MODELOS DE ESTIMACION DE COSTO Y ESFUERZO EN PROYECTOS DE SOFTWARE

RAFAEL ANTONIO GORDILLO
ALBERTO NOE GIRALDO
LUIS BALDOMERO DAVILA

Alumnos del curso de Investigación de VIII Semestre de Ingeniería de Sistemas del ICESI.

INTRODUCCION

En todo proyecto de desarrollo de software están incluidos factores que afectan, de una u otra forma, el costo final del producto, así como también los requerimientos de personal. El factor tiempo también ha de tenerse en cuenta a la hora de evaluar las características finales del producto. Es necesario, por lo tanto, contar con herramientas y técnicas que permitan estimar el costo, el esfuerzo y el tiempo inherentes al desarrollo de un producto de software.

Este documento presenta una descripción de los principales modelos utilizados en la estimación de costo, esfuerzo y tiempo de desarrollo, así como los parámetros y variables en que se basan estos modelos.

El principal modelo que se trata en este documento es el Modelo COCOMO, propuesto por Barry Boehm, puesto que, en la actualidad, es el más utilizado y referenciado. Este modelo no sólo aporta un conjunto de ecuaciones sino que, además, permite presentar razonadamente el porqué de sus resultados, comprobar las diferentes valoraciones que aportan sus tablas e identificar claramente los factores de mayor influencia en el proceso de desarrollo.

Ya que los modelos basan sus estimaciones en la determinación previa del tamaño del producto, se tratará la metodología de los Puntos de Función (PF) desarrollada por A. Albrecht, que permite valorar el tamaño de una aplicación, en las etapas previas de su desarrollo.

OBJETIVO GENERAL

Elaborar un documento sobre los diferentes modelos y herramientas que existen para la estimación de costo, esfuerzo y tiempo en el desarrollo de software, el cual sirva de guía, para su aplicación, a quienes desarrollan productos de software.

OBJETIVOS ESPECIFICOS

1. Identificar los diferentes modelos existentes para la estimación de costos y esfuerzos para el desarrollo de software.
2. Identificar los principales parámetros y variables utilizados por los modelos de estimación.

3. Determinar las condiciones básicas que permitan realizar estimaciones más confiables.
4. Definir en qué etapas del desarrollo de un proyecto de software son aplicables las estimaciones de los modelos.

1. ESTIMACION EN LOS PROYECTOS DE SOFTWARE

Aunque la estimación es más un arte que una ciencia, es una actividad importante que no debe llevarse a cabo de forma descuidada. Existen técnicas útiles para la estimación de costos y tiempos. Y dado que la estimación es la base de todas las demás actividades de la planificación y que la planificación del proyecto sirve como guía para una buena ingeniería del software, no es en absoluto aconsejable embarcarse sin ella.

Un proyecto de software generalmente comienza con la producción de un plan de desarrollo, donde se identifica el trabajo por ser realizado, el presupuesto y el itinerario. La efectividad de la planeación depende de una estimación correcta.

Una importante faceta en el desarrollo de software es la capacidad de estimar el costo asociado con las primeras etapas del ciclo de vida. La estimación de recursos, costos e itinerarios requiere experiencia y acceso a una buena información histórica. Además, esta estimación conlleva un riesgo inherente, y los factores que lo aumentan son:

- **La complejidad del proyecto:** es una medida relativa que se ve afectada por la familiaridad con anteriores esfuerzos.
- **El tamaño del proyecto:** es el factor más importante que afecta la precisión y la eficacia de las estimaciones. A medida que crece el tamaño, la interdependencia entre los distintos elementos del software crece rápidamente. Sin embargo, estimar el tamaño del software es un problema complicado que requiere conocimiento específico de las funciones del sistema en términos del entorno, complejidad e interacciones. Las medidas más frecuentes utilizadas son Líneas de Código (LDC) y Análisis de puntos de función (PF).
- **El grado de estructuración del proyecto:** en este contexto, la estructuración se refiere a la facilidad con que las funciones pueden ser modularizadas y a la naturaleza jerárquica de la información que debe ser procesada. A medida que aumenta el grado de estructuración, la posibilidad de estimar con precisión mejora y el riesgo disminuye.

La disponibilidad de información histórica también determina el riesgo de la estimación. Cuando se dispone de una amplia métrica del software de proyectos pasados se pueden hacer las estimaciones con gran seguridad, se pueden establecer métodos para evitar anteriores dificultades y se puede reducir el riesgo global.

1.1. Recursos humanos

Es necesario evaluar el entorno de desarrollo y seleccionar las habilidades técnicas que se requieren para llevar a cabo el desarrollo. Hay que especificar tanto la posición dentro de la organización como la especialidad.

El número de personas requerido para un proyecto de software sólo puede ser determinado después de hacer una estimación del esfuerzo del desarrollo (por ejemplo, personas-mes o personas-año).

1.2. Categorías del hardware

Durante la planificación del proyecto de software se deben considerar tres categorías de hardware: el **sistema de desarrollo,** que está compuesto por la computadora que se utilizará durante

la fase de desarrollo del software y sus periféricos asociados; la **máquina objetivo,** que es el equipo donde se ejecutará el software desarrollado, y los demás elementos de hardware del nuevo sistema.

1.3. Recursos del software

Al igual que utilizamos el hardware como herramienta para construir nuevo hardware, utilizamos el software como ayuda en el desarrollo de nuevo software. La primera aplicación que se le dio al software en el desarrollo de software fue lo que se denominaba **reconstrucción.** Se comenzó por escribir un primitivo traductor de lenguaje ensamblador a lenguaje máquina, y se usó para desarrollar un ensamblador más sofisticado. Aumentando las posibilidades de cada versión previa, los equipos de desarrollo fueron reconstruyendo eventualmente el software, hasta llegar a construir compiladores de lenguajes de alto nivel y otras herramientas.

1.4. Software reutilizable

Cualquier estudio sobre recursos de software no estará completo sino se considera la **reusabilidad**, esto es, la creación y reutilización de módulos constructivos de software. Estos módulos constructivos deben ser catalogados para una fácil referencia, estandarizados para una fácil aplicación y validados para una fácil integración.

La mayoría de los observadores de la industria del software están de acuerdo en que la mejora de la productividad del desarrollo del software y de la calidad de los productos minimizará el nivel de mantenimiento. En un mundo así, abundaría el software reutilizable. Los módulos constructivos de software estarían disponibles para la construcción de grandes programas, con un mínimo esfuerzo de desarrollo, partiendo "de la nada".

1.5. Formas de estimación

La estimación del costo y del esfuerzo del software nunca será una ciencia exacta. Son demasiadas las variables (humanas, técnicas, de entorno, políticas) que pueden afectar el costo final del software y el esfuerzo aplicado para desarrollarlo. Sin embargo, la estimación del proyecto de software puede dejar de ser un oscuro arte para convertirse en una serie de pasos sistemáticos que proporcionen estimaciones con un grado de riesgo aceptable.

Para realizar estimaciones seguras de costos y esfuerzos tenemos varias opciones posibles:

1. Dejar la estimación para más adelante (obviamente, podemos realizar una estimación del 100% fiable, tras haber terminado el proyecto).
2. Utilizar técnicas de descomposición relativamente sencillas, para generar las estimaciones de costo y esfuerzo del proyecto.
3. Desarrollar un modelo empírico, para el cálculo de costos y esfuerzos del software.
4. Adquirir una o varias herramientas automáticas de estimación.

Infortunadamente la primera opción, aunque atractiva, no es práctica. Las estimaciones de costos han de ser proporcionadas de antemano. Sin embargo, hay que reconocer que cuanto más tiempo esperemos más cosas sabremos, y cuanto más sepamos menor será la probabilidad de cometer serios errores en nuestras estimaciones.

Las técnicas de descomposición utilizan un enfoque de "divide y vencerás", para la estimación del proyecto de software. Mediante la descomposición del proyecto en sus funciones principales y en las tareas de ingeniería de software que le corresponden, la estimación del costo y del esfuerzo pueden realizarse de una forma escalonada e idónea.

Igualmente, se pueden utilizar los modelos empíricos de estimación como complemento de las técnicas de descomposición. Cada modelo se basa en la experiencia (datos históricos) y toma como base: **$d=f(v_i)$ donde:**

d: es uno de los valores estimados (por ejemplo: esfuerzo, costo, duración del proyecto).

v_i: son determinados parámetros independientes (por ejemplo: Líneas de código o Puntos de función estimados).

Las herramientas automáticas de estimación implementan una o varias técnicas de descomposición o modelos empíricos. Cuando se combinan con una interfaz atractiva hombre-máquina, las herramientas automáticas resultan muy atractivas para la estimación.

Cada una de las opciones viables para la estimación de costos y esfuerzo del software sólo será buena si los datos históricos que se utilizan como base de la estimación son buenos. Si no existen datos históricos, la estimación del costo descansará sobre una base muy inestable.

1.6. Parámetros y variables utilizados en la estimación

Las estimaciones que suelen realizarse para el software involucran variables y constantes, además de datos históricos. Se suele utilizar el supuesto volumen del software (líneas de código-LDC), con otras entradas que caracterizan los factores de riesgos principales en el desarrollo.

Debido a que las estimaciones de costo y esfuerzo para un proyecto de software son demasiado complejas si se consideran como un solo problema, se hace necesario descomponerlo en problemas más pequeños. Las técnicas de estimación, las líneas de código LDC y los puntos de función PF difieren en el nivel de detalle que requiere la descomposición del problema. Los datos de LDC y PF se emplean de dos formas en el proceso de estimación: como **variables de estimación** y como **métricas de base,** recogidas de anteriores proyectos.

Debe tenerse en cuenta que mientras el LDC se estima directamente, los PF se determinan indirectamente, mediante la estimación del número de entradas, salidas, archivos de datos, peticiones e interfaces externas.

1.6.1. *Líneas de código (LDC)*

La métrica de tamaño tradicional para estimar tamaño, esfuerzo y para medir productividad ha sido el de las líneas de código. Un gran número de modelos de estimación de costo ha sido propuesto, muchos de los cuales están fundamentados en las líneas de código (o miles de líneas de código - KLDC). Generalmente, los modelos de estimación de esfuerzo se componen de dos partes; la primera ofrece una base de estimación en función del tamaño del software y es de la forma:

$$E=A+B\ (KLDC)^C$$

Donde E es el esfuerzo estimado en hombres-mes; **A, B** y **C** son constantes y **KLDC** es el número estimado en miles de líneas de código en el sistema final.

En la segunda parte, se ajusta la base de estimación para responder a la influencia de factores ambientales. Entre estos factores ambientales se incluyen el uso de técnicas, tales como el código estructurado, el diseño ***Top-Down***, la revisión estructurada y los equipos de programadores especializados; la capacidad de personal; los requerimientos de hardware. Por ejemplo, el Modelo COCOMO usa líneas de código elevadas a una potencia entre 1.05 y 1.20, para determinar la base de estimación.

El exponente específicamente depende de la simplicidad promedio o de la complejidad del proyecto.

Los siguientes son ejemplos de modelos típicos:

$E=5.2(KLDC)^{0.91}$	Modelo Wlaston-Félix
$E=5.5+0.73(KLDC)^{1.16}$	Modelo Bailey-Basili
$E=3.2(KLDC)^{1.05}$	Modelo COCOMO Básico
$E=3.0(KLDC)^{1.12}$	Modelo COCOMO Intermedio
$E=2.8(KLDC)^{1.2}$	Modelo COCOMO Avanzado
$E=5.288(KLDC)^{1.044}$	Modelo Doty.

La definición de KLDC es importante a la hora de comparar estos modelos. Algunos modelos incluyen líneas de comentarios y otros no. La definición de qué tipo de esfuerzo se ha estimado es igualmente importante, ya que el esfuerzo se puede asumir como la sola codificación o como el análisis total (el diseño, la codificación y la fase de pruebas conjuntamente). Esto nos indica que la comparación de modelos que usan criterios distintos, en cuanto a los mismos parámetros, es relativamente difícil.

Hay numerosos problemas con el uso de líneas de código como unidad de medida para el tamaño del software. El primero consiste en establecer una definición exacta de lo que es una línea de código y, a la vez, que esta definición sea aceptada universalmente. Otra dificultad con las líneas de código es su dependencia respecto del lenguaje utilizado. No es posible comparar directamente proyectos desarrollados en lenguajes diferentes. Por ejemplo, el tiempo por línea para un lenguaje de alto nivel puede ser mayor que para uno de bajo nivel. Puede que no sea posible lograr un mínimo de líneas de código en un lenguaje de alto nivel para una función de un lenguaje de bajo nivel.

Todavía se cuenta con un problema más y es el de estimar el número de líneas requeridas para desarrollar un sistema, contando sólo con la información proveniente de la fase de requerimientos o diseño. Si los modelos para costos, basados en el tamaño, son utilizados, es necesario tener la capacidad de predecir el tamaño del producto final tan pronto como sea posible. Es aquí donde entra en juego la recolección de datos. Infortunadamente la estimación del tamaño, usando las métricas LDC, depende de experiencias previas con proyectos similares, más específicamente de datos históricos.

Una forma de estimar el tamaño del proyecto, en líneas de código LDC, consiste en descomponer el proyecto en sus funciones principales y se estiman las LDC para cada una de éstas. Las estimaciones de las funciones se combinan para producir una estimación global del proyecto. A partir de los datos históricos, se estima tanto el valor optimista, más probable,como el pesimista del LDC. Si no se cuenta con datos históricos no queda más remedio que recurrir a la intuición para los valores antes mencionados. Luego, se calcula el **valor agregado** del LDC, el cual será una media ponderada de las estimaciones pesimistas (a), más probable (m) y optimista (b), así:

$$E=(a+4m+b)/6$$

Por ejemplo, consideremos un paquete de software por desarrollar, para una aplicación de diseño asistido por computador (CAD). Supongamos que las funciones principales para este proyecto son:

- Interfaz de usuario y facilidad de control.
- Análisis geométrico bidimensional.
- Análisis geométrico tridimensional.

- Control de periféricos.
- Módulos de análisis de diseño.

El paso siguiente consiste en establecer los valores optimista, probable y pesimista para cada una de las funciones anteriores. Tomemos por ejemplo el control de periféricos, para el cual se tiene un valor optimista de 2.000 LDC, un valor probable de 2.100 LDC y un valor pesimista de 2.450 LDC. Aplicamos entonces la función del valor esperado y obtenemos el siguiente resultado:

Función	Optimista	Probable	Pesimista	Esperado
Control de periféricos	2.000	2.100	2.450	2.140

Realizamos el mismo proceso con las demás funciones y construimos una tabla como la siguiente:

Función	Optimista	Probable	Pesimista	Esperado
Interfaz de usuario y facilidad de control	1.800	2.400	2.650	2.340
Análisis geométrico bidimensional	4.100	5.200	7.400	5.380
Análisis geométrico tridimensional	4.600	6.900	8.600	6.800
Control de periféricos	2.000	2.100	2.450	2.140
Módulos de análisis de diseño	6.600	8.500	9.800	8.400
Total				25.060

Sumando verticalmente, en la columna del valor esperado, se establece una estimación de 25.060 LDC.

1.6.2. *Análisis del punto de función*

El analisis del punto de función es un método de cuantificación del tamaño y complejidad del software, en términos de las funciones que el sistema entrega al usuario. La función entregada no está relacionada con el lenguaje o herramientas utilizadas en el desarrollo del proyecto. Este análisis está diseñado para aplicaciones comerciales; no es apropiado para aplicaciones de tipo técnico o científico, ya que éstas tratan con algoritmos complejos, que los puntos de función no están diseñados para manejar.

En este método se contabilizan los varios tipos de función de usuarios, los cuales se dividen en datos y transacciones. Los tipos de función de datos corresponden a Archivos lógicos internos yArchivos externos de interfase. Los tipos de función de transacciones corresponden a entradas, salidas y consultas externas. Todos estos términos serán tratados más adelante.

El enfoque de puntos de función (PF) tiene características que se sobreponen a las deficiencias en el uso de las líneas de código, mencionadas anteriormente. Primero, los puntos de función son independientes de la metodología, las herramientas y los lenguajes utilizados en el desarrollo del proyecto. Segundo, los puntos de función pueden estimarse con base en la especificación de los requerimientos o en el diseño; además, hacen posible la estimación del esfuerzo para el proyecto en las primeras fases del desarrollo. Ya que los puntos de función están directamente relacionados con la especificación de requerimientos, cualquier cambio en éstos puede ser seguido fácilmente por una nueva estimación. Tercero, los usuarios no especializados en el sistema tienen una mejor comprensión de qué están midiendo los puntos de función.

1.7. Estimación del costo del software

Básicamente, la estimación del costo se hace multiplicando las unidades del tamaño del software para factores apropiados de productividad. Muchas ecuaciones del costo tienen la forma:

$E=pL^c$ donde:

E=esfuerzo

p=ajuste de productividad

L=tamaño en líneas de código

c>1, una constante.

El ajuste de productividad puede ser un factor simple o compuesto de factores, tal como aparece en el Método COCOMO.

Los principales factores para determinar la productividad son, en su orden de importancia, la capacidad del personal, la complejidad de la aplicación, el software reutilizable y la tecnología aplicada.

La productividad se ve muy influenciada por el tipo de personal escogido para realizar el trabajo; luego está el tipo de trabajo ejecutado. Por ejemplo, COCOMO prefiere el ajuste de la estimación del tamaño (DSI) para contabilizar la reutilización antes de aplicar cualquier factor de productividad.

1.8 Estimación del esfuerzo

La estimación del esfuerzo es la técnica más utilizada para calcular el costo de un proyecto de ingeniería del software. En la resolución de cada tarea del proyecto se aplica un número determinado de personas/día,personas/mes o personas/año. Se asocia un costo a cada unidad de esfuerzo y se deriva el costo total estimado.

Al igual que las técnicas LDC y PF, la estimación del esfuerzo comienza con una delimitación de las funciones del software, obtenidas del ámbito del software. Para cada función debe realizarse una serie de tareas de ingeniería del software, tales como análisis de requisitos, diseño, implementación y pruebas. Se aplican las tarifas laborales (costo/unidad de esfuerzo) a cada una de las tareas de ingeniería del software.

En el último paso se calculan los costos y el esfuerzo para cada función y se deriva el costo total del proyecto.

1.9. Modelos empíricos de estimación

Un modelo de estimación utiliza fórmulas derivadas empíricamente para predecir los datos que se requieren, en el paso de planificación del proyecto de software. Los datos empíricos que soportan la mayoría de los modelos se obtienen de una muestra de proyectos limitada. Se identifican cuatro clases de modelos: **modelos univariables estáticos, modelos multivariables estáticos, modelos multivariables dinámicos y modelos teóricos.**

Un modelo univariable estático toma la forma:

Recurso $=c_1$ x(característica estimada)^{c_2}

donde el **recurso** podría ser el esfuerzo, la duración del proyecto, la cantidad de personal o las líneas requeridas de documentación del software. Las constantes c_1 y c_2 se derivan de los datos recopilados de los anteriores proyectos. La *característica estimada* puede ser la cantidad de líneas del código fuente, el esfuerzo (si ya está estimado) u otra

característica del software. Un ejemplo de este tipo de modelo es la versión básica del modelo de costo constructivo o COCOMO.

Los **modelos multivariables estáticos** emplean los datos históricos para obtener relaciones empíricas. Un modelo de esta categoría toma la forma:

$$\text{Recurso} = c_i1e_i + c_i2e2 + \ldots$$

donde e_i es la característica i-ésima del software y $c_i1 + c_i2$ con constantes obtenidas para e_i.

Los **modelos multivariables dinámicos** proyectan los requisitos de recursos como una función del tiempo. Los recursos se definen en una serie de pasos consecutivos en el tiempo, que asignan cierto porcentaje de esfuerzo (o de otro recurso) a cada etapa del proceso de ingeniería del software. Este modelo incluye una "curva continua de utilización del recurso" como hipótesis, y a partir de ella obtiene ecuaciones que modelizan el comportamiento del recurso.

El **modelo teórico de recurso** examina el software desde un punto de vista microscópico; es decir, las características del código-fuente (por ejemplo: el número de operadores y operandos).

2. MODELO COCOMO

Es el modelo más completo existente hasta el momento y ha sido propuesto por Barry Boehm en su libro *Software Engineering Economics*. En este libro, Boehm trata el ciclo de vida del software desde una perspectiva económica, en términos del tiempo y el esfuerzo requeridos para completar cada fase del ciclo de vida del software. Establece una relación matemática que permite estimar el esfuerzo y el tiempo requerido para desarrollar un proyecto, en función del número de instrucciones-fuente desarrolladas.

El modelo inicial, que tenía un único modo de desarrollo, se ajustó utilizando los datos correspondientes a doce proyectos completos. El modelo obtenido fue evaluado contrastando sus resultados frente a los obtenidos en otros 36 proyectos.

Sin embargo, la aplicación del modelo a otros proyectos, en una amplia variedad de entornos, indicaron que un único modo de desarrollo no era suficiente para explicar las variaciones observadas, lo que indujo a introducir en el modelo actual tres modos de desarrollo, de forma que sus predicciones pudiesen aplicarse, con un buen grado de precisión, en cualquier entorno de desarrollo.

Boehm propuso entonces una jerarquía de modelos de estimación para el software denominada COCOMO, por Constructive COst MOdel, o Modelo Constructivo de Costo; dicha jerarquía de modelos está constituida así:

Modelo básico

Modelo intermedio

Modelo avanzado o detallado.

Para poder aplicar adecuadamente estos modelos se deben tener en cuenta las siguientes consideraciones:

1. El factor principal sobre el que se basan las estimaciones de costos es el tamaño del producto, es decir, el número de instrucciones-fuente entregadas DSI *(Delivered Source Instructions)*.

   Este término incluye todas las instrucciones creadas por el personal asignado al proyecto, y procesadas en código máquina, excluyendo las líneas de comentarios.

   Igualmente, deben considerarse las sentencias de Lenguaje de Control de Trabajos (JCL), las sentencias de formatos y las declaraciones de datos.

   Las instrucciones se definen como líneas de código y, por tanto, toda línea que contenga dos o tres sentencias se considerará como una sola instrucción.

2. COCOMO considera sólo las fases del período de desarrollo, comprendidas desde el comienzo de la fase de diseño del producto hasta el final de la fase de integración y prueba.

   Los costos, esfuerzo y tiempo de las otras fases (planificación, especificaciones, mantenimiento, etc.) deben estimarse por separado.

3. Los costos estimados mediante este modelo no incluyen los correspondientes a las actividades de formación del usuario, la instalación y los trabajos de conversión, aunque estas actividades se realicen durante la etapa de desarrollo. Se cubren todos los costos directos del proyecto; es decir, la gestión del proyecto, los trabajos de documentación y librería, pero se excluyen los correspondientes al personal del centro de cómputo, las secretarias, los subalternos y los altos directivos que no estén directamente involucrados en el desarrollo del proyecto.

4. La unidad de esfuerzo HM (Hombre-Mes) supone un total de 152 horas de trabajo por persona, valor tomado basándose en la experiencia práctica, que considera aspectos tales como vacaciones, permisos, festivos, licencias, etc.

   Para convertir las unidades HM a otras unidades empleamos las siguientes equivalencias:

   1 HM = 152 MH (MH: hombre-hora)
   1 HM = 19 MD (MD: hombre-día)
   1 HM = 1 MY/12 (MY-Hombre-año).

5. Se supone que después de la fase de especificación de requerimientos, éstas no cambiarán substancialmente, aunque evidentemente esto será inevitable. En el caso que se produzcan modificaciones significativas, se estaría obligado a revisar las estimaciones anteriores.

6. El modelo avanzado del COCOMO asume que los factores que influyen sobre los costos de desarrollo dependen de la fase en que aparecen. El COCOMO básico y el intermedio no consideran los factores que afectan el costo, excepto para distinguir entre desarrollo y mantenimiento.

7. En los costos por fase se incluyen todos aquellos que se produzcan durante ese espacio de tiempo. Es decir que los costos relativos a los trabajos de actualización del plan de integración y prueba, la terminación del plan de pruebas de aceptación, la actualización de la documentación, etc., se incluyen dentro de los costos de la fase de diseño detallado.

8. Para convertir las estimaciones obtenidas de hombres-mes (HM) a unidades monetarias, la mejor forma es aplicar un valor medio de unidades monetarias, por unidad de esfuerzo, para cada una de las fases en el desarrollo.

2.1. Modos de desarrollo

Como se mencionó anteriormente, COCOMO comprende tres modos de desarrollo que de algún modo reflejan las diferentes condiciones o entornos de desarrollo, y aunque matemáticamente pueden expresarse en forma similar, conducen a estimaciones diferentes; estos modos son los siguientes:

2.1.1. *Modo orgánico*

Este modo abarca sólo los proyectos relativamente pequeños y sencillos; en éstos trabajan pequeños equipos, con buena experiencia en otros proyectos relacionados con la misma organi-

zación y con pleno conocimiento de cómo el sistema en desarrollo contribuirá con los objetivos de la organización.

Esto significa que la mayoría de las personas pueden contribuir de forma efectiva a la terminación puntual de cada una de las etapas, sin necesidad de mucha comunicación, para determinar con precisión las tareas que cada uno debe desarrollar en el proyecto. Existe, por lo tanto, facilidad para establecer los requisitos y las especificaciones de cada una de las fases del proyecto.

Otros factores característicos de este modo son:

- Un ambiente generalmente estable de desarrollo, con muy poca ocurrencia de nuevo hardware y procedimientos operacionales.
- Mínima necesidad para innovar arquitectura de procesamiento de datos o algoritmos.
- Muy pocos proyectos, en este modo, han desarrollado productos con más de cincuenta KLDC.

2.1.2. *Modo semilibre*

Representa un estado intermedio entre el modo orgánico y el modo restringido. Son proyectos intermedios en tamaño y complejidad en los que equipos de trabajo, con variados niveles de experiencia, deben satisfacer requisitos poco o medio rígidos.

Con respecto a la experiencia del equipo de trabajo, se consideran las siguientes situaciones:

- Todos los miembros del equipo tienen un nivel intermedio de experiencia en sistemas relacionados con el proyecto.
- El equipo de desarrollo está formado por una mezcla de gente experta e inexperta.
- Los miembros del equipo tienen experiencia en algunos de los aspectos del sistema que se pretende desarrollar, pero no en todos.

Un típico proyecto, en modo semilibre, puede ser un sistema de proceso de transacciones con pocas interfaces rigurosas (por ejemplo con la terminal hardware) y algunas otras interfaces muy flexibles (naturaleza y formatos de presentación de mensajes). El término semilibre hace referencia a la flexibilidad parcial que presenta un proyecto de este tipo, el cual generalmente sobrepasa las 300 KLDC.

2.1.3. *Modo restringido*

La principal característica de un proyecto, en este modo, es la necesidad de operar dentro de fuertes restricciones. El término incorporado se refiere básicamente al hecho que el producto debe operar dentro de un complejo altamente interconectado de hardware, software, reglas y procedimientos operacionales, como es el caso, por ejemplo, de un sistema de control de tráfico aéreo.

En condiciones de modo de desarrollo restringido, el costo de modificar parte del complejo sistema es tan alto, que sus características se podrían considerar inmodificables. Como resultado, este modo de proyecto no siempre tiene la opción de realizar fácilmente cambios de software, por más sencillos que sean, debido a la modificación de los requerimientos y la especificación de las interfaces. Por lo tanto, el proyecto debe emplear más esfuerzo en realizar y adecuar los cambios y correcciones, y en asegurar que el software actualmente satisfaga las especificaciones. También se debe emplear más esfuerzo en verificar que los cambios se realicen correctamente.

Los proyectos, en este modo, generalmente abarcan áreas más amplias y menos conocidas que en los casos anteriores.

Los modelos COCOMO calculan esencialmente el costo a la entrega, que puede ser una pequeña proporción del costo total de la vida del software.

2.2. **Modelo COCOMO básico**

Es un modelo univariable, estático, que calcula el costo y el esfuerzo de un proyecto de software, exclusivamente en función de su tamaño, expresado en líneas de código (KLDC) estimadas. Este modelo es apropiado para una pronta y rápida estimación del costo del software, pero su precisión es necesariamente limitada, porque carece de factores que involucren los aspectos del hardware, la calidad del personal, la experiencia, el uso de modernas herramientas y técnicas y otros tributos del proyecto que tienen una significativa influencia en los costos del software.

La siguiente tabla presenta las ecuaciones para los tres modos de desarrollo del modelo COCOMO básico:

Modo	Esfuerzo	Tiempo estimado
Orgánico	$E=2.4(KLDC)^{1.05}$	$T=2.5(E)^{0.38}$
Semilibre	$E=3.0(KLDC)^{1.12}$	$T=2.5(E)^{0.35}$
Restringido	$E=3.6(KLDC)^{1.20}$	$T=2.5(E)^{0.32}$

La ecuación **$E=2.4(KLDC)^{1.05}$** es utilizada para la estimación de hombres-mes (HM) requeridos para desarrollar el tipo más común de productos de software, en términos de miles de instrucciones-fuente entregadas KLDC.

Se obtiene también que **$T=2.5(HM)^{0.38}$** es una ecuación para estimar el tiempo de desarrollo, en meses.

Estas ecuaciones son aplicables a la mayoría de los proyectos de software, de tamaño mediano/pequeño, desarrollados en un ambiente de software familiar. Ejemplo:

Du Bridge Chemical Inc. es una importante compañía de productos químicos, que planea desarrollar una aplicación para mantener la información sobre materias primas. Se desarrollará una aplicación por un equipo propio de programadores y analistas, quienes han desarrollado aplicaciones similares durante varios años. Además, es un buen ejemplo de un software de modo orgánico. Un estudio inicial ha determinado que el tamaño del programa será aproximadamente de 32.000 líneas de código (32 KLDC). De las ecuaciones básicas obtenemos las características del proyecto, que son:

esfuerzo:	$E=2.4(32)^{1.05}$ =91 hombres-mes
productividad:	32.000 LDC/91HM=352 LDC/HM
tiempo de desarrollo:	$T=2.5\ (91)^{0.38}$= 14 meses,

y el valor medio del número de personas que, de tiempo completo, serían necesarias para desarrollar el proyecto es:

P_{ε}= 91 HM14 meses = 65 FSP

FSP: *Full-time-equivalent Personnel:* personal de tiempo completo para el software.

E=91 HM es el esfuerzo total en hombres-mes necesario en el proyecto.

T=14 meses es el tiempo de desarrollo, expresado en meses.

El Modelo COCOMO ofrece perfiles obtenidos de aplicar las anteriores ecuaciones a proyectos de tamaño estándar. Los tamaños estándares, utilizados por el modelo, son los siguientes:

Pequeño 2.000 líneas de código

Intermedio 8.000 líneas de código

Medio 32.000 líneas de código

Grande 128.000 líneas de código.

En la siguiente tabla puede observarse que un proyecto pequeño es esencialmente un trabajo de una persona,

mientras que un proyecto considerado grande requiere un nivel medio de 16 personas. Cabe anotar que para un proyecto pequeño el esfuerzo estimado de 5 HM hace referencia al desarrollo de 2.000 instrucciones del producto, incluyendo, por lo tanto, el esfuerzo de documentación, las pruebas, las correcciones, etc.

Tamaño	Esfuerzo (HM)	Productividad (línea/HM)	Tiempo(meses)	P_E
Pequeño 2 KLDC	5.0	400	4.6	1.1
Intermedio 8 KLDC	21.3	376	8.0	2.7
Medio 32 KLDC	91.0	352	14.0	6.5
Grande 132 KLDC	392.0	327	24.0	16.0

Obviamente, el tiempo para desarrollar un programa, por ejemplo de 2000 instrucciones para uso personal, será menor, ya que éste no requerirá de muchas exigencias, mientras que si el mismo se desarrolla como un producto profesional, entonces requerirá un poco más de tiempo y esfuerzo.

2.2.1. *Distribución del esfuerzo y tiempos por fases*

Es de esperar que las estimaciones para cada una de las fases del proyecto difieran considerablemente de un modo de desarrollo a otro. La distribución por fases varía en función del tamaño; los proyectos de gran tamaño precisan mayor tiempo y esfuerzo para desarrollar actividades, tales como la integración y la prueba, mientras que pueden reducir el tiempo durante la fase de programación, distribuyendo esta actividad entre un número mayor de programadores. Los pequeños proyectos presentan una distribución más uniforme y tienen que dedicar, relativamente, más recursos a las fases de diseño y programación que a las de prueba e integración.

Las siguientes tres tablas presentan los porcentajes de distribución del esfuerzo de desarrollo, entre las distintas fases, en los tres modos de desarrollo.

Distribución del esfuerzo estimado (%). Modo orgánico

FASE	TAMAÑO (KLDC)				
	Pequeño (2)	Interm. (8)	Medio (32)	Grande (128)	Muy grande (512)
Planificación y especificaciones	6	6	6	6	–
Diseño del producto	16	16	16	16	–
Programación	68	65	62	59	–
Diseño detallado	26	25	24	23	–
Codificación y prueba de unidades	42	40	38	36	–
Integración y pruebas	16	19	22	25	–

Distribución del esfuerzo estimado (%). Modo semilibre

FASE	TAMAÑO (KLDC)				
	Pequeño (2)	Interm. (8)	Medio (32)	Grande (128)	Muy grande (512)
Planificación y especificaciones	7	7	7	7	7
Diseño del producto	17	17	17	17	17
Programación	64	61	58	55	52
Diseño detallado	27	26	25	24	23
Codificación y pruebas de unidades	37	35	33	31	29
Integración y pruebas	19	22	25	28	31

Distribución del esfuerzo estimado (%). Modo restringido

Fase	Tamaño (KLDC)				
	Pequeño (2)	Interm. (8)	Medio (32)	Grande (128)	Muy grande (512)
Planificación y especificaciones	8	8	8	8	8
Diseño del producto	18	18	18	18	18
Programación	60	57	54	51	48
Diseño detallado	28	27	26	25	24
Codificación y prueba de unidades	32	30	28	26	24
Integración y pruebas	22	25	28	31	34

Las siguientes tres tablas presentan los porcentajes de tiempo de desarrollo, para cada una de las fases, en los tres modos de desarrollo.

Distribución del tiempo (%). Modo orgánico

Fase	Tamaño (KLDC)				
	Pequeño (2)	Interm. (8)	Medio (32)	Grande (128)	Muy grande (512)
Planificación y especificaciones	10	11	12	13	–
Diseño del producto	19	19	19	19	–
Programación	63	59	55	51	–
Integración y pruebas	18	22	26	30	–

Distribución del tiempo (%). Modo semilibre

Fase	Tamaño (KLDC)				
	Pequeño (2)	Interm. (8)	Medio (32)	Grande (128)	Muy grande (512)
Planificación y especificaciones	16	18	20	22	24
Diseño del producto	24	25	26	27	28
Programación	56	52	48	44	40
Integración y pruebas	20	23	26	29	32

Distribución del tiempo (%). Modo restringido

Fase	Tamaño (KLDC)				
	Pequeño (2)	Interm. (8)	Medio (32)	Grande (128)	Muy grande (512)
Planificación y especificaciones	24	28	32	36	40
Diseño del producto	30	32	34	36	38
Programación	48	44	40	36	32
Integración y pruebas	22	24	26	28	30

Pero ¿cómo usar estas tablas? Retomando el ejemplo anterior, vamos a calcular el esfuerzo y el tiempo durante la fase de programación. Recordemos que en este proyecto se consideran 32 KLDC estimadas; por lo tanto, de acuerdo con la tabla para el esfuerzo en el modo orgánico, tenemos que el porcentaje para el esfuerzo estimado, en programación para 32 KLDC, es del 62%; entonces:

$$Eprog=(0.62)(E)$$

$$Eprog=(0.62)(91\ HM)=56\ HM$$

De la misma manera, el porcentaje de tiempo destinado a la fase de programación, para 32 KLDC, en el modo orgánico, es del 55%; por lo tanto:

$$Tprog=(0.55)(T)$$

$$Tprog=(0.55)(14\ meses)=7.7\ meses$$

El nivel medio de personal equivalente sería de:

$$P_E=(56\ HM)/(7/7\ meses)$$
$$=7.3\ hombres.$$

Un proceso similar se puede realizar con las otras tablas, para los otros modos de desarrollo, de acuerdo con el tamaño estimado del producto cuando éste se ajusta a los tamaños estándares.

La siguiente tabla muestra los valores nominales obtenidos de aplicar los porcentajes de las tablas de esfuerzo y tiempo de desarrollo, en el modo orgánico, a los proyectos de tamaño estándar. Además, se muestran los valores relativos a la productividad del proyecto y el personal medio equivalente necesario en cada fase.

Conceptos	Fases	Pequeño (2 KLDC)	Intermedio (8 KLDC)	Medio (32 KLDC)	Grande (128 KLDC)
Esfuerzo (HM)	Planificación y requisitos	0.3	1.3	5	24
	Diseño del producto	0.8	3.4	15	63
	Programación	3.4	13.8	56	231
	Diseño detallado	1.3	5.3	22	90
	Codificación y prueba de unidades	2.1	8.5	34	141
	Integración y pruebas	0.8	4.1	20	98
	Esfuerzo total	5.0	21.3	91	392
Tiempo (Meses)	(Planificación y requisitos	0.5	0.9	1.7	3.1
	Diseño del producto	0.9	1.5	2.7	4.6
	Programación	2.9	4.7	7.7	12.2
	Integración y pruebas	0.8	1.8	3.6	7.2
	Tiempo total estimado	4.6	8.0	14	24
Personal 8.0 Equiv. (P_E)	Planificación y requisitos	0.6	1.4	2.9	
	Diseño del producto	0.9	2.3	5.6	14
	Programación	1.2	2.9	7.3	19
	Integración y pruebas	1.0	2.3	5.6	14
Productividad (LDC/HM)		400	376	352	327

De igual manera, aplicando los porcentajes y ecuaciones adecuados para los modos semilibre y restringidos, obtenemos tablas similares a la anterior para proyectos de tamaño estándar.

A continuación se presenta una tabla donde se totalizan los valores de esfuerzo, tiempo, productividad y personal equivalente, para todos los modos de desarrollo, en proyectos de tamaño estándar. Por lo tanto, no se discrimina por fases.

Esfuerzo (HM)	Pequeño (2 KLDC)	Intermedio (8 KLDC)	Medio (32 KLDC)	Grande (128 KLDC)	Muy grande (512 KLDC)
Orgánico	5.0	21.3	91	392	
Semilibre	6.5	31	146	687	3.250
Restringido	8.3	44	230	1.216	6.420
Productividad (KLDC/HM)	**Pequeño (2 KLDC)**	**Intermedio (8 KLDC)**	**Medio (32 KLDC)**	**Grande (128 KLDC)**	**Muy grande (512 KLDC)**
Orgánico	400	376	352	327	
Semilibre	308	258	219	186	158
Restringido	241	182	139	105	80

Tiempo (meses)	Pequeño (2 KLDC)	Intermedio (8 KLDC)	Medio (32 KLDC)	Grande (128 KLDC)	Muy grande (512 KLDC)
Orgánico	4.6	8.0	14	24	
Semilibre	4.8	8.3	14	24	42
Restringido	4.9	8.4	14	24	41
Personal equivalente –P_E (Hombres)	**Pequeño (2 KLDC)**	**Intermedio (8 KLDC)**	**Medio (32 KLDC)**	**Grande (128 KLDC)**	**Muy grande (512 KLDC)**
Orgánico	1.1	2.7	6.5	16	
Semilibre	1.4	3.7	10	29	77
Restringido	1.7	5.2	16	51	157

2.2.2. *Distribución del esfuerzo por actividades*

Por último, una vez determinada la distribución del esfuerzo entre las fases principales del ciclo de vida de un producto software, es necesario conocer cómo se distribuye entre las diferentes actividades, de forma que sirva de referencia para:

- preparar planes de distribución de los recursos y
- adecuar la estructura de la organización al tamaño de los equipos dedicados a cada una de las actividades que, en proyectos de gran tamaño, será necesario modificar a medida que avanza el proyecto, en función de los cambios de actividades y según la fase en que se encuentre el proyecto.

Las siguientes tablas muestran la distribución por actividades, dentro de cada una de las fases principales.

Distribución del esfuerzo por actividades. Modo restringido

	FASE																			
	Planificación y requisitos					Diseño del producto					Programación					Integración y prueba					Desarrollo				
Tamaño	P	I	M	G	MG	P	I	M	G	MG	P	I	M	G	MG	P	I	M	G	MG	P	I	M	G	MG
(%) fase completa	8	8	8	8	8	18	18	18	18	18	60	57	54	51	48	22	25	28	31	34					
ACTIVIDAD																									
Análisis de requisitos	50	48	46	44	42	10	10	10	10	10	3	3	3	3	3	2	2	2	2	2	4	4	4	4	4
Diseño del producto	12	13	14	15	16	42	42	42	42	42	6	6	6	6	6	4	4	4	4	4	12	12	12	12	12
Programación	2	4	6	8	10	10	11	12	13	14	55	55	55	55	55	32	36	40	44	48	42	43	43	44	45
Plan de pruebas	2	3	4	5	6	4	5	6	7	8	4	5	6	7	8	3	3	4	4	5	4	4	5	6	7
Verificación y validación	6	7	8	9	10	6	7	8	9	10	8	9	10	11	12	30	28	25	23	20	12	13	14	14	14
Oficina de proyectos	16	14	12	10	8	15	13	11	9	7	9	8	7	6	5	10	9	8	7	6	10	9	8	7	6
GC/CC	5	4	4	4	3	4	3	3	3	2	8	7	7	7	6	10	9	9	9	8	8	7	7	7	6
Manuales	7	7	6	5	5	9	9	8	5	5	7	7	6	5	5	9	9	8	7	7	8	8	7	7	6

Distribución del esfuerzo por actividades. Modo orgánico

	FASE																			
	Planificación y requisitos				Diseño del producto				Programación				Integración y prueba				Desarrollo			
Tamaño	P	I	M	G	P	I	M	G	P	I	M	G	P	I	M	G	P	I	M	G
(%) Fase completa				6				16	69	65	62	25	16	19	22	25				
ACTIVIDAD																				
Análisis de requisitos	46				15				5				3				6			
Diseño del producto	20				40				10				6				14			
Programación	3				14				58				34				48	47	46	45
Plan de pruebas	3				5				4				2				4			
Verificación y validación	6				6				6				34				10	11	12	13
Oficina de proyectos	15				11				6				7				7			
GC/CC	2				2				6				7				7			
Manuales	5				7				5				7				7			

Distribución del esfuerzo por actividades. Modo semilibre

	FASE																			
	Planificación y requisitos					Diseño del producto					Programación									
Tamaño	P	I	M	G	MG	P	I	M	G	MG	P	I	M	G	MG					
(%) Fase completa	7	7	7	7	7	17	17	17	17	17	64	61	58	51	52					
ACTIVIDAD																				
Análisis de requisitos	48	47	46	45	44	12.5	12.5	12.5	12.5	12.5	4	4	4	4	4					
Diseño del producto	16	16.5	17	17.5	18	41	41	41	41	41	8	8	8	8	8					
Programación	2.5	3.5	4.5	5.5	6.5	12	12.5	13	13.5	14	56.5	56.5	56.5	56.5	56.5					
Plan de pruebas	2.5	3	3.5	4	4.5	4.5	5	5.5	6	6.5	4	4.5	5	5.5	6					
Verificación y validación	6	6.5	7	7.5	8	6	6.5	7	7.5	8	7	7.5	8	8.5	9					
Oficina de proyectos	15.5	14.5	13.5	12.5	11.5	13	12	11	10	9	7.5	7	6.5	6	5.5					
GC/CC	3.5	3	3	3	2.5	3	2.5	2.5	2.5	2	7	6.5	6.5	6.5	6					
Manuales	6	6	5.5	5	5	8	8	7.5	7	7	6	6	5.5	5	5					

Distribución del esfuerzo por actividades. Modo semilibre
(Continuación)

FASE										
	Integración y pruebas					Desarrollo				
Tamaño	P	I	M	G	GM	P	I	M	G	MG
(%) Fase completa	19	22	25	28	31					
ACTIVIDAD										
Análisis de requisitos	2.5	2.5	2.5	2.5	2.5	5	5	5	5	5
Diseño del producto	5	5	5	5	5	13	13	13	13	13
Programación	33	33	37	39	41	45	45	44.5	44.5	44.5
Plan de pruebas	2.5	2.5	3	3	3.5	4	4	4.5	5	5.5
Verificación y validación	32	31	29.5	28.5	27	11	12	13	13.5	14
Oficina de proyectos	8.5	8	7.5	7	6.5	8.5	8	7.5	7	6.5
GC/CC	8.5	8	8	8	7.5	6.5	6	6	6	5.5
Manuales	8	8	7.5	7	7	7	7	6.6	6	6

2.2.3. *Proyectos de tamaño no estándar (Interpolación)*

En el caso que el tamaño estimado del producto no se ajuste a los tamaños estándares, podemos obtener el perfil del proyecto por interpolación de los datos de los proyectos estándares de tamaño inferior y superior.

Por ejemplo, supongamos que deseamos obtener el esfuerzo de desarrollo en la fase de programación de un producto de software, cuyo tamaño fijamos en 12.800 líneas fuente en el modo orgánico.

Aplicando las ecuaciones de estimación del esfuerzo total y el tiempo de desarrollo, tenemos:

Esfuerzo total $E=2.4(12.8)^{1.05}=35$ HM

Tiempo de desarrollo $T=2.5(35)^{0.38}$
$=9.7$ meses

Para obtener el esfuerzo en la fase de programación, tenemos que referirnos a los proyectos de 8 KLDC y 32 KLDC. El porcentaje sobre el esfuerzo total de nuestro proyecto estará comprendido entre el 65% del proyecto de 8 KLDC y el 62% del proyecto de 32 KLDC.

El porcentaje buscado, empleando interpolación lineal, será:

$$y=yo+\left(\frac{y-xo}{x1-xo}\right)(y1-yo)$$

donde: **y** es el porcentaje que se desea calcular

x es el tamaño del proyecto en KLDC

yo y **y1** corresponden al 65% y 62%, respectivamente.

xo y **x1** corresponden a 8 KLDC y 32 KLDC, respectivamente.

$$y=65+\frac{12.8-8}{32-8}(62-65)=64.4$$

El porcentaje de esfuerzo, para el tamaño no estándar, es del 64.4% y, por lo tanto, el esfuerzo dedicado, en la fase de programación, será:

$$Eprog=(0.644)(E)$$
$$Eprog=(0.644)(35)=22.5 \text{ HM}$$

De la misma forma podemos obtener el porcentaje de tiempo de programación, interpolando los porcentajes de tiempo, del 59% para 8 KLDC y del 55% para 32 KLDC.

$$y=59+\frac{12.8-8}{32-8}(55-59)=58.2$$

Entonces, el tiempo en la fase de programación corresponde al 58.2% del tiempo total, es decir:

$Tprog=(0.582)(9.7 \text{ meses})=5.6$ meses

El valor medio de personal, equivalente en la fase de programación, será de:

$P_E=22.5\text{HM}/5.6 \text{ meses}=4.01$ hombres

2.3. **Modelo COCOMO intermedio**

En este modelo se calcula el esfuerzo de desarrollo de software en función del tamaño del programa y de un conjunto de atributos conductores del costo, que incluyen la evaluación subjetiva del producto, del hardware, del personal y de los atributos del proyecto.

El COCOMO intermedio es una versión ampliada del COCOMO básico, pues presenta un mayor nivel de detalle y de seguridad, pero conservando la misma sencillez.

Los atributos conductores de costos son 15 factores, compuestos por cuatro categorías y que no se tienen en cuenta en el modelo básico. Dichos atributos son:

- *Atributos del producto:*
  - Confiabilidad requerida del software.
  - Tamaño de la base de datos.
  - Complejidad del producto.
- *Atributos del hardware:*
  - Restricciones de tiempo de ejecución.
  - Restricciones de memoria.
  - Volatilidad de máquina virtual
  - Tiempo de respuesta del equipo
- *Atributos del personal:*
  - Capacidad de análisis
  - Experiencia en aplicaciones
  - Capacidad de los programadores
  - Experiencia en el sistema operativo utilizado
  - Experiencia con el lenguaje de programación
- *Atributos del proyecto:*
  - Aplicación de métodos de ingeniería del software
  - Utilización de herramientas de software
  - Restricción del tiempo de desarrollo.

Cada uno de estos atributos tiene asociado un factor multiplicativo, el cual estima el efecto del atributo en el esfuerzo del desarrollo del software; este factor se denomina **multplicador de esfuerzo.** Estos multiplicadores se aplican a un estimativo-base del esfuerzo, para obtener un estimativo refinado (ajustado) del esfuerzo del desarrollo. Este modelo inicia la estimación, con la generación de un estimativo nominal o básico del esfuerzo, usando funciones escalares similares a las del modelo básico. Este estimativo nominal es, entonces, ajustado, aplicando los multiplicadores de esfuerzo, derivados de la clasificación del proyecto respecto de los 15 atributos del costo.

La siguiente tabla presenta las ecuaciones del esfuerzo nominal, para los tres modos de desarrollo, utilizados en el COCOMO, intermedio.

Modo de desarrollo	Ecuación nominal de esfuerzo
Orgánico	$E_{nom}=3.2(KLDC)^{1.05}$
Semilibre	$E_{nom}=3.0(KLDC)^{1.12}$
Restringido	$E_{nom}=2.8(KSDI)^{1.2}$

El modelo intermedio presenta una tabla de multiplicadores de esfuerzo, donde cada atributo conductor de costo tiene asociado un conjunto de multiplicadores. Estos, a su vez, están relacionados con un conjunto de clasificaciones de proyectos.

Atributos directores del costo	Muy bajo	Bajo	Nominal	Alto	Muy alto	Extra alto
Restricciones fiabilidad del software	Efecto: Inconvenientes de escasa consideración	Bajo, pérdidas fácilmente recuperables	Moderado, pérdidas recuperables	Pérdidas financieras altas	Riesgo de vidas humanas	
Tamaño de la base de datos		DB Bytes/prog D5<10	10<=D/F<100	100<=D/P<1.000	D/P>=1.000	
Restricciones de tiempo de ejecución			<=50% de ejecución disponible	70%	85%	95%
Restricciones de memoria principal			<=50% de la memoria existente	70%	85%	95%
Volatilidad máquina virtual		Cambios significativos: cada 12 meses poco significativos: 1 mes	Significativos: 6 meses Poco signif.: 2 semanas	Significativos: 2 meses Poco signif.: 1 semana	Significativos: 2 semanas Poco signif.: 2 días	
Tiempo de respuesta		Interactivo	Tiempo medio de retorno<4Horas	4-12 horas	>12 horas	
Capacidad de los analistas	15%	35%	55%	75%	90%	
Experiencia en la aplicación	<=meses de exper.	1 año	3 años	6 años	12 años	
Capacidad de los programadores	15%	35%	55%	75%	90%	
Experiencia en el S.O. utilizado	<=1 mes	4 meses	1 año	3 años		
Empleo de técnicas utilizadas en programación	No se usan	Comienzan a emplearse	Alguna utilización	Uso general	Uso rutinario	
Utilización de herramientas software	Herramientas básicas de micros	Herramientas básicas de minis	Grandes computadores	Herramientas avanzadas de Programación y verificación	Incluyendo herramientas de diseño, gestión y documentación	
Restricciones de tiempo de desarrollo	75% del nominal	85%	100%	130%	160%	

Atributos	VALOR					
	Muy bajo	Bajo	Nominal	Alto	Muy alto	Extra alto
Atributos del producto						
Confiabilidad del software requerida	0.75	0.88	1.0	1.15	1.40	–
Tamaño de la base de datos	–	0.94	1.0	1.08	1.16	–
Complejidad del producto	0.70	0.85	1.0	1.15	1.30	1.65
Atributos del hardware						
Restricciones de tiempo de ejecución	–	–	1.0	1.11	1.30	1.66
Restricciones de memoria	–	–	1.0	1.06	1.21	1.56
Volatilidad de la máquina virtual	–	0.87	1.0	1.15	1.30	–
Tiempo de respuesta del equipo	–	0.87	1.0	1.07	1.15	–
Atributos del personal						
Capacidad de análisis	1.46	1.19	1.0	0.86	0.71	–
Experiencia en aplicaciones	1.29	1.13	1.0	0.91	0.82	–
Capacidad de programadores	1.42	1.17	1.0	0.86	0.70	–
Experiencia en el S.O. utilizado	1.21	1.10	1.0	0.90	–	–
Experiencia con el lenguaje	1.14	1.07	1.0	0.95	—	—
Atributos del proyecto						
Aplicación de la ingeniería del software	1.24	1.10	1.0	0.91	0.82	–
Utilización de herramientas del software	1.24	1.10	1.0	0.91	0.83	–
Restricción del tiempo de desarrollo	1.23	1.08	1.0	1.04	1.10	–

Como ayuda para seleccionar el factor aplicable en cada caso, la siguiente tabla establece los criterios que sirven para elegir el factor adecuado.

Por ejemplo, si consideramos el desarrollo de una aplicación del modo semilibre, con 32 KLDC, que presenta diferentes requerimientos de confiabilidad debido a la naturaleza de su ambiente de trabajo, el esfuerzo nominal se calculará así:

$E_{nom}=3.0(32)^{1.12}=146$ hombres–mes.

Indica un esfuerzo total de 146 hombres-mes que son requeridos para desarrollar un producto de 32 KLDC, en modo semilibre, independiente de cualquier consideración de los requerimientos de confiabilidad.

A partir de aquí, los ajustes en el estimativo final deben hacerse de acuerdo con el tipo de producto final. Por ejemplo, si el sistema por desarrollar es un modelo de predicción del clima, podría tener un relativo nivel bajo de calidad, puesto que su impacto operacional es generalmente a largo plazo y muchas fallas podrían ocasionar pérdidas para los usuarios, fácilmente recuperables. Estas características le dan un peso, en la tabla mencionada, de 0.88 (bajo) para un ajuste de:

$E=(E_{nom})$(Factor multiplicativo)
E=(146 HM)(0.88)-128 hombres-mes

Pero si el sistema por desarrollar es, por ejemplo, un sistema de control de reactor nuclear, podría tener un muy alto requerimiento de confiabilidad. Aquí el efecto de fallas del software podría provocar la pérdida de vidas humanas; por lo tanto, tiene un factor multiplicativo mayor que el anterior. La estimación del esfuerzo será, de acuerdo con el nivel de confiabilidad 1.40 (muy alto), la siguiente:

$E=(E_{nom})$(Factor multiplicativo)
E=(146 HM)(1.40)=204 hombres-mes

Por lo tanto, para ajustar la estimación del esfuerzo usamos la siguiente ecuación, empleando los factores multiplicativos:

$$E=(KLDC)^{S} \prod^{15}_{j=1} F_J$$

Donde S es el exponente correspondiente al modo de desarrollo y F el factor multiplicativo.

Veamos ahora un ejemplo sobre cómo aplicar los factores multiplicativos en el modelo de COCOMO intermedio. Supongamos que estamos negociando con la compañía Megabit Comunicaciones el precio de desarrollar un producto de software para el procesamiento de funciones de comunicación, en un microprocesador comercial, cuyo tamaño se estima en 10 KLDC y se desarrolla en modo restringido. De acuerdo con la ecuación nominal del esfuerzo para este modo, obtenemos el siguiente esfuerzo nominal:

$$2.8(10)^{1.20}=44 \text{ HM}$$

Deseamos ahora determinar el efecto de varias características del proyecto, en el esfuerzo y costo de desarrollo. Por ejemplo, el software para el procesamiento de comunicaciones generalmente tiene una valoración muy alta, en la escala de complejidad, pero se planea usar personal analista y programador con alta capacidad, lo cual balancea la tendencia a incrementar los costos, debido a la complejidad. El costo del personal será de $6.000 por mes.

En la siguiente tabla tenemos las características presentes para el desarrollo del producto:

Atributo	Situación	Clasificación	Multiplicador de esfuerzo
Confiabilidad del software requerida	Uso local del sistema No hay problemas serios de recuperación	Nominal	1.00
Tamaño de la base de datos	20.000 bytes	Baja	0.94
Complejidad del producto	Procesamiento de comunicaciones	Muy alta	1.30
Restricciones de tiempo de ejecución	Se usará el 70% del tiempo disponible	Alta	1.11
Restricciones de memoria	45K de 64K disponible	Alta	1.06
Volatilidad de la máquina virtual	Basado en microprocesador comercial	Nominal	1.00
Tiempo de respuesta del equipo	2 horas promedio	Nominal	1.00
Capacidad de análisis	Analistas con experiencia	Alto	0.86
Experiencia en aplicaciones	3 años	Nominal	1.00

Atributo	Situación	Clasificación	Multiplicador de esfuerzo
Capacidad de programadores	Programadores con experiencia	Alta	0.86
Experiencia en el S.O. utilizado	6 meses	Baja	1.10
Experiencia con el lenguaje	12 meses	Nominal	1.00
Aplicación de la ingeniería del software	Muchas técnicas por más de un año	Alta	0.91
Utilización de herramientas del sotfware	Herramientas	Baja	1.10
Restricción del tiempo de desarrollo	9 meses	Nominal	1.00
Factor de esfuerzo ajustado (producto de multiplicadores de esfuerzo)			1.17

Nótese que el factor de ajuste se incrementa en 1.30, debido a la complejidad muy alta, pero más adelante se reduce el valor a 0.86, debido a la alta capacidad de analistas y programadores. El factor de ajuste final, de 1.17, representa un incremento del 17% en el esfuerzo nominal; entonces, el costo y el esfuerzo estimados finalmente son:

Esfuerzo: (44 HM)(1.17)=51 HM

Costo: (51 HM)($6.000/HM)=$306.000

2.3.1. *Análisis de sensibilidad*

El modelo COCOMO intermedio permite realizar un análisis de sensibilidad con respecto a los atributos directores del costo, logrando así estimar el efecto sobre el costo de desarrollo, debido a los cambios en los niveles de clasificación de los atributos.

Por ejemplo, supongamos que tenemos la opción de realizar el proyecto del ejercicio anterior con personal de menor capacidad y menos costoso. Para este caso, el costo por persona-mes podría ser de $5.000 en lugar de los $6.000 anteriores, y la capacidad de análisis y programación la podríamos clasificar como Nominal. De acuerdo con la tabla anterior, este nivel de capacidad del personal resulta en un factor multiplicador de 1.00, en lugar del 0.86 obtenido anteriormente. Esto significa que el estimativo de COCOMO debe ser ajustado de acuerdo con las consideraciones anteriores:

Nuevo factor de ajuste de esfuerzo =1.58

Esfuerzo=(44 HM)(1.58)=70 HM

Costo: (70 HM)($5.000/HM)=$350.000

Según lo anterior podemos notar que resulta más ventajoso usar personal de mayor capacidad, según el análisis realizado con el factor de ajuste del esfuerzo, porque el estimativo de costo es menor.

Veamos otro ejemplo para el factor de ajuste de restricción del almacenamiento. Supongamos que por $10.000, Megabit podría comprar 96K palabras de memoria para el microprocesador por usar, en lugar de usar 64K. Esto podría cambiar las restricciones del almacenamiento principal del 70% al 47%. El resultado en el multiplicador del esfuerzo sería de 1.00, que corresponde a una clasificación Nominal, en lugar de 1.06 (alta), que era el anterior caso. Con lo anterior se tiene que:

Factor de ajuste del esfuerzo=1.10

Esfuerzo=(44 HM)(1.10)=48HM

Costo=(48 MM)($6.000/HM)=$288.000

Lo anterior nos indica que se logra un ahorro de $18.000, lo cual compensa la inversión de $10.000 en costos de hardware. Además, al negociar con la compañía Megabit, se puede proponer esta opción como forma para reducir el costo del software.

2.3.2. *Estimación de los efectos de software reutilizable*

Hasta ahora todas las estimaciones se habían basado en el supuesto que la totalidad del producto estaba siendo especificado, diseñado y desarrollado por completo, sin considerar que alguno de sus componentes hubieran sido desarrollados para otros proyectos previamente y que pudiesen utilizarse directamente o por adaptación en el nuevo producto.

Obviamente, las líneas-fuente de un programa adaptado no pueden tener la misma consideración que la del nuevo software, para efectos de la estimación. Por ejemplo, supongamos que utilizamos, como parte de nuestro nuevo producto, una rutina simple de entrada/salida, que ya forma parte de la librería de utilidades de otros proyectos, con lo cual se incrementa el número de líneas-fuente de nuestro nuevo producto, pero a la vez no incrementa la cantidad de esfuerzo requerido para el desarrollo del nuevo producto.

Por otra parte, en muchas ocasiones no se necesita adaptar completamente un paquete, sino que sólo se precisa realizar un esfuerzo adicional, para rediseñar el software adaptado, para ajustarlo a los objetivos del nuevo producto, el cual consiste en rehacer algunas porciones para acomodarlas a los cambios del entorno del nuevo producto.

Los efectos de la adaptación de productos existentes son tratados por el Modelo COCOMO, a través del cálculo del número equivalente de instrucciones fuente-IE, que se emplean en sustitución de las líneas de código-fuente LDC.

El número de líneas IE de un producto adaptado se calcula a partir de la estimación de las siguientes cantidades:

- Número de líneas-fuente adaptadas al nuevo producto, IA.
- Porcentaje de rediseño del software MD. Es el porcentaje de diseño del software que ha sido modificado para adaptarlo al nuevo entorno. Necesariamente ésta es una estimación subjetiva.
- Porcentaje de código modificado. Es aquél que es modificado con el fin de adaptarlo al nuevo entorno.
- Porcentaje del esfuerzo de integración del software adaptado MI. Es el porcentaje de esfuerzo requerido para integrar el software adaptado al nuevo producto, el cual se obtiene por comparación con el esfuerzo de integración necesario para un producto de tamaño similar.

El total de líneas equivalentes se calcula a través de un factor de adaptación definido como FA:

FA=0.40 MD+0.30MC+0.30MI

A partir del cual el número de líneas equivalentes vendría dado por:

IE=IA*FA Líneas equivalentes.

Este valor puede añadirse a las estimaciones de nuestro producto y tratarlas como hemos estado haciendo hasta ahora, de forma que el tamaño final del producto será:

LDC_{total}=LDC+IE.

Como ejemplo, supongamos que deseamos convertir un programa de análisis de circuitos, escrito en FORTRAN, de 50.000 líneas, desarrollado en modo orgánico desde un entorno hardware-software determinado en otro diferente.

Para esta situación podríamos considerar que:

MD=0 (es decir, no hay cambios en el diseño del programa).

MC=15 (posiblemente el 15% de las líneas de código deban cambiarse para adaptarlo a las características propias del nuevo sistema operativo, del lenguaje de control, etc.).

MI=5 (se precisa una pequeña cantidad de esfuerzo para integrar estos cambios).

Los resultados de la adaptación serían:

FA=0.40(0)+0.30(15)+0.30(5)=6

y por lo tanto, el número de líneas equivalentes sería:

IE=50.000 LDC*0.06=3.000 líneas equivalentes.

Utilizando las estimaciones del modelo básico, se obtiene un esfuerzo de adaptación de:

$E=2.4(3)^{1.05}=7.6$ HM

Los coeficientes de la ecuación del factor de adaptación se determinan a partir del valor medio de los porcentajes dedicados a las tareas de diseño, codificación e integración y pruebas dadas por el modelo:

Diseño:	40%
Codificación:	30%
Integración y prueba:	30%

Para estimar el tiempo de desarrollo se emplean las mismas ecuaciones utilizadas en el Modelo COCOMO básico.

En cuanto a la distribución del esfuerzo por fase, el modelo intermedio utiliza las mismas tablas empleadas en el modelo básico.

Los multiplicadores de esfuerzo, en este modelo, pueden aplicarse a la fase de mantenimiento, del mismo modo que se realizan en la fase de desarrollo.

2.4. **Modelo COCOMO detallado**

Este modelo incorpora todas las características del modelo intermedio y lleva a cabo una evaluación del impacto de los atributos conductores del costo, en cada fase del proceso de la ingeniería del software.

El modelo intermedio es altamente efectivo para muchos propósitos de estimación del software. Sin embargo, tiene dos limitantes principales que pueden ser significativas en estimaciones de costo más detalladas para proyectos extensos:

- La estimación distribuida de esfuerzo, por fases, puede ser inexacta.
- Puede ser muy engorroso para usarlo en productos con muchos componentes.

El modelo detallado provee dos principales facultades que consignan las limitantes del modelo intermedio.

- Multiplicadores de esfuerzo de fase sensitiva: en la práctica, factores tales como confiabilidad, experiencia y desarrollo interactivo afectan a algunas fases mucho más que otras. El modelo detallado ofrece un con-

junto de multiplicadores de esfuerzo, de fase sensitiva, para cada atributo del costo. Estos son usados para determinar la cantidad de esfuerzo requerido para completar cada fase.

Los multiplicadores se usan de acuerdo con la precisión que refleje el efecto de los conductores de costo, en la distribución de fase del esfuerzo. Por ejemplo, un nivel bajo en experiencia de aplicaciones puede significar esfuerzo adicional en las primeras fases. En las últimas fases el equipo podrá familiarizarse con la aplicación y, así, no se requerirá mucho esfuerzo adicional.

De otra parte, un tiempo de respuesta alto en el computador tendrá muy poco efecto en las fases de diseño y requerimientos, pero ciertamente consumirá más tiempo del equipo durante la codificación y las pruebas.

- Jerarquía de producto de tres niveles: en el modelo intermedio la clasificación separada de los conductores de costo puede ser suministrada por diferentes componentes del producto. Este proceso puede ser tedioso e innecesariamente repetitivo si un número de componentes se agrupa en un subsistema, con prácticamente todas las mismas clasificaciones. El modelo detallado excluye este problema, ofreciendo la jerarquía de producto de tres niveles, para lo cual:
- Algunos efectos, que tienden a variar con cada módulo de nivel interno, son tratados en el nivel de módulo.
- Algunos efectos, que varían menos frecuentemente, son tratados en el nivel de subsistema.
- Algunos efectos, tales como el efecto de tamaño total del producto, son tratados en el nivel del sistema.

La jerarquía módulo-subsistema-sistema se da por la descomposición jerárquica del producto, del cual se va a estimar el costo.

El nivel más bajo, el nivel de módulo, se describe por el número de líneas de código-fuente en el módulo, y por aquellos atributos de costo que tienden a variar al más mínimo nivel. Por ejemplo, la complejidad del módulo y su adaptación al software existente; la capacidad de los programadores y la experiencia en el manejo del lenguaje en el que se desarrolla el software.

El segundo nivel, nivel de subsistema, se caracteriza por el resto de los conductores de costo (tiempo, condiciones de almacenamiento, capacidad de analistas, herramientas, etc.), los cuales tienden a variar de un subsistema a otro, por lo cual tiende a ser lo mismo para todos los módulos del subsistema.

El nivel más alto, el nivel del sistema, se usa para aplicar las principales relaciones del proyecto, tales como las ecuaciones de esfuerzo y tiempo, y para aplicar, durante las interrupciones, en el esfuerzo y los tiempos del proyecto.

3. METODOS DE LOS PUNTOS DE FUNCION

La métrica de los puntos de función permite cuantificar una aplicación, basándose en dos áreas de evaluación: La primera consiste en el cálculo de puntos de función simples. La segunda área consiste en ajustar el modelo, en función de la complejidad y del ambiente de desarrollo de la aplicación.

La finalidad de esta metodología es, en primer lugar, estimar el tamaño de un producto de software en las etapas previas de su desarrollo y, por otra parte, estimar el esfuerzo de desarrollo, expresado en este caso en horas trabajadas, por el punto de función desarrollado. Un punto de función es una métrica que describe una unidad del producto de trabajo, tales como el esfuerzo de desarrollo y/o mantenimiento del software. Una componente o función es un proceso único o requerimiento de datos de la aplicación, por ejemplo, una pantalla donde se adicionan datos o un reporte impreso.

En este método se evalúa la funcionabilidad de una aplicación en términos de **qué** se le entrega al usuario en la aplicación y no cómo se le entrega. Unicamente los componentes visibles y los requerimientos del usuario son contabilizados. Estos componentes son tratados como Tipos de Función, los cuales están divididos en Datos y Transacciones:

Tipos de Función Datos: Archivos Lógicos Internos (ALI) y Archivos Externos de Interface (AEI).

Tipos de Función Transacciones: Funciones de Entrada (Entradas Externas), Funciones de Salida (Salidas Externas), Consultas Externas (CE).

Los objetivos principales de esta metodología son:

Medir lo que el usuario requiere y lo que se le entrega.

Medir la aplicación, independientemente de la tecnología usada para la implementación.

Proveer una métrica de tamaño que soporte análisis de calidad y productividad.

Proveer un vehículo para la estimación del software.

Este último aspecto es el que se trata en este documento.

3.1. Tipos de función

Los puntos de función sin ajuste tratan exclusivamente del conteo de las funciones del usuario y se realiza tras una previa clasificación de las funciones, basándose en aquellos componentes de una aplicación que son requeridos y son visibles al usuario.

3.1.1. ***Archivo Lógico Interno:*** Es un conjunto de datos del usuario que están lógicamente relacionados y que son mantenidos y utilizados, por la aplicación, dentro de sus propios límites. Mantener los datos se refiere a la capacidad de adicionar, modificar o borrar datos a través de procesos estandarizados de la aplicación. También entra en este grupo la información de control, es decir, los datos usados por la aplicación para asegurar que se cumplan los requerimientos de funcionamiento especificados por el usuario.

Para identificar estos archivos se identifican todos los datos o grupos de datos que:

Son almacenados dentro del alcance de la aplicación.

Son mantenidos a través de procesos estandarizados de la aplicación.

Son definidos como requerimientos de la aplicación por el usuario.

También se identifican estos archivos agrupando los datos lógicamente, desde el punto de vista del usuario:

Agrupar los datos hasta un nivel de detalle, en el cual el usuario pueda identificar los datos como satisfacción a sus requerimientos.

Tratar los datos lógicamente; no deben tomarse en cuenta los archivos físicos.

Los siguientes son los tipos de datos que pueden relacionarse con uno o más Archivos Lógicos Internos, dependiendo de la visión de usuario.

Datos de la aplicación (archivos maestros, tales como información de personal o información consolidada).

Datos de seguridad de la aplicación.

Datos de auditoría.

Mensajes de ayuda.

Mensajes de error.

Ejemplos de Archivos Lógicos Internos son:

Datos de respaldo (Backup), los cuales se cuentan únicamente si son específicamente requeridos por el usuario, ya sea por requerimientos legales o algo similar.

Archivos Lógicos Internos que sean utilizados y mantenidos por más de una aplicación, es decir, archivos compartidos.

Los siguientes archivos no se consideran Archivos Lógicos Internos:

Archivos temporales.

Archivos de trabajo.

Archivos de ordenamiento.

3.1.2. ***Archivos Externos de Interface.*** Son grupos de datos lógicamente relacionados o información de control utilizados por la aplicación, pero que no son mantenidos por ésta sino por otras aplicaciones. En este tipo se cuentan archivos transferidos o distribuidos entre aplicaciones.

Para identificar estos archivos es necesario:

- Identificar todos los datos que estén almacenados fuera del alcance de la aplicación.
- Que no sean mantenidos por la aplicación propia.
- Que estén dentro de los requerimientos del usuario.

También se identifican estos archivos agrupando los datos lógicamente, desde el punto de vista del usuario, para lo cual se debe:

- Agrupar los datos hasta un nivel de detalle, en el cual el usuario pueda identificar los datos como satisfacción a sus requerimientos.
- Tratar los datos lógicamente; no deben tomarse en cuenta los archivos físicos.

Los siguientes son tipos de datos que pueden relacionarse con uno o más Archivos Lógicos Internos, dependiendo de la visión del usuario:

- Datos de referencia (datos externos utilizados por la aplicación, pero que no hacen parte de sus Archivos Lógicos Internos).
- Mensajes de ayuda.
- Mensajes de error.

Los siguientes no se consideran como Archivos Externos de Interface:

- Datos que son mantenidos por la aplicación, pero a los que se tiene acceso y son utilizados por otra aplicación.
- Datos formateados y procesados, para ser utilizados por otra aplicación.

Los Archivos Externos de Interface no se atribuyen a la aplicación fuente, es decir, a la aplicación donde se originan estos archivos. Esto significa que se atribuyen estos archivos a las aplicaciones, básandose en la dirección del flujo y el uso de los datos, por la aplicación.

El tipo de archivo se determina básándose en cómo los utiliza la aplicación que recibe los datos. Si los datos son usados para actualizar los Archivos Lógicos Internos, se considera como una Entrada Externa o una Salida Externa, dependiendo del flujo. Si los datos no se usan para el mantenimiento de un Archivo Lógico Interno, se consideran como un Archivo Externo de Interface, de acuerdo con el flujo de datos.

El cálculo de los puntos de función debe ser actualizado si, después del cálculo, el acceso a un Archivo Lógico Interno se le permite a otra aplicación.

No siempre se puede determinar cómo otra aplicación está utilizando los datos y varios métodos no han involucrado esta situación, lo cual ha dado

como resultado cálculos inconsistentes. Para resolver este problema, únicamente la aplicación que recibe los datos puede tener Archivos Externos de Interface, es decir, que el cálculo se asume desde el punto de vista de la aplicación que se analiza. Como resultado, el cálculo de los puntos de función depende únicamente del sistema, como existe actualmente, y no de futuros eventos o de la utilización de los datos.

3.1.3. ***Funciones de Entrada (Entradas Externas - E.E).*** Se hace referencia a las funciones de entrada de datos. Se consideran entonces los datos únicos de entrada o de control que deben suministrarse a la aplicación con el fin de añadirlos a un Archivo Lógico Interno o de modificarlo. Se deben contar las introducidas directamente por el usuario y las obtenidas como resultado de una transacción, desde otra aplicación.

Una Función de Entrada (Entrada Externa) se considera única si tiene diferente formato o si, por su diseño interno, requiere un proceso lógico, diferente de otra función de entrada, con el mismo formato.

El formato puede ser un conjunto de datos únicos o un arreglo único de datos o un orden único de datos.

Para identificar las Entradas Externas se debe:

- Identificar todos los procesos que actualizan los Archivos Lógicos Internos.
- Por cada proceso identificado, considerar cada formato como un proceso separado, si los datos usados por el proceso pueden ser recibidos en más de un formato; asignar una Entrada Externa por cada una de las actividades de mantenimiento desempeñadas; esto es, adicionar, modificar y borrar.

Las siguientes son Entradas Externas, tomando en cuenta las anteriores consideraciones:

- Datos de transacciones: Datos externos usados para mantener los Archivos Lógicos Internos.
- Pantallas de entrada: Se cuenta una Entrada Externa para cada una de las funciones de mantenimiento. Si adiciona, modifica y borra, la pantalla debe contarse como tres Entradas Externas.
- Por cada proceso único que mantiene un Archivo Lógico Interno, contar una Entrada Externa por cada adición, modificación y borrado.

Dos o más archivos físicos pueden corresponder a una sola Entrada Externa, si el procesamiento lógico y el formato para cada uno de estos archivos es idéntico.

- Entradas Externas duplicadas: Se hace referencia a aquellos procesos que presentan la misma Entrada Externa, pero por diferentes medios. Por ejemplo, en un sistema bancario que acepta idénticas transacciones de depósito, a través de un proceso automático (cajero automático) y a través de procesos manuales. Por lo tanto, ambos procesos se cuentan cada uno como Entrada Externa.

Las siguientes no se consideran como Entradas Externas:

- Datos suministrados a la aplicación, por razones de tecnología empleada.
- Datos externos utilizados por la aplicación, pero no mantenidos en Archivos Lógicos Internos.
- Datos de entrada, usados para seleccionar la recuperación de datos.
- Pantallas de menú que únicamente permiten viajar por las pantallas de la aplicación y no contribuyen di-

rectamente en el mantenimiento de los Archivos Lógicos.

- Pantallas que facilitan la entrada a una aplicación; por ejemplo, pantallas de logon.
- Múltiples formas de invocar la misma entrada. Por ejemplo, digitar **A** o **Add** como un comando o como una tecla de función, para seleccionar la opción de adicionar; sólo se cuenta una Entrada Externa.
- Cualquier tipo de consulta no se cuenta, puesto que pertenece a un tipo de función diferente.

3.1.4. ***Funciones de Salida (Salidas Externas - SE).*** Se hace referencia a las funciones de salida de datos. Se consideran entonces datos únicos, de usuario o de control, los suministrados por la aplicación. Se incluyen dentro de este tipo, salidas tales como informes, mensajes al usuario, mensajes a otras aplicaciones, a través de archivos.

Una Función de Salida (Salida Externa) se considera única si tiene diferente formato o si, por su diseño interno, requiere un proceso lógico diferente de otra función de salida, con el mismo formato.

Para identificar las Entradas Externas es preciso:

- Identificar todos los procesos que suministran datos, fuera de los límites de la aplicación.
- Por cada proceso identificado, considerar cada formato como un proceso separado si los datos usados por el proceso pueden ser suministrados en más de un formato; asignar una Salida Externa por cada proceso identificado.

Las siguientes son Salidas Externas, tomando en cuenta las anteriores consideraciones:

- Datos transferidos hacia otras aplicaciones.
- Reportes. Cada reporte producido por la aplicación se cuenta como Salida Externa. Dos reportes idénticamente formateados se cuentan cada uno como Salida Externa si tienen procesamiento lógico diferente.
- Salidas en línea de datos que no se originan por una Entrada Externa.
- Reportes idénticos, pero producidos por medios diferentes; por ejemplo, un reporte originado por una impresora y por tarjetas perforadas.
- Una Salida Externa debe asignarse por cada Entrada Externa que origina mensajes de error o confirmación.

No se consideran Salidas Externas los siguientes tipos de datos:

- Múltiples formas de invocar la misma salida. Por ejemplo, para seleccionar un reporte, si se necesita digitar **R** o **Report** o una tecla de función, sólo se cuenta una Salida Externa.
- Mensajes de confirmación o error, originados por una consulta.

3.1.5. ***Consultas Externas - CE.*** Estas constituyen combinaciones únicas de Entrada/Salida, donde una entrada causa una salida inmediata. La consulta se considera única si tiene un formato diferente de otra función del mismo tipo (para la entrada y la salida), o si su diseño externo requiere un proceso lógico diferente.

Para identificar las Salidas Externas se debe tener en cuenta:

- Identificar todos los procesos en los que una entrada dispara una recuperación inmediata de datos.
- Por cada proceso identificado, verificar que cada combinación de Entrada/Salida es única y considerar cada combinación como un proceso diferente y acreditar una Consulta Externa por cada proceso.

Los siguientes son ejemplos de Consultas Externas:

- Pantallas de Modificación/Borrado, en las que se provoca la recuperación de datos, antes de ejecutar la función de Modificación/Borrado.
- Si las entradas y salidas de la consulta son idénticas en ambas funciones de Modificación y Borrado, se cuenta únicamente una Consulta Externa. Si se dispone de idénticas funciones de consulta en pantallas de Modificación/Borrado y en una pantalla aparte de consulta, se cuenta sólo una Consulta Externa.
- Pantallas de logon que proveen funciones de seguridad, se cuentan como Consultas Externas.

Tres categorías de ayudas son consideradas como Consultas Externas:

- Pantallas totales de ayuda, que son aquéllas que muestran el texto de ayuda relacionado con la pantalla que le llamó.
- Ayuda sensitiva del campo, que es aquella dependiente de la posición del cursor o de algún método de identificación, que muestra la documentación de ayuda para ese campo. Se cuenta una Consulta Externa por cada pantalla.
- Subsistema de ayuda, que es una facilidad de ayuda a la que se puede acceder y que puede ser recorrida, independiente de la aplicación asociada. Si el texto recuperado es idéntico al de una ayuda total, no se cuenta.

3.2. Cálculo de los Puntos de Función (sin ajuste)

El cálculo de los Puntos de Función consta de tres pasos:

- Obtener Puntos de Función sin ajuste.
- Obtener el Factor de Ajuste.
- Ajustar los Puntos de Función, usando el Factor de Ajuste para obtener los Puntos de Función reales.

Para cada tipo de función se determina la complejidad de forma diferente, así:

Para los Tipos de Función de Datos, la complejidad funcional se identifica de acuerdo con el número de Elementos de Tipo Registro (ETR) y con el número de Elementos de Tipo Datos (ETD).

Los ETR son formatos únicos de registros, dentro de un ALI o un AEI.

Los ETD son ocurrencias únicas de datos, también tratados como elementos de datos, variables o campos.

Por cada ALI y AEI identificado, se le asigna una complejidad funcional, basándose en los ETD y los ETR.

Cómo identificar ETR's: Son subgrupos de los ALI's, vistos desde la perspectiva lógica que el usuario tenga de los datos, es decir, de acuerdo con sus requerimientos. ALI's que no puedan ser categorizados, se considera que tienen un solo ETR.

Cómo identificar ETD's: Son campos reconocibles por el usuario y no recursivos, que residen en un ALI.

Cada campo en un ALI debe identificarse como un ETD, tomando en cuenta las siguientes consideraciones:

- Campos que deberían ser vistos desde un nivel reconocible por el usuario. Por ejemplo, un número de cuenta o una fecha que es físicamente almacenada en múltiples campos se cuenta como un ETD.
- Campos que aparecen más de una vez en un ALI, debido a la tecnología o a las técnicas de implementación, deben contarse como un ETD, sólo una vez. Por ejemplo, si un ALI se compone de más de una tabla en una base de datos relacional, la clave usada para relacionar las tablas se cuenta una sola vez.

- Campos repetitivos que son idénticos en formato y existen para permitir múltiples ocurrencias de un valor de un dato, se cuentan una sola vez. Por ejemplo, un ALI que contiene 12 campos de presupuestos mensuales y un campo con un presupuesto anual (la suma de los mensuales) deberían contarse como dos DTE, uno para el campo mensual y otro para el campo anual.

Por cada ALI y AEI se identifican sus ETR y ETD; luego se le asigna un grado de complejidad, de acuerdo con la siguiente tabla:

		(1 a 19) ETD	(20 a 50) ETD	(51 o más) ETD
(1)	ETR	Baja	Baja	Media
(2 a 5)	ETR	Baja	Media	Alta
(6 o más)	ETR	Media	Alta	Alta

El peso para cada grado de complejidad es el siguiente:

	ALI	AEI
Baja	7	5
Media	10	7
Alta	15	10

Para calcular el total de Puntos de Función, para los ALI y AEI, se usa una tabla como la siguiente:

Tipo de función	Complejidad funcional	Complejidad total
ALI	—— Baja x 7 =	——
	—— Media x 10 =	——
	—— Alta x 15 =	——

Por ejemplo, supongamos que se tienen tres ALI con complejidad baja, tres con complejidad media y tres con complejidad alta:

Tipo de función	Complejidad funcional	Complejidad total
ALI	3 Baja x 7 =	21
	3 Media x 10 =	30
	3 Alta x 15 =	45
Total de puntos de función		96

Sumando la columna de complejidad total, tenemos un total de puntos de función sin ajuste de 96.

De igual forma, tomemos el caso de tres AEI con complejidad baja, tres con complejidad media y tres con complejidad alta:

Tipo de función	Complejidad funcional	Complejidad total
AEI	3 Baja x 5 =	15
	3 Media x 7 =	21
	3 Alta x 10 =	30
Total de puntos de función		66

Sumando la columna de complejidad total, obtenemos un total de puntos de función sin ajuste de 66.

El siguiente paso consiste en calcular la complejidad funcional de las funciones de entrada EE (Entradas Externas), y SE (Salidas Externas) basándose en el número de Tipos de Archivo Referenciados TAR y el número de ETD.

Un Tipo de Archivo Referenciado se cuenta por cada ALI y por cada AEI referenciado, durante el procesamiento de una Entrada Externa, EE, o una SE, según el caso.

Para las EE, los ETD son tratados de igual forma que en los casos anteriores y, adicionalmente, un ETD se cuenta, para una EE, con las siguientes consideraciones:

- Líneas de comando o teclas de función que proveen la capacidad para especificar la acción que se tomará por la EE. Un ETD adicional por EE, no por tecla de comando.
- Campos que no son suministrados por el usuario, pero a través de EE son mantenidos en un ALI, deberían ser contados. Por ejemplo, un sistema de clave secuencial, mantenido en un ALI, pero no suministrado por el usuario, debería contarse como un ETD.

Para las SE se tratan inicialmente de igual forma los EDT, con las siguientes excepciones: No se incluyen literales como ETD.

No se cuentan las variables de páginas o los sistemas generados por rótulos por el sistema.

La complejidad funcional, para las funciones de entrada EE, se basa en la tabla siguiente:

		(1 a 4) ETD	(5 a 15) ETD	(16 o más) ETD
(0 a 1)	FTR	Baja	Baja	Media
(2)	FTR	Baja	Media	Alta
(3 o más	FTR	Media	Alta	Alta

La complejidad funcional para las funciones de salida, SE, se basa en la tabla siguiente:

		(1 a 5) ETD	(6 a 19) ETD	(20 a más) ETD
(0 a 1)	FTR	Baja	Baja	Media
(2 a 3)	FTR	Baja	Media	Alta
(4 o más)	FTR	Media	Alta	Alta

El peso para cada grado de complejidad es el siguiente:

	EE	SE
Baja	3	4
Media	4	5
Alta	6	7

Para las funciones de consulta, Consultas Externas, se siguen los siguientes pasos para determinar el valor de los puntos de función sin ajuste:

Calcular la complejidad funcional para la parte de entrada de la Consulta Externa.

Calcular la complejidad funcional para la parte de salida de la Consulta Externa.

Seleccionar el valor más alto de las dos complejidades funcionales. Usando una tabla adecuada para los Puntos de Función sin Ajuste, se transcribe la clasificación de la complejidad a Puntos de Función sin Ajuste.

Se cuenta un Tipo de Archivo Referenciado por cada ALI y AEI, referenciados durante el proceso de la consulta.

Un ETD se cuenta por aquellos campos suministrados que especifican la Consulta por ejecutar o que especifican los criterios de selección de los datos.

Un ETD se cuenta por cada campo no recursivo que aparece en la parte de salida de la consulta.

La complejidad funcional para las Consultas Externas, CE, se basa en las tablas siguientes:

Complejidad de entrada				
		(1 a 4) ETD	(5 a 15) ETD	(16 o más) ETD
(0 a 1)	FTR	Baja	Baja	Media
(2)	FTR	Baja	Media	Alta
(3 o más)	FTR	Media	Alta	Alta

Complejidad de salida				
		(1 a 5) ETD	(6 a 19) ETD	(20 o más) ETD
(0 a 1)	FTR	Baja	Baja	Media
(2 a 3)	FTR	Baja	Media	Alta
(4 o más)	FTR	Media	Alta	Alta

El peso para cada grado de complejidad es el siguiente:

	CE
Baja	3
Baja	4
Alta	6

Una vez obtenidos los puntos de función sin ajuste para cada tipo de función, podremos obtener el total de puntos de función sin ajuste, mediante la siguiente tabla:

Tipo de función	Complejidad funcional	Complejidad total	Total de puntos de función
ALI	3 Baja x 7 =	21	
	3 Media x 10 =	30	
	3 Alta x 15 =	45	
			96
AEI	3 Baja x 5 =	15	
	3 Media x 7 =	21	
	3 Alta x 10 =	30	
			66

Tipo de función	Complejidad funcional	Complejidad total	Total de puntos de función
EE	3 Baja x 3 =	9	
	3 Media x 4 =	12	
	3 Alta x 6 =	18	
			39
SE	3 Baja x 4 =	12	
	3 Media x 5 =	15	
	3 Alta x 7 =	21	
		48	
CE	3 Baja x 3 =	9	
	3 Media x 4 =	12	
	3 Alta x 6 =	18	
		39	
Total de puntos de función sin ajuste			288

3.3. Características generales del sistema

La funcionalidad del modelo puede variar dependiendo del entorno de desarrollo. Por esta razón, el modelo introduce la valoración de 14 características que influyen sobre la complejidad del proceso.

Estas características se evalúan de conformidad con una escala de grado de influencia que toma valores enteros entre 0 y 5, para ajustar la complejidad del proceso.

Las características generales del sistema son:

Transmisión de datos.
Proceso distribuido.
Rendimiento, respuesta.
Configuración.
Indice de transacciones.
Entrada de los datos en línea.
Eficiencia del usuario.
Actualización en línea.
Complejidad del proceso.
Reutilización.
Facilidad de instalación.
Sencillez de operación.
Adaptabilidad.
Flexibilidad de cambio.

Los grados de influencia se clasifican así:

0: No incluye.
1: Influencia insignificante.
2: Influencia moderada.
3: Influencia media.
4: Influencia significativa.
5: Influencia fuerte.

3.3.1. *Transmisión de datos*

Los datos e información de control, usados en la aplicación, son enviados o recibidos por medio de facilidades de comunicación.

Puntaje:

0: La aplicación ocurre únicamente en el procesamiento por lotes o en un computador aislado.

1: La aplicación se hace por lotes, pero se tiene una entrada remota de datos o una salida remota hacia la impresora.

2: La aplicación se hace por lotes, pero se tiene una entrada remota de datos y una salida remota hacia la impresora remota.

3: Recolección de datos "on line" o por teleprocesamiento frontal a un proceso por lotes o un sistema de consulta.

4: Más de un proceso frontal, pero la aplicación soporta únicamente un protocolo de teleprocesamiento de comunicaciones.

5: Más de un proceso frontal, pero la aplicación soporta más de un protocolo de teleprocesamiento de comunicaciones.

3.3.2. ***Proceso distribuido***

Son características de la aplicación.

Puntaje:

0: La aplicación no se preocupa por la transferencia de los datos o el procesamiento entre los componentes del sistema.

1: La aplicación prepara datos para procesamiento del usuario final sobre otros componentes del sistema, tales como micros.

2: Los datos son preparados para ser transferidos y procesados en otros componentes del sistema (no son para el usuario final).

3: El procesamiento distribuido y la transferencia de datos están en línea y en una sola dirección.

4: El procesamiento distribuido está en línea y entre direcciones.

5: Las funciones de procesamiento son dinámicamente ejecutadas en más de un componente del sistema asociado.

3.3.3. ***Rendimiento, respuesta***

Los objetivos del desempeño de la aplicación son aprobados por el usuario, como respuesta o por la influencia del diseño, el desarrollo, la instalación o el soporte de la aplicación.

0: No hay requerimientos especiales de desempeño por parte del usuario.

1: Los requerimientos del desempeño y del diseño fueron establecidos y revisados sin ninguna acción requerida.

2: Tiempos de respuesta "on line" críticas durante las horas pico. Ningún diseño especial se ha requerido para el uso de la CPU.

3: Tiempos de respuesta críticos durante todas las horas de trabajo. Ningún diseño especial se ha requerido para el uso de la CPU.

4: Los requerimientos establecidos por el usuario, para el desempeño, son estrictos; lo suficiente para implementar análisis de desempeño en la fase de diseño.

5: Adicionalmente, las herramientas del análisis del desempeño han sido usadas en el diseño, desarrollo y/o implementación, para conocer los requerimientos del usuario.

3.3.4. ***Configuración***

Un uso pesado de la configuración operacional que requiere consideraciones especiales de diseño, es una característica de la aplicación (por ejemplo, el usuario quiere correr la aplicación sobre un equipo que será altamente usado).

Puntaje:

0: No están explícitas o implícitas las restricciones de la operación.

1: Existen restricciones en la operación, pero son menos restrictivas que en una aplicación típica. No se necesita esfuerzo especial para responder ante las restricciones.

2: Algunas consideraciones de seguridad o tiempo.

3: Los requerimientos específicos del procesador son necesarios para una parte de la aplicación.

4: Las limitaciones establecidas para la operación requieren restricciones especiales sobre la aplicación, en el procesador central o en un procesador dedicado.

5: Adicionalmente, hay restricciones especiales para la aplicación en los componentes distribuidos del sistema.

3.3.5. ***Indice de transacciones***

El índice o promedio de transacciones es alto y afecta el diseño, el desarrollo, la instalación y el soporte de la aplicación.

Puntaje:

0: No se adelantan los períodos críticos de la transacción.

1: Se anticipan los períodos críticos de transacción mensual.

2: Se anticipan los períodos críticos de transacción semanal.

3: Se anticipan los períodos críticos de transación diaria.

4: Altos promedios de transacción son planteados por el usuario en los requerimientos o acuerdos de la aplicación, y son lo suficientemente altos como para requerir tareas de análisis de desempeño en la fase de diseño.

5: Al igual que en el punto anterior, pero adicionalmente se requiere el uso de herramientas de análisis de desempeño en las fases de diseño, desarrollo y/o instalación.

3.3.6. ***Entrada de datos en línea***

La entrada de datos en línea (on-line) y las funciones de control son provistas en la aplicación.

Puntaje:

0: Todas las transacciones son procesadas en lotes.

1: Del 1% al 7% de las transacciones son interactivas.

2: Del 8% al 15% de las transacciones son interactivas.

3: Del 16% al 23% de las transacciones son interactivas.

4: Del 24% al 30% de las transacciones son interactivas.

5: Más del 30% de las transacciones son interactivas.

3.3.7 ***Eficiencia del usuario***

Las funciones en línea provistas enfatizan un diseño para la eficiencia del usuario final. Esto incluye:

- Menús.
- Documentación y ayudas en líneas.
- Movimiento del cursor automatizado.
- Desplazamiento de imágenes ("scrolling").
- Impresión remota.
- Teclas de función preasignadas.
- Selección de datos en pantalla por medio del cursor.
- Uso constante de video inverso, resaltamiento, delineado en color y otros indicadores.
- Copia permanente de la documentación del usuario sobre transacciones en línea.
- Interfases con el "mouse".
- Ventanas "Pop-Pup".
- Fácil navegación entre las pantallas.
- Soportar dos o más lenguajes.

Puntaje:

0: No se presenta nada de lo anterior.

1: De una a tres de las anteriores.

2: De cuatro a cinco de las anteriores.

3: Seis o más de las anteriores, pero no hay requerimientos específicos de los usuarios con respecto a la eficiencia.

4: Seis o más de las anteriores, pero se establecen requerimientos sobre la eficiencia de los usuarios, hasta un nivel en el que se requieren tareas de diseño para los factores humanos por incluirse.

5: Seis o más de los anteriores y se establecen requerimientos para la eficiencia del usuario, a un nivel en el que se requieren herramientas y procesos especiales para verificar que se cumplan los objetivos planteados.

3.3.8. ***Actualización en línea***

La aplicación provee actualización en línea para los archivos lógicos internos.

Puntaje:

0: Ninguna.

1: Actualización en línea de uno a tres archivos de control. La cantidad de actualización es baja y de fácil recuperación.

2: Actualización en línea de cuatro o más archivos de control. La cantidad de actualización es baja y de fácil recuperación.

3: Actualización en línea de los principales archivos lógicos internos.

4: Adicionalmente, la protección contra pérdida de datos es esencial y ha sido especialmente diseñada y programada en el sistema.

5: Adicionalmente, altas cantidades involucran consideraciones de costos en el proceso de recuperación.

Procedimientos altamente automatizados de recuperación, con un mínimo de intervención de operadores.

3.3.9. ***Complejidad del proceso***

La complejidad se categoriza así:

- El control sensible (por ejemplo, un procedimiento especial de auditoría) y/o un procesamiento específico de seguridad de una aplicación.
- El procesamiento lógico extenso.
- El procesamiento matemático extenso.
- Mucho procesamiento de excepción, que resulta en transacciones incompletas, las cuales deben ser procesadas de nuevo.
- Un procesamiento complejo para manipular múltiples posibilidades de entrada-salida; por ejemplo, multimedia, dispositivos independientes.

Puntaje:

0: Ninguno de los anteriores.

1: Alguno de los anteriores.

2: Dos cualquiera de los anteriores.

3: Tres cualquiera de los anteriores.

4: Cuatro cualquiera de los anteriores.

5: Cualquiera de los anteriores.

3.3.10. ***Reutilización***

Se refiere al grado de volver a utilizar la aplicación en otros proyectos.

Puntaje:

0: Sin código reutilizable.

1: El código reutilizable se usa dentro de la aplicación.

2: Menos del 10% de los módulos producidos consideran más de una necesidad del usuario.

3: El 10% o más de los módulos producidos consideran más de una de las necesidades de los usuarios.

4: La aplicación fue específicamente compactada y/o documentada, para fácil reutilización, y la aplicación está diseñada para los usuarios, a nivel del código-fuente.

5: La aplicación fue específicamente compactada y/o documentada, para fácil reutilización, y la aplicación está diseñada para usarse por medio de los parámetros de mantenimiento del usuario.

3.3.11. ***Facilidad de instalación***

Puntaje:

0: No se tuvieron consideraciones especiales, por parte del usuario, para la instalación.

1: No se tienen consideraciones especiales, por parte del usuario, pero sí se requiere configuración especial para la instalación.

2: Los requerimientos de instalación y conversión son requeridos por el usuario. El impacto de la conversión en el proyecto no es importante.

3: Los requerimientos de instalación y conversión son requeridos por el usuario. El impacto de la conversión, en el proyecto, se considera importante.

4: Adicionalmente al puntaje 2, las herramientas automatizadas de conversión e instalación fueron provistas y probadas.

5: Adicionalmente al puntaje 3, las herramientas automatizadas de conversión e instalación fueron provistas y probadas.

3.3.12. ***Sencillez de operación***

En esta característica se tiene en cuenta la efectividad del arranque, el respaldo y el procedimiento de recuperación, durante la fase de prueba. La aplicación minimiza la necesidad de actividades manuales, tales como montajes de cintas, manipulación del papel.

Puntaje:

0: No hay consideraciones especiales diferentes de los de procesos de "backup" normal.

- 1 - 4: Se seleccionan los siguientes ítems, de acuerdo con la aplicación. Cada ítem tiene un valor de un punto, si no se especifica lo contrario.
- Procesos efectivos de arranque, respaldo y recuperación, pero se requiere la intervención del operador.
- Igual que el ítem anterior, pero no se requiere la intervención del operador (2 ítems).
- La aplicación minimiza la necesidad del montaje de la cinta.
- La aplicación minimiza la necesidad de la manipulación del papel.

5: La aplicación no requiere la intervención directa del operador, salvo en los procesos de arranque y apagado.

3.3.13. ***Adaptabilidad***

Esta característica se refiere a la capacidad que tiene la aplicación de ser instalada en múltiples sitios, para múltiples organizaciones.

Puntaje:

0: No se consideran requerimientos necesarios para más de un sitio de instalación.

1: Se consideran múltiples sitios necesarios. La aplicación es diseñada para operar únicamente en idénticas condiciones de software y hardware.

2: La aplicación está diseñada para operar únicamente en condiciones similares de hardware y/o software.

3: La aplicación está diseñada para operar en condiciones diferentes de hardware y/o software.

4: La documentación y el plan de soporte son provistos y probados para que la aplicación soporte múltiples sitios. Tal como en los puntajes 1 y 2.

5: La documentación y el plan de soporte son provistos y probados para que la aplicación soporte múltiples sitios. Tal como en el puntaje 3.

3.3.14. ***Facilidad de cambio***

La aplicación ha sido específicamente diseñada, de forma tal que pueda soportar cambios fácilmente.

Puntaje:

0: No hay requerimientos especiales del usuario, con respecto al diseño de la aplicación, para minimizar o facilitar el cambio.

1-5: Seleccionar cualquiera de los siguientes ítems, de acuerdo con la aplicación.

- Flexibilidad y consultas sencillas (un ítem).
- Flexibilidad en las consultas, de forma que puedan manipular consultas simples o medianamente complejas (dos ítems).
- Flexibilidad en las consultas, de forma que se manejen consultas fáciles y complejas (tres ítems).
- El control de los datos se guarda en tablas que son mantenidas por el usuario con procesos interactivos, pero los cambios se efectúan únicamente pasado un tiempo.
- Igual que en el anterior, pero los cambios se efectúan inmediatamente (dos ítems).

3.4. **Cálculo del valor del factor de ajuste**

El factor de ajuste se calcula sobre la base de las 14 características generales del sistema. Este valor más adelante servirá para ajustar los puntos de función sin ajuste. El proceso para obtener este valor de ajuste es el siguiente:

Evaluar las 14 características generales del sistema, de acuerdo con el grado de influencia de cada uno.

Sumar los 14 grados de influencia para producir un grado de influencia total.

Introducir el grado de influencia, en la siguiente ecuación, para producir el factor de ajuste.

(TGI*0.01)+0.65=VFA

Donde TGI es el total de los 14 grados de influencia y VFA es el valor del factor de ajuste. Tomemos el siguiente ejemplo, en el cual se han evaluado las 14 características generales de un sistema, obteniendo, para cada una, su grado de influencia:

Características generales del sistema	Grados de influencia
Transmisión de datos	3
Procesamiento distribuido	4
Desempeño	4
Configuración	2
Indice de transacciones	3
Entrada de datos en línea	3
Eficiencia de usuario	4
Actualización en línea	3
Complejidad del proceso	3
Reutilización	2
Facilidad de instalación	3
Sencillez de operación	3
Adaptabilidad	2
Flexibilidad	3
Total de los grados de influencia (TGI)	42
Valor del Factor de Ajuste (VAF) (42 * 0.01)+0.65 =	1.07

Ahora estamos en capacidad de calcular los puntos de función, ajustados de acuerdo con la siguiente ecuación:

Puntos de Función sin Ajuste * VFA = Puntos de Función Ajustados

Supongamos ahora que el ejemplo de los puntos de función sin ajuste anterior, cuyo resultado fue de 288, corresponde a la misma aplicación a la que se le calculó el VAF anterior de 1.07. Aplicando la ecuación anterior, obtenemos los puntos de función, así:

288*1.07=308.16; es decir, obtenemos un valor aproximado de 308 puntos de función. Finalmente, la estimación del número de intrucciones-fuente se obtiene a partir de la relación siguiente:

LDC=66*PF.

Para el ejemplo anterior, entonces, tenemos 20.328 LDC.

Existen relaciones empíricas entre las líneas de código y los puntos de función para los lenguajes conocidos, así:

Assembler	320	LDC/PF
C	150	LDC/PF
Cobol	107	LDC/PF
Fortran	106	LDC/PF
ADA	71	LDC/PF
Lenguajes de bases de datos	40	LDC/PF
Generadores de lenguajes	32	LDC/PF
Orientado a objetos	29	LDC/PF
Lenguaje de consulta	25	LDC/PF
Pascal	85	LDC/PF
Hoja de cálculo	6	LDC/PF

De la tabla anterior se debe tener claro que, por ejemplo, para lenguaje C, 150 líneas de código equivalen a un punto de función.

CONCLUSIONES

1. Son muchos los modelos existentes para la estimación y que ofrecen diversas formas de obtener estimativos de costos y esfuerzo.
2. Cualquiera que sea el método escogido para la estimación del costo-esfuerzo, es importante que la organización que desee realizar estimaciones recolecte y mantenga un conjunto de datos históricos de estimaciones y métricas de proyectos anteriores, de forma que las estimaciones futuras sean mucho más confiables.
3. Todo proyecto de desarrollo de software tiene asociado un conjunto de atributos, ya sean tecnológicos, económicos y de mano de obra, los cuales afectan directamente el proceso de estimación.
4. Uno de los principales factores que influyen en el costo y esfuerzo de un proyecto de software es la complejidad misma del proyecto y el grado de confiabilidad requerida.
5. Es posible realizar estimaciones, en los proyectos de software, desde sus primeras etapas de vida. Para ello, quienes llevan a cabo la estimación,

deben conocer a fondo el sistema por desarrollar, la tecnología por emplear y la capacidad humana con que contará, puesto que con estos factores se logra un óptimo desarrollo del producto.

6. El estudio de los modelos de estimación requiere de buen tiempo de dedicación, ya que involucra muchos factores que son difíciles de evaluar, tales como tecnología, personal, herramientas de desarrollo, etc.

BIBLIOGRAFIA

- PRESSMAN, Rogger S. *Ingeniería del software: Un enfoque práctico*. Cap. 2 y 3. Segunda edición.
- BOEHM, Barry W. *Software Engineering Economics*. Prentice–Hall 1981.
- CARD, David N. with GLASS, R. *Measuring Software Design Quality*. Mc-Graw Hill.
- FRANCISCO, Gisbert Cantó. *Evaluación de Costes de Desarrollo - Modelos Algorítmicos*. Facultad de Informática. Universidad Politécnica de Madrid.
- IEEE. Transactions on software engineering. *Software development cost estimation using functions points*. Vol. 20 No. 24 April 1994, pág. 275.
- IEEE. Transactions on software engineering *Function point analysis: Dificulties and improvements*. Vol. 14 No. 1. January 1988.

PROPUESTA DE INTERCONEXION A INTERNET

JUAN CARLOS MACHADO
ANDRES FELIPE MILLA
JOHN FREDDY VALENCIA

Alumnos del curso de Investigación de VIII Semestre de Ingeniería de Sistemas del ICESI

INTRODUCCION

De las tres partes en las que ha sido dividido el proyecto de investigación (propuesta de Interconexión a **Internet**) presentamos la primera, que constituye la presentación teórica, la que permite ubicar un poco el contexto en el que se va a trabajar.

Con esto pretendemos empezar a tratar de una forma cronológica todo lo necesario para llegar a entender, de manera clara, lo que es el trabajo en **Internet**, lo cual permitirá, posteriormente, realizar un estudio de carácter técnico.

1. DEFINICION

La pregunta que más frecuentemente se hace es, "¿qué es **Internet?**".

La razón por la cual se hace tan a menudo esta pregunta es que no ha habido un acuerdo sobre una respuesta que ingeniosamente recapitule a **Internet.**

Internet puede ser considerada, a través de la relación con sus protocolos más comunes, como una colección física de enrutadores (routers) y circuitos, como un conjunto de recursos compartidos o simplemente como una actitud de interconexión e intercomunicación.

Algunas de las definiciones más comunes dadas en el pasado son:

- Una red de redes, basada en los protocolos del TCP/IP.
- Una comunidad de gente que usa y desarrolla las mencionadas redes.
- Una colección de recursos que pueden ser alcanzados, a través de estas redes.

Hoy en día, **Internet** es un recurso global que interconecta a millones de usuarios, los cuales empezaron como un experimento (veinte años atrás) del Departamento de Defensa de los Estados Unidos de América.

Mientras que las redes que "inventaron" a **Internet** están basadas en un conjunto estándar de protocolos (un acuerdo mutuo de métodos de comunicación entre los interesados), **Internet** cuenta con pasarelas (gateways), hacia redes y servicios que están basados en otros protocolos.