UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

ESTUDIO DE COCOMO II PARA PROYECTOS BASADOS EN BASE DE DATOS

T E S I S

QUE PARA OBTENER EL TÍTULO DE INGENIERO EN COMPUTACIÓN
P R E S E N T A :

MONTERRUBIO RETANA RUBER UNAF

DIRECTOR DE TESIS:
M en C. REYNALDO ALANIS CANTÚ

MÉXICO, D. F. ENERO 2007

Agradecimientos

Quizá las palabras habladas se hayan quedado atrás, pero lo que todos ustedes saben, es que las tintas en una hoja son lo único que deja uno como prueba de nuestra existencia en este mundo, por tal motivo aquí plasmaré un poco de mi, y a lo mejor dentro de más años me anime a vencer de nuevo el temor que tengo, al universo en blanco de las letras. Las primeras palabras escritas se las dedicare a mis padres, quienes con su bigardía y coraje nunca dudaron de mi, a mi madre que siempre ha estado a un lado mío y que es el reflejo fiel de la nueva mujer mexicana: decidida, libre, luchadora y sobre todo el mejor ejemplo de coraje ante la vida que he recibido en estos años. Mi padre, aquel ser que fue mi verdadero amigo durante este trayecto que con todo mi ser, quisiera ser una décima parte de lo que él ha logrado; me acercó al bello universo de las matemáticas y de la física, las cuales son el lenguaje íntimo de la naturaleza.

Mis hermanos, quienes han crecido junto a mi y siguen sus propios caminos y espero que este pequeño escalón de la vida que he decido tomar, les sirva de inspiración para ir mas allá y lleguen a lugares que jamás me he imaginado.

Todos aquellos que se cruzaron en mi camino y que tomaron el riesgo de crecer conmigo, los que reprobaron conmigo, rieron, lloraron, triunfaron y fracasaron solo les podría decir, el acostumbrado: gracias por crecer conmigo, y sigan creciendo a pesar de las caídas y resbalones.

A todos los maestros, quienes me han enseñado y me han cultivado a lo largo de mi vida de estudiante, continúen con esa entrega y pasión de enseñar y aprender, por que esa es la clave para formar a hombres y mujeres libres de pensamiento y espíritu, preparados para darnos el cambio que necesita nuestra sociedad mexicana.

Las últimas letras no son las menos importantes, son las dedicadas a todos aquellos que no están aquí, quienes me vieron crecer y se perdieron mis nuevas aventuras, solo les diré: desde aquí los haré recordar y algún día estaré sentado en un bar con ustedes y les platicare todas mis andadas por este camino.

Les entrego un estudio que me ha consumido cierto tiempo de mi vida, pero me ha dado un tesoro que es invaluable y es el de aprender por tal motivo continuaré con la aventura de vivir y cuando esté postrado en una silla viejo y con canas en mi pelaje, recordare estas letras y sonreiré diciendo: aun sigo aprendiendo, gracias a la vida, por haberme dado tanto.

Ruber Unaf Monterrubio Retana

ÍNDICE

Primera Parte.

Segunda Parte.

Capítulo 3

Capítulo 4

Capítulo 5

Capítulo 1

¿Podría construir una casa sin saber cuánto estaría dispuesto a gastar?
Roger S. Pressman

1. Introducción

La gestión de un proyecto de software comienza con un conjunto de actividades que generalmente se denominan planificación de proyecto. Para la estimación del esfuerzo y tiempo de desarrollo de algún producto de software, existen muchos modelos y herramientas que sirven de apoyo. Antes de que el proyecto comience, el gestor y el equipo de software deben elaborar una estimación del trabajo a realizar, de los recursos necesarios y del tiempo que transcurrirá desde el comienzo hasta el final de su realización.

Al principio de la década de los 70s, el costo del software constituía un pequeño porcentaje del costo total de los sistemas basados en una computadora. Un error considerable en las estimaciones del tiempo y esfuerzo de desarrollo tenía relativamente poco impacto. Hoy en día, el software es el elemento más caro de la mayoría de los sistemas informáticos. *Para sistemas complejos, personalizados, un gran error en las estimaciones puede ser lo que marque la diferencia entre beneficios o pérdidas. (Pressman, 2000).*

Desde siempre se ha intentado conocer por adelantado cuantos recursos y tiempo tomará desarrollar, una aplicación específica. Esto es importante, ya que no sólo el presupuesto está en juego, sino también la reputación de la firma o bien, del líder del proyecto que se compromete a desarrollar el proyecto.

Sabemos que el esfuerzo y el tiempo de desarrollo representa únicamente una fracción del costo total asociado con un producto durante todo su ciclo de vida hasta que es, aceptado por el cliente, debido a que existen otros tipos de costos asociados al proyecto tales como: comercialización, contratación, adquisición de equipo, etc.

No obstante, podemos considerar que antes de adquirir el compromiso de desarrollo de determinada aplicación lo que necesitamos saber de manera crítica es el esfuerzo que será necesario invertir, antes de establecer un compromiso de desarrollo.

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

La precisión de una estimación del proyecto es tan buena como la estimación del tamaño del trabajo que va a llevarse a cabo, el *tamaño* representa el *primer reto* (ver ***Capítulo 4***) importante del planificador de proyectos. Este término se refiere a una producción cuantificable del proyecto de software; si se toma un enfoque directo, el tamaño se puede medir en *líneas de código* o de manera indirecta: *puntos de función*.

Es por ello que entre las metodologías básicas para la estimación de costos se encuentran el conteo *de líneas de código (LDC)* y los *puntos de función (PF)*. Dichas técnicas se discutirán más adelante a detalle.

Basándose en algunas de estas técnicas, o en la combinación de dos o más de ellas, se construyeron algunos modelos que pretenden hacer más exacta la estimación del costo del proyecto, tal es el caso de COCOMO, el cual es un ejemplo de un modelo de estimación que usa tanto las *métricas cuantitativas* (conteo de líneas de código), como *métricas cualitativas o indirectas* (puntos de función).

Este modelo ha evolucionado hasta considerar casi todos los factores que podría decirse que pueden incrementar o disminuir el costo del desarrollo de un proyecto, debido al incremento de varios factores (la complejidad del sistema, el tamaño del equipo de desarrollo, etc.) que intervienen en el momento de desarrollar software esto dio como resultado la versión titulada COCOMO II (Bohem, 2000).

Es bien sabido que existen en el mercado diversas herramientas, e incluso grupos o empresas, que ofrecen la estimación de costo como un servicio y que emplean el COCOMO II como una herramienta básica. El propósito fundamental del por qué es considerado como un modelo de amplia difusión (de dominio público), de fácil comprensión, y de continua calibración, fue elegido para el desarrollo de este trabajo de tesis.

La evolución del modelo COCOMO hasta la consideración de casi todos los factores que influyen para la realización de la estimación del costo de un proyecto, no hace hincapié en un factor que es importante e imposible de ignorar y es la inclusión de una base de datos en un proyecto de software.

De ninguna manera se afirma que COCOMO no considera, dentro de la estimación del costo de un proyecto, a la base de datos, si no por el contrario se considera la cantidad de instrucciones necesarias para la conectividad e interacción con ella (tamaño de la base de datos), como una parte más del proyecto general (para descripción del modelo COCOMO II, ***Capítulo 3***)

Hace falta considerar la dificultad asociada al manejo de grandes cantidades de información, un apartado donde se pueda cuantificar la necesidad de agregar funcionalidad específica para el manejo y presentación de los datos almacenados en la base de datos, esto es ¿acaso COCOMO considera en la estimación del esfuerzo la inclusión de motores de persistencia[1] para lidiar con estos factores?.

Además de otros puntos que resultan al analizar el estudio del proceso de desarrollo de una base de datos, como son los tiempos de desarrollo de acuerdo con la experiencia del equipo a cargo de la base de datos, aspectos relacionados con la normalización, cantidad de entidades a desarrollar, número de atributos asociados a éstos, tipos de relaciones entre entidades, entre otros.

1.1 Definición del problema

El problema a resolver está basado en los siguientes aspectos:

- ✓ Falta de estimaciones más apegados a la realidad para el desarrollo de un sistema basado en una base de datos.
- ✓ Necesidad de disminuir el esfuerzo asociado al desarrollo de software, que cuente, para su funcionamiento, una base de datos.
- ✓ Establecer un precedente para la realización de un modelo de estimación que incluya el esfuerzo de desarrollo de una base de datos.
- ✓ Proponer una metodología para el desarrollo de un proyecto que ayude a la reducción del esfuerzo y tiempo de desarrollo.

El aumento en el esfuerzo y retrasos en el tiempo de calendario asociados al desarrollo de un proyecto de software tienen varias consecuencias tales como: administrativas, técnicas o económicas; aunado a esos detalles, la implementación de enormes bases de datos, incluidas en algunos proyectos, sólo complica el panorama agregando nuevas consideraciones al problema de estimación del esfuerzo y del tiempo de desarrollo.

[1] Motores de persistencia :
Para una mayor referencia consultar la siguiente página: http://www.programacion.com/bbdd/articulo/joa_persistencia/

En este punto incluiré un comentario de un prefacio de un libro sobre gestión de proyectos de software de Meiler Page-Jones :

"*He visto docenas de empresas, buenas y malas, y he observado a muchos administradores de procesos de datos, también buenos y malos. Frecuentemente, he visto con horror cómo estos administradores luchaban inútilmente contra proyectos de pesadilla sufriendo por fechas límite imposibles de cumplir, o que entregaban sistemas que decepcionaban a sus usuarios y que devoraban ingentes cantidades de horas de mantenimiento*". *(ob. cit., Pressmam)*.

Es por eso que tomando en cuenta, que a través de la estimación del costo de un proyecto de software podemos adquirir una mejor comprensión de un proyecto en particular, por tal motivo, se tendrá la capacidad de asignar eficazmente los recursos tanto materiales y humanos necesarios para el desarrollo satisfactorio del proyecto, y como resultado obtener la reducción de incertidumbre, que siempre acompaña a una estimación de un proyecto, y el estrés asociado a éste.

Al tomar todas estas consideraciones, resulta fácil darse cuenta en qué medida es fundamental una estimación eficiente que asegure el éxito del desarrollo del proyecto en cuestión.

Si la complicación del proyecto aumenta, es más difícil determinar los recursos y el esfuerzo necesarios para satisfacer las necesidades que el proyecto requiere. Además si el proyecto, no tiene precedentes en la experiencia del equipo de desarrolladores, la situación se complica problema muy común entre las empresas de desarrollo de software, cuando llega a las manos del equipo de desarrolladores la tarea de desarrollar un proyecto totalmente nuevo en su haber, del cual no se sabe nada acerca de tiempos efectivos de desarrollo, ni en cuánto deberá venderse (síntoma muy conocido), se desconoce la cantidad de personal que deberá asignarse y para aumentar la carga de obstáculos por librar, dentro de los planteamientos del proyecto, se necesita implementar una base de datos que alimente de información a este nuevo sistema.

La estimación no sólo se trata de asignar recursos de acuerdo con los objetivos planteados, es más que eso. Se trata de ahorrar esfuerzo, tiempo del calendario de desarrollo, recursos y sobre todo, reducir el estrés producido por la incertidumbre asociada a todo gran proyecto.

El factor humano es el más importante al momento de estimar, ya que si no se tienen buenas estimaciones, se puede caer en el desgaste del ánimo, del equipo de desarrollo, de continuar en la elaboración el proyecto, al estar acatando las tareas, propias de la metodología de desarrollo, que se exige a cada miembro; por ende se ve una repercusión en la efectividad del trabajo.

Por otro lado nos, centramos en la base de datos la cual es uno de los dos componentes más importantes de un sistema; el otro es el programa; que realiza propiamente la funcionalidad deseada. Para que estos dos trabajen adecuadamente es necesario que se comuniquen y sean compatibles.

La evolución de estos dos ha tomado caminos distintos y la tarea de hacer que se entiendan, se logra a través de los motores de persistencia. *(Palasí, 2003*).

El por qué del estudio del trabajo de tesis es determinado por la obtención de una mejor estimación de un proyecto que incluya una base de datos y por otro lado mejorar el proceso de estimación del modelo COCOMO II (considerado como un buen modelo de estimación) tomando en cuenta las dificultades asociadas a la inclusión de una base de datos.

1.2 Objetivos

Objetivo General

Proponer el estudio de un procedimiento que estime el esfuerzo adicional provocado por el desarrollo de una base de datos y mejorar así los resultados de cualquier proyecto de software que incluya una base de datos, utilizado el modelo de estimación de costos COCOMO II de Barry Bohem.

Objetivos específicos

- ✓ Proveer al modelo COCOMO II de un agregado conceptual que ponga en consideración la complejidad asociada al desarrollo de una base de datos como parte de un proyecto regular de software.
- ✓ Identificar los parámetros presentes dentro del desarrollo de una base de datos, siguiendo un proceso de desarrollo que conlleve buenas prácticas de desarrollo de software, para calibrar el modelo con el propósito de obtener una estimación más exacta de un proyecto que incluye una base de datos.
- ✓ Proponer las métricas pertinentes y necesarias asociadas al agregado del modelo.
- ✓ Revisión y estudio del Modelo COCOMO II para encontrar parámetros relacionados con el desarrollo de base de datos.

1.3 Alcances y limitaciones

Alcances

Con la finalidad de presentar un marco conceptual para el modelo COCOMO II, se propone en esta tesis, el inicio de una línea de investigación que llegue al establecimiento de un modelo de estimación que considere características propias del desarrollo de software basado en una base de datos y no hacer que se agregue nueva funcionalidad al paquete de estimación COCOMOII.1999 (usado para el cálculo de estimaciones en el ***Capítulo 4***).

Limitaciones

- ✓ No se pretende demostrar que el modelo COCOMO II es el mejor modelo para realizar estimaciones, la propuesta de manejar este modelo surge por su fácil disponibilidad, constante calibración y ajuste a las características de los diferentes proyectos que se desarrollen en una organización.

- ✓ Esta tesis se apoyará en referencias a proyectos que están en desarrollo o en las etapas finales de desarrollo y además que cumplan con la inclusión o uso de una base de datos.

- ✓ La propuesta de la obtención de un apartado teórico que estime el esfuerzo de desarrollo de una base de datos, como parte integral del proyecto, necesita más investigación de campo, para calcular estimaciones más apegadas a la realidad del análisis, diseño e implementación de una base de datos, con base en constantes calibraciones del apartado, tal y como sucede con el modelo COCOMO II original.

1.4 Precedentes

Lo que precede para realizar la planificación de un proyecto software, es necesario poseer una estimación certera del esfuerzo necesario para el desarrollo lo más temprano posible, idealmente, con sólo la etapa de especificación de requisitos cubierta (este punto se aborda de manera extensa en el ***Capítulo 3***).

Una de las fases en la gestión del proyecto es la estimación del esfuerzo y plazo de cada una de las actividades de las que constará, por lo tanto, es necesario disponer de buena información sobre la duración y el esfuerzo en personas-mes necesarias para realizar cada tarea. (Boletín 2004, Villanueva).

Los dos principales determinantes para estimar el esfuerzo son: *el tamaño de lo que se requiere* y *la productividad de quién lo va a hacer*, para lo cual al momento de utilizar un modelo de estimación (en nuestro caso COCOMO II) se necesita hacer énfasis en la utilización de *métricas de software* ¿Cómo medir el software que requiero ¿ (el modelo de estimación establece una serie de pasos para realizar el cálculo de estimaciones, descritos en el ***Capítulo 4***).

El determinante del tamaño de lo que se requiere no es privativo del desarrollo del software, si no que es utilizado en cualquier inversión, a las empresas lo que les interesa es la capacidad de hacer algo con lo que adquieren. Por ejemplo, si se compra un camión, de alguna forma se adquirió la capacidad de transportar mercancías y con un tamaño de metros cúbicos por viaje. En la primera instancia lo relevante es qué tanto requiero transportar por viaje.

1.4.1 Métricas del software

El tamaño del software podría medirse en términos de los bytes que ocupa en el disco, el número de programas, el número de líneas de código, la funcionalidad que proporciona o simplemente en el número de pantallas o reportes que tiene.

Podríamos escoger alguna de estas maneras de medir el tamaño que tenga más correlación con el esfuerzo, pero se necesitan hacer otras consideraciones, tales como:

- ✓ Una aplicación de software es un conjunto de líneas de código que se ejecutan en una computadora. Sin embargo mucho del costo de producir ese software, no está directamente relacionado con la codificación, que es entre el 20 y 25% del costo total.[2]

- ✓ La amplia gama de lenguajes y herramientas para producir software, lo que ha provocado que pueda generarse la misma funcionalidad con lenguajes de programación distintos.

Explicaremos dos formas de medir el tamaño del software, ya que son éstas las que utiliza COCOMO para estimar. Las dos tienen dos enfoques diferentes de medición, cuantitativos (conteo por *Líneas de Código*) y cualitativos (*Puntos de Función*), más apegado a la funcionalidad y tratando de cumplir con las características[3] que debe tener una métrica del software.

---

[2] Elementos como la administración del proyecto, el nivel de detalle de la documentación técnica o la documentación de pruebas, y las pruebas por sí mismas también deben considerarse.

[3] Las características que deben tener las métricas del software son las siguientes:
- ✓ INDEPENDIENTE DE LA TECNOLOGÍA
- ✓ SIMPLE
- ✓ ENFOQUE EN LA FUNCIONALIDAD PROPORCIONADA
- ✓ BASADA EN LOS REQUERIMIENTOS DEL USUARIO
- ✓ CONSISTENCIA

Para mayor información acerca de estas características referirse al Boletín de Política Informática Núm. 6 2003 *Puntos por Función Una métrica estándar para establecer el tamaño del software*. Sergio Eduardo Durán Rubio

- **Líneas de código (LDC)**

El conteo de las líneas de código es una forma de saber el tamaño del software de manera directa, ya que se tiene que hacer un conteo del código del que está compuesto el sistema.

Este tipo de métrica está dividido en dos áreas diferentes: SLOC (Source Lines Of Code) y SDI (Source Delivered Instruccions). Las diferencias entre estas dos es que SLOC, toma en cuenta todo el trabajo que debe hacer el desarrollador, tales como cabeceras y comentarios incluidos. En cambio, SDI, sólo toma en cuenta el número de líneas de código ejecutable (Cohen, 2004).

El modelo de COCOMO de Barry Bohem usa las líneas de código *(SLOC)*. Este modelo como otros más que utilizan las SLOC o SDI, únicamente no hacen uso de estas métricas, si no que también trabajan con otros factores tales como limitaciones del hardware, ambiente de desarrollo, atributos del producto y personal (estos parámetros se denominan factores de escala, definidos en el ***Capítulo 2***).

- **Puntos de función (PF)**

Los *Puntos de Función* miden la aplicación desde una perspectiva del usuario, dejando de lado los detalles de codificación.

La estimación de costos basada en la funcionalidad esperada del sistema fue propuesta por Albrecht en 1979, y desde entonces ha sido muy estudiada.

Es una técnica totalmente independiente de todas las consideraciones de lenguaje y ha sido aplicada en más de 250 lenguajes diferentes.

PF evalúa confiabilidad:

- ✓ el valor comercial de un sistema para el usuario.
- ✓ tamaño del proyecto, costo y tiempo de desarrollo.
- ✓ calidad y productividad del programador.
- ✓ esfuerzo de adaptación, modificación y mantenimiento.
- ✓ posibilidad de desarrollo propio.
- ✓ beneficios de implementación en 4GL[4].

Un *Punto de Función* se define como una función comercial de usuario final. De esta manera, un programa que tenga "x" PF's entrega "x" funciones al usuario final. El mejor modo de trabajo es la interacción analista-usuario (para mayor detalle acerca de los PF consultar el ***Apéndice A***).

[4] Fourth Generation Language (Lenguajes de cuarta generación).

El proceso requiere dos etapas fundamentales:

1. *Se identifican las funciones disponibles para el usuario y se organizan en cinco grupos de preferencia en este orden:*

    - ✓ *Salidas externas.*
    - ✓ *Consultas.*
    - ✓ *Entradas externas.*
    - ✓ *Ficheros (Archivos internos lógicos).*
    - ✓ *Interfaces externas.*

Después se clasifica y pondera cada función por su nivel de complejidad simple, media y compleja.

2. Se ajusta este total de acuerdo con unas características del entorno, (Dreger, 1989).

Como ya se mencionó, esta forma de estimación necesita la identificación de todas las ocurrencias de los cinco tipos de funciones. La suma de todas esas ocurrencias es llamada *Conteo Bruto de Funciones* (Raw Function Counts). Este valor debe ser modificado por un grado de peso por ciertos factores técnicos de complejidad (TFC).
Los *Puntos de Función* son equivalentes a:

$$PF = FC \times TCF \qquad \textbf{...(1.1)}$$

Donde:

PF: Puntos de Función.
FC: Conteo bruto de funciones.
TCF: Factores técnicos de complejidad.

Esta técnica ha sido evaluada por varios autores, y se han hecho algunos intentos por refinar el modelo. Estas estimaciones han probado ser más exitosas que el modelo original. (ob cit. Kemerer).

En resumen los modelos de *Puntos de Función* parecen ser más exactos para predecir el esfuerzo necesario para un proyecto específico que los modelos basados en las LDC.

La tendencia en estimación de costos ha sido olvidarse de las LDC (ya sean SDL y las SDI) debido al uso de prototipados para evaluar usuabilidad y el ambiente deseado para el sistema, las herramientas CASE[5] y los generadores de código que facilitan el trabajo de los programadores, el re-uso de software y la adquisición de software prefabricado que ajusta o complementa a los proyectos de desarrollo.

Por tal motivo se ha dado paso al trabajo basado en *Puntos de Función* (PF). Esta tendencia se debe a que los PF son más independientes (son menos dependientes del lenguaje y del ambiente de programación) y además tratan de ajustarse a las características de las métricas de software que las LDC.

Si se encuentra el equipo de desarrollo en el inicio del proyecto, difícilmente se puede realizar una estimación certera de la *cantidad de líneas de código (LDC)* que tendrá la aplicación, ya que en este nivel no tiene por qué estar decidida la herramienta de desarrollo. (Esta métrica es usada para realizar estimaciones tempranas en el ***Capítulo 4***).

Sólo se podría entrar a realizar una estimación certera en los comienzos de la etapa de construcción, con un diseño acabado. Por otro lado, la determinación de los *PF* podría realizarse a partir de la especificación de requisitos, pero los factores correctores según la complejidad de la aplicación pueden estimarse con certeza una vez que se ha entrado de lleno a la etapa de diseño. Si bien es posible realizar la estimación de *PF* en fases anteriores que la estimación de *LDC,* se desearía poder realizar una estimación certera en base únicamente a la especificación de requisitos.

De una "buena" especificación de requisitos se pueden obtener las características de la aplicación a desarrollar, antes de que comience el desarrollo del software. Si a partir de estas características se puede obtener una estimación del tamaño del software, se tendría una estimación temprana del tamaño del mismo.

Para representar una especificación de requisitos nos podemos valer del uso de los Casos de Uso los cuales es una manera sencilla de describir la funcionalidad que deberá tener el nuevo sistema a desarrollar (documentados ampliamente en el ***Capítulo 3***), lo que permite apreciar el esfuerzo que será necesario invertir.

De este modo, al contar con una estimación temprana del tamaño de lo que se desea desarrollar, se puede realizar una estimación del esfuerzo en etapas tempranas del desarrollo. Esto es debido a que el *tamaño del software* es la *variable manejadora de costo principal del desarrollo*.

[5] Computer Aided Software Engineering (CASE)

Una estimación temprana sería útil para generar la planificación del proyecto, la cual podría corregirse con el apoyo de las técnicas basadas en los *Puntos de Función* o *líneas de código* en etapas más avanzadas del desarrollo.

1.4.1.2. Modelos de estimación

Una vez explicadas las métricas de software, las cuales son la primera parte esencial de la estimación de un proyecto y por lo tanto es utilizado por los modelos de estimación, siendo así se describirán dichos modelos y técnicas de estimación de costo y sus características deseables, las cuales serán las directrices de la línea de investigación que propone este trabajo de tesis.

- **Modelos Empíricos**

Un modelo de estimación para el software de computadora utiliza fórmulas derivadas empíricamente para predecir el esfuerzo como una función de *LDC (líneas de código)* o *PF (puntos de función)*.

Los datos que soportan la mayoría de los modelos de estimación obtienen una muestra limitada de proyectos. Por esta razón, el modelo de estimación no es adecuado para todas las clases de software y no en todos los entornos de desarrollo. Por lo tanto los resultados obtenidos de dichos modelos se deben utilizar con prudencia. Por tal motivo los autores del modelo COCOMO II dan la oportunidad de calibrar el modelo de acuerdo con datos históricos de estimaciones y a las características de la organización que utiliza el modelo de estimación, esto es una gran ventaja, ya que este modelo se puede adaptar a las características particulares del tipo de organización (este punto se aborda en el ***Capítulo 3***).

1.4.1.3. Características que debe poseer un modelo de estimación del esfuerzo

Se establecen las siguientes características como deseables en todo modelo de estimación que pretenda apoyar la etapa de planificación de proyectos de desarrollo de software.

- ✓ Poder adaptarse a la productividad de la organización.
- ✓ Considerar el costo de comunicación entre personas.
- ✓ Incorporar guías útiles para estimar aquellos parámetros que son subjetivos o no se deducen en forma explícita a partir del modelo.
- ✓ Usable.
- ✓ Constar de etapas simples de entender y definidas en forma precisa.
- ✓ Objetivo.
- ✓ Costo efectivo.
- ✓ Proveer medios para adaptarse a cambios en el ambiente de desarrollo.
- ✓ Permitir una estimación temprana.

1.4.1.4. Técnicas y Modelos de Estimación de Costo

En las últimas décadas algunos modelos de estimación de software han sido desarrollados a partir de las intensivas investigaciones de 1965 donde se realizó un estudio de 169 proyectos de software y 104 de sus atributos *(Nelson,96)*.

A finales de los años 70 se desarrollaron técnicas de estimación de costos basados en un modelo, tales como:

- ✓ CHECKPOINT *(Jone,s 1997)*.
- ✓ SLIM *(Putman, 1992)*.
- ✓ COCOMO *(ob. cit.)*.
- ✓ SEER *(Jensen, 1983)*.
- ✓ PRICE-S *(Park, 1994)*.

Tuvieron su éxito ya que contribuyeron a construir los primeros cimientos de las posteriores investigaciones en estimación de costos. Pero a medida que pasa el tiempo y los proyectos de software van madurando y creciendo en tamaño y complejidad el proceso de estimación de costos encuentra mayores dificultades en proporcionar predicciones más aproximadas a lo que se necesita.

Muchos de los modelos desarrollados son registrados bajo la propiedad de alguna compañía y no pueden ser comparados con otros en términos de su estructura del modelo. La forma funcional de estos modelos tiene que ser determinada por teorías o experimentación, (SGAD-IMTA, 2003) (ver Figura 1.1).

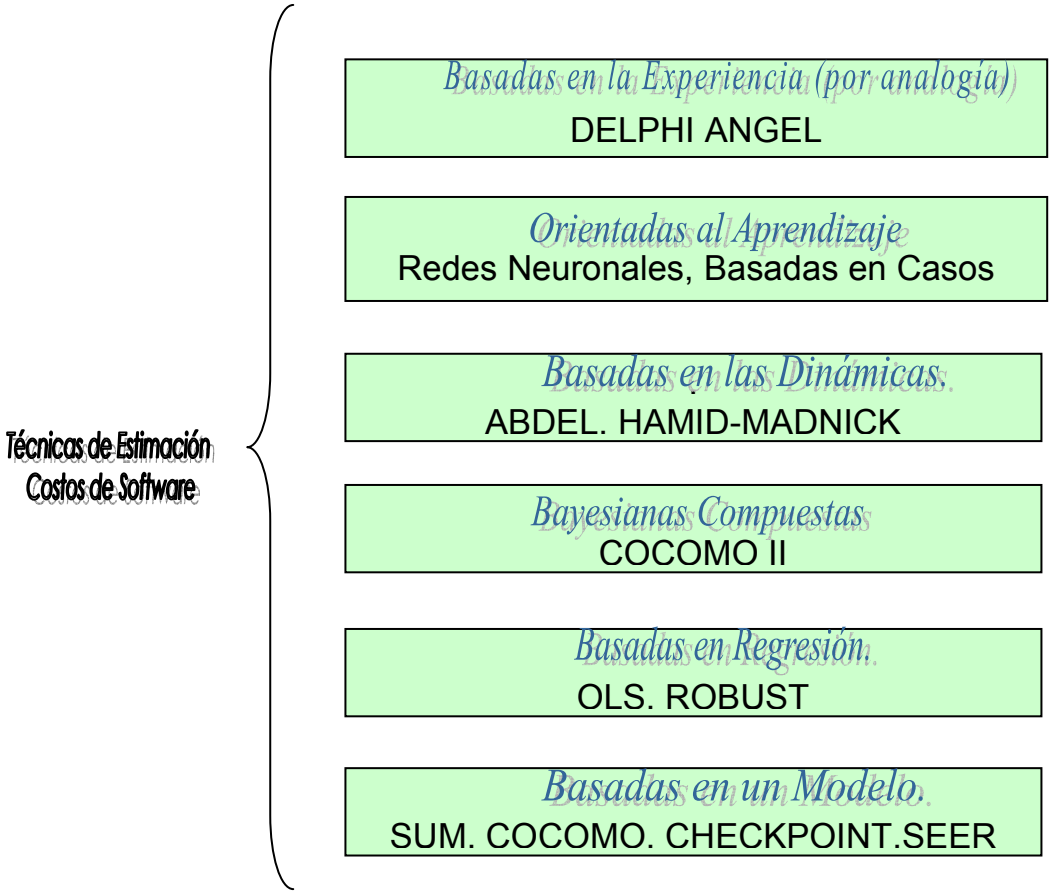


Figura 1.1 Técnicas de Estimación del Software

Ejemplos de Técnicas de Estimación basadas por la:

- ✓ Experiencia (por analogía): DELPHI, ANGEL
- ✓ Orientación al Aprendizaje: Redes Neuronales, Basadas en Casos
- ✓ Basadas en las Dinámicas: ABDEL, HAMD,MADNICK
- ✓ Basadas en Regresión: OLS ,ROBUST
- ✓ Bayesianas Compuestas: COCOMO II

Las técnicas de estimación basadas en un modelo son buenas para presupuestar, analizar las posibilidades de desarrollo, planear y controlar, y para el análisis de inversión. La mayoría de los modelos de estimación de software son apegados en un modelo matemático, pero debido a que son calibrados de acuerdo con la experiencia adquirida, su principal obstáculo se encuentra en aquellas situaciones que no tiene precedente; es decir, que al no contar con alguna base histórica de proyectos es muy difícil, llevar a cabo la aplicación de algún modelo de estimación de costo.

Para poder estimar el costo de un proyecto de software es necesario hacer mediciones acerca del tamaño del software o complejidad del proyecto por tal motivo se emplea las métricas del software, antes explicadas.

1.5 Estructura de la Tesis

El presente trabajo de tesis estudia la forma de obtener la estimación del esfuerzo de desarrollo de un proyecto de software tomando en cuenta el características de diseño, construcción e implementación de una base de datos en el proyecto, a partir del modelo COCOMO II DE Barry Bohem ayudándonos de la metodología denominada *Proceso Unificado Rotacional* (*Rational Unified Process o en sus siglas en inglés RUP*) ésta es una metodología estándar para desarrollar software más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos, el cual nos ayudara a reconocer los puntos claves del desarrollo de una base de datos y así, una vez identificados tales atributos esenciales, demostrar que aspectos tienen repercusiones en el esfuerzo en el desarrollo del proyecto de software e integrarlos como parámetros de esfuerzo dentro del modelo de estimación de COCOMO II.

Así proponer un nuevo apartado del modelo más apegado a los aspectos que conllevan la elaboración de una base de datos en un proyecto.

Para entender el procedimiento de solución que se propone en esta tesis, en el ***Capítulo 1 Introducción*** hacemos una descripción del problema que originó este estudio, la falta de un apartado en el modelo matemático de estimación de esfuerzo de COCOMO II que exprese de manera más detallada el esfuerzo inherente al desarrollo e implementación de una base de datos en un proyecto, los alcances y limitaciones que puede tener la solución propuesta en la tesis. Además de analizar aspectos básicos del modelo como: las métricas de software que existen actualmente (sus ventajas y desventajas) y aspectos generales que debe tener un modelo de estimación.

Una vez familiarizados con los objetivos, alcances y limitaciones de este estudio, además de entender plenamente la forma de operar de un modelo de estimación, se describe en el ***Capítulo 2 Fundamentos*** aquello que es necesario conocer acerca del funcionamiento del modelo COCOMO II, se analiza el apartado de COCOMO II para la estimación de una base de datos, lo cual nos lleva a la necesidad de formular de forma más exacta los parámetros relacionados con el desarrollo de una base de datos que se deben considerar en el modelo de estimación de esfuerzo de COCOMO II, que nos conduce al ***Capítulo 3 Estimación y diseño de una base de datos*** en el que se analiza en detalle el *Modelo Unificado (RUP)*.

Identificando en sus artefactos las características cuantificables relacionadas con el desarrollo de una base de datos y es así como se formula la descripción de la propuesta de solución de esta tesis.

Se describe toda la metodología propuesta para llevar acabo la estimación de la parte de la base de datos que corresponda a un proyecto y la manera de cómo se calibra el modelo COCOMO II con los nuevos atributos, demostrando su comportamiento como parámetros de esfuerzo válidos en la expresión matemática de estimación de esfuerzo.

Para verificar la nueva calibración del COCOMO II se propone un ejemplo de aplicación descrito en el ***Capítulo 4 Descripción del caso práctico,*** donde se maneja, como ejemplo de aplicación proyectos de software real que se encuentran en las fases de desarrollo. Uno de estos proyectos se enfoca en la parte de validación de información de los enlaces de fibra óptica que proporciona a las diversas entidades académicas que conforman la red UNAM y el otro es un sistema que automatiza la captura diversas mediciones de un sistema de red de abastecimiento de agua potable.

A estos proyectos se les aplica la metodología que propone COCOMO II originalmente y después se les calcula la estimación por medio de la solución de esta tesis descrita en el ***Capítulo 3***; posteriormente se estudian los resultados obtenidos del cálculo de la estimación del esfuerzo de la calibración propuesta por el trabajo de tesis y se hace una comparación con la estimación de esfuerzo que nos resulta al aplicar el modelo COCOMO II si se hubiera utilizado desde el inicio para apoyar la gestión del proyecto de software.

En el ***Capítulo 5 Conclusiones y prospectiva*** se discuten las conclusiones a las que se llega el estudio, además de las limitaciones que tiene la solución propuesta.

Capítulo 2

Nunca planeamos fracasar pero con mucha frecuencia fracasamos por no planear. *(Anónimo)*

2. Fundamentos

Se exponen en este capítulo las bases teóricas que fundamentan la propuesta en el trabajo de tesis.

2.1 Antecedentes de COCOMO

El modelo original *COCOMO (Constructive Cost Mode o Modelo Constructivo de Costo)* se publicó por primera vez en 1981 por Barry Boehm y reflejaba las prácticas en desarrollo de software de aquel momento. Se convirtió en el modelo paramétrico más popular entre los modelos de estimación de costos.

En la década y media siguiente, las técnicas de desarrollo software cambiaron drásticamente. Estos cambios incluyen el gasto de tanto esfuerzo en diseñar y gestionar el proceso de desarrollo software como en la creación del producto software, un giro total desde los mainframe que trabajan con procesos batch nocturnos hacia los sistemas en tiempo real y un énfasis creciente en la reutilización de software ya existente y en la construcción de nuevos sistemas que utilizan componentes software a la medida.

Éstos y otros cambios hicieron que la aplicación del modelo COCOMO original empezara a resultar problemática. La solución al problema era reinventar el modelo para aplicarlo a los 90. Después de muchos años de esfuerzo combinado entre USC-CSE1[1], IRUS y UC Irvine22[2] y las Organizaciones Afiliadas al Proyecto COCOMO II, el resultado es COCOMO II, un modelo de estimación de costo que refleja los cambios en la práctica de desarrollo de software profesional que ha surgido a partir de los años 70.

[6] Universidad del Sur de California. Centro para la Ingeniería en Software.

[7] Unidad de investigación de software Irvine. Universidad Irvine de California

Por tanto, COCOMO II es un modelo que permite estimar el costo, esfuerzo y tiempo cuando se planifica una nueva actividad de desarrollo software. Está asociado a los ciclos de vida modernos. El modelo original COCOMO ha tenido mucho éxito, pero no puede emplearse con las prácticas de desarrollo software más recientes tan bien como con las prácticas tradicionales. El modelo COCOMO es un modelo de perfeccionamiento continuo, es decir que éste se va estar calibrando empíricamente a medida que se recopilan datos históricos sobre proyectos actualmente terminados.

Por tanto, COCOMO II es un modelo que permite estimar el costo, esfuerzo y tiempo cuando se planifica una nueva actividad de desarrollo software. Está asociado a los ciclos de vida modernos. El modelo original COCOMO ha tenido mucho éxito pero no puede emplearse con las prácticas de desarrollo software más recientes tan bien como con las prácticas tradicionales. El modelo COCOMO es un modelo de perfeccionamiento continuo, es decir que este se va estar calibrando empíricamente a medida que se recopilan datos históricos sobre proyectos actualmente terminados.

Es por eso que la versión 81 (COCOMO 81) el modelo experimentó serias dificultades al enfrentarse a nuevos cambios en la manera de producir software y en la complejidad de los nuevos proyectos que se requería desarrollar.

COCOMO II apunta hacia los proyectos software de los 90 y de la primera década del 2000, y continuará evolucionando durante los próximos años.

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

- ✓ *Desarrollar un modelo de estimación de tiempo y de costo del software de acuerdo con los ciclos de vida utilizados en los 90 y en la primera década del 2000.*

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

- ✓ *Desarrollar bases de datos con los costos de software y herramientas de soporte para la mejora continua del modelo, (COCOMOII, 2000).*

COCOMO en sus dos versiones, es un modelo algorítmico de estimación de costos, basado en una función matemática. Dichos modelos son también llamados matemáticos paramétricos (como se comentó en el ***Capítulo 1***) están fundamentados en la utilización de fórmulas matemáticas mediante las cuales se obtiene el valor de un conjunto de variables dependientes, como son por ejemplo: el esfuerzo y el tiempo de desarrollo. (ver *Figura 2.1*).

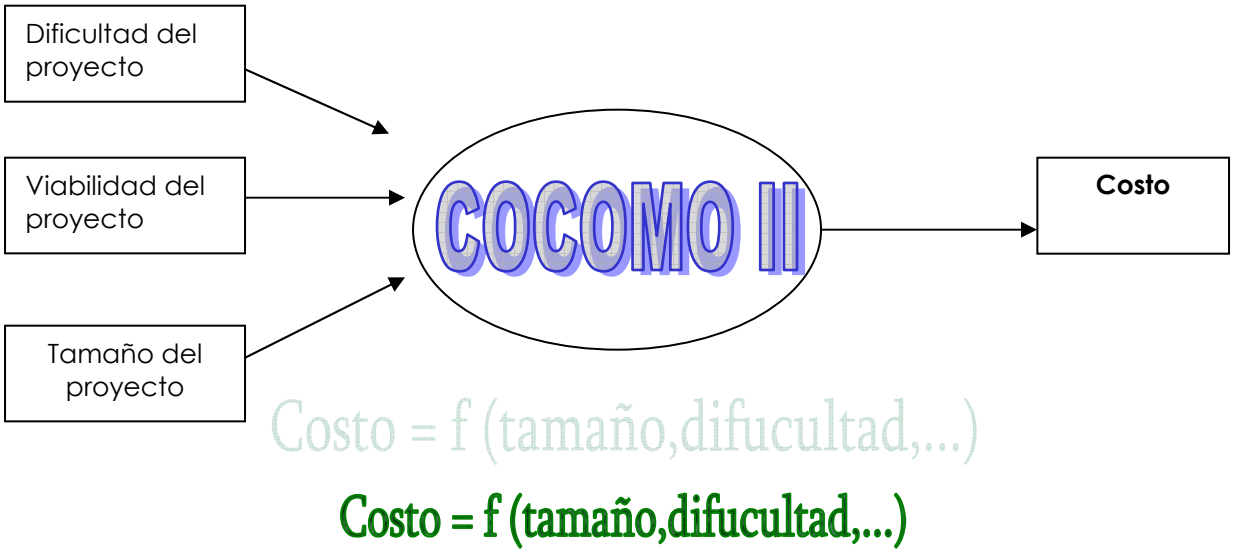


Figura 2.1 Modelo de estimación visto como función

Como se muestra en el diagrama anterior, los valores de salida son generados a partir de los valores de entrada dados por un conjunto de variables independientes, como: el tamaño de la aplicación en líneas de código, la fiabilidad requerida de la misma o su complejidad.

Las nuevas tendencias en el desarrollo de software van encaminadas al re-uso, re-ingeniería, paquetes comerciales listos para usarse, orientación a objetos, capacidades de composición de las aplicaciones, modelos de procesos no secuenciales, enfoque de desarrollo rápido y las capacidades distribuidas de la capa media (middleware), por lo tanto requieren de nuevos enfoques en la estimación de costo del software. Por eso COCOMO requiere parámetros como el *tamaño*, el cual lo define por las miles de instrucciones fuente obtenida (KDSI) como una medida de líneas de código, a la *dificultad* lo representa a partir de modos de proyecto como: orgánico, semi separado e incrustado y el *esfuerzo* lo mide en número de meses de trabajo.

En los modos que define el modelo para representar la dificultad del proyecto se van determinando por varios factores que necesitan ser ponderados; dichas calificaciones de cada factor se transforman en medidas cuantificables que se agregan a la función base del modelo como parámetros multiplicadores. Las medidas están ubicadas en rangos previamente establecidos que deben ser calibrados de acuerdo con cada tipo de proyecto mientras las condiciones de desarrollo y ambiente varíen constantemente.

Los modos del modelo se ilustran en la siguiente tabla (*Tabla 2.1*) con sus correspondientes grados de dificultad de acuerdo con las características del proyecto:

	Niveles de dificultad		
	Orgánico	**Semi detallado**	**Incrustado**
Comprensión de los objetivos	Amplia	Considerable	General
La correspondencia entre el software y las especificaciones de las Interfaces debe ser:	Básica	Considerable	Completa
Necesidad de algoritmos o arquitecturas innovadoras para el desarrollo del software en cuestión	Alguna	Moderada	Extensiva

Tabla 2.1 Modos del Modelo COCOMO II

Los modelos algorítmicos proveen estimados directos del esfuerzo o la duración del proyecto y la entrada principal suele ser una predicción del tamaño del software. Los modelos de predicción del esfuerzo se expresan en forma general de la siguiente manera:

$$Esfuerzo = K_P \times T \qquad \textbf{...(2.1)}$$

Donde:

K_p : Constante de productividad.

T: Tamaño del proyecto.

De esta misma manera trabaja el modelo COCOMO, considerando:

- ✓ Tamaño del proyecto.
- ✓ Esfuerzo.
- ✓ Productividad.
- ✓ Modos de dificultad del proyecto.

Pero también toma en cuenta los siguientes atributos:

- ✓ La capacidad del programador.
- ✓ La continuidad del personal.
- ✓ La dificultad de la plataforma.
- ✓ La experiencia, que tiene el equipo de desarrollo, con la plataforma a tratar.
- ✓ La confiabilidad y complejidad requeridas del producto.
- ✓ El re-uso del proyecto (re-ingeniería).
- ✓ Los cambios de los requerimientos.
- ✓ Los cambios que llegaran a hacerse a la plataforma.

Los anteriores aspectos han enriquecido el modelo y dieron como resultado su nueva versión: COCOMO II.

La transición del modelo COCOMO 81 a COCOMO II se dio como medio para la satisfacción indirecta de los siguientes objetivos:

- ✓ Mejorar de manera significativa la productividad del desarrollo del software.
- ✓ Mejoramiento de la *calidad* y *confiabilidad* del software.

Los modelos algorítmicos en general hacen un buen trabajo, pero la calibración es esencial y el ambiente de desarrollo es muy importante.

2.1.1. ¿Cómo está compuesto COCOMO II?

COCOMO II consta de tres submodelos, cada uno podrá usarse dependiendo de la etapa de desarrollo en la que se encuentre el proyecto.
Los modelos son:

- ✓ Composición.
- ✓ Diseño Temprano.
- ✓ Modelo pos-arquitectónico.

Cada uno se describe de manera breve, en la *Figura 2.2*

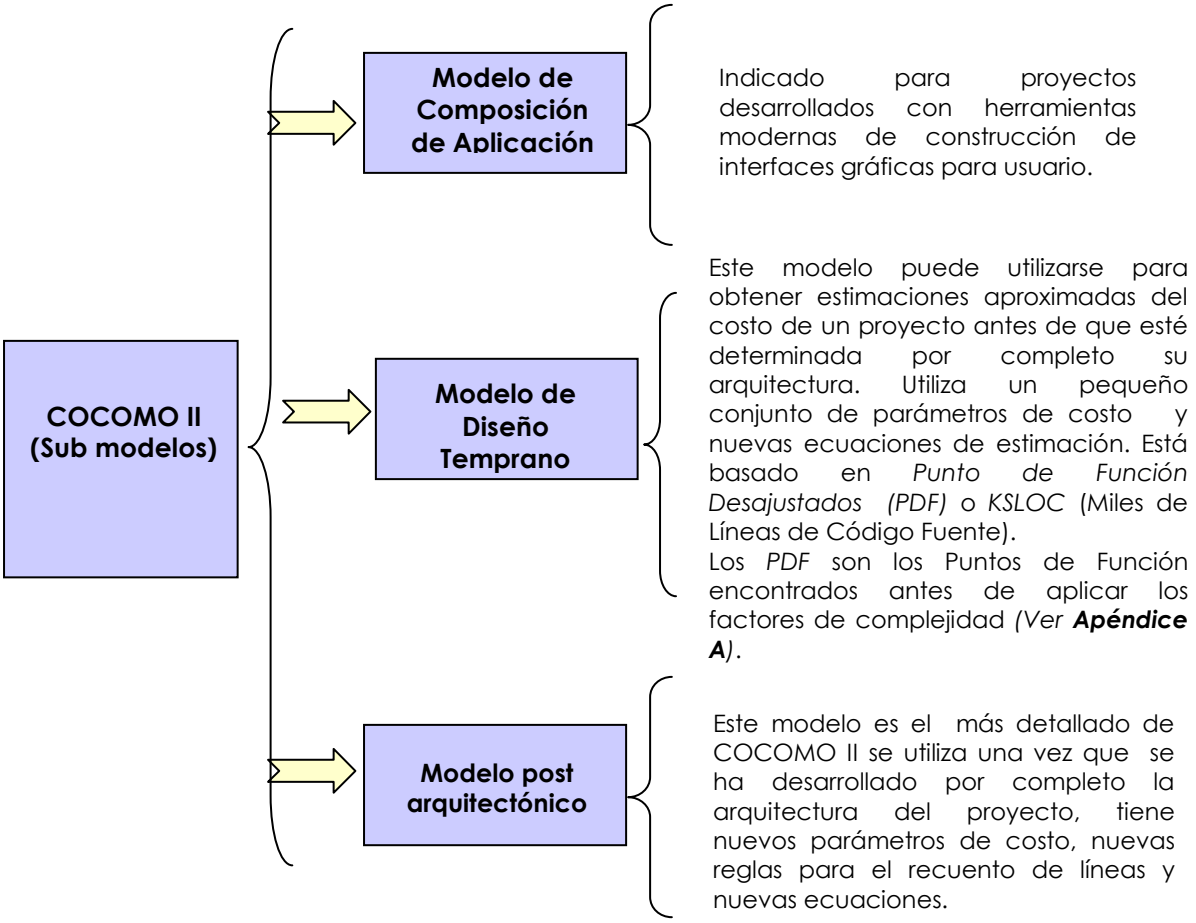


Figura 2.2 Submodelos pertenecientes a COCOMO II

2.1.1.1. Utilización de los submodelos

Modelo de composición

Las primeras fases o ciclos en espiral utilizados en proyectos de software relacionados con: la generación de aplicaciones, integración del sistema e infraestructura implicarán generalmente prototipado[3] y utilizarán los beneficios del *Módulo de Composición de Aplicaciones*.

El Módulo de Composición de Aplicaciones COCOMO II, soporta estas fases y cualquier otra actividad de prototipado; existen dos aproximaciones posibles:

- ✓ que sea desechable.
- ✓ que se mejore de forma incremental hasta conseguir la versión final del sistema.

Este modelo se dirige a aplicaciones que están demasiado diversificadas para crearse rápidamente en una herramienta de dominio específico, (como una hoja de cálculo) y que todavía no se conocen suficientemente como para ser compuestas a partir de componentes inter-operables.

Ejemplos de estos sistemas basados en componentes son los creadores de interfaces gráficas para usuario, bases de datos, gestores de objetos, middleware para proceso distribuido o transaccional, manejadores -hipermedia, buscadores de datos pequeños y componentes de dominio específico tales como paquetes de control de procesos financieros, médicos o industriales.

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

[3] Un prototipo es una versión incompleta de la aplicación, el propósito del prototipo es para que los clientes pudieran ver el producto antes que estuviese terminado.

Modelo de diseño anticipado

Hemos visto que las primeras fases o ciclos en espiral utilizados en proyectos de software relacionados con la generación de aplicaciones, integración del sistema e infraestructura se ajustaban mejor al *Modelo de Composición de Aplicaciones*.

Las siguientes fases o ciclos espirales normalmente incluirán la exploración de arquitecturas alternativas o estratégicas de desarrollo incremental, para sostener estas actividades.

COCOMO II proporciona un modelo de estimación anticipado, conocido con el nombre de Modelo de Diseño Anticipado. El nivel de detalle de este modelo puede ser consistente con el nivel general de información disponible y con el nivel general de aproximación de la estimación requerida en esta etapa, (*ob. cit COCOMO II*).

El Diseño Anticipado incluye la exploración de arquitecturas de software/sistema alternativas y conceptos de operación. En esta fase no se sabe lo suficiente como para dar soporte a la estimación de grano[4] fino.

La correspondiente capacidad de COCOMO II incluye el uso de Puntos de Función (definidos en el ***Capítulo 1***) y un conjunto de siete drivers de costo de grano grueso (por ejemplo, dos drivers de costo para capacidad del *personal* y *experiencia del personal* en lugar de los seis drivers de costo del Modelo Post-Arquitectura que cubren varios aspectos de capacidad del personal, continuidad y experiencia).

El modelo de Diseño Anticipado usa *Puntos de Función No Ajustados* o *Puntos de Función Desajustados* (*PFD*) como métrica de medida (para mayor información ver ***Apéndice A***). Este modelo se utiliza en las primeras etapas de un proyecto software, cuando se conoce muy poco sobre el tamaño del producto que se va a desarrollar, la naturaleza de la plataforma objetivo, la naturaleza del personal involucrado en el proyecto o especificaciones detalladas del proceso que se va a usar.

Este modelo puede aplicarse a cada uno de los sectores de desarrollo del desarrollo algún generador de aplicaciones, integración de sistemas o infraestructura.

[9] La estimación de grano se refiere al grado de detalle para realizar la estimación. Por ejemplo el número de parámetros de costo es un ejemplo del grano ya que da características que se incluyen en la estimación del proyecto.

Modelo post-arquitectura

Hemos visto que las primeras fases o ciclos en espiral usados en proyectos de software relacionados con la generación de aplicaciones, integración de sistema e infraestructura se ajustaban mejor al *Modelo de Composición de Aplicaciones* y que las siguientes fases o ciclos espirales normalmente serán sostenidas por el *Modelo de Diseño Anticipado*. Una vez que el proyecto está listo para desarrollar y sostener un sistema especializado, debe haber una arquitectura de ciclo de vida que proporcione información más precisa de los parámetros del esfuerzo o drivers de costo de entradas que permita cálculos de costo más exactos.

Para apoyar esta etapa COCOMO II proporciona el *Modelo Post-Arquitectura*.

El Modelo Post-Arquitectura incluye el actual desarrollo y mantenimiento de un producto software. Esta fase avanza rentablemente si se desarrolla una arquitectura de ciclo de vida software valida con respecto a la misión del sistema, al concepto de operación y al riesgo, estableciendo como una marca de trabajo del producto.

El modelo correspondiente de COCOMO II tiene aproximadamente la misma granularidad[5] que los anteriores modelos COCOMO y AdaCOCOMO[6].

Utiliza instrucciones fuente y/o PF para medir, con modificadores para reutilización y objetos; un conjunto de 17 drivers de costo multiplicativos; y un conjunto de 5 factores que determinan el exponente de escala del proyecto.

Estos factores sustituyen los modos de desarrollo (Orgánico, Semi-libre y Rígido) del modelo original COCOMO (COCOMO 81) y refina los 4 factores de exponente-escala en Ada COCOMO.

---

[5] La granularidad se refiere al grado de detalle que tiene los parámetros para realizar la estimación, es decir cuantas características del proyecto se cubren con los parámetros del modelo.

[6] Es el siguiente modelo de estimación que se desarrollo después de COCOMO o COCOMO 81 , es más refinado que el modelo original y asume el uso del proceso de desarrollo denominado ADA 87, para mayor información consultar : http://www.softstarsystems.com/advanced.htm

2.1.2. Ecuaciones nominales de estimación

Los *Modelos de Composición, de diseño y post-arquitectónico* utilizan las mismas ecuaciones nominales de estimación para calcular la cantidad de esfuerzo que se requerirá para desarrollar un proyecto de software.

2.1.2.1. Cálculo de la cantidad de esfuerzo

La cantidad de esfuerzo requerida se calcula con la siguiente expresión matemática denominada **Ecuación de Estimación de Esfuerzo**:

$$PM_{NS} = A \times Tamaño^{E} \times \prod_{i=1}^{n} EM_{i} \qquad \textbf{...(2.2)}$$

Donde:

PM: Cantidad de esfuerzo medida a partir de $\frac{Personas}{Mes}$

A: Constante de coeficiente de esfuerzo (coeficiente de ajuste).

Tamaño: Tamaño para el proyecto, en *PFD* o *KSLOC*

Donde:

$$Tamaño = \left[1 + \frac{BRAK}{100}\right] \qquad \textbf{...(2.3)}$$

En el cual COCOMO II utiliza un porcentaje de *Rotura BRAK (Breakage)* para ajustar el tamaño eficaz del producto; es decir, BRAK refleja la volatilidad de los requisitos en un proyecto. Es el porcentaje de código desperdiciado debido a la volatilidad de los requisitos.

Por ejemplo, un proyecto formado finalmente por 100,000 instrucciones de las que se han descartado otras 20,000 instrucciones adicionales, entonces BRAK tendrá un valor de 20. Esto debería usarse para ajustar el tamaño efectivo del proyecto a 120,000 instrucciones.

El factor BRAK no se usa en el Modelo de Composición de Aplicaciones, donde se espera un cierto grado de iteración en el producto y se incluye en la calibración de datos.

El tamaño de una aplicación se mide en unidades de líneas de código (SLOC) (discutidos en el ***Capítulo 1***), al igual que en la versión inicial del COCOMO, este valor se deriva de la medida de módulos software que constituirán el programa de aplicación; sin embargo, en la nueva versión COCOMO II puede estimarse también a partir de los *PFD* convirtiendo a SLOC y luego dividiendo por 1000. Dicho procedimiento lo aplicaremos para el estudio de los Casos de Uso convertidos en *PFD*, este procedimiento se describe a detalle en el ***Capítulo 3*** y se aplica en los casos prácticos que se muestran en el ***Capítulo 4***.

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

Si se escoge usar los *PFD* para determinar el tamaño del proyecto, éstos deben convertirse en líneas de código fuente en el lenguaje de implementación (ensamblador, lenguajes de alto nivel, lenguajes de cuarta generación, etc) para evaluar la relativamente la concisa implementación por *PF (*ver tabla de conversión de PFD a SLOC en ***Apéndice A****)*.

COCOMO II realiza esto tanto en el *Modelo de Diseño Anticipado* como en el de *Post-Arquitectura,* usando tablas que traducen PFD al equivalente SLOC. (El ejemplo de aplicación se describe en el ***Capítulo 4***).

Continuando con los parámetros de ***(2.2)***:

E: Exponente de escala. (Ahorro y gasto de software de escala).

EM: *Multiplicadores de Esfuerzo, Drivers de Costo o Parámetros de Esfuerzo* según las características del proyecto, (comprende los siguientes valores de 7 a 17según el modelo).

Estos multiplicadores sirven para capturar características de desarrollo del software que afectan al esfuerzo para completar el proyecto, los drivers de costo o parámetros de esfuerzo tienen un nivel de medida que expresa el impacto del driver en el esfuerzo de desarrollo.

*Estos valores pueden ir desde **extra bajo** hasta **extra alto**. Para el propósito del análisis cuantitativo, cada nivel de medida de cada driver de costo tiene un peso asociado. El peso se llama **multiplicador de esfuerzo (EM)**. La medida asignada a un driver de costo es 1.0 y el nivel de medida asociado con ese peso se llama nominal. Si un nivel de medida produce más esfuerzo de desarrollo de software, entonces su correspondiente EM está por encima de 1.*

Recíprocamente, si el nivel de medida reduce el esfuerzo entonces e l-correspondiente EM es menor que 1. La selección de multiplicadores de esfuerzo se basa en una fuerte razón que explicaría una fuente significativa de esfuerzo de proyecto o variación de la productividad independientemente, (Bohem, 2000).

El punto de estudio de esta tesis se centra en estos parámetros de esfuerzo, ya que se propone la creación de nuevos multiplicadores de esfuerzo que traten de estimar el esfuerzo que implica el desarrollo de un proyecto que utilice una base de datos, partiendo en que el proyecto total sigue un modelo de desarrollo de software. Este punto de partida se trata más ampliamente en ***Capítulo 3***.

En la ***(2.2)*** *n* representa el número de multiplicadores de esfuerzo **EM_j**, que depende del modelo que se esté usando. Si n = 17 multiplicadores, se usa en el *Modelo Post-arquitectónico* y 7 para el *Modelo de Diseño Temprano.*

El subíndice NS de ***(2.2)*** incluida en el término **PM_{NS}** indica que esta ecuación arroja estimaciones representativas del esfuerzo.

El exponente de escala en ***(2.2)*** denominado **ahorro y gasto de software de escala** se calcula con base en la siguiente expresión matemática:

$$E = B + K \times \sum_{j=1}^{5} SF_J \qquad \textbf{...(2.4)}$$

Donde:

E: Exponente de escala (ahorro y gasto de software de escala).
B: Constante del coeficiente de esfuerzo (constante de ajuste).
K: Constante multiplicadora que ajusta la escala, cuyo valor es K=0.001.
SF: Se refiere a 5 factores de escala, los cuales determinan las economías de esfuerzo. Son los mismos para cualquier submodelo.
(Bohem2, 2000 pp.1-2).

Los modelos de estimación de costo tienen un factor exponencial para considerar los gastos y ahorros relativos de escala encontrados en proyectos de software de distinto tamaño. (Ver más detalles ***Apéndice B***).

El exponente de escala E se obtiene mediante los denominados drivers de escala o multiplicadores de escala. La selección de estos drivers se basa en la razón de que ellos son un recurso significante de variación exponencial relacionado con el cambio en la productividad o en el esfuerzo.

2.1.2.2. Cálculo de la cantidad de tiempo de calendario

La cantidad de tiempo de calendario que tomará desarrollar el producto **$TDEV_{NS}$** está dada por la siguiente expresión matemática denominada **Cálculo del Tiempo de calendario**:

$$TDEV_{NS} = C \times (PM_{NS})^{F} \quad \text{...(2.5)}$$

Donde:

C: Constante ajustada (C= 3.67) obtenida por la calibración del calendario de desarrollo de los 161 proyectos vistos para la formación del modelo.

PM: Número de personas que trabajan por un mes estimadas sin considerar el plazo de tiempo que se requiere menor esfuerzo (SCED) $\frac{Persona}{Mes}$

F: Exponente de escala.

El subíndice **NS** de **$TDEV_{NS}$** expresa (al igual que en ***(2.1)***), que los resultados de las estimaciones son representativos del esfuerzo y el del tiempo de desarrollo.

La expresión que calcula el exponente de escala (F) de **(2.5)** es:

$$F = D + K_{FI} + K_{F2} \times \sum_{j=1}^{5} SF_J \quad \text{...(2.6)}$$

Donde:

D: constante ajustada cuyo valor es 3.67.
K_{F1}: constante de ajuste para el tiempo de desarrollo, K_{F1}=0.2.
K_{F2}: constante multiplicadora que ajusta la escala K_{F2}=0.001.
SF: Son 5 factores de escala los cuales determinan las economías de esfuerzo.

De la anterior expresión ***(2.4)*** y de ***(2.1)*** referente al factor de escala F, se puede hacer una sustitución para obtener una ecuación simplificada del exponente de escala F, ya que manejan la misma sumatoria de multiplicadores de esfuerzo ($\sum_{j=1}^{5} SF_J$) y tomando en cuenta que tienen la misma constante multiplicadora de 0.001; con estas consideraciones se puede obtener una ecuación para el cálculo del factor **F** simplificado.

$$F = D + 0.2 \times (E - B) \qquad \textbf{...(2.7)}$$

Quedando F en función de los factores de escala D, E, B y la constante de ajuste para el tiempo de desarrollo.

Los valores A, B, C, D, el rango de valores: EM_1 a EM_{16} y SF_1 a SF_5 para el modelo COCOMO II han sido obtenidos por medio de la calibración de los parámetros actuales, haciendo comparaciones con 161 proyectos que fueron analizados y a los que se le dio seguimiento el equipo de investigación del COCOMO.

Los valores C y D se han obtenido con la calibración del calendario de desarrollo de esos mismos proyectos.
Los valores A,B,C Y D y el rango SF_1a SF_5 son los mismos para *el modelo post-arquitectónico* y *el de diseño*.

El rango de valores EM_1,..., EM7 del Modelo de *Diseño Temprano* se obtienen combinando los valores de 17 parámetros del Modelo *Post arquitectónico.*

Para medir el tamaño del proyecto de software (como se había discutido en el ***Capítulo 1***) hay dos maneras por *miles de líneas de código fuente (KSLOC)* o los *PFD* detectados, (ver ***Apéndice A***).

El costo de desarrollo se obtiene multiplicando el esfuerzo en PM (esfuerzo medida por ($Persona/Mes$) por el promedio del PM.

Las calibraciones que se dan en el modelo de COCOMO II.1999 para los parámetros de calibración A,B,C y D son *(ob. cit COCOMO II)* :

- ✓ **A=2.94**
- ✓ **B=0.91**
- ✓ **C=3.67**
- ✓ **D=0.28**

Para dar una mejor explicación de cómo se emplean las expresiones de esfuerzo y de tiempo de calendario, podemos estimar cuánto tiempo y esfuerzo se necesitan para desarrollar un proyecto promedio; los multiplicadores de esfuerzo son iguales a 1.0; E se ajusta a 1.15 (para un proyecto promedio grande de acuerdo con lo que se establece en el ***Apéndice B***).

Además tomando en cuenta que el tamaño estimado del proyecto es de 100 KSLOC.

Calculando el estimado del esfuerzo:

De **(2.2)**:

$$PM_{NS} = A \times Tamaño^{E} \times \prod_{i=1}^{n} EM_i \qquad \textbf{...(2.2)}$$

Sustituyendo los valores correspondientes y sin tomar en cuenta los multiplicadores de esfuerzo.
A=2.94
Tamaño=100 KSLOC
E=1.15

$PM_{NS} = 2.94\ (100)^{1.15}$ **...(2.2.1)**

$PM_{NS} = 586.61\ \frac{Persona}{Mes}$ **...(2.2.2)**

Continuando con el ejemplo, la duración estimada del proyecto está dada por la *(2.5)*:

$$TDEV_{NS} = C \times (PM_{NS})^{F} \quad \textbf{...(2.5)}$$

Sustituyendo los valores correspondientes:

C=3.67
PM=586.61 (obtenido en el inciso anterior)

Para calcular F utilizando *(2.7)*:

$$F = D + 0.2 \times (E - B) \quad \textbf{...(2.7)}$$

Se tienen los siguientes datos:
D=0.28
E=1.15
B=0.91

$\therefore$ F=0.28+0.2 (1.15-0.91) $\Rightarrow$ F=0.328

Por lo que:

$TDEV_{NS} = 3.67\ (586.61)^{0.328}$

$TDEV_{NS}$=29.7 ≈30 meses de desarrollo, pero esto no está apegado mucho a la realidad, falta considerar mas factores importantes.

Para calcular el número de personas que se necesitaran para desarrollar el proyecto, lo calculamos haciendo un cociente entre el tiempo de desarrollo estimado y el esfuerzo estimado.

$$Personas - desarrollo = \frac{PM_{NS}}{TDEV_{NS}} \qquad \textbf{...(2.8)}$$

Así que:

Personas de desarrollo $\frac{586.6}{30} = 19.53 \approx 20$

Con este pequeño ejemplo podemos decir que, un proyecto promedio con un tamaño de *100 KSLOC* nos llevará aproximadamente *30 meses* con un equipo de desarrolladores de 20 personas promedio.

2.1.3. Factores de escala y parámetros de esfuerzo

En las primeras etapas de un proyecto es posible que se desconozcan ciertos aspectos sobre éste, tales como:

- ✓ Plataforma en que deberá desempeñarse el producto.
- ✓ La naturaleza del personal que estará involucrado en el proyecto.
- ✓ Detalles específicos sobre los procesos que deberán llevarse a cabo.
- ✓ Detalles de la base de datos.

La aplicación de *factores de escala exponenciales* (**SF**) es la misma para el modelo *de Diseño temprano* y para el *Post-arquitectónico*, las cuales se muestran en la siguiente *Tabla 2.2*:

Factor de escala	Descripción:
PREC	Establece el grado de asimilación de los objetivos del proyecto por parte del equipo de desarrollo.
RESL	Califica la asimilación de la arquitectura de software del producto y de los factores críticos de riesgo.
TEAM	Captura la consistencia entre los objetivos administrativos sobre el producto y la buena disposición de todos los miembros del equipo para trabajar juntos, como equipo. Cohesión.
PMAT	Califica la madurez del proceso de desarrollo usado para producir el producto. El Criterio está directamente relacionado con el modelo CMM.
FLEX	Expresa el grado de flexibilidad respecto al acatamiento de los requerimientos y a los estándares de las Interfaces Externas.

Tabla 2.2 Factores de Escala

Esos factores de escala, determinan el valor del exponente B de nuestra ecuación de estimación de esfuerzo ***(2.2)*** la selección de estos factores se basa en la razón de que ellos son un recurso significante de variación exponencial en un esfuerzo o variación de la productividad del proyecto.

Cada escala tiene un rango de niveles de valores desde ***muy bajo*** hasta **extra alto** (estas ponderaciones se obtiene por medio de una guía que se describe en el ***Capítulo 4***), cada nivel de valores tiene un peso (*SF*) y el valor específico del peso se llama *factor de escala (para ver valores de los factores ir al* ***Apéndice B****)*. Un factor de escala de un proyecto, SF_j se calcula sumando todos los factores y se usa para determinar el exponente de escala: E definido en ***(2.4)***.

2.1.3.1. Descripción de los parámetros del esfuerzo

Como se explicó anteriormente, los parámetros del esfuerzo ***(2.2)***:

$$\prod_{j=1}^{n} EM_i$$

Se usan para capturar las características de desarrollo del software que afectan el esfuerzo para completar el proyecto, ahora se describirán esos parámetros, dependiendo del tipo de modelo que se esté usando, por ejemplo para el *Modelo de Diseño Temprano* hay *7* multiplicadores de esfuerzo mientras tanto para el *Modelo de Post-arquitectura* hay 17, como se muestra en la *Tabla 2.3*.

Los *drivers de costo del Diseño Anticipado* corresponden a los parámetros de esfuerzo del *Modelo de Diseño Temprano* y se puede observar que hay una combinación de multiplicadores con respecto al *Modelo Post arquitectura (para mayor información ver COCOMO II, 2000 Pág. 51a 55)* , lo cual nos indica que los 7 parámetros del *Modelo de Diseño Temprano* son el resultado de la combinación de los 17 parámetros del *Modelo Post arquitectura*; de los cuales se describirán primero los parámetros del *Modelo de Diseñó Anticipado* y después dentro de la explicación de cada uno de éstos, se detallaran los drivers de costo que están involucrados correspondientes al *Modelo Post- Arquitectura*.

Drivers de coste del Diseño Anticipado	Drivers de coste combinados, homólogos del Post-Arquitectura
RCPX	RELY, DATA, CPLX, DOCU
RUSE	RUSE
PDIF	TIME, STOR, PVOL
PERS	ACAP, PCAP, PCON
PREX FCIL	AEXP, PEXP, LTEX TOOL, SITE
SCED	SCED

Tabla 2.3 Parámetros de esfuerzo, de acuerdo con el modelo utilizado

Los *drivers de costo del Diseño Anticipado* corresponden a los parámetros de esfuerzo del *Modelo de Diseño Temprano* y se puede observar que hay una combinación de multiplicadores con respecto al *Modelo Post-arquitectura (para mayor información ver COCOMO II, 2000 Pág. 51a 55)* , lo cual nos indica que los 7 parámetros del *Modelo de Diseño Temprano* son el resultado de la combinación de los 17 parámetros del *Modelo Post arquitectura*; de los cuales se describirán primero, los parámetros del *Modelo de Diseñó Anticipado* y después, dentro de la explicación de cada uno de éstos, se detallaran los drivers de costo que están involucrados correspondientes al *Modelo Post- Arquitectura*.

Los parámetros de esfuerzo que corresponden al *Modelo Post-Arquitectura* se clasifican de acuerdo con la característica que se está estimando del proyecto, tales categorías en las que se reúnen estos atributos identificables en el sistema son:

- ✓ Factores de proyecto.
- ✓ Factores del personal.
- ✓ Factores de la plataforma.

1) PERS

Parámetro referente a la capacidad personal del desarrollador. En el modelo de *Diseño Temprano* el parámetro PERS combina los parámetros de costo del modelo *Post arquitectónico*:

- ✓ Capacidad del analista (ACAP).
- ✓ Capacidad del programador (PCAP).
- ✓ Continuidad del personal (PCON).

Agregando las escalas numéricas de cada uno se producen rangos de 3 a 15 puntos.

ACAP (Capacidad del analista o habilidad del analista).

Los analistas son personal que trabaja en los requisitos de diseño de alto nivel y en diseño detallado. Los atributos principales que deben considerarse en esta medida son la habilidad de análisis y diseño, la eficiencia y minuciosidad y la habilidad para comunicar y cooperar.

La medida no debe considerar el nivel de experiencia del analista, eso se mide con AEXP, los analistas que obtienen un porcentaje de 90 % se evalúan como ***muy alto***.

PCAP (Habilidad del programador o capacidad del programador).

Las tendencias actuales continúan dando énfasis a la capacidad de los analistas. Sin embargo, el creciente papel de los paquetes complejos COTS[7] y la relevante influencia asociada a la capacidad de los programadores para tratar con esos paquetes COTS, también indica una tendencia a darle mayor importancia a la capacidad del programador.

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

Unos valores ***muy bajos*** del equipo de programadores corresponden a un 15 % y ***muy alto*** a un 90%.

PCON (Continuidad del personal.

La escala de valores de PCON se mide en términos del movimiento de personal del proyecto anualmente: desde 4%, ***muy alto***, hasta el 45%, ***muy bajo***.

2) RCPX (Confiabilidad y complejidad del producto).

En el modelo de *Diseño temprano* se combinan cuatro factores de costo:

- ✓ Confiabilidad requerida del software (RELY).
- ✓ Tamaño de la base de datos (DATA).
- ✓ Complejidad del producto (CPLX).
- ✓ Documentación adecuada para el ciclo de vida (DOCU).

A diferencia del PERS. Los componentes de RCPX tienen diferentes escalas con diferentes proporciones, como se describe a continuación:

Para RELY (Fiabilidad del software).

Esta es la medida de hasta qué punto el software debe realizar su función esperada durante un periodo de tiempo. Si el efecto de un fracaso es sólo una molestia ligera entonces RELY es ***bajo***. Si un fallo arriesgase vidas humanas entonces RELY es ***muy alto***.

[7] COTS : Commercial-Off-The-Shelf

DOCU (Documentación asociada a las necesidades del ciclo de vida).

Varios modelos de costo software tienen un parámetro de esfuerzo para el nivel de documentación requerida. En COCOMO II la escala de medida para el parámetro de costo se evalúa en términos de la adecuación de la documentación del proyecto a las necesidades de su ciclo de vida.

DATA (Medida del Volumen de Datos).

Esta medida intenta capturar lo que afecta en el desarrollo del producto unos requerimientos de almacenamiento de grandes cantidades de datos. La medida se determina calculando el cociente: $\frac{D}{P}$.

La razón por la que es importante considerar el tamaño de la Base de Datos es por el esfuerzo necesario para generar datos de prueba que se usarán para ejecutar el programa.

$$\frac{D}{P} = \frac{Databasesize[Bytes]}{PorgramSize[SLOC]}$$

Único apartado que hace COCOMO II referente al desarrollo de la base de datos dentro de un proyecto.

CPLX (complejidad del producto).

La complejidad se divide en cinco áreas:

- ✓ Funcionamiento de control.
- ✓ Funcionamiento computacional.
- ✓ Funcionamiento de Dispositivos dependientes.
- ✓ Funcionamiento del sector de datos.
- ✓ Funcionamiento del Gestor de Interfaz de Usuario.

Se seleccionará el área o combinación de áreas que caracterizan al software, o bien, a un subsistema del software de acuerdo con la tabla que se describe en el ***Apéndice B*** en el apartado de CPLX.

3) PDIF (Dificultad de la plataforma).

Este parámetro del Modelo Diseño Temprano combina los tres parámetros de esfuerzo del Modelo Post-Arquitectura siguientes:

- ✓ Tiempo de Ejecución (TIME).
- ✓ Restricciones de Almacenamiento (STOR).
- ✓ Volatilidad de la Plataforma (PVOL).

TIME (Tiempo de ejecución).

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

STOR (Restricciones de Almacenamiento).

Esta medida representa el grado de restricción de almacenamiento principal impuesto a un sistema o subsistema software.

Debido al aumento considerable del tiempo de ejecución disponible del procesador y del almacenamiento principal (actuales) uno puede cuestionar la evaluación de estas variables.

Los valores van desde nominal, menos que el 50%, a Extra Alto, 95%.

PVOL (Volatilidad de la plataforma).

La palabra *plataforma* se usa aquí para explicar la complejidad del hardware y software (por ejemplo un sistema operativo, DBMS,[8]... etc) que el sistema necesita para realizar sus tareas.

Si el software a desarrollar es un sistema operativo, entonces la plataforma es el hardware de la computadora. Si se desarrolla un DBMS, entonces la plataforma es el hardware y el Sistema Operativo.

[8] DBMS (Database Management System) o SGBD (Sistema Gestor de Bases de Datos) para mayor información : http://en.wikipedia.org/wiki/Database_management_system

La plataforma incluye cualquier compilador o ensamblador que soporta el desarrollo del sistema. Los valores van desde ***bajo*** donde 12 meses hay un cambio importante, hasta ***muy alto***, donde hay algún cambio importante cada dos semanas.

4) PREX (Experiencia del Personal).

Este parámetro combina tres factores del modelo post-arquitectónico:

- ✓ Experiencia sobre la aplicación (AEXP).
- ✓ Experiencia en el lenguaje y la herramienta (LTEX).
- ✓ Experiencia en la plataforma (PEXP).

Cuyas explicaciones de cada uno de estos parámetros de esfuerzo correspondientes al *Modelo Post-arquitectónico* se detallan a continuación:

AEXP (Experiencia sobre la aplicación).

Este módulo depende del nivel de experiencia en aplicaciones del equipo de proyecto al desarrollar sistemas o subsistemas de software. Los valores se definen en términos del nivel de experiencia del equipo de desarrollo del proyecto en este tipo de aplicaciones. Un valor muy bajo de experiencia en aplicaciones es menor a 2 meses. Un valor muy alto corresponde a 6 años o más.

LTEX (Experiencia en la herramienta y en el lenguaje).

Esta es una medida del nivel de experiencia en el lenguaje de programación y en la herramienta software del equipo del proyecto que desarrolla el sistema o subsistema software.

Además de la experiencia programando en un lenguaje específico, las herramientas que dan soporte tales como: el uso de herramientas que realizan representación de requisitos y diseño, análisis, gestión de la configuración, origen de los documentos, gestión de librería, estilo de programa y estructura, verificación de consistencia, entre otros; también influyen en el tiempo de desarrollo. Tener una experiencia de menos de dos meses corresponde a un valor ***bajo***. Si es de 6 o más años el valor es ***muy alto***.

PEXP (Experiencia en la plataforma).

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

5) FCIL (Instalaciones).

Este parámetro combina dos factores del modelo arquitectónico:

- ✓ Uso de las herramientas de software (TOOL).
- ✓ Desarrollo multi sitios (SITE).

Donde cada parámetro del *Modelo Post-arquitectónico* significa:

TOOL (Uso de las herramientas de software).

Las herramientas software han mejorado significativamente desde los 70 proyectos usados para calibrar COCOMO. Los valores para la herramienta van desde edición y código simple, (***muy bajo***) hasta herramientas integradas de gestión del ciclo de vida, para la cual le corresponde un valor de ***muy alto***.

SITE (Desarrollo multi sitios).

Dada la frecuencia creciente de desarrollos multi-sitios o multi-lugar. Los efectos del desarrollo en diferentes lugares han sido muy grandes por eso se ha añadido en COCOMO II el parámetro *SITE*.

Determinar la medida del parámetro incluye el cálculo y la medida de dos factores:

- ✓ Localización del lugar (desde totalmente localizado, hasta distribución internacional).

- ✓ Soporte de comunicación (desde correo y algún acceso telefónico hasta multimedia totalmente interactivo).

6) SCED (Calendario requerido para el desarrollo).

Este valor mide las restricciones impuestas al equipo de desarrollo. Los valores se definen en términos de porcentajes de aceleración o alargamiento sobre el calendario con respecto al calendario nominal para un proyecto que requiere una cantidad de esfuerzo dado.

Los calendarios acelerados tienden a producir un mayor esfuerzo en las fases más tardías de desarrollo debido a que se dejan por solucionar más problemas a causa de la falta de tiempo para resolverlos de manera inmediata. Una compresión de calendario del 74% se evalúa como ***muy bajo***. Un alargamiento de calendario produce más esfuerzo en las etapas tempranas del desarrollo, donde hay más tiempo para planificar, hacer validaciones y especificaciones más minuciosas. Un alargamiento del 160% se valora como ***muy alto***.

7) RUSE (Re usabilidad).

Este parámetro se usa para considerar el esfuerzo adicional requerido para construir componentes, con la intención de volver a utilizarlos en proyectos futuros o actuales.

Este esfuerzo se consume al crear diseños más generales del software, documentación más elaborada, y pruebas más exhaustivas para asegurarse de que los componentes están listos para ser usados por otras aplicaciones, ya que se necesita una confiabilidad en estos componentes muy alta.

Para una mejor comprensión de los parámetros de esfuerzo se muestra una tabla resumen comparativa de cada uno de los factores, correspondientes al *Modelo Post- arquitectura, (ver Tabla 2.4).*

	Very Low	Low	Nominal	High	Very High	Extra High
RELY	slight inconve-nience	low, easily recoverable losses	Moderate, eas-ily recoverable losses	high financial loss	risk to human life	
DATA		DB bytes/Pgm SLOC < 10	10 ≤ D/P < 100	100 ≤ D/P < 1000	D/P ≥ 1000	
CPLX	see Table II-15					
RUSE		none	Across project	across program	across product line	across multi-ple product lines
DOCU	Many life-cycle needs uncovered	Some life-cycle needs uncovered.	Right-sized to life-cycle needs	Excessive for life-cycle needs	Very exces sive for life-cycle needs	
TIME			50% use of available exe cution time	70%	85%	95%
STOR			50% use of available stor age	70%	85%	95%
PVOL		major change every 12 mo.; minor change every 1 mo.	major: 6 mo.; minor: 2 wk.	major: 2 mo.; minor: 1 wk.	major: 2 wk.; minor: 2 days	
ACAP	15th percentile	35th percentile	55th percentile	75th percentile	90th percentile	
PCAP	15th percentile	35th percentile	55th percentile	75th percentile	90th percentile	
PCON	48% / year	24% / year	12% / year	6% / year	3% / year	
AEXP	≤ 2 months	6 months	1 year	3 years	6 years	
PEXP	≤ 2 months	6 months	1 year	3 years	6 year	
LTEX	≤ 2 months	6 months	1 year	3 years	6 year	
TOOL	edit, code,	simple, fron	basic lifecycle	strong, mature	strong, mature,	

	debug	tend, backend CASE, little integration	tools, moder ately inte grated	lifecycle tools, moderately integrated	proactive life cycle tools, well inte grated with processes, methods, reuse	
SITE: Colloc ation	International	Multi-city and Multi-com pany	Multi-city or Multi-com pany	Same city or metro. area	Same building or complex	Fully collo cated
SITE: Comm unicati ons	Some phone, mail	Individual phone, FAX	Narrowband email	Wideband electronic communica tion.	Wideband elect. comm, occasional video conf.	Interactive multimedia
SCED	75% of nomi nal	85%	100%	130%	160%	

Tabla 2.4 Tabla resumen de parámetros de esfuerzo modelo Post arquitectura

En las siguientes tablas: Tabla *2.6* y *Tabla 2.7* se muestran los valores multiplicativos de cada uno de los parámetros del modelo *Diseño Temprano* y *Post arquitectura* respectivamente, de acuerdo con la versión USC-COCOMOII.1999.0

A= 2.94 y B**=0.91** (Baseline Effort Constants)

C=3.67 Y **D=0.28** (Baseline Schedule Constants)

Dirver	Extra Low	Very Low	Lowl	Nominal	High	Very High	Extra High
PERS	2.12	1.62	1.26	1.00	0.83	0.63	0.50
RCPX	0.49	0.60	0.83	1.00	1.33	1.91	2.72
PDIF			1.00	1.00	1.00		
PREX	1.59	1.33	1.12	1.00	0.87	0.74	0.62
FCIL	1.43	1.30	1.10	1.00	0.87	0.73	0.62
RUSE			0.95	1.00	1.07	1.15	1.24
SCED		1.43	1.14	1.00	1.00	1.00	

Tabla 2.5 Valores multiplicativos de los EM para Modelo de Diseño Temprano.

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

Tabla 2.6 Valores multiplicativos de los EM para Modelo Post-arquitectura

Estos valores multiplicativos son los que afectan directamente al esfuerzo de desarrollo del proyecto de software, es en este punto donde centraremos nuestra primera premisa de nuestro estudio, ya que al tener parámetros que califiquen de manera cuantitativa el esfuerzo inherente al desarrollo de una base de datos como parte del proyecto, nos estará modificando nuestra estimación del esfuerzo.

Observaremos que las características del trabajo al realizar una base de datos desde el modelo lógico hasta llegar al modelo físico, se comportan como un parámetro de esfuerzo o driver de esfuerzo aquí; la ventaja de utilizar COCOMO II ya que hay esa posibilidad de ajustar el modelo a nuestras necesidades.

Capítulo 3

El cliente sólo busca tres cosas de un producto de software: calidad, costo y tiempo de entrega
Alanís Cantú

3.1 Diseño de una base de datos siguiendo el Proceso Unificado de Rational (RUP)

Utilizaremos el Proceso Unificado de Rational (RUP en sus siglas en inglés) para el diseño de una base de datos[1] como parte del proyecto en general de software, debido a que es más sencillo estimar si se sigue un proceso establecido que recopila las mejores prácticas de desarrollo y que facilita el trabajo de los diseñadores y por ende, se reducen el esfuerzo y tiempo.

Se seguirá un método para determinar el tamaño del software, el cual es una aplicación más simple para encontrar los Puntos de Función (PF) definidos en el ***Capítulo 1***, necesarios para la estimación con COCOMO II, utilizando los Casos de Uso, debido a que RUP trabaja con éstos durante todo el proceso.

3.1.1 Descripción de RUP

- **Breve historia**

El antecedente más importante se ubica en 1967 (la *Figura 3.1* ilustra la historia de RUP) con la Metodología Ericsson (*Ericsson Approach*) elaborada por Ivar Jacobson, una aproximación de desarrollo basada en componentes, que introdujo el concepto de Caso de Uso. Entre los años de 1987 a 1995 Jacobson fundó la compañía *Objectory* AB y lanza el proceso de desarrollo *Objectory* (abreviación de *Object Factory*).

[1] Una base de datos es en esencia una colección de archivos relacionados entre sí, de la cual los usuarios pueden extraer información sin considerar las fronteras de los archivo

Posteriormente en 1995 *Rational Software Corporation* adquiere *Objectory AB* y entre 1995 y 1997 se desarrolla *Rational Objectory Process* (ROP) a partir de *Objectory* 3.8 y del Enfoque Rational (*Rational Approach*) adoptando el **Unified Modeling Language o Lenguaje Unificado de Modelado** [2] (**UML**) como lenguaje de modelado.

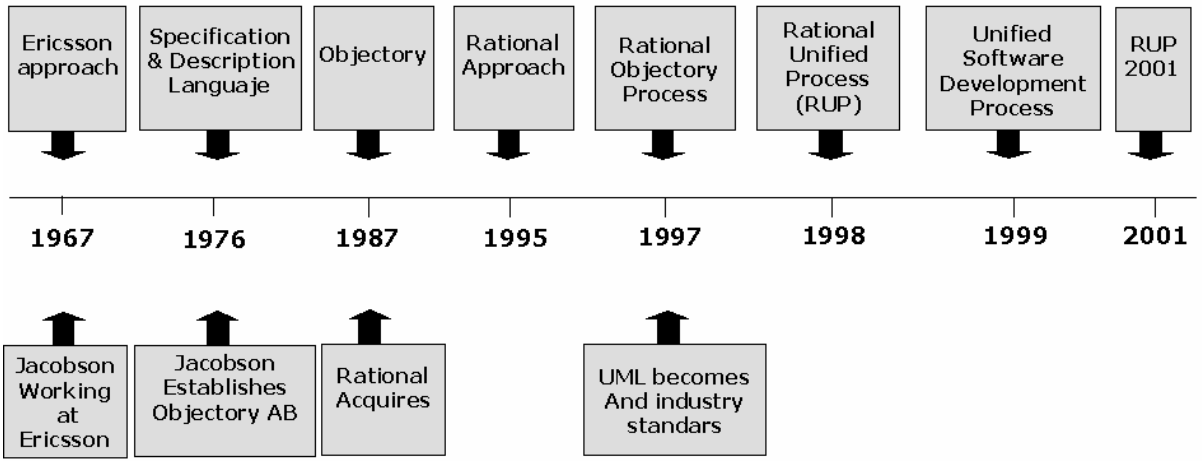


Figura 3.1: Historia de RUP *(Apuntes U.PV, 2001)*

Posteriormente, en 1995 *Rational Software Corporation* adquiere *Objectory AB* y entre 1995 y 1997 se desarrolla *Rational Objectory Process* (ROP) a partir de *Objectory* 3.8 y del Enfoque Rational (*Rational Approach*) adoptando el **Unified Modeling Language o Lenguaje Unificado de Modelado** (**UML**) como lenguaje de modelado.

En resumen este proceso se deriva de metodologías anteriores desarrolladas por estos tres autores, la "metodología Objectory" de Jacobson, la "metodología de Boch" y la "técnica de modelado de objetos" de Rumbaugh, *(Braude, 2003)*.

Desde ese entonces y a la cabeza de Grady Booch, Ivar Jacobson y James Rumbaugh, Rational Software desarrolló e incorporó diversos elementos para expandir RUP, destacándose especialmente el flujo de trabajo conocido como modelado del negocio. En junio del 1998 se lanza Rational Unified Process,. (Apuntes de U.P.V., 2001).

[2] Para mayor información de UML consultar : http://www.uml.org y http://www.omg.org

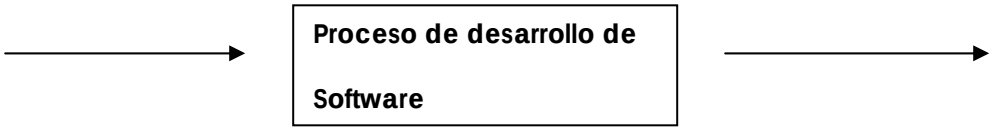
- **Bases teóricas**

El enfoque que se dará al RUP es aplicarlo para el desarrollo de una base de datos, ya que es una magnífica guía que define paso a paso las responsabilidades y los artefactos necesarios para el desarrollo y diseño de una base de datos. Provee de una manera sencilla la plantilla que orienta al diseñador sobre las interacciones necesarias entre él y los desarrolladores, aporta una lista de comprobación que enumera cada uno de los componentes.

RUP es un proceso de Ingeniería de Software. Proporciona un acercamiento disciplinado a la asignación de tareas y responsabilidades en una organización de desarrollo, *(Martínez, 2001)*.

Analizando el párrafo anterior, se puede decir que el RUP es un modelo de un proceso de desarrollo de software, el cual es el conjunto de actividades necesarias para transformar los requisitos de un usuario en un sistema de software, *(ver Figura 3.2)*.

RUP es una metodología que intenta integrar todos los aspectos a tener en cuenta durante todo el ciclo de vida del software (para el propósito de este trabajo nos referiremos al ciclo de vida de una base de datos); tiene como objetivo: *especializarse para una gran variedad de sistemas de software, para diferentes áreas de aplicación, tipos de organizaciones, niveles de aptitud y tamaños de proyecto*.



Requisitos del usuario
(no necesariamente humano)

Sistema de software
(en nuestro caso una BD[3])

Figura 3.2 Un proceso de desarrollo de software

[3] BD.-Base de datos

El RUP está basado en componentes, lo cual quiere decir que el sistema de software en construcción está formado por *componentes de software interconectados* a través de *interfaces* bien definidas y como se había mencionado, el RUP utiliza el estándar de modelado visual, el *Lenguaje Unificado de Modelado* (*UML*), además se sostiene en tres ideas básicas: *Casos de Uso, Arquitectura,* y *Desarrollo, (Jacobson, 2000)*.

No obstante, los verdaderos aspectos definitorios del RUP se resumen en las tres frases clave mencionadas por Jacobson:

- ✓ **Dirigido por Casos de Uso**

La captura de requisitos tiene dos objetivos: encontrar los verdaderos requisitos y representarlos de un modo adecuado para los usuarios, clientes y desarrolladores; nos referimos a verdaderos requisitos a aquellos que al ser implementados cubran las necesidades para los usuarios. Con respecto a la representación de los requisitos de modo adecuado para los usuarios, desarrolladores y clientes nos referimos a que la descripción obtenida de éstos debe ser comprensible.

Un sistema tiene muchos tipos de usuarios, cada tipo de usuario se representa por un actor. Los actores utilizan el sistema interactuando con los *Casos de Uso*, los cuales son una secuencia de acciones que el sistema lleva a cabo para ofrecer algún resultado de valor para un actor y constituyen la guía fundamental establecida para las actividades a realizar durante todo el proceso de desarrollo, incluyendo el diseño, la implementación y las pruebas del sistema, *(ob.cit. Jacobson)*.

Hay una definición más formal de Casos de Uso y es:

Un Caso de Uso especifica una secuencia de acciones, incluyendo variantes, que el sistema puede llevar a cabo, y que producen un resultado observable de valor para un actor en concreto, (Kruchten, 2000).

Los Casos de Uso constituyen un elemento integrador y una guía del trabajo como se muestra en la *Figura 3.3*.

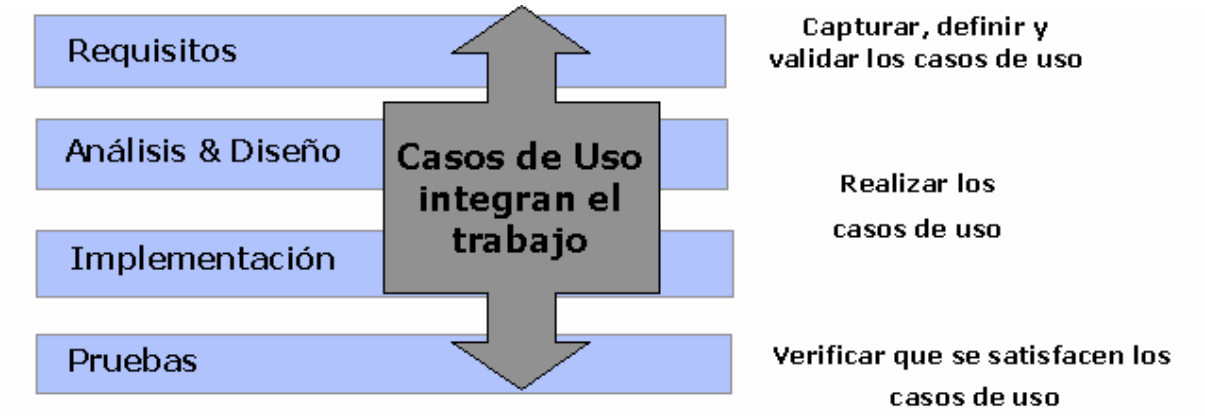


Figura 3.3 Los Casos de Uso integran el trabajo *(Apuntes U.PV, 2001)*

Basándose en los Casos de Uso se van creando los Modelos de Análisis y Diseño, posteriormente la implementación que los lleva a cabo, y se verifica que efectivamente el producto sea implementado de manera adecuada en cada Caso de Uso. Todos los modelos deben estar sincronizados con el Modelo de Casos de Uso, *(ver Figura 3.4)*.

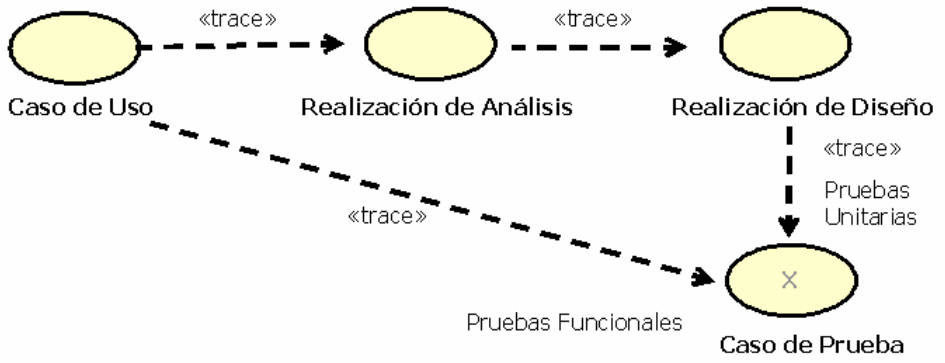


Figura 3.4 Trazabilidad[4] de los módulos a partir de los Casos de Uso*(ob cit. Apuntes U.V.)*

En esta idea aplicaremos una metodología, para obtener los PF a partir de los Casos de Uso, necesarios para la obtención de una estimación del desarrollo de una base de datos utilizando COCOMO II.

[4] Trazabilidad (traza o trace en lenguaje UML) se refiere a la dependencia que indica una relación histórica o de un proceso entre dos elementos que representan el mismo concepto, sin reglas específicas para la derivación de una a la otra.

- ✓ **Centrado en la arquitectura.**

La arquitectura involucra los elementos más significativos del sistema y está influenciada por plataformas, sistemas operativos, manejadores de bases de datos, protocolos, consideraciones de desarrollo. (Braude, 2003).

Además la arquitectura está relacionada con la toma de decisiones que indican como tiene que estar construido el sistema y ayuda a determinar en que orden.

Es una radiografía del sistema que se está desarrollando, lo suficientemente completa como para que las personas implicadas en el desarrollo del sistema tengan una idea de lo que están haciendo, pero lo suficientemente simple como para que si no especificamos alguna parte del sistema el proyecto se puede ir a pique, (Martínez, 2001).

La arquitectura debe tomar en consideración elementos de calidad del sistema, rendimiento, reutilización y capacidad de evolución, por lo que debe ser flexible durante todo el proceso de desarrollo.

Cada producto tiene tanto una función como una forma. La función corresponde a la funcionalidad reflejada en los Casos de Uso y la forma la proporciona la arquitectura. Existe una interacción entre los Casos de Uso y la arquitectura; los Casos de Uso deben encajar en la arquitectura cuando se llevan a cabo y la arquitectura debe permitir el desarrollo de todos los Casos de Uso requeridos, actualmente y en el futuro. Esto provoca que tanto arquitectura como Casos de Uso deban evolucionar en paralelo durante todo el proceso de desarrollo de software.

En la *Figura 3.5* se ilustra la evolución de la arquitectura durante las fases de RUP. Se tiene una arquitectura más robusta en las fases finales del proyecto. En las fases iniciales lo que se hace es ir consolidando la arquitectura por medio de *baselines*[5] y se va modificando dependiendo de las necesidades del proyecto.

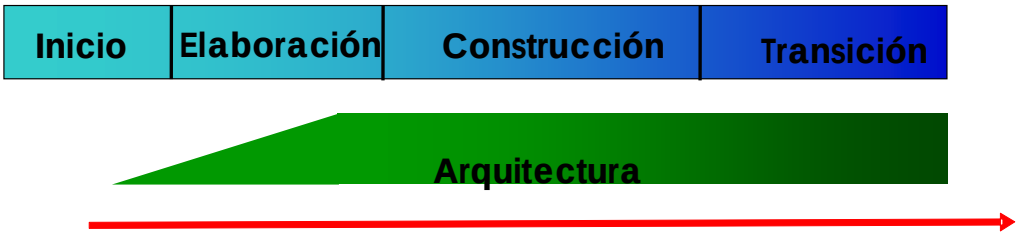


Figura 3.5 Evolución de la arquitectura del sistema. ***(Apuntes U.PV, 2001)***

- ✓ **Iterativo e incremental.**

Para hacer más manejable un proyecto RUP propone, dividirlo en ciclos, para conseguir el equilibrio correcto entre la forma y la función en el desarrollo del producto. Para cada ciclo se establecen fases de referencia, cada una se considera como un mini-proyecto, el cual estará constituido por una o más intenciones de las actividades principales básicas de cualquier proceso de desarrollo[6].

Es decir RUP propone tener un proceso iterativo e incremental en donde el trabajo se divide en partes más pequeñas o también denominados mini proyectos. Permitiendo que el equilibrio entre los Casos de Uso y arquitectura (forma y función del producto) se vaya logrando durante la realización de cada mini proyecto, así durante todo el proceso, cada uno de estos mini proyectos se puede ver como una iteración, el cual permite obtener un incremento que produce un crecimiento en el producto.

---

[5] Baseline es una foto instantánea del estado de todos los artefactos (generación de información) del proyecto, registrada para efectos de gestión de configuración y control de cambios.

[6] Existen diversos procesos de desarrollo, RUP es uno de ellos, por ejemplo: Proceso XP (Extreme Programming) para mayor información consultar:
http://en.wikipedia.org/wiki/Extreme_Programming

Pequeños proyectos que incorporan *incrementalmente* nueva funcionalidad y cuyo desarrollo es una *iteración*.

RUP, divide el proceso de desarrollo en cuatro fases (**iniciación, elaboración, construcción y transición**) dentro de las cuales se realizan un número variable de iteraciones según el proyecto, y en las que se hace hincapié en las distintitas actividades a desarrollar, como se puede observar en la *Figura 3.6.*

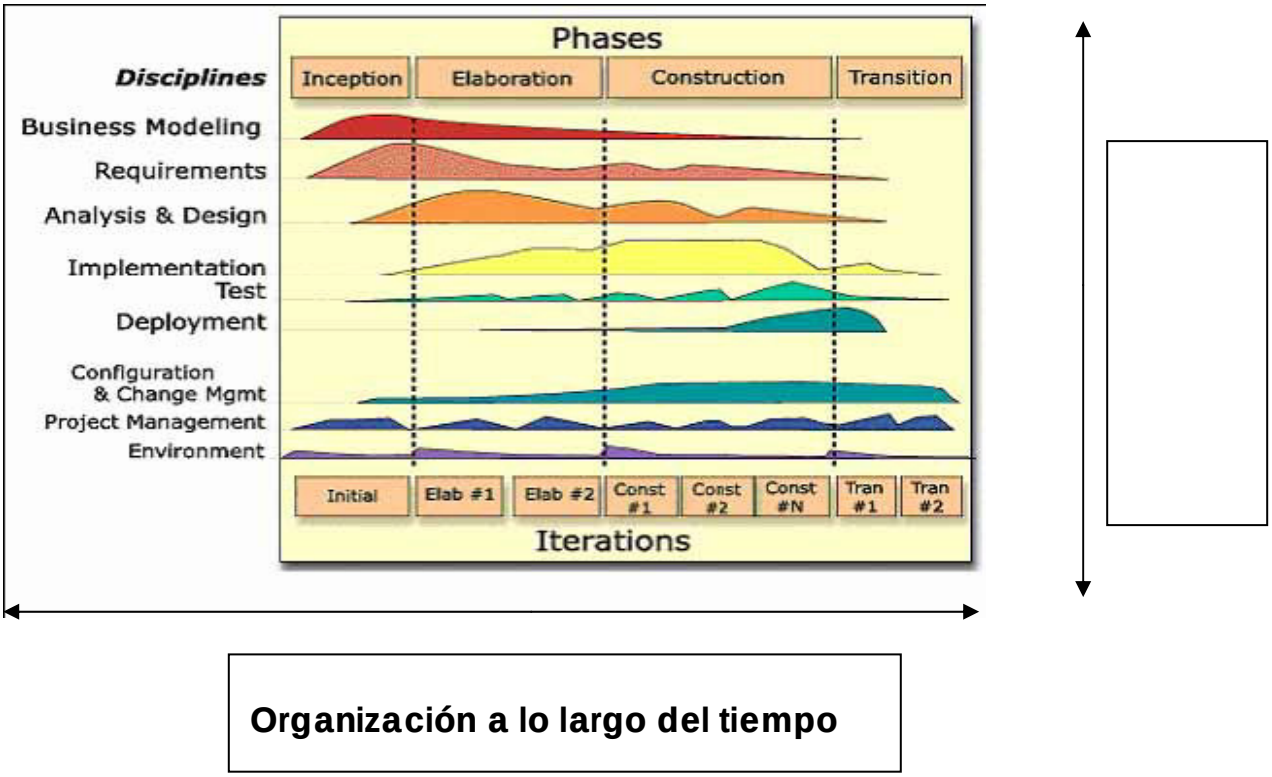


Figura 3.6 Principales ciclos de RUP con sus respectivos esfuerzos por fase. Fuente: IBM RUP Rational Unified Process® Versión 2002.05.00. Rational Software Corporation

Las primeras iteraciones (en las fases de Inicio y Elaboración) se enfocan hacia:

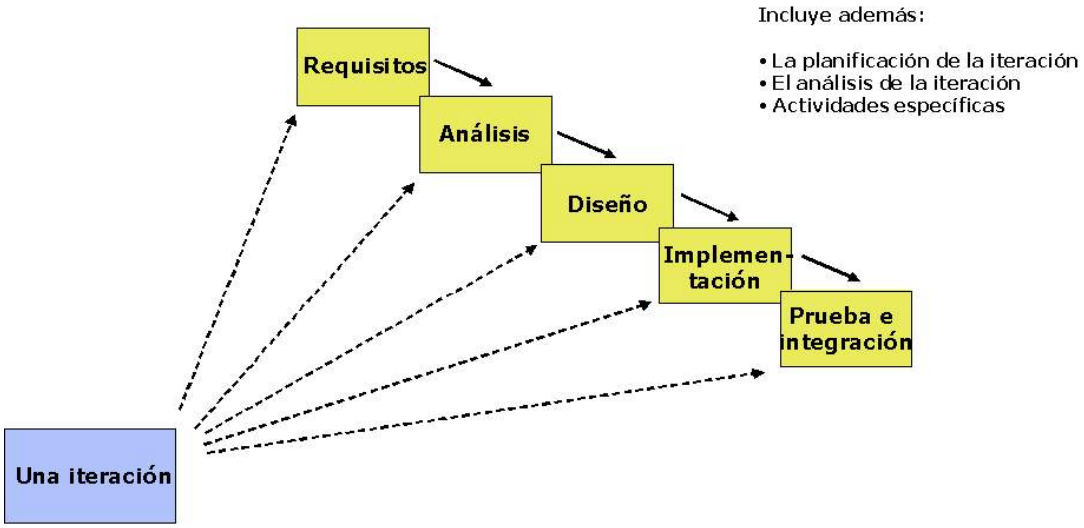
- ✓ La comprensión del problema y la tecnología.
- ✓ La delimitación del ámbito del proyecto.
- ✓ La eliminación de los riesgos críticos.
- ✓ Al establecimiento de una baseline de la arquitectura.

Durante la fase de inicio de cada una de las iteraciones se pone mayor énfasis en actividades de ***modelado del negocio*** y *de* ***requisitos***.

En la fase de elaboración, las iteraciones se orientan al desarrollo de la *baseline* de la arquitectura, abarcan más flujos de trabajo de requerimientos, refinamiento del Modelo de negocios, análisis de diseño y una parte de implementación orientado a la *baseline* de la arquitectura.

En la fase de construcción, se lleva a cabo la construcción del producto por medio de una serie de iteraciones.

Para cada iteración se selecciona algunos Casos de Uso, se refina su análisis y diseño, después se procede a su implementación y pruebas. Se realiza un *Modelo en cascada*[7] (ver *figura 3.7*) para cada ciclo. Se realizan tantas iteraciones hasta que se termine la implementación de la nueva versión del producto.



F*igura 3.7* Como se realiza una iteración siguiendo RUP *(ob cit. Apuntes U.V.)*

Este *Modelo en cascada* pasa por las 4 fases, antes mencionadas, en las que divide RUP el proyecto, también existen consideraciones que se toman en cuenta: la planificación de la iteración, un análisis de la iteración y algunas actividades específicas correspondientes a la iteración. Al finalizar se realiza una integración de los resultados, con lo obtenido de las iteraciones anteriores.

En la fase de transición se pretende garantizar que se tiene un producto preparado para su entrega a la comunidad de usuarios.

---

[7] Modelo en Cascada , es un modelo de desarrollo en el cual se basa en que se completa cada fase del proyecto y se continua con la siguiente fase con mayor detalles , para mayor información consultar: http://en.wikipedia.org/wiki/Waterfall_model

Para que estas ideas funcionen se necesita un proceso polifacético, que tenga en cuenta los ciclos, fases, flujos de trabajo, gestión del riesgo, control de calidad, gestión del proyecto y control de la configuración *(ob cit. Jacobson)*, por lo tanto RUP identifica 6 practicas para el desarrollo efectivo del software (*ver Tabla 3.1*) con las que define una forma efectiva de trabajar para los equipos de desarrollo.

Desarrollar Iterativamente el software
Administrar los requerimientos
Usar la arquitectura de componentes
Modelar visualmente
Verificar continuamente la calidad
Administrar los cambios

Tabla 3.1 Mejores prácticas en el desarrollo de software *(Krutchen, 2000*)

- ✓ **Administrar los requisitos**.

RUP nos da una útil guía para encontrar, documentar y seguir los cambios de los requerimientos funcionales y restricciones. En esta práctica se utilizan los Casos de Uso para representar los requisitos.

- ✓ **Verificación continúa de la calidad**.

La verificación de la calidad de todos los artefactos en varios puntos durante el proceso de desarrollo, especialmente al final de cada iteración, es importante debido a que juegan un papel fundamental y se integran a lo largo de todo el proceso. Para los artefactos no ejecutables las revisiones e inspecciones también deben ser continuas.

- ✓ **Desarrollo de software iterativo**.

El desarrollo del producto se hace por medio de iteraciones con ciclos o fases bien definidos en las cuales se repiten las actividades, pero con distinto énfasis, dependiendo de la fase del proyecto.

- ✓ **Desarrollo basado en componentes**.

Se necesita la creación de sistemas intensivos de software por eso se divide el sistema en componentes con interfaces bien definidas, que posteriormente serán ensamblados para generar el sistema. Esta característica es un proceso de desarrollo que permite que el sistema se vaya creando a medida que se obtienen o se desarrollan sus componentes.

- ✓ **Administración de los cambios**.

El cambio es un factor de riesgo crítico en los proyectos de software. Los artefactos de software cambian no sólo a acciones de mantenimiento posteriores a la entrega del producto, sino que durante el proceso de, desarrollo, los que son de mayor relevancia son los cambios en los requisitos.

Otro factor a considerar es la posible localización de desarrolladores en distintos lugares geográficos y hasta de plataformas, sin una adecuada gestión de cambios se podría crear un caos.

- ✓ **Modelado visual (usando UML).**

El uso de herramientas de modelado visual facilita la gestión de dichos modelos, permitiendo ocultar o exponer detalles cuando sea necesario. El modelado visual también ayuda a mantener la consistencia entre artefactos del sistema: requisitos, diseños e implementaciones.

Además, de estas buenas prácticas, RUP proporciona un Modelo de organización de personal, con tareas a realizar para cada uno de los que ocupan el puesto en la organización.

CARGOS/POSICIONES	TAREAS ASIGNADAS
Gestor del proyecto	Establecer Condiciones de Trabajo
Analista del sistema	Encontrar Actores y Casos de Uso Estructurar el Modelo de Casos de Uso
Arquitecto del sistema	Priorizar los Casos de Uso Efectuar el Análisis Arquitectural Efectuar el Diseño Arquitectural Efectuar la Implementación Arquitectural
Especificador de casos de uso	Detallar un Caso de Uso
Diseñador de interfaz de usuario	Prototipar una Interfaz de Usuario
Ingeniero de casos de uso	Analizar un Caso de Uso Diseñar un Caso de Uso
Ingeniero de componentes	Analizar una Clase Analizar un Paquete Diseñar una Clase Diseñar un Subsistema Implementar un Subsistema Implementar una Clase Realizar una Prueba de Unidad Implementar una Prueba
Integrador del sistema	Integrar el Sistema
Ingeniero de pruebas	Planear las Pruebas Diseñar las Pruebas Evaluar las Pruebas
Verificador de integración	Realizar una Prueba de Integración
Verificador del sistema	Realizar las Pruebas del Sistema

Tabla 3.2 Conformación del equipo de trabajo según RUP

En resumen, el RUP consta de:

- ✓ Dos etapas: **Ingeniería y Producción.**
- ✓ Cuatro fases: **Concepción, Elaboración, Construcción, y Transición.**
- ✓ Nueve flujos de trabajo o workflows denominados: *fundamentales* (detallados más adelante).

- ✓ 131 artefactos o productos.

- ✓ 136 actividades.

- ✓ Además de líneas directivas, listas de comprobación y manuales de herramientas.

Todos estos aspectos que conforman el proceso se resumen en los *10 elementos esenciales* a considerar los cuales se listan en la *Tabla 3.3*.

Visión	Visión de proyecto
Plan de proyecto	Administrase de acuerdo al plan de trabajo
Riesgos	Mitigar los riesgos y rastrear aspectos relacionados con ellos
Casos de Negocio	Examinar los casos de negocio.
Arquitectura	Diseñar una arquitectura de componentes.
Prototipos:	Construir de manera incremental y probar cada uno de los productos
Evaluación	Evaluar regularmente los resultados de cada iteración.
Cambio de requerimientos	Administrar los cambios y controlarlos.
Soporte al usuario	Implementar un producto usable
Proceso	Adoptar un proceso que se ajuste al proyecto.

Tabla 3.3 Aspectos Esenciales

Todos estos *elementos esenciales* a considerar dan la introducción a cada una de los *flujos de trabajo fundamentales de RUP* (antes mencionados), pertenecientes a las dos etapas que se divide RUP.

Flujos de trabajo correspondientes a la etapa de Ingeniería[8]:

- ✓ *Modelado de negocio.*
- ✓ *Requerimientos o requisitos.*
- ✓ *Análisis y Diseño del proyecto*, *(Castro, 2004).*
- ✓

Flujos de trabajo pertenecientes a la etapa de producción:

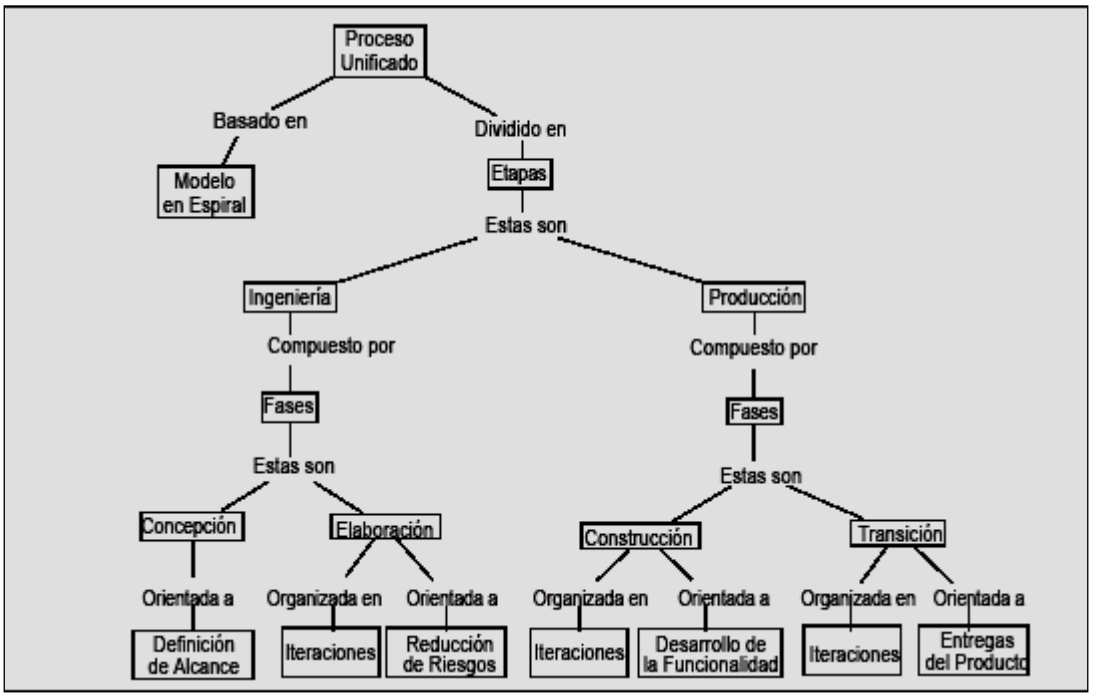
- ✓ Implementación
- ✓ Pruebas
- ✓ Despliegue
- ✓ Administración del proyecto. (*)
- ✓ Configuración y Administración de Cambios. (*)
- ✓ Ambiente o Entorno. (*)

(*) Según *(Krutchen,2001)* estas etapas corresponden a los denominados : *flujos de trabajo de apoyo*.

8 *La Etapa de Ingeniería* agrupa las fases de *Concepción y Elaboración*, lo que básicamente le da por objetivos la conceptualización del sistema y el diseño inicial de la solución del problema.

Podemos entender mejor la composición del proceso a partir de este diagrama conceptual mostrado en la *Figura 3.8.*

Figura 3.8



Estructura básica del RUP *(ob. cit. Castro)*

Estos flujos de trabajo de RUP proporcionan una vista de los elementos más significativos de los procesos o las 4 fases de RUP y, a su vez, describen un conjunto de actividades y artefactos.

RUP detalla cada una de los flujos de trabajo desde un nivel de visión general, otorga un resumen de roles, actividades y artefactos que requieren un conjunto dado de habilidades para desempeñar las disciplinas (también llamados así a los flujos de trabajo); pero a un nivel más detallado, dichas disciplinas presentan como colaboran los roles de manera interdisciplinaria para alcanzar los objetivos propuestos y deseados, a este trabajo desempeñado se le conoce como *detalles del flujo de trabajo*.

Al estudiar RUP dentro de su parte de Ingeniería e inmersa en las dos primeras etapas del proceso: *Concepción y Elaboración* el flujo de trabajo denominado: ***Analisis y* Diseño** (ver *Figura 3.9*) los objetivos son:

- ✓ *Transformar los requisitos al diseño del futuro sistema.*
- ✓ *Desarrollar una arquitectura para el sistema.*
- ✓ *Adaptar el diseño para que sea consistente con el entorno de implementación, diseñando para el rendimiento, (ob.cit. Jacob*

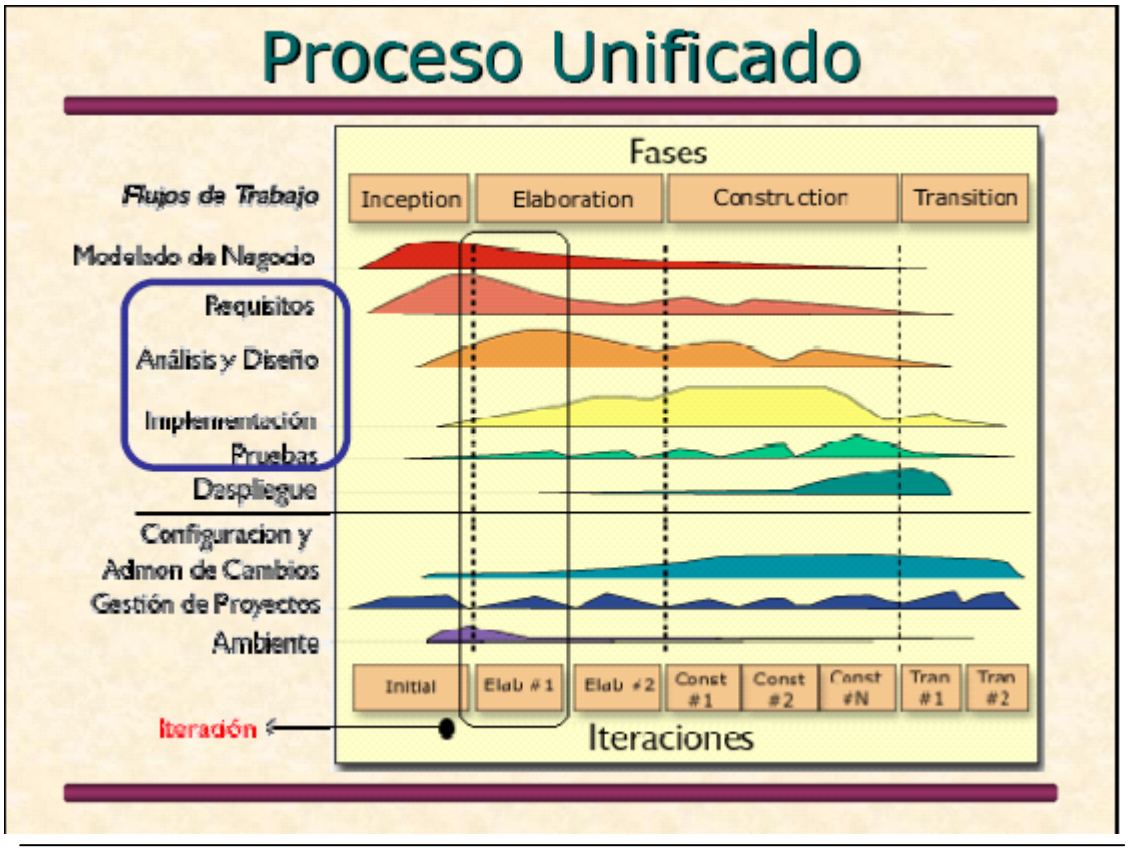


Figura 3.9 Flujos de trabajo de RUP, mostrando flujo de interés dentro de la vida de desarrollo

El análisis consiste en obtener una visión del sistema que se preocupa de ver qué hace, de modo que sólo se interesa por los requisitos funcionales. Por otro lado el diseño es un refinamiento del análisis que tiene en cuenta los requisitos no funcionales, en definitiva cómo cumple el sistema sus objetivos.

Al principio de la fase de elaboración hay que definir una arquitectura candidata: crear un esquema inicial de la arquitectura del sistema, identificar clases de análisis y actualizar las realizaciones de los Casos de Uso con las iteracciones de las clases de análisis.

Durante la fase de elaboración se va refinando esta arquitectura hasta llegar a su forma definitiva. En cada iteración hay que analizar el comportamiento para diseñar componentes. Además si el sistema usará una ***base de datos***, habrá que diseñarla también, obteniendo un ***Modelo de Datos***.

El resultado final más importante de este flujo de trabajo será el Modelo de diseño. *El cual consiste en colaboraciones de clases, que pueden ser agregadas en paquetes y subsistemas. Además de la elaboración de la documentación de la arquitectura de software, que captura varias vistas arquitectónicas del sistema, (ob cit Martinez*).

Aquí el por qué se centra nuestro estudio en esta disciplina ya que RUP especifica la necesidad de diseñar una base de datos como parte del proyecto de desarrollo de software.

Las bases de datos en los sistemas automatizados constituyen el núcleo de la misma pues allí reside lo más valioso del sistema, la información; ésta debe encontrase organizada y es manipulada por un conjunto de programas que darán acceso en tiempo real a usuarios concurrentes con diferentes necesidades de información que la solicitarán en tiempo real.

3.2 Modelos que conforman la actividad de diseño de la base de datos

El principal objetivo de un sistema de base de datos o base de datos es proporcionar a los usuarios finales una visión abstracta de los datos, esto se logra escondiendo ciertos detalles de cómo se almacenan y mantienen los datos. Esto es la principal directriz que tiene el diseño de una base de datos es por eso que para realizar un diseño satisfactorio y completo RUP propone completar los siguientes 3 modelos:

1) Modelo Conceptual.
2) Modelo Lógico.
3) Modelo Físico.
   Arquitectura.

Cada uno de estos modelos están relacionados con los niveles de *abstracción de la base de datos*, las cuales se presentan en *la Figura 3.10*.

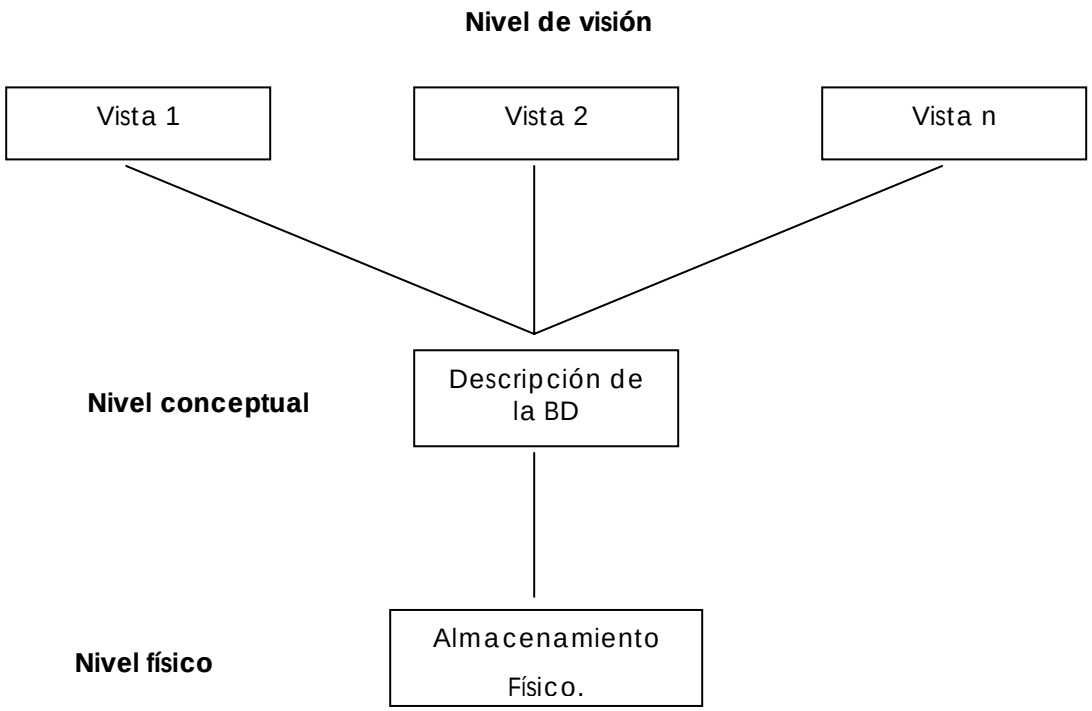


Figura 3.10 Niveles de abstracción de una base de datos.

Nivel Físico: Se detalla donde se guardarán los datos en los dispositivos de almacenamiento, tales como: memoria cache, memoria principal, memoria flash, almacenamiento en discos magnéticos o almacenamiento óptico (CD-DVD).

Nivel Conceptual: Hace una descripción de la base de datos (estructura del diseño) además de definir el almacenamiento real de los datos y las relaciones entre las entidades que conforman a la base.

Nivel de Visión: Es lo que el usuario final puede visualizar del sistema terminado.

Estos diferentes modelos tienen que cumplir con el objetivo de sistema de base de datos los cuales se centran en reducir:

- ✓ Redundancia e inconsistencia en los datos.
- ✓ Dificultad para tener acceso a los datos.
- ✓ Aislamiento de los datos.
- ✓ Anomalías del acceso concurrente.
- ✓ Problemas de seguridad.
- ✓ Problemas de integridad.
- ✓ Problemas de abstracción del los datos.

Analizaremos los diferentes modelos que propone RUP para el desarrollo de una base de datos, para determinar los parámetros de esfuerzo relativos al diseño de la base, necesarios para la construcción del apartado que titularemos: **Esfuerzo de Desarrollo de una Base de Datos (EDBD)**, el cual integraremos con los demás parámetros de esfuerzo que propone la expresión matemática que permite el cálculo de la estimación de esfuerzo PM_{NS}.

Es necesario comentar que, los parámetros de esfuerzo que se propondrán después del análisis que hacemos en este trabajo de tesis son definidos obedeciendo a que deben ser parámetros que puedan ser medidos de manera cuantitativa, además de que si se hace otro análisis más profundo se puede encontrar otros y posiblemente más refinados y concisos, todo depende del tipo de proyecto que necesite el uso de una base de datos para su funcionamiento; además hasta el tipo de base de datos
(relacional , orientada a objetos o híbrida) que se implante tiene repercusiones en el esfuerzo de desarrollo de un proyecto de software.

- **Modelo Conceptual**

Este modelo captura todos los requerimientos o requisitos del sistema, los cuales expresan que se supone debe hacer una aplicación: por lo común no intentan expresar cómo lograr tales funciones.

El objetivo de este modelo es la captura de los requerimientos del sistema, establecer cuáles son las entidades base del sistema o entidades principales y sus relaciones o asociaciones, así como las fronteras del sistema.

El *Modelo Conceptual* describe el conjunto de la estructura lógica de un sistema de base de datos, dicho modelo hace caso omiso a todas las restricciones tecnológicas que se pueden presentar; es decir, no hace caso a alguna arquitectura aún.

En este modelo hay un mayor esfuerzo en tratar de especificar los requerimientos, además de su entendimiento y asimilación por parte del equipo de desarrollo, de tal manera que la base de datos tenga un diseño apegado a lo que necesita el usuario final (ya sea humano o máquina).

Para obtener el *Modelo Conceptual* podemos valernos de los Casos de Uso, ya que es una manera de captura y representación de los requerimientos funcionales[9] del sistema; además de un análisis; exhaustivo de los requisitos por medio de la propuesta de RUP (obtención del Modelo de negocio).

Con lo antes dicho destacamos que en el *Modelo Conceptual* no hay procedimientos o metodologías que hagan la captura de los requerimientos de todo sistema de manera sistemática, si no que existen herramientas que nos pueden ayudar a comprender y analizar los requerimientos (tales como el uso de los Casos de Uso, Diagramas Jerárquicos, Diagramas de Flujo, Inteligencia Artificial[10], etc.).

[9] Los requerimientos funcionales especifican los servicios que debe proporcionar las aplicaciones. (ob. cit. Braude)

[10] Véase articulo relacionado con la utilización de la inteligencia artificial para el estudio de los requisitos en la siguiente pagina: http://www.inf.udec.cl/revista/ediciones/edicion8/Rbc.pdf

Este modelo es crucial para el desarrollo de una base ya que si no se comprende con cabalidad cada requisito que nos dicta nuestro cliente, tendremos un mal diseño e implementación de la base de datos, esto se convierte en el no cumplimiento de los objetivos del sistema, además de la no satisfacción de las expectativas de los usuarios y difícilmente aporte al valor agregado al negocio para el que debe ser concebido.

Para prevenir tal situación, se sugiere una amplia comunicación del equipo de desarrollo con los clientes (ya sean entre las mismas personas del equipo de desarrollo o ajenos a éste).

Otro punto a considerar dentro del diseño de la base de datos de manera conceptual, es la disposición de la información para el desarrollo del sistema, ya que no obstante, las personas no son la única fuente de requerimientos, esto quiere decir que, el tipo de información depende de las características del sistema o aplicación en sí, por qué no es lo mismo manejar datos relacionados con aspectos académicos que con cuestiones militares. Podemos observar una gráfica que propone Bracket en el libro *(Braude, 2001)* que clasifica el nivel de disposición de información, dependiendo del tipo de aplicación o sistema y el porcentaje de requerimientos obtenidos de personas (*Ver Figura 3.11*).

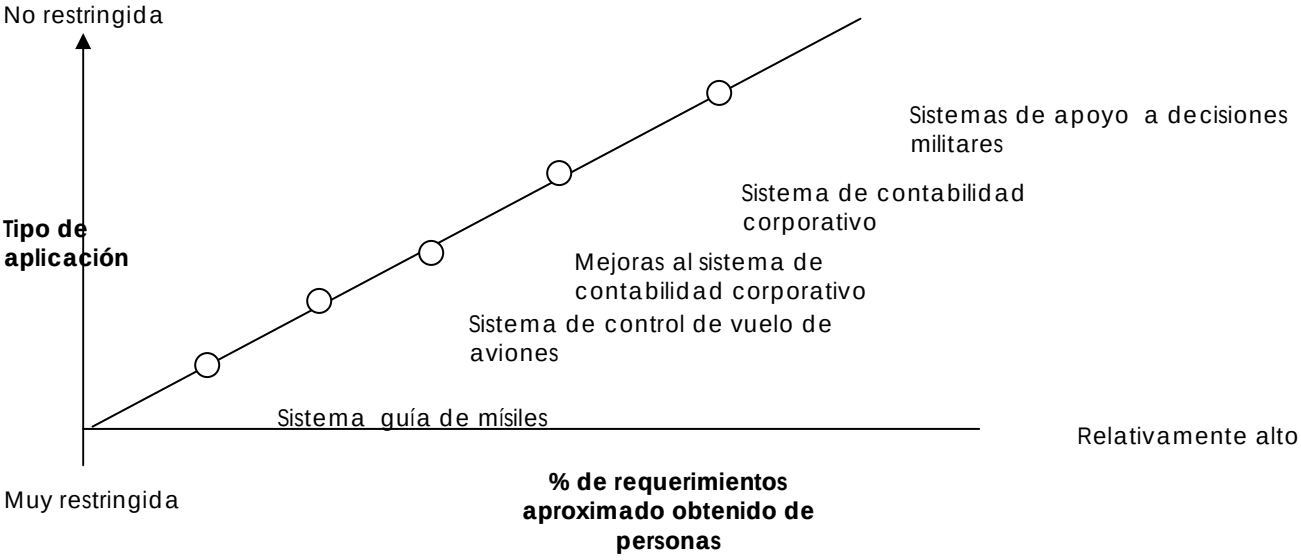


Figura 3.11 Fuentes de requerimientos: personas contra otras (ob. cit Braude)

En la *Figura 3.11* clasifica las aplicaciones en términos del grado de restricción, esto se refiere a las restricciones sobre la aplicación que no se puede alterar. *Cuanto menos restringido esté el problema más requerimientos se obtienen de las personas*, *(ob. cit Braude)*.

- **Modelo Lógico**

Establece el diseño general del sistema, sin considerar la tecnología que está asociada a éste, refina el Modelo conceptual haciéndolo más preciso.

El Modelo lógico se obtiene a partir del Modelo conceptual, pero éste se va refinando en aspectos tales como:

- ✓ Agrupación de los objetos (polimorfismo).
- ✓ Cantidad de información.
- ✓ Calidad de la información.
- ✓ Relaciones entre la información.

Este modelo representa los flujos de información del sistema, las entidades que manipulan la información y las relaciones entre ellas.

De acuerdo con RUP para construir el modelo se debe hacer:

- ✓ *La construcción de un Modelo de objetos, el que estará representado por un diagrama de clase (tal como el diagrama entidad-relación, la cual es una representación gráfica de la estructura lógica de una base de datos[11]).*

- ✓ *La identificación de atributos, operaciones y asociaciones en el diagrama de entidades.*

- ✓ *Transformar el Modelo de objetos en un Modelo de Datos[12] (similar a la reducción de el diagrama entidad-relación a una representación de datos por medio de tablas) (Shelberschatz 2002) (ob. cit Jacobson)*.

[11] En principio se puede asegurar que el modelo de objetos puede ser representado por el diagrama entidad-relación, pero teniendo en cuenta que podría existir clases con métodos u operaciones.

[12] Bajo la estructura de la base de datos se encuentra el modelo de datos, el cual es una colección de herramientas conceptuales para describir datos. Ejemplos de modelos de datos: Modelo entidad-relación, Modelo relacional, Modelo de datos relacional orientado a objetos, Modelo de datos orientado a objetos, etc.

Se puede observar que para llevar a cabo la construcción del Modelo de Objetos en su mayor parte, depende del Modelo de Diseño de Objetos (diagrama de clases). Los elementos del Modelo de Objetos son correspondientes al Modelo de Datos. Esto se puede extrapolar a que hay correspondencia entre el Diagrama Entidad-Relación y el Modelo Relacional, donde efectivamente los elementos más importantes de acuerdo con la perspectiva de modelado de datos son:

- ✓ Clases.
- ✓ Asociaciones.
- ✓ Generalización.
- ✓ Operaciones.

Donde el Modelo de Objetos comprende clases y sus asociaciones. Estas clases definen la estructura y comportamiento de un conjunto de objetos, los cuales son una instancia de las clases. Esta estructura está representada por los valores de los datos, denominados atributos y las relaciones entre las clases también llamadas asociaciones.

Durante el final del proceso del Modelo Lógico se obtiene el Modelo de Datos. Éste es usado para describir la estructura lógica y física de la información persistente administrada por el sistema.

Este Modelo de Datos puede ser creado inicialmente a través de *ingeniería inversa;* es decir, que se puede obtener a partir de un Modelo Relacional, pero se considera como *ingeniería inversa,* ya que primero se debió haber construido el Modelo de Datos y después el relacional.

Al momento de seguir las cuatro actividades que RUP propone para el diseño de la base de datos, el Modelo de Datos es constantemente transformado; es decir, que es usado para definir las estructuras persistentes de datos a través de las clases persistentes de diseño.

El diseñador de la base de datos debe identificar las clases persistentes del Modelo de Diseño, son estas mismas además de sus atributos y asociaciones los que darán forma al Modelo de Datos, por lo cual a partir de esta identificación se puede decir que el Modelo de Objetos, representado por el Diagrama de Clases, se transforma en un esquema del Modelo de Datos.

- **Modelo Físico**

Dentro del Modelo Físico se establece el *diseño físico* de la base de datos y comienza a incluir especificaciones tecnológicas asociadas a éste.

El objetivo del Modelo físico es definir cómo se implementara la base de datos, definir las tablas y las restricciones detalladas, tal y como se consideraron en el Modelo de Diseño tomando en cuenta la tecnología de almacenamiento, el DBMS, la arquitectura, las reglas de implementación, etc. El Modelo Físico redefine el Modelo de Datos obtenido en el Modelo de Diseño de acuerdo con restricciones tecnológicas.

Los pasos concretos para llevar a cabo el modelo son:

- ✓ Establecer el diseño físico de la base de datos.
- ✓ Orientar el diseño a la tecnología específica seleccionada.
- ✓ Dar detalle al diseño físico.
- ✓ Construir físicamente la base de datos.

Este modelo se basa en aspectos relacionados con hardware, estructuras y tipos de datos.

Se puede proceder de la siguiente manera para la construcción de este modelo: una vez teniendo el *Modelo Relacional* o en forma más general el *Modelo de Datos*, puede ser modificado hasta detallar las condiciones específicas del modelo propias del diseño físico.

Para llevar a cabo el Modelo Físico es necesario tener cubiertas las siguientes actividades:

- ✓ Normalizar las tablas obtenidas del Modelo lógico.
- ✓ Considerar restricciones de desempeño.
- ✓ Especificar los índices de desempeño.

Estos tres puntos a tratar se deben llevar a cabo tomando en cuenta los requerimientos de almacenamiento.

En este modelo se debe considerar el aspecto de la seguridad de la información, debido a que es un recurso valioso para la organización y debe ser administrada adecuadamente.

La seguridad implica garantizar que los usuarios están autorizados para llevar a cabo lo que tratan de hacer sobre la base de datos y por otro lado, la integridad de los datos implica asegurar que lo que tratan de hacer los usuarios es correcto. Por lo tanto, la seguridad e integridad se, especifican mediante restricciones de distintos tipos. *(Pastor, 2000)*.

Estos tres modelos, si se observa obedecen a un desarrollo de tipo cascada donde se va refinado cada uno de los modelos entrantes, pero en algunos modelos no hay una forma metodologica clara para su construcción, por ende, se utilizan diversas herramientas y modelos que ayudan a su construcción.

- **Arquitectura Física**

No se puso como modelo, por una sencilla razón: no es un modelo como tal, entendiéndose como modelo un mini-proyecto, si no que es una actividad que debe estar presente en los tres modelos y debe evolucionar junto con estos según RUP. Pero se toma en cuenta, ya que este aspecto es importante para un diseño completo y exitoso de una base de datos.

La Arquitectura se concentra en definir los procesadores, su conectividad y su distribución. Describe la topología física de las aplicaciones y de las bases de datos, también la configuración de ejecución.

Concretamente, en la Arquitectura Física se debe especificar todos los aspectos técnicos tales como nodos de la red en la que se ejecutará el sistema, sus configuraciones y capacidades, el ancho de banda de cada segmento de la red, la capa física de la red en la que opera el sistema, etc.

3.3 Evaluación del desarrollo de la base de datos por modelos

En este apartado, estableceremos los parámetros de evaluación del desarrollo de una base de datos, utilizando RUP con el propósito principal de reducir el esfuerzo invertido en el diseño y construcción de cualquier base de datos que forme parte de un sistema que se encuentre en desarrollo. Una vez obtenidos estos parámetros se definirá el apartado mencionado con anterioridad denominado: ***Esfuerzo de Desarrollo de una Base de Datos (EDBD)***.

La obtención de los parámetros de esfuerzo relacionados con la base de datos, se hará por medio de un cuestionario, el cual tratará de recopilar la información necesaria para realizar las estimaciones correspondientes. Dicho cuestionario está diseñado para solo encontrar la información de manera objetiva, además que corresponda con la definición de los parámetros de esfuerzo del modelo de estimación de COCOMO II, cuyo propósito es capturar características de desarrollo del software que afectan al esfuerzo para completar el proyecto, (Ver ***Capítulo 2***).

Los parámetros obtenidos (**EDBD**) estarán regidos por la evaluación que COCOMO II define para los parámetros de esfuerzo que denominaremos: *parámetros originales* correspondientes a la ecuación de estimación de esfuerzo PM_{NS} **(2.1).** Éstos se evalúan de acuerdo con el siguiente criterio de **Extra Alto** hasta **Extra Bajo** para el propósito de tener un análisis cuantitativo.

A cada uno de estos parámetros **EDBD** se les debe asociar un peso el cual expresa el impacto del parámetro en el esfuerzo de desarrollo. Este peso o multiplicador se denomina ***Multiplicador de Esfuerzo (EM)***. El EM tiene asignado un valor nominal (1.0), el cual si este EM>1 se traduce en un aumento de la cantidad de esfuerzo en este parámetro, de manera recíproca se puede concluir que la cantidad de esfuerzo es menor. La calibración de cada EM propuesto para el desarrollo de la base se discute más adelante en este capítulo.

3.3.1Evaluación del desarrollo de la base de datos

Para evaluar el proceso de desarrollo de una base de datos que siga RUP, primero se necesita conocer si se ha seguido el proceso de RUP de manera total, parcial o definitivamente no se ha considerado el proceso. La evaluación la haremos en cada uno de los tres modelos de desarrollo de la base de datos.

Esta evaluación es hecha por el encargado de realizar las estimaciones de esfuerzo que implica el desarrollo de una base de datos y es aplicada al equipo de desarrollo de la base, en concreto al Arquitecto y Analista del sistema (según RUP *Ver tabla 3.2*)

- **Evaluación del Modelo conceptual**

1) ¿Se ha seguido el RUP como guía de desarrollo del sistema?

Es necesario conocer primero si se ha llevado RUP ya que esto dará una idea al analista de proyectos o al estimador, cuál será el esfuerzo que se deberá invertir si se ha seguido RUP desde el inicio del desarrollo del sistema, por el contrario si no se ha llevado el proceso desde el inicio del proyecto, nos dicta la necesidad de desarrollar los mecanismos que utiliza RUP para construir el Modelo de datos.

2) ¿Cómo calificaría la comunicación entre el equipo de desarrollo de la base de datos y el equipo de desarrollo de aplicaciones?

La comunicación entre clientes y desarrolladores es un principio imperativo en el desarrollo por eso RUP se basa en el análisis continuo de requerimientos, por medio de la interacción cliente-desarrolladores. Lo mismo sucede con el personal dentro de la organización que está encargada del proyecto. Debe existir comunicación dentro del equipo de desarrollo del proyecto. El equipo encargado del desarrollo de aplicaciones y el personal encargado de la base de datos deben tener una comunicación coherente y consistente en todas las actividades.

La respuesta a esta pregunta nos da un resultado parecido como el factor de escala TEAM que califica la cohesión entre los miembros del equipo de desarrollo. (Ver ***Capítulo 2***).

3) ¿Cómo considera la disposición de información para los requerimientos de la base de datos?

Se plantea el aspecto del acceso y disposición de la información para la obtención de requerimientos el cual depende mucho del tipo de aplicación que estemos desarrollando.

Dependiendo del grado de acceso a los requerimientos mejor resulta el diseño de la base de datos.

4) ¿En qué grado se ha analizado y construido el Modelo de Negocios?

Esta pregunta es clave debido a que a falta de una representación de los procesos que va a automatizar el sistema y un esbozo de una estructura de este mismo, no se puede tener un Modelo de Casos de Uso, donde se detalla de manera más concisa los escenarios que acatara el sistema.

5) ¿En qué grado se ha analizado y construido el Modelo de Casos de Uso?

La respuesta de esta pregunta trata de evaluar el análisis de cada uno de los Casos de Uso, éstos reflejan los requerimientos que debe cubrir el diseño de la base de datos; además este análisis nos ayudará a determinar los PF que serán utilizados al aplicar COCOMO II

Los diferentes Casos de Uso, es una forma de representación de los requerimientos o requisitos que nos exige el usuario, tal forma de descripción de un sistema, está dirigido a dos niveles uno es el que ve el usuario y el otro a nivel diseñador.

6) ¿En qué grado se ha completado el Modelo de Diseño de la base de datos?

Se considera esta respuesta debido a que si no se está siguiendo RUP durante las etapas de desarrollo del proyecto, seguramente se ha considerado este modelo.

Además de que la elaboración del Modelo de datos depende de los diagrama de clases que pertenecen al Modelo de diseño.

- **Evaluación del Modelo lógico**

El Modelo Lógico, se obtiene a partir del Modelo de Casos de Uso y el Modelo de Negocio. En esta etapa se van refinando algunos aspectos como: agrupación de objetos, relaciones de la información, calidad de la información y cantidad de información

7) ¿En qué grado se ha construido el diagrama de clases?

Esta pregunta va enfocada a cuestiones de diseño relacionadas a una base de datos, y es la forma de identificar y representar entidades, atributos, relaciones entre éstas y los métodos que se utilizan para leer, modificar o actualizar los datos que contengan los atributos; esto nos lleva a los modelos que ayudan a modelar el componente de representación de datos de un sistema, tales como: el diagrama de clase (estos diagramas son similares al diagrama entidad-relación[13]).

El diagrama de clase es un componente de UML al igual que los diagramas de los Casos de Uso que utiliza el Modelo de Casos de Uso.

[13] La diferencia entre el diagrama entidad-relación y el diagrama de clases de UML es que UML modela realmente objetos, mientras que E-R modela entidades.

La respuesta de esta pregunta trata de indagar si las diferentes clases que se han identificado son lo más apegados a los requisitos del sistema (a la forma de modelar la empresa) y el esfuerzo que lleva diseñarlos y construirlos. Parte fundamental para el diseño Modelo Lógico. Y representa al Modelo de Objetos, que servirá para la construcción del Modelo de Datos, perteneciente ya al Modelo Físico.

8) El diseño del Modelo de Objetos, captura las clases, atributos, relaciones, interacciones que conforman el sistema de manera: (eficiente, no eficiente, deficiente)

Una vez obtenido un diagrama de clases, se cuenta con la representación gráfica del Modelo de Objetos, por ende, se necesita evaluar si este modelo captura y representa de manera correcta la especificación de las clases , sus relaciones y atributos; ya que si no está bien diseñado el Modelo de Objetos , tampoco lo estará el Modelo de Datos.

9) ¿Se han encontrado todas las clases, atributos y relaciones pertenecientes al Modelo de Objetos?

La obtención del Modelo de Objetos, es una parte importante de la construcción del Modelo Lógico y de los sucesivos modelos, pero se necesitan identificar las clases persistentes para determinar las tablas de la base de datos, por lo tanto la información que se tenga en un renglón de la tabla corresponde a un objeto persistente.

Esto nos podría dar una estimación de que tamaño tendrá la base de datos, de acuerdo al número de clases persistentes se hayan identificado.

10) ¿Cómo considera el nivel de complejidad de las relaciones pertenecientes a las relaciones persistentes?

Después de haber identificado las diferentes clases persistentes en nuestro análisis del Modelo de Objetos, tendremos que evaluar las relaciones que hay entre cada una de estas clases, de acuerdo con su nivel de complejidad, obedeciendo a las diferentes cardinalidades y tipos de relaciones.

- **Evaluación del Modelo físico**

11) ¿Qué grado de complejidad tiene las consultas a la base?

Dentro de la evaluación del Modelo físico podemos encontrar aspectos importantes a tomar en cuenta, para la implementación física de la base de datos, para llevar a cabo el Modelo Físico, es necesario normalizar las tablas obtenidas del Modelo Lógico, esto es un punto importante en la implementación, ya que al aplicar las diferentes reglas de normalización[14] a las tablas, se evita la redundancia de datos y/o pérdida.

12) ¿Cómo definiría la complejidad que tienen las restricciones en el lenguaje anfitrión para asegurar la integridad de la base de datos?

13) ¿Cómo considera el grado la seguridad en la base de datos?

Estas dos preguntas tratan de evaluar la seguridad y la integridad las cuales son partes que se definen desde el Modelo de Casos de Uso y además hay algunas reglas implícitas en el diseño que garantizan una seguridad e integridad dependiendo del SGBD, pero según las características del sistema, es probable que se necesiten otros tipos de mecanismos de seguridad para preservar los datos contenidos en la base batos contra operaciones no admitidas. Por ejemplo, la necesidad de cifrar cada uno de los valores de las tablas, es indispensable si se trata de una base de datos que esté en una arquitectura cliente servidor, pero si dentro de los requerimientos se pide una seguridad de estos registros, más exigente, se traduce en encontrar un algoritmo más complejo para el cifrado que se traduce en esfuerzo de diseño implementación importante.

3.4 Multiplicadores de esfuerzo de una base de datos (EDBD)

La principal finalidad de las preguntas antes planteadas, es indagar el esfuerzo aproximado que tomará diseñar y construir una base de datos, basándose en las actividades que plantea RUP. Cada una de las preguntas de la evaluación de los diferentes modelos, plantea un multiplicador de esfuerzo importante a considerar, que expresará una disminución o aumento de el esfuerzo, del diseño y construcción de una base de datos. Los multiplicadores de esfuerzo obtenidos anteriormente, denominados ***Esfuerzo de Desarrollo de una Base de Datos (EDBD)_*** se listan a continuación en la *tabla 3.4.*

[14] Formas normales: Primera Forma Normal. Segunda, Tercera, Cuarta y Boyce-Codd , son las más utilizadas en el proceso de diseño de un base de datos relacional, existen excepciones sustanciales, en cuanto normalizar una base de datos , cuando se está implementando una base de datos orientada a objetos o una base de datos relacional y orientada a objetos.

El orden de los multiplicadores fue considerado debido a la experiencia de desarrollo de una base de datos.

Multiplicador	Descripción de los EDBD propuestos
Modelo conceptual	
1) DRUP	Desarrollo de RUP (DRUP), captura en qué grado se sigue el proceso unificado en el desarrollo de una BD.
2) CEBD	Comunicación entre Equipo de Desarrollo(CEBD) , indaga si la comunicación entre el equipo de desarrollo del proyecto es fluida o no.
3) DINFO	Disposición de la Información (DINFO), investiga qué grado de disposición de información se tiene en la captura del los requerimientos de la base.
4)MODNG	Modelo de Negocio (MODNG), busca si se ha concluido el Modelo de negocios y en qué medida está completo.
5) MODCU	Modelo de Casos de Uso (MODCU), trata de obtener si el análisis de los Casos de Uso es correcto y completo.
6) MODD	Modelo de diseño (MODD) busca el grado de desarrollo del Modelo de Diseño.
Modelo lógico	
7) CDIGC	Construcción del Diagrama de Clases (CDIGC) , investiga si el Diagrama de Clases ya se ha terminado tal diagrama que representa el Modelo de Objetos .
8) DMOO	Diseño del Modelo de Objetos (DMOO), proporciona la información necesaria para saber en qué grado de análisis y desarrollo se encuentra el Modelo de Objetos.
9) CLPMOO	Clases Persistentes del Modelo de Objetos (CLPMO), después de haber hecho el análisis del Modelo de Objetos, el resultado es la identificación de las Clases Persistentes, las cuales serán las tablas pertenecientes a la BD.
10) NCLPXR	Nivel de Complejidad de las Relaciones (NCLPX), trata de evaluar la complejidad que puede existir entre las relaciones de las clases persistentes.

Modelo físico.	
11) CPLXC	Complejidad de las Consultas (CLPXC), investiga la necesidad de saber cuál será la complejidad de las consultas de la base de datos, la cual nos dará una idea del esfuerzo de la normalización de las tablas de la base de datos.
12)CLPXRI	Complejidad de las Relaciones para la Integridad (CPLXRI), analiza el grado de dificultad que puede estar asociado a la creación de reglas que garantizan la integridad de la base de datos.
13)SEGBD	Seguridad de la Base de Datos (SEGBD) hace el análisis del esfuerzo que conlleva la creación de reglas que nos garanticen la seguridad de la base de datos contra posibles operaciones no válidas, dentro de la base de datos.

Tabla 3.4 Multiplicadores de Esfuerzo de Desarrollo de una Base de Datos (EDBD)

Una vez obtenidas los **EDBD** de la evaluación propuesta a los modelos que propone RUP, ahora se detalla su comportamiento de cada uno de los multiplicadores, de acuerdo al planteamiento de los multiplicadores originales de COCOMO II ,siguiendo la escala que propone el modelo:

- ✓ HIGH si la actividad a realizar esta causa más esfuerzo del debido, cuyo valor multiplicativo es mayor al nominal. HIGH>1.0.

- ✓ NOM es el valor nominal (no causa ningún esfuerzo extra al desarrollar la actividad) NOM=1.0.

- ✓ LOW si la actividad a realizar causa menos esfuerzo del debido, cuyo valor multiplicativo es menor al nominal. NOM<1.0.

A continuación se lista el planteamiento del comportamiento de cada uno de los **EDBD**:

- ✓ **EDBD pertenecientes al Modelo Conceptual**

1) DRUP (Desarrollo de RUP)

Este multiplicador deberá tener el rango de valores HIGH no se ha seguido el RUP, por lo tanto evalúa el esfuerzo que conlleva ajustar actividades y procesos de desarrollo al proceso unificado, si es LOW se ha llevado RUP, por lo tanto no causa mayor esfuerzo a la actividad.

DRUP se considera de muy alta prioridad, debido a que todo el desarrollo de una base de datos, debe de seguir la Actividad de Diseño de una Base de Datos.

2) CEBD (Comunicación de Equipo de la Base de Datos)

Este parámetro es parecido con TEAM de COCOMO II, pero la diferencia radica en que este multiplicador trata de evaluar la comunicación entre el equipo de desarrollo de la base de datos y el equipo de aplicaciones. Los valores van de desde LOW que mide una amplia cohesión del personal, por lo tanto no hay problemas de comunicación entre ambos equipos, NOM establece sin problemas de cohesión y HIGH hay un problema serio entre el personal, comunicación muy complicada.

3) DINFO (Disposición de la información)

Multiplicador que investiga el acceso a la información para la comprensión y desarrollo de los requerimientos que necesita acatar la base de datos. Se mide en HIGH si el acceso a información es demasiado complicado y hay un esfuerzo mayor para el entendimiento de los requisitos y posteriormente su desarrollo, NOM hay acceso suficiente a la información por lo tanto puede llegarse a una comprensión de los requisitos muy confiable, y LOW hay total disposición, por lo tanto no hay problemas de comprensión y desarrollo de los requerimientos de la base de datos.

4) MODNG (Modelo de Negocios)

Dentro del Modelo de negocios se verifican todos los requerimientos, y además se da una aproximación de la estructura del sistema. Los valores definidos para este parámetro de esfuerzo es HIGH: construcción nula del modelo, LOW: construcción óptima y confiable del modelo, captura todos los requisitos, esbozos de los flujos de información, etc.

5) MODCU (Modelo de Desarrollo de Casos de Uso)

Este modelo es el más importante de RUP, debido a que todo el desarrollo es guiado por los Casos de Uso. Este parámetro analiza en que grado se ha construido dicho modelo, según el valor es el esfuerzo estimado (HIGH no se ha construido el modelo, NOM se ha construido lo suficiente para empezar un desarrollo confiable y LOW el modelo está analizado y construido lo suficiente para confiar plenamente en él).

6) MODD (Modelo de Diseño)

La conclusión de este modelo engloba todos los aspectos antes mencionados de acuerdo con RUP, para la obtención del Modelo de datos. Su valor es HIGH cuando no se ha hecho o terminado el modelo, LOW si se ha terminado y es confiable su diseño.

- ✓ **EDBD correspondientes al Modelo Lógico**

7) CDIGC (Construcción de Diagrama de Clases)

Estima el esfuerzo que lleva la construcción del diagrama de clases, el cual representa al Modelo de Objetos, se plantea una escala de HIGH correspondiente a que no se ha construido el diagrama, NOM se ha construido plenamente las clases y/o relaciones y LOW la construcción del diagrama de clases está completo (con todas las clases, relaciones y métodos que modelan al sistema).

8) DMOO (Diseño del Modelo de Objetos)

Este multiplicador tiene que ver con el análisis del modelo y se considera los tres valores como HIGH no es suficiente el diseño, implica el esfuerzo de replantear el modelo, NOM es un modelo que si trata de capturar las clases, atributos, relaciones que conforman el sistema de manera óptima, esto es, se necesita una mayor profundidad en la investigación de las clases y sus relaciones que trata de modelar y LOW el análisis es satisfactorio.

9) CLPMOO (Clases Persistentes del Modelo de Objetos)

Este parámetro tiene que ver con el parámetro DMOO si no está concluido el diseño del modelo, no tiene ningún sentido tomar en cuenta este multiplicador, en caso contrario, si es conveniente este parámetro debido a que estas clases serán transformadas en las tablas de la base de datos y nos da una idea aproximada del tamaño que tendrá dicha base en desarrollo. Por lo tanto agregara esfuerzo y complejidad al Modelo de datos. Los valores corresponden a HIGH: clases persistentes no se han identificado, LOW: todas las clases persistentes han sido identificadas.

10) NCLPXR (Nivel de Complejidad de las Relaciones)

Una vez identificadas las clases persistentes, se necesita averiguar cuál es el nivel de complejidad de las relaciones entre estas mismas, esto resulta en aumento en la complejidad del modelo y además del esfuerzo de analizar e implementar dichas relaciones. La escala de NCLPXR es HIGH: la complejidad es muy alta de las relaciones entre las clases, NOM: las relaciones no tienen mucha complejidad al analizarlas y entenderlas y LOW: la complejidad de las relaciones entre las clases es muy sencilla, y por ende, resulta fácil de entender, analizar e implementar y no contribuyen esfuerzo extra al proyecto de la base de datos.

- ✓ **EDBD correspondientes al Modelo Físico.**

11) CLPXC (Complejidad de las Consultas)

Mide la complejidad de las consultas que generara el sistema o el usuario de acuerdo con los requerimientos. Este parámetro determina el nivel extra de esfuerzo involucrado por la creación de un índice y de la normalización de las tablas de la base de datos.

12) CLPXRI (Complejidad de las Reglas de Integridad)

Analiza la complejidad de las diferentes reglas de integridad que nos demandan los requerimientos de la base de datos.

13) SEGBD (Seguridad de la Base de Datos)

Multiplicador que mide el nivel requerido de seguridad para preservar los datos dentro de la base de datos, esto es el posible diseño de algo- ritmos de seguridad de cifrado o de acceso, independientemente de que SGBD se está utilizando.

3.5 Calibración de los multiplicadores de esfuerzo de una base de datos obtenidos a partir de RUP

3.5.1 Método de modelado de COCOMO II

Los autores de COCOMO II sugieren que antes de hacer cualquier recolección de datos para los parámetros de estimación, primero hay que definirlos muy bien, con el fin de que proporcionen una eficiente estimación del esfuerzo.

Esto significa que *una de las primeras necesidades que se deben de cubrir para determinar una forma funcional para el modelo de estimación, es la determinación de cuáles son los parámetros que influyen de manera determinante en el esfuerzo a estimar. Para realizar tal análisis de los parámetros obtenidos* (COCOMO II, 2000) sugiere que se sigan siete pasos que definen una metodología de modelado de COCOMO II, el cual se ilustra en la siguiente *Figura 3.12*:

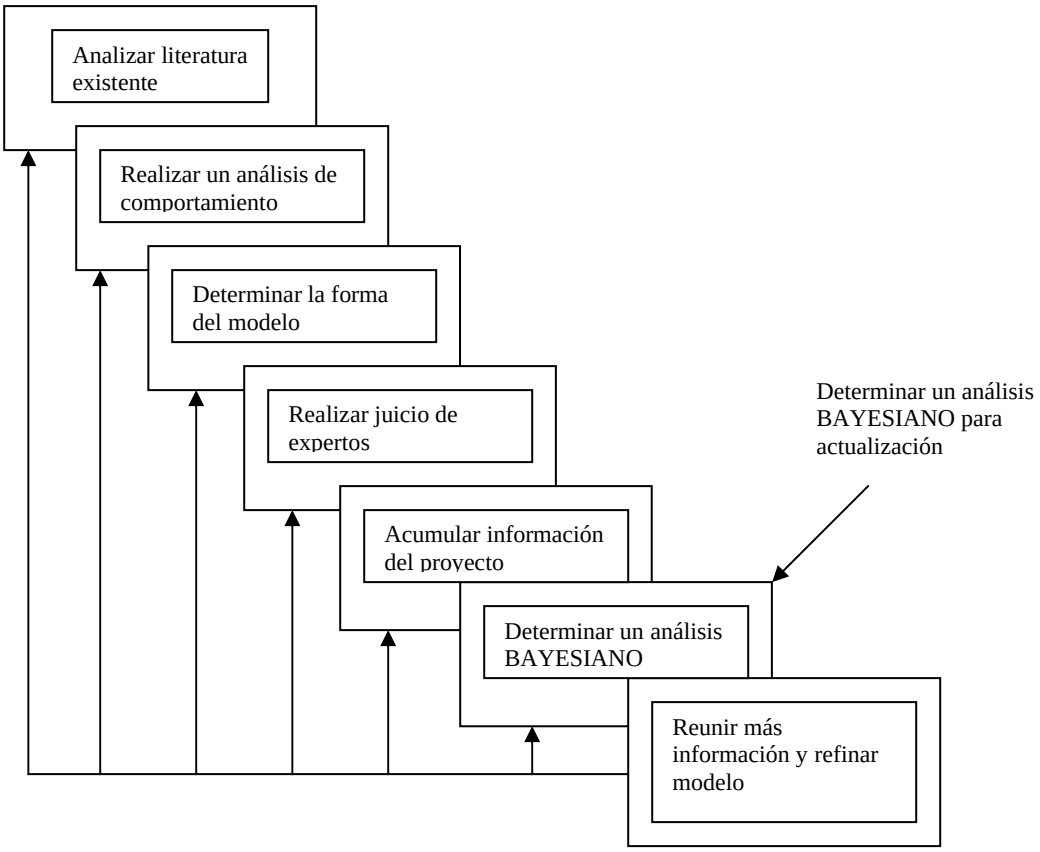


Figura 3.12 Metodología de modelado de COCOMO II (ob. cit COCOMO II)

Prácticamente a lo largo del estudio de este ***Capítulo 3*** hemos recorrido esta metodología, primero se empezó con el *paso 1*, buscando información relativa a los nuevos parámetros propuestos, concernientes al desarrollo de una base de datos siguiendo el Proceso Unificado de Racional (RUP), después, se realizó el análisis de comportamiento de los parámetros que se encontró al revisar el flujo de trabajo que concierne al diseño y desarrollo de una base de datos.

Una vez definidos los parámetros (**EDBD**), se dio lugar al *paso 2*, el cual analiza el comportamiento de cada parámetro; es decir, determinar las diferencias entre una evaluación de *Low* y *Extra-High* de cada uno de los parámetros, el significado conceptual de cada uno de los valores de los multiplicadores los cuales determinaran el esfuerzo.

Estos parámetros (**EDBD**) dan origen a la conformación del modelo de estimación correspondiente al *paso 3*, el cual se sugiere como un agregado conceptual (ver ***Capítulo 1*)** a la ecuación ***(2.1)*** de COCOMO II (Ver ***Capítulo 2***), como una multiplicación añadida a la multiplicatoria original de parámetros de esfuerzo del COCOMO II (ver para la construcción del modelo ***Apéndice C***), por las siguientes razones:

- ✓ El modelo de estimación de *Esfuerzo de Desarrollo de una Base de Datos* (**EDBD**) fue planteada siguiendo el comportamiento, de que los parámetros que lo componen, traducen el esfuerzo inherente al diseño , desarrollo e implementación de una base de datos, por lo tanto miden lo mismo $\mathbf{EM_j}$ y $\mathbf{EDBD_j}$.

- ✓ Cada uno de los parámetros encontrados siguiendo el flujo de la actividad de *Diseño de una base de datos*, están representados de manera general de acuerdo con las diferentes etapas de RUP, se puede observar que los esfuerzos invertidos en el proyecto, varían en cada fase, esto es debido a las diferentes actividades que impone la elaboración del proyecto.

Continuando con el segundo punto encontramos que el equipo que desarrolló RUP, halló una distribución típica de recursos humanos necesarios a lo largo de todo el proyecto (Véase *tabla 3.5*).

	Inicio	Elaboración	Construcción	Transición
Esfuerzo	5%	20%	65%	10%
Tiempo de dedicación	10%	30%	50%	10%

Tabla 3.5 Distribución típicas de esfuerzo y tiempo (ob. cit Jacobson).

La distribución se puede ver con la gráfica entre tiempo contra recursos, en la *Figura 3.13.*

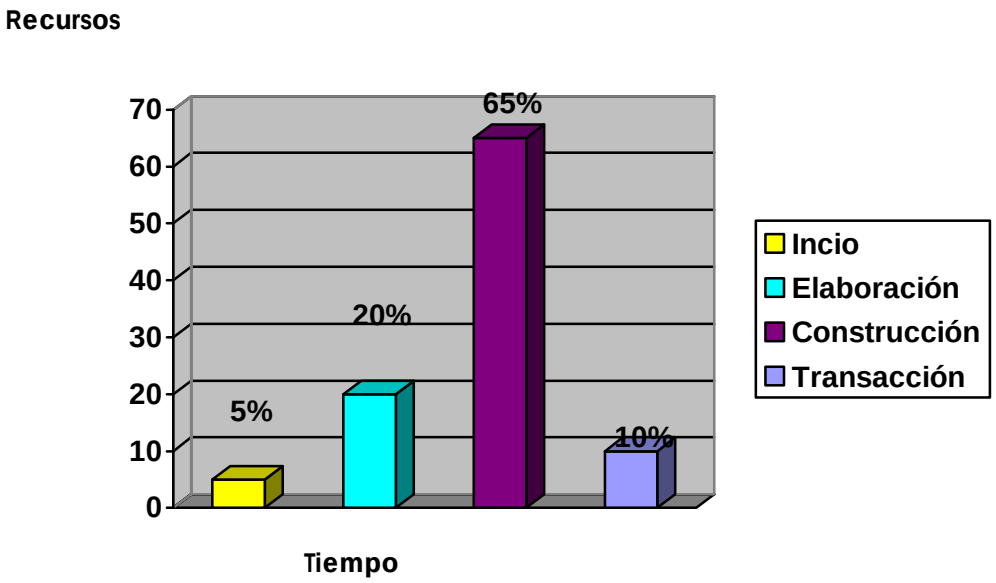


Figura 3.13 Distribución de esfuerzo tiempo-recursos (ob. cit Jacobson)

Las distribuciones de personal mostradas en el *Figura 3.13*, nos dan una idea de cómo se está gestando el esfuerzo a lo largo del proyecto, encontrando que la fase de construcción se necesita más personal y tiempo que en las demás fases del proyecto.

De las cifras dadas por el *(CSE-USC, 2000)* tenemos que por cada fase de RUP, la actividad de diseño consume un porcentaje representado en la siguiente *Figura 3.14*.

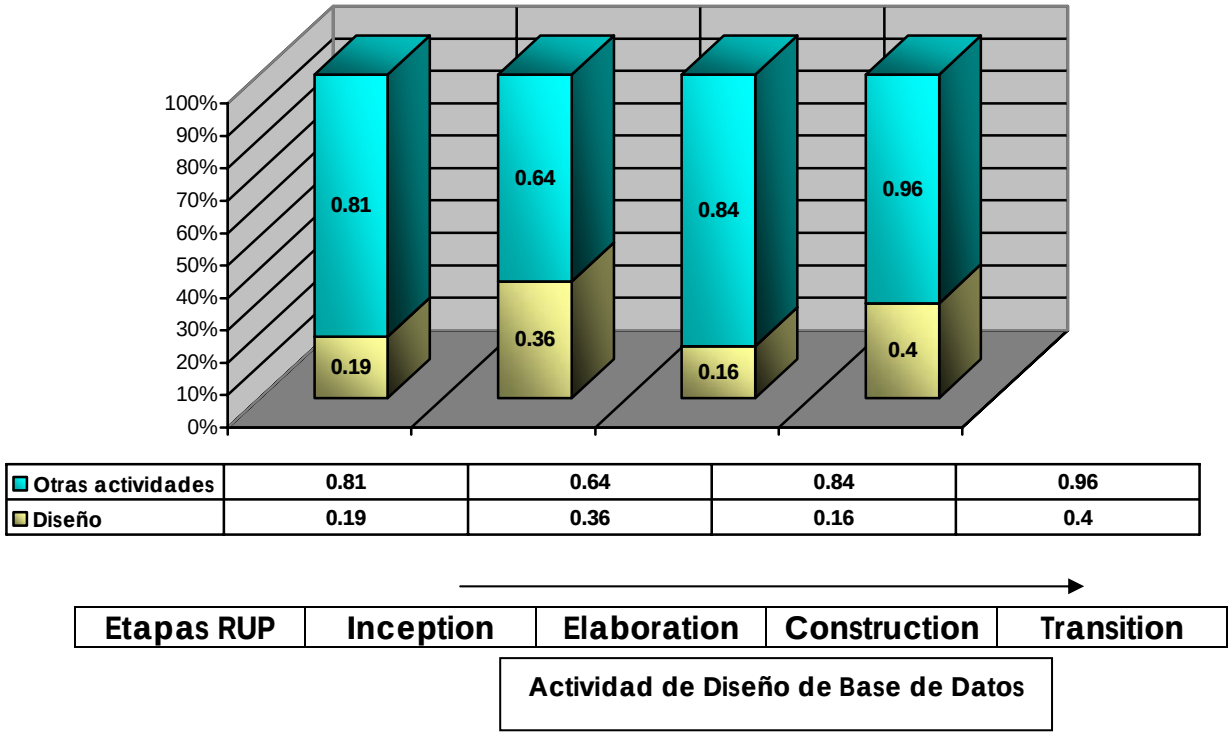


Figura 3.14 Esfuerzo invertido en la actividad: de Diseño para cada fase de RUP (ob. cit Jacobson)

A partir de los datos encontrados del esfuerzo en cada fase de RUP y de acuerdo con que la *Actividad de Diseño de una Base de Datos* comienza en la fase de Elaboración y continua a través de las fases de Construcción y Transición (ver *Figura 3.4*) podemos observar que consume un 30 % en la fase de Elaboración y un 16% y 4% respectivamente en las demás fases de RUP; además de investigaciones del Center for Engineering Software del USC se obtuvo que el 17 % del esfuerzo en el desarrollo de software es invertido en la parte de diseño[15] y este esfuerzo está repartido entre las 3 etapas de RUP que pueden corresponder a la parte de diseño de un producto de software, considerando el 30% de esfuerzo en el diseño de la base de datos.

[15] Este porcentaje es respecto a un *modelo de desarrollo en cascada*.

Por lo tanto tenemos que la *Actividad de Diseño de una Base de Datos* corresponde al 5.29% del esfuerzo invertido en total a través del proceso de desarrollo de un producto de software.

Estas premisas nos permiten asegurar que el comportamiento de los **EDBD**, obtenidos a partir de la evaluación de la actividad de *Diseño de una Base de Datos* se comporta de la misma forma que los ***EM*[16]** que corresponden a la ecuación ***(2.2)*** de COCOMO II *(ver **Capítulo 2**)*:

$$PM_{NS} = A \times Tamaño^{E} \times \prod_{i=1}^{n} EM_{i} \qquad \textbf{...(2.2)}$$

En ***(2.2)*** los multiplicadores de esfuerzo capturan las características del desarrollo que afectan el esfuerzo necesario para completar el proyecto.

Los **EM**, como se venía discutiendo en el ***Capítulo 2***, es un factor del modelo que incrementa o disminuye el costo estimado por el modelo. Todos los EM tienen un nivel de evaluación cualitativo que expresa el impacto del multiplicador en el esfuerzo del desarrollo del proyecto. De esta forma una valuación cualitativa se transforma en una evaluación cuantitativa, que puede usarse en el modelo.

De esta manera trabajan los **EDBD** por lo tanto podemos afirmar, como lo hemos estado haciendo a lo largo del estudio que proponemos en este ***Capítulo 3***, que son **EM**; pero agregados al modelo original ***(2.2)***.

[16] Los autores de COCOMO II proponen un modelo de búsqueda de multiplicadores de esfuerzo, ese modelo se describe en el ***Apéndice C***.

Estos **EDBD** fueron planteados para que funcionen de la misma manera que los **EM,** así que podemos agregarlos a ***(2.2)*** de la siguiente manera y da como resultado nuestra propuesta de **estimación de esfuerzo** denominada**: Ecuación propuesta de esfuerzo con multiplicadores EDBD (PM)**.

$$PM'_{NS} = A \times Tamaño^{E} \times \prod_{i=1}^{n} EM_i \times \prod_{i=1}^{n=13} EDBD \quad \textbf{...(3.1)}$$

Una observación que es pertinente hacer, en la ecuación ***(3.1)*** es que se está definiendo en el límite superior n=13 de la mutiplicatoria, por qué son el número de multiplicadores correspondientes a **EDBD** que se encontraron en nuestra evaluación de la actividad de diseño de una base de datos (*paso 2 de la metodología de modelado de COCOMO II*), pero podrían existir más si se hace un análisis más a fondo de los requerimientos funcionales que usa esta actividad, por lo tanto sugerimos que no hay límites para nuevos **EM** que estimen el esfuerzo que conlleva una base de datos.

La calibración de los **EDBD** obtenidos al evaluar la metodología que sugiere RUP, para el desarrollo de la base de datos, acata la misma forma en cómo se calibraron los multiplicadores originales de COCOMO II, debido a que se planteó que **EDBD** sean el agregado del modelo y necesitan apegarse al planteamiento del modelo original.

Por tal motivo , dentro de la conformación del modelo **EDBD**$_{k=13}$ se le asigna el valor multiplicativo con el cual realizará las estimaciones, por lo tanto se muestran en la siguiente *Tabla 3.6* una propuesta de calibración, que se basa en una regresión lineal multi-valorada con base a *tres* proyectos (*Ver* ***Apéndice C***), estos datos obtenidos han servido únicamente como una apreciación inicial de lo que se propone en este trabajo de tesis , son valores base a partir de los cuales se puede calibrar el modelo **PM** por ende no se puede asegurar que los valores sean lo más factibles o aproximados a la realidad absoluta que pudieran aplicarse en cualquier proyecto que haga uso de una base de datos, abundaremos más acerca de esta limitante en el ***Capítulo 5***.

Orden de los EDBD	Nombre de los EDBD	LOW	NOM	HIGH
1	DRUP	0.052	*	1.052
2	CEBD	0.75	1	1.75
3	DINFO	0.75	1	1.75
4	MODNG	0.75	1	1.75
5	MODCU	0.75	1	1.75
6	MODD	0.75	1	1.75
7	CDIGC	0.75	1	1.75
8	DMOO	0.75	1	1.75
9	CLPMO	0.75	1	1.75
10	NCLPXR	0.75	1	1.75
11	CPLXC	0.75	1	1.75
12	CPLXRI	0.75	1	1.75
13	SEGBD	0.75	1	1.75

Tabla 3.6 Multiplicadores EPBD con una calibración inicial

En primer lugar se encuentra DRUP, ya que se considera el multiplicador más importante debido a que el desarrollo de la base es llevado de acuerdo a RUP. La respuesta NOM de DRUP implica que se siguió desde el inicio RUP y por lo tanto se ha desarrollado todo el flujo de trabajo (con todas sus actividades) y HIGH nos indica que no se ha seguido el RUP por ende se tendrá que ajustar las actividades a las filosofías que nos dictan el RUP.

La metodología de modelado de COCOMO II , continua con el *Paso 4* en el cual los resultados de los análisis a la que fueron sometidos los parámetros de esfuerzo[17], se someten a experimentación de acuerdo con criterios de los expertos en la materia , para así asegurar que cada multiplicador realizará un buen desempeño estimando. El *Paso 5* es la recolección de la información de todos los tipos de proyectos que puedan aportar nuevos datos a los multiplicadores de esfuerzo; el *paso 6* sugiere realizar un análisis *Bayesiano a Posteriori* para actualizar cada uno de los multiplicadores, una vez obtenida, se vuelve a recolectar información de proyectos para una futura actualización (*paso 7*).

Estos últimos 4 pasos no están en el alcance de esta tesis, debido a que se necesita realizar una investigación y experimentación de más proyectos de diferentes características para formar nuestra propia base de datos , con la información histórica de esfuerzos , para así tener actualizado o quizá editar nuestros parámetros propuestos: $\mathbf{EDBD}_{k=13}$. Este punto será abordado con mayor profundidad en el ***Capítulo 5***.

[17] Este análisis, lo hemos seguido de acuerdo con un flujo de datos que propone COCOMO II, el cual se detalla en el ***Apéndice C***.

3.6 Obtención del tamaño por medio de Casos de Uso

La estimación del costo de un proyecto es más útil cuando se realiza en las etapas más tempranas del proyecto y de acuerdo con el Proceso Unificado, comúnmente en las primeras etapas de desarrollo no se haya elaborado código alguno, excepto cuando se va a re utilizar código de un proyecto anterior. Por eso en las primeras etapas no se puede utilizar el conteo de líneas de código (LDC) para determinar el tamaño del proyecto, ya que no se ha programado alguna línea de código en las primeras fases de RUP.

En sustitución de la métrica de conteo de líneas, resulta factible utilizar los Casos de Uso, los cuales representan una buena posibilidad para estimar el esfuerzo, porque los Casos de Uso es una manera excelente de captura y represtación de la funcionalidad de un sistema. Aunado a este criterio RUP utiliza los Casos de Uso como artefactos principales para el desarrollo de todo proyecto, en sus tres primeros flujos de trabajo (modelado de negocios, requerimientos, análisis y diseño).

Un Caso de Uso documenta una actividad completa que el sistema deberá desempeñar, demuestra la interacción entre el actor y el sistema (relación usuario-sistema) y la secuencia de acciones que deberán suceder para alimentar al sistema y por lo tanto que éste mismo responda de la manera deseable.

Por eso un sistema puede tener muchos Casos de Uso y entre todos éstos pueden representar una funcionalidad completa esperada para este sistema, al observar cada caso de uso por separado se puede obtener una estimación intuitiva del esfuerzo que se requerirá para la actividad documentada por el Caso de Uso, pero esto requiere un cierto grado de experiencia por parte del estimador. Los Casos de Uso por sí solos no pueden proporcionar un estimado del tamaño que tendrá el sistema ni del esfuerzo que tomará implementarlo.

Una de las formas de estimar el tamaño de un proyecto es por medio de los puntos de función (PF), en sus etapas tempranas, donde se formulan y elaboran los requerimientos del sistema. Además los PF pueden cuantificar el tamaño del sistema de manera independiente del lenguaje de programación utilizado para el desarrollo del sistema (ver ***Capítulo 1*** y ***Apéndice A***).

COCOMO II utiliza PF (en el *Modelo de diseño anticipado* y el *Post arquitectura*) para la realización de las estimaciones. Pero existe una propuesta alternativa de obtención de la métrica del software que se llama Análisis de Puntos de Casos de Uso, el cual es muy similar al análisis que se les hace a los PF, para su implementación.

Dependiendo de las definiciones de los PF y los Casos de Uso se puede llegar a una relación natural, debido a que los PF permiten estimar el tamaño de un proyecto por medio de sus requerimientos dependiendo de las funciones que necesite el usuario y los Casos de Uso permiten documentar los requerimientos definidos para el proyecto.

Las estimaciones en las etapas tempranas del proyecto son un poco burdas e imprecisas debido a que sólo se tienen la identificación de los actores y los Casos de Uso pero a medida que se va detallando y documentando cada Caso de Uso, son capaces de proporcionarnos detalles suficientes como para obtener una buena idea del esfuerzo necesario para desarrollar el proyecto, además detener una aproximación de su tamaño. Ahora si a los mismos Casos de Uso se aplica el análisis que sugieren para encontrar los PF, la estimación es más precisa.

Una vez que se han identificado los actores (usuarios) y los Casos de Uso que intervienen en el sistema que se va a desarrollar, se especifican a detalle cada uno de los Casos de Uso, describiendo los posibles escenarios que podrían observarse en cada Caso de Uso, se identificarían los posibles escenarios de acción que cumpliría el sistema.

Los escenarios muestran paso a paso las interacciones entre el actor y el sistema y la manera en que dicho sistema va a responder a las peticiones del actor y además, que en cada escenario las situaciones encontradas son distintas, es decir; cada escenario representa las alternativas de funcionamiento del sistema dado un Caso de Uso.

De los escenarios encontrados se desprende la descripción de secuencias que dan lugar a una o más *transacciones* (EO, EI y EQ) de los PF (*Ver* ***Apéndice A***). Proporcionando un mejor panorama de la complejidad de los Archivos Lógicos Internos (*ILF)* y Archivos Lógicos Externos *(ELF)*.

Para dar un ejemplo del Análisis de Casos de Uso para obtener PF analizaremos un Caso de Uso de un sistema que consulta su historial académico, por cuestiones de simplicidad supongamos que el actor llamado alumno cuenta con un password válido para ingresar al sistema de consulta, el cual merece un Caso de Uso que represente esta interacción actor-sistema (*Ver Figura 3.15*).

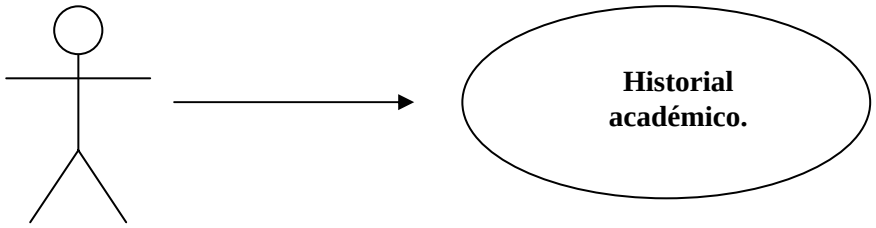


Figura 3.15 Caso de uso Historial Académico

Breve descripción	El actor alumno desea consultar su historial académico
Flujo de eventos	
Flujo principal	
	El actor solicita su historial
Flujo alternativo	1. Necesita entrar por medio de la página principal de alta
	2. No hay historial disponible

Tabla 3.7 Caso de Uso escenarios

De la especificación de cada uno de los flujos del caso de uso Historial Académica se identifica que:

- ✓ En el Flujo de eventos principal, se define un escenario que contiene una consulta externa (EQ).
- ✓ En el Flujo alternativo 1 se identifica un escenario que contiene una consulta externa (EQ).
- ✓ En el Flujo alternativo 2 se identifica un escenario que contiene una salida del sistema (EI).

De esta manera se analiza cada Caso de uso se obtienen los distintos escenarios, las transacciones y archivos que especifican los PF. Se puede cuantificar también el nivel de complejidad de las transacciones y de los archivos, y a medida que se va refinando los Casos de Uso se mejoran las estimaciones de los PF.

Los PFD son suficientes para estimar el tamaño de un proyecto, pero no para estimar esfuerzo y tiempo, se necesita aplicar los ajustes pertinentes.

Por eso existen dos métodos para estimar esfuerzo uno de estos, se denomina Cálculo de los PFA, visto en el ***Apéndice A*** y el segundo método que permite estimar esfuerzo (fórmula de PM de COCOMOII) éste usa directamente PFA (*ver* ***Capítulo 2***), es el mejor método cuando no se tiene ninguna información histórica de proyectos para ajustar los PF .

Como se vio en el ***Capítulo 2*** COCOMO II consiste en convertir los PF a SLOC, utilizando un Factor de Conversión que depende del lenguaje que se está utilizando para crear código del sistema (Ver ***Capítulo 2*** *Tabla 2.2. Conversión de Puntos de Función a Líneas de código*).

Este valor sustituye por el tamaño en la ecuación nominal del modelo $Tamaño = \left[1 + \frac{BRAK}{100}\right]$. **...(3.2)**

Y aplicando un procedimiento matemático se llega a la siguiente expresión:

$$Tamaño = PFD \times FC \quad \textbf{...(3.3)}$$

Donde:

PDF: Puntos de Función Desajustados
FC: Factor de conversión según lenguaje de programación

La manera de utilizar la ***(3.2)*** es muy sencilla, una vez obtenidos los PFD y vemos que FC le corresponde al lenguaje de programación que se utilizara para programar el sistema.
Ejemplo:

Si

PDF= 23

Y vamos a utilizar C++ para programar el sistema, entonces le corresponde un FC (Ver *Tabla 2.2*) cuyo valor es 55

Por lo tanto al aplicar ***(3.2)*** y tenemos una estimación de tamaño:

$$Tamaño = 23 * 56 \quad \textbf{...(3.3.1)}$$

Cuyo resultado es **: 1228 SLOC**

Capítulo 4

Nos estamos convirtiendo en una compañía de software
Dicho común entre las empresas

4 Aplicación del modelo a un caso práctico

En este capítulo, aplicaremos la expresión **PM´** obtenida en el ***Capítulo 3*** (*Diseño de una base de datos siguiendo el Proceso Unificado de Rational (RUP)*) a casos prácticos de desarrollo de un sistema que considera el uso de una base de datos. Estimaremos el esfuerzo determinado por el cociente: *personas/mes*, utilizando el software COCOMO II 1999.0, dicho software no considera los multiplicadores EDBD (*Esfuerzo de Desarrollo de la Base de Datos*) obtenidos en el capítulo anterior. Después se hará un análisis entre las mediciones de esfuerzo estimadas con el COCOMO II más el agregado ***EDBD*** (*Esfuerzo de Desarrollo de una Base de Datos*) y el modelo de estimación original.

4.1 Herramientas disponibles para realizar estimaciones basadas en el modelo COCOMO II

El software COCOMO II 1999.0 del Center for Software Engineering (CSE) de la University of South Carolina (USC) fue lanzado a mediados de 1999, y soporta los modelos de Diseño-Temprano y el Post-arquitectura y corre bajo las plataformas de Windows XP, NT, 98,95 y Sun OS 4x y 5x.

Aparte de esta aplicación existen otras herramientas basadas en el modelo como: versiones disponibles en la WEB para estimación, pero son aplicaciones experimentales, tal como COCOMO II with Heuristic Risk Assessment, el cual está disponible en la siguiente dirección electrónica:
http://sunset.usc.edu/research/COCOMOII/expert_cocomo/expert_cocomo2000.html

Esta herramienta está calibrada con nuevos valores de multiplicadores correspondientes al año 2000, y soporta únicamente el modelo Post-arquitectura (ver ***Capítulo 2***).

Básicamente las dos herramientas, están diseñadas con las ecuaciones de esfuerzo (**2.2**) y tiempo de desarrollo (**2.5**) del COCOMO II (ver ***Capítulo 2***) pero las calibraciones de los multiplicadores y factores de escala se ajustan a nuevos parámetros obtenidos a partir de los diversos análisis estadísticos que se le aplican a nuevos proyectos considerados en la base de datos del proyecto de COCOMO II (para mayor referencia acerca de la calibración ver ***Apéndice C***).

Escogimos utilizar la aplicación COCOMO II 1999.0[1], por ser una herramienta que no está sujeta a pruebas de experimentación y nos ayudará a agilizar la estimación del esfuerzo que necesitamos para nuestro estudio.

Están también disponibles las versiones anteriores a COCOMO II 1999.0, las cuales están a disposición en la siguiente página del CSE:

http://sunset.usc.edu/research/COCOMOII/cocomo_main.html#downloads

4.2 Pasos concretos para la realización de una estimación utilizando el modelo COCOMO II

La modelo COCOMO II propone los siguientes pasos para realizar una estimación:

Paso 1) Obtener una estimación del tamaño del proyecto, ya sea por medio de PF o por medio de los Puntos de Casos de Uso (procedimiento descrito en el ***Capítulo 3***), los cuales se transforman en KSLOC, dentro de este paso se tiene que tener en cuenta que si emplea código re utilizable, si es el caso se aplican otras expresiones matemáticas para estimar el tamaño del producto (Ver ***Apéndice C***) , todo esto depende del tipo de proyecto y de las decisiones que tome la organización al desarrollar el sistema.

[1] Para corroborar los resultados de los cálculos de las estimaciones se empleó una aplicación comercial basada en COCOMO II llamada COSTAR 7.0. El demo se encuentra disponible en esta página : www.softstars.com

Paso 2) Obtenemos los valores de los cinco Factores de Escala (SFj) y los diecisiete o los siete valores (dependiendo del modelo de estimación que necesitemos) de los Multiplicadores de Esfuerzo (EM_j) para saber las características del proyecto tales como: forma de trabajar de las personas relacionadas con el proyecto, la plataforma y la organización encargada de generar el producto (ver ***Capítulo 2***). Los valores de los SF_j se pueden obtener a partir de nivel proyecto, mientras que los EM pueden obtenerse por medio del proyecto a nivel de componente. Para nuestro caso de estudio, utilizaremos el Modelo de Diseño-Temprano, debido a que nuestros proyectos no están definidos las arquitecturas, y se encuentran en las etapas finales. La recolección de información tanto de SF_j y EM_j se hace a partir de cuestionarios y tablas para que, posteriormente la evaluación sea cuantificada por medio de los valores calibrados por COCOMO II (Ver ***Capítulo 2*** Tabla 2.3 para valores de SF_j y Tablas 2.6 y 2.7 para EM_j).

Paso 3) Una vez comprendidos y obtenidos los SF, los EM y las diferentes constantes calibradas del modelo, se aplica las ecuaciones de COCOMO II (ver ***Capítulo 2*** (2.2) y (2.5)) para obtener la estimación de Esfuerzo y Tiempo de Desarrollo.

Paso 4) Las estimaciones tempranas, casi por lo general, necesitan un ajuste, para estimar de manera más exacta conforme se obtiene un mayor número de información del proyecto; en esta parte se entra en un ciclo el cual es: si no se cumple con la satisfacción de la estimación obtenida, se regresa uno al paso 1 ó al paso 2, para volver a obtener los SF_j y EM_j más detallados y realizar nuevamente el paso 3, si no es así, se continua con el paso 5.

Paso 5) Se tiene una estimación satisfactoria del esfuerzo y del tiempo de desarrollo. En ese punto, COCOMO II sugiere hacer una base de datos de conocimientos, en la cual se guardarían todos los datos históricos y experiencias en las estimaciones de los proyectos.

A continuación se ilustra en la *Figura 4.1*el escenario de desarrollo de COCOMO II:

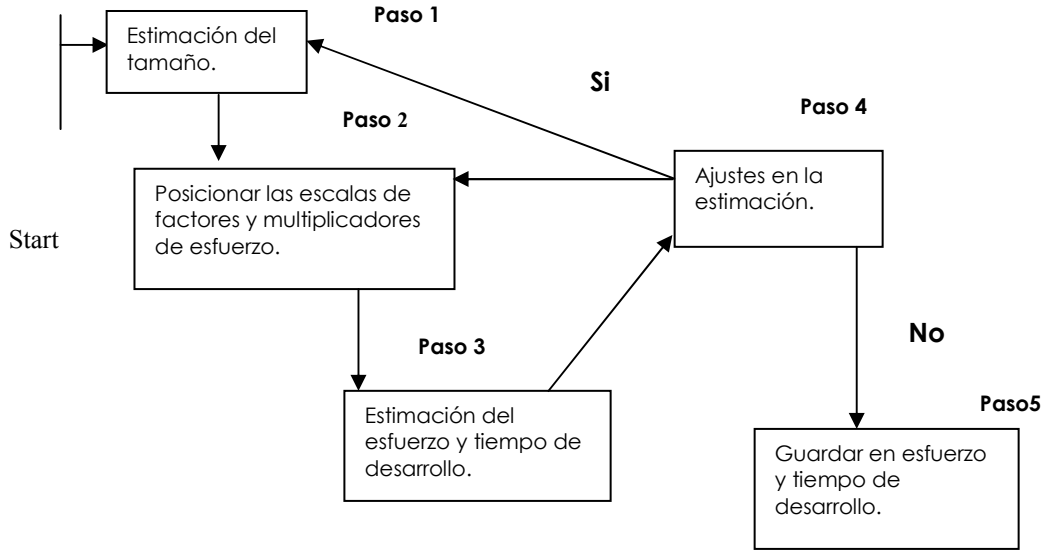


Figura 4.1 Desarrollo de estimación de COCOMO II

Un punto importante que nos recalcan los autores del modelo, es que al momento de hacer estimaciones tempranas, se deben hacer supuestos del proyecto; de acuerdo con la metodología de desarrollo, las estimaciones tempranas se pueden calibrar de acuerdo al grado de avance que tiene el proyecto.

4.3 Descripción de los proyectos a estimar

Utilizaremos dos proyectos para hacer nuestras estimaciones, siguiendo el COCOMO II original y posteriormente el modelo de estimación con el agregado conceptual (**$EDBD_{k=13}$**), el primero es un proyecto que se encuentra en las etapas finales, y es título de otra tesis, denominada: Sistema de Documentación de la Infraestructura de Telecomunicaciones (SIDIT)[2] , proyecto destinado para la administración y mantenimiento de los enlaces de fibra óptica que opera la Dirección General de Computo Académico de la UNAM (DGSCA). El objetivo es automatizar la administración y el mantenimiento de los enlaces de fibra óptica, cubriendo las siguientes necesidades:

- ✓ Debe contar con una interfaz gráfica amigable para el usuario, que permita registrar los movimientos, tales como consultas, mantenimiento, actualización, etc.
- ✓ Debe basar su desarrollo en la arquitectura cliente/servidor (aún no se decide que arquitectura se implementará más específicamente).
- ✓ Debe ser modular y flexible.
- ✓ Debe permitir tener centralizada la información.
- ✓ Debe facilitar la manipulación de la información.
- ✓ Debe ser un sistema confiable y seguro. *(SIDIT, 2006)*.

[2] La información aquí presentada pertenece a los autores del trabajo de tesis, su uso autorizado es únicamente para investigación de nuestra propuesta de estimación

El procedimiento que se lleva a cabo actualmente para la administración y mantenimiento de los enlaces de fibra óptica por medio de la DGSCA, se puede observar por medio de este diagrama de flujo.

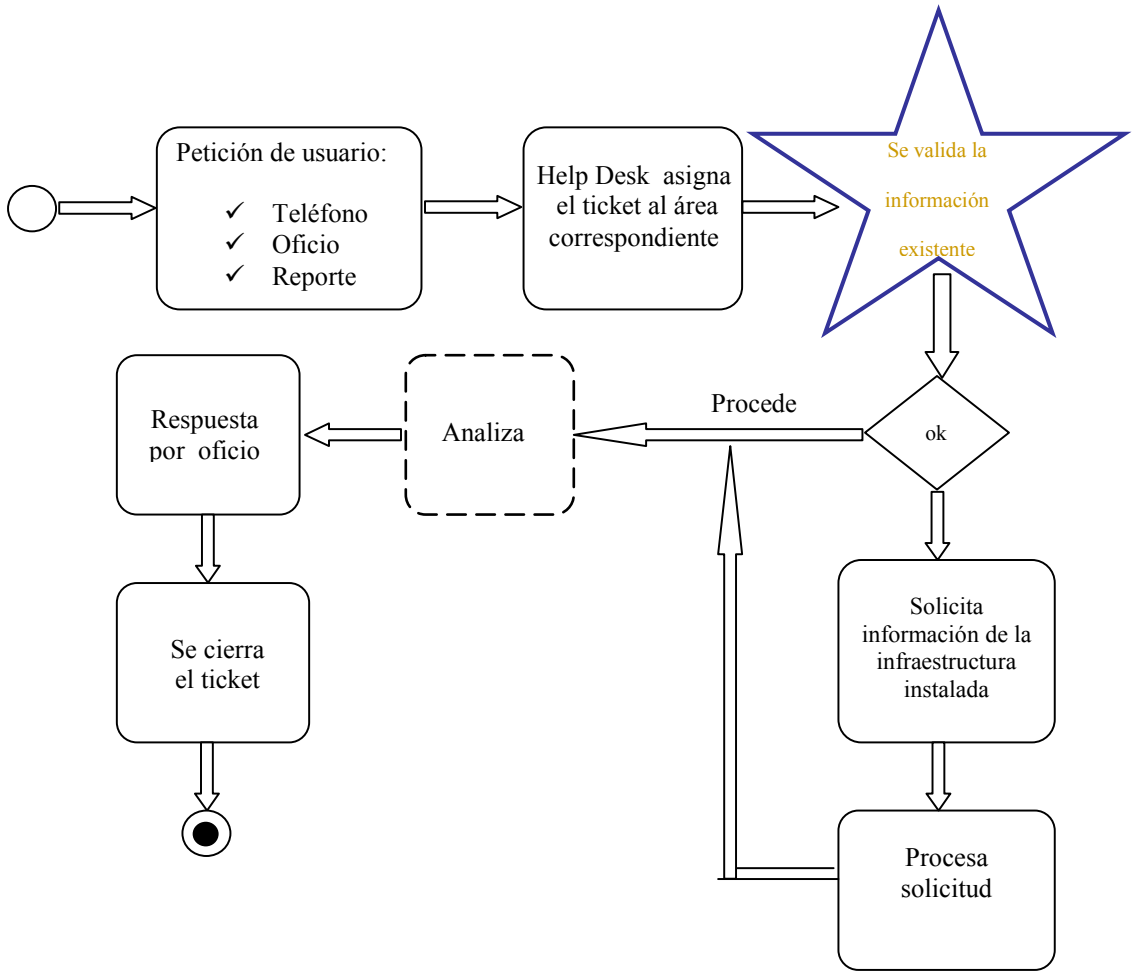


Figura 4.2 Flujo de información actual de administración de conexiones (ob. cit. SIDIT)

El sistema se centra en automatizar la parte de validación de información, el cual se realiza de la siguiente forma:

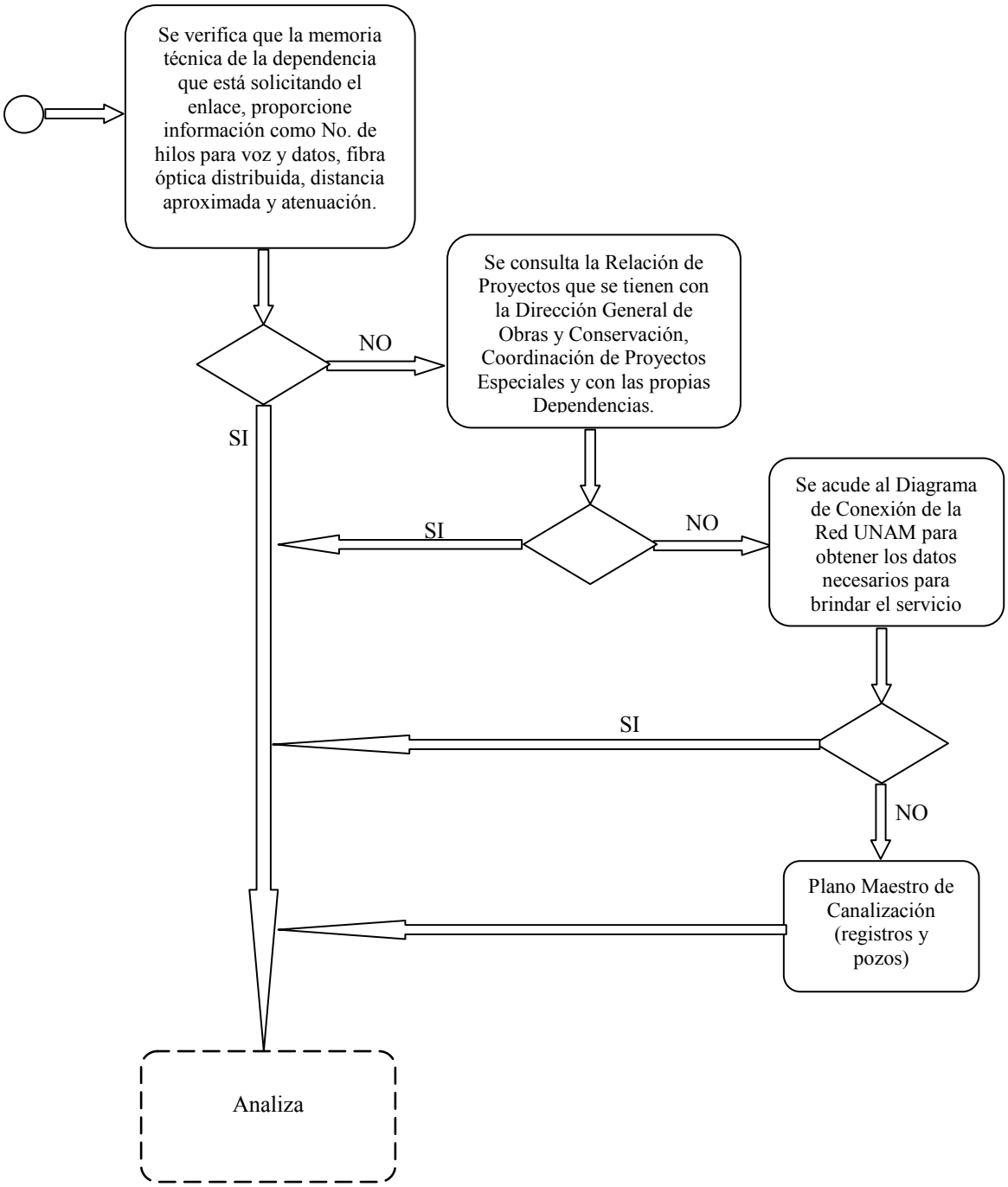


Figura 4.3 Flujo de la información del sistema SIDIT (ob. cit SIDIT)

La propuesta de utilizar este proyecto es debido a que en estos momentos está en desarrollo, utiliza una base de datos y además es interés por parte del equipo de desarrollo saber cuánto esfuerzo se empleará para su terminación.

La propuesta de solución para el desarrollo de este sistema, siguiendo un análisis por parte del equipo de desarrollo, de los requerimientos que el cliente (DGSCA) les impone son las siguientes:

- ✓ Sigue un modelo de desarrollo en espiral (no siguen RUP), pero será interesante analizar este punto, para hacer estimaciones sin seguir el proceso.
- ✓ Utiliza el SGBD llamado MySQL por ser actualmente con el que cuenta la DGSCA, resulta ser el más recomendado para la implementación de la base de datos que manejara SIDIT.
- ✓ Desarrollo de la aplicación por medio de ASP (Active Server Pages[3]) el cual utiliza el lenguaje de programación: Visual Basic, para realización de los diferentes scripts.

[3]ASP (Active Server Pages) es la tecnología desarrollada por Microsoft para la creación de páginas dinámicas del servidor. ASP se escribe en la misma página web, utilizando el lenguaje Visual Basic Script o Java Script.

Las características del sistema son las siguientes:

- ✓ Sistema bajo un ambiente de trabajo gráfico que opera sobre los sistemas operativos actualmente comerciales, integra funciones de administración y control de la información relacionada.
- ✓ La información es centralizada en un medio digital de almacenamiento que da seguridad y rapidez en su manipulación.
- ✓ El sistema da servicio a miembros del departamento de Telecomunicaciones quien es el encargado del control, mantenimiento y actualización de los enlaces.
- ✓ El sistema cuenta con un módulo que permite mostrar la información detallada de los enlaces, que puede ser vista en cualquier momento desde una computadora personal con servicio de Internet.

Clasificación de la fibra óptica.

- ✓ Permite la captura de los datos general de los enlaces como:

Existencia de enlaces de fibra óptica.

- ✓ No. de hilos.
- ✓ Tipo de conector.
- ✓ Distancia.
- ✓ Atenuación.
- ✓ Fecha de instalación.
- ✓ Nodo origen.
- ✓ Nodo destino.
- ✓ Pares ocupados.
- ✓ Tipo de fibra.

Permite la captura de la información personal de los responsables como:

- ✓ Nombre completo.
- ✓ Teléfono.
- ✓ E-mail.

El sistema realiza la administración de la información mediante altas, bajas y actualización sólo para el Departamento de Telecomunicaciones de la DGSCA, además de incorporar mecanismos de seguridad de acceso.

El sistema es capaz de soportar gran cantidad de información, además de ofrecer un amplio manejo y óptimo desempeño en cualquier instante de tiempo, por uno o varios usuarios autorizados conectados simultáneamente.

El sistema permite realizar búsquedas de registro existentes bajo diferentes criterios como: dependencia, tipo de enlace, tipo de conector, entre otros.

Tipos de usuarios (actores)

Administradores de sistema. Usuario encargado de la administración de los enlaces que contará con permiso de control total sobre el sistema.

Miembros del departamento de Telecomunicaciones de la DGSCA. Usuarios encargados del control y mantenimiento de los enlaces, asimismo, están facultados para consultar la información y brindar un servicio, teniendo acceso a todas las pantallas del sistema pero sin permiso para hacer modificaciones (*ob. cit SIDIT*).

Características de la base de datos

Se generará una base de datos, la cual residirá en una computadora central en donde se almacenarán, las entidades con sus respectivos atributos identificados en el planteamiento del proyecto:

- ✓ Dependencias.
- ✓ Enlaces proporcionados.
- ✓ Usuarios.
- ✓ Mantenimiento.
- ✓ Enlace cobre.
- ✓ Enlace fibra óptica.

Definición de casos de usos de SIDIT.

En este apartado se hace una breve explicación de cada uno de los Caso de Uso que diseño el equipo de desarrollo, teniendo así un Modelo de Caso de Uso para la representación de los requerimientos.

Se identifican dos actores:

Casos de uso del administrador del SIDIT.

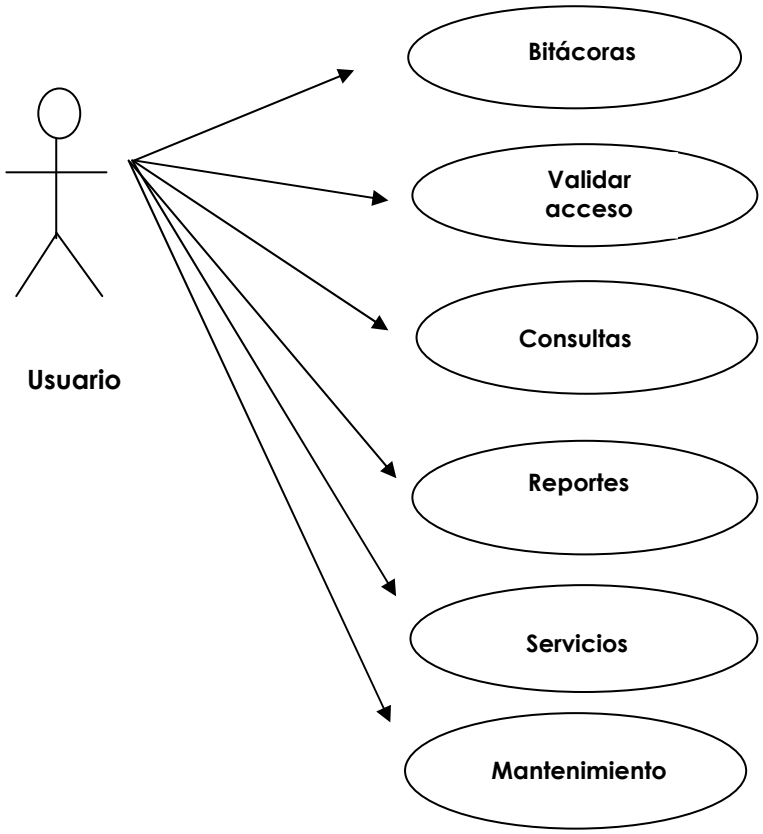


Figura 4.4 Casos de uso para el actor Administrador de SIDIT

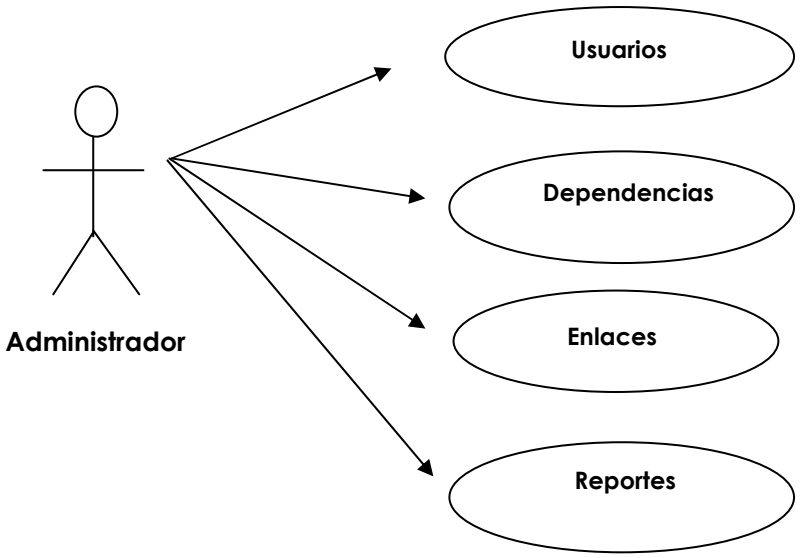


Figura 4.5 Casos de uso para el actor Usuario de SIDIT

La breve descripción de los casos de usos para cada uno de los actores se explican a continuación (para mayor referencia consultar ***Capítulo 4*** SIDIT):

- ✓ **Casos de uso del Actor: Administrador**.

Dependencias.

Dependencias (Actualización de dependencias) Breve descripción:	El proceso mediante el cual el administrador del sistema puede editar información de una dependencia, o bien, agregar o eliminar registros.

Usuarios.

Usuarios (Actualización de usuarios) Breve descripción:	El proceso mediante el cual el administrador del sistema puede editar la información concerniente a un usuario, o bien agregar o eliminar registros de usuarios.

Validar acceso.

Validar acceso Breve descripción:	El proceso mediante el cual el administrador proporciona sus datos de acceso al sistema (login, password) el cual asigna los privilegios asociados al tipo de usuario.

- ✓ **Casos de uso del Actor: Usuario.**

Enlaces.

Actualización de enlaces Breve descripción:	El proceso mediante el cual un usuario captura información sobre el(los) enlace(s) proporcionado(s) a alguna(s) dependencia(s).

Mantenimiento.

Mantenimiento Breve descripción:	El proceso mediante el cual un usuario captura información sobre el ò (los) mantenimiento(s) realizado(s) a uno ò a varios enlace(s).

Consulta.

Consulta Breve descripción:	El proceso mediante el cual un usuario realiza una consulta de la memoria técnica de una dependencia.

Reportes.

Reportes Breve descripción:	El proceso mediante el cual un usuario imprime un informe de una dependencia y un tipo de enlace seleccionado, teniendo la opción de personalizarlo, seleccionando qué información desea que contenga dicho reporte.

Validar acceso.

Validar acceso Breve descripción:	El proceso mediante el cual el usuario proporciona sus datos de acceso al sistema (login, password) el cual asigna los privilegios asociados al tipo de usuario logeado.

El otro proyecto a estimar, es el denominado: Sistema de Gestión y Adquisición Automatizada de Información del *Sistema de Medición de la Red de Agua Potable* SGAD IMTA[4] (*SGAD IMTA, 2003*), el cual es un proyecto desarrollado como parte de un trabajo anterior de tesis, en el cual este sistema fue tomado como caso práctico de estimación utilizando COCOMO II, pero sin tomar en cuenta los nuevos multiplicadores de esfuerzo que proponemos en este trabajo de tesis.

La problemática del sistema es que los instrumentos de medición instalados en la red de IMTA presentan la deficiencia de no analizar la información proporcionada, por lo que no es posible modificar la frecuencia de la adquisición de datos ni almacenarlos en una base de datos que permita determinar patrones de comportamiento del fluido.

La razón del desarrollo del sistema es que para la obtención y procesamiento de la información de un histórico de lecturas se depende de terceras personas ajenas al IMTA, aunado a ese punto, se tiene que existe una perdida de datos de medición al no ser modificable la frecuencia de la adquisición de datos.

El sistema permitirá la automatización de la adquisición de los datos, controlará la información procesada por los equipos de medición sin necesidad de personal ajeno al Instituto, se generará una base de datos con los históricos de las lecturas obtenidas y se accederá a la aplicación por medio de una página Web.

[4] IMTA Instituto Mexicano de Tecnología del Agua de la Secretaria del Medio Ambiente y Recursos Naturales (SEMARNAT) para mayor información del Instituto visitar : http://www.imta.mx/

Las características específicas del sistema son:

- ✓ El software de administración será diseñada para que trabaje bajo la plataforma de Linux.

- ✓ Despliegue de datos y gráficas se hará en un formato compatible con las hojas de cálculo comerciales, por ejemplo Excel.

- ✓ El programa se presentará en dos formatos: un programa ejecutable y vía Web.

- ✓ Existen tres niveles de acceso de acuerdo en la disposición de la información de las lecturas.

- ✓ La pantalla del sistema desplegará La distribución de los módulos en el mapa de instalaciones del Instituto en donde se tendrá una lista de los 32 puntos de medición y la última lectura registrada de cada uno, además de información específica de cada módulo.

- ✓ Habrá una selección de tipo de despliegue de la información de las lecturas.

- ✓ Contará con la opción de generación de reportes con cuatro opciones de despliegue: semanal, mensual, anual o aquella especificada por el usuario (cada 15 minutos, quincenal, etc.).

- ✓ Para la impresión de los reportes se contará con las mismas opciones de despliegue de lecturas.

El equipo de desarrollo de SGAD-IMTA identificó 2 tipos de actores, los cuales se describirán a continuación con sus respectivos casos de uso:

Administrador:

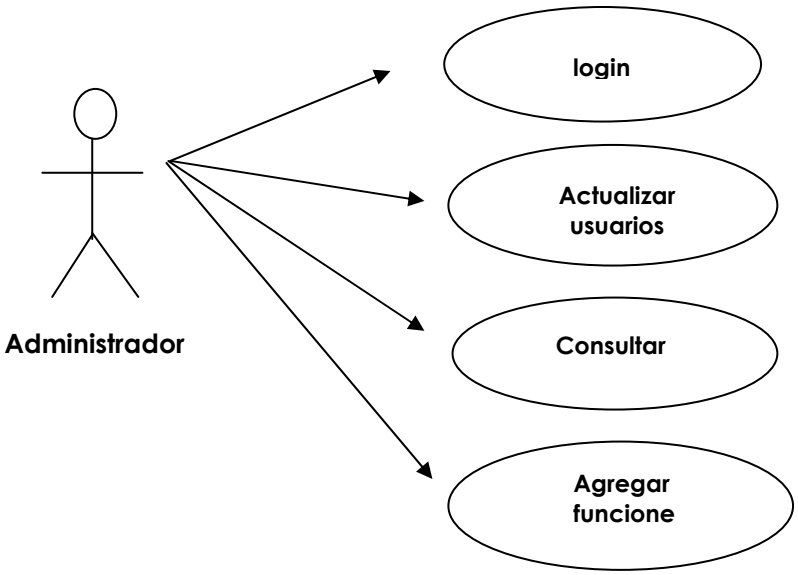


Figura 4.6 Casos de uso de administrador de SGAD-IMTA

Usuarios:

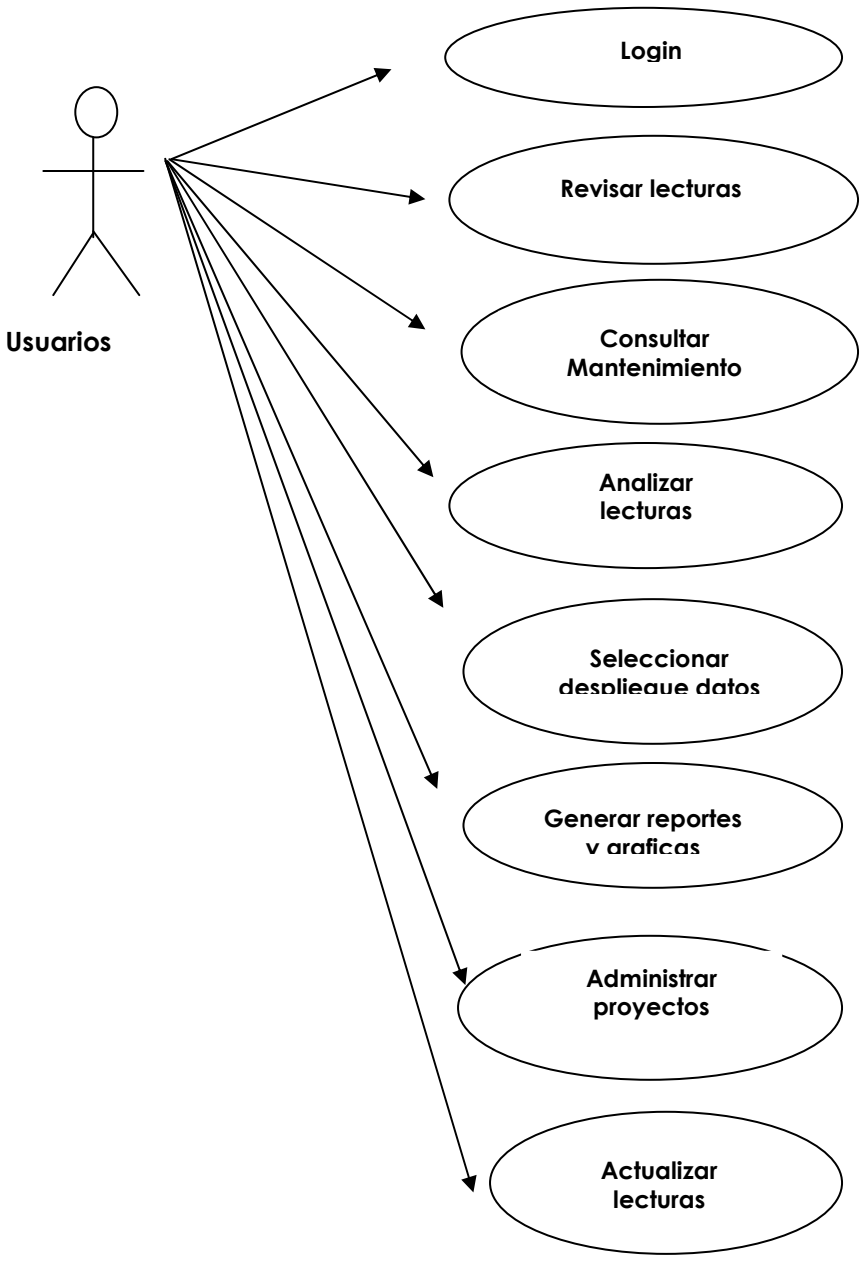


Figura 4.7 Casos de uso de usuarios de SGAD-IMTA

La breve descripción de los casos de usos para cada uno de los actores se explican a continuación (para mayor referencia consultar ***Capítulo 4*** *SGDAD-IMTA*).

- ✓ **Casos de Uso del Administrador**.

Login.

Login Breve descripción:	Login describe el proceso de autenticación y acceso al sistema de cualquier tipo de usuario.

Actualizar Usuarios.

Actualizar Usuarios Breve descripción:	Describe las interacciones entre el sistema para dar de alta, baja al usuario ó modificar sus privilegios.

Consultar historial de accesos.

Consultar historial de accesos Breve descripción:	Se refiere al despliegue de las veces que el usuario ha accedido al sistema dado un periodo determinado, el sistema proporciona un listado de los usuarios que ha accedido y las funciones que ha utilizado durante su sesión.

Agregar funciones.

Agregar funciones Breve descripción:	El administrador indica al sistema las nuevas funciones que deberán estar disponibles en el siguiente acceso.

- ✓ **Casos de Uso del Usuario**.

Revisar lecturas.

Revisar lecturas Breve descripción:	El usuario selecciona el proyecto que desea consultar de una lista que se despliega, una vez seleccionado se despliega las últimas lecturas obtenidas.

Consultar Historial de Mantenimiento.

Consulta de Historial Breve descripción:	El usuario elige un proyecto de la lista y selecciona la opción de equipos, al escoger el equipo que desea revisar y el sistema se despliega el historial de mantenimiento del equipo seleccionado.

Consultar Historial de Mantenimiento.

Consulta de Historial Breve descripción:	El usuario elige un proyecto de la lista y selecciona la opción de equipos, al escoger el equipo que desea revisar el sistema. Despliega el historial de mantenimiento del equipo seleccionado.

Analizar Lecturas.

Analiza lecturas Breve descripción:	El usuario elige el proyecto y de este mismo se elige el punto que deberá ser analizado, Se selecciona el periodo del cual se tomarán las lecturas que serán analizadas, y dado este acotamiento, el sistema realiza los cálculos necesarios para el análisis de las lecturas, despliega los resultados y genera un reporte de la información obtenida.

Administración de proyectos.

Administración de proyectos Breve descripción:	El usuario puede crear, monitorear y cancelar un proyecto en el sistema. Si decide cancelarlo lo selecciona de la lista de proyectos, así como si desea monitorearlo. Si desea crear un proyecto debe seleccionar el punto, los módulos y equipo que se consultaran para ese proyecto.

Actualizar lecturas.

Actualizar lecturas. Breve descripción:	El usuario selecciona la opción lecturas, el punto para la lectura y se solicita las ultimas lecturas que obtiene el sistema.

Ambos proyectos fueron seleccionados para estimarlos debido a que cuentan con una documentación muy amplia, por lo que la falta de disposición de información no dificultó la actividad de aplicar el modelo de COCOMO II. Los datos proporcionados a esta tesis serán utilizados *únicamente y exclusivamente para fines académicos y de investigación*, por ningún motivo el autor de esta tesis, pretende adjudicarse el crédito del trabajo realizado por los equipos de desarrollo de estos sistemas.

4.4 Aplicación del método de estimación de COCOMO II.

4.4.1 Estimación del proyecto Sistema de Documentación de la Infraestructura de Telecomunicaciones (SIDIT)

Como se había descrito anteriormente en el apartado 4.2, el modelo de estimación COCOMO II establece una serie de pasos para realizar el trabajo de estimar, estos los aplicaremos a los proyectos ejemplo, para obtener una estimación siguiendo el modelo original.

- ✓ Para el Sistema de Documentación de la Infraestructura de Telecomunicaciones (SIDIT)

Antes de entrar en al proceso de estimación, (*COCOMOII, 2000*) nos previene que al realizar esta actividad debemos hacer ciertos supuestos, ya que algunas veces no se cuenta con la suficiente información acerca de alguna característica que deseamos evaluar del proyecto.

En este caso, para este proyecto suponemos que todavía está en construcción y que no se ha definido en su totalidad su arquitectura, por lo tanto utilizaremos el modelo de Diseño-Temprano (de acuerdo con las características del modelo descritas en el ***Capítulo 2***) para estimar, el esfuerzo (PM) y el tiempo de desarrollo (TDEV).

Paso 1) Estimación de Tamaño de SIDIT

Encontramos los siguientes PF con sus respectivas características y pesos, utilizando el procedimiento de Puntos de Casos de Uso (descrito en el ***Capítulo 3*** y ***Apéndice A***). Posteriormente obtendríamos los PFA para que finalmente convertirlos en SLOC, los cuales utiliza COCOMO II, para estimar tamaño.

Otro punto importante, en este desarrollo no se re utilizó código, por ende, hay una reducción del esfuerzo de comprensión del código, modificación, en fin puntos que implican un aumento en el esfuerzo característicos al re uso de código. Para llegar a esta conclusión se contestó el siguiente cuestionario (basado en las guías de ayuda para la estimación de re uso de COCOMO II), que nos ayuda a saber si en el proyecto hubo re-uso.

- ✓ **Modelo de re uso**

1) ¿Existe software que se haya desarrollado previamente y que pueda volver a usarse para el proyecto ¿

Si	**No**

Si su respuesta fue afirmativa:

2) ¿Cómo calificaría la claridad del código en función a la comprensión de la :

Claridad de la aplicación:

Muy Baja	***Baja***	***Nominal***	***Alta***	***Muy Alta***

Auto descripción:

Muy Baja	***Baja***	***Nominal***	***Alta***	***Muy Alta***

Estructura:

Muy Baja	***Baja***	***Nominal***	***Alta***	***Muy Alta***

3) ¿Qué tanto necesitaría para entender los módulos desarrollados previamente?

Nada	***Revisión básica del módulo y de la documentación***	***Pruebas y evaluación del módulo y estudio de la documentación***	***Pruebas y evaluaciones considerables del módulo y del estudio de la documentación***	***Pruebas y evaluaciones exhaustivas del módulo y del estudio de la documentación***

4) Si alguna vez ha visto este software: ¿Cómo diría que resulta para usted?

Completamente familiar	***Muy familiar***	***Algo familiar***	***Considerablemente familiar***	***Muy desconocido***	***Completamente desconocido***

5) ¿Cómo calificaría el proyecto?

Nuevo: Comenzando el desarrollo desde cero	***Adaptado, con cambios a un software pre desarrollado***	***Reutilizado, con módulos inalterados de software existente.***	***Software off the shelf (OTS)***

Los Puntos de Casos de Uso encontrados se resumen en la siguiente *Tabla 4.1*:

Caso de Uso	**Transacciones**			**Total**	**Ponderación**
	E	*S*	*C*		
Validar acceso	*1*	*0*	*1*	2	Simple
Actualización-Dependencias	*3*	*0*	*3*	6	Medio
Actualización-Enlaces	*3*	*0*	*3*	6	Medio
Actualización-Usuarios	*3*	*0*	*3*	6	Medio
Actualización-Servicios	*1*	*0*	*1*	3	Simple
Actualización-Mantenimientos	*1*	*0*	*1*	3	Simple
Consultas	*3*	*0*	*2*	5	Medio
Reportes-Generar	*1*	*1*	*1*	3	Simple
Reportes - Bitácora	*1*	*1*	*1*	3	Simple

Tabla 4.1 Puntos de función de SIDIT

Donde:

S: archivo de salida

E: archivo de entrada

C: archivo de consulta

Paso 2) Obtención de valores para Factores de Escala (SF_j) y Multiplicadores de Esfuerzo (EM_j)

La recolección de la información para la obtención de los valores Factores de Escala, la podemos realizar por medio del cuestionario (basado en tablas 2.11, 2.12, 2.13, 2.14 y 2.15 de COCOMO II) que los desarrolladores de SIDIT, contestaron (ver apartado 4.4.1) Estas preguntas ayudan a obtener los diferentes pesos de los factores: **PREC, FLEX, RESL, TEAM, PMAT** (definidos en el ***Capítulo 2***).

- ✓ **Cuestionario para la evaluación de factores de escala de COCOMO II.**

Cuestionario para la obtención de factores de escala de COCOMO II

Nombre del proyecto: Sistema de Documentación de la Infraestructura de Telecomunicaciones

Contestó: Gonzáles Uribe Roxana (**analista**) y Sierra Moncayo José Pablo (**programador y diseñador de la base de datos**).

Subraye ó sombreé la opción que mas le parezca adecuada. Si no sabe alguna respuesta, continúe con la siguiente.

1) ¿Qué proyectos similares existen?

Sin precedentes	**Alguno que otro similar**	**Pocos precedentes**	**Varios precedentes**	**Muchos precedentes**	**Producto muy comercial**

2) ¿Cómo calificaría la comprensión organizacional de los objetivos del producto?

General	***Considerable***	***Muy buena***

3) ¿Cómo calificaría su experiencia con proyectos similares a éste?

Moderada	***Considerable***	***Extensiva***

4) ¿Hay necesidad de desarrollo concurrente asociado a hardware e a nuevos procedimientos operacionales para este sistema?

Moderada	***Alguna***	***Extensiva***

5) ¿Hay necesidad de innovar algoritmos o arquitecturas de procesamientos de datos para este sistema?

Mínima	***Alguna***	***Considerable***

De acuerdo con los requisitos que proporciona el cliente:

6) ¿Qué tan flexible es el proyecto de acuerdo a la configuración del software o requerimientos preestablecidos?

Muy flexible	***Flexible***	***Poco Flexible***

7) ¿Que tan flexible es el proyecto de acuerdo a la configuración de la interfaz del sistema?

Muy Flexible	***Flexible***	***Poco Flexible***

8) ¿Qué tan flexible es el proyecto de acuerdo con la configuración y los requerimientos del sistema como recompensa a la finalización temprana del proyecto ¿

Muy Flexible	***Flexible***	***Poco Flexible***

9) ¿Se lleva propiamente un Plan de Administración de Riesgos?

No	***Poco***	***Algo***	***De manera muy general***	***En su mayor parte***	***Completamente***

10) El calendario y el presupuesto ¿Son compatibles con el Plan de Administración de Riesgos?

(* No hay presupuesto)

No	Poco	Algo	De manera muy general	En su mayor parte	Completamente

11) ¿Cuál es el porcentaje de calendario de desarrollo reservado para definir y establecer la arquitectura según los objetivos del sistema?

5	10	17	25	33	40

12) Porcentaje de arquitectura de software disponible para este proyecto es de:

20 %	40%	60%	80%	100%	120%

13) Emplean herramientas de soporte disponibles para resolver factores de riesgo, aspectos de desarrollo y verificación de la arquitectura:

Ninguna	Una	Algunas	Varias	Muchas	Incontables

14) ¿Qué tanta incertidumbre hay en los aspectos clave de la arquitectura, tales como: misión del proyecto, interfaz del usuario, hardware, tecnología usada, desempeño?

Extrema	Poca	Considerable	Alguna	Muy poca	Casi nada
20%	40%	60%	80%	100%	120%

15) ¿Cuál es el número de los factores de riesgo y su criticidad en el proyecto?

>10	5-10	2-4	1	>5	<5
Críticos	Críticos	Críticos	Critico	Críticos	Críticos

16) ¿Cómo podría definir las interacciones y la disponibilidad para trabajar tiene el equipo de desarrollo?

Muy difíciles	Un poco difíciles	Básicamente cooperativas	Mucha cooperación	Altamente cooperativas	Disponibilidad y cooperación en todo momento

17) ¿En qué nivel equivalente al proceso de madurez sitúa al equipo de desarrollo?

Muy bajo	Bajo	Nominal	Alto	Muy alto	Extra alto

18) ¿Qué tanta consistencia hay entre los objetivos y la cultura de todo el equipo de desarrollo?

Casi nada	Poca	Básica	Considerable	Fuerte	Completamente

19) ¿Qué tanta habilidad y deseo existe entre el equipo de desarrollo para homogenizar los objetivos de sus integrantes?

Casi nada	***Poca***	***Básica***	***Considerable***	***Fuerte***	***Completamente***

20) ¿Qué tanta experiencia de trabajo en este equipo de desarrollo tienen sus integrantes?

Ningun	***Muy poca***	***Poca***	***Básica***	***Considerable***	***Extensiva***

21) ¿Cual será la capacidad de los miembros del equipo para cumplir sus compromisos y alcanzar una visión compartida?

Ningun	***Muy poca***	***Poca***	***Básica***	***Considerable***	***Extensiva***

Después de analizar (en base a ***COCOMO II, Pág. 30 Capítulo 2***) cada una de las preguntas para la obtención de SF_j se obtuvieron los siguientes valores de los factores de escala que se muestran en la *Tabla 4.2*.

SF_j	**Valores**
PREC	6.20
FLEX	1.01
RESL	1.41 (90%)
TEAM	1.10
CMM	7.80

Tabla 4.2 Factores de escala para SIDIT

Para obtener los diferentes multiplicadores de esfuerzo correspondientes al Modelo de Diseño-Temprano, utilizaremos la forma *SPD-1 Información en general para COCOMO II*, para proyectos que no han sido terminados o apenas se están empezando a desarrollarse*, (Ver en* ***Apéndice C****)*.

Los resultados de la evaluación de los multiplicadores de esfuerzo se listan a continuación en la *Tabla 4.3*.

EM_i	Ponderaciones	Valores
RCPX	HIGH	1.33
RUSE	NOM	1.00
PDIF	VERY HIGH	1.81
PERS	LOW	1.62
PREX	LOW	1.22
FCIL	NOM	1.0
SCED	VERY LOW	1.43

Tabla 4.3 Valores de los multiplicadores de esfuerzo de SIDIT.

Una vez recolectada la información acerca del tamaño del proyecto en SLOC (SIZE), los factores de escala (SF_j) y los multiplicadores de esfuerzo (EM_i), pasamos a utilizar el software de COCOMOII .1999, para realizar los cálculos de las estimaciones de esfuerzo (PM) y de tiempo de desarrollo (TDEV), descritos en el apartado **(4.5)**.

4.4.2 Estimación del proyecto: (SGAD-IMTA)

Para la realización de esta estimación, mostraremos unidamente los resultados de los pasos del desarrollo de la estimación de COCOMO II, ya que es prácticamente volver a mostrar el procedimiento anterior, para mayor detalle de donde se obtuvo la información, ver el ***Apéndice D***).

Paso 1) Estimación de tamaño.

La estimación del tamaño del proyecto es la siguiente: (Ver *Tabla 4.4*)

Caso de Uso	Transacciones			Total	Ponderación
	E	*S*	*C*		
Validar acceso	*1*	*0*	*1*	2	Simple
Actualización-Usuarios	*2*	*0*	*3*	*5*	Medio
Consultar historial de accesos	*0*	*1*	*1*	2	Simple
Agregar funciones	*1*	*0*	*1*	2	Simple
Revisar lecturas	*2*	*1*	*2*	*5*	Medio
Consultar Historial de Manto	*3*	*1*	*1*	*5*	Medio
Analizar lecturas	*3*	*2*	*2*	*7*	Medio
Seleccionar despliegue de datos	*2*	*0*	*1*	3	Simple
Generar reportes	*2*	*3*	*1*	*6*	Medio
Gestionar Proyectos				8	Complejo
Analizar lecturas	1	1	1	3	Simple

Tabla 4.4 Puntos de Caso de Uso del proyecto (ob. cit SGAD-IMTA)

En este proyecto se calificó que no había código re utilizable por lo tanto no se toma en cuenta los factores aunados a la re utilización de código (para mayor información ver ***Apéndice D***).

Paso 2) Obtención de los Factores de Escala (SF_j) y de los Multiplicadores de Esfuerzo (EM_i).

El resultado de la evaluación de los factores de escala se resume en la *Tabla 45*.

SF_j	Valores
PREC	3.72
FLEX	2.03
RESL	None
TEAM	5.48
CMM	7.80

Tabla 4.5 Valores de los factores de escala del proyecto (ob. cit SGAD-IMTA)

Los multiplicadores de esfuerzo para el sistema son los siguientes (*Ver Tabla 4.6*):

EM_i	***Ponderaciones***	***Valores***
RCPX	*NOM*	*1.00*
RUSE	*LOW*	*0.95*
PDIF	*LOW*	*0.87*
PERS	*LOW*	*1.26*
PREX	*LOW*	*1.12*
FCIL	*NOM*	*1.00*
SCED		*None*

Tabla 4.6 Valores de los multiplicadores de esfuerzo del proyecto

4.5 Resultados de estimaciones

- **Estimación de SIDIT**

El resultado de la estimación, usando COCOMO II. 1999.0 para el Proyecto SIDIT, son los siguientes: (*Ver Figura 4.7*)

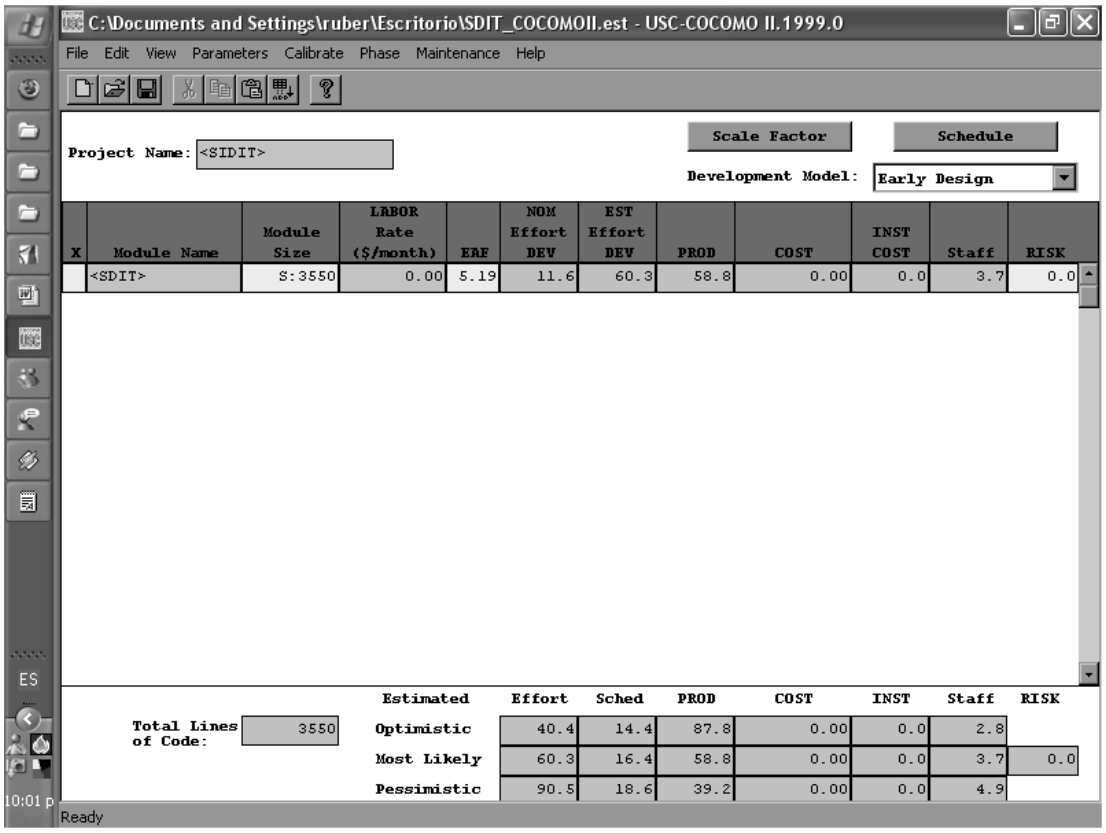


Figura 4.7 Resultados de estimación de SIDIT

Donde podemos observa que COCOMO II presenta tres escenarios:

- ✓ Estimación optimista.
- ✓ Mejor estimación.
- ✓ Estimación pesimista.

Por lo que las estimaciones para SIDIT, tomando en cuenta que se realizó la estimación de todo el proyecto, revelan que:

- ✓ El esfuerzo mejor estimado es de : 60.3 (*personas / mes*)
- ✓ El tiempo de desarrollo mejor estimado es de : 16.4 (meses)
- ✓ Y el número de personas que pueden hacer el desarrollo (mejor estimado) es de : 4 personas

Esto nos revela que el proyecto puede tardar un año y 4 meses en terminarlo, considerando el proyecto como un todo, es decir no estimando por módulos, lo cual nos establece una mejor estimación, pero si recordamos que están en la etapa de diseño temprano, todavía no se tienen bien definidas las características esenciales tales como arquitectura , algunos puntos de diseño apenas se están revisando, tales como el diseño de la base de datos y hay mucha volatilidad en los cambios por parte del cliente.

- **Estimación de SGAD-IMTA**

Los resultados de la estimación del proyecto SGAD-IMTA son (Ver *Figura 4.8*)

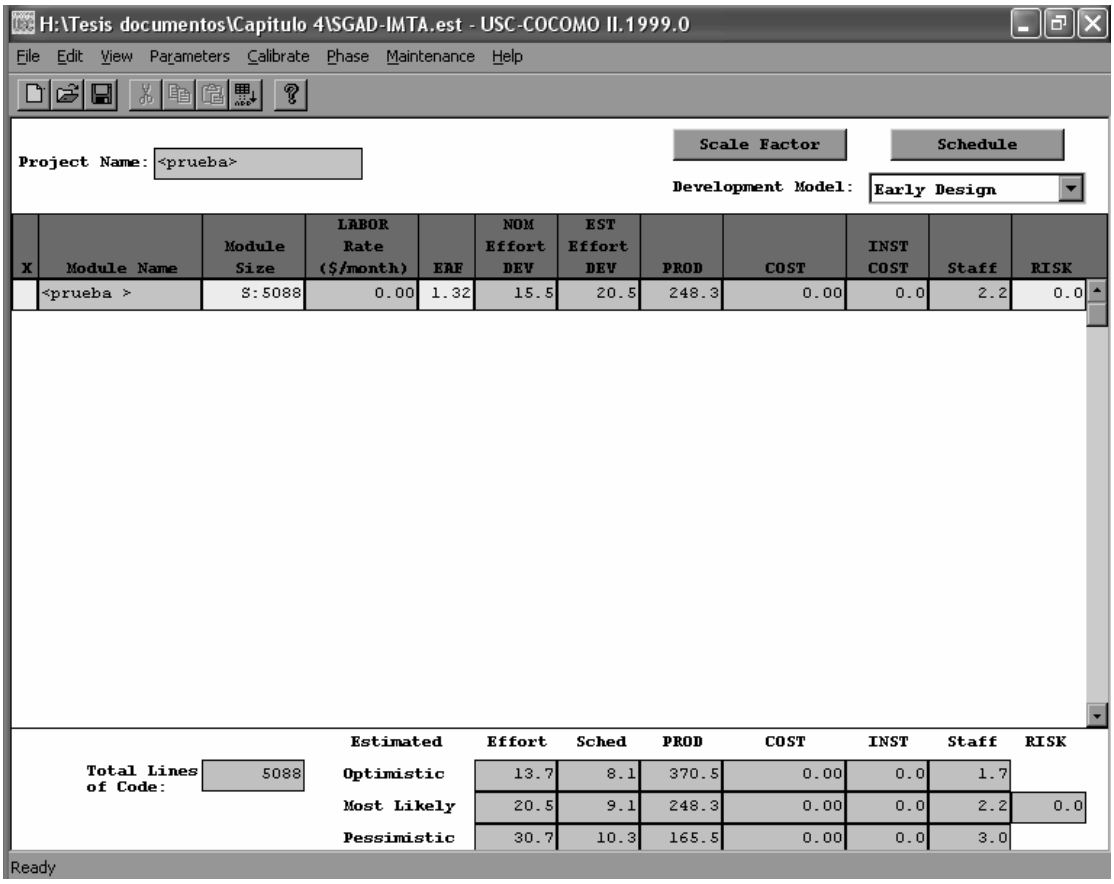


Figura 4.8 Resultado de estimación de SGAD-IMTA

Los resultados de las mejores estimaciones resultaron:

- ✓ El esfuerzo mejor estimado es de: 20.5 (*personas / mes*).
- ✓ El tiempo de desarrollo mejor estimado es de: 9.1 (meses).
- ✓ Y el número de personas que pueden hacer el desarrollo (mejor estimado) es de: 2 personas.

4.6 Aplicación del modelo de estimación COCOMO II con el agregado conceptual Esfuerzo de Desarrollo de una Base de Datos (EDBD)

Ahora aplicaremos a nuestros proyectos de ejemplo nuestra propuesta de estimación con los multiplicadores de Esfuerzo de Desarrollo de una Base de Datos (definidos y discutidos en el ***Capítulo 3***).

Utilizando el cuestionario de Evaluación de la Actividad de Desarrollo de una Base de Datos siguiendo RUP (discutido en el ***Capítulo 3***), para recolectar información de los $\mathbf{EDBD_j}$. (Ver ***Apéndice E***), encontramos los siguientes valores de estimación del apartado.

- ✓ **Valores de EDBD para SIDIT**

Los valores encontrados al aplicar el cuestionario guía al proyecto los resultados fueron los siguientes de *EDBD* (Para ver cuestionarios consultar ***Apéndice F***). *Ver Tabla 4.7*.

Número	$EDBD_{13}$	Ponderación	Valor
1	DRUP	HIGH	1.052
2	CEBD		
3	DINFO		
4	MODNG		
5	MODCU		
6	MODD	NOM	1
7	CDIG		
8	DMOO		
9	CLPMOO		
10	NCLPXR		
11	CLPXC		
12	CLPXRI		
13	SEGBD		

Tabla 4.7 Valores de EDBD

Nota:

Los demás parámetros no se tomaron en cuenta, debido a que el equipo de desarrollo de SIDIT, no siguió un desarrollo apegado al RUP, por ende el equipo necesita, ajustar actividades a RUP para mejorar su eficacia al desarrollar la base de datos. Otro punto importante es que se califica sólo C*onstrucción del Modelo de Datos* (DMODD), por qué este parámetro es importante para cualquier modelo de desarrollo (ya sea un modelo en cascada, un modelo en espiral), por tal motivo se ha diseñado este apartado para considerar este aspecto importante que puede ayudar a disminuir o aumentar el esfuerzo de desarrollo sin seguir RUP.

Por lo tanto, el total de la multiplicadora de EDBD es para SIDITT:

$$\prod_{k}^{k=13} EDBD_k = 1.052$$

Este resultado nos advierte un aumento en el esfuerzo de desarrollar una base de datos sin seguir RUP desde las etapas de inicio del proyecto.

- ✓ **Valores de EDBD para SGAD-IMTA**

Los valores encontrados (*Ver Tabla 4.8*) al aplicar el cuestionario guía al proyecto los resultados fueron los siguientes *EDBD* (Para ver cuestionarios consultar ***Apéndice F***).

Número	$EDBD_{13}$	Ponderación	Valor
1	DRUP	LOW	0.052
2	CEBD	LOW	0.75
3	DINFO	NOM	1
4	MODNG	LOW	0.75
5	MODCU	LOW	0.75
6	MODD	LOW	0.75
7	CDIG	LOW	0.75
8	DMOO	NOM	1
9	CLPMOO	LOW	0.75
10	NCLPXR	NOM	1
11	CLPXC	NOM	1
12	CLPXRI	NOM	1
13	SEGBD	HIGH	1.75

Tabla 4.8 Valores de EDBD

Este equipo tomó en cuenta el RUP como modelo de desarrollo desde sus inicios, por lo tanto se puede proceder a evaluar todos los aspectos relacionados con la Actividad de Desarrollo de una Base de Datos, así que el resultado de la mutiplicatoria de EDBD para SGAD-IMTA es:

$$\prod_{k}^{k=13} EDBD_k = 0.0162$$

Las cifras obtenidas concluyen que hubo una reducción en el esfuerzo de desarrollo de la base de datos.

Ambos resultados se obtienen a partir de una disminución de los valores de los parámetros de esfuerzo del desarrollo de una base , los cuales son referidos a un valor base de %5.20 lo cual es el porcentaje de la Actividad de Desarrollo de una Base de Datos (ver ***Capítulo 3***).

4.6.1 Resultados de estimaciones de los proyectos.

Los resultados de la evaluación del Esfuerzo de Desarrollo de una Base de Datos siguiendo el RUP son:

- ✓ **Para SIDIT**

  - ✓ Esfuerzo de desarrollo para SIDIT es:

Tamaño Estimado= 4160

$$PM_{NS} = 62.08 \ {}^{peronas}\!/_{mes}$$

Esto significa que al momento de no llevar RUP, el aumento en el esfuerzo es considerable, teniendo en cuenta que se necesitan ajustar muchas actividades al proceso unificado, por tal motivo hay un aumento en el esfuerzo de desarrollo en forma general.

- ✓ **Cálculo del esfuerzo de SGAD-IMTA**

Tamaño estimado = 5088

$$PM_{NS} = 19.54 \ {}^{peronas}\!/_{mes}$$

Hay una reducción sensible en el esfuerzo de desarrollo, esto es debido a que se esta siguiendo el proceso unificado, este valor hace que concluyamos que el esfuerzo de desarrollo de un proyecto de software se ve afectado si se sigue un proceso de desarrollo, que reúna las mejores prácticas y filosofías para analizar, diseñar y desarrollar software.

4.7 Situación actual de los proyectos.

En la documentación de ambos proyectos, los equipos de desarrollo muestran sus resultados finales de número de líneas programadas fechas de inicio de los proyectos y en un solo caso se cuenta con la fecha de terminación; además relatan sus experiencias a lo largo del ciclo de vida de los proyectos.

Empezaremos por el proyecto SIDIT, en donde el equipo de desarrollo estuvo haciendo conteos de líneas por funcionalidad programada, es decir; al seguir un ciclo de desarrollo en espiral, se empezó a contar el número de líneas por cada modulo del proyecto compilado (ver *Tabla 4*.9), dando como resultado: 7894 KSLOC.

Estos datos proporcionados son las cifras finales que se contabilizaron hasta la terminación del proyecto, por tal motivo hay una gran diferencia entre el tamaño estimado y el tamaño final, debido a que el tamaño estimado fue obtenido por medio de los Casos de Uso y a lo largo del desarrollo de SIDIT, hubo una gran volatilidad en los requerimientos, por tal motivo se tuvo la necesidad de agregar nuevas funcionalidades al sistema.

Página	No. Líneas
acceso.asp	100
conexion.asp	4
conf_pass.asp	97
dep_sin.asp	870
deps.asp	690
enla_pro.asp	668
errorInicio.asp	50
esc_dep.asp	1955
list_usua.asp	752
manto.asp	593
menuAdmin.asp	159
menuOper.asp	156
Principal.asp	49
RepBit.asp	350
RepEnProp.asp	373
RepExcel.asp	584
Reportes.asp	433
salir.asp	3
valida_session.asp	8
TOTAL	**7894**

Tabla 4.9 Número de líneas

La fecha de inicio de SIDIT fue en 12 de marzo del 2005 y su fecha de terminación, tomando en cuenta la fase de pruebas, es el día 24 de septiembre del 2006, lo cual nos lleva a contabilizar: 17 meses de desarrollo, sin tomar en cuenta días festivos y de asueto.

La estimación calculada del tiempo[5] para SIDIT fue de 16.8 meses, la cual nos da una idea de que este resultado está muy cercana a la realidad, debido a que se tomaron las características del desarrollo de SDIT para el cálculo de estimación de manera correcta. El staff estimado es de 2 personas para el desarrollo, el equipo de desarrollo estuvo conformado por 3 desarrolladores, todos tesitas de la Facultad de Ingeniería, *(ob. cit SIDIT)*.

Para el proyecto SGAD-IMTA, los desarrolladores documentaron que el tiempo consumido ha sido del doble del estimado: 13 meses (fecha de inicio: 14 de abril del 2004 y el documento de SGAD-IMTA tiene como fecha de documentada de avance hasta el 1 de julio del 2005) y el tamaño del proyecto se acerca al estimado pero solo se ha concluido el 70 % del proyecto (4688 KSLOC) *(SGAD-IMTA, 2005)*.

El número de integrantes del staff estimado es el mismo pero el equipo de desarrollo de SGAD-IMTA es un equipo que ha tenido diferentes integrantes, por lo tanto no hay mucha cohesión en el equipo.

Lo que ha ayudado en las estimaciones de SGAD-IMTA es que se ha seguido el RUP como modelo de estimación del proyecto.

[5] Tomando en cuenta el agregado conceptual del esfuerzo de desarrollo de una base de datos.

Capítulo 5

5. Conclusiones y prospectiva.

Como se pudo constatar el uso de una metodología para desarrollar proyectos de software, resulta conveniente, debido a que, al momento de poner en práctica las actividades que sugiere RUP, estamos siguiendo las buenas filosofías de desarrollo de software, por tal motivo esto repercute en los cálculos de las estimaciones de esfuerzo y de tiempo.

La metodología del proceso unificado, la cual está basada en la arquitectura y guiada por el desarrollo de los Casos de Uso , quizá implique un esfuerzo extra al momento de aplicarlo, si se tiene un equipo de desarrollo con poca experiencia siguiendo dicha metodología (tal es el caso visto por los diferentes proyectos realizados por los alumnos en la facultad) , pero al momento de implantarlo y llevarlo a cabo, este proceso basado en el paradigma de la programación a objetos, nos ayuda a prevenir y arreglar las dificultades que se pueden presentar en el ciclo de vida del software. Esta misma idea se extrapola para cuando nos centramos en el desarrollo de una base de datos.

Es por eso que decimos que para el objetivo de: proponer una metodología de estimación que trate de reducir los costos que caracterizan al desarrollo de un producto de software que use una base de datos , ha sido parcialmente cumplida , ya que un proceso de este tipo necesita aun más estudio, tanto por el lado de la Ingeniería de Software como por la parte de la experimentación (véase ***Apéndice C***) debido a que como proponen los autores de COCOMO, el camino para llegar al establecimiento de parámetros que puedan reflejar un verdadero aumento o disminución en el esfuerzo de desarrollo de software durante todo el ciclo de vida de diferentes tipos de proyectos, se deben tener los cimientos teóricos más firmes (por la parte de Ingeniería de Software) , además de contar con datos históricos de mediciones que refuten la idea de que este nuevo driver o multiplicador de costo censa realmente y correctamente una realidad de desarrollo, esto nos lleva a que el mismo apartado que estamos incluyendo , que trata de conocer el esfuerzo de desarrollar una base de datos como parte integral de un proyecto, a un muy buen modelo de estimación de costos COCOMO II.

Este necesita sujetarse a más experimentación, con diferentes tipos de bases de datos, tamaños, relaciones, etc.

En ***EDBD$_{k=13}$*** trata incluir una característica que en estos últimos años ha resultado de gran interés por parte de loas organizaciones que basan su actividades en alguna aplicación de software, y es la seguridad y la integridad de la información, la introducción de este par de apartados aún se considera inclusive en una etapa muy temprana (ver parámetros de ***EDBD*** en ***Capítulo 3***), ya que existe la necesidad de investigar detalladamente aspectos más concretos, tales como la realización de un análisis de entorno, etc.

Por lo tanto nuestro apartado ***EDBD$_{k=13}$*** está muy lejos de poder calcular estimaciones de cualquier proyecto basado en una base de datos, tiene sus limitaciones propias de un modelo de estimación paramétrico .

El precedente de el establecimiento de un apartado de estimación que incluya el desarrollo de una base de datos , que se ha propuesto a lo largo del desarrollo de esta tesis está pensado para ser un punto de partida para el establecimiento de una línea de investigación en donde se pueda recabar la información de los proyectos realizados por parte de estudiantes, maestros y personal de la Facultad , con el fin de proponer un modelo de estimación basado en COCOMO II , propio de la institución educativa , con el objetivo de contar con una herramienta que ayude a la toma decisiones relacionadas con el desarrollo de un proyecto de software, ya sea para fines académicos (proyectos escolares y de tesis) , administrativos y de investigación.

Las métricas asociadas al estudio que se realizó en este trabajo de tesis, aun son muy rudimentarias y carecen de información, es por eso que para la realización de los Puntos de Casos de Uso (ver ***Capítulo 3*** y ***Capítulo 4***), se necesita mucha experiencia en el desarrollo y gestión de proyectos, esto ha ayudado al autor de esta tesis a tener una idea general del campo de la Estimación de proyectos de desarrollo de software, el cual es un tema que nunca acabará.

Los objetivos y sus correspondientes conclusiones de esta tesis pretenden estar orientados al mejoramiento de los procesos de negocio de cualquier organización que desarrolle software, cuyo valor de lo que se vende son las soluciones técnicas, la consultaría y la capacitación de sus productos.

Otro elemento a considerar es el agregar valor a lo que se hace y por eso se entiende la capacidad de creación para re-inventar el mercado, pues para dar soluciones a las empresas muchas veces es necesario crear soluciones novedosas, funcionales y rentables, es por eso que lo visto en este trabajo de tesis trata de hacer que las soluciones que se puedan hallar, sean las más viables tanto desde el punto de vista técnico como económico para un cliente.

La aportación que surge de este trabajo de tesis, a parte de el planteamiento de una línea de investigación, es la necesidad del desarrollo de software que calcule estimaciones de esfuerzo y tiempo pero tomando en cuenta aspectos inherentes al planteamiento, desarrollo e implementación de una base de datos.

El resultado de esta línea de investigación, con su consecuente software puede ser enfocado a pequeñas empresas y corporativos donde el verdadero negocio no esta en las licencias si no en el soporte técnico, con la capacitación de los recursos humanos y la parte de consultaría.

Apéndice A

Puntos de Función (ISO 14143)

Esta métrica se define como una métrica funcional, dado que se enfoca a la funcionalidad que el software proporciona al usuario.

Es una métrica para establecer el tamaño y complejidad de los sistemas informáticos basada en la cantidad de funcionalidad requerida y entregada a los usuarios. Los PF miden el tamaño lógico o funcional de los proyectos o aplicaciones de software basado en los requerimientos funcionales del usuario *(Dekers, 2003)*.

Al analizar el párrafo anterior, podemos entender las características de la métrica:

Tamaño:

Es una métrica o tamaño, no de la calidad con la que se hizo ese software o del valor de ese producto, o del esfuerzo requerido para desarrollarlo, etc.

Aplicaciones:

Mide las aplicaciones del software, no toma en consideración el hardware que estará utilizando, ni la administración del proyecto, ni la documentación, etc.

Funcionalidad:

Se refiere a la capacidad del software para que un usuario pueda realizar transacciones (lectura, escritura, etc.) y el guardar datos.

Usuario:

Quien va a usar el software y no quién lo desarrolló o quién lo diseñó.

Así como existe el metro lineal para medir longitudes, kilogramos masa para medir la cantidad de masa que tiene un cuerpo, los PF son el "metro" o "kilogramo masa" para medir tamaño de una aplicación de software.

El Método Estándar de Análisis de PF ó FPA (Function Point Análisis) nombre formal del método que utiliza los PF para medir el tamaño de un software definido por el International Function Point Users Group (IFPUG[1]) se basa principalmente en la identificación de los componentes del sistema informático en términos de transacciones y grupos de datos lógicos que son relevantes para el usuario en su negocio. A cada uno de estos componentes les asigna un número de puntos por función basándose en el tipo de componente y su complejidad; la sumatoria de esto nos da los PFA (Puntos de Función sin Ajustar). El ajuste es el paso final basándose en las características generales de todo sistema informático que se está contando.

1 Pagina del IFPUG : www.ifpug.org

Procedimiento:

La siguiente figura nos muestra como es el procedimiento de manera general.

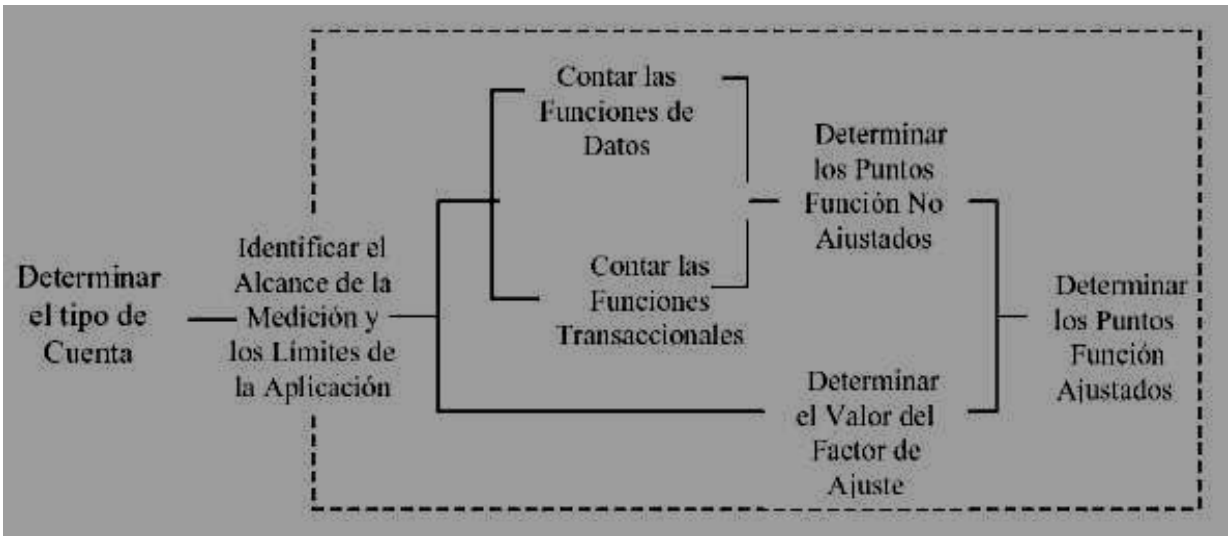


Figura A.1 Procedimiento de conteo de PF

Paso 1) Determinar el tipo de conteo.

Este paso consiste en definir el tipo de conteo entre: desarrollo, mantenimiento de una aplicación ya instalada. Éste es una forma de determinar el objetivo del conteo.

Paso 2) Identificar los alcances de la medición y los límites de la aplicación.

El propósito de una aplicación consiste en dar una respuesta a un problema de negocio. El alcance de la medición define la funcionalidad que va ser incluida en una medición específica y puede abarcar más de una aplicación y establece una frontera de medición.

En este paso sabemos qué vamos a medir pero necesitamos conocer hasta dónde vamos a medir.

La frontera marca el borde entre el proyecto o aplicación a ser medida y las aplicaciones externas. Una vez establecida la frontera, los componentes pueden ser clasificados y contados.

Se define qué es lo externo a la aplicación. Actúa como una membrana a través de la cual los datos procesados por transacciones cruzan la frontera hacia adentro y/o hacia fuera de la aplicación.

Determina los datos lógicos mantenidos por la aplicación, así como por consiguiente facilita la identificación de cuáles datos lógicos son referenciados por la aplicación pero mantenidos por otro sistema.

Las siguientes reglas deben aplicarse para el establecimiento de la frontera:

- ✓ La frontera es determinada basándose en el punto de vista del usuario. Se focaliza en qué el usuario puede entender o describir.

- ✓ La frontera entre aplicaciones relacionadas está basada en áreas funcionales separadas siempre desde el punto de vista del usuario y no en consideraciones técnicas.

- ✓ La frontera establecida por una aplicación existente que está siendo modificada no se influencia por el alcance de la cuenta.

La localización de la frontera de medición entre el software que está siendo medido y otras aplicaciones de software suele ser algo subjetivo y difícil de determinar dónde una aplicación termina y la otra comienza.

Es importante que la frontera sea dibujada con cuidado dado que todo aquel dato que atraviesa la frontera puede potencialmente ser incluido en el alcance de la medición. Para ilustrar como se define una frontera de aplicación titulada Aplicación de Recursos Humanos, obsérvese la *figura A.2*.

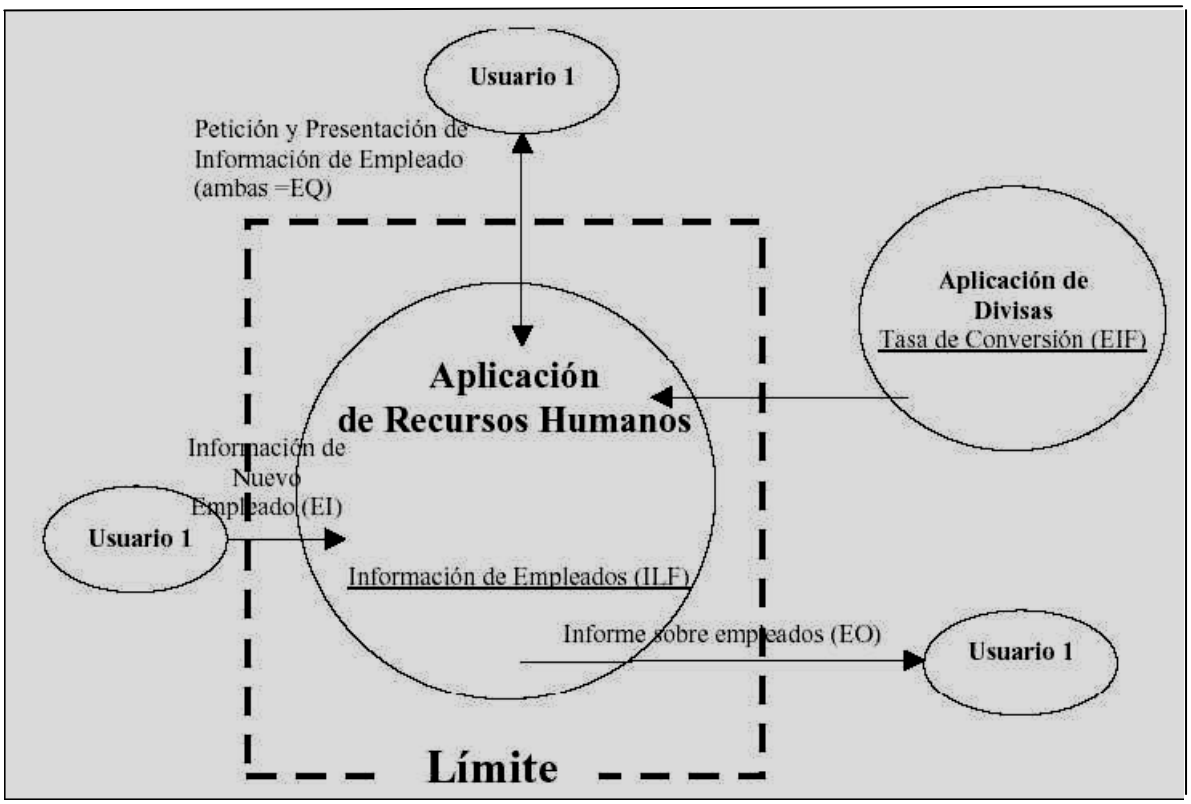


Figura A.2 Frontera de Aplicación de Recursos Humanos

Paso 3) Contar las funciones de datos

Este paso consiste en identificar y contar la capacidad de almacenamiento de los datos.

Las funciones de datos representan las funcionalidades brindadas al usuario para reunir los requerimientos de datos internos y externos a la frontera de la aplicación.

Se distinguen dos tipos de funciones de datos:

- ✓ *Archivo interno lógico (Internal Logical File) ó ILF*

  Es un grupo de datos relacionados que el usuario identifica, cuyo propósito principal es almacenar datos mantenidos a través de alguna transacción que se está considerando en el conteo.
  Dicho grupo de datos se encuentran dentro de la frontera de conteo de la aplicación.

- ✓ *Archivo de Interfaz Externo (External Logical File) ó ELF*
  Es un grupo de datos relacionados y referenciados pero no mantenido por alguna transacción dentro del conteo.

Se puede observar cómo podemos encontrar un Archivo Lógico dentro y fuera de la aplicación en la *figura A.2*.

*Para identificar el **ILF**, se siguen las siguientes reglas*:

Regla 1: El grupo de datos o información de control es un grupo de datos lógicos identificables por el usuario que cubre de manera completa requisitos específicos de éste.

Regla 2: El grupo de datos es mantenido dentro de los límites de la aplicación.

Regla 3: El grupo de datos es mantenido o modificado por medio de un proceso elemental de la aplicación.

Regla 4: El grupo de datos identificado no se ha contado como *ELF* de la aplicación.

*Para identificar un **ELF**, se debe tomar en cuenta las siguientes reglas*:

La principal diferencia entre un archivo lógico interno ILF y una interfase de archivo externa ELF es que el ILF es mantenido dentro de la frontera de la aplicación mientras que el ELF es mantenido por otra aplicación. El ELF es referenciado por la aplicación que se está contando pero no mantenido por ella.

Regla 1: El grupo de datos o información de control es un grupo de datos lógico identificable por el usuario que cubre de manera completa requisitos específicos de éste.

Regla 2: El grupo de datos es referenciado y es externo a la aplicación que está siendo contada.

Regla 3: El grupo de datos no es mantenido por la aplicación que está siendo contada.

Regla 4: El grupo de datos se ha contado como ILF en al menos otra aplicación.

R5: El grupo de datos identificado no ha sido contado como un ILF para la aplicación.

A cada componente identificado se le asigna una complejidad (bajo, medio o alto) considerando principalmente el número de datos, de acuerdo con las siguientes reglas:

Para determinar la complejidad de las funciones de los datos, se basa principalmente en el conteo de *DET (Data Element Type)* y *RET (Record Element Type)*:

Donde para contar un *DET* dentro de los archivos lógicos, está dado por las siguientes reglas:

Regla 1) Contar un *DET* por cada uno de los campos reconocibles por el usuario que componen el archivo lógico ya sea ILF ó ELF.

Regla 2) Los campos que existen como claves foráneas para referenciar otro ILF según los requerimientos, también se consideran como *DET*.

Regla 3) Un campo lógico único, almacenado en varios campos físicos, se cuenta como un único *DET*.

Regla 4) Los campos repetitivos se cuentan como un único *DET* cada uno.

Regla 5) Los campos que aparecen más de una vez por técnicas de implementación se cuentan como un único *DET*.

Regla 6) Los campos que aparecen debido a técnicas de implementación, que no son reconocibles por el usuario, no se cuentan como *DET.*

El conteo de *RET* en un archivo lógico, se define de acuerdo con las siguientes proposiciones:

Regla 1) Contar un *RET* por cada subgrupo de datos elementales reconocibles por el usuario.

Existen dos tipos de subgrupos:

- ✓ ***Opcional****:* son aquellos que el usuario tiene la opción de usar uno o ningún subgrupo durante un proceso elemental que agrega o crea una instancia de datos.

- ✓ ***Obligatorio****:* son subgrupos donde el usuario debe usar al menos un subgrupo de datos.

A la hora de contar los *RET*, una de las siguientes reglas debe aplicar:

Regla 1.2) Cuente como un *R ET* para cada subgrupo opcional u obligatorio de un ILF ó ELF.

Regla 2.2) De no haber subgrupos, cuente el ILF o ELF como un *RET*.

En caso de identificar grupos repetitivos en una entidad, sume 1 RET por cada grupo repetitivo.

Por ejemplo, para una entidad factura donde se identifican los datos de cada línea que se repiten, súmele un *RET* a la entidad factura.

Una vez identificados los *DETy RET* de un fichero lógico, utilice la siguiente tabla para determinar su complejidad. *(Ver Tabla A.1)*

	1 a 19 DETs	20 a 50 DETs	51 o más DETs
1 RET	Baja	Baja	Media
2 a 5 RETs	Baja	Media	Alta
6 o más RETs	Media	Alta	Alta

Tabla A.1 Niveles de complejidad

Paso 4) Contar las funciones trasnsaccionales.

Este paso consiste en identificar y contar la capacidad de realizar operaciones.

Se distinguen tres tipos de funciones de este tipo, los cuales son:

- ✓ *Entrada Externa o EI (External Inputs)*

  Es un proceso cuyo propósito principal es mantener uno o más archivos lógicos internos.

  Los datos pueden venir desde otra aplicación o desde una pantalla de ingreso de datos.

  El objetivo fundamental es mantener uno o más archivos lógicos internos (*ILF*) y/o alterar el comportamiento del sistema.

- ✓ *Salida Externa o EO (External Output)*

  Es un proceso cuyo propósito principal es presentar información al usuario por medio de un proceso lógico diferente al de solo la recuperación de los datos.

  El procesamiento lógico debe contener al menos una fórmula o un cálculo matemático o crear datos derivados de los obtenidos. Un *EO* podría mantener uno o más *ILF* y/o alterar el comportamiento del sistema.

- ✓ *Consulta Externa EQ (External Query)*

  Es un proceso cuyo propósito principal es presentar información al usuario leída de uno o más grupos de datos.

  Cada componente debe ser identificado de acuerdo con una serie de reglas, las cuales se listan a continuación con base en cada uno de las funciones transaccionales.

  A diferencia del EO, el procesamiento lógico no debe contener ninguna fórmula o cálculo matemático, ni tampoco debe crear datos derivados de los obtenidos.

  Por otra parte ningún ILF es mantenido mientras se procesa la acción de un EQ ni tampoco el comportamiento del sistema se ve alterado.

Identificación y asignación de complejidad de un EI.

Para identificar un EI se aplican las siguientes reglas:

Regla 1) Los datos se reciben desde fuera de los límites de la aplicación.

Regla 2) Los datos mantienen un ILF a través de un proceso elemental de la aplicación.

Regla 3) El proceso es la unidad más pequeña de actividad que es significativa para el negocio del usuario final.

Regla 4) El proceso es auto contenido y deja la aplicación que está siendo contada en un estado consistente.

Regla 5) El proceso identificado debe verificar alguna de estas reglas:

- ✓ Su lógica de proceso es única respecto de otras entradas externas de la aplicación.
- ✓ Los elementos de datos identificados son distintos a los de las otras *EI* de la aplicación.
- ✓ Los ficheros lógicos referenciados son diferentes.

Una vez identificada los *EI,* se pasa a la asignación de su complejidad, por medio de la cuenta de DET y de FTR (File Type Referenced).

Encontrando los *DET* de acuerdo con las siguientes reglas:

Regla 1) Contar un *DET* por cada uno de los campos que cruzan las fronteras de la aplicación.

Regla 2) No contar los campos que son recuperados o derivados por el sistema y almacenados en un *ILF* si éstos no cruzan la frontera de la aplicación.

Regla 4) Contar los siguientes elementos como un *DE*T extra cada uno:

- ✓ Definición del comienzo de la ejecución del proceso de entrada (botón de "Aceptar", "Ingresar" o "Enter"). Si hay varias formas, contar una sola.
- ✓ Mensajes de error que pueden desplegarse durante la ejecución del proceso (un *DET* para todos).

Y para contar un *FTR*

Regla 1) Contar un *FTR* por cada fichero (interno o externo) leído/mantenido por el proceso de la entrada.

Una vez identificados los DET y FTR de las entradas, utilice la siguiente tabla para determinar su complejidad. *Ver Tabla A.2*

	1 a 4 DETs	5 a 15 DETs	16 o más DETs
1 FTR	Baja	Baja	Media
2 FTRs	Baja	Media	Alta
3 o más FTRs	Media	Alta	Alta

Tabla A.2 Niveles de Complejidad

Identificación y asignación de complejidad de un EO.

Para La identificación de un EO se acatan las siguientes reglas:

Regla 1) El proceso envía datos información de control.

Regla 2) Los datos o información de control se envían a través de un proceso elemental de la aplicación.

Regla 3) El proceso es la unidad más pequeña de actividad que es significativa para el negocio del usuario final.

Regla 4) El proceso es auto contenido y deja a la aplicación en un estado consistente.

Regla 5) El proceso identificado debe verificar alguna de estas reglas:

- ✓ Su lógica de proceso es única respecto de otras salidas externas de la aplicación.
- ✓ Los elementos de datos identificados son distintos a los de otros EO de la aplicación.
- ✓ Los ficheros lógicos referenciados son distintos.

Regla 6) Debe cumplirse al menos una de las siguientes condiciones:

- ✓ El proceso elemental contiene al menos una fórmula matemática o cálculo.
- ✓ El proceso crea datos derivados.
- ✓ El proceso que genera la salida mantiene algún ILF.
- ✓ El proceso que genera la salida altera el comportamiento del sistema.

Regla 7) La transferencia de datos a otras aplicaciones se cuenta como salidas.

Regla 8) Los informes escritos y online se cuentan como salidas independientes.

Regla 9) Los gráficos se cuentan como una salida cada uno.

Para la identificación de los EQ.

EQ es una combinación de entrada/salida que se obtiene de una búsqueda de datos, no actualiza archivos lógicos y no contiene datos derivados (aquellos que requieren un proceso distinto a búsqueda, edición o clasificación).

Para su identificación se aplican las siguientes reglas:

Regla 1) Una petición entra dentro del límite de la aplicación.

Regla 2) Un resultado sale del límite de la aplicación.

Regla 3) Hay recuperación de datos.

Regla 4) Los datos recuperados no contienen datos derivados.

Regla 5) El proceso lógico no contiene fórmulas matemáticas o cálculos.

Regla 6) El proceso que genera la consulta no mantiene ningún *ILF* ni altera el comportamiento del sistema.

Regla 7) La petición de entrada y el resultado de salida juntos, hacen del proceso la unidad de actividad más pequeña que es significativa para el negocio del usuario final.

Regla 8) El proceso es auto contenido y deja a la aplicación que está siendo contada en un estado consistente.

Regla 9) El proceso no actualiza a algún *ILF*.

Regla 10) El proceso verifica alguna de estas dos reglas:

- ✓ La lógica del proceso sobre la entrada y la salida es única respecto a otras consultas de la aplicación.
- ✓ Los elementos de datos que forman la entrada y la salida son distintos a los de las otras consultas de la aplicación.
- ✓ Los ficheros lógicos referenciados son distintos.

Para la determinación de la complejidad del EO y EQ, se aplican además las siguientes reglas:

La complejidad de las entradas se basa en la cuenta de DET y FTR:

DET: Data Element Type. Contar un DET por cada uno de los campos que cruzan las fronteras de la aplicación (tanto en la entrada como en la salida para el caso de las consultas).

Regla 1) Contar un DET por cada campo reconocible por el usuario, no repetido, que entra a la aplicación y que se requiere para especificar cuándo, qué o como deben ser recuperados los datos de la salida.

Regla 2) Contar un DET por cada campo reconocible por el usuario no repetido que sale de los límites de la aplicación.

Regla 3) Si un dato entra y sale de la aplicación, contarlo una sola vez.

Regla 4) Sumar un DET por todos los mensajes de error del proceso.

Regla 5) Sumar un DET por la iniciación del proceso, solo una vez aunque haya varias formas de hacerlo.

Regla 6) No contar los datos recuperados o derivados por el sistema si no cruzaron los límites de la aplicación.

FTR: File Type Referenced. Contar un FTR por cada fichero (interno o externo) leído por el proceso de la salida o consulta.

Una vez identificados los DET y FTR de la salida, utilice la siguiente tabla para determinar su complejidad (*Ver Tabla A.3*).

	1 a 5 DETs	6 a 19 DETs	20 o más DETs
1 FTR	Baja	Baja	Media
2 a 3 FTRs	Baja	Media	Alta
4 o más FTRs	Media	Alta	Alta

Tabla A.3 Nivel de complejidad

Paso 5) Determinar los puntos de función no ajustados (PFA)

Este paso consiste en sumar el número de componentes de cada tipo conforme a la complejidad asignada y utilizar la siguiente tabla para obtener el total.

Para ello, a cada uno de los elementos se le asigna un peso según su complejidad, de forma tal que multiplicando la cantidad de elementos por su peso correspondiente, y sumando todos estos resultados se obtiene el resultado final.

En la tabla que se adjunta a continuación se especifican los pesos de cada uno de los elementos identificados durante la cuenta.

Parámetro	Complejidad	Peso	Cantidad	Total = cantidad * peso
Ficheros Lógicos Internos	Alta	15		
	Media	10		
	Baja	7		
Ficheros Lógicos Externos	Alta	10		
	Media	7		
	Baja	5		
Entradas	Alta	6		
	Media	4		
	Baja	3		
Salidas	Alta	7		
	Media	5		
	Baja	4		
Consultas	Alta	6		
	Media	4		
	Baja	3		
			Total	Puntos Función = Suma de Totales

Tabla A.4 Pesos de elementos identificados

Paso 6) Determinar el valor de ajuste

El factor de ajuste se obtiene sumando 0.65 a la sumatoria de los grados de influencia de las 14 *Características Generales GSC* (*General System Characteristics*) del sistema, multiplicado por 0.01.

Las 14 GSC del sistema son:

I. Comunicaciones de Datos.
II. Procesamiento Distribuido de Datos.
III. Desempeño.
IV. Configuración fuertemente usada.
V. Tasa de Transacciones.
VI. Entrada de datos en línea.
VII. Eficiencia de usuario final.
VIII. Actualización en línea.
IX. Procesamiento complejo.
X. Re usabilidad.
XI. Facilidad de instalación.
XII. Facilidad de operación.
XIII. Sitios múltiples.
XIV. Facilidad de cambio.

Cada GSC debe ser evaluada en términos de su DI (Degree of Influence), en una escala de 0 a 5:

0: No presente, o no tiene influencia.
1: Influencia incidental.
2: Influencia moderada.
3: Influencia promedio.
4: Influencia significativa.
5: Fuerte influencia.

Paso 7) Determinar los puntos de función ajustados.

Para determinar los puntos de función ajustados (PFA) se multiplican los PF no ajustados por el factor de ajuste VAF (Value Adjustment Factor).

Dentro de este Apéndice, expresamos algunas deficiencias al utilizar los PF como métrica para medir software:

- En un inicio, es necesaria una mayor preparación en PF que en líneas de código.
- Es una medida indirecta, en contraposición a las líneas de código.
- Puede variar fuertemente el número de puntos de función encontrados si no se sigue el estándar, o si se descuida la capacitación.

Tabla de conversión de los Puntos de Función Desajustados (PFD) EN Líneas de Código.

Lenguaje	SLOC/ PF
Ada	71
AI Shell	49
APL	32
Assembly –Basic	320
Assembly-Macro	213
Basic-ANSI	64
Basic-Compiled	91
C	128
C++	55
Cobol (ANSI 85)	91
Database-default	40
Fifth Generation language	4
First Generation language	320
Forth	64
Fortran 77	107
Fortran 95	71
Fourth generation language	20
High level, language	64
HTML 3.0	15
Java	53
Jovial	107
Lisp	64
Machine code	640
Modula 2	80
Pascal	81
PERL	27
Power Builder	16
Prolog	64
Query-default	13
Report generator	80
Second generation language	107
Simulation-default	46
Spreadsheet	6
Third generation language	80
Unix Shell Scripts	107
USR_1	1
USR_2	1
USR_3	1
USR_4	1
USR_5	1
Visual Basic	50
Visual C++	34

Tabla A.5 Tabla de conversión

Apéndice B

COCOMO II

En este apéndice se muestran todas las tablas de ajuste de los parámetros (EM) que toma COCOMO II para los diferentes tipos de proyectos; además de expresiones matemáticas acciónales, para el cálculo de coeficientes definidos en las ecuaciones nominales de PM (**2.1**) y TDEV (**2.5**) correspondientes al ***Capítulo 2: Fundamentos***.

B.1) Ahorro y gasto de software de escala (E)

El comportamiento del valor calculado de la expresión (***2.4***) **E**, nos puede arrojar los siguientes resultados:

Si E < 1.0

El proyecto presenta ahorros de escala. Si el tamaño del producto se dobla, el esfuerzo del proyecto es menor que el doble.

La productividad del proyecto aumenta a medida que aumenta el tamaño del producto. Pueden lograrse algunos ahorros de escala del proyecto con herramientas de proyecto específicas (por ejemplo las simulaciones) pero normalmente es difícil lograrlo. Para proyectos pequeños, fijar costos de salida tales como herramientas a medida
y normas de montaje, e informes administrativos, son a menudo una fuente de ahorro de escala.

Si E = 1.0

Los ahorros y gastos de escala están equilibrados. Este modelo lineal se usa a menudo para la estimación de costos de proyectos pequeños. Se usa para el modelo COCOMO II: *Composición de Aplicaciones*.

Si E>1.0

El proyecto presenta gastos de escala. Esto se debe normalmente a dos factores principales:

1. El crecimiento del gasto en comunicaciones.
2. El gasto en crecimiento de la integración de un gran sistema.

Los proyectos más grandes tendrán más personal y por lo tanto más vías de comunicación interpersonales produciendo gasto. Integrar un producto pequeño como parte de uno más grande requiere no sólo el esfuerzo de desarrollar el producto pequeño sino también el gasto adicional en la integración de los proyectos.

El exponente E se obtiene mediante los denominados factores de escala. La selección de factores de escala se basa en la razón de que ellos son un recurso significante de variación exponencial en un esfuerzo o variación de la productividad del proyecto. Cada factor de escala tiene un rango de niveles de valores desde ***muy bajo*** hasta ***extra alto***.

Cada nivel de valores tiene un peso, *SF*, y el valor específico del peso se llama factor de escala. Un factor de escala de un proyecto, SF_j (*Tabla B.1*) se calcula sumando todos los factores y se usa para determinar el exponente de escala, E.

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

Tabla B.1 Factores de escala para el Modelo de COCOMO II para los modelos de Diseño Temprano y Post-arquitectura

Una nota importante que menciona el manual de COCOMO II Versión 99 es que los factores de escala PREC y FLEX capturan las características de los modelos: Orgánico, Semi orgánico y Embebido del original COCOMO 81.

B.2) Tablas de ajuste de los multiplicadores de esfuerzo (EM) para modelo de Diseño Temprano.

PERS (Capacidad personal del desarrollador)

	Extra Low	Very Low	Low	Nominal	High	Very High	Extra High
Sum of ACAP, PCAP, PCON Ratings	3, 4	5, 6	7, 8	9	10, 11	12, 13	14, 15
Combined ACAP and PCAP Percentile	20%	39%	45%	55%	65%	75%	85%
Annual Personnel Turnover	45%	30%	20%	12%	9%	5%	4%

RCPX (Confiabilidad y complejidad del producto)

	EXTRA Low	VERY Low	LOW	NOM	HIGH	VERY High	VERY High
Sum of RELY, DATA, CPLX, DOCU Ratings	5, 6	7, 8	9 - 11	12	13 - 15	16 - 18	19 - 21
Emphasis on reliability, documentation	Very little	Little	Some	Basic	Strong	Very Strong	Extreme
Product complexity	Very simple	Simple	Some	Moderate	Complex	Very complex	Extremely complex
Database size	Small	Small	Small	Moderate	Large	Very Large	Very Large

RUSE (Re usabilidad)

	Very Low	Low	Nominal	High	Very High	Extra High
RUSE		None	across project	across pro-gram	across product line	across mul-tiple product lines

PDIF (Dificultad de la plataforma)

	Low	Nominal	High	Very High	Extra High
Sum of TIME, STOR, and PVOL ratings	8	9	10 - 12	13 - 15	16, 17
Time and storage constraint	□ 50%	□ 50%	65%	80%	90%
Platform volatility	Very stable	Stable	Somewhat volatile	Volatile	Highly volatile

PREX (Experiencia del Personal)

	Extra Low	Very Low	Low	Nomina l	High	Very High	Extra High
Sum of AEXP, PEXP, and LTEX ratings	3, 4	5, 6	7, 8	9	10, 11	12, 13	14, 15
Applications, Platform, Language and Tool Experi-ence	≤ 3 mo.	5 months	9 months	1 year	2 years	4 years	6 years

FCIL (Instalaciones)

	Extra Low	Very Low	Low	Nominal	High	Very High	Extra High
Sum of TOOL and SITE ratings	2	3	4, 5	6	7, 8	9, 10	11
TOOL support	Minimal	Some	Simple CASE tool collection	Basic life-cycle tools	Good; moderately integrated	Strong; moderately integrated	Strong; well integrated
Multisite conditions	Weak support of complex multisite development	Some support of complex M/S devel.	Some support of moderately complex M/S devel.	Basic support of moderately complex M/S devel.	Strong support of moderately complex M/S devel.	Strong support of simple M/ S devel.	Very strong support of collocated or simple M/S devel.

SCED (Calendario requerido para el desarrollo)

	Very Low	Low	Nominal	High	Very High	Extra High
SCED	75% of nominal	85%	100%	130%	160%	

Para mayor detalle de los ajustes ver Manual de COCOMO II Versión 99, referido en la bibliografía del ***Capítulo 2***.

B.3) Tablas de ajuste para el Modelo Post –Arquitectura

Tablas de ajuste de los multiplicadores de esfuerzo (EM) Para diseño Post-Arquitectura.

- **Products Factors**

RELY (Fiabilidad del software)

	Very Low	Low	Nominal	High	Very High	Extra High
RELY	slight incon venience	low, easily recoverable losses	moderate, easily recov erable losses	high finan cial loss	risk to human life	

DATA (Tamaño de la Base de Datos)

	Very Low	Low	Nominal	High	Very High	Extra High
DATA		DB bytes/ Pgm SLOC < 10	10 ≤ D/P < 100	100 ≤ D/P < 1000	D/P ≥ 1000	

RUSE (Re usabilidad)

	Very Low	Low	Nominal	High	Very High	Extra High
RUSE		none	across project	across pro gram	across prod uct line	across mul tiple prod uct lines

DOCU (Documentación de los ciclos de vida del software)

	Very Low	Low	Nominal	High	Very High	Extra High
DOCU	Many life-cycle needs uncovered	Some life-cycle needs uncovered.	Right-sized to life-cycle needs	Excessive for life-cycle needs	Very exces sive for life-cycle needs	

- **Plataform Factors**

TIME (Tiempo de ejecución)

	Very Low	Low	Nominal	High	Very High	Extra High
TIME			≤ 50% use of available execution time	70%	85%	95%

STOR (Restricciones de almacenamiento)

	Very Low	Low	Nominal	High	Very High	Extra High
STOR			≤ 50% use of available storage	70%	85%	95%

PVOL (Volatilidad de la plataforma)

	Very Low	Low	Nominal	High	Very High	Extra High
PVOL		major change every 12 mo.; minor change every 1 mo.	major: 6 mo.; minor: 2 wk.	major: 2 mo.; minor: 1 wk.	major: 2 wk.; minor: 2 days	

- **Personel Factors**

ACAP (Capacidad del analista)

	Very Low	Low	Nominal	High	Very High	Extra High
ACAP	15th percentile	35th percentile	55th percentile	75th percentile	90th percentile	

PCAP (Capacidad del programador)

	Very Low	Low	Nominal	High	Very High	Extra High
PCAP	15th percentile	35th percentile	55th percentile	75th percentile	90th percentile	

AEXP (Experiencia del Programador)

	Very Low	Low	Nominal	High	Very High	Extra High
AEXP	2 months	6 months	1 year	3 years	6 years	

LTEX (Experiencia en el lenguaje)

	Very Low	Low	Nominal	High	Very High	Extra High
LTEX	2 months	6 months	1 year	3 years	6 year	

PCON (Continuidad del personal)

	Very Low	Low	Nominal	High	Very High	Extra High
PCON	48% / year	24% / year	12% / year	6% / year	3% / year	

- **Project Factors**

TOOL (Herramientas de desarrollo)

	Very Low	Low	Nominal	High	Very High	Extra High
TOOL	edit, code, debug	simple, fron tend, back end CASE, little integra tion	basic lifecy cle tools, moderately integrated	strong, mature life cycle tools, moderately integrated	strong, mature, pro active lifecy cle tools, well inte grated with processes, methods, reuse	

SITE (Desarrollo multi sitios)

	Very Low	Low	Nominal	High	Very High	Extra High
SITE: Communica tions	Some phone, mail	Individual phone, FAX	Narrowband email	Wideband electronic communica tion.	Wideband elect. comm, occasional video conf.	Interactive multimedia

SCED (Calendario requerido para el desarrollo)

	Very Low	Low	Nominal	High	Very High	Extra High
SCED	75% of nom inal	85%	100%	130%	160%	

Apéndice C

Calibración del agregado conceptual EDBD (Esfuerzo de Desarrollo de una Base de Datos).

La calibración del apartado conceptual **$EDBD_{k=13}$** sigue, lo propuesto por los autores de COCOMO II en el apartado de Calibración *(COCOMO II, 2000)*, donde se explica que en algunas ocasiones, el modelo de estimación posiblemente no tome en cuenta otros importantes factores que puedan aumentar el costo de la estimación de esfuerzo o tiempo de desarrollo. (*ob. cit COCOMO II)*

Debido a tal necesidad de inclusión de parámetros que evalúen el esfuerzo de desarrollo de una base de datos, desarrollamos un modelo estático el cual acata trata de acatar el modelo de construcción que propone COCOMO II (ver *figura 1.C*), hay limitaciones en este trabajo de tesis, en el sentido de que falta más datos experimentales, al momento de pasar a examinar el modelo propuesto.

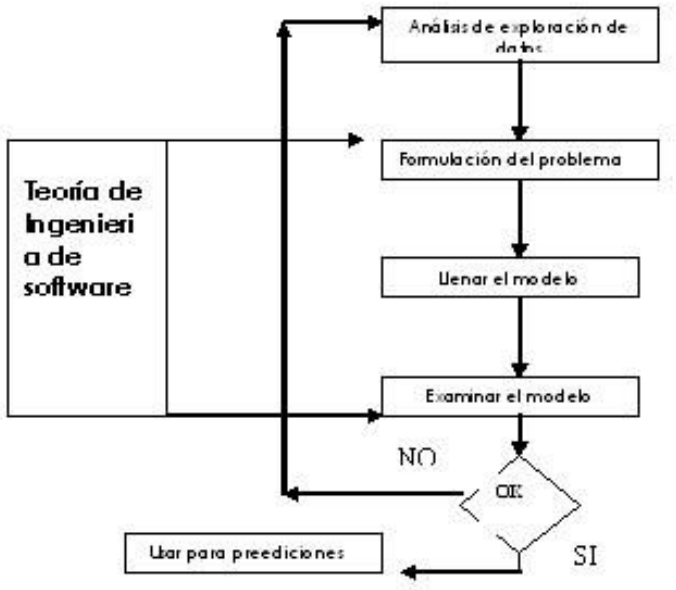


Figura 1.C Proceso de construcción del modelo Físico

La calibración que es la que corresponde al paso 3 de nuestro método de modelado propuesto por COCOMO II, haremos una regresión lineal multivalorada, según la primer propuesta de estimación que hicieron los autores de COCOMO II hicieron en la calibración de 1997 al modelo, debido a que es nuestra primera aproximación al cálculo de estimaciones de nuestro modelo **PM´**.

Según la calibración de 1997 de COCOMO II , podemos hacer lineal el modelo por medio de la aplicación de logaritmos a la expresión de *Estimación de esfuerzo* (PM) (definida en el ***Capítulo 2***), de la siguiente manera:

$$PM_{NS} = A \times Tamaño^{E} \times \prod_{i=1}^{n} EM_i \qquad \textbf{...(2.2)}$$

Aplicando logaritmos y sus propiedades, la ecuación resultante queda de esta manera:

$$\ln(PM) = \ln(A) + \beta_1 \ln(Tamaño) + \beta_2 SF_1 \ln(Tamaño) + ... + \beta_6 SF_5 \ln(Tamaño) + ... + \beta_7 \ln(EM_1) + ... + \beta_{23} \ln(EM_{17}) + \varepsilon$$

Donde ln(A)= β_0

De acuerdo con *(Chulani et al , 1997)*se uso una muestra de 86 proyectos y se utilizó la técnica de regresión lineal, ya que el modelo quedaba simplificado de acuerdo al modelo de regresión lineal :

$$y = \beta_0 + \beta_1 X_{i1} + ... + \beta_k X_{ik} + \varepsilon_i \qquad \textbf{...(C.2)}$$

Para encontrar los valores de predicción β_i , los cuales dieron lugar a la calibración del modelo de COCOMO II para el año de 1997.

Esta misma idea la aplicamos para calibrar nuestros propios multiplicadores de esfuerzo de desarrollo de base de datos (**$EDBD_{k=13}$**), si el modelo de COCOMO original está calculando la estimación de un proyecto de acuerdo con un 100% de trabajo realizado, nuestro EDBD solo representa un 5.29% de la actividad total del desarrollo de software (con base a lo discutido en el ***Capítulo 3***), entonces **$EDBD_{k=13}$** lo podemos calibrar de acuerdo al porcentaje antes mencionado, como un módulo a parte agregado a COCOMO II , tomando en cuenta de que al aplicar el cálculo de esfuerzo con el agregado de la base de datos, se debe hacer un ajuste de acuerdo con los nuevos porcentajes que implica el anexar este apartado al modelo original.

Por lo tanto si aplicamos a nuestra ecuación (EDBD) (definida en el ***Capítulo 3***)

Aplicamos logaritmos a la ecuación y respectivamente hacemos uso de las propiedades de los logaritmos, queda definida la siguiente ecuación:

$$\ln(EDBD) = \beta_1 \ln(EM_1) + \beta_2 \ln(EDBD_{12}) + \ldots + \beta_{13} \ln(EDBD_{13}) \quad \textbf{...(C3)}$$

En virtud de lo anterior esta ecuación la podemos despejar para que queden todos los elementos de EDBD como función de los demás parámetros y después aplicaremos el método de regresión lineal *(Mendenhall, 1997)* para encontrar cada uno de los factores de predicción correspondientes a los EDBD que estamos proponiendo.

El número de proyectos para dar la calibración inicial será de 3 proyectos, los cuales algunos están en desarrollo y otros están terminados.

La información que se obtuvo de estos sistemas, fue autorizada por los desarrolladores.

Cuestionarios de EDBD.

A continuación se presentan los cuestionarios resueltos por los equipos de desarrollo, para evaluar el **$EDBD_{k=13}$** de cada uno de sus proyectos utilizados en nuestro proyecto.

Cuestionario del Modulo: Esfuerzo de Desarrollo de Bases de Datos.

Nombre del proyecto: *Sistema de Documentación de la Infraestructura de Telecomunicaciones*

Contestó: Sierra Moncayo José Pablo (***programador y diseñador de la base de datos***)

Subraye o sombreé la opción que mas le parezca adecuada. Si no sabe alguna respuesta, continúe con la siguiente pregunta.

- ✓ **Evaluación del Modelo conceptual**

1) ¿Se ha seguido el RUP como guía de desarrollo del sistema?

Si	***No***

Si su respuesta fue afirmativa, por favor pasar a contestar las demás preguntas. Si su respuesta es negativa favor de contestar la *pregunta* 6 de este cuestionario.

2) ¿Cómo calificaría la comunicación entre el equipo de desarrollo de la base de datos y el equipo de desarrollo de aplicaciones?

Deficiente	***Eficiente***	***Muy eficiente***

3) ¿Cómo considera la disposición de información para los requerimientos de la base de datos?

Moderada	***Moderada***	***Extensiva***

4) ¿En qué grado se ha analizado y construido el Modelo de Negocios?

Deficiente	***Suficiente***	***Totalmente***

5) ¿En qué grado se ha analizado y construido el Modelo de Casos de Uso?

Deficiente	***Suficiente***	***Totalmente***

6) ¿En qué grado se ha completado el Modelo de Diseño de la base de datos?

Deficiente	***Suficiente***	***Totalmente***

- ✓ **Evaluación del Modelo lógico**

7) ¿En qué grado se ha construido el diagrama de clases?

Deficiente	***Suficiente***	***Totalmente***

8) El diseño del Modelo de Objetos, captura las clases, atributos, relaciones, interacciones que conforman el sistema de manera:

Deficiente	***Eficiente***	***Muy Eficiente***

9) ¿Se han encontrado todas las clases, atributos y relaciones pertenecientes al Modelo de Objetos?

Deficiente	***Suficiente***	***Totalmente***

10) ¿Cómo considera el nivel de complejidad de las relaciones pertenecientes a las relaciones persistentes?

Muy Fácil	***Fácil***	***Difícil***

- ✓ **Evaluación del Modelo físico**.

11) ¿Qué grado de complejidad tiene las consultas a la base?

Muy Moderada	***Moderada***	***Extensiva***

12) ¿Cómo definiría la complejidad que tienen las restricciones en el lenguaje anfitrión para asegurar la integridad de la base de datos?

Muy Fácil	***Fácil***	***Muy difícil***

13) ¿Cómo considera el grado la seguridad en la base de datos?

Moderada	***Alguna***	***Extensiva***

Cuestionario del Módulo: Esfuerzo de Desarrollo de Bases de Datos.

✓ Nombre del proyecto: **Sistema de Gestión y Adquisición Automatizada de Información del Sistema de Medición de la Red de Agua Potable**.

Contesto: Equipo de Desarrollo de SGAD-IMTA.

✓ Subraye ó sombree la opción que mas le parezca adecuada. Si no sabe alguna respuesta, continúe con la siguiente pregunta.

✓ **Evaluación del Modelo conceptual**

1) ¿Se ha seguido el RUP como guía de desarrollo del sistema?

Si	***No***

Si su respuesta fue afirmativa, por favor pasar a contestar las demás preguntas. Si su respuesta es negativa favor de contestar la *pregunta 6* de este cuestionario.

2) ¿Cómo calificaría la comunicación entre el equipo de desarrollo de la base de datos y el equipo de desarrollo de aplicaciones?

Deficiente	***Eficiente***	***Muy eficiente***

3) ¿Cómo considera la disposición de información para los requerimientos de la base de datos?

Moderada	***Moderada***	***Extensiva***

4) ¿En que grado se ha analizado y construido el Modelo de Negocios?

Deficiente	***Suficiente***	***Totalmente***

5) ¿En que grado se ha analizado y construido el Modelo de Casos de Uso?

Deficiente	***Suficiente***	***Totalmente***

6) ¿En que grado se ha completado el Modelo de Diseño de la base de datos?

Deficiente	***Suficiente***	***Totalmente***

- ✓ **Evaluación del Modelo lógico**

7) ¿En que grado se ha construido el diagrama de clases?

Deficiente	***Suficiente***	***Totalmente***

8) El diseño del Modelo de Objetos, captura las clases, atributos, relaciones, interacciones que conforman el sistema de manera:

Deficiente	***Eficiente***	***Muy Eficiente***

9) ¿Se han encontrado todas las clases, atributos y relaciones pertenecientes al Modelo de Objetos?

Deficiente	***Suficiente***	***Totalmente***

10) ¿Cómo considera el nivel de complejidad de las relaciones pertenecientes a las relaciones persistentes?

Muy Fácil	***Fácil***	***Difícil***

- ✓ **Evaluación del Modelo físico**.

11) ¿Qué grado de complejidad tiene las consultas a la base?

Muy Moderada	***Moderada***	***Extensiva***

12) ¿Cómo definiría la complejidad que tienen las restricciones en el lenguaje anfitrión para asegurar la integridad de la base de datos?

Muy Fácil	***Fácil***	***Muy difícil***

13) ¿Cómo considera el grado la seguridad en la base de datos?

Moderada	***Alguna***	***Extensiva***

Apéndice D

Cuestionarios de recaudación de información del proyecto : Sistema de Gestión y Adquisición Automatizada de Información del Sistema de Medición de la Red de Agua Potable (SGAD-IMTA)

- ✓ **Para la obtención del tamaño de SGAD-IMTA.**

- ✓ **Modelo de re uso**

1) ¿Existe software que se haya desarrollado previamente y que pueda volver a usarse para el proyecto?

Si	**No**

Si su respuesta fue afirmativa:

2) ¿Cómo calificaría la claridad del código en función a la comprensión de la :

Estructura:

Muy Baja	Baja	Nominal	Alta	Muy Alta

Claridad de la aplicación:

Muy Baja	*Baja*	*Nominal*	*Alta*	*Muy Alta*

Auto descripción:

Muy Baja	Baja	Nominal	Alta	Muy Alta

3) ¿Qué tanto necesitaría para entender los módulos desarrollados previamente?

Nada	Revisión básica del modulo y de la documentación	Pruebas y evaluación del modulo y estudio de la documentación	Pruebas y evaluaciones considerables del modulo y del estudio de la documentación	Pruebas y evaluaciones exhaustivas del modulo y del estudio de la documentación

4) Si alguna vez ha visto este software: ¿Cómo diría que resulta para usted?

Completamente familiar	Muy familiar	Algo familiar	Considerablemente familiar	Muy desconocido	Completamente desconocido

5) ¿Cómo calificaría el proyecto?

Nuevo: Comenzando el desarrollo desde cero	Adaptado, con cambios a un software pre desarrollado	Reutilizado, con módulos inalterados de software existente.	Software off the shelf (OTS)

6) ¿Cómo calificaría el proyecto?

Nuevo: Comenzando el desarrollo desde cero	Adaptado, con cambios a un software pre desarrollado	Reutilizado, con módulos inalterados de software existente.	Software off the shelf (OTS)

- **Cuestionario para la evaluación de factores de escala de COCOMO II.**

Cuestionario para la obtención de factores de escala de COCOMO II

Nombre del proyecto: Sistema de Documentación de la Infraestructura de Telecomunicaciones

Subraye o sombreé la opción que más le parezca adecuada. Si no sabe alguna respuesta, continúe con la siguiente.

1) ¿Qué proyectos similares existen?

Sin precedentes	Alguno que otro similar	Pocos precedentes	Varios precedentes	Muchos precedentes	Producto muy comercial

2) ¿Cómo calificaría la comprensión organizacional de los objetivos del producto ¿

General	Considerable	Muy buena

3) ¿Cómo calificaría su experiencia con proyectos similares a este?

Moderada	Considerable	Extensiva

4) ¿Hay necesidad de desarrollo concurrente asociado a hardware e a nuevos procedimientos operacionales para este sistema?

Moderada	Alguna	Extensiva

5) ¿Hay necesidad de innovar algoritmos o arquitecturas de procesamientos de datos para este sistema?

Mínima	*Alguna*	*Considerable*

De acuerdo con los requisitos que proporciona el cliente:

6) ¿Qué tan flexible es el proyecto de acuerdo con la configuración del software o requerimientos preestablecidos?

Muy flexible	Flexible	Poco Flexible

7) ¿Qué tan flexible es el proyecto de acuerdo con la configuración de la interfaz del sistema?

Muy Flexible	Flexible	Poco Flexible

8) ¿Qué tan flexible es el proyecto de acuerdo con la configuración y los requerimientos del sistema como recompensa a la finalización temprana del proyecto?

Muy Flexible	Flexible	Poco Flexible

9) ¿Se lleva propiamente un Plan de Administración de Riesgos?

No	Poco	Algo	De manera muy general	En su mayor parte	Completa mente

10) El calendario y el presupuesto ¿Son compatibles con el Plan de Administración de Riesgos?

No	Poco	Algo	De manera muy general	En su mayor parte	Completamente

11) ¿Cuál es el porcentaje de calendario de desarrollo reservado para definir y establecer la arquitectura según los objetivos del sistema?

5	10	17	25	33	40

12) Porcentaje de arquitectura de software disponible para este proyecto es de:

20 %	40%	60%	80%	100%	120%

13) Emplean herramientas de soporte disponibles para resolver factores de riesgo, aspectos de desarrollo y verificación de la arquitectura:

Ninguna	Una	Algunas	Varias	Muchas	Incontables

14)
¿Qué tanta incertidumbre hay en los aspectos clave de la arquitectura, tales como: misión del proyecto, interfaz del usuario, hardware, tecnología usada, desempeño?

Extrema	Poca	Considerable	Alguna	Muy poca	Casi nada
20%	40%	60%	80%	100%	120%

15) ¿Cuál es el número de los factores de riesgo y su criticidad en el proyecto?

>10	5-10	2-4	1	>5	<5

16) ¿Cómo podría definir las interacciones y la disponibilidad para trabajar del equipo de desarrollo?

Muy difíciles	Un poco difíciles	Básicamente cooperativas	Mucha cooperación	Altamente cooperativas	Disponibilidad y cooperación en todo

17) ¿En qué nivel equivalente al proceso de madurez sitúa al equipo de desarrollo?

Muy bajo	Bajo	Nominal	Alto	Muy alto	Extra alto

18) ¿Qué tanta consistencia hay entre los objetivos y la cultura de todo el equipo de desarrollo?

Casi nada	Poca	Básica	Considerable	Fuerte	Completamente

19) ¿Qué tanta habilidad y deseo existe entre el equipo de desarrollo para homogenizar los objetivos de sus integrantes?

Casi nada	Poca	Básica	Considerable	Fuerte	Completamente

20) ¿Qué tanta experiencia de trabajo en este equipo de desarrollo tienen sus integrantes?

Ninguna	Muy poca	Poca	Básica	Considerable	Extensiva

21) ¿Cual será la capacidad de los miembros del equipo para cumplir sus compromisos y alcanzar una visión compartida?

Ninguna	Muy poca	Poca	Básica	Considerable	Extensiva

Apéndice E

Diagramas.

A continuación se muestran los diagramas de los proyectos que se aplicaron como casos prácticos en el ***Capítulo 4*** , se usaron para estimar los tamaños de los proyectos de acuerdo con el conteo de Puntos de Casos de Uso.

El material expuesto está autorizado por los equipos de desarrollo para su uso exclusivo de investigación.

1. **Diagramas SIDIT**

Los diagramas de SIDIT son los siguientes:

1 **Diagrama entidad-relación.**
2 **Diagrama lógico.**
3 **Diagrama físico de la base de datos.**

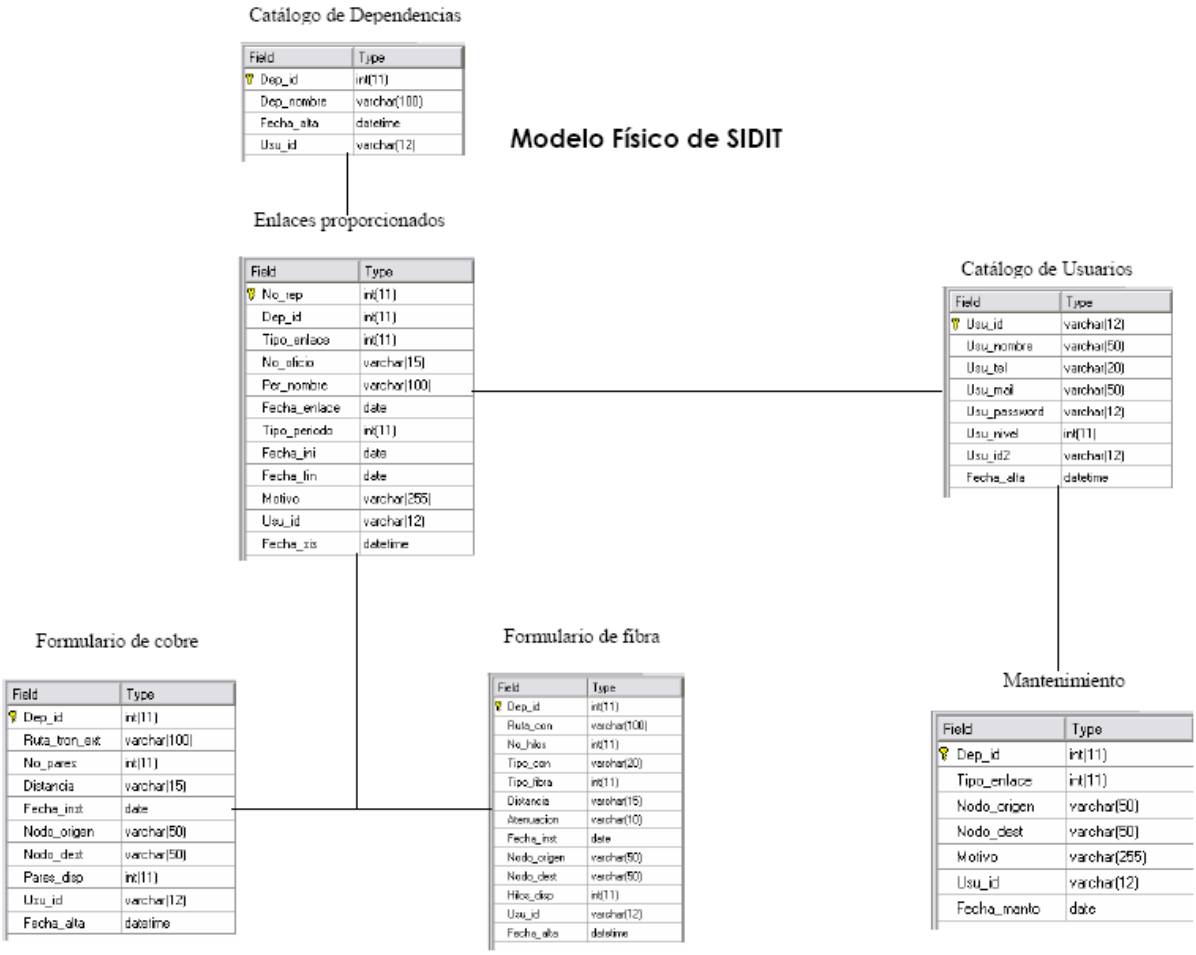
2. **Diagramas de SGAD-IMTA.**

Los diagramas de SGAD-IMTA se listan a continuación:

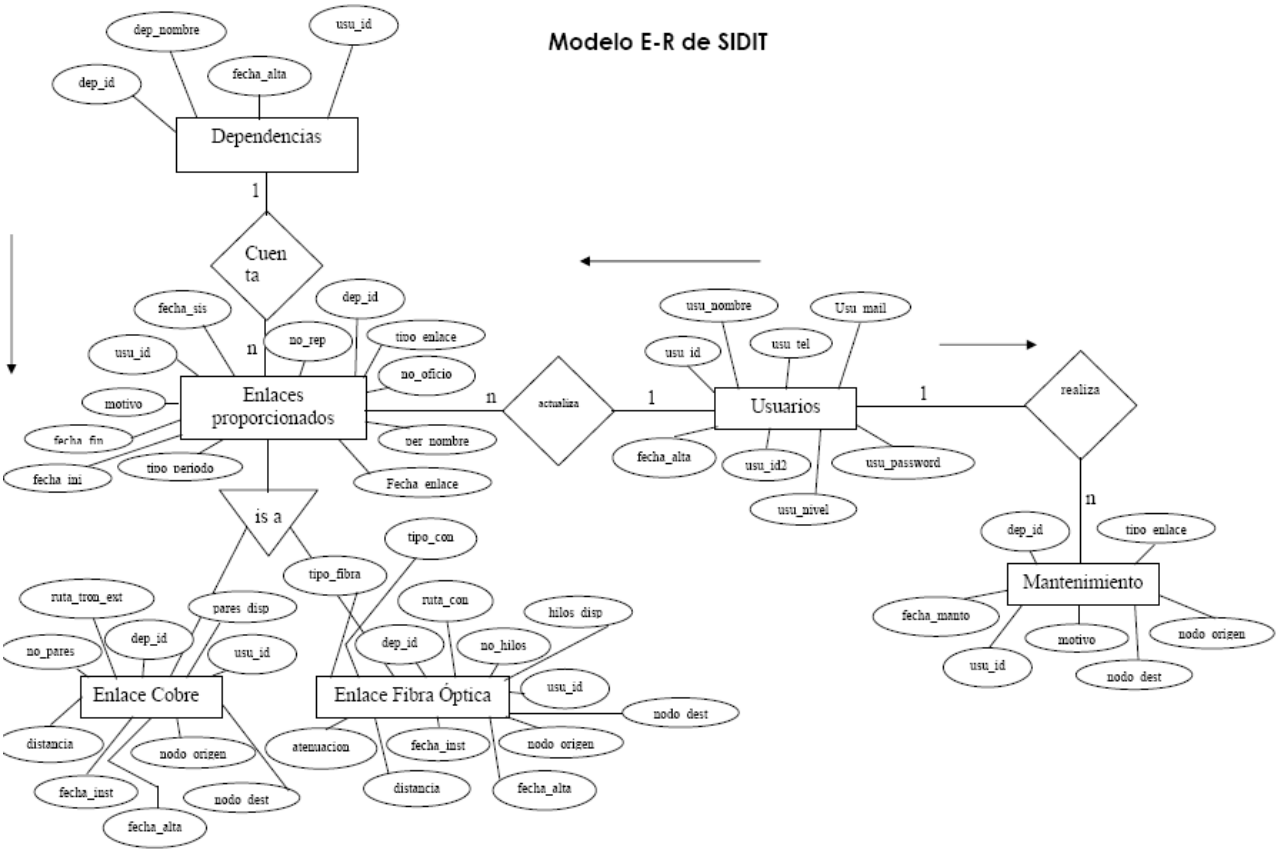
1 **Diagrama de Modelo de Datos**.

Los diagramas se muestran en las siguientes páginas de este ***Apéndice E.***

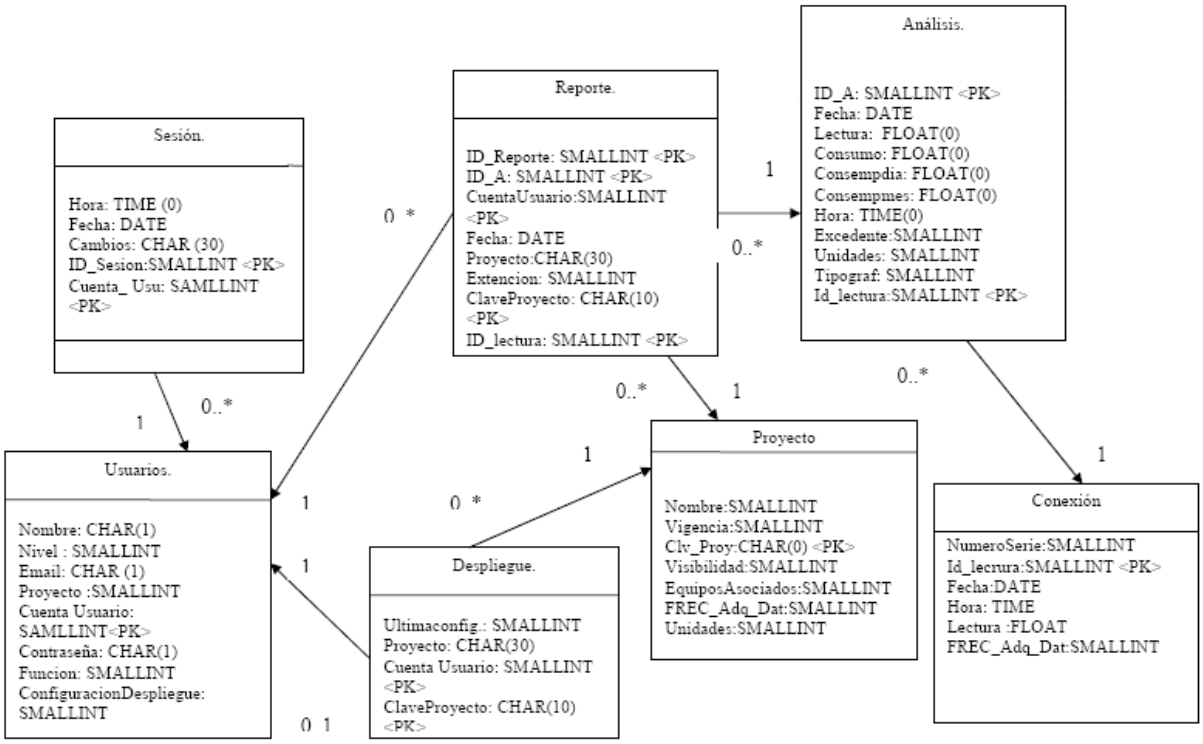
Modelo Físico de SIDIT



Modelo E-R de SIDIT



Modelo DE Datos del Sistema SGAD-IMTA



Glosario de términos

Mainframe

Macro computador. En la actualidad se utiliza esta palabra para referirse a los grandes ordenadores. Un ejemplo típico sería la arquitectura 390 de IBM. Es decir, máquinas capaces de gestionar muchos terminales y unidades periféricas de memoria con capacidad para varios gigabytes. Con el aumento de potencia de los llamados mini computadoras, la frontera entre éstos y los mainframes está cada vez menos clara. Originalmente, mainframe no era sino el armario metálico que contenía la unidad central de los grandes ordenadores. (Definición tomada de http://www.mastermagazine.info/definicion/5660.php)

Middleware

Capa media se refiere a la capa 2 del modelo OSI denominada: Capa de enlace. Este nivel proporciona facilidades para la transmisión de bloques de datos entre dos estaciones de red. Esto es, organiza los 1's y los 0's del Nivel Físico (capa 1) en formatos o grupos lógicos de información. Para:

- ✓ Detectar errores en el nivel físico.
- ✓ Establecer esquema de detección de errores para las retransmisiones o re configuraciones de la red.
- ✓ Establecer el método de acceso que la computadora debe seguir para transmitir y recibir mensajes. Realizar la transferencia de datos a través del enlace físico.
- ✓ Enviar bloques de datos con el control necesario para la sincronía.
- ✓ En general controla el nivel y es la interfaces con el nivel de red, al comunicarle a éste una transmisión libre de errores.

Calidad del software.

Entiéndase como *calidad del software* como: el conjunto de cualidades que lo caracterizan y que determinan su utilidad y existencia.

La calidad es sinónimo de eficiencia, flexibilidad, corrección, confiabilidad, mantenibilidad, portabilidad, usabilidad, seguridad e integridad.

La calidad del *software* es medible y varía de un sistema a otro o de un programa a otro. Un *software* elaborado para el control de naves espaciales debe ser confiable al nivel de "cero fallas"; un *software* hecho para ejecutarse una sola vez no requiere el mismo nivel de calidad; mientras que un producto de *software* para ser explotado durante un largo período (10 años o más), necesita ser confiable, mantenible y flexible para disminuir los costos de mantenimiento y perfeccionamiento durante el tiempo de explotación.

La calidad del software puede medirse después de elaborado el producto. Pero esto puede resultar muy costoso si se detectan problemas derivados de: imperfecciones en el diseño, por lo que es imprescindible tener en cuenta tanto la obtención de la calidad como su control durante todas las etapas del ciclo de vida del *software*.

Para mayor información acerca de calidad de software consulte la siguiente página Web: http://www.bvs.sld.cu/revistas/aci/vol3_3_95/aci05395.htm

Arquitectura del software.

Existen muchas definiciones de Arquitectura del Software y no parece que ninguna de ellas haya sido totalmente aceptada. En un sentido amplio podríamos estar de acuerdo en que la Arquitectura del Software es el diseño de más alto nivel de la estructura de un sistema, programa o aplicación y tiene la responsabilidad de:

- ✓ Definir los módulos principales.
- ✓ Definir las responsabilidades que tendrá cada uno de estos módulos.
- ✓ Definir la interacción que existirá entre dichos módulos:
- ✓ Control y flujo de datos.
- ✓ Secuenciación de la información.
- ✓ Protocolos de interacción y comunicación.
- ✓ Ubicación en el hardware.

La Arquitectura del Software aporta una visión abstracta de alto nivel, posponiendo el detalle de cada uno de los módulos definidos a pasos posteriores del diseño.

La definición oficial de Arquitectura del Software es la IEEE Std 1471-2000 que reza así: "La Arquitectura del Software es la organización fundamental de un sistema formada por sus componentes, las relaciones entre ellos y el contexto en el que se implantarán, y los principios que orientan su diseño y evolución".

http://www.desarrolloweb.com/articulos/1622.php?manual=5

Modelo de datos

Una colección de herramientas conceptuales para describir los datos, las relaciones, la semántica y las restricciones de consistencia. Ejemplo de modelos de datos: Modelo entidad-relación, Modelo relacional.

Diagrama de clases

Parte de los diagramas estáticos de UML

Un diagrama de clases muestra las clases, descripciones de objetos que comparten características comunes que componen el sistema y cómo se relacionan entre sí.

Workflows o Flujos de Trabajo

Término de RUP.

Un workflow o flujo de trabajo es la automatización completa o parcial, de un proceso de negocio, su misión es controlar los procesos que se inician en una compañía para entender una demanda externa.

Artefacto.

Término de RUP.

Un artefacto es un trozo de información que es producido, modificado o usado durante el proceso de desarrollo de software. Los productos son los resultados tangibles del proyecto, las cosas que va creando y usando hasta obtener el producto final

Actividad.

Término de RUP.

Una actividad en concreto es una unidad de trabajo que una persona que desempeñe un rol puede ser solicitado a que realice. Las actividades tienen un objetivo concreto, normalmente expresado en términos de crear o actualizar algún producto

Bibliografía

Capítulo 1

(Pressman,2000) PRESSMAN Roger *Ingeniería del Software: un enfoque practico*, Ed. Mc Graw Hill, 2000. Quinta Edición.

(Nelson, 1996) NELSON, *Programming Cost*.

(Dreger 1989) DREGER, *Function Point Analysis*, Ed. Prentice-Hall, 1989

(Putman, 1992) LAWRENCE H, Putman, Measures for Excellence Ed. Yourdon Press Computing Serires, 1992

(Kemerer, 1987) KEMERER *Communications of the ACM*, 1987

(Palasí, 2003) PALASÍ, *Mecanismos de Persistencia*, Ed. Aurum Solutions 2003

Páginas de referencias:

Puntos de función:

http://www.getec.etsit.upm.es/articulos/gproyectos/art4.htm

Modelos de estimación:

http://www.inf.udec.cl/~mvaras/papers/arica/arica.htm

Publicaciones:

Puntos por Función. Una métrica estándar para establecer el tamaño del software. De Sergio Eduardo Duran Rubio Boletín de Política Informática Num. 6 2003 de la Secretaria de Economía.

Capítulo 2

(Bohem, 2000) BOHEM, W Barry , *COCOMO II Model Definition Manual*, 2000

(COCOMOII, 2000) BOEHM, Barry et al. *Software Cost Estimation with COCOMO II*. Editorial Prentice-Hall, 2000

COCOMO II (Artículos y notas):

Seminar on Software Cost Estimation presented by Nancy Merlo – Schett Requirements Engineering Research Group, Department of Computer Science, University of Zurich, Switzerland

Notas del curso de Ingeniería en Software del Prof. Raimundo Vega V. , maestro de la Universidad Austral Universidad Austral de Chile, Facultad de Ciencias de la Ingeniería, Instituto de Informática

Página: http://www.inf.uach.cl/rvega/asignaturas/info265/cocomoii.pdf

Páginas relacionadas con COCOMO II

Universidad del Sur de California:

http://sunset.use.edu/COCOMOII/cocomo.html
http://sunset.use.edu/techRpts/Reports.html
http://sunset.usc.edu/research/COCOMOII/cocomo_main.html

COCOMO II y Puntos de Función.

http://sisbib.unmsm.edu.pe/bibvirtualdata/publicaciones/risi/N1_2004/a10.pdf.

http://www.monografias.com/trabajos13/modosi/modosi.shtml

http://www.bvs.sld.cu/revistas/aci/vol3_3_95/aci05395.htm

Capítulo 3

(Silberschatz,2002) SILBERSCHATZ, Avi et al. *Fundamentos de Bases de Datos* , Ed. Mc Grawll Hill, 2002 Cuarta Edición.

(Jacobson, 2000)Jacobson Ivar El Proces Unificado de Desarrollo de Software. Ed. Pearson Addison-Wesley, 2000

(Krutchen, 2001) Kruchten, P. Architectural Blueprints—The "4+1" View Model of Software Architecture. IEEE Software 12 (6), pp. 42-50.

(Pastor,2000), PASTOR LÓPEZ Oscar *Gestión de Bases de Datos*
Ed. Universidad Politécnica de Valencia –Servicio de Publicaciones, 2000

(Braude, 2002) BRAUDE *Ingeniería de Software Orientada a Objetos* Editorial Alfa-Omega 2002

(Mendenhall,1997) MENDENHALL William y Terry Sincich *Probabilidad y Estadística para Ingenieria y Ciencias*. Editorial Prentice-Hall, 1997

(CSE-USC, 2000) https://www.cse.sc.edu/search.shtml

(UML,2002) Perdita Stevens y Rob Pooley *Utilización de UML en ingeniería del software con objetos y componentes* , Editorial Addison Wesley, 1ª. Edición, Madrid 2002

(UML 2.0, 2002) Chonoles*, UML 2.0 for Dummies*, Editorial Wiley Publishing, Inc, 2002
Artículos y páginas relacionadas con RUP:

(Castro, 2004) CASTRO GIL Alberto "Estructura básica
del proceso unificado de desarrollo de software" , Universidad ICESI (http://www.icesi.edu.co)

(Martínez, 2002) MARTÍNEZ Alejandro y Raúl. Castro Gil Universidad de Castilla la Mancha España.

(Apuntes UPV, 2004) LETELIER, "Rational Unifled Process (RUP)" Departamento de Sistemas Informáticos y Computación Universidad Politécnica de Valencia (www.https://pid.dsic.upv.es)

Portal de Desarrollo de Software de la Universidad Politécnica de Valencia España:

https://pid.dsic.upv.es

Páginas relacionadas con diseño y desarrollo de bases de datos:

Biblioteca digital Facultad de Ingeniería:

http://www.ingenieria.unam.mx/biblioteca_digital/biblioteca_general.php

Instituto Politécnico de la Paz

http://www.itlp.edu.mx/publica/tutoriales/basedat1/index.htm

http://www.itlp.edu.mx/publica/tutoriales/basedat2/index.htm

Desarrollo de una base de datos.

http://www.mailxmail.com/curso/informatica/disenobasesdatosrelacionales

Modelo de datos:

http://www.programacion.com/bbdd/tutorial/moddatos/3/#moddatos_ciclovida_4

http://atenea.ucauca.edu.co/~gramirez/archivos/AnotacionesRUP.pdf

Capítulo 4

(Bohem, 2000) BOHEM, W Barry , *COCOMO II Model Definition Manual*, 2000

(COCOMOII, 2000) BOEHM, Barry et al. *Software Cost Estimation with COCOMO II*. Editorial Prentice-Hall, 2000

Referencias de proyectos:

(SIDIT, 2006) González Roxana, Sierra et al. *Sistema de Documentación de la Infraestructura de Telecomunicaciones (Capítulo 1 y 4)*. Tesis de Licenciatura. UNAM, 2006 (Uso del material autorizado por los autores para fines académicos y de investigación)

(SGAD-IMTA) *Sistema de Gestión y Adquisición Automatizada de Información del Sistema de Medición de la Red de Agua Potable (SGAD-IMTA)*

(DGSCA-COAPA, 2006) Hawley Verónica *Desarrollo de Portal DGSCA-COAPA*

(SVIPFI, 2006) Rodríguez Verónica, Monterrubio *Sistema de Video-Proyecciones de la Facultad de Ingeniería*. Sistema desarrollándose para la Secretaria General, Salas de Video Proyecciones de la Facultad de Ingenieria UNAM 2006