad de Informática de La Coruña un Grupo de Investigación de Métricas del Software, formado por alumnos de quinto curso (que realizaron su trabajo como parte de las prácticas de la asignatura de Ingeniería del Software), para llevar a cabo un desarrollo controlado de proyectos y para obtener los datos necesarios para realizar un estudio sobre diferentes métricas del software. Se formaron veinte equipos de trabajo, de tres a cinco personas cada uno, y se eligieron varios proyectos, en un total de diecisiete, los cuales se analizaron y se entregaron a los diferentes equipos con

instrucciones concretas para su desarrollo. Además, entre los diecisiete proyectos se escogió uno que se entregó a tres equipos y otro que se entregó a dos equipos, para que fuesen desarrollados por completo por cada uno de ellos siguiendo unas pautas de programación diferentes, para probar distintos aspectos de las métricas a validar.

Los proyectos desarrollados (ver relación al final de este apartado) eran de diferente tipo, complejidad y tamaño, y se emplearon lenguajes de programación diferentes, como FoxPro para Windows, Clipper, dBase IV, y Visual Basic, entre otros.

Con los resultados obtenidos, referidos al tiempo de desarrollo y tamaño final de cada uno de los proyectos, y como una parte del proyecto de investigación, se reajustaron los coeficientes del modelo Cocomo Básico, para que proporcionaran estimaciones más acordes con este tipo de aplicaciones y lenguajes. Las ecuaciones modificadas se indican a continuación.

Cocomo Básico, Modo Orgánico:

$$MH = 1.4\ (KNINST)^{0.7}$$

$$TDES = 1.5\ (MH)^{0.12}$$

Cocomo Básico, Modo Híbrido:

$$MH = 2.0\ (KNINST)^{0.77}$$

$$TDES = 1.5\ (MH)^{0.09}$$

Cocomo Básico, Modo Embebido:

$$MH = 2.6\ (KNINST)^{0.84}$$

$$TDES = 1.5\ (MH)^{0.06}$$

Como los entornos de desarrollo son cambiantes, y los proyectos a desarrollar también lo son, cada cierto tiempo se debería de llevar a cabo una investigación similar a la anterior, utilizando lenguajes de programación y proyectos actuales, con el fin de poder reajustar los coeficientes de los modelos Cocomo para conseguir que su precisión se mantenga dentro de límites razonables.

Relación de proyectos desarrollados

01. Gestión de Hojas de Tiempos
02. Gestión de almacén
03. Gestión de empresas de mensajería
04. Cita previa e historiales de la Seguridad Social
05. Control de alumnos
06. Gestión de una banda de música
07. Control de entrada / salida de productos
08. Bibliobase
09. Guía de transporte para lonjas de pescado
10. Gestión de bibliotecas
11. Cooperativas de farmacia
12. Gestión de distribuidoras de fruta
13. Diccionario de datos para ordenadores personales
14. Interfaz para el compilador de ayudas de Windows
15. Gestión de instalaciones deportivas
16. Gestión de hoteles
17. Gestión de empresas de reparto

2.8 *Otros Modelos de Estimación*

Se estudiarán a continuación otros modelos de estimación de costes que van a servir para poder establecer comparaciones con los hasta ahora vistos. Revisaremos aquí la Ecuación de Putnam, el Método de Puntos Función, adoptado posteriormente por IBM y el Modelo de Rayleigh-Norden.

2.8.1 *La Ecuación de Putnam*

Uno de los modelos para la estimación del tamaño y el esfuerzo necesario para el desarrollo de productos software es el propuesto por Putnam en 1978. Por medio de un estudio de datos [Putnam, 1978], define Putnam una ecuación para estimar el tamaño (T) del software, expresado en líneas de código, de la siguiente manera:

$$T = C * K^{1/3} * t_d^{4/3}$$

Donde C es una constante que mide el nivel tecnológico de la empresa, K representa el esfuerzo total de desarrollo, expresado en personas / año, incluyendo el esfuerzo debido al mantenimiento, y t_d es el tiempo de desarrollo expresado en años.

La constante C toma valores en el intervalo [2000, 11000], donde el valor más bajo indica un entorno de desarrollo pobre y carente de metodologías, y el más alto indica un entorno excelente en el que se utilizan herramientas automáticas de desarrollo y las metodologías y técnicas aplicadas son de uso común. Es de notar que, según Putnam, la relación esfuerzo / tiempo no es lineal.

2.8.2 *El Método de Puntos-Función*

Una vez conocidos los problemas que presentaban los modelos de estimación basados en Cocomo, sobre todo el Modelo Cocomo Básico, cuyo único dato de entrada era el tamaño del Proyecto, y dada la dificultad existente para conocer dicho tamaño con precisión, en la década de los 70 Albrecht [Albrecht-Gaffney, 1983] sugiere que podría estimarse dicho tamaño teniendo en cuenta las diferentes funciones que el software deberá de realizar, y posteriormente asignando un valor, en puntos, a cada una de dichas funciones.

Nace así el Modelo de Estimación de Puntos-Función, que fundamentalmente consiste en contar las funciones que debe realizar el software, ajustarlas según su complejidad y hacer una estimación basada en la suma de los pesos de los Puntos-Función resultantes.

El método, se puede resumir como sigue:

1. Se cuentan, a partir del diseño, las siguientes funciones:

   - Nº de entradas externas (External Inputs: EI)
   - Nº de salidas externas (External Outputs: EO)
   - Nº de ficheros lógicos internos (Internal Logical Files: ILF)
   - Nº de ficheros de interfase externos (External Interfase Files: EIF)
   - Nº de consultas externas (External Queries: EQ)

2. El número total de funciones de cada clase se ajusta según su complejidad y se obtiene el total de Puntos-Función iniciales, PF.

3. Se ajustan los Puntos-Función obtenidos de acuerdo con diversos factores que afectan al Proyecto, para obtener los Puntos-Función Ajustados (PFA)

4. Se utilizan los PFA para estimar el esfuerzo, el tiempo y el tamaño del proyecto

2.8.2.1 Definiciones

Para poder contarlas correctamente, se definen a continuación los cinco tipos de funciones que utiliza el método. La nomenclatura utilizada corresponde a la adoptada en 1990, tras la revisión de Albrecht, por el Grupo Internacional de Usuarios de Puntos Función (International Function Points Users Group, IFPUG)

Entradas Externas (External Inputs, EI)

Cada entrada única de *datos* del usuario o cada entrada única de *control* que entra en los límites de la aplicación que está siendo medida.

Una EI se debe de considerar única cuando tiene un formato diferente de otras EI, o si el diseño externo necesita una lógica de proceso diferente de la de otras EI del mismo formato.

Se deben incluir las EI que entran directamente en los límites de la aplicación como transacciones del usuario, o como transacciones procedentes de otras aplicaciones, tales como ficheros de entrada de transacciones.

Cada EI se debe de clasificar dentro de tres niveles de complejidad:

- Simple: Cuando se incluyen pocos tipos de elementos de datos en la EI y cuando la EI referencia a pocos ficheros internos lógicos. Además, cuando las consideraciones de los factores humanos del usuario no son significativas en el diseño de la EI.

- Media: Cuando la EI no se puede claramente definir como simple o como compleja.

- Compleja: Cuando se incluyen múltiples tipos de elementos de datos en la EI y cuando la EI referencia a muchos ficheros internos lógicos. Además, las consideraciones de los factores humanos del usuario afectan al diseño de la EE.

No se debe de incluir como EI:

- Aquellas EI que se introducen en la aplicación por necesidades de la tecnología empleada.

- Ficheros de entrada (de registros), porque estos se contarán como ficheros externos de interfase.

- La parte de entrada de las consultas externas, porque éstas se contarán como consultas externas.

Salidas Externas (External Outputs, EO)

Salida única de *datos* del usuario o salida única de *control* que sale de los límites de la aplicación que está siendo medida.

Una EO se debe de considerar única cuando tiene un formato diferente de otras EO, o si el diseño externo necesita una lógica de proceso diferente de la de otras EO del mismo formato.

Se deben incluir las EO que salen directamente de los límites de la aplicación como informes y mensajes para el usuario, y aquellas que salen directamente hacia otras aplicaciones, tales como ficheros de salida de informes y mensajes.

Cada EO se debe de clasificar dentro de tres niveles de complejidad, usando definiciones similares a las de las EI. Además, para los informes se deberá de utilizar la complejidad siguiente:

- Simple: Una o dos columnas. Transformaciones simples de datos.
- Media: Columnas múltiples con subtotales. Transformaciones múltiples de datos.
- Compleja: Transformaciones complicadas de datos. Referencias múltiples y complejas a ficheros que deban ser correlacionados. Consideraciones significativas de rendimiento.

No se debe de incluir como EO:

- Aquellas EO que se introducen en la aplicación por necesidades de la tecnología empleada.
- Ficheros de salida, porque estos se contarán como ficheros externos de interfase.
- Las respuestas de salida de las consultas externas, porque éstas se contarán como consultas externas.

Ficheros Lógicos Internos (Internal Logical Files, ILF)

Contar como ILF cada grupo lógico de *datos* del usuario o de información de *control* en la aplicación. Incluir cada fichero lógico, o, dentro de una Base de Datos, cada grupo lógico de datos que sea generado, usado y mantenido por la aplicación, desde el punto de vista del usuario. Es decir, contar los ficheros lógicos según se han descrito en el diseño externo, y no los ficheros físicos.

Los ILF se deben de clasificar atendiendo a tres criterios de complejidad:

- Simple: Pocos tipos de registros; pocos tipos de elementos de datos; consideraciones poco significativas de rendimiento o de recuperación.
- Media: Cuando no se puede claramente definir como simple o como compleja.
- Compleja: Muchos tipos de registros; muchos tipos de elementos de datos; consideraciones significativas de rendimiento y de recuperación.

No se debe de incluir como ILF:

- Aquellos que no son accesibles por el usuario por medio de entradas externas, salidas, ficheros de interfase o consultas.

Ficheros de Interfase Externos (External Interface Files, EIF)

Dentro de *cada* aplicación, se deben de contar como EIF los ficheros compartidos entre aplicaciones, o que se pasan entre aplicaciones. Contar como EIF cada grupo lógico de *datos* del usuario o de información de *control* que entra o sale de la aplicación.

Los EIF se deben de clasificar según tres niveles de complejidad, usando definiciones similares a las de los ILF.

Los EIF salientes también se deberían de contar como ILF para la aplicación que se está midiendo.

Consultas Externas (External Queries, EQ)

Cada combinación única de entrada / salida, donde una entrada origine una salida inmediata. Incluir aquellas EQ que provengan directamente del usuario o de otras aplicaciones.

Una EQ se debe de considerar única cuando tiene un formato diferente de otras EQ en su parte de entrada o en su parte de salida, o si el diseño externo necesita una lógica de proceso diferente de la de otras EQ del mismo formato.

Las EQ se deben de clasificar de acuerdo con el siguiente criterio:

- Clasificar la parte de entrada de la EQ usando definiciones similares a las de las EI.
- Clasificar la parte de salida de la EQ usando definiciones similares a las de las EO.
- Tomar como complejidad de la EQ la más alta de las dos anteriores.

Para ayudar a distinguir entre EQ y EI, se tendrá en cuenta que la entrada de datos de una EQ se hace solo para dirigir la búsqueda y no se realiza ninguna actualización de un ILF.

No se debe de confundir una "facilidad" de consulta con una EQ. Una EQ es una búsqueda directa de datos específicos, utilizando normalmente una sola clave, mientras que una "facilidad" de consulta proporciona una estructura organizada de entradas externas, salidas y consultas para componer muchas consultas posibles utilizando muchas claves y operaciones. Se deben de contar *todas* las entradas externas, salidas y consultas para medir una "facilidad" de consulta.

2.8.2.2 Detalle del método de Puntos-Función

Una vez definidas las funciones que es necesario contar, veremos ahora el método con un poco más de detalle.

Paso 1: Definir los límites de la aplicación a medir

En primer lugar, para poder contar las funciones es necesario definir con precisión los límites de la aplicación que se quiere medir. Este límite se utilizará para definir el alcance del sistema y para ayudar a identificar las funciones que intervienen. Esto es particularmente importante dado que será necesario identificar entradas, salidas, ficheros internos a la aplicación y ficheros externos a ella, así como consultas externas hechas por el usuario o desde otras aplicaciones.

Por ejemplo, una aplicación puede estar formada por una parte "batch" y otra parte "on-line", y será necesario establecer si se va a medir una de las partes o si se va a medir toda la aplicación. En el primer caso, habrá que determinar qué funciones pertenecen a la parte que se va a medir y cuales no.

Paso 2: Contar las funciones

Una vez identificado el entorno de medida, se deberá de contar el número de funciones que existen, de acuerdo con las cinco clases especificadas anteriormente. Este número de funciones ya se deben de conocer una vez que se haya completado el análisis lógico de la aplicación que se quiere medir.

El método proporciona una serie de recomendaciones para contar correctamente las funciones. Son las siguientes:

Recomendaciones de conteo - Ficheros	
Entidad lógica de datos, desde el punto de vista del usuario	1 ILF
Ficheros lógicos internos generados o mantenidos por la aplicación	1 ILF
Ficheros accesibles al usuario por medio de palabras clave o parámetros	1 ILF
Ficheros utilizados para datos o para control por aplicaciones batch secuenciales	1 ILF
Ficheros intermedios o ficheros de clasificación	0 ILF
Indices alternativos	0 ILF
Ficheros de respaldo (backup) para relanzamientos / recuperación	0 ILF
Ficheros de respaldo (backup) solicitados por el usuario para satisfacer una necesidad del negocio	1 ILF
Fichero leído desde otra aplicación	1 EIF
Base de datos compartida (sólo en lectura) con otra aplicación	1 EIF
Superficheros (Nº de DET's > 100) – Cada fichero lógico leído	1 EIF

Un DET (Data Element Type) es un campo único no recursivo de un ILF o de un EIF, reconocible por el usuario.

Tabla 2.6 Recomendaciones de conteo - Ficheros

Recomendaciones de conteo - Entradas	
Pantalla de datos de entrada	1 EI
Pantallas de datos múltiples, introducidas, acumuladas y procesadas como una única transacción, sin proceso de las pantallas individuales	1 EI
Entrada de pantalla de funciones	1 EI
Entrada de pantalla de funciones con múltiples funciones diferentes	1 EI por función
Datos automáticos o transacciones procedentes de otra aplicación	1 EI
Consulta seguida de una entrada de actualización	1 EQ / 1 EI
Entrada alternativa con la misma lógica de proceso que la entrada primaria, si la requiere específicamente el usuario	1 EI
Tecla de función duplicada de una pantalla ya contada como una entrada	0 EI
Dos pantallas de entrada con el mismo formato y la misma lógica de proceso	1 EI
Dos pantallas de entrada con el mismo formato y diferente lógica de proceso	2 EI
Pantalla de entrada y salida	1 EI / 1 EO
Formularios de entrada (OCR)	1 EI

Tabla 2.7 Recomendaciones de conteo - Entradas

Recomendaciones de conteo - Salidas	
Pantalla de datos de salida	1 EO
Datos automáticos o transacciones que salen hacia otras aplicaciones	1 EO
Salida de mensajes individuales de error dentro de un marco de mensajes	0 EO
Salida de mensajes individuales de confirmación dentro de un marco de mensajes	0 EO
Informes batch impresos	1 EO
Informes batch de errores	1 EO
Informes impresos por terminal	1 EO
Salida de totales de control	1 EO
Informe de Lista de Auditoría o de Lista de Chequeo	1 EO
Salida de pantalla repetida	0 EO
Salida de pantalla de comienzo (Log on)	0 EO
Salida de pantalla de finalización	1 EO

Tabla 2.8 Recomendaciones de conteo - Salidas

Recomendaciones de conteo - Consultas	
Entrada on-line y salida on-line sin actualización de datos en ficheros	1 EQ
Consulta seguida de una entrada de actualización	1 EQ / 1 EI
Entrada y salida de pantalla con menú de selección	1 EQ
Nota: Si se tiene la posibilidad de realizar consultas de alto nivel, éstas se deben de descomponer en su estructura jerárquica de EI's.	

Tabla 2.9 Recomendaciones de conteo - Consultas

Paso 3: Ajustar las funciones según su complejidad

Una vez obtenido el número total de cada una de las funciones anteriores, éste se deberá de ajustar según su complejidad, multiplicándolo por un valor determinado, que varía entre 3 y 15 dependiendo de si la complejidad de la función es baja, si tiene una complejidad promedio o si es una función compleja.

Funciones identificadas	Baja	Media	Alta
Entradas externas (EI)	3	4	6
Salidas externas (EO)	4	5	7
Ficheros lógicos internos (ILF)	7	10	15
Ficheros de interfase externos (EIF)	5	7	10
Consultas externas (EQ)	3	4	6

Tabla 2.10 Coeficientes de ajuste de la complejidad de las funciones

Para ayudar a determinar esta complejidad, el método proporciona las siguientes tablas:

Complejidad de Entradas Externas (EI)			
Tipos de Ficheros referenciados (File Types Referenced, FTR)	Tipos de Elementos de Datos (Data Element Types, DET)		
	1 a 4	5 a 15	Más de 15
0 ó 1	Baja	Baja	Media
2	Baja	Media	Alta
Más de 2	Media	Alta	Alta

Tabla 2.11 Matriz de complejidad de EI

Donde:

Un FTR es

- Un ILF que es leído o mantenido por un tipo de función
- Un EIF que es leído por un tipo de función

Un DET es un campo único no recursivo de un ILF o de un EIF, reconocible por el usuario.

Complejidad de Salidas Externas (EO) y Consultas Externas (EQ)			
Tipos de Ficheros referenciados (File Types Referenced, FTR)	Tipos de Elementos de Datos (Data Element Types, DET)		
	1 a 5	6 a 19	Más de 19
0 ó 1	Baja	Baja	Media
2 ó 3	Baja	Media	Alta
Más de 3	Media	Alta	Alta

Una EQ debe de tener al menos un FTR

Tabla 2.12 Matriz de complejidad de EO y EQ

Complejidad de Ficheros Lógicos Internos (ILF) y Ficheros de Interfase Externos (EIF)			
Tipo de Elemento de Registro (Record Element Type, RET)	Tipos de Elementos de Datos (Data Element Types, DET)		
	1 a 19	20 a 50	Más de 50
0 ó 1	Baja	Baja	Media
2 a 5	Baja	Media	Alta
Más de 5	Media	Alta	Alta

Tabla 2.13 Matriz de complejidad de ILF e EIF

Donde un RET es un subconjunto de elementos de datos, reconocibles por el usuario, dentro de un ILF o un EIF.

Una vez multiplicado el número de funciones por sus correspondientes coeficientes de ajuste, se suman los resultados y se obtiene así el total de Puntos-Función iniciales, PF.

Paso 4: Estimar la complejidad global de la aplicación

Seguidamente, es necesario calcular la complejidad de la aplicación, con respecto a 14 características que pueden estar presentes y que se valoran de 0 a 5, dependiendo de su influencia. Un valor cero indica que la característica no está presente o no tiene influencia en la complejidad global de la aplicación y un valor 5 indica una fuerte influencia.
Estas características son las siguientes:

- Comunicación de datos
- Proceso de datos distribuido
- Rendimiento
- Configuración para utilización muy alta
- Ratio de transacciones
- Entrada de datos “on-line”
- Eficiencia del usuario final
- Actualización “on-line”
- Procesamiento complejo
- Reutilización
- Facilidad de instalación
- Facilidad de operación
- Lugares múltiples
- Facilidad de cambios

Una vez más, el método proporciona tablas de ayuda para determinar cómo se debe de valorar cada una de estas características.

Características Generales del Sistema Si ninguna de las descripciones que se proponen más adelante se ajustase por completo a la aplicación medida, se deberá de valorar cuál es el grado de influencia que encaje mejor, de acuerdo con los valores indicados en esta tabla.	
0	No presente o sin influencia.
1	Influencia ocasional.
2	Influencia moderada.
3	Influencia media.
4	Influencia significativa.
5	Influencia fuerte.

Tabla 2.14 Características generales del Sistema

1. Comunicación de datos La información de datos y de control se envía a través de sistemas de comunicación. Se considera que los terminales conectados localmente a la unidad de control utilizan sistemas de comunicación. Un protocolo es un conjunto de convenios que permiten el intercambio o transferencia de información entre dos sistemas o entre dos dispositivos. Todos los enlaces de comunicación de datos necesitan algún tipo de protocolo.	
0	La aplicación sólo tiene procesos batch o se ejecuta en un PC aislado.
1	La aplicación es batch pero tiene entrada remota de datos o impresión remota.
2	La aplicación es batch pero tiene entrada remota de datos e impresión remota.
3	La aplicación utiliza recogida de datos on-line o es un “front-end” de teleproceso (TP) para un proceso batch o para un sistema de consultas.
4	La aplicación es más que un “front-end” de TP, pero soporta un único protocolo de comunicaciones.
5	La aplicación es más que un “front-end” de TP y soporta más de un tipo de protocolos de comunicaciones.

Tabla 2.15 Comunicación de Datos

	2. Proceso de datos distribuido Los datos distribuidos o las funciones de procesamiento son una característica de la aplicación, dentro de sus límites.
0	La aplicación no realiza transferencia de datos o procesamiento de funciones entre los componentes del sistema.
1	La aplicación prepara datos para que sean procesados por un usuario final en otro componente del sistema, como una hoja de cálculo o una base de datos en un PC.
2	La aplicación prepara datos que van a ser transferidos y procesados por otro componente del sistema y no por procesos de usuario final.
3	El procesamiento distribuido y la transferencia de datos se realizan on-line y en una única dirección.
4	El procesamiento distribuido y la transferencia de datos se realizan on-line y en ambas direcciones.
5	Las funciones de procesamiento se ejecutan dinámicamente en el componente del sistema que sea más adecuado para ello.

Tabla 2.16 Proceso de datos distribuido

	3. Rendimiento Los objetivos de rendimiento indicados o aprobados por el usuario, bien sean de tiempos de respuesta o de capacidad de procesamiento, han influenciado o influenciarán el diseño, desarrollo, implantación y soporte de la aplicación.
0	El usuario no ha solicitado requisitos especiales de rendimiento.
1	Se han indicado requisitos especiales de diseño y de rendimiento, pero no se ha necesitado realizar acciones especiales para implementarlos.
2	Los tiempos de respuesta o la capacidad de procesamiento son críticos en horas punta. No se ha solicitado un diseño especial para la utilización de la CPU. La fecha tope de los procesos es el siguiente día hábil.
3	Los tiempos de respuesta o la capacidad de procesamiento son críticos durante todas las horas de trabajo. No se ha solicitado un diseño especial para la utilización de la CPU. Los requisitos de fecha tope de procesos que interactúan con otros sistemas son restrictivos.
4	Además, los requisitos de rendimiento especificados por el usuario son lo suficientemente rigurosos como para tener que realizar tareas de análisis de rendimiento durante la fase de diseño.
5	Además, se han tenido que utilizar herramientas de análisis de rendimiento durante las fases de diseño, desarrollo y/o implementación, para cubrir los requisitos de rendimiento especificados por el usuario.

Tabla 2.16 Rendimiento

4. Configuración para utilización muy alta Es una característica de la aplicación el tener una configuración operativa, que requiera consideraciones especiales de diseño, para permitir una utilización muy alta. Por ejemplo, el usuario desea ejecutar la aplicación en un equipo que va a tener una utilización masiva.	
0	No se especificaron restricciones operativas implícitas o explícitas.
1	Existen restricciones operativas, pero son menos restrictivas que las de una aplicación típica. No se requiere un esfuerzo especial para cubrir esas restricciones.
2	Existen algunas consideraciones de seguridad o de coordinación.
3	Se tienen requisitos de procesadores específicos para una parte concreta de la aplicación.
4	Las restricciones operativas indicadas implican restricciones especiales sobre la aplicación en el procesador central o en un procesador dedicado.
5	Además, existen restricciones específicas sobre la aplicación en los componentes distribuidos del sistema.

Tabla 2.17 Configuración para utilización muy alta

5. Ratio de transacciones La tasa de transacciones es alta y ha tenido influencia en el diseño, desarrollo, instalación y soporte de la aplicación.	
0	No se prevén períodos con picos de transacciones.
1	Se prevén períodos (mensuales, quincenales, anuales, o por temporadas) con picos de transacciones.
2	Se prevén picos de transacciones semanales.
3	Se prevén picos de transacciones diarios.
4	La tasa de transacciones especificada por el usuario en los requisitos de la aplicación o en los requisitos de nivel de servicio es lo suficientemente alta como para tener que realizar tareas de análisis de rendimiento durante la fase de diseño.
5	La tasa de transacciones especificada por el usuario en los requisitos de la aplicación o en los requisitos de nivel de servicio es lo suficientemente alta como para tener que realizar tareas de análisis de rendimiento y además utilizar herramientas de análisis de rendimiento durante las fases de diseño, desarrollo y/o instalación.

Tabla 2.18 Ratio de transacciones

6. Entrada de datos "on-line" La aplicación permite entrada de datos y control de funciones "on-line".	
0	Todas las transacciones se procesan en batch.
1	Del 1% al 7% de las transacciones tiene entrada de datos interactiva.
2	Del 8% al 15% de las transacciones tiene entrada de datos interactiva.
3	Del 16% al 23% de las transacciones tiene entrada de datos interactiva.
4	Del 24% al 30 de las transacciones tiene entrada de datos interactiva.
5	Más del 30% de las transacciones tiene entrada de datos interactiva.

Tabla 2.19 Entrada de datos "on-line"

7. Eficiencia del usuario final Las funciones "on-line" proporcionadas por la aplicación hacen énfasis en un diseño que facilite la eficiencia del usuario final. El diseño incluye: • Ayudas a la navegación • Menús • Documentación y ayuda "on-line" • Movimiento automático del cursor • Desplazamientos • Impresión remota (vía transacciones "on-line") • Teclas de función preasignadas • Trabajos batch arrancados por transacciones "on-line" • Selección de cursor de pantallas de datos • Uso intensivo de resaltado, negrilla, colores, subrayado y otros indicadores • Documentación de usuario en papel de transacciones "on-line" • Uso del ratón • Ventanas desplegables • Uso del número mínimo de ventanas para realizar una función • Soporte bilingüe (Contar como 4 items) • Soporte multilingüe (Más de dos lenguajes; contar como 6 items)	
0	Ninguna de los anteriores.
1	De 1 a 3 de las anteriores.
2	De 4 a 5 de las anteriores.
3	6 o más de las anteriores, pero sin que existan requisitos específicos de

	eficiencia por parte del usuario.
4	6 o más de las anteriores, y con fuertes requisitos de eficiencia para el usuario final, que implican tareas de diseño condicionadas por factores humanos (Por ejemplo, minimizar las pulsaciones de teclado, maximizar las opciones por defecto, uso de plantillas, etc.)
5	6 o más de las anteriores y con requisitos específicos lo suficientemente fuertes como para necesitar el uso de herramientas y procesos especiales para demostrar que se han alcanzado los objetivos.

Tabla 2.20 Eficiencia del usuario final

8. Actualización "on-line" La aplicación permite la actualización "on-line" de los ficheros lógicos internos (ILF)	
0	Ninguno.
1	Actualización "on-line" de 1 a 3 ficheros de control. El volumen de actualización es bajo y la recuperación sencilla.
2	Actualización "on-line" de 4 o más ficheros de control. El volumen de actualización es bajo y la recuperación sencilla.
3	Actualización "on-line" de los ILF maestros.
4	Además, es esencial la protección contra pérdidas de datos y ésta se ha diseñado y programado de modo específico en el sistema.
5	Además, los grandes volúmenes de datos obligan a consideraciones de coste en el proceso de recuperación. Se incluyen procedimientos de recuperación altamente automatizados con intervención mínima del operador.

Tabla 2.21 Actualización "on-line"

9. Procesamiento complejo El procesamiento complejo es una característica de la aplicación. Están presentes los siguientes componentes: • Control sensible (por ejemplo procedimientos especiales de auditoría) y/o procesos de seguridad específicos de la aplicación • Gran cantidad de procesamiento lógico • Gran cantidad de procesamiento matemático • Considerable cantidad de procesamiento de excepciones que origina que las transacciones incompletas se tengan que procesar de nuevo. Por ejemplo, transacciones incompletas de Cajeros Automáticos causadas por interrupciones del Teleproceso, valores de datos perdidos o ediciones fallidas • Procesamiento complejo para manipular posibilidades múltiples de entrada / salida. Por ejemplo, multimedia, independencia de dispositivos, etc.	
0	Ninguna de las anteriores.
1	Una de las anteriores.
2	Dos de las anteriores.
3	Tres de las anteriores.
4	Cuatro de las anteriores.
5	Todas las anteriores.

Tabla 2.22 Procesamiento complejo

10. Reutilización La aplicación y su código se han diseñado, desarrollado y documentado de modo específico para poder ser utilizados en otras aplicaciones.	
0	No hay código reutilizable.
1	El código reutilizable se usa dentro de la propia aplicación.
2	Menos del 10% de la aplicación se ha diseñado para que sirva a las necesidades de más de un usuario.
3	El 10% o más de la aplicación se ha diseñado para que sirva a las necesidades de más de un usuario.
4	La aplicación se ha desarrollado y/o documentado para facilitar la reutilización, y el usuario la ha personalizado a nivel de código fuente.
5	La aplicación se ha desarrollado y/o documentado para facilitar la reutilización, y el usuario la personaliza a través del mantenimiento de parámetros.

Tabla 2.23 Reutilización

11. Facilidad de instalación La facilidad de conversión e instalación son una característica de la aplicación. Durante la fase de pruebas del sistema se dispuso de un plan de conversión e instalación y/o se proporcionaron y probaron herramientas de conversión.	
0	El usuario no especificó consideraciones especiales y no se necesitó un procedimiento de instalación especial para la aplicación.
1	No se especificaron consideraciones especiales, pero se necesitó un procedimiento de instalación especial para la aplicación.
2	El usuario especificó requisitos especiales para la conversión e instalación y se proporcionaron y probaron guías de conversión e instalación. El impacto de la conversión en el proyecto no se considera importante.
3	El usuario especificó requisitos especiales para la conversión e instalación y se proporcionaron y probaron guías de conversión e instalación. El impacto de la conversión en el proyecto se considera importante.
4	Además del punto 2, se proporcionaron y probaron herramientas automáticas de conversión e instalación.
5	Además del punto 3, se proporcionaron y probaron herramientas automáticas de conversión e instalación.

Tabla 2.24 Facilidad de instalación

12. Facilidad de operación La facilidad de operación es una característica de la aplicación. Se proporcionaron y probaron procedimientos efectivos de arranque, copias de seguridad y recuperación durante la fase de pruebas del sistema. La aplicación minimiza la necesidad de realizar actividades manuales, tales como montaje de cintas, manipulación de papel, e intervención manual directa en el sitio.	
0	El usuario no especificó ninguna condición especial de operaciones, excepto los procedimientos típicos de copias de seguridad.
1 a 4	Uno, alguno o todos los puntos siguientes son aplicables. Seleccione todos los que procedan. Cada item vale un punto, a menos que se indique lo contrario. • Se proporcionaron procedimientos efectivos de arranque, copias de seguridad y recuperación, pero se necesita la intervención del operador. • Se proporcionaron procedimientos efectivos de arranque, copias de seguridad y recuperación, pero no se necesita la intervención del operador (contar como dos items) • La aplicación minimiza la necesidad de montar cintas. • La aplicación minimiza la necesidad de manipulación de papel.
5	La aplicación está diseñada para operación desatendida. Esto implica que no se necesita la intervención del operador para manipular el sistema, con la excepción del arranque o finalización de la aplicación. La recuperación automática de errores es una característica de la aplicación.

Tabla 2.25 Facilidad de operación

13. Lugares múltiples La aplicación ha sido especialmente diseñada, desarrollada y preparada para que pueda ser instalada en lugares múltiples, para múltiples organizaciones.	
0	Los requisitos del usuario no consideran la necesidad de instalación en más de un sitio o en más de un usuario.
1	Durante el diseño se ha tenido en cuenta la necesidad de instalación en lugares múltiples, y la aplicación está diseñada para operar sólo en entornos de hardware y software idénticos.
2	Durante el diseño se ha tenido en cuenta la necesidad de instalación en lugares múltiples, y la aplicación está diseñada para operar sólo en entornos de hardware y/o software similares.
3	Durante el diseño se ha tenido en cuenta la necesidad de instalación en lugares múltiples, y la aplicación está diseñada para operar en entornos de hardware y/o software diferentes.
4	Se proporciona y se prueba la documentación y el plan de apoyo para permitir el funcionamiento en lugares múltiples, y la aplicación es como se describe en el punto 1 o en el punto 2.
5	Se proporcionan y se prueba la documentación y el plan de apoyo para permitir el funcionamiento en lugares múltiples, y la aplicación es como se describe en el punto 3.

Tabla 2.26 Lugares múltiples

14. Facilidad de cambios

La aplicación ha sido especialmente diseñada, desarrollada y preparada para facilitar los cambios. Las características que se describen a continuación son aplicables:

- Se proporcionan consultas flexibles y facilidad de informes para manejar peticiones simples; por ejemplo, lógica and/or aplicada a un único fichero lógico interno (contar como un item)
- Se proporcionan consultas flexibles y facilidad de informes para manejar peticiones de complejidad media; por ejemplo, lógica and/or aplicada a más de un fichero lógico interno (contar como dos items)
- Se proporcionan consultas flexibles y facilidad de informes para manejar peticiones complejas; por ejemplo, combinaciones de lógica and/or aplicadas a uno o más ficheros lógicos internos (contar como tres items)
- Los datos de control del negocio se almacenan en tablas mantenidas por el usuario con procedimientos operativos "on-line", pero los cambios tienen efecto el siguiente día hábil.
- Los datos de control del negocio se almacenan en tablas mantenidas por el usuario con procedimientos operativos "on-line", y los cambios tienen lugar inmediatamente (contar como dos items)

0	Ninguna de las anteriores.
1	Una de las anteriores.
2	Dos de las anteriores.
3	Tres de las anteriores.
4	Cuatro de las anteriores.
5	Cinco de las anteriores.

Tabla 2.27 Facilidad de cambios

Paso 5: Obtener los Puntos-Función ajustados

Una vez valoradas todas las características, se suman los resultados parciales y se obtiene la complejidad total del proyecto C_p, que a continuación se ajusta obteniéndose así la complejidad ajustada del proyecto, C_{pa}, según la siguiente fórmula:

$$C_{pa} = 0{,}65 + (0{,}01 C_p)$$

Finalmente, se calculan los Puntos-Función ajustados, PF_a, como:

$$PF_a = PF \times C_{pa}$$

A continuación, y como resumen del método, se muestra una plantilla para recoger todos los datos hasta ahora expuestos y calcular los Puntos-Función ajustados.

Cálculo de Puntos Función Ajustados (PF)

	Complejidad			
Funciones	Baja	Media	Alta	Total
EI	___ x3 = ___	___ x4 = ___	___ x6 = ___	_____
EO	___ x4 = ___	___ x5 = ___	___ x7 = ___	_____
ILF	___ x7 = ___	___ x10= ___	___ x15= ___	_____
EIF	___ x5 = ___	___ x7 = ___	___ x10= ___	_____
EQ	___ x3 = ___	___ x4 = ___	___ x6 = ___	_____
			Total PF	_____

Cálculo de Complejidad del Proyecto (CP)

Comunicación de datos	_____	Actualización “on-line”	_____
Proceso de datos distribuido	_____	Procesamiento complejo	_____
Rendimiento	_____	Reutilización	_____
Configuración para utilización muy alta	_____	Facilidad de Instalación	_____
Ratio de transacciones	_____	Facilidad de operación	_____
Entrada de datos “on-line”	_____	Lugares múltiples	_____
Eficiencia del usuario final	_____	Facilidad de cambios	_____
		Total CP	_____

Introduzca uno de los valores siguientes:

No presente, o sin influencia	0	Influencia media	3
Influencia ocasional	1	Influencia significativa	4
Influencia moderada	2	Influencia fuerte	5

CP Ajustada (CPa = 0.65 + (0.01 x CP)

PF Ajustados (PFa = PF x CPa)

Fig. 2.16 Plantilla para la obtención de los Puntos Función Ajustados

Paso 6: Estimar el tamaño de la aplicación

Los Puntos-Función ajustados los utiliza Albretch para estimar el esfuerzo y el tamaño del Sistema.

Albretch usa fórmulas de regresión distintas para cada entorno de desarrollo. Estudió 22 Proyectos durante 5 años, (1.974 - 1.978) y basándose en sus resultados propone un número de líneas de código "fuente" diferente por cada Punto Función, dependiendo del lenguaje de programación.

Lenguaje	Líneas de Código Fuente por PF
Assembler	320
C	150
Algol	106
Cobol	106
Fortran	106
Jovial	106
Pascal	91
RPG	80
PL/I	80
Ada	71
Lisp	64
Basic	64
Bases de Datos de 4ª generación	40
APL	32
Smalltalk	21
Lenguajes de Consultas	16
Hojas de Cálculo	6

Tabla 2.28 Puntos-Función por lenguaje

Presenta Albretch además el siguiente modelo general simplificado:

$$Nº \text{ de Líneas de Código "fuente"} = 66PF_a$$

2.8.2.3 Procedimiento de conteo y ejemplos de conteo

Como resumen del método de estimación de Puntos-Función, se muestra un procedimiento para contar correctamente las funciones de un sistema.

El procedimiento es el siguiente:

1. Seleccionar el sistema a medir.
2. Determinar los límites de la aplicación, desde el punto de vista del negocio por parte de los usuarios y de la gerencia, así como desde el punto de vista de programación y del sistema.
3. Recoger toda la documentación e identificar a los expertos.
4. Completar las 14 características del sistema y valorarlas de 0 a 5.
5. Contar y valorar las funciones, comenzando por ILF e EIF.
6. Contar y valorar el resto de componentes: EI, EO y EQ.
7. Utilizar una plantilla para introducir todos los datos y obtener los Puntos-Función ajustados.
8. Revisar los resultados con los gerentes de proyecto si procede.
9. Enviar los resultados al Coordinador o al Equipo de Puntos-Función, incluyendo la documentación obtenida.

En las páginas siguientes se dan algunos ejemplos para determinar correctamente la complejidad de pantallas e informes.

Ejemplo 1: Pantalla de entrada de datos

Contar los DET de la siguiente pantalla de entrada de datos, para poder determinar su complejidad usando la Matriz de complejidad de EI.

SISTEMA DE PLANIFICACIÓN DE VENTAS
PETICIÓN DE ALTA

CODIGO PLAN: ____
TIPO DE PLAN: ____

CLIENTE:	_____	ESTADO:	_____
DIVISIÓN:	_____	FECHA COMIENZO:	__/__/__
CONTACTO:	_____	FECHA FINALZCN:	__/__/__
PRESUPUESTO		CODIGO GRUPO:	_____
DEL PLAN:	_____	CODIGO REGION:	_____
TOTAL:	_____		

OBSERVACIONES: _____

VENTAS ESTIMADAS

PRIMER TRIMESTRE	_____
SEGUNDO TRIMESTRE	_____
TERCER TRIMESTRE	_____
CUARTO TRIMESTRE	_____
TOTAL	_____

PULSE ENTER PARA DAR DE ALTA EL PLAN

MENSAJE DE ERROR: _____
_____
F1: AYUDA F2: MENU F3: SIGUIENTE F9: SALIR

Cada campo se cuenta como un DET, excepto los campos repetitivos de los cuatro trimestres, que cuentan como un solo DET los cuatro. El uso de la tecla Enter (o el clic del ratón) también añade 1 DET, y otro más para el posible mensaje de error. El uso de las teclas PF no se cuenta. En total, la pantalla tendría 17 DET. Suponiendo que informe se obtuviera a partir de dos ficheros (2 FTR), entonces la complejidad, de acuerdo con la Tabla 2.11, sería alta.

Ejemplo 2: Informe impreso

Contar los DET del siguiente informe por pantalla, para poder determinar su complejidad usando la Matriz de complejidad de EO.

SISTEMA DE PLANIFICACIÓN DE VENTAS
PLANES DE VENTAS INACTIVOS

CODIGO PLAN	CLIENTE	DIVISIÓN	REGION	PRESUPUESTO
xxxxxxxx	xxxxxxxxxxxxxxxxxxx	xxxxxxxxxx	xxxxxxxx	xxxxxxxxxxx
xxxxxxxx	xxxxxxxxxxxxxxxxxxx	xxxxxxxxxx	xxxxxxxx	xxxxxxxxxxx
xxxxxxxx	xxxxxxxxxxxxxxxxxxx	xxxxxxxxxx	xxxxxxxx	xxxxxxxxxxx
xxxxxxxx	xxxxxxxxxxxxxxxxxxx	xxxxxxxxxx	xxxxxxxx	xxxxxxxxxxx
xxxxxxxx	xxxxxxxxxxxxxxxxxxx	xxxxxxxxxx	xxxxxxxx	xxxxxxxxxxx
xxxxxxxx	xxxxxxxxxxxxxxxxxxx	xxxxxxxxxx	xxxxxxxx	xxxxxxxxxxx
xxxxxxxx	xxxxxxxxxxxxxxxxxxx	xxxxxxxxxx	xxxxxxxx	xxxxxxxxxxx
			TOTAL:	xxxxxxxxxxxxx

F1: AYUDA F2: MENU F3: SGNTE F4: ANTERIOR F5: IMPRIMIR F9: SALIR

Cada columna del informe se cuenta como un DET, independientemente del número de líneas de detalle del informe, que son repeticiones de los mismos tipos de dato. El total cuenta como otro DET, por lo que en total se tendrían 6 DET.

Suponiendo que informe se obtuviera, por ejemplo, a partir de dos ficheros (2 FTR), entonces la complejidad, de acuerdo con la Tabla 2.12, sería media.

2.8.2.4 Consideraciones sobre el método de Puntos-Función

Algunos de los inconvenientes del método vienen dados por la arbitrariedad de los coeficientes de ajuste utilizados por Albretch, así como a la dificultad para identificar las funciones del software a partir de las especificaciones, ya que éste es un proceso bastante subjetivo.

Se debe de tener en cuenta, a la hora de valorar el método de Albrecht que sus datos no incluyen factores tomados en cuenta por el Cocomo Intermedio, como por ejemplo la experiencia de los programadores. Por tanto, se impone la misma conclusión obtenida al estudiar los modelos de estimación basados en el Cocomo: es necesario estimar utilizando diferentes métodos para poder comparar sus resultados. Una buena aproximación podría ser la utilización del método de Puntos Función para obtener el número de instrucciones (NINST) que sirve de entrada a los modelos Cocomo.

De todos modos, hoy en día, tal vez la influencia fuerte sobre la productividad esté en disponer de Sistemas de Desarrollo Interactivo On-line, tal como se ha probado en desarrollos realizados en programación de workstations, factor que no está totalmente recogido en los modelos estudiados, por no estar la tecnología disponible en la época en que fueron propuestos.

Tras los estudios de Albretch, y para probar su validez, Behrens [Behrens, 1983] aplica el Método de Puntos-Función a 24 Proyectos de Desarrollo realizados ya con tecnología más reciente y establece que "*el tamaño, entorno de desarrollo (on-line, batch) y el lenguaje de programación son determinantes de la productividad del programador"*, y como consecuencia de la posterior calidad del Sistema.

2.8.3 *El Modelo de Rayleigh-Norden*

Otro de los modelos de estimación que se pueden considerar a la hora de determinar una estimación del coste de desarrollo de un Proyecto software, y al que ya se ha hecho referencia al estudiar el modelo Cocomo Intermedio, se basa en los estudios realizados inicialmente por Lord Rayleigh (1842 - 1919), y posteriormente retomados por P. V. Norden [Norden 1958, Norden 1970]

Lord Rayleigh, además de ser conocido por sus estudios sobre el color del cielo en los atardeceres, investigó las fuerzas que concurren en las mareas y en el movimiento de las olas, formulando teorías sobre lo que posteriormente se conoció como la *Ola de Rayleigh.*

La Ola de Rayleigh es, pues, una perturbación de la superficie causada por dos fuerzas, de elevación y de descenso:

r(t) Fuerza ascendente
d(t) Fuerza descendente

Posteriormente, Norden utiliza la curva de Rayleigh para explicar el comportamiento de los Proyectos de Investigación y Desarrollo (I+D) y define la ecuación de la Ola de Rayleigh en un instante t, como:

$$\frac{dw(t)}{dt} = r(t)d(t) \qquad (1)$$

Donde w(t) es la ecuación (trabajo) de la Ola en un instante de tiempo t.

Afirma Norden que esta ecuación puede representar los fenómenos organizativos que se presentan en Proyectos de I+D, siendo:

r(t) Curva de aprendizaje (ascendente)
d(t) Cantidad de trabajo a realizar en t (descendente)
w(t) Trabajo total (Unidad) $0 \leq w(t) \leq 1$

Y donde el esfuerzo de desarrollo, para K personas-hora, viene dado por:

Esfuerzo = Kw(t) K = personas-hora

Norden propone, como valores para r(t) y d(t) los siguientes:

r(t) = 2at Donde: a = Constante que determina la forma de la curva

d(t) = 1-w(t) Siendo: 1-w(t) el trabajo pendiente, y d(t) la disminución esfuerzo

Entonces, substituyendo en (1), se tiene:

$$\frac{dw(t)}{dt} = 2at(1 - w(t))$$

Y resolviendo:

$$w(t) = (1 - e^{-at^2})$$

Así, el esfuerzo total Kw(t), con K personas-hora será:

$$Kw(t) = K(1 - e^{-at^2})$$

Y finalmente representando de modo gráfico se tiene:

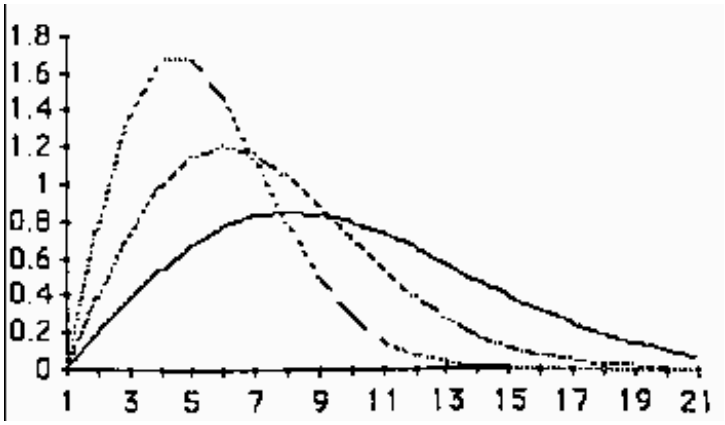


Fig. 2.17 Curvas de Rayleigh

Con posterioridad a Norden, otros investigadores validan los resultados obtenidos por él, aunque sugieren diferentes valores para r(t) y para d(t), proponiendo:

$$r(t) = w(t)$$
$$d(t) = 1\text{-}w(t)$$

Lo cual parece bastante lógico si se tiene en cuenta que w(t) es el esfuerzo necesario para completar el desarrollo, por lo que 1-w(t) será el esfuerzo que falta para terminarlo.

Con estos nuevos valores, la ecuación diferencial resultante sería diferente de la planteada por Norden, quedando expresada de la siguiente manera:

$$\frac{dw(t)}{dt} = w(t)(1 - w(t))$$

$$w(t) = (1 - e^{\frac{-wt}{2}})$$

Asimismo, se pueden proponer nuevos valores para r(t) y para d(t), que arrojarían nuevas ecuaciones para la estimación de costes de desarrollo de Aplicaciones software. Valen para este modelo las mismas consideraciones realizadas para los modelos anteriores, es decir, que sus resultados son estimaciones y que se deben de contrastar con los obtenidos por otros modelos, realizando además estimaciones periódicas a medida que vamos avanzando en el desarrollo y a medida que se dispone de más datos sobre el Proyecto.

2.9 *Ley de Brooks de los Rendimientos Decrecientes*

Para finalizar este capítulo, enunciaremos y demostraremos, aunque de modo informal, la Ley de Brooks, mencionada ya con anterioridad cuando se revisaban los Modelos Cocomo.

La Ley de Brooks de los Rendimientos Decrecientes establece el efecto que tiene el tamaño de un programa en la productividad de los programadores, debido fundamentalmente a dos causas: los problemas de comunicación que surgen en los Equipos de Trabajo cuando aumenta el número de programadores y la complejidad de los programas a crear.

Brooks enuncia su Ley así: "*Añadir programadores a un Proyecto de Programación retrasado, hace que su fecha de finalización se retrase aún mas*".

Y justifica la Ley basándose en el incremento del esfuerzo debido al aumento de las comunicaciones necesarias entre los nuevos programadores y los ya existentes en el Proyecto. Afirma Brooks que dicho esfuerzo de comunicación se incremente más rápidamente que el incremento lineal del trabajo que se puede obtener al aumentar programadores.

Veamos una demostración informal de la Ley de Brooks, utilizando un ejemplo.

Sea *NINST* el número de instrucciones de un Proyecto y *Ed* el esfuerzo de desarrollo, en horas, del mismo. Estudiaremos dos casos:

Caso 1: Un único programador

En este caso, la solución es trivial. El Esfuerzo Total (*Et*) será el debido al único programador, y su productividad (*Pp*) será el número de instrucciones por hora que es capaz de realizar dicho programador. Así:

$$Et = Ed \text{ (horas)}$$

$$Pp = \frac{NINST}{Et} \text{ (Instrucciones / hora)}$$

Caso 2: n programadores

Ahora, como el número de personas que se tienen que comunicar aumenta hasta n, el número de interacciones entre ellos aumentará hasta $n(n-1)^2$

Esto es fácil de ver si analizamos los siguientes gráficos que muestran este aumento de comunicaciones.

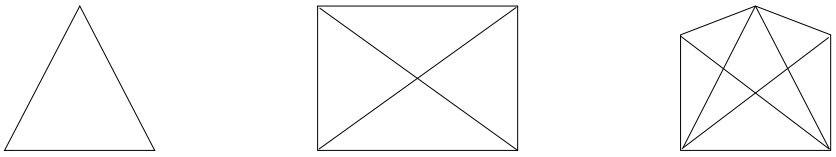


Fig. 2.18 Número de interacciones entre diferentes personas

Efectivamente, el número de interacciones *i* entre n personas responde a la fórmula:

$$i = n(n-1)^2$$

Definimos ahora *Ec* el esfuerzo debido a las comunicaciones como:

$$Ec = \mu n^2$$

donde n(n-1) se ha simplificado como n^2 y μ es una constante que representa el incremento del esfuerzo por cada interacción.

Si calculamos ahora el Esfuerzo Total y la Productividad de los n programadores (*Pn*) se tiene que:

Et = Ed + Ec	Esfuerzo de desarrollo + Esfuerzo debido a las comunicaciones

$$Pn = \frac{NINST}{Et} \quad \text{Instrucciones / hora}$$

Y substituyendo:

$$Et = Ed + \mu n^2$$

$$Pn = \frac{NINST}{Ed + \mu n^2}$$

Obviamente, esta productividad es inferior que la productividad de un único programador, puesto que la ratio Pn / Pp es menor que 1.

Aunque este modelo simplificado es inadecuado para explicar todos los efectos del tamaño del Equipo de Trabajo en el desarrollo del Proyecto, no obstante muestra como el añadir nuevos programadores a un Proyecto retrasado tiene un impacto negativo en el progreso del mismo. Piénsese que esos nuevos programadores tienen que ser puestos al corriente de los pormenores del Proyecto por los programadores que ya están trabajando en él, y que son los únicos que saben en que situación se encuentra. Este aumento de comunicaciones origina pérdidas de tiempo superiores al tiempo que se ganaría con la ayuda de los nuevos programadores.

2.10 ***Bibliografía***

[Adams, 1980] E.N.Adams, *Minimizing Cost Impact of Software Defects*. IBM Research Report RC 8228, Abril 1980.

[Albrecht, 1979] A.J. Albrecht. *Measuring Application Development Productivity*, Proceedings of the IBM Application Development Symposium, Monterey, California, October 1979, pp. 83-92.

[Albrecht-Gaffney 1983] Albrecht y Gaffney. *Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation.* IEEE Transactions on Software Engineering, Volume SE-9, Number 6, November 1993, pp. 639-648.

[Baker, 1972] F.T.Baker. *Chief Programmer Team*. IBM Syst. J., 11, 1, 1972

[Basili-Weiss, 1981] V.R.Basili y D.M.Weiss. *Evaluation of a Software Requirements Document by Means of Change Data*. Proceedings, Fifth International Conference on Software Engineering, IEEE, Marzo 1981, pp. 314-323

[Boehm, 1973] B.W.Boehm. *Software and its Impact: A Quantitative Assessment*. Datamation, Mayo 1973, pp. 48-59.

[Boehm, 1976] B.W.Boehm. *Software Engineering*. IEEE Trans. Computers, Diciembre 1976, pp. 1226-1241.

[Boehm-Wolverton, 1978] B.W.Boehm y R.W.Wolverton. *Software Cost Modelling: Some Lessons Learned*. Proceedings, Second Software Life-Cycle Management Workshop, U.S.Army Computer Systems Command, Atlanta, Agosto 1978.

[Boehm, 1981] B.W.Boehm. *Software Engineering Economics*. Prentice-Hall, Inc., 1981.

[Brooks, 1975] F.P.Brooks Jr.. *The Mytical Man-Month*. Addison-Wesley, Reading, MA, 1975

[Brooks, 1980] W.D.Brooks. *Software Technology Payoff: Some Statistical Evidence*. IBM-FSD, Bethesda, MD, Abril 1980, pp. 2-7.

[Emery, 1971] J.Emery. *Cost-Benefit Analysis of Information Systems.* J. SMIS, 1971, pp. 16-46.

[Jones, 1986] Capers Jones, *Programming Productivity*, McGraw-Hill, 1986.

[Jones, 1995] Capers Jones, *What are Function Points*" http://www.spr.com/library/funcmet.html, latest version, March 1995.

[Kemerer, 1987] Chris F. Kemerer, *An Empirical Validation of Software Cost Estimation Models*, Communications of the ACM, Volume 30, Number 5, May 1987, pp. 416-429.

[Norden, 1958] P.V.Norden. *Curve Fitting for a Model of Applied Research and Development Scheduling*. IBM J. Rsch. Dev., Vol. 2, Nº 3, Julio 1958.

[Norden, 1970] P.V.Norden. *Useful Tools for Project Management.* en Management of Production, M.K. Starr, ed. Penguin Books, Baltimore Md, 1970, pp. 71-101.

[Putnam, 1976] L.H.Putnam. *A Macro-Estimating Methodology for Software Development*. Proceedings, IEEE COMPCOM 76 Fall, Septiembre 1976, pp. 138-143.

[SHARE-GUIDE, 1979] *SHARE, Inc., y GUIDE International*, Proceedings, Application Development Symposium, Octubre 1979.

2.11 Bibliografía sobre el Modelo de Puntos-Función

En Inglés

1. Abran, Alain, "A Case Study in Function Point Implementation", 1989 Conference on Improving Productivity in System Development, Applied Computer Research, Phoenix, AZ, Jan. 1989.
2. Abran, Alain, "The State of the Union - Annual Productivity Report", IFPUG Spring Conference, Orlando, Florida, April 4, 1990.
3. Abran, Alain, Robillard, Pierre N., "Software Management Based on Software Deliverables", Proceedings CIPS Congress 90, Ottawa, May 17, 1990, pp. 237-245.
4. Abran, Alain, Robillard, Pierre N., "Identification of the Structural Weakness of the Function Points Metrics", 3rd Annual Oregon Workshop on Software Metrics, Portland, Oregon, March 18, 1991.
5. Abran, Alain, Robillard, Pierre N., "Identification of the Structural weaknesses of the Function Points Metrics", Montreal Trust, Technical research report 91-04, Feb. 1991.
6. Abran, Alain, Robillard, Pierre N. "Reliability of Function Points Productivity Models for Enhancement Projects (A Field Study)", ConfErence on Software Maintenance 1993-CSM-93, Montreal, September 27-30 1993, IEEE Computer Society Press, Los Alamitos, pp.80-97.
7. Abran, Alain, Robillard, Pierre N., "Function Points : A Study of their Measurement Processes and Scale Transformations", Journal of Systems and Software, May 1994, pp.171-184.
8. Abran, Alain, Robillard, Pierre N., "Empirical Validation of Function Points Measurements Processes", IEEE Transactions on Software Engineering (To be published).
9. Abran, Alain, Desharnais, Jean-Marc, Meyerhoff Dirk, Mullerburg Monika, St. Pierre, Denis, "Structured Hypertext for Using and Learning Function Point Analysis", SEKED94 6th International Conference on Software Engineering and Knowledge Engineering, Jurmala, Latvia, June 1994, pp.164-171.
10. Abran, Alain, "Function Points Models : Empirical Conditions for Reliability and Ease of Use", European Software Cost Modelling Conference, Ivrea, Italy, 11-13 May, 1994.
11. Abran, Alain, Desharnais, Jean-Marc, Meyerhoff, Dirk, Mullerburg, Monika, St. Pierre, Denis, "Structured hypertext for using and learning function point analysis", In: Berztiss, A.T. (Hrsg.): Proceedings SEKE'94 - Sixth International Conference on Software Engineering and Knowledge Engineering. Skokie: Knowledge Systems Institute, 1994. S.164-171. ISBN 0-964-1699-0-8.
12. Abran, Alain, Desharnais, Jean-Marc, "Measurement of Functional Reuse in Maintenance", Journal of Software Maintenance : Research and Practice, Fall 1995.
13. Abran, Alain, "Function Point-Based Production Models", GMD, Sankt Augustin, Summer 1995.
14. Albrecht, Allan J., "Measuring Application Development Productivity", Proceedings SHARE/GUIDE IBM Applications Development Symposium, Monterey, CA., Oct 14-17, 1979.

15. Albrecht, Allan J., "Function Point helps managers assess application". Computerworld, SR/20, 26 August 1985.
16. Albrecht, Allan J., "Measuring Application Development Productivity", Tutorial -- Programming Productivity: Issues for the Eighties, IEEE Computer Society, ISBN 0-8186-0681-9, 1986, pp. 35-44.
17. Albrecht, Allan J., "Application Development and Maintenance Measurement and Analysis Guideline", IBM Corporate Information System and Administration, White Plains, N.Y., 1981.
18. Albrecht, Allan J., Gaffney, John E., "Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation", IEEE Transactions on Software Engineering, Vol. SE-9, no. 6, pp. 639-648, Nov. 1983.
19. Albrecht, Allan J., "Function Points Fundamentals", IFPUG Fall Conference, MontrEal, Oct. 1988.
20. Albrecht, Allan J., "Development and History of the function point measure", Proceedings of the NGI seminar "FPA in beweging", Scheveningen, 21-22 November 1988.
21. Albrecht, Allan J., "Measurement and Function Metrics - A Current Perspective", IFPUG Spring Conference, Florida, April 3, 1990.
22. Albrecht, Allan J., "Function Points - A Method for Measuring Application Development Productivity", Q.A.I., White Plains, N.Y.
23. Banker, Rajiv D., Kemerer, Chris F., "Scale Economies in New Software Development", IEEE Transactions on Software Engineering, Vol. 15, no. 10, pp. 1199-1205, October 1989.
24. Banker, Rajiv D., Chang H., Kemerer, Chris F., "Evidence on Economies of Scale in Software Development", Information and Software Technology, vol. 36, no. 5, 1994.
25. Banker, Rajiv D., Kauffman, Robert J., "Reuse and productivity in integrated computer-aided software engineering: an empirical study", MIS Quarterly, vol.15 no.3, p375(27), September 1991.
26. Banker, Rajiv D., Kauffman, Robert J., Wright C., Zweig D., "Automating output size and reuse metrics in a repository-based computer-aided software engineering (CASE) environment", IEEE Transactions on Software Engineering, vol.20 n.3, p169(19), March 1994
27. Barrow, Dean, Nilson, Susan, Timberlake, Dawn, "STSC Software Estimation Technology Report", Technical Report, Software Technology Support Center OO-ALC/TISE Hill Air Force Base, Utah, March 1993.
28. Behrens, Charles A., "Measuring the Productivity of Computer Systems Development Activities with Function Points", IEEE Transactions on Software Engineering, Vol. SE-9, no. 6, November 1989.
29. Benyahia, Hadj, Desharnais, Jean-Marc, Hudon, Georges, Martin, Charles, "Adjustement Model for Function Points Scope Factors. A Statistical Study", IFPUG Special Issues, Montreal, april 1990.
30. Betteridge, Roger, Fisher, David, Goodman, Paul, "Function Points vs Lines of Code", System Development, August 1990.
31. Betteridge Roger, "Successful experience of using function points to estimate project costs early in the life-cyle",Information and software Technoloy,Vol. 34 No 10, October 92.
32. Belden, Andy, McNamara Don, Paulson Rich, "Function Points Analysis - Management Briefing", IFPUG Spring Conference, Feb. 1989.

33. Biderman, Bev, "Using Function Points", Computing Canada, Feb. 15, 1990, vol 16 no 4 p30(2).
34. Bilow, Steve C., Henderson-Sellers, Brian, "Report on the Workshop on Pragmatic and Theoretical Directions in Object-Oriented Software Metrics, October 23, 1994" In Ott, L., (ed.) Q Methods Report, Committee on Quantitative Methods, Technical Council on Software Engineering, IEEE Computer Society, No. 7, Winter 1995, p. QMR 4.
35. Bock, Douglas B., Klepper, Robert, "FP-S: A simplified Function Point Counting method.", Working Paper, Southern Illinois Universitity at Edwardsville, Ill., 1990.
36. Bock, Douglas B., Klepper Robert, "A Simplified Function Point Counting Method", Dept. Management Information Sciences, Southern Illinois University at Edwardsville, IL, Nov. 17, 1989, IFPUG, Spring Confl. Orlando, FL.
37. Bock, Douglas B., Klepper, Robert, "FP-S: a simplified function point count method", The Journal of Systems and Software, July 1992 vol 18 no 3 p245(10).
38. Bouldin, Barbara M., "What are you measuring? Why are you measuring it?" Software Magazine, August 1989 v9 n10 p30(7).
39. Brooks, Irwin L., "Engineering Function Points and Earned Value Tracking Systems", CrossTalk Journal, Volume 7, Issue 11, November 1994. to obtain the document
40. Brown, Darlene, "Productivity Measurement Using Function Points", Software Engineering, Auerbach Publ., July-August 1990.
41. Connolley, Michael J., "An empirical study of Function Points analysis reliability", Master Thesis, MIT Sloan School of Management, Cambridge, Mass., 1990.
42. Connolley, Michael J.,"Summary of Microcase Results",IFPUG/MIT Function Point Reliability Study, Jul 04 1990.
43. Davis, Dwight B., "Develop applications on time, every time", Datamation, Nov 1, 1992 v38 n22 p85(4).
44. Desharnais, Jean-Marc, "Adjustment Model for Function Points Scope Factors - A Statistical Study", IFPUG Spring conf., Florida, April 1990.
45. Development Support Center Inc., "The Who, What, When Where, Why and How of Function Point Counting", Elm Grove, Wisconsin, 1990.
46. Douglas Neil, "The Metrics Conundrum: How do we choose the best metric?", American Programmer, Vol. 8 No. 12, December 1995.
47. Dreger, J. Brian, "Function Points Analysis", Prentice-Hall, 1989.
48. Emrick, Ronald D. "Software Development Productivity - Second Industry Survey", IFPUG Spring Conference, Dallas, May, 1988.
49. Emrick, Ronald D. "Further Analysis - Software Development Productivity - Second Industry Survey", IFPUG Fall Conference, MontrEal, Sept. 1988.
50. ESPRIT, projet MERMAID, Survey report, Volume 1: Effort and size estimation models, Feb. 1989.
51. ESPRIT, projet MERMAID, Survey report, Volume 2 - Software metrics, Feb. 1989.
52. Ferens, Daniel V., Gurner, Robert B., "An evaluation of three function point models for estimation of software effort", Institute of Electrical and Electronics Engineers, Inc. p. 635-642. 1992, CASI Accession Number: 93A42834.
53. Gaffney, John E., "The Impact on Software Development Costs of Using HOL's", IEEE Transactions on Software Engineering, vol. 12 no 3, pp. 496-499, 1986.

54. Gaffney, John E. Jr."A Generalization of Function Points and Application to Aerospace Software Estimation", Software Productivity Consortium Inc., Virginia, 1991.
55. Garmus, Dave, Herron, David, "Measuring The Software Process: A Practical Guide to Functional Measurements", Prentice Hall, ISBN 0-13-349002-5, October 1995.
56. Glass, Robert L.,"Quality Measurement:Two Very Different Ways", System Development, June 1990
57. Grupe, F.H., Clevenger, Dorothy F., "Using Function Point Analysis as a software development tool", Journal of Systems Management', 12, pp. 23-26, 1991.
58. GUIDE International Corp., "Measurement of Productivity", Guide Publications GPP-65, 1981.
59. GUIDE International Corp., "Estimating Using Function Points Handbook", GUIDE Publications GPP-134, 1985, Reprint 1989.
60. Hadlock, Wayne, "Estimation Earlier with Function Points", IFPUG Fall Conf., San Antonio, Tx, Oct. 1990.
61. Hadlock, Wayne W.,"Estimating Earlier With Function Points", Software Productivity Research, Inc. Burlington, MA.
62. Harrison Warren, Miluk Gene,"A Progress Report on Using Code Metrics to Approximate Function Points for Existing Code Assets".
63. Hastings, T., "Adapting function points to contemporary software systems: A review", In Jeffery, R. (ed.): Second Australian Conference on Software Metrics, University of New South Wales, Sydney, Australia, 1995. to obtain the document
64. Hau Zhao, Stockman Tony, "Software sizing for OO software development - Object Function Point Analysis", 2nd Guide Share Europe International Conference on Information and Communication Technology and its related Management, Berlin, 9 - 12 octobre, 1995, 12 pp.
65. Heemstra F. J., Kusters R. J., "Function point analysis: evaluation of a software cost estimation model," European Journal of Information Systems, vol. 1, pp. 229--37, 1991.
66. Henderson, Garland S., "The application of function points to predict source lines of code for software development - M.S. Thesis", Air Force Inst. of Tech., Wright-Patterson AFB, OH., Report Number: AD-A258447, AFIT/GCA/LSY/92S-4, September 1992.
67. Hetzel, Bill, "Making Software Measurement Work - Building an Effective Measurement Program", QED Technical Publishing Group, Boston, 1993.
68. Horner Simon A.,"Position paper for OOPSLA Metrics Workshop", OOPSLA Workshops on O-O Metrics, 1994. to obtain the document
69. Hufschmidt, B., "What is the International Function Point Users Group (IFPUG)?", METRICVIEWS, Newsletter of the International Function Point Users Group, Westerville, Ohio, July 1992.
70. International Function Point Users Group (IFPUG), "Function Points as an Asset - Reporting to Management", IFPUG, Westerville, Ohio, April 1990.
71. International Function Point Users Group, "Counting Practices Manual Release 4.0", January 1994.

72. IBM, "AD/M Productivity measurement and estimate validation", IBM Corporate Information Systems and Administration, Document Number CIS & Guideline 313, January 85.
73. Inwood, Clifford, "Function point remains metric of choice", Computing Canada, Sept 14, 1994 v20 n19 p20(1).
74. ISO/IEC/SC7: CD 14143, "Information Technology - Software measurement - Definition of functional size measurement", 1995.
75. Jeffery D.R.,Low G.C., Barnes M.,"A comparison of Function Point Counting Techniques", IEEE Transactions on Software Engineering, Vol.19, No.5, May 1993.
76. Jensen, R.L., Bartley, J.W., "Parametric estimation of programming effort: an object-oriented model", J. Systems and Software, 15 (2) 1991, 107-114.
77. Jones, Capers, Measuring Programming Productivity and Quality", IBM Systems Journal, vol. 17, no. 1, 1978.
78. Jones, Capers, "Programming Productivity : Issues for the Eighties", IEEE Press, 1981 (Revised 1986), ISBN 0-8186-0681-9.
79. Jones, Capers, "Programming Productivity" McGraw Hill, 1986, ISBN 0-070032811-0.
80. Jones, Capers, "Measuring the Economic Productivity of Software", Perspective on Technology, Vo. 2, no. 2, Metropolitan Life, Summer 1988.
81. Jones, Capers, "Building a Better Metric", Computerworld Extra, June 20, 1988.
82. Jones, Capers, "Feature Points (Function Point Logic for Real Time and System Software)", IFPUG Fall 1988 Conference, Montreal, QuEbec, Oct. 1988.
83. Jones, Capers, "A Short History of Function Points and Feature Points", Software Productivity Research Inc., Technical paper, Cambridge, Mass., 1988.
84. Jones, Capers, "Metric With Muscle - Measuring software productivity in economic terms", System Development, Applied Computer Research Publ, Phoenix, AZ, August 1989.
85. Jones, Capers, "Cost of a Lifetime", Software Maintenance News, vol 7, no 9, p. 14, Sept. 1989.
86. Jones, Capers,"Measuring Software Productivity in Economics Terms",System Development Function Points Metric with Muscle, August 1989, p.1
87. Jones, Capers, "Using Functional Metrics to Evaluate CASE", IFPUG Spring Conference, Baltimore, Maryland, April 2-5, 1991.
88. Jones, Capers, "Applied Software Measurement, Assuring Productivity and Quality", McGraw-Hill, ISBN 0-07-032813-7, 1991.
89. Jones, Capers, "Critical Problems in Software Measurement", Information Systems Management Group, 1993, ISBN 1-56909-000-9.
90. Jones, Capers, "Software Productivity and Quality Today : The Worldwide Perspective", Information Systems Management Group, 1993, ISBN -156909-001-7.
91. Jones, Capers, "Table of Programming Languages and Levels", Technical Report, SPR Inc., Burlington, MA, January 1994.
92. Jones, Capers, "Function points: A new way of looking at tools", Computer, august 1994, pp. 66 - 67.
93. Jones, Capers, "Global Software Quality in 1995". Proc. of the 5ICSQ, October 24-26, 1995, Austin, Texas, pp. 283-290.
94. Jones, Capers, "New Directions in Software Management", Information Systems Management Group, ISBN 1-56909-009-2.

95. Kemerer, Chris F., "An Empirical Validation of Software Cost Estimation Models", Communications of the ACM, Vol. 30, no. 5, May 1987.
96. Kemerer, Chris F., Porter, Benjamin S., "Improving the Reliability of Function Point Measurement:An Empirical Study", IEEE Transactions on Software Engineering, Vol.18, No.11, pp.1011-1024, November 1992.
97. Kemerer, Chris F.,"Reliability of Function Points Measurement. A Field Experiment", Communications of the ACM, Vol.36, No.2, pp.85-97, February 1993.
98. Keuffel, Warren, "Predicting with function point metrics", Software Development, July 1994 v2 n7 p27(5).
99. Kitchenham, Barbara, Taylor, N.R., "Software Cost Models", ICL Tech. J., vol. 4, no. 1, pp. 73-102, May 1984.
100. Kitchenham, Barbara, Kirakowski J., "2nd Analysis of MERMAID Data", ESPRIT Project P2046, Deliverable D3.3B, Oct. 7, 1991.
101. Kitchenham, Barbara A., "Empirical studies of assumptions that underlie software cost-estimation models", Information and Software Technology, vol 34, no 4, April 1992.
102. Kitchenham, Barbara, "Using Function Points for Software Cost Estimation - Some Empirical Results", Proceedings of the Tenth Annual Conference of Software Metrics and Quality Assurance in Industry, Amsterdam, 29 September - 1 October 1993.
103. Kitchenham, Barbara, Kaensaelae, K., "Iter-item Correlations among Function Points . Proceedings of the First International Software Metrics Symposium", Baltimore, May 21-22, 1993, pp. 11-14.
104. Knaff, G.J., Sacks J., "Software Development Effort Prediction Based on Function Points", Proceedings, COMPSAC '86, Chicago, Il. 1986.
105. Knight, Caroline, "Starting an FP Program", System Development, Applied Computer Research Publ. Phoenix, AZ, August 1989.
106. Knight, Caroline, "Starting an FP Program", System Development Function Points, August 1989, p.9
107. Koch, Warren B., "Function Points at Bell Canada", System Development, ACR Publ. Phoenix, AZ, August 1989.
108. Koch, Warren B., "Productivity Results RE Function Points", IFPUG Fall Conference, 1989.
109. Low, Graham C., Jeffery D. Ross, "Function Points in the Estimation and Evaluation of the Software Process", IEEE Transactions on Software Engineering, Vol. 16, no. 1, pp. 64-71, Jan. 1990.
110. Lindskog, Donna, "Measurement Theory Applied to Function Points", Internal Report of the University of Regina, Canada, December 2, 1986.
111. Longstreet David H., "How Are Function Points Useful?", American Programmer, Vol. 8 No. 12, December 1995.
112. MacDonell, Stehpen G., "Comparative review of functional complexity assessment methods for effort estimation", Software Engineering Journal, may 1994, pp. 107 - 116.
113. Matson, Jack E., Mellichamp, Joseph M., "An object-oriented tool for function point analysis", Expert Systems, Vol. 10, No. 1, 1993, pp 3 - 14.
114. Matson, Jack E., Barret, Bruce E., Mellichamp, Joseph M., "Software Development Cost Estimation Using Function Points", IEEE Transactions on Software Engineering, Vol. 20, No. 4, 1994, pp. 275 - 287.

115. Mazzucco, Frank A., "Automation of Function Point Counting - An Update", IFPUG, Spring Conference, Orlando, Floride, April 1990.
116. McNamara, Don, "IFPUG Survey of Function Point Use for Management Decisions", IFPUG Fall Conference, MontrEal, Oct. 1988.
117. Meredith, Denis C., "A View From the Field", System Development Function Points, August 1989, p.10
118. Miller, James C., "Measurement Using Function Point Analysis", IFPUG Spring Conf. April 1989.
119. Miluk, Gene, "Introduction to Function Points", Proceedings of the International Software Quality Conference", Dayton, Ohio, 1991, pp. 89-94.
120. Mullerburg, Monika, "Structured hypertext applied to function point analysis: a joint German-Quebec activity", Elletries in Software Evolution, GMD, CRIM. Sankt Augustin, 20.10.93.
121. Mullerburg, Monika, "The METKIT CAI System : Supporting functional point analysis", Montreal Trust. Montreal, 09.02.93.
122. Nishiyama, S., Furuyama, T., "The validity and applicability of function point analysis - as related to specification quality and ergonomics", Proc. of the Fourth European Conference of Software Quality, October 17-20, Basel, Switzerland, pp. 479-490.
123. Onvlee, Jolijn, "Use of Function Points for Estimation and Contracts", Proceedings of the Tenth Annual Conference on Application of Software Metrics and Quality Assurance in Industry, Amsterdam, 29 September - 1 October 1993, Section 13.
124. Paton, K., Abran, Alain, "A Formal Notation for the Rules of Function Points Analysis", Research Report, DEpartement d'informatique, UniversitE du QuEbec a Montreal, May 1995.
125. Pfleeger, Shari Lawrence, Palmer, J., "Software estimation for object-oriented systems", 1990 International Function Point Users Group Fall Conference, San Antonio, TX, 1990, 181-196.
126. Porter, Benjamin, "Function Point Measures - A Critical Comparison", 8th QAI International conf. on Measuring, Orlando Fl, March 1990.
127. Porter, Benjamin, "A Critical Comparison of Function Point Counting Techniques", IFPUG Fall Conference, Montreal, Canada, Oct 11-14, 1988.
128. Porter, Benjamin, "Using CASE to Count", IFPUG Spring Conf. Proceedings, Lake Buena Vista, Florida, April 1990.
129. Putnam, Lawrence H. "Tutorial - Software Cost Estimating and Life-Cycle Control: Getting the Software Numbers", IEEE, NY., 1980.
130. QAI, "Survey on Function Point Measurement", Quality Assurance Institute(QAI), Orlando,Florida, 1991
131. Rakos, John, "Using function point analysis can give you sharp estimates", Computing Canada, Feb 15, vol 19 no 4 p29(1), 1993.
132. Rains Ernie, "Function Points in an ADA Object-Oriented Design?", OOPS Messenger, Vol. 2, No. 4, 1991, pp. 23 - 25.
133. Rask R, Laamanen P. Lyytinen K, "Simulation and Comparison of Albrechts's Function Point and DeMarco's Function Bang Metrics in a CASE Environment", IEEE Transactions on Software Engineering, vol 19 no 7, p.661-671, July 1993.
134. Ratcliff, Bryan, Rollo, Anthony L., "Adapting Function Point analysis to Jackson system development", Softw. Eng. J. pp. 79-84, 1990.

135. Reifer, Donald J., "Asset-R: A function point sizing tool for scientific and real-time systems", Proceedings of the International Society of parametric analists, vol 3, n 1, may 1984.
136. Reifer, Donald J., "Real-time Function Point Extensions", IFPUG Spring Conference, Baltimore, Maryland, April, 1991.
137. Reifer, Donald J., "Asset-R: A function point sizing tool for scientific and real-time systems", JOURNAL SYST. SOFTWARE., vol. 11, no. 3, pp. 159-171, 1990.
138. Reinold, K., "Processes and metrics for object-oriented software development", OOPSLA `93 Workshop on Processes and Metrics for Object Oriented Software Development, Washington DC, 26 September, 1993.
139. Rollo, Anthony L., Ratcliff, Bryan,"Function Point Analysis and Jackson System Development", European COCOMO User's Group, May 1990.
140. Roman, David, "A measure of programming: function point analysis offers MIS managers a reliable way to measure programmer productivity - and to end beat-the-clock development", Computer Decisions, Jan 26, 1987 v19 n2 p32(2).
141. Rudolph, Eberhard .E., "Function Point Analysis, Cookbook", March 1983.
142. Rudolph, Eberhard .E., "Precision of Function Point Counts", IFPUG Spring Conference, San Diego, CA, April 1989.
143. Shepperd, Martin, "Some Observations on Function Points", Proc. of the 11th CSSR Conference on Software Evolution, Models and Metrics, September 7-9, Dublin, Ireland, Section 21, 1994.
144. Schofield, Joseph R., "Standardizing Complexity Characteristics in Function Point - A Process Improvement", 8th QAI International Conference on Measuring, Orlando, Fa, March 1990.
145. Shinn, Jon, "Measuring the Inspection Process with Regard to Project Size and Life Cycle Phases", QAI Conference, Orlando, FL, April 1990.
146. Snow, John R., "Management Reporting Using Function Points", System Development, ACR Publ. Phoenix, AZ, August 1989.
147. Symons, Charles R., "Function Points Analysis: Difficulties and Improvements, IEEE Transactions on Software Engineering, Vol. SE-14, no. 1, January 1988.
148. Symons, Charles R. "Mark II Function Points For Productivity Measurement & Estimating", Quality Assurance Institute: International Conf. on Project Management, Planning & Estimating, FL, May 1990.
149. Symons, Charles R., "Software Sizing and Estimating Mk II Function Point Analysis", John Wiley & Sons, First edition, ISBN 0-471-92985-9, 1991.
150. Tate, Graham, Verner, June M., "Approaches to measuring size of application products with CASE tools", Information and Software Technology, Volume 33 Number 9, November 91.
151. Thomson, Neil, Johnson, Rick, MacLeod, Ross, Miller, Granville, Hansen, Todd, "Project Estimation Using an Adaptation of Function Points and Use Cases for OO Projects", OOPSLA 94 Workshop on Pragmatic and Theoretical Directions in Object-Oriented Software Metrics, October 23, 1994.
152. Thurlow, John, "Effects of Measuring", IFPUG Conference, Spring, 1989, reported in Systems Development, Phoenix, Az, August 1989.
153.

154. Treble, Steve, Douglas Neil, "Sizing & estimating software in practice : Making Mk II Function Points Work", McGraw-Hill, ISBN 0-07-707620-6, 1995.
155. Umholtz, Donald C., Leitgeb, Arthur J., "Engineering Function Points and Tracking System", Crosstalk November 1994.
156. UK Function Point Users Group, "Mark II FPA Counting Practices Manual Version 1.1", October 1994.
157. Vacca, John R., "Function Points: The new measure of software", Computerworld, 18 november 1985.
158. Vacca, John R., "Function Point Analysis",DATAPRO : Application Development Software reports, report number 1055, June 92.
159. Verner, June M., Tate, Graham, "Estimating size and effort in fourth-generation development", IEEE Software, July 1988 v5 n4 p15(8).
160. Verner, June M., Tate, Graham, Jackson B., Hayward R. G., "Technology Dependence in Function Point Analysis: A Case Study and Critical Review", Proceedings of the 11th International Conference on Software Engineering, pp. 375--382, May 1989.
161. Whisehunt, Charlie, "How to Implement Function Points Both Ways (Right and Wrong) and still Survive", 8th QAI International Conference on Measuring, Orlando, Fl, March 1990.
162. Whitmire, Scott A, "3D Function Points: Scientific and Real-Time Extensions to Function Points", Pacific Northwest Software Quality Conference, 1992.
163. Whitmire, Scott A, "An Introduction to 3D Function Points", Software Development, pp 43-53, April 95
164. Whitmire, Scott A, "Applying Function Points to Object-Oriented Software" in Keyes, J. (ed.), Software Engineering Productivity Handbook, chapter 13.
165. Zuse, Horst, "Software Measures, the COCOMO-model and the Function Point Method from a Measurement Theoretic View".
166. Zwanzig, K (ed), "Handbook for estimation using function points", GUIDE Project DP-1234, Guide Int., November 1984.

En Francés

1. Abran, Alain, "L'implantation des points de fonction comme outil de gestion", Presentation au Comite des Responsables Informatiques du Secteur Public, Quebec, 6 Oct. 1989.
2. Abran, Alain, "Demonstration de la faisabilite et de l'utilite de l'implantation des metriques de logiciel des points de fonction dans l'entretien des systemes informatiques", Seminaire 3.6999, Ecole Polytechnique de Montreal, Dept. genie electrique, 17 Oct. 1989.
3. Abran, Alain, Robillard, Pierre N., "Analyse experimentale des modeles de mesure et de productivite des points de fonction par rapport a la relation avec l'effort", Rapport EPM/RT - 92/02, Ecole Polytechnique de Montreal, Septembre 1992.

4. Abran, Alain, Robillard, Pierre N., "Analyse comparative des points de fonction comme modele de productivite", Rapport EPM/RT - 92/03, Ecole Polytechnique de Montreal, Septembre 1992.
5. Abran, Alain, Robillard, Pierre N., "Analyse du processus de mesure de la metrique des points de fonction", Rapport EPM/RT - 92/01, Ecole Polytechnique de Montreal, Janvier 1992.
6. Abran, Alain, Robillard, Pierre N., "Analyse comparative de la fiabilte des points de fonction comme modele de productivite",ICO Revue de la liaison de la recherche en informatique cognitive des organisations,Vol. 4 nrs 3 & 4,Janvier 93.
7. Abran, Alain, "Analyse du processus de mesure des points de fonction", These de doctorat, Ecole Polytechnique de Montreal, Mars 1994, 405 pages.
8. Desharnais, Jean-Marc, "Analyse statistique de la productivite des projets de developpement en informatique a partir de la technique des points de fonction", maitrise en informatique de gestion a l'Universite du Quebec a Montreal, Decembre 1988.
9. Le Groupe DMR Inc., "Points de Fonction - Informations de Base", DMR, Montreal, Juin 1990.
10. Beaudoin, David, "Mesure de l'effort de developpement dans un environnement relationnel",Mars 92.
11. Roux, Frederic Georges, "La methode des points de fonctions d'Albrecht", L'informatique professionelle, Nr 86, Aug-90.
12. Sow, Thierno, "Dossier : la gestion de projets informatiques - de la gestion de projets informatiques", Journ'Almin, No 18, 1991.

En Holandés

1. Dam J V T, Langbroek P L, "Gebruik van functiepuntanalyse vraagt om beleid", Informatie,vol 34 number 6, Juni 92.
2. De Haas, B.G.M., "Functiepuntanalyse: een instrument om produktiviteit van automatisering te meten en projecten te begroten", Tijdschrift voor Produktiviteitsmanagement, nr. 2, pp. 5-8, 1986.
3. De Kater, A.L., "Functiepuntanalyse produktiviteitsmeting en budgettering van automatiseringsprojecten", Beleidsinformaticatijdschrift, vol 11, nr 2, 1985.
4. FPA Congres, "FPA in beweging", Proceedings van het Functiepuntanalysecongres, Nederlands Genootschap voor Informatica, november 1988.
5. Hermes, S., "Onderzoek naar de bruikbaarheid FPA in planningssituaties", computable, 23 september 1988, pp. 27-35.
6. Jolink D, "FPA: van begrotings- tot beheersinstrument", Computable,Jaargang 24, week 25
7. Klomp, Alberts, P.J.A., "Automatisch tellen: fictie of werkelijkheid?", FPA in beweging, Proceedings van het Functiepuntanalysecongres, Nederlands Genootschap voor Informatica, november 1988. pp 173-181, 1988.
8. Koning, Elmer, "FPA: de grenzen verleggen", FPA in beweging, Proceedings van het Functiepuntanalysecongres, Nederlands Genootschap voor Informatica, november 1988. pp 173-181, 1988.
9. Nefpug, "Definities en telrichtlijnen voor de toepassing van functiepuntanalyse: een handboek voor de praktijk, release 1", Vereniging Nederlandse Functiepuntgebruikers (NEFPUG), mei 1991.

10. Onvlee, Jolijn, Siskens W, "FPA voor CAM-software", Informatie,vol 34 number 6, Jun-92
11. Onvlee, Jolijn, "Teveel dialiecten in omloop van functiepuntanalyse", Computable, jaargang 24, 1 november 1991.
12. Poels, Geert , "Recente ontwikkelingen van functiepunt-analyse", Beleidsinformaticatijdschrift,Vol 19, nr 2 tweede kwartaal 1993
13. Rowold Paul, "Schatten en begroten van software-projecten : gegist bestek", Tutein Nolthenius - Amsterdam,ISBN 90-72194-08-X
14. Schimmel H.P. (ed), "Functiepuntanalyse", Samsom Uitgeverij, ISBN 90-14-04326-0, 1989.
15. Speyer, Th. J., "Functiepuntanalyse in de praktijk", Datex Informatica Instituut, 1983.
16. Van Straten, R., "Functiepuntanalyse: theorie, praktijk en resultaten", Informatie, jaargang 29, extra editie, pp. 619-628, 1987.
17. Van Wonderen Laurens J., "Een andere kijk op functiepuntanalyse", Informatie, vol 34 number 6, Jun-92
18. Zaal, R., "Over functiepunten valt niet te twisten", IT-forum 2, nr 6, pp. 32-35, 1990.

En Alemán

1. Dumke, Reiner, Zuse, Horst, "Theorie und Praxis der Softwaremessung", Deutscher Universitaetsverlag, Wiesbaden, 1994.
2. Frach, K., "Complexity and effort in the development of a large scale software project (german)", Study, IBM Hamburg, Technical University of Magdeburg, 1993
3. Grossjohann, R., "Significance of the Function Point Method under Recession", in: Dumke/Zuse: Theorie und Praxis der Softwaremessung, Deutscher Universitaetsverlag, Wiesbaden, 1994, pp.20-34 .
4. Huerten, R., "Man month and lines of code are secundary measures" (german). Computerwoche, 46(1992) Nov., pp. 13-14
5. Knoell H.-D. Busse J., "Aufwandsschaetzung von Software-Projekten in der Praxis", BI, ISBN 3-411-14341-X, 1991.
6. Kronsberg Frank, "Projektmanagement und Softwareengineering", Braunschweig, 1987.
7. Kummrow Frank, "Implementierung der Function Point Methode zur Aufwandsabschaetzung von EDV-Projekten", Braunschweig, 1990.
8. Kuhl Stefan, "Softwarekosten-Abschaetzung fuer technisch-wissenschaftliche Software", Braunschweig, 1991.
9. Noth Thomas, Kretzschmar Mathias, "Aufwandsschaetzung von DV-Projekten", Springer, Berlin, ISBN 3-540-16069-8, 1986.

10. Volkswagen, "The function point method and his application (german)", Volkswagen AG, Wolfsburg 1989 .

Otras referencias

1. University of Quebec - Software Engineering Management Research Laboratory Bibliography
2. University of Magdeburg - Software Metrics - A subdivided bibliography
3. South Bank University - Object-Oriented Metrics: People and Publications
4. University of Mainz - Papers on OO Metrics
5. University of Southern California - Cocomo 2.0 Bibliography
6. Software Productivity Research Inc. - Articles, Books, and White Papers

2.12 *Prácticas*

Como complemento al capítulo, se recomiendan las siguientes prácticas:

1. Estimar el número de Puntos-Función de una aplicación conocida, y obtener el número de líneas de código estimadas.

2. Aplicar el Modelo Cocomo, utilizando como origen el número de instrucciones obtenido en la evaluación anterior.

3. Repetir la estimación utilizando el Modelo de Rayleigh-Norden

4. Comparar los resultados obtenidos y ajustar las estimaciones en caso de fuertes discrepancias, para obtener la mejor estimación.