# Universidad de las Ciencias Informáticas
# Facultad 2


**"Evaluación de posibles elementos a considerar en la integración de los métodos de Boehm y Humphrey para la estimación de la duración de un proyecto de software para aplicaciones Multimedia"**

Trabajo de Diploma para optar por el título de

*Ingeniero en Ciencias Informáticas*

**Autores:**

Raymari Reyes Chirino
Ariel Díaz Rodríguez

**Tutor:**
MSc. Yadira Ruiz Constanten

**Ciudad de La Habana, Junio de 2008**

*"Lo que caracteriza a una inteligencia formada es que puede descansar satisfecha con el grado de precisión que la naturaleza de un asunto permite, y no buscar la exactitud cuando solo una aproximación de la verdad es posible."*

*Aristóteles*

# DECLARACIÓN DE AUTORÍA

Declaro que soy el único autor de este trabajo y autorizo a la Facultad 2 de la Universidad de las Ciencias Informáticas (UCI) a hacer uso del mismo en su beneficio.

Para que así conste firmo la presente a los __ días del mes de ____del año ____.

_____
**Firma del Autor**
Raymari Reyes Chirino

_____
**Firma del Autor**
Ariel Díaz Rodríguez

_____
**Firma del Tutor**
Yadira Ruiz Constanten

# AGRADECIMIENTOS

A mis abuelos, por ser lo más grande que tengo en la Vida;
A mis padres, por ser mis ejemplos, por apoyarme siempre y por su amor;
A mi novio Dargenky, por ser mi amigo, mi confidente, mi amor…. gracias por tus consejos que tanta confianza me dan;
A mis hermanas, por ser tres lindos tesoros;
A toda mi familia; por estar a mi lado en todo momento;
A mi tutora Yadira, por su incomparable apoyo, su consejo más oportuno y esperar por ella casi una noche completa;
A Yudeisy, por ser mi Ada madrina durante estos 5 años;
A Ariel, por ser el mejor compañero de tesis que pude tener;
A mi eterno grupo 2102, porque conocerlos fue lo más lindo que me pasó en la Universidad;
A Adriana y Larisa, por compartir mis llantos y mis alegrías durante muchos años, y a Nadia, que aunque lejos siempre será mi mejor amiga;
A todos mis amigos, por estar siempre ahí, en las buenas y en las malas;
A todas las personas que colaboraron con nuestro trabajo;

Raymarí

A mis padres, por sus consejos, por ser mi inspiración, los principales autores;
A mis abuelos, por ser los mejores abuelos del mundo;
A mi hermano Pedro Aciel, uno de mis orgullos;
A mi novia Anaysa, por su amor, por cuidarme, por todo su apoyo;
A mis amigos, los de siempre, el enano, el flaco, el crudo y Pedry;
A mis amigos de la UCI, mi familia durante 5 años;
A mi tutora Yadira, por ser la mejor, mientras está despierta;
A Raymari, por ser la mejor compañera de tesis;
A todos aquellos que han colaborado en mi formación;

Aríel

**DEDICATORIA**

*A mis padres. . .*
*A mis abuelos. . .*

Raymarí

*A mis padres. . .*
*A mis abuelos. . .*

Aríel

# RESUMEN

Los objetivos de la estimación de proyectos son reducir los costes e incrementar los niveles de servicio y de calidad. Midiendo determinados aspectos del Proceso de Desarrollo del Software se puede tener una visión de alto nivel de lo que sucederá durante el desarrollo. Las mediciones anteriores permiten realizar predicciones sobre los actuales y las de proceso guían la toma de decisiones antes del comienzo del desarrollo y durante todo el ciclo de vida del proyecto.

El presente trabajo surge como respuesta a un conjunto de problemas que se están dando en la estimación de los proyectos en la Universidad de las Ciencias Informáticas, donde generalmente se estima la duración de un proyecto solo una vez durante su proceso de desarrollo, y además empleando un solo método, lo que impide establecer comparaciones que puedan asegurar la confiabilidad de las estimaciones realizadas, para lo cual se plantea una propuesta de integración entre los elementos comunes de las teorías de Boehm y Humphrey para una mejor estimación de la duración de un proyecto de software para aplicaciones Multimedia.

Se evaluó dicha propuesta mediante el Método de Expertos, con el fin de obtener mejores resultados en la estimación de la duración de los proyectos de Multimedia de la Universidad, que conllevaría a lograr productos de mejor calidad.

# PALABRAS CLAVE

- Estimación
- Esfuerzo
- Tiempo
- Integración
- Multimedia

# ÍNDICE DE CONTENIDOS

# ÍNDICE DE FIGURAS

# ÍNDICE DE TABLAS

# INTRODUCCIÓN

A través de la historia de la humanidad el hombre ha buscado siempre conocer; aunque la conciencia y los estudios sobre la nueva posición del conocimiento en los sistemas económicos tuvieron sus inicios en los albores del siglo XIX.

La tasa de innovación tecnológica presente es la más alta y acelerada que ha conocido la historia de la humanidad. El impacto de la ola de innovaciones está cambiando radicalmente la forma en que se producen, comercializan, distribuyen y consumen los bienes y servicios principales.

Los avances en múltiples sectores - entre ellos, el de las tecnologías de información y las comunicaciones – están dejando obsoletas las matrices tecnológicas predominantes y tienen considerables efectos en los mercados y en las estructuras organizacionales.

Durante muchos años el Proceso de Desarrollo de Software ha sido considerado como un arte dejado a la improvisación del jefe del proyecto. Los proyectos se dirigían más bajo consideraciones técnicas, que de gestión. Las actividades de estimación y de planificación quedaban relegadas a un mero acto protocolario al comienzo del proyecto. Posteriormente, el seguimiento y control se realizaban sin un mínimo de rigor, dada la baja calidad de la estimación y la planificación realizada. Las desviaciones en costos y tiempo han sido consideradas como algo natural en un proyecto software. Algo así como si nadie estuviera obligado a saber cuándo se terminará el sistema ni cuánto costará.

Por otra parte no se recogen datos sobre el proceso de desarrollo del software. Sin datos históricos como guía, la estimación de los costos no es buena y los resultados previstos muy pobres. Sin una indicación sólida de la productividad, no es posible evaluar con precisión la eficacia de las nuevas herramientas, técnicas o estándares.

La Industria del Software actual afronta una crisis debido a una serie de factores entre los que se encuentra:

- Insuficiente calidad del producto final.
- Estimaciones de duración de proyectos y asignación de recursos inexactas.
- Retrasos en la entrega de productos terminados, están fuera de control los costos de desarrollo y mantenimiento de productos.
- Escasez de personal calificado en un mercado laboral de alta demanda.
- Tendencia al crecimiento del volumen y de la complejidad de los productos.

La gestión de un proyecto de software comienza con un conjunto de actividades que, globalmente se denominan planificación del proyecto. Antes de que el proyecto comience, el gestor y el equipo de

software deben realizar una estimación del trabajo a realizar, de los recursos necesarios y del tiempo que transcurrirá desde el comienzo hasta el final de su realización. Siempre que se estima se mira hacia el futuro y se acepta resignado cierto grado de incertidumbre.

Al ser la estimación la base de la planificación, hay que prestarle especial atención, porque más que una ciencia es un arte, es una importante actividad que no debe llevarse a cabo de forma descuidada y que sirve como guía para una buena ingeniería de software. Capers Jones, especialista en metodologías de estimación para la Ingeniería de software planteó : "Buenos enfoques de estimación y datos históricos sólidos ofrecen la mejor esperanza de que la realidad puede vencer sobre las demandas imposibles" (PRESSMAN 2005).

Las claves del éxito en la gestión del desarrollo de software son: una adecuada gestión del proyecto de desarrollo de software y una adecuada gestión del proceso de software.

La medición del software está adquiriendo una gran importancia debido a que cada vez se hace más patente la necesidad de obtener datos objetivos que permitan evaluar, predecir y mejorar la calidad del software así como el tiempo y coste de desarrollo del mismo. El valor de las mediciones aumenta cuando se realiza sobre modelos construidos en las primeras fases del proyecto ya que los resultados obtenidos permiten tenerlo bajo control en todo momento y corregir a tiempo posibles desviaciones. La proliferación actual de métricas y el gran volumen de datos que se maneja ha puesto de manifiesto que las técnicas clásicas de análisis de datos son insuficientes para lograr los objetivos perseguidos.

Desde los años 70 se comienza a hablar del proceso de estimación del software. Sin embargo, y desafortunadamente, el arte y la ciencia de la estimación están hoy en día es su infancia. La industria del software sigue fuera de control, con costes y tiempos desmedidos.

La estimación de software tiene dos usos en la administración de proyectos:

1. *Durante la etapa de planeamiento:* Consiste en decidir cuántas personas son necesarias para llevar a cabo el proyecto y establecer el cronograma que se seguirá en el progreso del proyecto.
2. *Para controlar el progreso del proyecto:* Evaluar si el proyecto está evolucionando de acuerdo al cronograma y tomar las acciones correctivas si fuera necesario. Para esto se requiere contar con métricas que permitan medir el nivel de cumplimiento del desarrollo del software.

***¿Qué hace que la estimación sea tan difícil de realizar?***

Las razones para esta complejidad son las siguientes(S.-CAPUCHINO 1996):

1. No existe un modelo de estimación universal o una fórmula que pueda ser usada para todas las organizaciones. El hecho es que se pueden definir unos principios generales, pero cada interpretación es particular y diferente del resto. Cada organización tiene sus propios recursos, procedimientos e historia; y es necesario ajustar los procesos de estimación a esos parámetros únicos. Además, a medida que estos parámetros cambien, deben cambiar los procesos de estimación.

2. Hay muchas personas implicadas en los proyectos que precisan de estimaciones. La alta dirección de la empresa necesita las estimaciones para tomar decisiones de negocio, sobre la viabilidad del proyecto y su continuidad a lo largo del desarrollo. La dirección del proyecto necesita las estimaciones para hacer sugerencias a la alta dirección, para obtener los resultados necesarios para el desarrollo del proyecto, y para hacer una planificación detallada y controlar el proyecto. Cada recurso del proyecto también necesita estimaciones para planificar y controlar su propio trabajo.

3. La utilidad de una estimación también dependerá de la etapa de desarrollo en la que se encuentre. Al comienzo de un proyecto, normalmente sólo se necesitan estimaciones de coste y duración aproximadas. A medida que el proyecto madura las estimaciones que se necesitan serán más exactas. Con lo que una estimación útil al comienzo del proyecto puede no serlo más tarde.

4. La estimación, a menudo, se hace superficialmente, sin apreciar el esfuerzo requerido para hacer un trabajo. Además, también se suele dar el caso de que la estimación sea necesaria antes de obtener las especificaciones de requisitos del sistema. Por esta razón, una situación típica es que se presiona a los estimadores para que se apresuren en escribir una estimación anticipada del sistema que no comprenden aún.

5. Las estimaciones claras, completas y precisas son difíciles de formular, especialmente al inicio del proyecto. Los cambios, adaptaciones y ampliaciones son más la regla que la excepción: como consecuencia de ello, deben adaptarse también las planificaciones y los objetivos.

6. Las características del software y de su desarrollo hacen difícil la estimación. Por ejemplo, el nivel de abstracción, la complejidad, el grado de medición del producto y del proceso, los aspectos innovadores.

**Hay cuatro factores que influyen significativamente en la complejidad de las estimaciones:**

- La complejidad del proyecto.
- El tamaño del proyecto.
- El grado de incertidumbre estructural.
- Disponibilidad de información histórica.

Al presentarse ante un cliente y brindar la posibilidad de ayudar a automatizar algún proceso siempre se trata de asegurar calidad en el producto con tal eficiencia en el trabajo que muchas veces se afirma que se realizará en un breve período de tiempo. Decía Capers Jones que: "Las planificaciones temporales irracionales o excesivas son probablemente la influencia más destructiva del software"(PRESSMAN 2005). La mayoría de las veces se cumple el plazo de entrega del producto y no lo hemos acabado aún. Este fenómeno puede ir afectando el prestigio del especialista y de la entidad para la cual trabajamos.

En la industria de software en Cuba existen problemas con la productividad de los trabajadores, los tiempos de entrega de los productos y de las documentaciones, la calidad de las pruebas y la falta de comunicación efectiva entre los usuarios, desarrolladores, administradores, clientes e investigadores.

En el informe presentado por el Ministerio de Relaciones Exteriores de Cuba en la Cumbre Mundial para la Sociedad de la Información en el 2004 plantea:

La Industria Cubana del Software (InCuSoft) está llamada a convertirse en una significativa fuente de ingresos para el país, como resultado del correcto aprovechamiento de las ventajas del alto capital humano disponible. La promoción de la industria cubana del software en el ámbito internacional ha tenido como línea estratégica aprovechar la enorme credibilidad que tiene Cuba en sectores tales como la salud, la educación y el deporte. El continuar la producción sostenida de software de alta calidad en prestaciones, imagen y soporte, para satisfacer las necesidades nacionales en estos sectores, tendrá una positiva repercusión en el incremento de la exportación. (ROQUE 2005).

En la Universidad de las Ciencias Informáticas generalmente se estima la duración de un proyecto solo una vez durante su proceso de desarrollo, y además empleando un solo método, lo que impide establecer comparaciones que puedan asegurar la confiabilidad de las estimaciones realizadas. Desde otro punto de vista, en la mayoría de los casos se utiliza el método de estimación por Puntos de Casos de Uso, que se basa en el cálculo a partir de la clasificación de las transacciones obtenidas tras la descripción detallada de los casos, a pesar de ser la teoría planteada por Boehm para estimar el esfuerzo y la duración de los proyectos de software la más utilizada en el mundo.

Surgen entonces otros problemas porque ya no dependen sólo de la persona encargada de realizar la estimación, sino además de las que hicieron la descripción detallada de los casos de uso, que son los que determinan la cantidad de transacciones existentes. Es importante señalar también que el proceso de estimación muchas veces no es realizado por una persona especializada en el tema, por lo que los resultados no son lo más fieles posibles.

Además, el poco tiempo que lleva de creada la Universidad deja claro que no presenta una vasta experiencia en la realización de aplicaciones, ni en la estimación de duración de los proyectos durante la fase de Planificación. Existen áreas de desarrollo que tienen un trabajo más extenso en este sentido y ni siquiera ahí se aplica la teoría defendida por Humphrey para emplear, en el momento de la estimación, la experiencia en la que puede basarse el equipo de desarrollo en la realización de aplicaciones similares.

Sin embargo, esta investigación tiene como precedente, la realizada con el título ¨Elementos a considerar en la integración de los métodos de Boehm y Humphrey para la estimación de la duración de un proyecto de software¨, donde se vinculan ambos métodos con el fin de encontrar elementos en la integración de ellos que ayudan a dar una estimación de tiempo de duración de proyectos mucho más exacta. Se aplicará esta investigación fundamentalmente en proyectos reales de la Universidad de la Ciencias informáticas y que estén sustentados sobre la base de aplicaciones Multimedia.

De ahí que, a partir de la situación problémica descrita con anterioridad surge un ***problema***: ¿Cuáles son los resultados que ofrece evaluar los elementos de integración de los métodos de Boehm y Humphrey para la estimación de la duración de un proyecto de software para aplicaciones Multimedia?

Definiéndose en la investigación como ***Objeto de estudio*** proyectos de software para aplicaciones Multimedia.

Se reconoce además como ***Campo de Acción*** la estimación de tiempo en los proyectos de software para aplicaciones Multimedia.

Con el fin de solucionar el problema planteado se tiene como ***Objetivo general*** evaluar los elementos de integración de los métodos de Boehm y Humphrey para una mejor estimación de la duración de un proyecto de software para aplicaciones Multimedia. Concretándose los ***Objetivos específicos*** en:

1. Analizar teóricamente la estimación de la duración de proyecto de software.
2. Validar la integración entre los elementos comunes de las teorías de Boehm y Humphrey en los proyectos de Multimedia.

Por lo que se propone llevar a cabo el cumplimiento de estos objetivos desarrollando las siguientes ***tareas:***

1. Análisis de métricas de software vinculadas con la estimación para la duración de un proyecto.
2. Análisis de las teorías de estimación planteadas por Boehm y Humphrey.
3. Realizar un estudio significativo del método Puntos de Función.
4. Estudio de los elementos de integración entre las teorías de Boehm y Humphrey en la duración de proyectos de software.
5. Evaluar la integración entre los elementos comunes de las teorías de Boehm y Humphrey en los proyectos de Multimedia.

Esta investigación está basada sobre la siguiente ***Idea a defender:*** La aplicación de la integración entre los elementos comunes de las teorías de Boehm y Humphrey da como resultado una mejor estimación en la duración de los proyectos de Multimedia.

Para la realización de esta investigación se utilizaron métodos teóricos y empíricos que favorecieron el levantamiento de información.

**Métodos teóricos utilizados.**

**Método hipotético deductivo:** A partir de las teorías planteadas por Boehm y Humphrey y del estudio de los elementos de integración entre ellos, se propuso una posible integración, la que posteriormente fue sometida a verificaciones mediante el Método de Expertos.

**Métodos históricos:** Se realizó un análisis del proceso de estimación dentro de la Universidad, haciendo un recorrido por sus principales problemas.

**Métodos empíricos utilizados.**

Los métodos empíricos utilizados están vinculados a las técnicas de recolección de datos. Los utilizados fueron:

**La entrevista:** Se le realizaron entrevistas a los Líderes de proyecto de la Facultad 8 para hacer un levantamiento del estado actual de las estimaciones, los métodos utilizados para realizarlas y el tratamiento que le daban a los elementos propuestos en la integración de las teorías de Boehm y Humphrey.

**La encuesta:** Se le aplicó una encuesta a los Expertos que tenía como fin validar hasta qué punto la propuesta planteada favorece al proceso de estimación de los proyectos de Multimedia.

Este trabajo de tesis consta de tres capítulos, estos quedaron estructurados de la siguiente manera:

**Capítulo 1:** Fundamentación teórica. Aborda los conceptos asociados a la Gestión de proyectos de software, haciendo énfasis en la Planificación y Seguimiento de los mismos. Finalmente, se aborda sobre el proceso de estimación y los aspectos más importantes dentro del mismo.

**Capítulo 2:** Estudio del Modelo COCOMO, profundizando en la técnica Puntos de Función. Posteriormente se describe los elementos de integración entre las Teorías de Boehm y Humphrey.

**Capítulo 3:** Propuesta de integración y evaluación de la misma.

# CAPÍTULO 1

**Estado del arte**

El continuo incremento de la potencia de los ordenadores ha hecho posible concebir sistemas cada vez más complejos. El cerebro humano tiene solamente una capacidad limitada para manejar tales sistemas, y esto puede aplicarse igualmente al desarrollo del software para tratarlos. Además, conforme los costes del hardware disminuyen, el coste de producir el software tiene un mayor peso dentro del coste del proyecto. Conforme los costes de desarrollo y mantenimiento del software crecen es necesario predecirlos y controlarlos. Esto es algo que hasta el momento los constructores de software han encontrado muy difícil de realizar. Otro problema existente es que no es siempre posible evitar errores en los sistemas complejos, lo cual puede producir costes elevados, y perdidas fatales. El software controla actualmente sistemas médicos, tráfico aéreo, sistemas financieros o sistemas de misiles. Los errores en estos sistemas pueden implicar serios desastres. Los ejemplos son innumerables en todos los dominios de la aplicación de las Tecnologías de la Información. Según ha crecido la experiencia en la construcción de los sistemas software, se han elaborado técnicas para el desarrollo de las especificaciones y el diseño. Estas disciplinas pueden, en la actualidad, enseñarse y aplicarse según reglas muy precisas. Sin embargo, se ha puesto de manifiesto que el uso sistemático de estas técnicas para la especificación y el diseño de software no ha resuelto el problema de la producción del software. En la industria se sigue hablando de "crisis del software"; la cantidad de esfuerzo perdido en el desarrollo de software continúa en situación similar a hace años y los productos a menudo son entregados con errores significativos que producen costes y peligros graves.

El hecho es que no es suficiente avanzar a través de las etapas tradicionales del proceso de construcción de software y esperar un producto satisfactorio al final del mismo. El proceso de producción del software tiene que ser gestionado y dirigido de una manera extremadamente rigurosa y cuantitativa. De este modo se podrá verificar que el trabajo correspondiente a cada fase se ha realizado completamente dentro de los plazos de tiempo y coste establecidos y de acuerdo con estándares específicos de calidad.

Aunque a primera vista el proveedor de la solución es el responsable de los resultados del proyecto, una adecuada planeación, gestión y evaluación del avance constituyen herramientas fundamentales para la toma de decisiones del cliente (VARGAS 2006).

Un alto porcentaje de los riesgos inducidos en los proyectos de tecnologías de información se deben a deficiencias en la planeación inicial. Definición pobre del alcance del proyecto y asignación de recursos, presupuesto y cronograma aislada de los objetivos que se quieren alcanzar son algunos de los riesgos comunes que inducen los clientes en las primeras instancias de los proyectos.

El mercado de soluciones de software demanda estrategias orientadas hacia la definición de cronogramas y costos de los proyectos en instancias incipientes, en las cuales no se ha efectuado una definición detallada de requerimientos, hecho que obliga a tener amplios márgenes de error en la estimación de tiempo y costo en los proveedores.

Algunos proveedores de soluciones de software optan por mitigar el riesgo elevando el costo de sus servicios con el fin de poder asumir la responsabilidad en caso tal que el riesgo se concrete. Esta práctica disminuye significativamente la competitividad del proveedor y eleva innecesariamente los costos de desarrollo para los clientes, hecho que aumenta su inversión general en soluciones de software pero empobrece el alcance de los proyectos (VARGAS 2006).

Resulta conveniente efectuar labores de planeación detallada al interior de las organizaciones cliente, previas al inicio de los proyectos y a la apertura de concursos y licitaciones. Dichas tareas permitirán establecer con un buen nivel de detalle el objetivo que se pretende alcanzar con el desarrollo e implantación de la solución, las prestaciones o funcionalidad necesarias en la solución, una adecuada prioridad de los requerimientos y una asignación de recursos, no solo económicos sino humanos, responsables de manejar el flujo de información y requerimientos cliente-proveedor y proveedor-cliente.

La planeación detallada conducirá a obtener los máximos beneficios, adicionalmente le dará elementos para la toma de mejores decisiones en la ejecución del proyecto y será una herramienta para la evaluación de sus proveedores de tecnología.

De modo general, se ha visto que la estimación, es más un arte que una ciencia, es una actividad importante que no debe llevarse a cabo de forma descuidada. Existen técnicas útiles para la estimación de costes de tiempo, dado que la estimación es la base de todas las demás actividades de planificación del proyecto y sirve como guía para una buena Ingeniería de Sistemas y Software.

“Al estimar tomamos en cuenta no solo del procedimiento técnico a utilizar en el proyecto, sino que se toma en cuenta los recursos, costos y planificación. El Tamaño del proyecto es otro factor importante que puede afectar la precisión de las estimaciones. A medida que el tamaño aumenta, crece rápidamente la interdependencia entre varios elementos del Software” (CONCEPCIÓN 2007).

Se plantea además la importancia de la disponibilidad de la información histórica como un elemento que determina el riesgo de la estimación.

A partir de lo que se ha visto, se puede comprobar que es muy amplio el universo de definiciones o conceptos que giran en torno a la estimación y que cada proyecto asume los riesgos de acuerdo a sus características, enfrentándolas con cuidado a partir de las condiciones de trabajo y el modo de operar de cada uno de los integrantes del equipo de trabajo. Por lo tanto, pueden ser muchas las métricas usadas en el mundo para estimar duración en los proyectos de software, pero solo uno debe ser el objetivo común: Lograr efectividad en la estimación.

## 1.1 Proyectos de software

### Definición de proyecto

Son muchos los criterios que giran en torno a esta definición, a continuación se proponen varios de ellos:

- Un proyecto es un esfuerzo temporal emprendido para crear un producto o un servicio único. Así, el resultado final buscado puede diferir con la misión de la organización que la emprende, ya que el proyecto tiene determinado específicamente un plazo y el esfuerzo es temporal (WIKIPEDIA 2007c)**.**
- Un proyecto es una acción en la que recursos humanos, financieros y materiales se organizan de una nueva forma para acometer un trabajo único.
- Un proyecto es un esfuerzo temporal, único y progresivo, emprendido para crear un producto o un servicio también único (FRAN J. RUIZ-BERTOL 2004).

El aspecto esencial de un proyecto es el ser un trabajo único que se realiza con una nueva organización para producir un cambio beneficioso. Estos elementos implican que un proyecto lleva una incertidumbre considerable y riesgo, y por lo tanto su éxito dependerá en gran medida de una adecuada gestión.

El Project Management Institute (PMI) es considerado la asociación profesional para la gestión de proyectos sin fines de lucro más grande del mundo, con más de 200 000 miembros en 125 países. Su oficina central está ubicada en la localidad de Newtown Square, a las afueras de la ciudad de Filadelfia en Pennsylvania, Estados Unidos. Entre sus principales objetivos se encuentran formular estándares profesionales, generar conocimiento a través de la investigación, y promover la Gestión de Proyectos

como profesión a través de sus programas de certificación. Por supuesto, esta institución también presentó una nueva definición de proyecto, en la que señala que “un proyecto es un esfuerzo temporal para crear un único servicio o producto”.

La Guía del PMBOK, desarrollada por el Project Management Institute, contiene una descripción general de los fundamentos de la Gestión de Proyectos reconocidos como buenas prácticas. Actualmente en su tercera edición, es el único estándar ANSI para la gestión de proyectos. Todos los programas educativos y certificaciones brindadas por el PMI están estrechamente relacionados con el PMBOK.

### 1.1.1 Elementos básicos de los proyectos

**Etapas de un proyecto**

Planificación: Etapa de un proyecto en la que se valoran las opciones, tácticas y estrategias a seguir teniendo como indicador principal el objetivo a lograr.

Ejecución: Etapa de acción, en la que ocurre propiamente el proyecto.

Evaluación: Etapa final de un proyecto en la que éste es revisado, y se llevan a cabo las valoraciones pertinentes sobre lo planeado y lo ejecutado, así como sus resultados, en consideración al logro de los objetivos planteados.

**Razones para el fracaso de los proyectos**

- Fallas en la planificación del proyecto.
- Pobre justificación de la necesidad del proyecto para el negocio.
- Falta de compromiso y soporte de la gestión.
- Mal análisis en los requerimientos.
- No tener una negociación (documento, contrato) con el cliente.
- No hacer un análisis costo beneficio.
- Desconocer el ambiente de trabajo de los usuarios.
- Desconocer los usuarios que trabajan con el sistema.
- Mala elección de recursos (hardware, software, humanos).

## 1.2 Gestión del software

La **gestión de proyectos** es la disciplina de organizar y administrar recursos de manera tal que se pueda culminar todo el trabajo requerido en el proyecto dentro del alcance, el tiempo, y coste definidos (WIKIPEDIA 2007).

Si se quiere conseguir que un proyecto software se lleve a cabo con éxito, primero se debe comprender el ámbito del trabajo a realizar, los riegos en los que podemos incurrir, las tareas que se han de llevar a cabo, las etapas que hemos de recorrer, el coste del proyecto, y el plan a seguir. Este conocimiento nos lo proporciona la gestión del proyecto de software. Empieza antes de que comience el trabajo técnico, continúa a medida que el software evoluciona desde un nivel conceptual hasta la realidad y finaliza en el momento en que se abandona el software.

**¿Qué entendemos por gestión del proyecto de desarrollo de software?**

La gestión del proyecto consiste en la utilización de las técnicas y actividades de gestión requeridas para conseguir un producto software de alta calidad, de acuerdo con las necesidades de los usuarios, dentro de un presupuesto y con una planificación de tiempos establecidos previamente.

El primer desafío de la gestión de proyectos es asegurarse de que el proyecto sea entregado dentro de los parámetros definidos. El segundo es la asignación y la integración de las entradas necesarias para resolver esos objetivos predefinidos. El proyecto, por lo tanto, es un sistema cuidadosamente seleccionado de actividades definidas para utilizar los recursos (tiempo, dinero, recursos humanos, materiales, energía, espacio, provisiones, comunicación, calidad, riesgo, etc.) para resolver los objetivos predefinidos (Ver Figura 1).

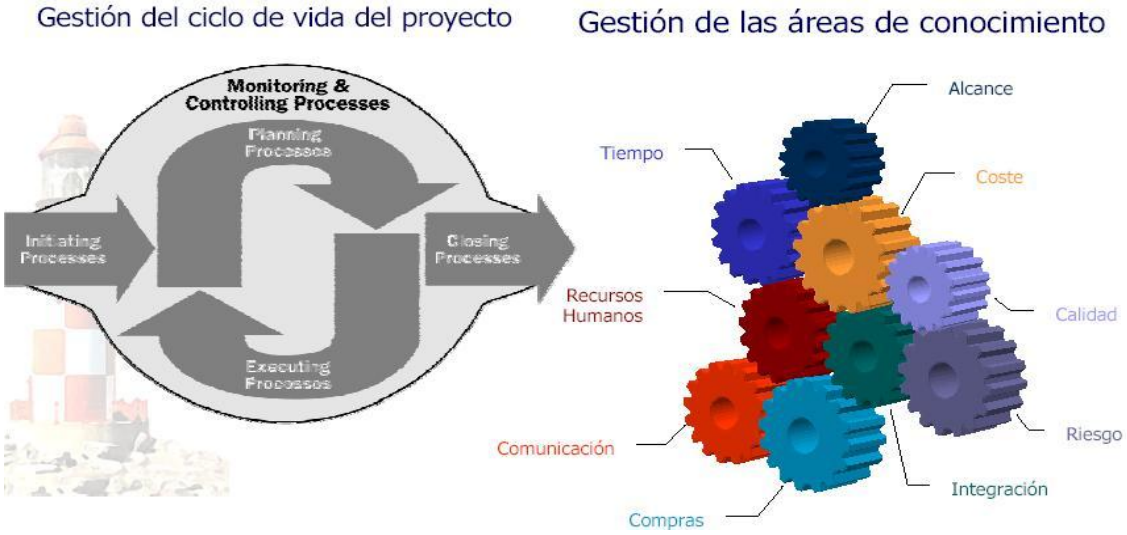


Figura 1. Gestión de proyecto.

### 1.2.1 Tareas Críticas en la Gestión de Proyectos

El número de tareas identificables a realizar por un líder de proyecto, dentro del área de la gestión de proyectos excede de cien. Sin embargo, hay tres que son críticas y que deben ser desarrolladas correctamente si se desea que el proyecto termine bien.

Estas tareas son:

- Estimación de duración, coste y esfuerzo necesarios para construir el producto.
- Planificación de tareas a realizar, asignación de personas, tiempos, etc. para construir el producto.
- Seguimiento, durante la realización del trabajo, para asegurar el cumplimiento de lo planificado en cuanto a costes, fechas, etc. En caso de desviaciones del plan, se deben tomar las medidas pertinentes.

**Las tres restricciones tradicionales**

Como cualquier empresa humana, los proyectos necesitan ser ejecutados y entregados bajo ciertas restricciones. Tradicionalmente, estas restricciones han sido alcance, tiempo y costo. Esto también se conoce como el Triángulo de la Gestión de Proyectos, donde cada lado representa una restricción. Un lado del triángulo no puede ser modificado sin impactar a los otros. Un refinamiento posterior de las restricciones separa la calidad del producto del alcance, y hace de la calidad una cuarta restricción (Ver Figura 2).

**Triángulo de la Gestión de Proyectos**


Figura 2. Refinamiento del Triángulo de la Gestión de Proyectos

Estas tres restricciones son frecuentemente competidoras entre ellas: incrementar el alcance típicamente aumenta el tiempo y el costo, una restricción fuerte de tiempo puede significar un

incremento en costos y una reducción en los alcances, y un presupuesto limitado puede traducirse en un incremento en tiempo y una reducción de los alcances.

La disciplina de la gestión de proyectos consiste en proporcionar las herramientas y técnicas que permiten al equipo de proyecto (no solamente al Líder del proyecto) organizar su trabajo para cumplir con todas esas restricciones.

**Tiempo**

El tiempo se descompone para propósitos analíticos en el tiempo requerido para completar los componentes del proyecto que es, a su vez, descompuesto en el tiempo requerido para completar cada tarea que contribuye a la finalización de cada componente. Cuando se realizan tareas utilizando gestión de proyectos, es importante partir el trabajo en pedazos menores para que sean fáciles de seguir.

**Costo**

El costo de desarrollar un proyecto depende de múltiples variables incluyendo costes de mano de obra, costes de materiales, administración de riesgo, infraestructura (edificios, máquinas, etc.), equipo y utilidades.

**Alcance**

Requerimientos especificados para el resultado final. La definición global de lo que se supone que el proyecto debe alcanzar y una descripción específica de lo que el resultado final debe ser o debe realizar. Un componente principal del alcance es la calidad del producto final. La cantidad de tiempo dedicado a las tareas individuales determina la calidad global del proyecto. Algunas tareas pueden requerir una cantidad dada de tiempo para ser completadas adecuadamente, pero con más tiempo podrían ser completadas excepcionalmente. A lo largo de un proyecto grande, la calidad puede tener un impacto significativo en el tiempo y en el costo (o viceversa).

### 1.2.2 Actividades de la gestión de proyectos

La gestión de proyectos trae consigo un conjunto de actividades, donde los líderes de proyectos son responsables, entre otras, de las siguientes actividades:

1. Redacción de la propuesta. La propuesta describe los objetivos del proyecto y cómo se llevaría a cabo. Incluye estimaciones de costo y tiempo y justifica por qué el contrato del proyecto se debe dar a una organización o equipo en particular.
2. Planificación y calendarización del proyecto. Se refiere a la identificación de actividades, hitos y entregas del proyecto.
3. Estimación de costos del proyecto. Es una actividad relacionada con la estimación de los recursos requeridos para llevar a cabo el plan del proyecto.
4. Supervisión y revisión del proyecto. La supervisión es una actividad continua. El gestor debe conocer el progreso del proyecto con los costos actuales y los planificados. También, es normal tener varias revisiones formales de su gestión. Se hace una revisión completa del progreso y de los desarrollos técnicos del proyecto, teniendo en cuenta el estado del proyecto. El resultado puede dar lugar a una cancelación.
5. Selección y evaluación del personal. Los gestores, generalmente, seleccionan a las personas que trabajarán en su proyecto o establecen un equipo ideal mínimo para el proyecto.
6. Redacción y presentación de informes. Los gestores son los responsables de informar a los clientes y contratistas sobre el proyecto. Deben redactar documentos concisos y coherentes que resuman la información crítica de los informes detallados del proyecto.

### 1.2.3 Planificación del Desarrollo del Software

La planificación de un proyecto se define como el proceso de selección de una estrategia para la producción de unos productos finales dados, así como la definición de las actividades a realizar para conseguir ese objetivo, la concurrencia y solapamiento de dichas actividades. También debe asignar recursos a las actividades anteriores en función del plan establecido. La estimación y la planificación son actividades relacionadas pero difieren en su alcance y propósito. La estimación normalmente está orientada al proyecto en su conjunto, mientras que la planificación esta dirigida a los individuos. Obviamente, debe existir una fuerte correlación entre la estimación realizada y las tareas específicas a realizar día a día por el equipo de proyecto. La estimación se realizará al principio del proyecto, precisamente para pedir presupuesto, saber cuánta gente se necesita, otros recursos, etc., y la planificación es la etapa donde se asigna exactamente quién hace qué y en cuánto tiempo.

El objetivo de la Planificación del proyecto de Software es proporcionar un marco de trabajo que permita al gestor hacer estimaciones razonables de recursos, costos y planificación temporal. Estas estimaciones se hacen dentro de un marco de tiempo limitado al comienzo de un proyecto de software,

y deberían actualizarse regularmente a medida que progresa el proyecto. Además, las estimaciones deberían definir los escenarios del mejor y peor caso, de modo que los resultados del proyecto pueden limitarse. El objetivo de la planificación se logra mediante un proceso de descubrimiento de la información que lleve a estimaciones razonables (CONCEPCIÓN 2007).

Una diferencia técnica entre las herramientas de planificación y estimación es que estas últimas son normalmente sistemas expertos, construidos a partir de las reglas derivadas de miles de proyectos. Las herramientas para la planificación, en cambio, no son sistemas expertos, sino herramientas para ser utilizadas por personas expertas. La razón para esta diferencia es que las herramientas de estimación están basadas en miles de proyectos y pueden llegar a alcanzar una gran exactitud, pero el trabajo día a día de las personas que participan en el proyecto con sus conocimientos, planes de vacaciones e interrupciones requieren un Líder de proyecto experto y casi con ajustes diarios de la planificación.
Se puede sacar una conclusión y es que las herramientas de planificación dan los mejores resultados con los mejores Jefes de proyectos. En cambio, las herramientas de estimación pueden aumentar y mejorar las capacidades de los nuevos jefes de proyecto o de los expertos cuando tienen que planificar proyectos en los que no existe una experiencia previa. Las herramientas de planificación son las más utilizadas por los jefes de proyectos.

La planificación efectiva de un proyecto de software depende de la planeación detallada de su avance, anticipando problemas que puedan surgir y preparando con anticipación soluciones tentativas a ellos (S.-CAPUCHINO 1996). Se supondrá que el administrador del proyecto es responsable de la planeación desde la definición de requisitos hasta la entrega del sistema terminado.

### 1.2.4 Seguimiento de Proyectos

El seguimiento es la recolección de datos y su almacenamiento, sobre tiempos, recursos, costes e hitos asociados con un proyecto, para el análisis y estudio de la evolución real de dicho proyecto, comparando el progreso real con el planificado (S.-CAPUCHINO 1996) Desafortunadamente, el seguimiento de los proyectos de desarrollo de software no ha sido todo lo correcto que cabría esperar.
Durante la realización del proceso de seguimiento se puede producir una replanificación si nos apartamos del plan original. Una fuerte desviación durante el seguimiento puede ser la consecuencia, por ejemplo, de un cambio en la naturaleza del proyecto. En ese caso, necesitaremos una reestimación y replanificación en consecuencia. La gestión del desarrollo del software es ineficaz a causa de que dicho desarrollo es extremadamente complejo, disponiéndose de pocas medidas para

guiar y evaluar el proceso. De esta manera sin una estimación eficaz y exacta, la planificación y el seguimiento eficaces son prácticamente imposibles de conseguir (Ver Figura 3)**.**

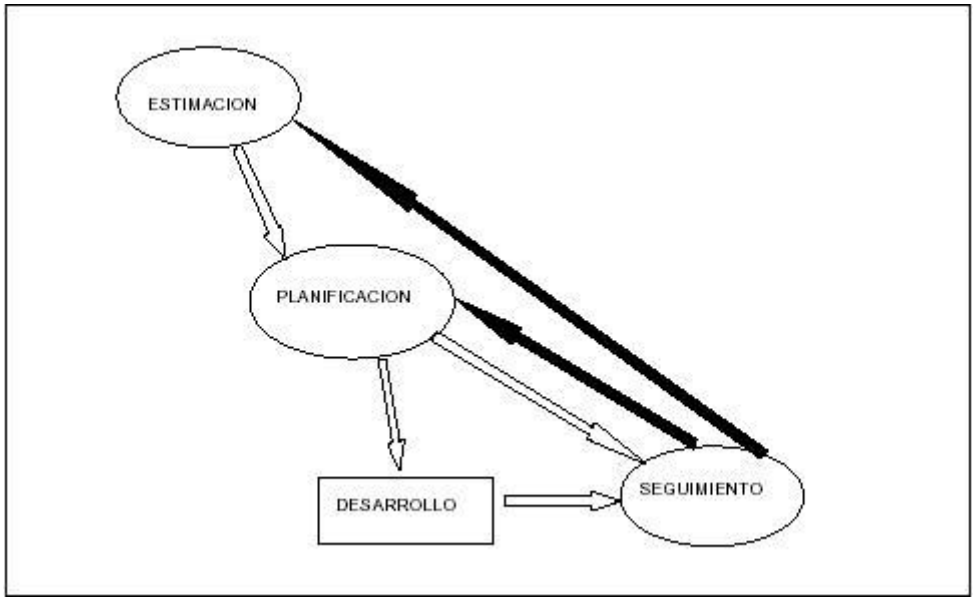


Figura 3. Seguimiento de Proyectos

Los puntos analizados posteriormente, son requeridos generalmente por grandes sistemas de programación, sin embargo estos puntos son válidos también para sistemas pequeños:

- **Panorama.** Hace una descripción general del proyecto, detalla la organización del plan y resume el resto del documento.

- **Plan de fases.** Se analiza el ciclo de desarrollo del proyecto como es: análisis de requisitos, fase de diseño de alto nivel, fase de diseño de bajo nivel, etc. Asociada con cada fase debe de haber una fecha que especifique cuándo se debe terminar estas fases y una indicación de cómo se pueden solapar las distintas fases del proyecto.

- **Plan de organización.** Se definen las responsabilidades específicas de los grupos que intervienen en el proyecto.

- **Plan de pruebas.** Se hace un esbozo general de las pruebas y de las herramientas, procedimientos y responsabilidades para realizar las pruebas del sistema.

- **Plan de control de modificaciones.** Se establece un mecanismo para aplicar las modificaciones que se requieran a medida que se desarrolle el sistema.

- **Plan de documentación.** Su función es definir y controlar la documentación asociada con el proyecto.
- **Plan de capacitación.** Se describe la preparación de los programadores que participan en el proyecto y las instrucciones a los usuarios para la utilización del sistema que se les entregue.
- **Plan de revisión e informes.** Se analiza cómo se informa del estado del proyecto y se definen las revisiones formales asociadas con el avance de proyecto.
- **Plan de instalación y operación.** Se describe el procedimiento para instalar el sistema en la localidad del usuario.
- **Plan de recursos y entregas.** Se resume los detalles críticos del proyecto como fechas programadas, marcas de logros y todos los artículos que deben entrar bajo contrato.
- **Índice.** Se muestra en dónde encontrar las cosas dentro del plan.
- **Plan de mantenimiento.** Se establece un bosquejo de los posibles tipos de mantenimiento que se tienen que dar para futuras versiones del sistema.

### 1.3 Estimación de software

La primera tarea en la gestión de proyectos es la estimación (CAO 2006). Una de las tareas de mayor importancia en la planificación de proyectos de software es la estimación, la cual consiste en determinar, con cierto grado de certeza, los recursos de hardware y software, costo, tiempo y esfuerzo necesarios para el desarrollo de los mismos. Por otra parte el Project Management Institute define un área de conocimiento (Knowlegde Area) dedicada a la administración del tiempo del proyecto (CAO 2006).
El PMI también define un grupo de procesos llamado “Planificación”. Dentro de ese grupo de procesos, en el área de conocimiento mencionada tenemos una actividad relativa a la estimación de la duración de las actividades. En ese ítem se define la estimación de la duración de actividades como el proceso de tomar información sobre el alcance del proyecto y los recursos, y entonces determinar las duraciones para ser usadas como información de entrada en los cronogramas (CAO 2006).

Ana Sánchez Capuchino, Profesora titular de Universidad del área de conocimiento de "Ciencia de la Computación e Inteligencia Artificial", de la Universidad Politécnica de Madrid, define la estimación

como “el proceso que proporciona un valor a un conjunto de variables para la realización de un trabajo, dentro de un rango aceptable de tolerancia” (S.-CAPUCHINO 1996).

Se puede definir también como la predicción de personal, del esfuerzo, de los costes y de la planificación que se requerirá para realizar todas las actividades y construir todos los productos asociados con el proyecto.
Según diría Tom DeMarco, consultor y profesor en diversas partes del mundo de temas relacionados con el desarrollo del software, “Una estimación es una predicción que tiene la misma probabilidad de estar por encima o por debajo del valor actual “(PRESSMAN 2005).
Los objetivos de la estimación de proyectos son reducir los costes e incrementar los niveles de servicio y de calidad.
Uno de los parámetros críticos de la estimación es determinar su exactitud. La estimación puede realizarse a partir de datos históricos o con herramientas. Curiosamente, en la actualidad, está ocurriendo que algunas herramientas actualmente existentes proporcionan una estimación igual de exacta que la obtenida por la empresa a partir de sus datos históricos.
Las variables a estimar dentro del marco de un proyecto de desarrollo de software pueden ser muchas. Estas van desde la estimación de las horas necesarias para realizar tareas tales como análisis, diseño, gestión, desarrollo etc. hasta la estimación de índices tales como cantidad de defectos por unidad de medida, casos de prueba por caso de uso etc. No es necesario enfatizar la importancia de las estimaciones en la gestión de proyectos. Cualquier cronograma de tareas pendientes implica estimaciones. Cualquier intención de calcular tiempos, recursos o costos a futuro implica estimaciones. Sin estimaciones no podría haber gestión de proyectos porque el proyecto es, como su nombre lo indica, una proyección.

### 1.3.1 Técnicas básicas y comunes para la estimación

1. La opinión de los expertos. Esta técnica se basa en la experiencia profesional de los participantes en el proyecto de estimación.

2. La analogía. Es una aproximación más formal que la experiencia de los expertos y se basa en la comparación directa de uno o más proyectos pasados. La estimación inicial se ajusta dependiendo de las diferencias entre el proyecto pasado y el nuevo.

3. La descomposición. Consiste en la descomposición de un producto en componentes más pequeños, o descomponer un proyecto en tareas de nivel inferior. La estimación se hace a partir del esfuerzo requerido para producir los componentes más pequeños o para realizar las tareas de nivel inferior. La estimación global del proyecto resultará de sumar las estimaciones de los componentes.

4. Las ecuaciones de estimación. Son fórmulas matemáticas que establecen la relación de algunas medidas de entrada (que normalmente es la medida del tamaño del producto) y determina el esfuerzo que se requerirá.

**Precisión y exactitud de las estimaciones**

La precisión hace referencia al número de cifras significativas que tiene una medida. La exactitud se refiere a la cercanía de una medida a su objetivo. Una predicción es útil si tiene una exactitud razonable. Es preferible expresar las estimaciones como un rango que como un número simple.

**Requisitos de un Buen Método de Estimación**

Un método de estimación tendrá éxito si:

1. La estimación inicial está dentro del 30% de desviación del coste final real.

2. El método permite el refinamiento de la estimación durante el ciclo de vida del producto software.

3. El método es fácil de usar por el estimador. Esto permite una rápida re-estimación cuando sea necesario; por ejemplo, para evaluar distintas alternativas.

4. Las reglas de la estimación son entendidas por todas las personas a las que afectan los resultados de la misma. Los directivos se sienten más seguros cuando el proceso de estimación es fácilmente comprensible.

5. El método es soportado por herramientas y está documentado. La disponibilidad de herramientas aumenta la eficacia de cualquier método. Esto es debido, principalmente, a que los resultados pueden ser obtenidos más rápidamente y de una forma estándar.

**Problemática del Proceso de Estimación del Software**

Para hablar de las posibilidades actuales de la estimación, primero se debe revisar su estado actual y explorar las necesidades de la comunidad de desarrollo de software.

¿Qué es la estimación?

Vista desde el punto de vista de un diccionario, una estimación es un conjunto aproximado de valores para algo que ha de ser hecho. En el mundo del desarrollo de software, Larry Putnam, creador del método de estimación SLIM (Modelo del Ciclo de Vida del Software), ha apuntado que la gestión del desarrollo de software considera la estimación como una actividad que permite obtener, principalmente, respuestas aproximadas a las siguientes preguntas:

¿Cuánto costará?

¿Cuánto tiempo llevará hacerlo?

### 1.3.2 Marco Temporal de la Estimación de Proyectos

¿Cuándo se debe realizar la estimación de un proyecto software?

A continuación se verá en qué momento del desarrollo de un proyecto se ha de realizar el proceso de estimación. La estimación, como ya se ha anticipado, es un proceso continuo. A medida que el proyecto avanza, más se conoce de él, y por lo tanto más parámetros están disponibles para introducir en un modelo de estimación (Ver figura 4).

La estimación continua permite el uso de un único modelo coherente que pueda capturar y utilizar la información sobre el proyecto a medida que este se conozca.

La naturaleza del proceso de estimación cambia a medida que el programa progresa. Por ejemplo, si se tiene un proyecto con un ciclo de vida tradicional. Al principio, en la concepción del sistema, sólo se necesitan estimaciones a groso modo para determinar el tamaño del proyecto y estudiar su viabilidad. Es interesante conocer el esfuerzo total del proyecto, su duración, riesgos, necesidades de personal, etc. A esta primera estimación se le denomina macro-estimación de un proyecto.

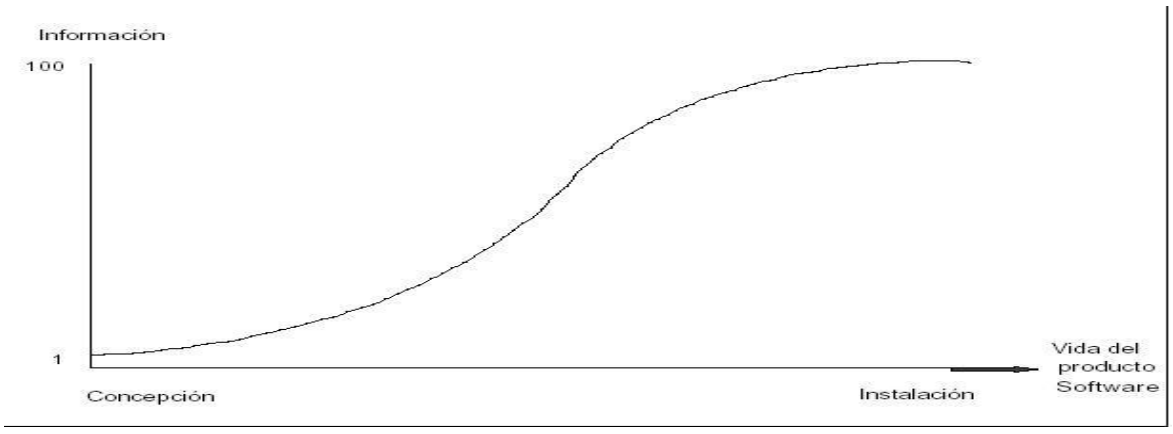


Figura 4. Grado de información sobre un proyecto.

Sin embargo, con el desarrollo del proyecto se puede evidenciar que hacen falta más estimaciones a lo largo de la vida del mismo y estas son más exactas a medida que avanza el proceso de desarrollo (Ver figura 5)**.**

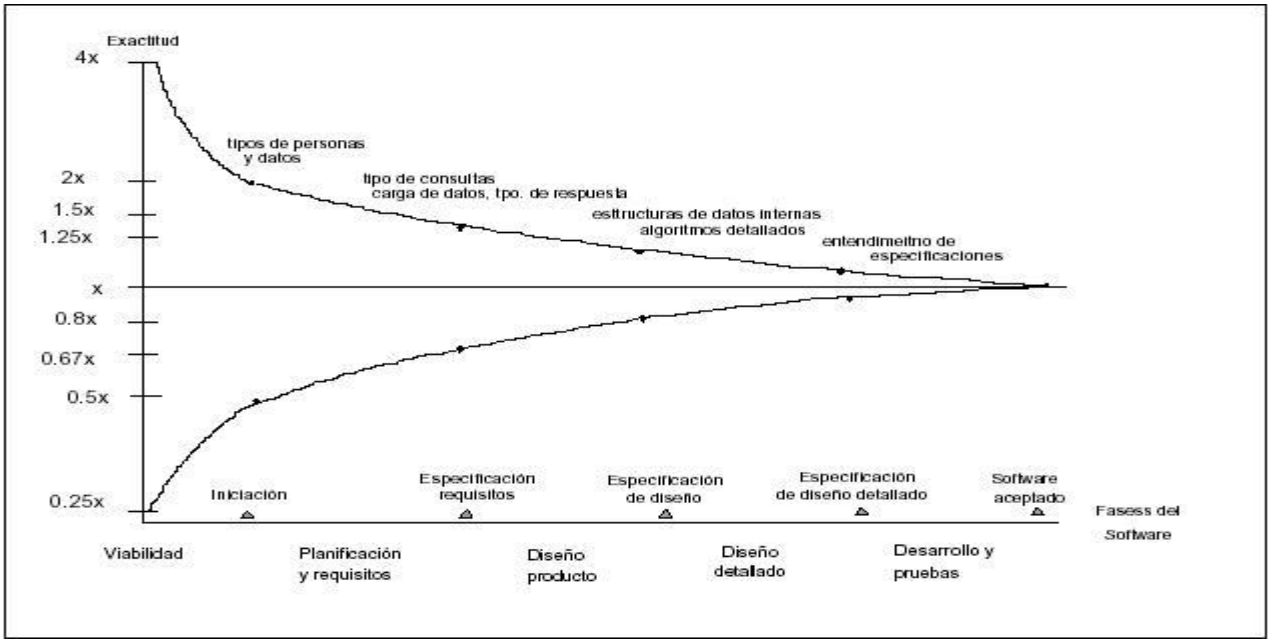


Figura 5. Estimaciones a lo largo de un proyecto.

### 1.4 Medición del software

**Medición:** Es el proceso por el cual se asignan números o símbolos a atributos de entidades del mundo real para que lo describan de acuerdo a reglas definidas claramente (MARTIG 2005).
**Entidad:** es un objeto o un evento del mundo real.
**Atributo:** es una propiedad de una entidad.

- Se miden atributos de las entidades.
- Los números y símbolos utilizados son abstracciones que usamos para reflejar nuestra percepción del mundo real y preservan las relaciones que observamos entre las entidades.

La idea de medición hace los conceptos más visibles y por lo tanto más comprensibles y controlables.
Existen dos clases de cuantificaciones:

- Medición: es una cuantificación directa.
- Cálculo: es una cuantificación indirecta. Se toman las medidas y se combinan en un ítem cuantificado que refleja algún atributo.

La Medición debe:

1. Representar adecuadamente los atributos a los que se asocian números o símbolos.

2. Preservar las relaciones que tienen los elementos, objeto de la medición.

La medición del software juega un papel muy importante en la Ingeniería del Software. Actualmente, las métricas del software están demostrando ser muy eficaces en la construcción de sistemas de predicción de alta calidad para grandes proyectos de bases de datos, en la comprensión y mejora de los proyectos de desarrollo y mantenimiento del software, en la evaluación y garantía de calidad de sistemas, evidenciando las áreas problemáticas, en la determinación de mejores prácticas de trabajo con el fin de ayudar a los usuarios e investigadores en su trabajo, etc. Además, las métricas de software son herramientas importantes que ayudan en la evaluación y en la institucionalización de la Mejora del Proceso Software (MATEUS FERREIRA A 2006) .

Sin embargo, la medición del software sufre los síntomas típicos de cualquier disciplina relativamente joven. A pesar de los esfuerzos y de los avances en la investigación y en la estandarización internacional durante la última década, la medición del software se encuentra en una fase en que terminología, principios y métodos aún están siendo definidos, consolidados y acordados. En particular, no hay todavía consenso sobre los conceptos y terminologías utilizadas en este campo.

**Objetivos de Medir Software**

Hay varias razones para medir un producto.

1. Para indicar la calidad del producto.
2. Para evaluar la productividad de la gente que desarrolla el producto.
3. Par evaluar los beneficios en términos de productividad y de calidad, derivados del uso de nuevos métodos y herramientas de la ingeniería de software.
4. Para establecer una línea de base para la estimación.
5. Para ayudar a justificar el uso de nuevas herramientas o de formación adicional.

### 1.4.1 Métricas de Software

**MÉTRICA** es una Metodología de Planificación, Desarrollo y Mantenimiento de Sistemas de información. Promovida por el Ministerio de Administraciones Públicas del gobierno español para la sistematización de actividades del ciclo de vida de los proyectos software en el ámbito de las administraciones públicas (WIKIPEDIA 2007).

La aplicación de técnicas basadas en las medidas de los procesos de desarrollo del software y sus productos debe ser continua, para producir una información de gestión significativa y a tiempo. Esta información se utilizará para mejorar esos procesos y los productos que se obtienen de ellos.

Las métricas de software implican medir: medir involucra números; el uso de números para hacer cosas mejores. Las métricas de software pretenden mejorar los procesos de desarrollo del software y mejorar, por tanto, todos los aspectos de la gestión de aquellos procesos. Estas medidas son aplicables a todo el ciclo de vida del desarrollo, desde la iniciación, cuando se debe estimar los costes, al seguimiento y control de la fiabilidad de los productos finales, y a la forma en que los productos cambian a través del tiempo debido a la aplicación de mejoras. Las métricas incluyen el uso de técnicas por parte de ingenieros de software y programadores para detectar y corregir anticipadamente los errores de los distintos componentes de los productos, antes de llegar a la codificación. Además las métricas controlan el progreso del proyecto, de tal manera que lo que pueda ocurrir seis meses más tarde se pueda identificar tan pronto como sea posible.
Esencialmente, las Métricas de Software se aplican al producto software y a los procesos mediante los que se desarrolla. Por tanto, las medidas del software y los modelos de medida son entonces útiles para estimar y predecir costes y para medir la productividad y la calidad del producto.

**¿Para qué podemos utilizar las métricas?**

Hay diferentes formas en las que pueden ser utilizadas las Métricas de Software, algunas de las cuales constituyen una especialidad por sí solas. La más consolidada de las áreas en el estudio de las métricas es la correspondiente a las técnicas de estimación de costes y tamaño. Existen distintos paquetes en el mercado que proporcionan estimación del tamaño del software a desarrollar, coste de desarrollo del sistema y duración del proyecto de desarrollo o mejora del software**.**
El uso más común de las medidas de software es la provisión de información de gestión**,** que incluye datos acerca de la productividad, calidad y eficacia de los procesos. El valor de esta información está en analizar los datos de las tendencias, día a día. ¿Está mejorando o empeorando la calidad de un equipo de desarrollo?. Si es así, ¿por qué ocurre?, ¿que puede hacer la dirección para mejorar la situación? Este campo ofrece pues, importantes aspectos para mejorar la calidad de los procesos de desarrollo de software.

#### 1.4.1.1 Características de las Métricas de Software

La calidad de las medidas debería facilitar el desarrollo de modelos que sean capaces de predecir el comportamiento de determinados parámetros que afectan al desarrollo de productos o de procesos.

Una medida ideal debería ser:

- Objetiva.
- Sencilla, definible con precisión para que pueda ser evaluada.
- Fácilmente obtenible (a un coste razonable).
- Válida, la métrica debería medir exactamente lo que se quiere medir y no otra cosa.
- Robusta, debería de ser relativamente insensible a cambios poco significativos en el proceso o en el producto.

Además, para una mejor utilización de estas medidas, a la hora de realizar estudios analíticos o análisis estadísticos deberían de tener unos valores que se ajusten a una cierta escala de medida.

#### 1.4.1.2 Clasificaciones de métricas

Las Métricas de Software se pueden clasificar, de una manera general, en Métricas de producto y Métricas de proceso.

**La Lic. Sánchez Capuchino** define:

Las **métricas de producto** son medidas del producto software durante cualquier fase de su desarrollo, desde los requisitos hasta la instalación…pueden medir la complejidad del diseño, el tamaño del producto final (fuente u objeto) o el número de páginas de documentación producida (CAO 2006).

Las **métricas de proceso** son medidas del proceso de desarrollo del software tales como tiempo de desarrollo total, esfuerzo en días/hombre o meses/hombre de desarrollo del producto, tipo de metodología utilizada o nivel medio de experiencia de los programadores (CAO 2006).

Existen otras clasificaciones de métricas, que son las que están relacionadas con el desarrollo del software como funcionalidad, complejidad, eficiencia, las cuales se proponen a continuación:

***MÉTRICAS TÉCNICAS:*** Se centran en las características de software, por ejemplo: la complejidad lógica, el grado de modularidad. Mide la estructura del sistema, el cómo esta hecho (GIRALDO 2005).

***MÉTRICAS DE CALIDAD:*** Proporcionan una indicación de cómo se ajusta el software a los requisitos implícitos y explícitos del cliente. Es decir, cómo voy a medir para que mi sistema se adapte a los requisitos que me pide el cliente (GIRALDO 2005).

***MÉTRICAS DE PRODUCTIVIDAD:*** Se centran en el rendimiento del proceso de la ingeniería del software. Es decir, qué tan productivo va a ser el software que voy a diseñar (GIRALDO 2005).

***MÉTRICAS ORIENTADAS A LA PERSONA***: Proporcionan medidas e información sobre la forma que la gente desarrolla el software de computadoras y sobre todo el punto de vista humano de la efectividad de las herramientas y métodos. (GIRALDO 2005).

***MÉTRICAS ORIENTADAS AL TAMAÑO:*** Es para saber en qué tiempo voy a terminar el software y cuántas personas voy a necesitar. Son medidas directas al software y el proceso por el cual se desarrolla, si una organización de software mantiene registros sencillos, se puede crear una tabla de datos orientados al tamaño (GIRALDO 2005).

***MÉTRICAS ORIENTADAS A LA FUNCIÓN***: Son medidas indirectas del software y del proceso por el cual se desarrolla. En lugar de calcular las LDC, las métricas orientadas a la función se centran en la funcionalidad o utilidad del programa (GIRALDO 2005).

### 1.5 Métodos utilizados para estimar en los proyectos

La mayor parte de los sistemas que son desarrollados por terceros, requieren fijar un precio antes de contratarse el desarrollo. La estimación es necesaria para la definición de ese precio. Actualmente la metodología de análisis, diseño y programación orientada a objetos ha obtenido gran preponderancia en el ámbito del desarrollo de software.

Las diversas técnicas de estimación buscan entonces reducir el elemento a estimar a la valoración de unidades comunes que permitan, por agregación, cuantificar un sistema. En ese sentido se han definido puntos de función, puntos de casos de uso, clases clave, etc.

La actividad de estimación no se hace una sola vez en el proyecto. A medida que se cuenta con más datos se hacen estimaciones más precisas que permiten una mejor planificación de lo que resta del proyecto.

De todos los puntos en los cuales puede realizarse la estimación, cuando menos datos se tienen para hacerla es en el momento inicial, cuando todavía se está evaluando la factibilidad del proyecto. Desde el punto de vista del desarrollo y venta de software específico para terceros, esa estimación que se llamará temprana, es una de las más importantes.

En efecto, la correcta estimación temprana de un proyecto de software es una tarea difícil o casi imposible. Esta situación es conocida y es por eso que cada vez más las empresas de desarrollo

intentan vender el análisis y diseño separado de la codificación, a fin de que esta última pueda ser estimada sobre bases más firmes.
En particular el Proceso Unificado de Desarrollo de Software sostiene que sólo al final de la fase de Elaboración se está en condiciones de hacer una propuesta económica firme, lo cual implica haber consumido del 25% al 30% de los recursos del proyecto al llegar a ese punto (CAO 2006).

Si se tiene en cuenta en la estimación temprana que la cantidad de información que se tiene sobre el sistema es bastante limitada, entonces se debe pensar que el elemento de estimación debe ser lo suficientemente sencillo y fácilmente obtenible como para poder ser tomado con la información que habitualmente existe en la etapa de preventa. Esta información, en general, se limita al comportamiento funcional del sistema y, eventualmente, a la tecnología a utilizar.

Todas las normas o metodologías de gestión de proyectos que existen actualmente hacen hincapié en la importancia de la gestión de plazos y costes dentro de cualquier tipo de proyecto y mucho más en los proyectos de sistemas de información debido a las peculiaridades propias.

El sistema elegido para realizar las estimaciones ha de tener la confianza del Líder del proyecto y permitir adaptarse a las necesidades cambiantes de la producción de los nuevos sistemas de información.

La recopilación de datos históricos en el cierre del proyecto es imprescindible para actualizar la base de datos de proyectos y para que el sistema pueda ajustar sus parámetros a las condiciones cambiantes de los sistemas de información.

Son muchos los métodos y técnicas de estimaciones utilizadas en el mundo entero, sin embargo, los más utilizados por los especialistas son el propuesto por Barry Boehm y el desarrollado por Watts Humphrey, de ahí que se centrará este estudio en esos dos.

### 1.5.1 COCOMO

COCOMO, propuesto y desarrollado por Barry Boehm, es uno de los modelos de estimación de costos mejor documentados y utilizados. El modelo permite determinar el esfuerzo y tiempo que se requiere en un proyecto de software a partir de una medida del tamaño del mismo expresada en el número de líneas de código que se estimen generar para la creación del producto software (GONZÁLEZ-FANJU 1997).

### 1.5.2 Proceso Personal de Software

Fue propuesto por Watts Humphrey en 1995 y estaba dirigido a estudiantes. A partir de 1997 con el lanzamiento del libro "An introduction to the Personal Software Process" se dirige a ingenieros principiantes.

El PSP se caracteriza por ser de uso personal y se aplica a programas pequeños de menos de 10.000 líneas de código. Se centra en la administración del tiempo y en la administración de la calidad a través de la eliminación temprana de defectos. En el PSP se excluyen los siguientes temas: Trabajo en equipo, Administración de configuraciones y Administración de requerimientos.

## 1.6 Conclusiones

Durante todo el capítulo se han abordado los conceptos esenciales dentro de los métodos y el proceso de estimación, definiéndose esencialmente lo relacionado con la planificación, el seguimiento y la gestión de los proyectos de software. Especial interés tiene por supuesto, las métricas de software y las metodologías propuestas por Barry Boehm y Watts Humphrey, las cuales se estudiarán con detalles en el próximo capítulo.

# CAPÍTULO 2

**Introducción**

La sociedad moderna cada vez se ha tornado más dependiente del software**.** Este se ha convertido en el bien de nuestra era; un bien al que cada vez se le invierte más en las empresas o instituciones, en su búsqueda por ofrecer más y mejores servicios, o simplemente por reducir costos operativos. Es conocido que grandes proyectos han fracasado al no estar a tiempo o dentro de presupuesto por una mala estimación de esfuerzo o duración, o de las capacidades requeridas de los ingenieros y de la empresa. Este capítulo muestra cómo el uso de métricas y en este caso de una métrica de tamaño basada en la funcionalidad, Puntos de Función, puede ayudar a tener mejor control y una mejor evaluación de la inversión en proyectos de tecnología basados en software.

## 2.1 Barry Boehm y COCOMO

Barry Boehm se licenció de matemáticas en University of California, Los Ángeles en 1964. Fue Programador-Analista en General Dynamics entre 1955 y 1959.

Sus intereses actuales en la investigación incluyen modelar los procesos del software, ingeniería de requisitos del software, las arquitecturas del software, métrica del software y los modelos de contabilidad de coste, los ambientes de la tecnología de dotación lógica, y tecnología de dotación lógica basada en el conocimiento (BOEHM 2005).

Sus contribuciones al campo incluyen el modelo constructivo del coste (COCOMO), el modelo espiral del proceso del software, el acercamiento de la teoría W (ganar-ganar) a la determinación de la gerencia y de los requisitos del software y a dos ambientes avanzados de la tecnología de dotación lógica.

### 2.1.1 COCOMO

El modelo **COCOMO** ha evolucionado debido a los constantes avances en el mercado de desarrollo de software. En el año 1981 Barry Boehm publica el modelo COCOMO, acorde a las prácticas de desarrollo de software de aquel momento.

Durante la década de los 80, el modelo se continuó perfeccionando y consolidando, siendo el modelo de estimación de costos más ampliamente utilizado en el mundo.

Al aparecer las computadoras personales y generalizarse su uso, surgieron algunas implementaciones. Varias empresas comenzaron a comercializar herramientas de estimación computarizadas. En el año 1983 se introduce el lenguaje de programación Ada (American National Standard Institute) para reducir los costos de desarrollo de grandes sistemas. Algunos aspectos de Ada provocaron un gran impacto en los costos de desarrollo y mantenimiento, así Barry Boehm y Walker Royce definieron un modelo revisado, llamado Ada COCOMO.

En los 90, las técnicas de desarrollo de software cambiaron dramáticamente, surgieron la necesidad de reutilizar software existente, la construcción de sistemas usando librerías, etc. Estos cambios comenzaron a generar problemas en la aplicación del modelo COCOMO. La solución fue reinventar el modelo. Después de algunos años y de un esfuerzo combinado de USC-CSE (University of Southern California- Center For Software Engineering) y organizaciones privadas, aparece COCOMO II. Las incorporaciones a este modelo lo reforzaron e hicieron apto para ser aplicado en proyectos vinculados a tecnologías como orientación a objetos, desarrollo incremental, composición de aplicación, y reingeniería. COCOMO II consta de tres modelos, cada uno de los cuales ofrece una precisión acorde a cada etapa de desarrollo del proyecto. Enunciados en orden creciente de fidelidad son, modelo de Composición de Aplicación, Diseño Temprano y Post Arquitectura.

El USC-CSE implementó los dos últimos modelos en una herramienta de software. Esta herramienta le permite al planificador hacer rápidamente una exploración de las posibilidades de un proyecto, analizando que efectos provoca el ajuste de requerimientos y recursos sobre la estimación de costos y tiempos. Para evitar confusión el modelo COCOMO original fue redesignado con el nombre COCOMO 81. Así todas las referencias de COCOMO encontradas en la literatura antes de 1995 se refieren a lo que ahora llamamos COCOMO 81. La mayoría de las referencias publicadas a partir de 1995 se refieren a COCOMO II.

#### 2.1.1.1 COCOMO 81

Como ya se ha mencionado fue presentado en 1981 y se convirtió en el más conocido y referenciado, además del más documentado de todos los modelos de estimación de esfuerzo de las actividades de diseño, codificación, pruebas y mantenimiento.

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

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

- Orgánico. Proyectos de no más de 50 KLDC (50.000 LDC), sobre áreas muy específicas y bien conocidas por el equipo participante.

- Semiempotrado (semilibre). El nivel de experiencia del equipo de desarrollo se sitúa en niveles intermedios y suelen ser sistemas con interfaces con otros sistemas, siendo su tamaño menor a 300 KLDC.

- Empotrado (restringido). Proyectos de gran envergadura, con una exigencia de altos niveles de fiabilidad y en los que participan muchas personas.

COCOMO 81 está compuesto por tres modelos que corresponden a distintos niveles de detalle y precisión, contemplando cada uno los tres modos de desarrollo explicados anteriormente. Mencionados en orden creciente son: Modelo Básico, Intermedio y Detallado. La estimación es más precisa a medida que se toman en cuenta mayor cantidad de factores que influyen en el desarrollo de un producto de software.

- Básico. Modelo que calcula el esfuerzo de desarrollo como función del tamaño estimado del software en Líneas de código. Adecuado para realizar estimaciones de forma rápida, aunque sin gran precisión.

- Intermedio. En este el esfuerzo se calcula como función del tamaño del producto, modificado por la valoración de los atributos directores del coste, los cuales incluyen una valoración subjetiva del producto, del hardware, del personal, etc. Los valores de los diferentes atributos se consideran como términos de impacto agregado al coste total del proyecto.

- Detallado. En él la valoración de los atributos tiene en cuenta su influencia en cada una de las fases de desarrollo del proyecto.

COCOMO 81 permite estimar cómo se distribuye el esfuerzo y el tiempo en las distintas fases del desarrollo de un proyecto y dentro de cada fase, en las actividades principales. Las fases consideradas por COCOMO 81 son:

**Diseño del Producto (PD)**

Se define la arquitectura del hardware, software y las estructuras de datos y control. También se desarrolla un bosquejo del manual del usuario y los planes de aceptación y testeo.

**Diseño Detallado (DD)**

**Codificación y Testeo de Unidades (CT)**

En estas dos fases el diseño global de la fase anterior es implementado, creando las componentes de software, que son testeadas y evaluadas individualmente.

**Integración y Testeo (IT)**

Se fusionan todas las componentes de software desarrolladas con el fin de lograr que el producto de software funcione correctamente. Los requerimientos definidos son usados para controlar las aptitudes del producto liberado.

Los costos y tiempos de las fases excluidas (Requerimientos y Mantenimiento) deben ser estimados en forma separada empleando otros modelos.

Se distinguen las siguientes actividades principales:

Análisis de Requerimientos

Determinación, especificación, revisión y actualización de la funcionalidad, performance e interface del software.

#### 2.1.1.2 COCOMO II

Los objetivos principales que se tuvieron en cuenta para construir el modelo COCOMO II fueron (S.-CAPUCHINO 1996) :

1. Desarrollar un modelo de estimación de costo y cronograma de proyectos de software que se adaptara tanto a las prácticas de desarrollo de la década del 90 como a las futuras.

2. Construir una base de datos de proyectos de software que permitiera la calibración continua del modelo, y así incrementar la precisión en la estimación.

3. Implementar una herramienta de software que soportara el modelo.

4. Proveer un marco analítico cuantitativo y un conjunto de herramientas y técnicas que evaluaran el impacto de las mejoras tecnológicas de software sobre los costos y tiempos en las diferentes etapas del ciclo de vida de desarrollo.

Estos objetivos apoyan las necesidades primarias expresadas por los usuarios de la estimación de costes del software. En orden de prioridades, estas necesidades eran: el apoyo de la planificación de proyectos, la previsión de personal del proyecto, la preparación del proyecto, la re-planificación, el seguimiento del proyecto, la negociación del contrato, la evaluación de la propuesta, la nivelación de recursos, exploración de conceptos, la evaluación del diseño y decisiones referentes a la

oferta/demanda. Para cada una de estas necesidades COCOMO II proporcionará un apoyo más moderno que sus predecesores, el COCOMO original y Ada COCOMO (S.-CAPUCHINO 1996).
COCOMO II es un modelo que permite estimar el coste, esfuerzo y tiempo cuando se planifica una nueva actividad de desarrollo software. Está asociado a los ciclos de vida modernos. El modelo original COCOMO ha tenido mucho éxito pero no puede emplearse con las prácticas de desarrollo software más recientes tan bien como con las prácticas tradicionales. Las principales capacidades de COCOMO II son los ajustes dependiendo del software a desarrollar, involucrando en la estimación del coste a los puntos objeto, puntos función y líneas de código fuente; utilizando modelizaciones no lineales para atender a la reingeniería y reusabilidad del software, y todo esto sobre la base del anterior COCOMO.

Los cuatro elementos principales de la estrategia que ha seguido COCOMO II son (S.-CAPUCHINO 1996) :

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

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

COCOMO II está compuesto por tres modelos denominados:

- Composición de Aplicación
- Diseño Temprano
- Post-Arquitectura

Estos surgen en respuesta a la diversidad del mercado actual y futuro del desarrollo de software. Esta diversidad podría representarse con el siguiente esquema:


Figura 6. Distribución del Mercado de Software Actual y Futuro.

- Aplicaciones desarrolladas por Usuarios Finales: En este sector se encuentran las aplicaciones de procesamiento de información generadas directamente por usuarios finales, mediante la utilización de generadores de aplicaciones tales como planillas de cálculo, sistemas de consultas, etc. Estas aplicaciones surgen debido al uso masivo de estas herramientas, conjuntamente con la presión actual para obtener soluciones rápidas y flexibles.

- Generadores de Aplicaciones: En este sector operan firmas como Lotus, Microsoft, Novell, Borland con el objetivo de crear módulos pre-empaquetados que serán usados por usuarios finales y programadores.

- Aplicaciones con Componentes: Sector en el que se encuentran aquellas aplicaciones que son específicas para ser resueltas por soluciones pre-empaquetadas, pero son lo suficientemente simples para ser construidas a partir de componentes interoperables. Componentes típicas son constructores de interfaces gráficas, administradores de bases de datos, buscadores inteligentes de datos, componentes de dominio-específico (medicina, finanzas, procesos industriales, etc.). Estas aplicaciones son generadas por un equipo reducido de personas, en pocas semanas o meses.

- Sistemas Integrados: Sistemas de gran escala, con un alto grado de integración entre sus componentes, sin antecedentes en el mercado que se puedan tomar como base. Porciones de estos sistemas pueden ser desarrolladas a través de la composición de aplicaciones. Entre las empresas que desarrollan software representativo de este sector, se encuentran grandes firmas que desarrollan software de telecomunicaciones, sistemas de información corporativos, sistemas de control de fabricación, etc.

- Infraestructura: Área que comprende el desarrollo de sistemas operativos, protocolos de redes, sistemas administradores de bases de datos, etc. Incrementalmente este sector direccionará sus soluciones, hacia problemas genéricos de procesamiento distribuido y procesamiento de transacciones, a soluciones middleware.

Los tres modelos de COCOMO II se adaptan tanto a las necesidades de los diferentes sectores descritos, como al tipo y cantidad de información disponible en cada etapa del ciclo de vida de desarrollo, lo que se conoce por granularidad de la información.
Se puede afirmar que para las aplicaciones desarrolladas por usuarios finales no se justifica la utilización de un modelo de estimación de costos. Estas aplicaciones normalmente se construyen en poco tiempo, por lo tanto requieren solamente una estimación basada en actividades.

**Composición de Aplicación**

El modelo Composición de Aplicación, es el modelo de estimación utilizado en los proyectos de software que se construyen a partir de componentes pre-empaquetadas, se dirige a aplicaciones que están demasiado diversificadas para crearse rápidamente en una herramienta de dominio específico, (como una hoja de cálculo) y que todavía no se conocen suficientemente como para ser compuestas a partir de componentes interoperables. Ejemplos de estos sistemas basados en componentes son los creadores de interfaces gráficas para usuario, bases de datos o 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.
En este caso, se emplean Puntos Objeto para estimar el tamaño del software, lo cual está acorde al nivel de información que generalmente se tiene en la etapa de planificación, y el nivel de precisión requerido en la estimación de proyectos de esta naturaleza. Este modelo se emplea en desarrollos de software durante la etapa de prototipación.

**Diseño Temprano**

El modelo Diseño Temprano se utiliza en las primeras etapas del desarrollo en las cuales se evalúan las alternativas de hardware y software de un proyecto. En estas etapas se tiene poca información, se conoce muy poco del tamaño del producto a ser desarrollado, de la naturaleza de la plataforma, del personal a ser incorporado al proyecto o aspectos específicos del proceso a utilizar. Este nivel de detalle en este modelo es consistente con el nivel general de información disponible y con el nivel general de estimación detallada que es necesaria en esta etapa, lo que concuerda con el uso de

Puntos de Función, para estimar tamaño usa Puntos de Función No Ajustados como métrica de medida y el uso de un número reducido de factores de costo.
El Diseño Temprano incluye la exploración de arquitecturas de software/sistema alternativas y un conjunto de siete drivers de coste (por ejemplo, dos drivers de coste para capacidad del personal y experiencia del personal en lugar de los seis drivers de coste del Modelo Post-Arquitectura que cubren varios aspectos de capacidad del personal, continuidad y experiencia). En esta fase no se sabe lo suficiente como para dar soporte a la estimación.

**Post-Arquitectura**
El modelo Post-Arquitectura se aplica en la etapa de desarrollo propiamente dicho, una vez que el proyecto está listo para desarrollar y sostener un sistema especializado, debe haber una arquitectura de ciclo de vida, la cual proporciona información más detallada sobre las entradas de los parámetros de coste, y permite mayor precisión en los cálculos de estimación del coste.
Este modelo incluye el actual desarrollo y mantenimiento de un producto software. Esta fase avanza rentablemente si se desarrolla una arquitectura de ciclo de vida software válida con respecto a la misión del sistema, al concepto de operación y al riesgo, y establecido como marca de trabajo del producto. El modelo correspondiente de COCOMO II tiene aproximadamente la misma granularidad que los anteriores modelos, COCOMO 81 y Ada COCOMO. Este modelo utiliza:

- Puntos Función y/o Líneas de Código Fuente para estimar tamaño, con modificadores que contemplan el rehúso, con y sin traducción automática, y el código desechado.
- Un conjunto de 17 atributos, denominados factores de costo, que permiten considerar características del proyecto referentes al personal, plataforma de desarrollo, etc., que tienen injerencia en los costos.
- Cinco factores que determinan un exponente, que incorpora al modelo el concepto de deseconomía y economía de escala (estos factores reemplazan los modos Orgánico, Semiacoplado y Empotrado del modelo COCOMO '81 y refina los 4 factores de exponente-escala en Ada COCOMO).

#### 2.1.1.2.1 Estimación del esfuerzo de desarrollo

**El modelo de diseño temprano y post-arquitectura**
Los modelos de **Diseño Temprano** y **Post-Arquitectura** se basan en la misma filosofía a la hora de hacer una estimación. Como se ha indicado ya, su principal diferencia se produce en la cantidad y

detalle de la información que se utiliza para obtener la estimación en cada uno de ellos. La fórmula básica para obtener una estimación de esfuerzo con ambos modelos es:

$$PM_{nominal} = A \times (KSLOC)^{B|}$$

Esta ecuación calcula el esfuerzo nominal para un proyecto de un tamaño dado expresado en Meses-persona, una persona mes es la cantidad de tiempo que una persona dedica a trabajar sobre el proyecto de desarrollo software durante un mes. El número de personas mes es diferente del tiempo que tomará el proyecto para ser completado; a esto se le llama planificación de desarrollo.
Esta ecuación es la base de los modelos de Diseño Inicial y Post-Arquitectura. Las entradas son el tamaño del desarrollo software, el tamaño se da en miles de líneas de código fuente (KSLOC), pudiéndose estimar también utilizando Puntos Función Desajustados, y convertirlos a SLOC dividiendo por 1000, una constante A, y un factor de escala B.

Esta investigación centrará su estudio en el Modelo de Diseño Temprano, puesto que este se aplica desde las etapas iniciales del proyecto, lo que satisface las necesidades de nuestra Universidad.

**Se analizará ahora cada uno de los 3 elementos de los que depende la ecuación**

**1- CONSTANTE A.**

La constante A, se usa para capturar los efectos multiplicativos de esfuerzo en proyectos de tamaño incremental. Provisionalmente se le ha estimado un valor de **2.94**.

**2- VARIABLE B (ahorro y gasto software de escala).**

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

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

**Si B < 1.0**. El proyecto presenta ahorros de escala. Si el tamaño del producto se dobla, el esfuerzo del proyecto es menor que el doble. La productividad del proyecto aumenta a medida que aumenta el

tamaño del producto. Pueden lograrse algunos ahorros de escala del proyecto con herramientas de proyecto específicas, pero normalmente es difícil lograrlo. Para proyectos pequeños, fijar costes de salida tales como herramientas a medida y normas de montaje, e informes administrativos, son a menudo una fuente de ahorro de escala.

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

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

**El exponente B se obtiene mediante los denominados drivers o factores de escala**.
La selección de drivers 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 driver de escala tiene un rango de niveles de valores desde **Muy Bajo** hasta **Extra Alto** (Tabla 1). Cada nivel de valores tiene un peso, W, y el valor específico del peso se llama factor de escala. Un factor de escala de un proyecto, Wj (Tabla 2), se calcula sumando todos los factores y se usa para determinar el exponente de escala, B.

| Factores de Escala (Wj) | **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 | Completamente familiar |
| FLEX | Riguroso | Relajación ocasional | Algo de relajación | Conformidad general | Algo de conformidad | Metas generales |
| RESL | Poco (20%) | Algo (40%) | A menudo (60%) | Generalmen- | En su mayor parte (90%) | Por completo (100%) |

| | | | | | | |
|---|---|---|---|---|---|---|
| | | | | te (75%). | | |
| TEAM | Interaccio-nes muy difíciles | Interacciones con alguna dificultad | Interacciones básicamente cooperativas | Bastante cooperativo | Altamente cooperativo | Completas Interacciones |
| PMAT | Se verá más adelante | | | | | |

Tabla 1. Factores de escala para el modelo de COCOMO II.

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

Tabla 2. Valores de los Factores de escala para el Modelo de COCOMO II.

**Precedencia y Flexibilidad en el Desarrollo (PREC Y FLEX)**

Estos dos factores de escala capturan en gran medida las diferencias entre los modos del COCOMO original, Orgánico, Semiempotrado y Empotrado.

El factor de precedencia (**PREC**) toma en cuenta el grado de experiencia previa en relación al producto a desarrollar, tanto en aspectos organizacionales como en el conocimiento del software y hardware a utilizar.

El factor de flexibilidad (**FLEX**) considera el nivel de exigencia en el cumplimiento de los requerimientos preestablecidos, plazos de tiempos y especificaciones de interface.

El modelo COCOMO II presenta la Tabla 3, en la cual se detallan las siete características a analizar para encontrar el peso de los factores ***PREC*** y ***FLEX.***

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

Tabla 3. Factores de Escala relacionados al modo de desarrollo de COCOMO II.

**Arquitectura y Determinación del Riesgo (RESL)**

Este factor combina a dos de los factores de escala del modelo Ada COCOMO: "Diseño minucioso mediante examen del Diseño del Producto (PDR)", y "Eliminación de riesgos mediante PDR". La Tabla 4 consolida estos factores del modelo Ada COCOMO para formar una definición más comprensiva de los niveles de valores RESL en COCOMO II. Este factor involucra aspectos relacionados al conocimiento de los ítems de riesgo crítico y al modo de abordarlos dentro del proyecto.

El nivel del factor RESL es el resultado de un promedio de los niveles de las características listadas en la Tabla 4.

| Características | Bajo | Muy Bajo | Nominal | Alto | Muy Alto | Extra Alto |
|---|---|---|---|---|---|---|
| El plan de gestión de riesgos identifica todos los ítems de riesgos críticos, establece hitos para resolverlos mediante PDR. | Ninguno | Poco | Algo | General mente | A menudo | Completa mente |
| Horario, presupuesto e hitos internos con PDR compatible con el Plan de gestión de riesgos. | Ninguno | Poco | Algo | General mente | A menudo | Completa mente |
| Tanto por ciento de horario desarrollado dedicado a establecer la arquitectura dados los objetivos generales del producto. | 5 | 10 | 17 | 25 | 33 | 40 |
| Porcentaje de arquitectos software de alto nivel requerido, disponible para el proyecto. | 20 | 40 | 60 | 80 | 100 | 120 |
| Herramientas de soporte disponibles para resolver ítems de riesgo, desarrollar y verificar garantías de la arquitectura. | Ninguno | Poco | Algo | Bueno | Fuerte | Completo |
| Nivel de incertidumbre en drives de arquitectura clave: misión, interface de usuario, Hw, tecnología, ejecución. | Externo | Significativo | Considera ble | Algo | Poco | Muy Poco |

Tabla 4. Componentes para calcular el factor de escala RESL de COCOMO II.

**Cohesión del Equipo (TEAM)**

El factor de escala de Cohesión del Equipo explica los recursos de turbulencia y entropía del proyecto debido a dificultades en la sincronización de los implicados en el proyecto, usuarios, clientes, desarrolladores, los que lo mantienen, etc. Estas dificultades pueden aparecer por las diferentes culturas y objetivos de los implicados; dificultades en conciliar objetivos y la falta de experiencia y de familiaridad de los implicados en trabajar como un equipo. La Tabla 5 proporciona una información detallada para el conjunto completo de niveles TEAM. El valor del factor TEAM se calcula como un promedio ponderado de las características listadas en la Tabla 5.

| Características | Bajo | Muy Bajo | Nominal | Alto | Muy Alto | Extra Alto |
|---|---|---|---|---|---|---|
| Compatibilidad entre los objetivos y culturas de los integrantes del equipo. | Poca | Alguna | Básica | Considerable | Fuerte | Total |
| Habilidad y predisposición para conciliar los objetivos. | Poca | Alguna | Básica | Considerable | Fuerte | Total |
| Experiencia en el trabajo en equipo. | Ninguna | Poca | Poca | Básica | Considerable | Vasto |
| Visión compartida de objetivos y compromisos compartidos. | Ninguna | Poca | Poca | Básica | Considerable | Amplia |

Tabla 5. Componentes del factor TEAM en COCOMO II.

**Madurez del Proceso (PMAT)**

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

Nivel de Madurez Global

| Nivel de CMM | PMAT |
|---|---|
| 1- Mitad Inferior | Muy bajo |
| 1- Mitad superior | Bajo |
| 2 | Nominal |
| 3 | Alto |
| 4 | Muy Alto |
| 5 | Extra Alto |

Tabla 6. Factor PMAT de acuerdo al nivel de CMM en COCOMO II.

Según el valor del nivel de CMM obtenido se asocia con su clasificación correspondiente (desde Muy Bajo a Extra Alto) de acuerdo a la Tabla 6 y este a su vez se asocia con el valor de factor de escala para PMAT correspondiente de acuerdo a la Tabla 2.

Áreas de Proceso Principales

La segunda está organizada en base a 18 Áreas de Proceso Principales (KPA´s) en el CMM. El procedimiento para determinar PMAT es decidir el porcentaje de conformidad para cada uno de las KPA´s. Si el proyecto ha sufrido una valoración CMM reciente entonces se usa el porcentaje de conformidad para la KPA global (basada en datos de valoración de la conformidad práctica principal). Si no se ha hecho una valoración entonces se usan los niveles de conformidad para las metas de las KPA´s con la escala de abajo (Ver tabla 7) para poner el nivel de conformidad.

El procedimiento para determinar el PMAT es establecer el porcentaje de cumplimiento de cada una de las áreas evaluando el grado de cumplimiento de las metas correspondientes. Para este procedimiento se emplea la Tabla 7.

- **Casi siempre:** Cuando los objetivos son consistentemente alcanzables y bien establecidos en procedimientos operativos estándar.

- **A menudo:** Cuando los objetivos son alcanzables con relativa frecuencia, pero en algunas ocasiones son emitidas bajo circunstancias difíciles (entre el 60% y 90% de las veces).

- **La mitad de las veces:** Cuando los objetivos son alcanzables sobre la mitad de las veces (entre el 40% y 60% de las veces).

- **Ocasionalmente:** Cuando los objetivos son alcanzados algunas veces, pero poco frecuentes (entre el 10% y 40% de las veces).

- **Casi Nunca:** Cuando los objetivos raramente son alcanzados (menos del 10% de las veces).
- **No se aplica:** Cuando se tiene el conocimiento requerido sobre el proyecto u organización y el KPA, pero se tiene un sentimiento de que las KPA´s circunstancialmente no se pueden aplicar.
- **No se conoce:** Cuando no se tiene certeza de cómo responderán las KPAs.

| **Áreas de procesos claves (KPA´s).** | **Casi siempre (90 %)** | **A menudo (60–90%)** | **La mitad de las veces (40-60%)** | **Ocasional-mente (10–40%)** | **Casi nunca (< 10 %)** | **No se aplica** | **No se conoce** |
|---|---|---|---|---|---|---|---|
| Administración de requerimientos. | | | | | | | |
| Planificación del Proyecto de Software | | | | | | | |
| Seguimiento y supervisión del Proyecto de Software. | | | | | | | |
| Administración de subcontratos. | | | | | | | |
| Aseguramiento de la calidad. | | | | | | | |
| Administración de la configuración. | | | | | | | |
| Objetivo del Proceso de Organización. | | | | | | | |
| Definición del Proceso de Organización. | | | | | | | |
| Programa de entrenamiento. | | | | | | | |
| Administración Integrada de Software. | | | | | | | |
| Ingeniería del Producto. | | | | | | | |
| Coordinación entre grupos. | | | | | | | |
| Revisión por Pares. | | | | | | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Administración Cuantitativa. | | | | | | | |
| Administración de la calidad. | | | | | | | |
| Prevención de defectos. | | | | | | | |
| Administración de las Tecnologías de Cambio. | | | | | | | |
| Administración de los Procesos de Cambio. | | | | | | | |

Tabla 7. Nivel de cumplimiento de los objetivos de cada KPA de COCOMO II.

Después de determinar el nivel de cumplimiento de cada KPA el factor PMAT es calculado según la fórmula:

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

Una vez obtenido el valor de este factor, se estará en condiciones de calcular la **Variable B**, aplicando la fórmula que se propone para ella y que tiene en cuenta cada uno de los cinco factores de escala, la cual fue presentada anteriormente.

**3- VARIABLE KSLOC o SIZE**

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

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

Si se opta por utilizar los Puntos de Función sin Ajustar para determinar el tamaño del proyecto, estos deben convertirse en líneas de código fuente en el lenguaje de implementación (ensamblador, lenguajes de alto nivel, lenguajes de cuarta generación, etc.) para evaluar la relativamente concisa

implementación por Puntos de Función. COCOMO II realiza esto tanto en el Modelo de Diseño Anticipado como en el de Post-Arquitectura usando tablas que traducen Puntos de Función sin ajustar al equivalente SLOC. En el apartado 2.3 se verá la técnica de Puntos de Función detenidamente.

**Ajuste Mediante Drivers de Coste. Factores Multiplicadores de Esfuerzo**

Los drivers de coste se usan para capturar características del desarrollo del software que afectan al esfuerzo para completar el proyecto. Los drivers de coste tienen un nivel de medida que expresa el impacto del driver en el esfuerzo de desarrollo. Estos valores pueden ir desde **Extra Bajo** hasta **Extra Alto**.

Para el propósito del análisis cuantitativo, cada nivel de medida de cada driver de coste tiene un peso asociado. El peso se llama Multiplicador de Esfuerzo (EM). La medida asignada a un driver de coste es 1.0 y el nivel de medida asociado con ese peso se llama nominal. Si un nivel de medida produce más esfuerzo de desarrollo de software, entonces su correspondiente EM está por encima de 1.0. Recíprocamente si el nivel de medida reduce el esfuerzo entonces el correspondiente EM es menor que 1.0 (Tabla 17).

Los driver de coste se clasifican en cuatro áreas:

**Producto**: Se refieren a las restricciones y requerimientos sobre el producto a desarrollar.

**Plataforma**: Estos factores analizan la complejidad de la plataforma subyacente.

**Personal**: Estos factores están referidos al nivel de habilidad que posee el equipo de desarrollo.

**Proyecto**: Estos factores se refieren a las condiciones y restricciones bajo las cuales se lleva a cabo el proyecto.

En el Modelo Post-Arquitectura el esfuerzo nominal se ajusta usando 17 drivers multiplicadores de esfuerzo. El mayor número de multiplicadores permite analizar con más exactitud el conocimiento disponible en las últimas etapas de desarrollo, ajustando el modelo de tal forma que refleje fielmente el producto de software bajo desarrollo. La fórmula ajustada para el cálculo del esfuerzo es la siguiente:

$$PM_{estimado} = PM_{nominal} \times \left( \prod_{i=1}^{17} EM_i \right)$$

El modelo de Diseño Temprano ajusta el esfuerzo nominal usando siete drivers de costo, acordes con la que se cantidad de información tiene hasta ese momento. La fórmula ajustada para el cálculo del esfuerzo es la siguiente:

$$PM_{estimado} = PM_{nominal} \times \left( \prod_{i=1}^{7} EM_i \right)$$

Los parámetros de coste del modelo de Diseño Temprano se obtienen por combinación de los parámetros de coste del modelo Post-Arquitectura.

| Áreas en que se agrupan | Drivers de Coste Diseño Temprano | Combinación Equivalente Post-Arquitectura |
|---|---|---|
| PRODUCTO | RCPX | RELY, DATA, CPLX, DOCU |
| | RUSE | RUSE |
| PLATAFORMA | PDIF | TIME, STOR, PVOL |
| PERSONAL | PERS | ACAP, PCAP, PCON |
| | PREX | AEXP, PEXP, LTEX |
| PROYECTO | FCIL | TOOL, SITE |
| | SCED | SCED |

Tabla 8. Multiplicadores de esfuerzo de Diseño Temprano y Post-Arquitectura.

**Obtención de Drivers de Coste para el Modelo de Diseño Temprano**

La siguiente aproximación es utilizada para "mapear" un conjunto completo de parámetros de coste y escalas de ratios de los participantes en el modelo de Diseño Inicial. Esto incluye el uso y combinación de equivalentes numéricos a los niveles de ratio.

| Niveles de Ratio | Combinación Numérica Equivalente |
|---|---|
| Muy Bajo, Extra Bajo | 1 |
| Bajo | 2 |
| Nominal o Medio | 3 |
| Alto | 4 |
| Muy Alto | 5 |
| Extra Alto | 6 |

Tabla 9. Niveles de Ratio y Equivalente Numérico.

Para los parámetros de coste combinados del modelo de Diseño Inicial, los valores numéricos de los parámetros de coste del modelo Post-Arquitectura (Tabla 8), son sumados y el resultado total es

asignado a un ratio de escala (desde Extra Bajo a Extra Alto) del modelo de Diseño Inicial. La escala de ratios del modelo de Diseño Inicial siempre tiene un total nominal igual a la suma de los ratios nominales de los elementos del modelo Post-Arquitectura con los que se han formado.

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

**(RCPX) Fiabilidad y Complejidad del Producto**

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

Los rangos de RELY y DOCU van desde Muy Bajo a Muy Alto; los rangos de DATA van desde Bajo a Muy Alto; y los rangos de CPLX van desde Muy Bajo a Extra Alto, por lo tanto la suma de ratios estará entre 5 (Muy Bajo(1), Bajo(2), Muy Bajo(1), Muy Bajo(1)) y 21 (Muy Alto(5), Muy Alto(5), Extra Alto(6), Muy Alto(5)).

| | **Extra Bajo** | **Muy Bajo** | **Bajo** | **Nominal** | **Alto** | **Muy Alto** | **Extra Alto** |
|---|---|---|---|---|---|---|---|
| Suma de ratios RELY, DATA, CPLX, DOCU. | 5,6 | 7,8 | 9,11 | 12 | 13,15 | 16,18 | 19,21 |
| Énfasis sobre confianza, documentación. | Muy Poco | Poco | Algo | Básico | Fuerte | Muy fuerte | Extremo |
| Complejidad del Producto. | Muy Simple | Simple | Algo | Moderado | Complejo | Muy Complejo | Extrema-damente Complejo |
| Tamaño de la Base de Datos. | Pequeño | Pequeño | Pequeño | Moderado | Grande | Muy Grande | Muy Grande |

Tabla 10.Niveles de Ratio de RCPX.

**(RUSE) Reutilización Requerida**

Este parámetro de coste del Diseño Inicial es el mismo a su homónimo en el modelo Post-Arquitectura. Este factor considera el esfuerzo adicional necesario para construir componentes que puedan ser reutilizados dentro de un mismo proyecto o en futuros desarrollos. El incremento del esfuerzo se debe a que se incorporan tareas inherentes a la reutilización, tales como: creación de diseños genéricos de

software, elaboración de mayor cantidad de documentación, testeo intensivo para asegurar que los componentes estén debidamente depurados, etc.

| | **Muy Bajo** | **Bajo** | **Nominal** | **Alto** | **Muy Alto** | **Extra Alto** |
|---|---|---|---|---|---|---|
| **RUSE** | | Ningún componente reusable | Reusable dentro del mismo proyecto | Reusable dentro de un mismo programa | Reusable dentro de una misma línea de productos | Reusable dentro de múltiples líneas de producto |

Tabla 11. Niveles de medida RUSE

**(PDIF) Inconvenientes de la plataforma**

Este parámetro de coste del Diseño Inicial combina los tres parámetros de coste Post-Arquitectura (tiempo de ejecución (TIME), almacenamiento principal (STOR), y volatilidad de la plataforma (PVOL)). TIME y STOR tienen rangos desde Nominal a Extra Alto; PVOL rangos desde Bajo a Muy Alto. Y el valor numérico de la suma de ratios estará entre 8 (Nominal(3), Nominal(3), Bajo(2)) y 17 (Extra Alto(6), Extra Alto(6), Muy Alto(5)).

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

Tabla 12. Niveles de ratio PDIF

**(PERS) Capacidad del Personal**

El parámetro de coste PERS del Diseño Inicial combina los parámetros de coste Post-Arquitectura de capacidad del analista (ACAP), capacidad del programador (PCAP), y continuidad del personal (PCON). Cada uno de estos tiene una escala de ratios desde Muy Bajo a Muy Alto. Sumando estos ratios numéricos se produce un rango de valores entre 3 y 15. Estos son diseñados sobre una escala, y los niveles de ratio PERS del Diseño Inicial son asignados a ellos, tal y como muestra la Tabla 13.

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

Tabla 13. Niveles de Ratio PERS

**(PREX) Experiencia Personal**

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

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

Tabla 14. Niveles de Ratio PREX

**(FCIL) Facilidades**

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

| | Extra Bajo | Muy Bajo | Bajo | Nominal | Alto | Muy Alto | Extra Alto |
|---|---|---|---|---|---|---|---|
| Suma de ratios de TOOL y SITE. | 2 | 3 | 4,5 | 6 | 7,8 | 9,10 | 11,12 |
| Soporte TOOL. | Mínima | Alguna | Simple colección de herramientas CASE | Herramientas de ciclo de vida básicas | Buena; moderadamente integrada | Fuerte; moderadamente integrada | Fuerte; Muy integrada |
| Condiciones Multisitio. | Escaso soporte de complejo desarrollo multisitio | Algún soporte de complejo desarrollo multisitio | Algún soporte de moderado desarrollo multisitio | Soporte básico de moderado desarrollo multisitio | Fuerte soporte de moderado desarrollo multisitio | Fuerte soporte de simple desarrollo multisitio | Muy fuerte de ordenado o simple desarrollo multisitio |

Tabla 15. Niveles de Ratio FCIL.

**(SCED) Planificación**

Este ratio mide la planificación temporal a establecer, obligada e impuesta por el equipo de desarrollo del proyecto del software. Los ratios son definidos en términos del porcentaje de planificación que se alarga o acelera con respecto a la planificación media para un proyecto que necesita una cantidad de esfuerzo determinado. Planificaciones temporales aceleradas tienden a producir más esfuerzo en las últimas fases de desarrollo debido a que más asuntos son dejados para ser determinados posteriormente. Una planificación comprimida un 74% es valorada como un porcentaje Muy Bajo, y un alargamiento del 160% es valorada como un porcentaje Muy Alto.

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

Tabla 16. Resumen de Niveles de Ratio SCED.

Después de ver cómo se buscan los niveles de medida de cada uno de los drivers de coste, se obtiene el multiplicador de esfuerzo respectivo en la siguiente tabla, haciendo solo coincidir cada driver con su nivel de medida correspondiente, que como ya se ha dicho estos van de Extra Bajo hasta Extra Alto.

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

Tabla 17. Multiplicadores de esfuerzo actualizados para el modelo de Diseño Anticipado pertenecientes a la versión USC-COCOMOII.

**Estimación del Tiempo de desarrollo**

La versión inicial de COCOMO II proporciona una capacidad de estimación de tiempo simplemente similar a las de COCOMO. La ecuación inicial de tiempos base para los modelos de Diseño Temprano y Post Arquitectura se muestra a continuación:

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

Donde:

- TDEV es el tiempo en meses desde la determinación de una línea base de requisitos del producto hasta que se completa una actividad de aceptación que certifica que el producto satisface los requisitos.
- PM negado es la estimación de meses-persona, excluyendo el estimador de esfuerzo SCED.
- B es la suma de los factores de escala del proyecto.
- SCED % es el porcentaje de compresión/expansión en el multiplicador de esfuerzo SCED (Tabla 15).

Como se ha visto, COCOMO II es un modelo robusto, abarcador y bien definido, que permite realizar las estimaciones en dependencia de la cantidad de información que se tenga, gracias a los tres

modelos que contempla. Tiene en cuenta los elementos que directa o indirectamente pueden influir en las estimaciones.

### 2.2 Watts Humphrey. Proceso Personal de Software (PSP)

Aunque la mayoría de las organizaciones industriales han adoptado principios modernos de calidad, la comunidad del software continúa confiando en la prueba como método principal de la administración de la calidad.

Usando inspecciones, las organizaciones han mejorado substancialmente la calidad del software. Otra medida significativa en la mejora de calidad del software fue tomada con la introducción del Modelo de Madurez de Capacidades (CMM) en 1987.

CMM es un modelo de evaluación de los procesos de una organización, el enfoque principal de este estaba en el sistema que administraba la ayuda que se les proporcionaba a los ingenieros de desarrollo. CMM ha tenido un efecto positivo en el funcionamiento de las organizaciones del software.

PSP se concentra en las prácticas de trabajo de los ingenieros en una forma individual. El principio detrás de PSP es ese, sirve para producir software de calidad, cada ingeniero debe trabajar en la necesidad de realizar trabajo de calidad. PSP se diseñó para ayudar a profesionales del software para que utilicen constantemente prácticas sanas de ingeniería de software.

Asimismo, les enseña a cómo planear y darle un seguimiento a su trabajo, a utilizar un proceso bien definido y medido, a establecer metas mesurables, y finalmente a la utilización del rastreo constante para alcanzar dichas metas. PSP les demuestra a los ingenieros a cómo manejar la calidad desde el principio del trabajo, a cómo analizar los resultados de cada trabajo, y a cómo utilizar los resultados para mejorar el proceso del proyecto siguiente.

Humphrey junto con el SEI han continuado trabajando en el desarrollo de PSP y asimismo han aplicado los mismos principios al Proceso en Equipo de Software o TSP.

#### 2.2.1 Principios de PSP

El diseño de PSP se basa en los siguientes principios de planeación y de calidad (HUMPHREY 2001):

- Cada ingeniero es esencialmente diferente; para ser más precisos, los ingenieros deben planear su trabajo y hacer sus planes teniendo en cuenta sus propios datos personales.
- Para mejorar constantemente su funcionamiento, los ingenieros deben utilizar personalmente procesos bien definidos y medidos.
- Para desarrollar productos de calidad, los ingenieros deben sentirse personalmente comprometidos con la calidad de sus productos.

- Cuesta menos encontrar y arreglar errores en la etapa inicial del proyecto que encontrarlos en las etapas subsecuentes.
- Es más eficiente prevenir defectos que encontrarlos y arreglarlos.
- La manera correcta de hacer las cosas es siempre la manera más rápida y más barata de hacer un trabajo.

Para hacer un trabajo de ingeniería de software de la manera correcta, los ingenieros deben planear de la mejor manera su trabajo antes de comenzarlo y deben utilizar un proceso bien definido para realizar de la mejor manera la planeación del trabajo.
Para que los desarrolladores lleguen a entender su funcionamiento de manera personal, deben medir el tiempo que pasan en cada proceso, los defectos que inyectan y remueven de cada proyecto y finalmente medir los diferentes tamaños de los productos que llegan a producir.
Para producir constantemente productos de calidad, los ingenieros deben planear, medir y rastrear constantemente la calidad del producto y deben centrarse en la calidad desde el principio de un trabajo (HUMPHREY 2001).
Finalmente, deben analizar los resultados de cada trabajo y utilizar estos resultados para mejorar sus procesos personales.

El Proceso Personal de Software fue diseñado para ayudar y guiar a los ingenieros de software a realizar bien su trabajo y haciendo esto pueden llegar a cubrir las KPA´s requeridas. PSP también muestra cómo aplicar métodos avanzados de ingeniería a sus proyectos y/o deberes diarios. Asimismo provee métodos de estimación y de planeación muy bien detallados que son necesarios para dar un seguimiento a su trabajo y provee un marco estructurado para desarrollar habilidades personales y métodos que se necesitarán más adelante para ir forjando al ingeniero de software.

### 2.2.2 Niveles de PSP

PSP tiene un marco de proceso de evolución similar al que tiene CMM. PSP trata parcialmente 12 de las 18 KPA´s definidas en el CMM vistas con anterioridad. Las KPA´s son las áreas de procesos clave, estas áreas ayudan a guiar a los programadores a que exista un mejoramiento notable en el proceso de software.

En CMM un nivel de madurez sólo se alcanza si se logran cumplir todas las KPA´s que exige cada nivel. Sin embargo PSP solamente cubre de manera parcial estas KPA´s debido a que es un

complemento de CMM y no depende uno del otro en ningún sentido por lo que es considerado como material de apoyo (HUMPHREY 2001).
Como se ha visto anteriormente el Instituto de la Ingeniería del Software (SEI) ha desarrollado el proceso personal del software para definir y reparar la holgura que existe entre el modelo de la madurez de la capacidad y el individuo. Por lo tanto es ideal utilizarlo junto con CMM pero no es obligatorio ya que es un proceso y no un modelo como lo es CMM. CMM es el "¿Qué?" y PSP es el ¿Cómo?.

La estrategia total de PSP es cerciorarse de que todos los componentes individuales se desarrollen con la más alta calidad. PSP lo logra proporcionando un marco de proceso personal ya definido que el programador puede utilizar. Este marco es:
Desarrollar un plan para cada proyecto y/o componente.

- Registrar su tiempo de desarrollo.
- Registrar sus defectos.
- Conservar sus datos en informes del proyecto.
- Utilizar sus datos para planear los proyectos y/o los componentes futuros.
- Analizar sus datos para desarrollar sus procesos con más calidad para mejorar su funcionamiento.

La figura muestra la Estructura Evolutiva de PSP, 7 pasos y 4 niveles de mejoramiento.

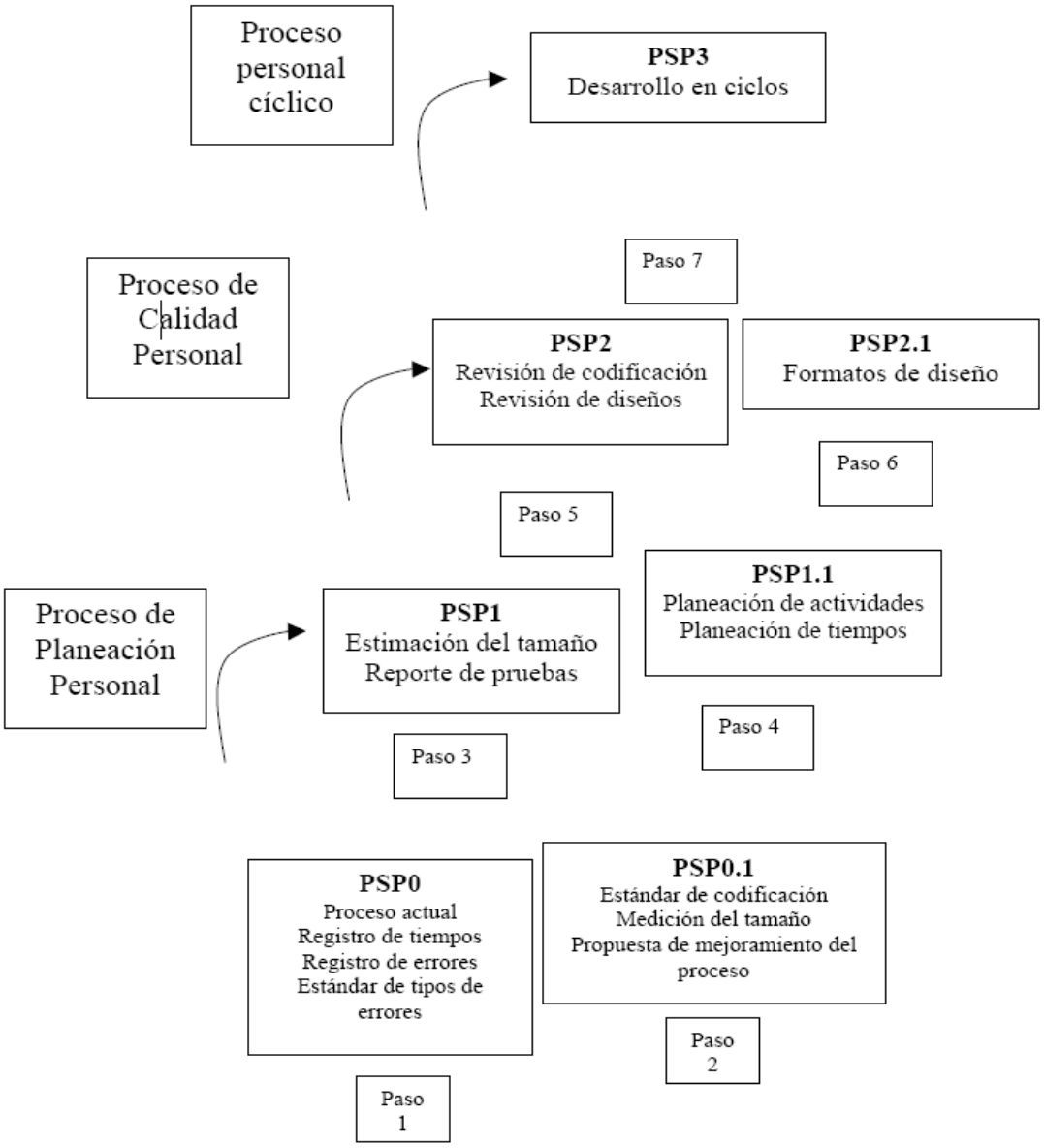


Figura 7. Evolución del Proceso Personal de Software.

En cada nivel de CMM existen KPA´s que pueden ser cumplidas siguiendo el Proceso Personal de Software, de hecho esa sería la mejor manera de cumplir con las exigencias de CMM. PSP junto a otros procesos, son el complemento ideal y el "atajo" que se podría tomar para cumplir cuanto antes con los niveles de CMM que se desean alcanzar (HUMPHREY 2001).

PSP tiene sus niveles internos y cuenta con niveles que se deben cumplir junto a su modelo por el cual fue creado, CMM. La figura presenta los elementos que tienen en común PSP con CMM. De esta manera puede analizarse cuáles se podrían aplicar a cada proyecto.

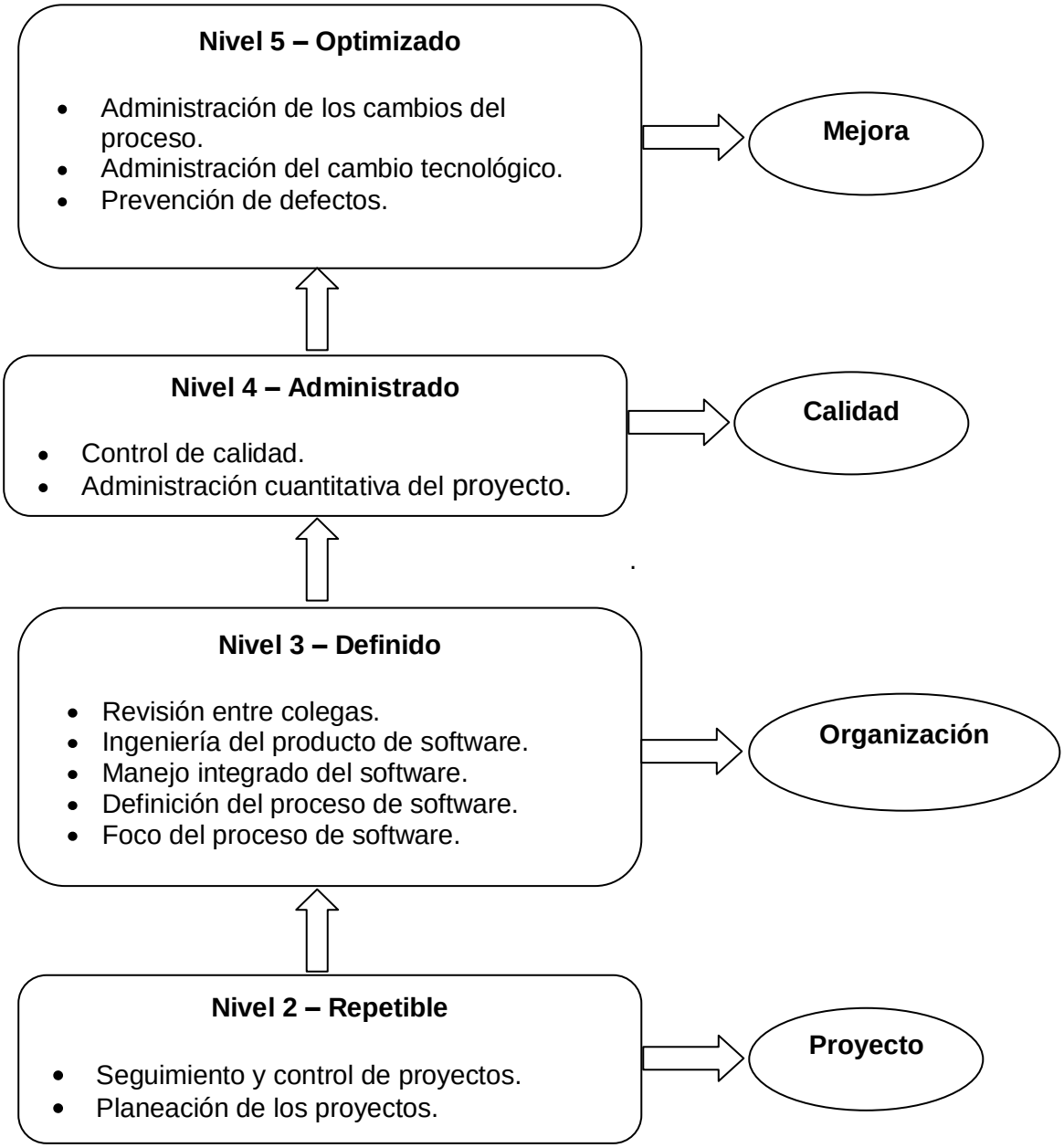


Figura 8. Áreas de procesos de PSP en CMM

### 2.2.3 PROBE

La estimación del tamaño y de los recursos del producto debe ser moderada por los equipos o individuos que intervienen en el desarrollo del proyecto. PSP comienza a estimar los tamaños de los productos que los ingenieros desarrollan personalmente y basado en esto y en los datos de la productividad de cada ingeniero estima el tiempo requerido para hacer el trabajo. Las estimaciones del tamaño del programa como de los recursos del mismo, se realizan con un método que se creó para estos fines, llamado PROBE (PROxy Based Estimating) por sus siglas en inglés, traducido al español se entiende como Estimación basada en la evaluación (HUMPHREY 2001).

La metodología de PROBE consiste en que los ingenieros deben determinar primero los objetos que se requieren para construir el producto descrito en el diseño conceptual. Después se determina el tipo probable de los métodos que se emplean en el programa y el número de métodos que cada objeto necesita. También se puede hacer uso de referencias de datos históricos sobre los tamaños de objetos similares que se han desarrollado previamente y que al mismo tiempo utilizan el cálculo de la regresión lineal para determinar el tamaño total del producto acabado.

Puesto que el tamaño del objeto está en función del estilo de programación, el método PROBE demuestra a los ingenieros cómo se deben utilizar los datos sobre los programas que ellos han desarrollado personalmente para que después se generen los distintos tipos del tamaño de cada programa y este dato esté presente en todo momento para su uso personal cada vez que se desee. Una vez que se estiman los tamaños de los objetos, se utiliza la regresión lineal para estimar la cantidad total de líneas de código fuente que planean desarrollar.

Para utilizar la regresión lineal, los ingenieros deben realizar una comparación de los datos históricos contra el resultado estimado del tamaño del programa actual, esta comparación se debe hacer con por lo menos tres programas anteriores.

El método PROBE también utiliza la regresión lineal para estimar los recursos que se emplea en el desarrollo completo.

Una vez más esta estimación se basa en el tamaño estimado del programa contra los datos reales del esfuerzo. Los datos deben demostrar una correlación razonable entre el tamaño del programa y el tiempo de desarrollo, PSP requiere que el resultado de esta correlación sea de por lo menos 0.5.

Una vez que se ha estimado el tiempo total que se empleará para el trabajo, los ingenieros deben apoyarse en sus datos históricos para estimar el tiempo necesario que cada fase del trabajo tomará. Por medio de los porcentajes que se obtienen en el campo del formato de registro de tiempo, los

ingenieros tienen que asignar su tiempo de desarrollo total estimado a las fases de planeamiento, diseño, revisión de diseño, código, revisión de código, compilación, pruebas y finalmente post-mortem. Cuando estos porcentajes han sido calculados, los ingenieros ahora cuentan con una estimación más real para el tamaño del programa, el tiempo de desarrollo total y el tiempo requerido para cada fase del desarrollo.

### 2.3 Puntos de Función

Realizada por Allan Albrecht en 1979 y revisada a continuación en 1983, esta técnica está basada en la teoría de la "ciencia del software", la cual está orientada al análisis del proceso de construcción de programas y se basa en la medida del número de "unidades sintácticas básicas" (operadores y operandos).
No se fija en el número de Líneas de Código sino en su funcionalidad.
La finalidad de la técnica de los puntos función es estimar el tamaño de un producto software y el esfuerzo asociado a su desarrollo (expresado éste en horas trabajadas por punto función, en las etapas previas a su desarrollo) cualificando la funcionalidad que proporciona externamente, basándose en el diseño lógico del sistema (ALBRECHT 1979).
Los estudios realizados sobre la utilización de este método reflejan la bondad del mismo y la existencia de un elevado grado de correlación entre el número de LDC y la estimación total de los puntos función.

Existen dos grandes orientaciones de medida del software:

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

Dentro de la primera se sitúa la técnica de los Puntos de Función, de A. Albrecht y de la segunda el modelo COCOMO (Constructive Const Model) de Barry Boehm (1981).
La utilización de LDC o DSI (delivered source instructions) es discutida por algunos autores pues la consideran una medida poco consistente, la cual presenta variaciones difícilmente ponderables:

- Longitud
- Dificultad
- Cantidad de Información
- Funcionalidad
- etc.

Más aún si se trata de líneas escritas en distintos lenguajes. En la actualidad también se debe tener en cuenta los lenguajes de alto nivel en los cuales es mucho más fácil trabajar gracias al completamiento y reutilización de código, así como la gran cantidad de librerías y frameworks que existen.
El modelo COCOMO, aún basándose en una estimación de LDC, ha sido ampliamente aceptado por:

- Ser un modelo público bien documentado.
- Debido a que los datos de entrada que solicita el modelo y sus resultados son mucho más claros y precisos que en otros modelos.
- Admite la posibilidad de calibrarse para entornos específicos.

Los objetivos de los puntos de función son:

- Medir lo que el usuario pide y lo que el usuario recibe.
- Medir independientemente la tecnología utilizada en la implantación del sistema.
- Proporcionar una métrica de tamaño que brinde soporte al análisis de la calidad y la productividad.
- Proporcionar un medio para la estimación del software.
- Proporcionar un factor de normalización para la comparación de distintos software.

Esta métrica se define como una métrica funcional, dado que se enfoca a la funcionalidad que el software proporciona al usuario. Es aceptada como estándar en el mercado. Es una métrica que se puede aplicar en las primeras fases de desarrollo. Se basa en características fundamentalmente "externas", como son ficheros lógicos, interfaces, etc. de la aplicación a desarrollar.
"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" (DEKKERS ISO Bulletin May 2003.) .
A partir de la primera definición, se pueden ver los siguientes conceptos para entender las características de la métrica:

- TAMAÑO – es una métrica de 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 de software, no considera el hardware que utilizará, 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. Si analizamos en detalle, con estos elementos podemos describir cualquier sistema.

- USUARIO – quién lo va a usar y no quién lo desarrolló o quién lo diseñó, así como existe el metro lineal para medir longitudes, Puntos Función es “el metro” para medir tamaño de una aplicación de software.

### 2.3.1 El Método Estándar Análisis de Puntos Función y su procedimiento

Este método utiliza como unidad de medida puntos función y su versión actual es la 4.1.1, cuyo manual de prácticas de conteo puede encontrarse tanto en inglés como en español. A continuación se describe brevemente en qué consiste este método. El método 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; y la sumatoria de esto da los puntos de función sin ajustar. El ajuste es un paso final basándose en las características generales de todo el sistema informático que se está contando.

**Paso 1. Determinar el tipo de conteo**

Este paso consiste en definir el tipo de conteo entre desarrollo, mantenimiento o de una aplicación ya instalada. Esta es una forma de determinar el objetivo del conteo (Ver figura 9).

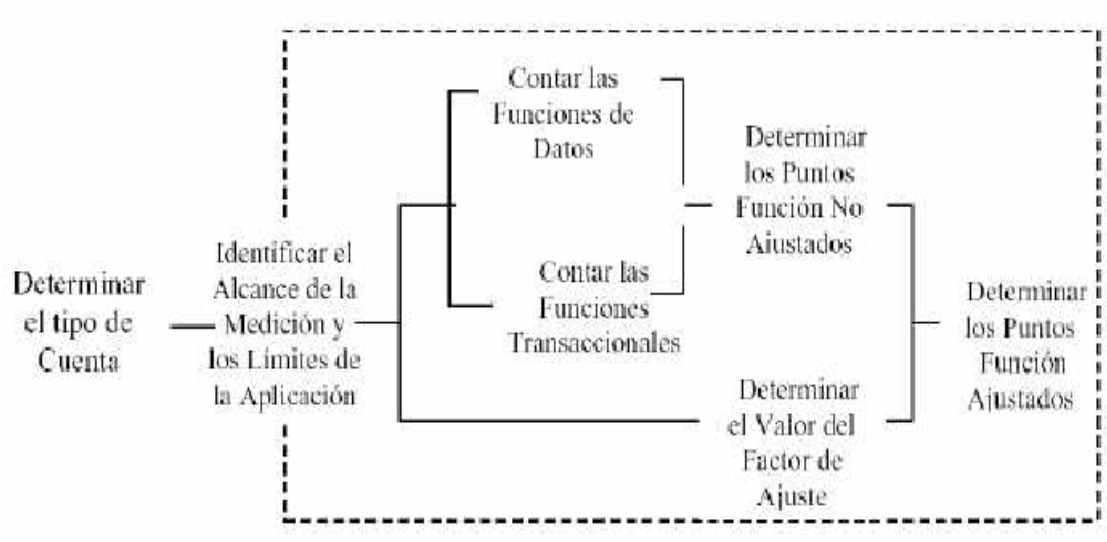


Figura 9. Ciclo completo hasta obtener los puntos ajustados.

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

El propósito de una medición consiste en dar una respuesta a un problema de negocio. El alcance de la medición define la funcionalidad que va a ser incluida en una medición específica y puede abarcar más de una aplicación (Ver figura 10).

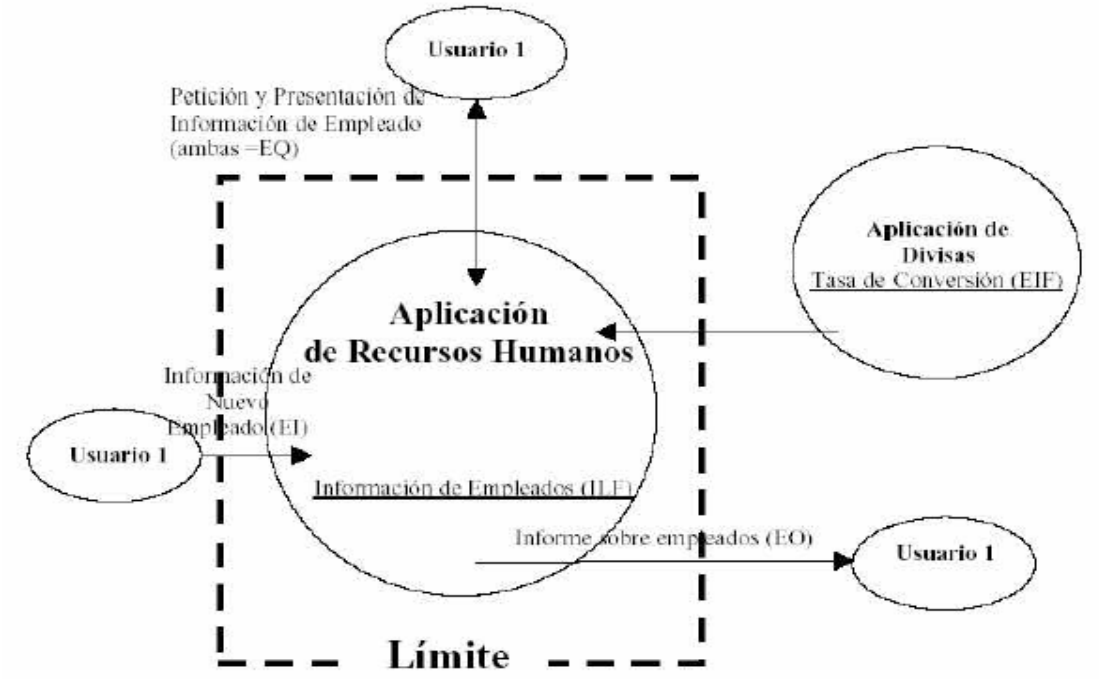


Figura 10. Límite de la aplicación.

**Paso 3. Contar las funciones de datos**

Este paso consiste en identificar y contar la capacidad de almacenamiento de los datos. Se distinguen dos tipos de funciones de datos:

Archivo Lógico Interno – 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.

Archivo de Interfaz Externo - es un grupo de datos relacionados y referenciados pero no mantenido por alguna transacción dentro del conteo.

A cada componente identificado se le asigna una complejidad (bajo, medio o alto) considerando principalmente el número de datos.

**Paso 4. Contar las funciones transaccionales**

Este paso consiste en identificar y contar la capacidad de realizar operaciones.

Se distinguen tres tipos de funciones transaccionales:

Entrada Externa – es un proceso cuyo propósito principal es mantener uno o más archivos lógicos internos.

Salida Externa – es un proceso cuyo propósito principal es presentar información al usuario mediante un proceso lógico diferente al de sólo recuperar los datos.

Consulta Externa – es un proceso cuyo propósito principal es presentar información al usuario leída de uno o más grupos de datos.

A cada componente identificado se le asigna una complejidad (bajo, medio o alto) considerando el número de datos utilizado en el proceso y los archivos referenciados.

Estos 5 componentes lógicos básicos son con los que se describe la funcionalidad de una aplicación y se podrán representar gráficamente de la siguiente forma (Ver figura 11):

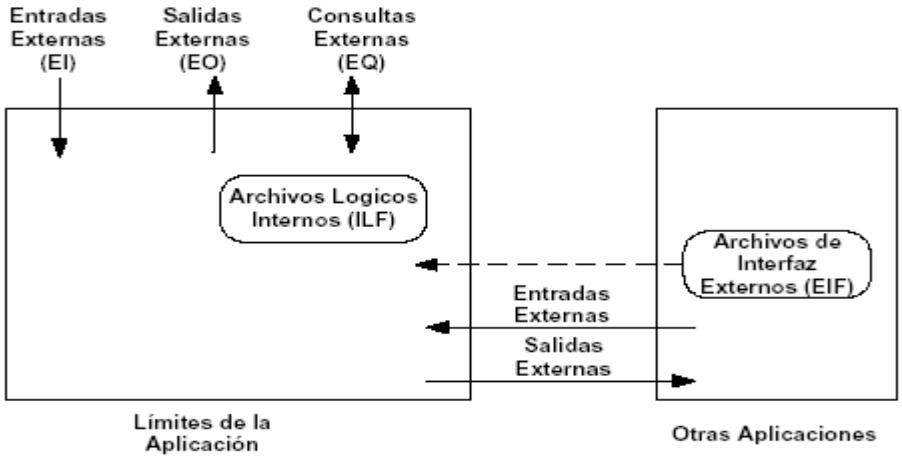


Figura 11. Componentes lógicos.

**Paso 5. Determinar los puntos de función no ajustados**

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

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

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

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

Tabla 18. Niveles de complejidad según tipo de datos y números de ficheros.

3. Aplicar los pesos asignados a cada nivel de complejidad: utilizar los pesos del siguiente esquema

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

Tabla 19. Valoraciones según el nivel de complejidad.

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

| | Bajo | Medio | Alto | Total |
|---|---|---|---|---|
| **EI** | ___x 3= ___ | ___x 4= ___ | ___x 6=___ | ____ |
| **EO** | ___x 4= ___ | ___x 5= ___ | ___x 7= ___ | ____ |
| **EQ** | ___x 3= ___ | ___x 4= ___ | ___x 6= ___ | ____ |
| **ILF** | ___x 7= ___ | ___x 10= ___ | ___x 15= ___ | ____ |
| **EIF** | ___x 5= ___ | ___x 7= ___ | ___x 10= ___ | ____ |
| | | | | ____ |

Tabla 20. Modelo para el Cálculo de Puntos de Función Desajustados.

Los Puntos Función Desajustados han de convertirse a líneas de código fuente que implementen el lenguaje (ensamblador, lenguaje de alto nivel, lenguaje de cuarta generación, etc). COCOMO II realiza

esto para los modelos de Diseño Inicial y Post-Arquitectura mediante el uso de tablas para poder trasladar estos Puntos Función Desajustados en el equivalente a SLOC.

| Lenguaje | Líneas de Código/ UFP |
|---|---|
| Ada | 71 |
| Al Shell | 49 |
| APL | 32 |
| Assembly | 320 |
| Assembly (Macro) | 213 |
| ANSI/Quick/Turbo Basic | 64 |
| Basic – Compiled | 91 |
| Basic Interpreted | 128 |
| C | 128 |
| C++ | 29 |
| Visual Basic | 32 |
| ANSI Cobol 85 | 91 |
| Fortran 77 | 105 |
| Forth | 64 |
| Jovial | 105 |
| Lisp | 64 |
| Modula 2 | 80 |
| Pascal | 91 |
| Prolog | 64 |
| Report Generator | 80 |
| Spreadsheet | 6 |
| ActionScript | 66 |

Tabla 21. Conversión de Puntos de Función Desajustados a Líneas de Código.

### 2.3.2 Elementos de los Puntos de Función

Son elementos fácilmente identificables en los diagramas de especificación del sistema. Los usuarios los entienden perfectamente.

Los Puntos de Función miden la aplicación desde una perspectiva del usuario, dejando de lado los detalles de codificación. Es una técnica totalmente independiente de todas las consideraciones de

lenguaje y ha sido aplicada en más de 250 lenguajes diferentes, por ejemplo C++, Pascal, Prolog, Fortran, y otros. Se supone que FPA evalúa con fiabilidad:

- El valor comercial de un sistema para el usuario.
- Tamaño del proyecto, coste y tiempo de desarrollo.
- Calidad y productividad del programador.
- Esfuerzo de adaptación, modificación y mantenimiento.
- Posibilidad de desarrollo propio.

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 (Boletín de Política Informática Núm. 6, 2003) .

El proceso requiere dos etapas fundamentales:

1. Se identifican las funciones disponibles para el usuario y se organizan en cinco grupos (mejor en este orden)
    - Salidas
    - Consultas
    - Entradas
    - Ficheros
    - Interfaces.

    Después se clasifica y pondera cada función por su nivel de complejidad (simple, media, compleja).
2. Se ajusta este total de acuerdo con unas características del entorno.

Cada instancia de estos tipos de funciones es clasificada según su nivel de complejidad. Los niveles de complejidad determinan un conjunto de pesos o valores, los cuales son aplicados a su correspondiente cuenta de tipo de función para determinar la cantidad de Puntos Función Desajustados. Esta es la función de medida del tamaño empleada por COCOMO II (GONZÁLEZ Mayo, 1999).

#### 2.3.2.1 Salidas

Se debe contar cada dato único de usuario o salida de control generado proceduralmente y que sale del límite de la aplicación. Esto incluye informes y mensajes a otras aplicaciones y usuarios.

Una salida se considera única si:

1. Tiene formato diferente.

2. Tiene el mismo formato que otra salida pero requiere diferente lógica de procesamiento.

Además de las pantallas y los listados (papel o pantalla), también pueden ser salidas:

- Fichero de transacción enviado a otra aplicación
- Facturas
- Cheques
- Transacciones automáticas
- Mensajes al usuario
- Cintas
- Gráficos

No se deben contar como salidas:

- Cabeceras de columna, títulos, número de página.
- Mensajes individuales (información, confirmación o respuestas a consultas de error).
- Salida en igual formato y lógica que ya se hay contado para otro soporte.

| | **Salidas** | | |
|---|---|---|---|
| | 1-5 items de datos referenciados | 6-19 items de datos referenciados | 20 o más items de datos referenciados |
| 0 o 1 fichero referenciado | Simple (4) | Simple (4) | Medio (5) |
| 2 o 3 ficheros referenciados | Simple (4) | Medio (5) | Complejo (7) |
| 4 o más ficheros referenciados | Medio (5) | Complejo (7) | Complejo (7) |

Tabla 22. Valoración de complejidad de las Salidas.

**2.3.2.2 Entradas**

Se debe contar cada dato único de usuario o entrada de control que se introduce en los límites de la aplicación y actualiza un fichero lógico interno, conjunto de datos, tabla o dato independiente. Esto incluye ficheros de entrada y transacciones recibidas de otras aplicaciones.

Una entrada se considera única si:

1. Tiene un formato diferente.

2. Tiene el mismo formato que otra entrada pero requiere una lógica diferente de procesamiento, o se modifica un fichero interno lógico diferente.

Si se tienen dos pantallas de entrada, cada una con el mismo formato pero con diferente lógica de procesamiento, se cuenta cada pantalla como una entrada diferente; pero si tuvieran la misma lógica sólo se contaría una. Lo mismo sucede con la repetición de pantallas.

Si se tiene una pantalla cuya función es actualizar un fichero o un conjunto de datos, puesto que cada una de las tres funciones de actualización (añadir, cambiar, borrar) requiere diferente lógica de procesamiento, entonces se tienen tres entradas, no una. Cada fichero tendrá tres entradas, así como una salida (el fichero formateado de salida) y una consulta.

Tipos de entradas pueden ser:

- El ratón.
- Documentos.
- Transacciones de cintas.
- Pantallas sensitivas.
- Lectores de código de barras, etc.

**Entradas**

| | 1-4 items de datos referenciados | 5-15 items de datos referenciados | 16 o más items de datos referenciados |
|---|---|---|---|
| 0 o 1 fichero referenciado | Simple (3) | Simple (3) | Medio (4) |
| 2 ficheros referenciados | Simple (3) | Medio (4) | Complejo (6) |
| 3 o más ficheros referenciados | Medio (4) | Complejo (6) | Complejo (6) |

Tabla 23. Valoración de complejidad de las Entradas.

### 2.3.2.3 Consultas

Se debe contar cada combinación única de entrada/salida en la que la entrada on-line definida por el usuario genera una salida inmediata on-line. Las consultas se pueden proporcionar a/desde otra aplicación; por ejemplo, responder a otra aplicación que pregunta por el precio de un producto se contaría como una consulta. Una consulta se considera única si:

1. Tiene un formato diferente de otras bien en su entrada o salida.

2. Tiene el mismo formato, tanto entrada como salida que otra consulta pero requiere diferente lógica de procesamiento en cualquiera de los dos.

Una consulta directa en una base de datos o fichero maestro es aquella que:

1. Utiliza claves simples para recuperar datos específicos -esto es, un registro simple o grupo de registros, no un rango-.
2. Requiere respuesta inmediata.
3. No realiza funciones de actualización (aunque se pueden efectuar cálculos).

Las consultas pueden aparecer en:

- Consulta de usuario/display sin actualización de fichero u otra entidad lógica.
- Fichero de transacción que sale del límite de la aplicación si está accesible al usuario on-line.
- Pantalla de selección de menú (todas las pantallas de menú cuentan como una consulta).
- Mensaje de información o pantalla de ayuda.

**Consultas**

| *Parte Salida* | 1-5 items de datos referenciados | 6-19 items de datos referenciados | 20 o más items de datos referenciados |
|---|---|---|---|
| 0 o 1 fichero referenciado | Simple (4) | Simple (4) | Medio (5) |
| 2 o 3 ficheros referenciados | Simple (4) | Medio (5) | Complejo (7) |
| 4 o más ficheros referenciados | Medio (5) | Complejo (7) | Complejo (7) |

| *Parte Entrada* | 1-4 items de datos referenciados | 5-15 items de datos referenciados | 16 o más items de datos referenciados |
|---|---|---|---|
| 0 o 1 fichero referenciado | Simple (3) | Simple (3) | Medio (4) |
| 2 ficheros referenciados | Simple (3) | Medio (4) | Complejo (6) |
| 3 o más ficheros referenciados | Medio (4) | Complejo (6) | Complejo (6) |

Tabla 24. Valoración de complejidad de las Consultas.

#### 2.3.2.4 Ficheros

Se debe contar cada grupo lógico mayor de datos de usuario o de información de control mantenidos dentro de los límites de la aplicación. FPA distingue entre dos tipos de ficheros: ficheros con transacciones temporales y ficheros con registros lógicos de datos permanentes. Sólo los almacenamientos de datos permanentes se ven como ficheros lógicos. Cuando se mantienen dentro

de la aplicación se clasifican como "ficheros internos lógicos". Si se comparten entre aplicaciones se clasifican como interfaces y como ficheros internos lógicos.
Las transacciones, por el contrario, se considera que son sucesos que desencadenan cambios en los ficheros lógicos internos; no se clasifican como ficheros. Un fichero transacción se puede clasificar como entrada si es leído para actualizar datos en un fichero lógico interno. Un fichero transacción puede ser una interface o una salida si transfiere transacciones de actualización a otra aplicación.

Cuando se utiliza análisis estructurado cada almacenamiento de datos contendrá al menos un fichero lógico interno. Hay que enfatizar que se habla de ficheros lógicos. Si se supone que un fichero físico contiene dos claves diferentes, entonces se contarán dos ficheros lógicos internos, puesto que cada camino presenta diferente información. Del mismo modo, cada vista lógica del usuario en una base de datos se cuenta como un fichero.
Se pueden encontrar ficheros en:

- Bases de datos: 1 por vista lógica o camino de acceso.
- Ficheros maestros: 1 por cada grupo de claves.
- Tablas mantenidas por los usuarios: estados, tarifas, mensajes, etc.
- Fichero de procesamiento batch.
- Índices de referencias cruzadas.

| | **Ficheros** | | |
|---|---|---|---|
| | 1-19 items de datos referenciados | 20-50 items de datos referenciados | 51 o más items de datos referenciados |
| 1 formato/relación de registro lógico | Simple (7) | Simple (7) | Medio (10) |
| 2-5 formatos/relaciones de registro lógico | Simple (7) | Medio (10) | Complejo (15) |
| 6 o más formatos/ relaciones de registro lógico | Medio (10) | Complejo (15) | Complejo (15) |

Tabla 25. Valoración de complejidad de los Ficheros.

### 2.3.2.5 Interfaces

Se debe contar como uno cada fichero lógico de otro grupo de datos (o información de control) que se envía fuera de los límites de la aplicación, o se comparte o es recibido desde otra aplicación. Los ficheros que se comparten entre aplicaciones se cuentan como ficheros y como interfaces en cada

aplicación en la que se utilizan; de otro modo sólo se puntuará como fichero en aquella aplicación que utilice o mantenga el fichero (la otra sólo recibirá puntos de interface). Esto es, cada fichero interface debe ser también un fichero interno lógico en esa aplicación, en otra o en ambas; o puede ser un fichero transacción o de impresión generado en la propia aplicación. Las interfaces presentan una de estas situaciones:

1. Datos o información de control se pasa del fichero A al fichero B. En A se puntúa fichero e interface y en B sólo interface.
2. Datos o información de control se pasa del fichero B a A. En B se puntúa fichero e interface y en A sólo interface.
3. Datos o información de control se comparte entre A y B. A y B reciben puntos de fichero e interface.

| Utilización del fichero: | en esta aplicación A contar | en las otras aplicaciones B |
|---|---|---|
| recibido de B | sólo interface (sin actualizaciones) | ambos fichero e interface |
| compartido con B | ambos fichero e interface | ambos fichero ( si se mantiene) e interface |
| enviado a B | ambos fichero e interface | sólo interface (sin actualizaciones) |

Tabla 26. Puntuación de ficheros e Interfaces.

Las interfaces habitualmente involucran ficheros maestros, no transacciones. Hay diferencia entre ficheros maestros lógicos y ficheros transacción. Si las aplicaciones se relacionan a través de transacciones entonces se puntuarán entrada, salida, y/o consulta, y, quizá, interface. Si lo hacen a través de ficheros maestros entonces se puntuará interface y, quizás, fichero. Un fichero transacción no se contará como interface si el formato con el que lo recibe el otro programa es el mismo (no hay conexión). El programa receptor lo contaría como entrada. Si el programa que lo envía realiza el trabajo de conversión entonces se contará (para éste) una salida y un interface.

Las interfaces se pueden encontrar en:

- Ficheros lógicos internos accesibles desde otra aplicación.
- Ficheros lógicos internos accedidos en otra aplicación.
- Base de datos compartida.
- Lista de parámetros compartida.
- Fichero de impresión exportado.

- Fichero transacción compartido que requiere conversión.

Se contarán como una interface.

- Ficheros de registros de otra aplicación (en la otra aplicación (+1 fichero, +1 interface).
- Fichero de registros a otra aplicación (+1 fichero) (otra aplicación +1 interface).
- Fichero de registros a varias aplicaciones (+1 fichero) - afecta al peso de complejidad también-.
- Fichero de registros compartido entre dos o más aplicaciones (+1 fichero) (para las otras aplicaciones: +1 interface, +1 fichero en cada aplicación si realizan mantenimiento).
- Base de datos compartida con otras aplicaciones (+1 fichero) 1 interface por cada vista realmente enviada (para la otra aplicación: +1 fichero, +1 interface por cada vista utilizada).
- Base de datos compartida de otras aplicaciones (+1 fichero) 1 interface por cada vista utilizada (para la otra aplicación: +1 fichero, +1 interface por vista).
- Fichero transacción de otra aplicación con conversión de datos (+1 entrada).
- Fichero transacción enviado a otra aplicación con conversión de datos (+1 salida). Los ficheros transacción sólo se cuentan en una aplicación (no en las dos).
- Lista de parámetros.

**Interfaces**

| | 1-19 items de datos referenciados | 20-50 items de datos referenciados | 51 o más items de datos referenciados |
|---|---|---|---|
| 1 formato/relación de registro lógico | Simple (5) | Simple (5) | Medio (7) |
| 2-5 formatos/relaciones de registro lógico | Simple (5) | Medio (7) | Complejo (10) |
| 6 o más formatos/ relaciones de registro lógico | Medio (7) | Complejo (10) | Complejo (10) |

Tabla 27. Valoración de complejidad de las Interfaces.

## 2.4 Proyectos de Multimedia

El término Multimedia en el mundo de la computación es la forma de presentar información que emplea una combinación de texto, sonido, imágenes, vídeo y animación.

Entre las aplicaciones informáticas multimedia más comunes podemos mencionar juegos, programas de aprendizaje y material de referencia.

Un proyecto hipermedia reúne diversos profesionales con distinta formación, interés y visión, tales como ingenieros, diseñadores, publicistas, fotógrafos, escritores, locutores, productores, artistas, programadores, psicólogos, etc. Todos tienen su propio lenguaje técnico, sus propias prioridades y cada uno debe usar herramientas y aplicar sus funciones dentro del proyecto (SOLAR 2000).

Las empresas desarrolladoras de software reconocen la importancia de la utilización de métricas de estimación para el desarrollo de los proyectos. Para los productos hipermedia no se tiene excepciones. Una estimación realista en las etapas tempranas del ciclo de vida de un proyecto hipermedia garantiza a gerentes y desarrolladores en una organización manejar la información de manera efectiva.

Las aplicaciones hipermedia son centradas en tres pilares: datos, funcionalidad y navegación en perspectivas diferentes de acuerdo con la aplicación que está siendo desarrollada (MENDES 2001).

Varios analistas dentro del campo de la Informática han centrado su estudio en las métricas de estimación en los proyectos de Multimedia. A continuación les proponemos unas síntesis de sus ponencias.

**Guión (Storyboard)**

Solar en su artículo "Un Modelo Para Diseñar Storyboard en Proyectos Multimedia" propone un modelo denominado Generación Gráfica de Guiones (GGG o G3) para apoyar el diseño de guiones basado en métodos generados a partir de modelos de proceso de Ingeniería de Software, utilizando grafos dirigidos y describe sus componentes genéricos a través de un modelo de objetos. Aprovechando la información contenida en el grafo diseñado, es presentado un modelo para obtener reportes sobre los medios descritos en cada nodo, resumen sobre la totalidad de la aplicación, así como también el cálculo de una aproximación del tiempo de desarrollo de una aplicación, el costo de desarrollo del proyecto, y el espacio de almacenamiento requerido para el producto final, información de gran utilidad en la toma de decisiones en la fase de diseño de un producto con tecnología Multimedia.

**Estimación de métricas para Hipermedia Web**

Mendes en su artículo "Measurement, prediction and risk analysis for web applications" (Medición, predicción y análisis de riesgos para aplicaciones Web), presenta un modelo de predicción para aplicaciones hipermedia de autoría y web durante un estudio de casos realizado con un proyecto asignado a los estudiantes en el curso de hipermedia y multimedia de la Universidad de Auckland.

Reúne una serie de métricas de la literatura de ingeniería de software y multimedia con adaptaciones. Para dicho estudio aplicó dos cuestionarios para recolectar los datos. El primero cualifica la experiencia de los diseñadores en cinco escalas que va de ninguna experiencia (1) hasta muy buena experiencia (5). El segundo cuestionario mide las características de las aplicaciones desarrolladas y el esfuerzo involucrado en el diseño y autoría de estas aplicaciones.

**Modelo WEBMO**

El Modelo WEBMO de estimación surge en un intento por dar respuesta a las notables diferencias existentes entre las estimaciones de los desarrollos clásicos de software y las de los proyectos de Internet, modelo desarrollado por Donald Reifer (REIFER 2000). Este modelo de estimación se centra en la web en su aspecto estático y multimedia, es decir, en la aparición de diversos elementos denominados “objetos web” como puedan ser videos, sonidos, componentes Active-X y Applets entre otros, como elementos diferenciales a la hora de desarrollar software.

Se puede considerar a la Ingeniería del software como una disciplina en constante evolución, algo que se logra verificar con la aparición de las nuevas tecnologías de la comunicación y la información. Sin embargo no se puede comprobar una evolución similar en las técnicas y modelos de estimación de software. De ahí que se utilicen estos argumentos para aplicar los elementos expuestos en este trabajo en algunos de los proyectos de Multimedia de la Universidad de las Ciencias Informáticas.

## 2.5 Elementos de integración

Después de estudiar con detenimiento la esencia de las teorías de Barry Boehm y Watts Humphrey, y apoyados en el material de tesis de maestría de la MSc. Yadira Ruiz Constanten, se puede ver que efectivamente existen varios puntos de contacto, con los cuales se analizará cómo se podría integrarlos o combinarlos para mejorar las estimaciones de duración de proyectos de software.

### 2.5.1 Líneas de código (LOC)

La métrica de tamaño tradicional para estimar el esfuerzo de desarrollo y productividad ha sido las LOC (Lines Of Code). Se han propuesto varios modelos de estimación, la mayoría de ellos son funciones de las líneas de código o de las miles de líneas de código que tendrá el software a desarrollar (LISET 2006).

COCOMO II realiza la estimación de tamaño según el modelo que utilice; para Composición de Aplicaciones usa Puntos Objetos, para Diseño Temprano y Post-Arquitectura usa Puntos de Función,

(métrica que aplicaremos en nuestra investigación, explicada en el epígrafe 2.3). Tanto COCOMO II como PROBE tienen en cuenta las LOC como medida de tamaño del software y una vez estimado el tamaño es usado este valor en los cálculos necesarios para estimar la duración de tiempo de los proyectos de software.

Puntos de Función se centra en la funcionalidad, basándose en el diseño lógico del sistema, siendo complicada su aplicación si no se tiene dominio del método y de los conceptos que este maneja. Los Puntos de Función Desajustados son los que se convierten a LOC considerando el lenguaje de implementación (ensamblador, lenguajes de alto nivel, lenguajes de cuarta generación, etc.)

PROBE, se basa esencialmente en referencias de datos históricos sobre los proyectos similares que se han desarrollado previamente, así como en la opinión de expertos, para esto sería necesario contar con los datos históricos de proyectos realizados y/o con el personal lo suficientemente calificado para realizar las estimaciones.

Se puede pensar entonces en la posibilidad de combinar ambos métodos. Si se está ante un experto en la realización de aplicaciones específicas, se utilizaría su vasta experiencia para estimar la cantidad de LOC que tendría el sistema a desarrollar, sin necesidad de aplicar Puntos de Función para estimar tamaño y tener de esta manera otro valor con el cual comparar el resultado. De la misma forma se podría estudiar el comportamiento del método PROBE, introduciéndole la cantidad de LOC estimadas por Puntos de Función y comparar entonces ambos resultados.

### 2.5.2 Reutilización de código

La reutilización de código se refiere al comportamiento y a las técnicas que garantizan que una parte o la totalidad de un programa informático existente se puedan emplear en la construcción de otro programa. De esta forma se aprovecha el trabajo anterior, se economiza tiempo y se reduce la redundancia. Tanto COCOMO II como PROBE dan tratamiento al código reutilizado y lo tienen en cuenta para estimar el tamaño del software en LOC.

El tratamiento que hace COCOMO II del software reutilizado utiliza un modelo de estimación no lineal para calcular LDCF equivalentes a nuevo desarrollo (ESLOC) a través de:

$$ESLOC = ASLOC * \frac{\left(AA + AAF * (1 + 0.02 * SU * UNFM)\right)}{100},\ si\ AAF \leq 0.5$$

$$ESLOC = ASLOC * \frac{\left(AA + AAF + SU * UNFM\right)}{100},\ si\ AAF > 0.5$$

Esto implica que hay que estimar la cantidad de software que se va a adaptar, ASLOC y tres parámetros de grado de modificación:

- **ASLOC**: Cantidad de LDCF que se va a adaptar.
- **DM**: % de diseño modificado. El porcentaje de diseño de software que es modificado para adaptarlo a los nuevos objetivos y al entorno. (Esto es necesariamente una cantidad subjetiva).
- **CM**: % de código modificado. El porcentaje de código software adaptado que es modificado para adaptarlo a los nuevos objetivos y al entorno.
- **IM**: % de integración requerida para software modificado. El porcentaje de esfuerzo requerido para integrar el software adaptado dentro de la totalidad del producto y comprobar el producto resultante comparado con la cantidad de esfuerzo normal de integración y pruebas para software de un tamaño similar.

**AAF** (adaptation adjustment factor): Factor de ajuste de la adaptación, cuyo valor es:

**AAF** =0.4*DM + 0.3*CM + 0.3* IM

**AA** (assesment and assimilation): Grado de valoración y asimilación necesarios para decidir cuando un módulo software reutilizado por completo es apropiado para la aplicación.

**SU** (software understanding): % de esfuerzo de reutilización debido a la comprensión del software.

**UNFM** (programmer unfamiliarity): Indicador de la familiaridad del programador con el software.

Según PROBE la categoría de reutilizado es sólo para partes sin modificaciones. Cuando se modifican los programas existentes, el programa sin modificar es la base, y se va a estimar sus adiciones, cambios y supresiones. Incluso si el programa no se modifica, no se considera reutilizado a menos que se destine específicamente para su reutilización. La categoría reutilizados es sólo para las partes que entran directamente de la biblioteca de reutilización sin ninguna modificación.

**Programa Base:** Tamaño total del programa sin modificar antes el desarrollo.

**Adiciones de Base:** Tamaño del código añadido para mejorar el Programa Base.

**Supresiones:** Tamaño código que es eliminado del Programa Base por no ser necesario.

**Modificaciones**: LOC que se añaden al programa de partida (código existente) como parte de una modificación.

En ambas teorías, tras aplicar el método particular de cada una para hacerlo, el valor obtenido de estimar la cantidad de LOC reutilizadas es usado directamente para estimar el tamaño total del

software y este a su vez es usado en los cálculos necesarios para estimar el tiempo de duración de proyectos. Se podría entonces utilizar en COCOMO II la forma en que PROBE estima el código reutilizado y viceversa.

### 2.5.3 Multiplicador de Esfuerzo PREX

Como se ha explicado anteriormente este Driver de Coste del Modelo Diseño Temprano combina los siguientes parámetros de coste que se agrupan entre los Factores Personales del modelo Post-Arquitectura (Experiencia en la aplicación (AEXP), Experiencia en la plataforma (PEXP), y Experiencia en las herramientas y lenguajes (LTEX)), de forma general captura la experiencia del personal involucrado en el proceso de desarrollo.

Por su parte PSP se concentra en las prácticas de trabajo de los ingenieros de forma individual para desarrollar software de calidad, a través de la utilización de un proceso planeado, medido y bien definido, dándole mucho peso a la experiencia del individuo y el análisis de los datos históricos registrados de proyectos anteriores que sirven de guía y ayuda en el proceso de desarrollo.

Después de analizar lo anterior, se propone integrar las prácticas propuestas por PSP para controlar el trabajo, dominio y experiencia de los ingenieros sobre las herramientas, lenguaje, plataforma y aplicaciones específicas en que este trabaja o ha trabajado, teniendo así una medida de la madurez y las potencialidades que se pudiera aprovechar del individuo ante un proyecto con características específicas, pudiéndose distribuir las tareas y responsabilidades de manera eficiente, lo que podría aumentar la productividad y disminuir el esfuerzo en el proceso de desarrollo y a su vez el tiempo de terminación de los proyectos, y gracias a este control que proporciona PSP sobre el trabajo de los individuos se tendrían criterios y datos históricos para realizar fácil y confiables las estimaciones.

### 2.5.4 Madurez del Proceso. Factor PMAT

La Madurez del Proceso se calcula a través del factor PMAT, este es uno de los cinco factores que influyen exponencialmente en la productividad y esfuerzo de un proyecto de software, según COCOMO II.

Como se analizó anteriormente este factor se calcula de dos formas, en esta investigación se utilizarán las 18 Áreas del Proceso Clave, que establece CMM, determinando el nivel de conformidad para cada una de las KPA's (Tabla 7). Cuando se aplica el modelo evaluativo pueden ocurrir fundamentalmente dos situaciones que atenten contra las estimaciones y el proceso de desarrollo respectivamente. La primera de ellas es que la(s) persona(s) encargadas de determinar el porcentaje de conformidad para cada una de las KPA's no tengan precisión a la hora de dar los valores, debido principalmente al

desconocimiento del estado de las mismas y la segunda es que se pueden obtener valores de la variable PMAT que demuestren un esfuerzo que se podría disminuir si se aumentan los porcentajes de cumplimiento en las diferentes KPA's.

Ahora, PSP es el proceso de aplicación de los principios de CMM para ayudar y guiar a los ingenieros a realizar bien su trabajo, fue desarrollado para lograr un proceso que acercara CMM al individuo. Como se analizó anteriormente, PSP cubre parcialmente 12 de las 18 KPA's que define CMM (Figura 8), estas son las que están orientadas a los procesos que puede dársele un enfoque personal (para profundizar y entender mejor consultar el epígrafe 2.2).

Según lo analizado hasta aquí, se propone que se integren las prácticas de PSP al trabajo de los ingenieros del equipo de desarrollo. Podría ser esta una forma para lograr primeramente que exista precisión y seguridad en los valores ofrecidos de los porcentajes de conformidad para cada una de las KPA's, lo que representaría realmente el estado de madurez de la organización. En segundo lugar podrían aumentar los porcentajes de cumplimiento de cada una de las KPA's, pues en esto se centra PSP, intentar guiar a los ingenieros a que exista un mejoramiento notable en el proceso de desarrollo, lo que pudiera significar una disminución sustancial del esfuerzo necesario para el desarrollo de proyectos y por consiguiente en el tiempo de duración de estos.

## 2.6 Conclusiones

Durante el desarrollo de este capítulo se han analizado y descrito las principales características de COCOMO II y PSP. A pesar de ser el primero un modelo y el segundo un proceso, se han encontrado elementos en común entre ambos que se tienen en cuenta a la hora de realizar las estimaciones. Se han visto las características de las aplicaciones multimedia y algunos métodos que se usan en el mundo para realizar estimaciones de este tipo de proyecto. A partir del estudio realizado se propone cómo podrían integrarse cada uno de estos elementos para mejorar las estimaciones de duración de tiempo en los proyectos.

# CAPÍTULO 3

**Introducción**

Los capítulos anteriores se han centrado en el estudio de métodos de estimación y elementos significativos de estos a través de las teorías planteadas por Humphrey y Boehm. En este capítulo se pondrán a consideración los resultados obtenidos en las entrevistas realizadas a los Líderes de proyectos de Multimedia de la facultad 8, donde se pudo comprobar el estado actual de las estimaciones en los proyectos de Multimedia de la Universidad. Teniendo en cuenta los elementos comunes de ambas teorías, se describe cómo pudiera ser la posible integración entre ellos, lo que podría dar un mejor resultado en la estimación de los proyectos. Por último, se validará la posible integración en cuanto a estos elementos comunes.

## 3.1 Aplicación de la entrevista

Se realizó una entrevista a los principales líderes de proyectos, arquitectos, planificadores y programadores líderes de proyectos de Multimedia de la Facultad 8 (Ver Anexo 1), con el objetivo de reunir información de personal calificado en nuestra Universidad, ver los distintos criterios que ellos tienen sobre los elementos importantes en el proceso de estimación de proyectos de software, fundamentalmente en aplicaciones Multimedia y cómo ellos realizan las estimaciones en sus respectivos proyectos.

De forma general, en todos los proyectos se realizan estimaciones y estas se tienen en cuenta para hacer la Planificación, coincidiendo todos en que para planificar, la estimación es un elemento fundamental. Sin embargo, no se posee un método o modelo definido para realizar las mismas. La entrevista demostró que se basan fundamentalmente en la experiencia acumulada de los miembros del equipo, los cuales llevan varios años trabajando en esos proyectos productivos dentro de la facultad, y el análisis conjunto de los principales dirigentes, teniendo en cuenta la complejidad del proyecto y la fuerza de trabajo con que cuentan, o sea, se utiliza un método prácticamente empírico. Sin embargo, aún contando el tiempo que llevan dedicándose a los proyectos de Multimedia, no se puede decir que son expertos en la materia, pues dada la diversidad de lenguajes y de técnicas de programación que surgen por día para el desarrollo de Multimedia, es muy difícil utilizar este término de expertos.

Se realizan varias estimaciones durante el ciclo de vida del proyecto. A partir de una estimación inicial se va redefiniendo la misma en dependencia de los cambios que vayan surgiendo durante el desarrollo del software, que pueden ser muchos debido al dinamismo de la Universidad que tiene su fuerza de

trabajo fundamental en los estudiantes, cuyo principal objetivo es la docencia, de ahí que no estén vinculados a tiempo completo a la producción.

Existe poco conocimiento acerca del Proceso Personal de Software (PSP) y de las ventajas que puede brindar este método si se utiliza en el proyecto para guiar a los desarrolladores, alegando que nunca les ha hecho falta para cumplir sus objetivos, de ahí que en ninguno de los casos se han tenido en cuenta las prácticas de PSP en los equipos de desarrollo. Sin embargo, con los conocimientos generales que tienen del tema, logran coincidir en que sería una buena práctica, porque administrando su tiempo comprenderían la cantidad de horas que le dedican a cada actividad y eso sería una buena base para estimar el trabajo individual del individuo y luego dentro del equipo de trabajo.

Reconocen la influencia de los factores de escala en el esfuerzo y la productividad del proyecto, pero de forma inconsciente, pues no existe una forma definida de medir cómo influyen estos cuando se realizan las estimaciones. De modo general analizan los factores de la siguiente manera:

PREC (Precedencia): Todos le dan una gran importancia a este factor de escala, pues tienen muy arraigado el concepto de precedencia para cada una de las actividades que realizan. De ahí dependen, en gran medida, las mediciones que realizan durante la vida del proyecto.

FLEX (Flexibilidad): Tiene también una gran importancia para ellos, aunque coinciden indistintamente en aspectos importantes, por ejemplo: No son muy estrictos para los plazos de tiempos de entrega del producto final, pero sí bastante en cuanto a los requerimientos, de ahí que la flexibilidad entre a jugar un papel importante dentro de los proyectos de Multimedia de la Universidad.

RESL (Arquitectura y Determinación del Riesgo): Este aspecto también tiene una gran importancia para ellos, pues realizan Planes de mitigación de riesgos una vez que se inicia la vida del mismo, en dependencia de los roles que se desempeñan dentro del proyecto.

TEAM (Cohesión del equipo): Este es el factor común para los entrevistados, pues coinciden en que juega un papel fundamental para lograr los objetivos del mismo. Que todos se sientan identificados con el trabajo que realizan contribuye en gran medida, al éxito del trabajo final.

**Líneas de código**

No se tiene un concepto definido de Líneas de código. La variante a nivel de programas más aceptada de las propuestas en la entrevista es "**Líneas de código, definiciones de datos y comentarios"**. Por

su parte, la más aceptada a nivel de proyecto es "**Contar líneas nuevas, líneas modificadas y líneas reutilizadas"**. No se realizan estimaciones de tamaño del software y por ende no se calcula la cantidad de LOC, pues no se tienen en cuenta para estimar esfuerzo y tiempo de duración del proyecto.

**Reutilización**

La reutilización de código se usa en un alto grado. Todos coinciden en que la reutilización depende del lenguaje de programación que vayan a utilizar, de ahí que traten siempre de implementar de modo genérico, lo que les permitirá el día de mañana utilizar las partes reutilizables. En la totalidad de los casos, independientemente de las funcionalidades que deba tener el software existe la posibilidad de reutilizar código gracias a la existencia de librerías, aplicaciones anteriores y el desarrollo de arquitecturas base, lo que resulta de mucha ayuda para disminuir el esfuerzo y el tiempo de duración del proyecto. A pesar de la importancia de la reutilización y el desarrollo siempre que sea posible de código reutilizable, no se tiene control de la cantidad de LOC destinadas a ser reutilizadas, simplemente las usan en la medida de lo posible, sin pensar en que pudiera ser importante medirlas para estimar o incluso podría ser hasta más costoso si la cantidad a reutilizar no es muy grande y entender y adaptar requiera más esfuerzo que desarrollar desde cero. En resumen, como ocurre con Líneas de Código, la reutilización da tamaño de la aplicación y este no se tiene cuenta en el proceso de estimación.

**Madurez del Proceso. Factor PMAT**

Tienen en cuenta algunas de las áreas de procesos claves dentro de la madurez de procesos y lo aplican en el proyecto, aunque de forma inconsciente. Trabajan mucho con la experiencia y sobre todo que se trabaje bien. A continuación se relacionan algunas de las áreas que trabajan fundamentalmente:

- Administración de requerimientos.
- Planificación del Proyecto de Software.
- Seguimiento y supervisión del Proyecto de Software.
- Coordinación entre grupos.
- Aseguramiento de la calidad.

- Prevención de defectos.
- Administración de las Tecnologías de Cambio.

Vale destacar además, que muchos ven la tecnología como un factor que puede influir también en el esfuerzo y la productividad del proyecto, pues se manejan términos como transacciones, video, audio, tratamiento de imágenes, cantidad de PCs, etc; que pudieran ser relevantes para una estimación final. Si no se tienen en cuenta estos elementos para realizar las estimaciones como características propias de los proyectos de Multimedia, las estimaciones no serían del todo confiables, porque se estarían descuidando aspectos significativos que pueden influir en gran medida en la productividad y el esfuerzo de un proyecto de software para aplicaciones Multimedia.

**Multiplicador de Esfuerzo PREX**

En la totalidad de los casos se considera muy importante la experiencia en el proceso de desarrollo, aunque a veces no saben con exactitud el tiempo que llevan desarrollando aplicaciones multimedia, lo que podría ser causa de no llevar un control personal de las actividades que realizan. No se tiene una forma para medir la evolución personal que pueda dar una idea del estado de conocimiento del individuo, a pesar de todo en ninguno de los casos se aplican las prácticas de PSP lo que podría ayudar a resolver estos problemas.

## 3.2 Validación de la propuesta

La entrevista aplicada a los líderes de proyectos de multimedia demuestra el estado de los procesos de estimación en dichos proyectos. No solo en estos, sino en la universidad de forma general, las estimaciones a veces son un puro formalismo para crear un cronograma de trabajo que se reajusta tantas veces como sea necesario. Intentando solucionar este problema en el epígrafe 2.5 se describió la propuesta de integración de las teorías de Boehm y Humphrey a partir de los puntos de contacto encontrados entre ambas, la cual podría ser esta adoptada y adaptada por los equipos de desarrollo.

Ante cada indagación científica nos enfrentamos al reto de demostrar la veracidad de nuestras investigaciones. Muchas veces la práctica se convierte en un método seguro, pero cuando la búsqueda es completamente teórica o no existe un conjunto de datos históricos útiles en los cuales pueda basarse un análisis. ¿Cómo demostrar la fidelidad de nuestra propuesta?. Esta interrogante se aclara en este apartado, donde se valida la propuesta de integración aplicando el método de Consultas a

Expertos o Criterios de Expertos, muy utilizado hoy a nivel mundial para demostrar la veracidad teórica de una investigación.

### 3.2.1 Resultados

Se realizó una encuesta a 8 expertos seleccionados según los criterios de evaluación que propone el método, el promedio del Coeficiente de Conocimiento o Información del grupo es de 0.8, lo que clasifica como alto.

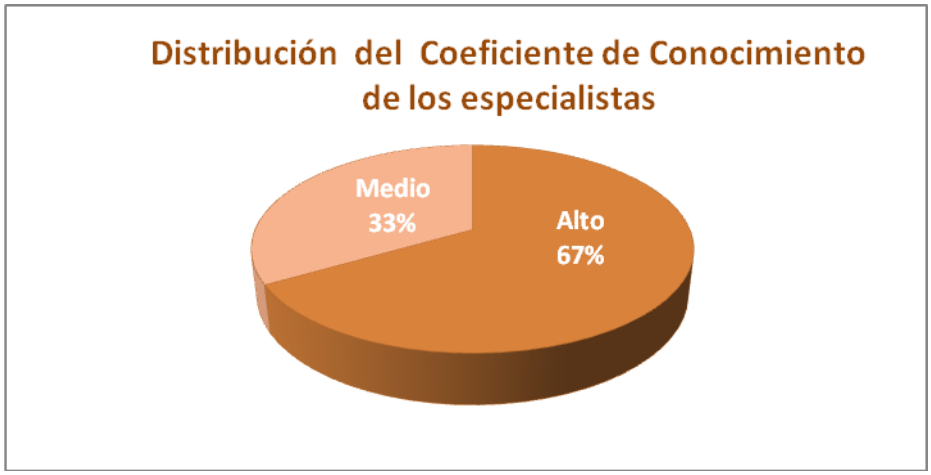


Figura 12. Porcentajes de distribución del Coeficiente de Conocimiento.

Se aprecia un dominio en la cantidad de expertos con nivel alto en el Coeficiente de Conocimiento, lo que da mucho peso a las valoraciones y criterios recogidos.

La encuesta fue realizada sobre la base de la propuesta de integración y los argumentos en que esta se basa. (Anexo 2).
Los resultados obtenidos se procesaron por cada uno de los elementos de integración que se proponen.

**Líneas de Código**

El cien por ciento de los expertos coincide en la importancia del tamaño del software en el proceso de estimación siendo un elemento a considerar siempre, y ubican las Líneas de Código como la menor unidad de medida para realizar estimaciones de proyectos de software.

Los gráficos siguientes demuestran la aceptación de la integración en cuanto a utilizar en COCOMO II la forma en que PROBE estima tamaño y viceversa.

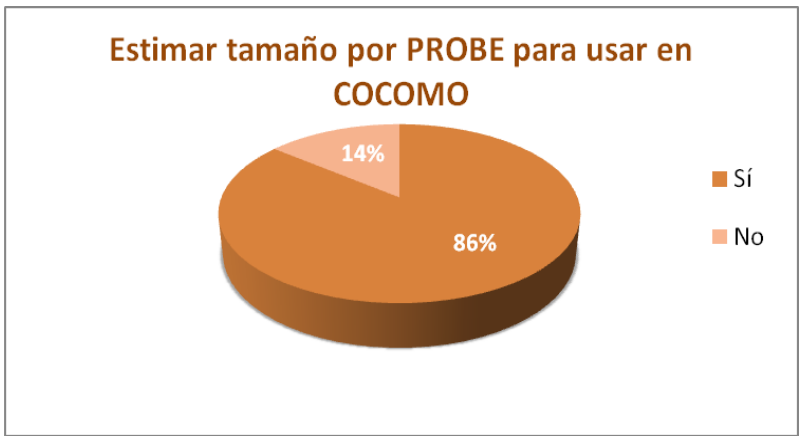


Figura 13. Porcentajes de aceptación de utilizar PROBE para usar en COCOMO II.

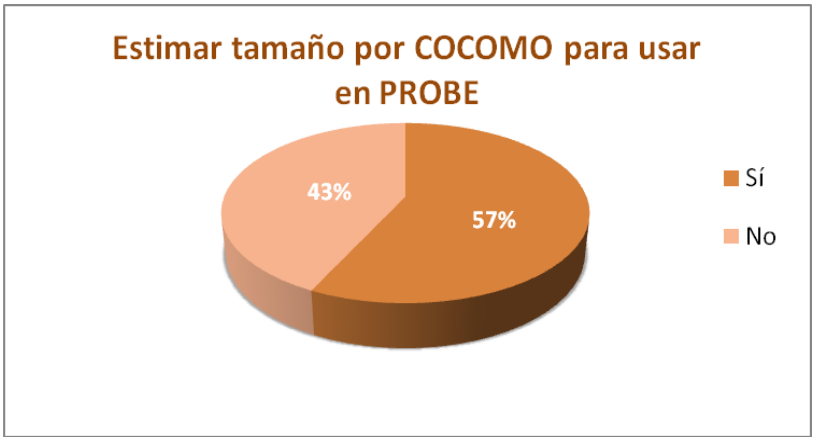


Figura 14. Porcentajes de aceptación de utilizar Puntos de Función en PROBE.

Respecto a la propuesta de integración de estimar tamaño como lo hace un método para utilizarla en el otro, el cien por ciento de los encuestados la consideran válida, aludiendo además que sería muy interesante llevarla a la práctica.

**Reutilización**

El cien por ciento de los encuestados considera la reutilización como un elemento importante para realizar las estimaciones, explican que es un arma para disminuir el esfuerzo y tiempo de desarrollo de los proyectos de software y fundamentalmente en Proyectos de Multimedia.

Los gráficos siguientes demuestran la aceptación de la integración en cuanto a utilizar en COCOMO II la forma en que PROBE estima el tamaño del código reutilizado y viceversa.

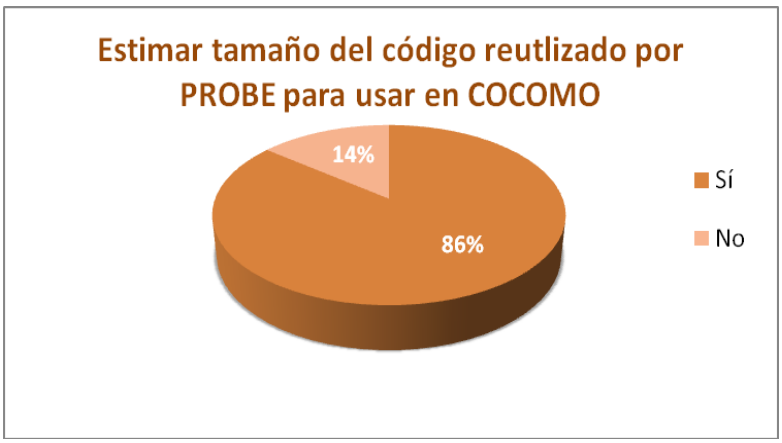


Figura 15. Porcentajes de aceptación de utilizar PROBE para usar en COCOMO II para estimar el código reutilizado.

La gran mayoría de los encuestados valida la posibilidad de utilizar en COCOMO II la forma en que PROBE realiza las estimaciones de código reutilizado.


Figura 16. Porcentajes de aceptación de utilizar COCOMO II para usarlo en PROBE para estimar el código reutilizado.

Aunque no es una diferencia tan marcada como la anterior, la mayoría de los expertos consideran que también se puede integrar la forma en que COCOMO II estima el código reutilizado y utilizarlo en PROBE.

Respecto a la propuesta de integración de estimar el tamaño del código reutilizado como lo hace un método para utilizarla en el otro, el cien por ciento de los encuestados la consideran válida, argumentando además que sería muy interesante llevarla a la práctica.

**Madurez del Proceso. Factor PMAT**

Se realizó un análisis de la influencia en la productividad y el esfuerzo de las organizaciones de las 12 áreas de procesos que trata PSP, la siguiente tabla refleja el criterio de los expertos respecto a esto.

| Áreas de Procesos (KPA's) | 0 | 1 | 2 | 3 | 4 | 5 | Índice de Influencia |
|---|---|---|---|---|---|---|---|
| Planeación de Proyectos | | | | | | 8 | 5,00 |
| Seguimiento y Control | | | | 2 | 1 | 5 | 4,38 |
| Foco del Proceso | | | 2 | 1 | 3 | 2 | 3,63 |
| Definición del proceso de SW | | | | | 1 | 7 | 4,88 |
| Manejo Integrado | | | 1 | 2 | 2 | 3 | 3,88 |
| Ingeniería del producto | | | | | 1 | 7 | 4,88 |
| Revisión entre colegas | | | 1 | 2 | 3 | 2 | 3,75 |
| Administración cuantitativa | | | 1 | 1 | 4 | 2 | 3,88 |
| Control Calidad | | | | | 3 | 5 | 4,63 |
| Prevención de defectos | | | | | 1 | 7 | 4,88 |
| Administración del cambio tecnológico | | | 1 | | 1 | 6 | 4,50 |
| Administración de los cambio del proceso | | | 1 | | 1 | 6 | 4,50 |
| TOTAL | 0 | 0 | 7 | 8 | 21 | 60 | **4.40** |

Tabla 28. Índice de influencia de las Áreas de Procesos en la productividad y el esfuerzo de la organización.

El índice de influencia total es de 4.40. Esto muestra cuánto influyen de forma general las áreas de procesos en la productividad y el esfuerzo de las organizaciones, lo que implica que no se pueden dejar de tener en cuenta en las estimaciones y que hay que encontrar variantes para mejorar estos indicadores, siendo la aplicación de las prácticas de PSP un camino seguro y al alcance de nosotros para poner estos elementos a nuestro favor.

Entre los objetivos de la propuesta se plantea que aplicar las prácticas de PSP permite realizar estimaciones más precisas con respecto al factor PMAT. La siguiente gráfica muestra el criterio de los expertos acerca de cómo serían las estimaciones al aplicarse PSP.

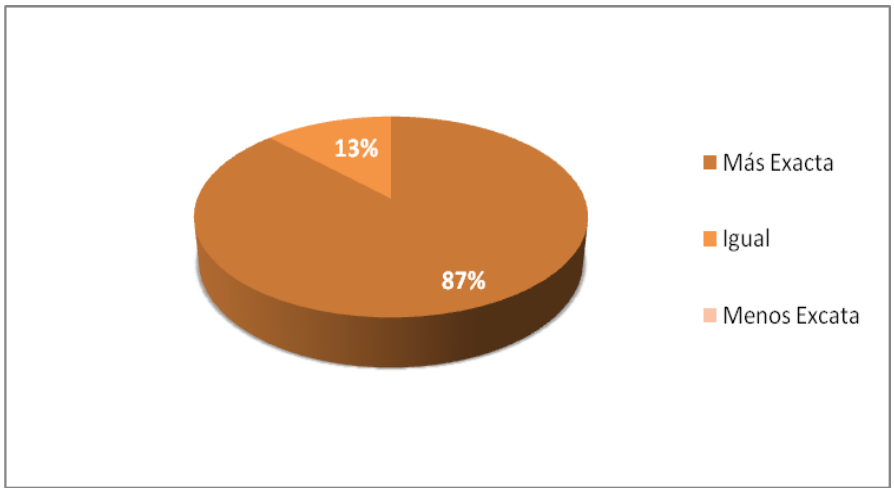


Figura 17. Porcentaje de aceptación de utilizar las prácticas de PSP para mejorar la exactitud de las estimaciones.

Como se aprecia, la gran mayoría de los expertos coincide en que habría un aumento en la precisión de las estimaciones, puesto que PSP en su aplicación conduce entre otras cosas, a dominar el estado en que se encuentra el cumplimiento de cada una de las áreas de procesos que trata.

**Multiplicador de Esfuerzo PREX**

La experiencia es un factor determinante en un equipo de desarrollo de aplicaciones multimedia y en esto concuerda el cien por ciento de los encuestados, por lo que es necesario manejar este término en nuestras estimaciones. Se aclara que no se estima la experiencia, se estima el esfuerzo basándonos en la experiencia.

Ahora bien, la experiencia sin una forma de gestionarla no se podrían realizar estimaciones confiables en las que se tuviera en cuenta este elemento, además de que sin un control no se aprovecharía al máximo dejando escapar la posibilidad de seguir mejorando y facilitando el proceso de desarrollo. El cien por ciento de los expertos valida que aplicar las prácticas de PSP es una buena forma de gestionar y controlar la experiencia y las habilidades del individuo para aprovecharlas al máximo y contribuir así a mejorar la calidad de las estimaciones en los proyectos y el proceso de desarrollo, sobre todo en aplicaciones multimedia.

**Propuesta General**

La propuesta de integración de forma general tuvo una aceptación del cien por ciento de los expertos, alegando las facilidades y flexibilidades que presenta, sin comprometer la calidad de las estimaciones,

permitiendo en dependencia del conocimiento y los datos que se posean, elegir la manera de realizar las estimaciones.

Resumiendo las respuestas de la encuesta se pudo elaborar un modelo de estimación atendiendo a la tendencia del criterio de los expertos. A continuación se muestra un análisis de la preferencia de los encuestados en cuanto a los métodos a utilizar.

Para la estimación de tamaño la siguiente gráfica muestra la preferencia de los expertos:

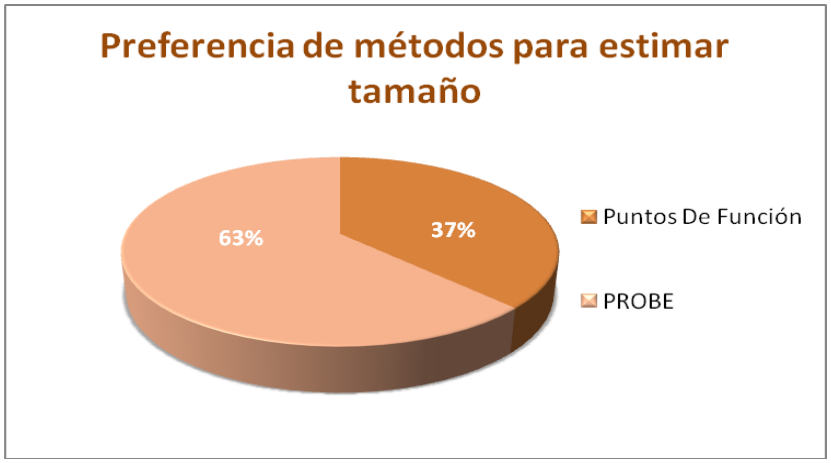


Figura 18. Preferencia de métodos para estimar tamaño.

Para estimar el tamaño del código reutilizado la siguiente gráfica muestra la preferencia de los expertos:

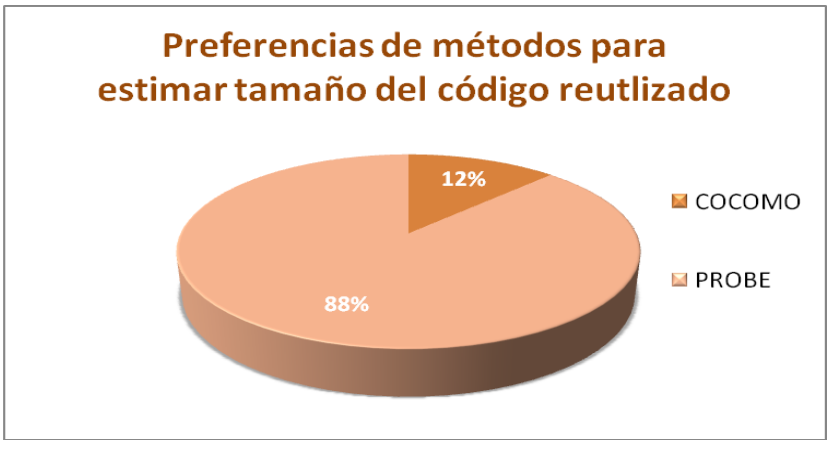


Figura 19. Preferencia de métodos para estimar tamaño del código reutilizado.

Todos coinciden en mantener la línea base de COCOMO II para realizar las estimaciones por ser un modelo robusto, abarcador, bien documentado y por su utilización a nivel mundial, además COCOMO II tiene en cuenta elementos que afectan el proceso de desarrollo y que en las estimaciones es difícil relacionar, como por ejemplo, la experiencia, la madurez de la organización y las relaciones personales del equipo.

El 63% de los expertos prefiere utilizar PROBE para estimar tamaño, contando con que existan los datos históricos necesarios, por su facilidad y rapidez, además para proyectos de multimedia es el que más se ajusta por las características específicas de estos, sin dejar de reconocer que Puntos de Función puede ser la herramienta a usar cuando no se tengan los elementos necesarios.

El 88% de los expertos prefiere utilizar PROBE para estimar el tamaño del código reutilizado, por la comodidad, rapidez y facilidad que brinda, además la forma en que COCOMO II hace esto puede ser complicada por algunas variables que maneja.

La siguiente figura muestra como sería la integración según las preferencias de los expertos.

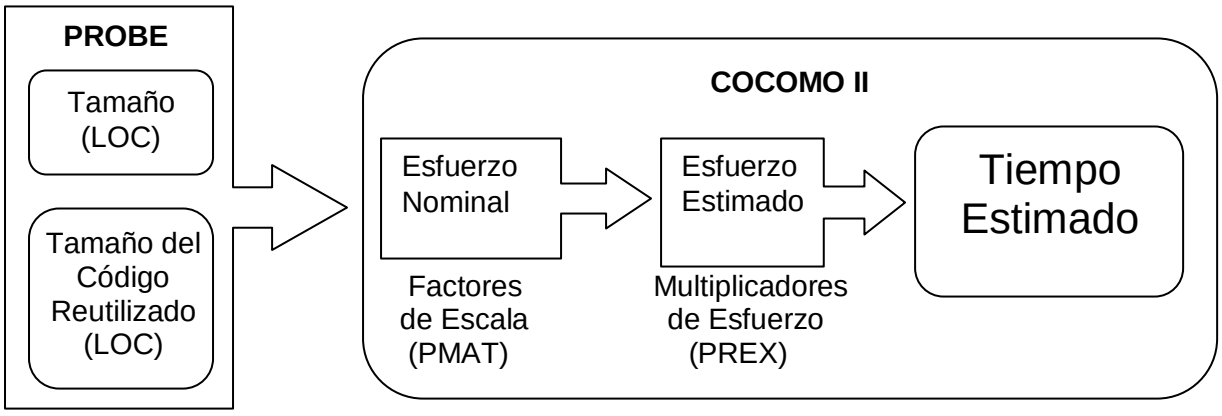


Figura 20. Modelo de integración según preferencia de los expertos.

Debe quedar claro que esta preferencia es según los resultados de la encuesta y no la que se tiene que aplicar. La validación ha demostrado que en cada organización se pueden utilizar los elementos de cualquiera de los métodos, en dependencia de las preferencias, el conocimiento y las herramientas que poseen los encargados de realizar las estimaciones.

**Otras consideraciones**

Las encuestas realizadas a los expertos, sirvieron para obtener otros resultados que pueden ser de gran utilidad a la hora de hacer las estimaciones en proyectos de software, específicamente para aplicaciones Multimedia, debido a la pecualiaridad que ellas presentan, por ejemplo:

- No debe centrarse en las aplicaciones Multimedia todo el peso en el software.
- Es una obra de comunicación. Debe valorarse el diseño, la comunicación interactiva, el procesamiento de materiales audiovisuales, etc.
- El peso más importante lo tiene la experiencia. Luego la reutilización y en último lugar la aptitud de las personas.
- Las multimedias para Web usan restricciones que las otras aplicaciones de este tipo no. Estas restricciones pueden ser las relacionadas con la cantidad de información , banners, flash, etc, que se coloque en este tipo de aplicaciones, lo que que haría más lento el proceso de levantar la aplicación y por tanto puede atentar contra el interés del usuario de interactuar con ella, debido al tiempo que se demora la llamada.

**Conclusiones**

En este capítulo se han recogido los principales problemas que presenta el proceso de estimación de aplicaciones Multimedia en nuestra Universidad, a través de una entrevista realizada a los líderes de estos proyectos. Además se validó la propuesta de integración realizada en el capítulo anterior aplicando el Método de Expertos, utilizando la encuesta como instrumento para el intercambio con los especialistas. Se propone un modelo de estimación como resultado de la integración según la preferencia de los expertos. Se mencionan consideraciones importantes a tener en cuenta en las estimaciones de las aplicaciones de Multimedia.

# CONCLUSIONES

Después de haber desarrollado esta investigación se pudo comprobar cómo la estimación es la base de la Planificación, aportando elementos que influyen sustancialmente a obtener un software de calidad. Proponer una solución válida para realizar estimaciones confiables fue el principal reto, de ahí que se abordaran distintas teorías utilizadas a nivel mundial para el buen desempeño de las mismas.

Se analizó y estudió Puntos de Función como la métrica para calcular el tamaño del proyecto expresado en líneas de código, que sirve como entrada a COCOMO II para calcular el esfuerzo nominal y luego el estimado, que finalmente servirán para calcular el tiempo de desarrollo del software.

Se estudió a PROBE como el método que propone PSP para hacer las estimaciones, a partir de datos históricos almacenados, lo que resulta de vital importancia si tenemos en cuenta que muchos proyectos se basan en la experiencia para realizar las estimaciones.

Se reconoció la robustez de COCOMO II, al manejar y relacionar todos los elementos que directa o indirectamente pueden influir en el proceso de desarrollo y por ende en las estimaciones.

La madurez en los procesos es un elemento fundamental en las organizaciones, así como la experiencia de los individuos y la calidad en el trabajo personal. Estos factores están muy bien enfocados en los planteamientos y prácticas de PSP, donde se pudo ver cómo ayuda al individuo a planificar, controlar y medir su trabajo.

La investigación requería el estudio profundo de los elementos en común que proponen las teorías planteadas por Boehm y Humphrey, y el modo de integrarlos con el fin de dar una propuesta que sirviera a mejorar las estimaciones en la Universidad, donde se demostró finalmente que la integración es válida en cualquier sentido que se trate, lo que queda a la preferencia de las personas encargadas de realizar las estimaciones en los equipos de desarrollo, atendiendo además a los recursos y disponibilidades con que cuenten.

Los proyectos de Multimedia no pueden ser vistos simplemente como un software de gestión, sino que tienen características propias que hacen que los encargados de hacer las estimaciones en los equipos de desarrollo las tengan en cuenta para hacer estimaciones mucho más confiables.

# RECOMENDACIONES

- Estudiar y aplicar las prácticas que propone PSP a los equipos de desarrollo de aplicaciones Multimedia de la Universidad de las Ciencias Informáticas.
- Tabular los datos necesarios para la aplicación de PROBE en los proyectos de Multimedia.
- Aplicar esta propuesta en los proyectos de Multimedia de la Universidad, con el fin de obtener mejores resultados en el proceso de estimación.

# REFERENCIAS BIBLIOGRÁFICAS


1. *ALBRECHT Estudio de técnicas basadas en Puntos de Función para la estimación del esfuerzo en proyectos de software, 1979.*
2. *BOEHM, B. COCOMO Suite Methodology and Evolution, 2005.*
3. *Boletín de Política Informática Núm. 6, 2003 Puntos por Función. Una métrica estándar para establecer el tamaño del software. Núm. 6, 2003.*
4. *CAO, M. I. J. I. Principios para un método de estimación de proyectos de software basado en los escenarios principales, 2006. Disponible en: http://www.itba.edu.ar/capis/epg-tesis-y-tf/cao-trabajofinaldeespecialidad.pdf*
5. *CONCEPCIÓN, P. Planificación de Proyectos de Software, 2007. [2007]. Disponible en: http://www.getec.etsit.upm.es/articulos/gproyectos/art4.htm*
6. *DEKKERS, C. "Measuring the logical or functional size of software projects and software application". ISO Bulletin May 2003.*
7. *FRAN J. RUIZ-BERTOL, J. D. Gestión Activa de Eventos en Proyectos Software*, 2004. [2007]. Disponible en: http://www.sc.ehu.es/jiwdocoj/remis/docs/FranAdis2004.pdf*
8. *GIRALDO, O. P. Métricas, Estimación y Planificación en Proyectos de Software, 2005. [Disponible en: http://www.willydev.net/descargas/willydev_planeasoftware.pdf*
9. *GONZÁLEZ-FANJUL1, C. A.; G. M. M. HUERTA, et al. Proyectos Software. Estimación del Coste, 1997.*
10. *GONZÁLEZ, F. R. Modelo de Estimación de Costes para proyectos software. Mayo, 1999. p.*
11. *HUMPHREY, W. Introducción al Proceso de Software Personal, 2001.*
12. *LISET. Planificación y estimación - Modelos de Estimación, 2006. [Disponible en: http://www.wikilearning.com/articulo/planificacion_y_estimacion-modelos_de_estimacion/9635-4*
13. *MARTIG, S. Administración y Gestión de Proyectos de Software Medidas Cotidianas. 2005. p.*
14. *MATEUS FERREIRA A, F. G. A., FRANCISCO RUIZ A, MANUEL F. BERTOA B, CORAL CALERO A, ANTONIO VALLECILLO B, MARIO PIATTINI A, BEATRIZ MORA A. Medición del Software Ontología y Metamodelo, 2006.*
15. *MENDES, E., FEWSTER, R., . Measurement, prediction and risk analysis for web applications", Proceedings of IEEE Metrics Symposium -7th International Software metrics Symposium, IEEE CS Pres. 2001. p.*
16. *PRESSMAN, R. S. Ingeniería de software. Un enfoque practico. 2005. p.*
17. *REIFER, D. J. "Web Development: Estimating Quick-to-Market Software," IEEE Computer Society. 2000. p.*

18. *ROQUE, F. P. Sitio del ministerio de relaciones exteriores de cuba 2005. [Disponible en: http://www.cubaminrex.cu/Sociedad_Informacion/Cuba_TIC/Informatizaci%C3%B3n.htm*

19. *S.-CAPUCHINO, A. M. M. Estimación de Proyectos Software, 1996.*

20. *SOLAR, M., VERDUGO, P., PARADA, P. "Un Modelo para Diseñar Storyboard en Proyectos Multimedia" - International Journal on Computer Applications in Engineering Education, Vol. 8, num. 3 and 4, pp. 221-228. 2000. p.*

21. *VARGAS, D. El cliente, factor determinante del éxito de un proyecto de software., 2006. [2007]. Disponible en: http://www.eltiempo.com/participacion/blogs/default/un_articulo.php?id_blog=3150&id_recurso=3268694*

22. *WIKIPEDIA. Gestión de proyectos, 2007a. [2007]. Disponible en: http://es.wikipedia.org/wiki/Gesti%C3%B3n_de_proyectos*

23. *WIKIPEDIA .Métrica, 2007b Disponible en: http://es.wikipedia.org/wiki/M%C3%89TRICA.*

24. *WIKIPEDIA .Proyecto, 2007c. Disponible en http://es.wikipedia.org/wiki/Proyecto.*

# BIBLIOGRAFÍA


1. *ALBRECHT Estudio de técnicas basadas en Puntos de Función para la estimación del esfuerzo en proyectos de software, 1979.*
2. *BOEHM, B. COCOMO Suite Methodology and Evolution, 2005.*
3. *Boletín de Política Informática Núm. 6, 2003 Puntos por Función. Una métrica estándar para establecer el tamaño del software. Núm. 6, 2003.*
4. *CAO, M. I. J. I. Principios para un método de estimación de proyectos de software basado en los escenarios principales, 2006. [2007]. [Disponible en: http://www.itba.edu.ar/capis/epg-tesis-y-tf/cao-trabajofinaldeespecialidad.pdf*
5. *CONCEPCIÓN, P. Planificación de Proyectos de Software, 2007. [2007]. [Disponible en: http://www.getec.etsit.upm.es/articulos/gproyectos/art4.htm*
6. *DEKKERS, C. "Measuring the logical or functional size of software projects and software application". ISO Bulletin May 2003.*
7. *FRAN J. RUIZ-BERTOL, J. D. Gestión Activa de Eventos en Proyectos Software*, 2004. [2007]. [Disponible en: http://www.sc.ehu.es/jiwdocoj/remis/docs/FranAdis2004.pdf*
8. *GIRALDO, O. P. Métricas, Estimación y Planificación en Proyectos de Software, 2005. [Disponible en: http://www.willydev.net/descargas/willydev_planeasoftware.pdf*
9. *GONZÁLEZ-FANJUL1, C. A.; G. M. M. HUERTA, et al. Proyectos Software. Estimación del Coste, 1997.*
10. *GONZÁLEZ, F. R. Modelo de Estimación de Costes para proyectos software. Mayo, 1999. p.*
11. *HUMPHREY, W. Introducción al Proceso de Software Personal, 2001.*
12. *LISET. Planificación y estimación - Modelos de Estimación, 2006. [Disponible en: http://www.wikilearning.com/articulo/planificacion_y_estimacion-modelos_de_estimacion/9635-4*
13. *MARTIG, S. Administración y Gestión de Proyectos de Software Medidas Cotidianas. 2005. p.*
14. *MATEUS FERREIRA A, F. G. A., FRANCISCO RUIZ A, MANUEL F. BERTOA B, CORAL CALERO A, ANTONIO VALLECILLO B, MARIO PIATTINI A, BEATRIZ MORA A. Medición del Software Ontología y Metamodelo, 2006.*
15. *MENDES, E., FEWSTER, R. Measurement, prediction and risk analysis for web applications", Proceedings of IEEE Metrics Symposium -7th International Software metrics Symposium, IEEE CS Pres. 2001. p.*
16. *PRESSMAN, R. S. Ingeniería de software. Un enfoque practico. 2005. p.*
17. *REIFER, D. J. "Web Development: Estimating Quick-to-Market Software," IEEE Computer Society. 2000. p.*

*18. ROQUE, F. P. Sitio del ministerio de relaciones exteriores de cuba 2005. [Disponible en: http://www.cubaminrex.cu/Sociedad_Informacion/Cuba_TIC/Informatizaci%C3%B3n.htm*

*19. S.-CAPUCHINO, A. M. M. Estimación de Proyectos Software, 1996.*

*20. SOLAR, M., VERDUGO, P., PARADA, P. "Un Modelo para Diseñar Storyboard en Proyectos Multimedia" - International Journal on Computer Applications in Engineering Education, Vol. 8, num. 3 and 4, pp. 221-228. 2000. p.*

*21. VARGAS, D. El cliente, factor determinante del éxito de un proyecto de software., 2006. [2007]. [Disponible en: http://www.eltiempo.com/participacion/blogs/default/un_articulo.php?id_blog=3150&id_recurso=3268694*

*22. WIKIPEDIA. Gestión de proyectos, 2007a. [2007]. Disponible en: http://es.wikipedia.org/wiki/Gesti%C3%B3n_de_proyectos*

*23. WIKIPEDIA .Métrica, 2007b [Disponible en: http://es.wikipedia.org/wiki/M%C3%89TRICA.*

*24. WIKIPEDIA .Proyecto, 2007c. [Disponible en http://es.wikipedia.org/wiki/Proyecto.*

*25. SOLANS, Manuel. "PMI, una referencia en la Gestion de Proyectos." febrero 2005*

*26. GONCALVES Matias. "Desarrollo de un nuevo Modelo de Estimación basado en Metodología ágil de Desarrollo y Generadores de Aplicaciones". 2005*

*27. SALVETTO, Pedro; Nogueira, Juan Carlos; Segovia, Javier. "Métodos formales de estimación de tiempo y esfuerzo adaptables a los cambios en proyectos software". 2005*

*28. Gómez, Adriana; López María del C.; Migani, Silvina; Otazú, Alejandra. "COCOMO. Un modelo de estimación de proyectos de software." 2000*

# GLOSARIO DE TÉRMINOS Y SIGLAS

## A

**ANSI:** American National Standards Institute. Es una organización que supervisa el desarrollo de estándares para productos, servicios, procesos y sistemas en los Estados Unidos.

**Ada:** American National Standard Institute

**Applet:** Es un componente de una aplicación que se ejecuta en el contexto de otro programa, por ejemplo un navegador web

**Active-X:** Son controles particulares de Internet Explorer. Es una tecnología de Microsoft para el desarrollo de páginas dinámicas. Tiene presencia en la programación del lado del servidor y del lado del cliente

## B

**Borland:** Borland Software Corporation. Es una compañía de software, conocida sobre todo por sus herramientas de programación

**Batch:** Es el modo por lotes, que ejecuta una secuencia predefinida de comandos guardada como un archivo de texto con la extensión .BAT

**Banners:** Pueden ser estáticos o animados, son de formatos rectangulares, cuadrados e irregulares, se realizan en flash y sirven para la publicidad

## C

**COCOMO:** Modelo de estimación de costos

**CMM:** Modelo de Madurez de Capacidad

**Caja negra:** Se denomina a las pruebas que le hacen al software a partir de las entradas y salidas que se producen en un sistema informático

**C++:** Lenguaje de programación

**Coeficiente de Conocimiento o Información:** Es una medida del nivel de conocimiento o información que posee un especialista acerca de un tema específico.

## D

**DSI:** Delivered source instructions

## F

**Framework:** Representa una arquitectura de software que modela las relaciones generales de las entidades del dominio. Provee una estructura y una metodología de trabajo la cual extiende o

utiliza las aplicaciones del dominio. Son diseñados con el intento de facilitar el desarrollo de software

**FPA:** Function Point Analysis

**Fortran:** Lenguaje de programación

**FC:** Factores de complejidad

**Flash:** Aparecen muy a menudo como animaciones en páginas Web y sitios Web multimedia. Son también ampliamente utilizados en anuncios de la web

## H

**Hw:** Hardware

## I

**InCuSoft:** La Industria Cubana del Software

## K

**Knowlegde Area:** área de conocimiento

**KLDC:** Miles de líneas de código

**KPA´s :** Áreas de Proceso Principales

## L

**LDC:** Líneas de código

**Lotus:** compañía de software Lotus fundada por Mitch Kapor y después adquirida por IBM

**LDCF:** Líneas de código fuente

## M

**Microsoft:** Es una empresa multinacional estadounidense dedicada al sector de la informática

## N

**Novell:** Es una compañía de origen estadounidense dedicada al software, específicamente en el área de sistemas operativos de redes

## P

**PMI:** El Project Management Institute

**PMBOK:** Project Management Body of Knowledge. Es un estándar en la gestión de proyectos desarrollado por el Project Management Institute (PMI)

**PSP:** Proceso de Software Personal

**PROBE:** PROxy Based Estimating por sus siglas en inglés, traducido al español se entiende como estimación basada en la evaluación

**Plan de mitigación de riesgos:** Se desarrolla con el fin de proporcionar una medida de los posibles riegos y la forma de evitarlos

**Pascal:** Lenguaje de programación

**Prolog:** Es un lenguaje de programación lógico e interpretado, bastante popular en el medio de investigación en Inteligencia Artificial

## S

**SEI:** Instituto de Ingeniería del Software

**SIZE:** Tamaño. Se refiere en la entrada que necesita COCOMO para estimar el esfuerzo de un equipo de desarrollo

**Storyboard:** Guión

## T

**Testbeds**

**TDEV:** Tiempo de desarrollo del proyecto

**Transacción:** Es un evento que genera o modifica los datos que se encuentran eventualmente almacenados en un sistema de información

**TP:** Teleproceso

**TCF:** Factor de Complejidad Técnica

## U

**UCI:** Universidad de las Ciencias Informáticas

**USC-CSE:** University of Southern California- Center for Software Engineering

**UFP:** Puntos Función no Ajustados

## W

**Webmo:** Es un modelo de estimación que se centra en la web en su aspecto estático y multimedia

# ANEXOS

**ANEXO 1**

**ENTREVISTA**

**1- Medir conocimiento y aplicación de estimación de SW.**

a. Han realizado estimaciones en el proyecto? Las han tenido en cuenta para hacer la planificación?
b. Qué método(s) utilizan para estimar?
c. En qué etapas de desarrollo del software realizan las estimaciones?
d. Quiénes o quién las realizan?
e. Considera que puede valorarse la estimación como un elemento importante antes de hacer la planificación?

**2- Detallar impactos de PSP (experiencia).**

a. Qué tiempo lleva desarrollando este tipo de aplicaciones? Se considera un experto?
b. Cuándo considera que una persona es experta en aplicaciones de multimedia? Qué tiempo puede demorar alcanzar esa categoría?
c. Qué valoración merece la experiencia adquirida en el desarrollo de una aplicación de multimedia?
d. Conoce las características del PSP? Cree usted que es importante para obtener software de calidad?
e. Se ha tenido en cuenta las practicas de PSP en el equipo de desarrollo? En caso afirmativo de detalles de su aplicación.

**3- Detallar Teoría de Boehm.**

a. Evaluar influencia de los factores de escala. Agregaría otros?
b. En que etapa de desarrollo considera que se requiere de un mayor esfuerzo?

**4- Líneas de Código y Reutilización de Código.**

a. Qué es una línea de código (LDC)?

Las variantes a nivel de programas son:

___ Contar solo las líneas de código ejecutable.

___ Líneas de código más definiciones de datos.

___ Líneas de código, definiciones de datos y comentarios.

___ Líneas de código, definiciones de datos, comentarios y lenguaje de control (Job Control Languaje JCL).

___ Líneas de código y líneas físicas visualizadas en una pantalla.

___ Líneas de código determinadas por delimitadores lógicos.

Las variantes a nivel de proyecto son:

___ Contar solo las líneas nuevas.

___ Contar líneas nuevas y líneas modificadas.

___ Contar líneas nuevas, líneas modificadas y líneas reutilizadas.

b. Reutiliza líneas de código? En qué porciento reutiliza o ha reutilizado líneas de código en esta u otras aplicaciones de multimedia anteriores?
c. Si fuera a construir un concepto de líneas de código en la UCI qué no dejaría de tener en cuenta?

**ANEXO 2**

## ENCUESTA

**Aspectos Generales**

1. Conoce usted el modelo COCOMO?

   Sí ___ No ___

2. Conoce usted el Proceso Personal de Software (PSP) ?
   Sí ___ No ___

3. Evalúe su conocimiento acerca del modelo COCOMO ?
   ___ Muy escaso

   ___ Escaso

   ___ Medio

   ___ Elevado

   ___ Muy Elevado.

4. Evalúe su conocimiento acerca de PSP?
   ___ Muy escaso

   ___ Escaso

   ___ Normal

   ___ Elevado

   ___ Muy elevado

5. Conoce el método de PROBE?
   Sí ___ No ___

**Primer Punto**

6. Considera usted que el tamaño del software es un elemento importante para realizar estimar esfuerzo y tiempo?
   Sí ___ No ___

7. De ambos métodos cuál prefiere para estimar tamaño?
   Puntos de Función ___ PROBE ___

   Por qué?
   _____
   _____
   _____
   _____

8. Considera usted que se puede estimar el tamaño del software por el método PROBE y usar este valor en COCOMO, sustituyendo Puntos de Función?
   Sí ___ No ___

9. Considera usted que se puede estimar el tamaño del software por el método Puntos de Función y usar este valor en PSP, sustituyendo el paso 2 del método PROBE?
   Sí ___ No ___

10. Considera que la integración propuesta respecto a Líneas de Código sea válida para estimaciones de Proyectos de Software.
    Sí ___ No ___

**Segundo Punto**

11. Considera usted la reutilización un elemento importante a tener en cuenta a la hora de estimar?
    Sí ___ No ___

12. Según su criterio cuál de ambos métodos utilizaría para estimar el tamaño del código reutilizado?
    COCOMO____ PROBE ____

13. Considera usted que se puede estimar el tamaño del código reutilizado aplicando el método PROBE y usar este valor en COCOMO, sustituyendo la forma en que este lo hace?
    Sí ___ No ___

14. Considera usted que pudiera utilizarse el valor obtenido de estimar el tamaño del código reutilizado por COCOMO para ser usado en PROBE, sustituyendo la forma en que este lo hace?
    Sí ___ No ___

15. Considera usted que la integración propuesta para estimar tamaño del código reutilizado sea válida?
    Sí ___ No ___

**Tercer Punto**

16. En una escala de 0 a 5 evalúe según considere como cada una de las siguientes áreas de procesos pueden influir en la productividad y el esfuerzo de las organizaciones.

| | |
|---|---|
| Planeación de Proyectos | ___ |
| Seguimiento y control de proyectos | ___ |
| Foco del proceso de software | ___ |
| Definición del proceso de software | ___ |
| Manejo integrado del software | ___ |
| Ingeniería del producto de software | ___ |
| Revisión entre colegas | ___ |
| Administración cuantitativa del proyecto | ___ |
| Control Calidad | ___ |
| Prevención de defectos | ___ |
| Administración del cambio tecnológico | ___ |
| Administración de los cambios del proceso | ___ |

17. Considera usted que la estimación del factor de escala PMAT cuando se aplica PSP es con respecto a cuando no se aplica PSP:
    ___ Menos exacta

    ___ Igual

    ___ Más exacta

**Cuarta Parte**

18. Considera usted que la experiencia es un elemento importante en la realización de proyectos de software? Exprese su criterio en una escala de 0 a 5 ___.

19. Considera usted que al aplicar las prácticas de PSP a cada elemento del equipo de desarrollo se aprovecha mejor la experiencia y habilidades de cada individuo?
    Sí ___ No ___

20. Considera usted que integrar el trabajo de PSP a los equipos de desarrollo mejore la calidad y el proceso de las estimaciones.
    Sí ___ No ___

**General**

Después de analizar la propuesta de integración considera usted que la aplicación de la misma puede mejorar las estimaciones de duración de tiempo en Proyectos de desarrollo de Software. Mencione detalles que crea importantes tener en cuenta

Sí ___ No ___

_____
_____
_____
_____
_____