UCI

Universidad de las Ciencias Informáticas

TESIS EN OPCIÓN AL TÍTULO ACADÉMICO DE
MASTER EN GESTIÓN DE PROYECTOS INFORMÁTICOS

Título: Elementos a considerar en la integración de los métodos de Boehm y Humphrey para la estimación en la duración de un proyecto de software.

Autor: Lic. Yadira Ruíz Constanten

Tutor: Dra. Anaisa Hernández González

Ciudad de La Habana, junio de 2007.

"Año 49 de la Revolución"

Resumen

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. A ello se suma el problema de la escasa información proporcionada por las técnicas de estimación existentes para su aplicación a ciclos de vida de desarrollo de software diferentes al de cascada, como, por ejemplo, los ciclos de vida iterativo-incrementales o en espiral.

En este documento se da una visión general del proceso de estimación del software. Se indican algunos fundamentos y se le ubica dentro de la gestión de proyectos software. Se divide la estimación, en predicción del tamaño, del esfuerzo y del tiempo empleado para realizar el proyecto. Se hace especial hincapié, en los modelos propuestos por Barry Boehm y Watts Humphrey para estimar la duración de proyectos software tratando de buscar puntos en común para una posible integración.

Palabras claves

estimación, estimación de tamaño, estimación de costos, estimación de tiempo, medición, Gestión de Proyectos, planificación

Abstract

Software metrics is achieving a great importance because it is more obvious the necessity of getting objective data to evaluate, predict and improve not only the quality of software but also its time and cost. The value of metrics grows when it is done based on models built in the early phases of a project, due to that with the obtained results can control it all time and can fix on time all possible deviations.

The current proliferation of metrics and the huge volume of data make more evident that classic techniques of data analysis are insufficient to achieve the pursued objectives. Added to this situation is the problem of the lack of information provided by the existing estimation techniques in the life-cycle of the software development different to the cascade type.

In this paper, it is given a global vision of the process of software estimation. Some basic principles are indicated and it is put within the project software management. The estimation is divided into predicting the size, effort and time used to fulfill the project. It is stressed the models proposed by Barry Boehm and Watts Humphrey to estimate the duration of project software trying to find common issues for a possible integration.

KEYWORDS

Estimation, size estimation, cost estimation, time estimation, metrics, Project Managements, planning.

Índice

ÍNDICE DE FIGURAS

Introducción

Los seres humanos transforman la realidad que los rodea con sus manos e ingenio. Desde la antigüedad se ha visto que para ser más productivos es necesario organizarse ante los objetivos que se pretenden alcanzar. He ahí que se han desarrollado teorías relacionadas con la obtención de metodologías para obtener productos específicos, este es el caso del desarrollo de aplicaciones de software con calidad.

El trabajo a través de proyectos es la forma habitual de actuación en el desarrollo de software. La decisión de emprender un proyecto puede ser consecuencia de distintas circunstancias y situaciones en cada organización. Sin embargo, cada proyecto depende de una gestión apropiada, adaptada a sus características, para alcanzar sus objetivos. Puede decirse que un proyecto es un conjunto de etapas, actividades y tareas para alcanzar un objetivo que implica un trabajo no inmediato, a un plazo relativamente largo.

Los investigadores han determinado que los usuarios se sienten cómodos cuando están seguros de que podrán usar la aplicación propuesta en un tiempo razonable; en este sentido, ellos se convierten en promotores del proyecto. Existen dos aspectos importantes para hacer que los usuarios se sientan cómodos: la planificación y los costos de los proyectos.

Todo proyecto de ingeniería de software debe partir con un buen plan, pero lamentablemente, la planificación no es una tarea trivial. Uno de los aspectos que dificultan la labor de administradores y jefes de proyecto en torno a la planificación es el difícil empeño de realizar una estimación de costos y plazos realista.

El proceso de gestión del proyecto de software comienza con un conjunto de actividades que, globalmente se denominan planificación del proyecto. La primera de estas actividades es la estimación de costes y tiempos. Al ser la estimación la base de la planificación, hay que prestarle especial atención. Como diría Suntzu en "El arte de la guerra", siglo IV a.c.: "... si las estimaciones efectuadas antes de las hostilidades indican victoria, es porque los cálculos muestran que la fortaleza propia es superior a la

del enemigo; si indican derrota, es porque los cálculos muestran que es inferior. Con muchos cálculos se puede ganar; con pocos, no. ¡Cuántas menos posibilidades de victoria tiene quien no hace ninguno!" [1]

Le medición de software se presenta en nuestros días como un medio esencial para realizar las estimaciones oportunas del esfuerzo, tiempo y coste necesarios para el desarrollo de productos software. Además, tiene el objetivo de permitir llevar a cabo diversos estudios relativos a la productividad y calidad de software. A lo largo de los últimos veinte años, se han desarrollado varios métodos de medición: los puntos de función, puntos de función extendidos, features points, líneas de código, etc.

Aunque no es una ciencia exacta no podemos prescindir de ella puesto que hoy en dia un error en las predicciones puede conducir a resultados adversos.

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

- Durante la etapa de planeamiento: Permite decidir cuantas personas son necesarias para llevar a cabo el proyecto y establecer el cronograma adecuado.
- Para controlar el progreso del proyecto: Es esencial 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.

Al estimar debemos tratar de ser fieles y exactos, pues las precisiones en las estimaciones son importantes para:

- realizar análisis de costo-beneficios y financieros,
- realizar análisis de inversión de Hardware y Software,
- proveer la base para la evaluación gerencial de múltiples proyectos,
- servir de fundamento para los cronogramas, asignación de personal, gerencia de proyectos y definición de estructura, y
- evitar problemas como la renegociación de contratos, sobretiempos, incrementos de los costos de los usuarios o de los costos de los proyectos.

Es importante reconocer la fuerte relación entre costo, cronograma y calidad. Estos tres aspectos están íntimamente relacionados y confrontados entre sí. De esta manera, se hace difícil incrementar la calidad sin aumentar el costo y/o el cronograma del software a desarrollar.

Similarmente, el cronograma de desarrollo no puede reducirse dramáticamente sin deteriorar la calidad del producto de software y/o incrementar el costo de desarrollo. Los modelos de estimación juegan un papel importante ya que permiten equilibrar estos tres factores.

El proceso de estimación no es un proceso único, sino que es una tarea a repetir según se vaya refinando la información y por tanto se irá aproximando a la evolución real del proyecto.

Hay cuatro factores que influyen significativamente en las estimaciones:

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

De manera general, los métodos de estimación han sido diseñados para medir un determinado tipo de software. Por tanto, la aplicación de cada método depende particularmente del dominio software o del tipo de desarrollo. Existen por un lado, los sistemas tiempo real, científicos, ingeniería, etc. donde no sólo se deben tener en cuenta los parámetros convencionales del software sino también parámetros adicionales propios a cada tipo. Por otro lado, existen los sistemas basados en un análisis y diseño orientados a objetos, cuya medición depende mucho de las características de ese tipo de software así como en su caso, de la utilización de la notación UML en todas las etapas de desarrollo.

Siempre se han expuesto las limitaciones de los métodos de medición de tamaño funcional, debido a que no producían resultados satisfactorios y adecuados para software en tiempo real, orientados a objetos o científicos. El hecho de que estos

métodos constituyen una solución ideal, no quiere decir que su utilización esté libre de problemas. Esto es debido a los límites y la diversidad del software cuyo tamaño se desea medir. Algunos de los problemas relativos más importantes para la estimación de tamaño son:

- Objetividad y fiabilidad.
- Aplicación al dominio de software tiempo real y de nuevas tecnologías.
- Utilización del factor de ajuste.
- Precisión.
- Medición en distintas fases de desarrollo.

Para remediar esas limitaciones y mejorar el proceso de medición de esos tipos de software, se han desarrollado diversas extensiones así como nuevos métodos.

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. Muchas veces se cumple el plazo y no se ha podido entregar el producto como se había prometido. Repetidas ocurrencias de este fenómeno pueden ir mellando el prestigio del especialista y el de la institución en la que trabaja.

Se han expuesto varios de los factores que influyen en que esto ocurra, la mayoría de ellos relacionados con los problemas existentes para estimar la duración del desarrollo de un proyecto de software por parte del equipo de trabajo.

En la Universidad de las Ciencias Informáticas por ejemplo, es una generalidad el hecho de que se estime la duración de un proyecto en una única oportunidad, y no solo esto, sino también empleando un solo método, imposibilitándose el poder establecer comparaciones en la búsqueda de alguna concordancia razonable para asegurar que las estimaciones realizadas son confiables. Por otra parte, 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, 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 de uso considerados como críticos. En este caso 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 quienes determinan la cantidad de transacciones existentes; teniendo en cuenta además el hecho de que debamos incluir o no algún caso de uso que en el primer instante no se consideraba como crítico y después se decide modificar su clasificación.

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, a pesar de lo joven que es la institución y que quizás aun no pueda decirse que sea basta su experiencia en la realización de aplicaciones, existen algunas áreas de desarrollo que tienen un trabajo más extenso en este sentido y ni en esos lugares 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.

Tras la situación problémica expuesta surge un **problema**: ¿Cuáles son los puntos de contacto entre las teorías planteadas por Boehm y Humphrey al estimar la duración de un proyecto de software?

Definiéndose en la investigación como **Objeto de estudio** la estimación de la duración de un proyecto de software.

En aras de solucionar el problema planteado se tiene como **Objetivo General** definir, tras el estudio de los métodos de estimación de la duración de un proyecto de software, planteados por Barry Boehm y Watts Humphrey, los elementos que indiquen una posible integración de los mismos. Concretándose los objetivos específicos en:

1. Realizar un estudio significativo de la gestión de proyectos que permita señalar en la fase de planificación los modelos fundamentales para la estimación del tiempo de duración.

2. Realizar el estudio del área de conocimiento vinculada directamente con la estimación de la duración de un proyecto: gestión o administración de tiempo.
3. Realizar un estudio de los métodos de estimación de software.
4. Analizar las teorias de estimación planteadas por Boehm y Humphrey.
5. Proponer posible integración entre las teorías de Boehm y Humphrey en la duración de proyectos de software a través de elementos que lo evidencien.

Siendo la **hipótesis** de la investigación que, con el análisis de las teorías planteadas por Barry Boehm y Watts Humphrey puede lograrse una posible integración entre ellos para obtener mejores resultados al estimar la duración de los proyectos de software.

En general, la estimación en proyectos de software es una tarea extremadamente compleja, que requiere, entre otras cosas, disponer de información detallada del proyecto o de los proyectos a estimar, realizar una primera planificación del proyecto y conocer los recursos disponibles. Aún disponiendo de todos los medios y de la información necesaria, las estimaciones de los proyectos de software suelen errar, normalmente, pronosticando resultados menores de los que finalmente se producen. Pues en realidad es una tarea realmente difícil teniendo en cuenta que la estimación del coste y del esfuerzo del software nunca será una ciencia exacta. Son demasiadas las variables –humanas, técnicas, de entorno, políticas- que pueden afectar al coste final del software, al esfuerzo aplicado para desarrollarlo y por ende la duración del mismo. Sin embargo, la estimación del proyecto de software puede dejar de ser un oscuro arte para convertirse en una serie de pasos sistemáticos que proporcionen estimaciones con un grado de riesgo aceptable.

Este trabajo de tesis consta de tres capítulos. Estos quedaron estructurados de la siguiente manera:

Capítulo 1: Dominio del problema. La gestión y la estimación de proyecto. Los conceptos básicos de estimación; su proceso y los elementos que en ella intervienen.

Capítulo 2: Estudio de las técnicas de estimación. En qué desarrollos es conveniente su aplicación y cuales son sus fortalezas y debilidades. Al final se realiza una clasificación de los distintos tipos de software que son susceptibles de ser medidos utilizando las diferentes técnicas de estimación.

Capítulo 3: Se concluyen los análisis entre las dos teorías a analizar. Se exponen además las propuestas para integrarlas.

Capítulo 1. Estimación en Proyectos de Software

Introducción

En el presente capítulo haremos referencia a los conceptos que están estrechamente relacionados con el proceso de estimación de proyectos de software, comenzando desde los más generales como la definición de proyecto hasta llegar a las especificidades de estimar, pasando por la gestión de proyectos y la planificación de los mismos. Por otra parte se analizan también los puntos de vista del PMI a través de su guía, el PMBOK, haciendo énfasis en las áreas de procesos vinculadas con la planificación y la estimación de software.

1.1 Proyectos de Software

Muchas son las personas e instituciones en el mundo dedicadas al estudio del proceso de desarrollo de aplicaciones de software, explorando las diferentes formas de organizar el trabajo de manera tal que se pueda contar con aquellos cuyos objetivos no se consideren repetitivos, pues el producto se realizará una sola vez y para obtenerlo se necesitará la realización de una serie de tareas específicas que quizás nunca antes se hayan realizado y no se vuelvan a realizar después. Esta manera de emprender el trabajo podría ser denominado proyecto, aunque varias son las definiciones dadas a este término en el área informática:

- Conjunto único de actividades necesarias para producir un resultado definido, en un rango de fechas determinado y con una asignación específica de recursos. [2]
- Forma de organizar el trabajo, que consiste en planificar el curso de las tareas que se realizarán, con el objetivo de obtener un bien o servicio determinado, y controlar el seguimiento de esta planificación, para evitar las desviaciones. Aun en el caso de haber desviaciones se deberá adaptar el plan de modo que se alcancen los objetivos propuestos.[3]
- Por su parte el Project Management Institute (PMI) plantea que "un proyecto es un esfuerzo temporal acometido para crear un único servicio o producto.

> Temporal quiere decir que todo proyecto tiene un comienzo claro y un final claro. Único significa que el producto o servicio es diferente de alguna forma clara también de todos los productos o servicios similares." [4]

De manera general todos coinciden en que el concepto fundamental de lo que es un proyecto se centra en el producir o alcanzar un bien u objetivo.

Además se caracterizan por:

- Existe un objetivo claro que se tiene que alcanzar en plazo de tiempo limitado.
- Se utilizarán recursos de diversos tipos que en su mayoría son limitados.
- Tiene una fecha de inicio y otra de final.
- Se requiere una planificación.
- El producto final tendrá que cumplir las especificaciones.
- Se desea un determinado nivel de calidad en el producto.

1.1.1 Gestión de Proyectos

¿Que es la gestión? Podemos comenzar por las definiciones que da la Real Academia de la Lengua Española:

Gestión: Acción y efecto de gestionar. ...

Gestionar: Hacer diligencias conducentes al logro de un negocio o de un deseo cualquiera.

Estas definiciones nos dan el significado de las palabras, pero cuando entramos en la de gestión de empresas, nos encontramos todo un mundo de autores y definiciones que dependen de lo que se vaya a gestionar. Aunque todas ellas coinciden en que las funciones de la gestión que son: planificar, organizar, controlar y dirigir.

La gestión de proyectos comenzó su andadura en los años 60. Momentos en los que le se consideraba como una manera de planificar actividades, que se aplicaba exclusivamente a proyectos grandes, era algo reservado a los ingenieros y se medía el éxito únicamente en el desarrollo tecnológico que llevaba el proyecto en sí. [5]

En los primeros años habían personas que solo veían problemas a la gestión de proyectos, ya que lo consideraban una forma de trabajar que creaba puestos de trabajo innecesarios y que el beneficio financiero de los proyectos disminuía. Otro concepto que se tenía entonces era que al cliente se le suministraban únicamente productos. Pero afortunadamente, en la era actual la gestión de proyectos ha ido demostrando que todos estos conceptos eran erróneos y por el contrario hoy en día hay métricas y evidencias que demuestran que la gestión de los proyectos incrementa los beneficios, es aplicable a todo tipo de proyectos, ayuda a la resolución de problemas y se ha cambiado el concepto: ya que al cliente se le provee de servicios, no solo de productos. Se tiene entonces que los 3 pilares fundamentales de la gestión de proyectos son: hacer un proyecto dentro del plazo establecido, dentro del presupuesto previsto y que el producto final esté de acuerdo con las especificaciones del cliente.

La gestión de proyectos consiste en la aplicación de conocimientos, habilidades, herramientas y técnicas de un proyecto para satisfacer sus requisitos y alcanzar sus objetivos. Para lo cual hay que identificar los requisitos, establecer objetivos claros y posibles, equilibrar las demandas concurrentes de calidad, alcance, tiempo y costos y adaptar las especificaciones, los planes y el enfoque a las diversas inquietudes y expectativas.

Para conseguir que un proyecto software se lleve a cabo con éxito se debe comprender el ámbito del trabajo a realizar, los riesgos en los que se puede incurrir, las tareas que se han de llevar a cabo, las etapas que se recorren, el coste del proyecto, y el plan a seguir. Este conocimiento 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.

1.1.2 Planificación de Proyectos de Software

La gestión de un proyecto de software da inicio con un conjunto de actividades denominadas planificación del proyecto, en las que se realizan estimaciones del trabajo a realizar, de los recursos necesarios y del tiempo que transcurrirá desde el comienzo

hasta el final de su realización. La planificación tiene como objetivos proporcionar un marco de trabajo que permita al gestor hacer estimaciones razonables de recursos, coste 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. Este objetivo se logra mediante un proceso de descubrimiento de la información que lleve a estimaciones razonables. [6]

Para lograr una buena planificación del proyecto se debe:

1. Conceptualizar el producto y estimar el tamaño
2. Realizar estudio de factibilidad (evaluar viabilidad)
3. Estimar los recursos:
    - Recursos humanos: habilidades, posición organizacional, especialidad, se determina el número de personas después de estimar el tamaño
    - Recursos de software reutilizable: componentes ya desarrollados, componentes experimentados, componentes de experiencia parcial y componentes nuevos
    - Recursos del ambiente: herramientas de software y hardware

1.2 PMBOK. Sus procesos y áreas de conocimientos. La planificación y la estimación de software.

Las organizaciones más conocidas por la investigación, difusión y creación de comunidades profesionales para la gestión de proyectos son: Project Management Institute (PMI), Internacional Project Management Association (IPMA) fundada en 1965 y Projects in Controlled Environments (Prince2), que comenzó a trabajar en 1989.

IPMA y PMI surgieron como organizaciones profesionales para el desarrollo de conocimientos, metodologías y procesos para la gestión de proyectos. Prince2 ha tenido una evolución inversa. Comenzó siendo una metodología, alrededor de la cual se ha terminado creando una organización. Por otra parte IPMA y PMI tuvieron desde un principio como finalidad el desarrollo de un conocimiento de gestión válido para cualquier proyecto, mientras que Prince2 comenzó siendo un modelo específico para

sistemas de referencia para proyectos concretos de Tecnología de la Información, y a partir de una revisión hecha en 1996 se decidió ampliar su ámbito de validez, para cualquier tipo de proyecto.

De ellas es el Project Management Institute (PMI), la principal organización mundial dedicada a la Gestión de proyectos, desde su fundación en 1969, ha crecido hasta convertirse en la mayor organización de su tipo sin fines de lucro. Esta institución tiene un libro que se ha erigido como buque insignia de su labor formativa y divulgadora, el Project Management Body Knowledge (PMBOK). Este libro contiene y describe los conocimientos y habilidades necesarias dentro de la gestión de proyectos. Es el más reconocido en el mundo sobre Gestión de Proyectos y está aprobado como estándar por el por el American Nacional Standard Institute (ANSI).

Teniendo como apoyo los estudios realizados por el PMI, se hará un análisis de las teorías más consultadas para la estimación de la duración de un proyecto de software, en aras de integrar sus resultados.

1.2.1 PMBOK. Sus áreas de conocimiento.

El PMBOK comprende un conjunto de conocimientos de prácticas tradicionales aceptadas dentro de la profesión. Su objetivo es proporcionar referencias básicas acerca de la gestión de proyectos, ya que, según PMBOK Guide, la dirección de proyectos es relativamente joven y existe discrepancia entre la terminología utilizada.

PMBOK describe la dirección de proyecto relacionando los conocimientos de la administración de proyectos con procesos de la "gestión general" como la planificación, la organización y el control de las operaciones, modificándolas en muchos casos. La gestión de proyectos abarca un campo más amplio que el proyecto en sí. También plantea que la dirección de proyectos es un esfuerzo integrador, haciendo referencia a que los procesos y las áreas que la componen actúan como un sistema, donde las acciones o la falta de ellas en un área específica repercute en las demás.

La gestión de proyectos puede ser analizada desde dos puntos de vista:

- sus procesos

- sus áreas de conocimientos

Los procesos están agrupados por etapas del Ciclo de Vida de un Proyecto:

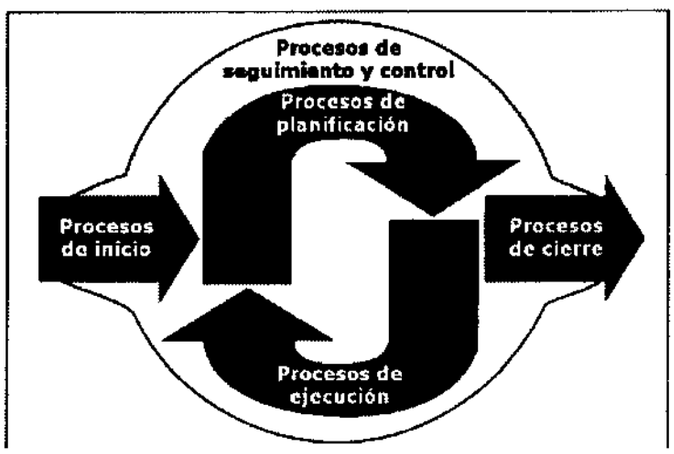


Figura 1: Grupos de procesos de la gestión de proyectos PMBOK 2004

- Inicio: lanzamiento formal del proyecto
- Planificación: definición del contexto y la organización necesarios para seleccionar la mejor solución para alcanzar los objetivos del proyecto y satisfacer al cliente.
- Ejecución: coordinación con las personas y gestión de los recursos necesarios para desarrollar todo el trabajo definido en el plan.
- Control: medición y análisis regular y frecuente del avance del proyecto para identificar variaciones con respecto al plan e implementar acciones correctivas, si fuese necesario.
- Cierre: aceptación formal de los productos y/o servicios generados como resultado del proyecto, por parte del cliente.

Todos los subprocesos que forman parte de los procesos mencionados anteriormente fueron agrupados en 9 áreas de conocimientos: [7]

1. Gestión de la Integración del proyecto: Incluye o agrupa los procesos requeridos para coordinar adecuadamente todos los elementos del proyecto. Los procesos principales de esa área son: el desarrollo del plan del proyecto, ejecución del

plan de proyecto y el control integrado de cambios.Los procesos de cada área interactúan con los procesos de las otras áreas, la integración resulta importante para relacionar los distintos procesos durante todo el ciclo de vida del proyecto, ya que estos procesos no son exclusivos de una sola área.

2. Gestión del Alcance del Proyecto: describe los procesos requeridos que definirán qué hacer y qué no hacer para tener éxito en este. La gestión del alcance tiene varios procesos: la iniciación, la planificación del alcance, definición del alcance, verificación del alcance y control de cambios del alcance.

   El alcance puede definirse como los requerimientos y características de los productos y el trabajo necesario para la realización del proyecto, determinando con precisión los resultados y sus limitaciones.

3. Gestión de Tiempos del Proyecto: describe los procesos requeridos para que el proyecto se complete a tiempo. Consta de los siguientes procesos:

   3.1 Definición de las actividades: se basa en la identificación de la documentación de las actividades;

   3.2 Secuenciamiento de las actividades: incluye la identificación de de actividades predecesoras o relaciones entre estas;

   3.3 Cálculo o estimación de la duración de las actividades: son las principales entradas al cronograma y, generalmente, esta estimación está dada por personas conocedoras de la actividad que puedan determinar el número de jornadas requeridas y el tiempo estimado

   3.4 Desarrollo del cronograma: significa la determinación de la fecha de inicio y culminación de las actividades de forma realista;

   3.5 Control del cronograma pretende lograr que los cambios del proyecto sean acordados y determinados.

4 Gestión de Costos del Proyecto: esta área es la encargada de garantizar que en el proyecto se realice sin sobrepasar el presupuesto estimado y aprobado del mismo.

Esta área también se encarga de analizar el impacto de las decisiones sobre los costos de los proyectos.

5 Gestión de la Calidad del Proyecto: área de gran importancia pues describe los procesos requeridos para asegurar que el proyecto va a satisfacer las necesidades por las cuales ha sido creado, además de lo establecido en las políticas de calidad. Contiene los procesos de Planificación de Calidad, Aseguramiento de la Calidad y Control de la Calidad.

6 Gestión de los Recursos Humanos del Proyecto: se define como un conjunto de procesos necesarios para mejorar la efectividad de las personas que van a estar involucradas en el proyecto, algunos de los temas que se tratan en esta dirección son: el liderazgo, la delegación, desarrollo de equipos y la evaluación del desempeño.

7 Gestión de las Comunicaciones del Proyecto: esta área incluye los procesos requeridos para asegurar la generación, la recolección la distribución, el almacenamiento y destino final de la información del proyecto para que se realice en tiempo y forma. Las comunicaciones afectan a todo el proyecto y a la organización, es por esto que todos los involucrados en él deben estar preparados para enviar y recibir comunicaciones.

8 Gestión de Riesgos del Proyecto: esta área se encarga de identificar, analizar y responder a los riesgos, permitiendo así que se pueda maximizar la probabilidad y el impacto de eventos positivos y minimizar la probabilidad y el impacto de sucesos adversos a los objetivos del proyecto.

9 Gestión de las Adquisiciones del Proyecto: describe los procesos requeridos para adquirir bienes y servicios (productos) requeridos para la realización del trabajo del proyecto.

El PMBOK ha tenido varias ediciones desde su fundación. Cada una de ellas se centra en asimilar, considerar y consolidar las recomendaciones que se les hacen, enfocándose siempre varios elementos claves:

- Mantener el tamaño actual del documento;

- Expandir el foco y la información sobre los grupos de procesos;
- Expandir el tratamiento de los procesos de integración;
- Expandir el tratamiento de los procesos de inicio;
- Mejorar la consistencia del contenido; y
- Expandir y mejorar el glosario.

Entre la edición del 2000 del PMBOK y la salida en el 2004 hubo diferencias en los procesos o actividades que forman el área de conocimientos que se pretende analizar.

Secciones de la Edición 2000	Secciones de la Edición 2004
6.1 Definición de Actividades	6.1 Definir las Actividades
6.2 Secuenciamiento de Actividades	6.2 Secuenciar las Actividades
	6.3 Estimar los Recursos de las Actividades
6.3 Estimación de Duración de las actividades	6.4 Estimar la Duración de las Actividades
6.4 Desarrollo del Cronograma	6.5 Desarrollar el Cronograma
6.5 Control del Cronograma	6.6 Controlar el Cronograma

Figura 2: Cambios del PMBOK en la Gestión de tiempo de los proyectos

Se movió el proceso de Planificar los Recursos al Capítulo 6 y se renombró como Estimar los Recursos de las Actividades (6.3). La nueva introducción del capítulo describe la necesidad de un plan para la gerencia del cronograma, un componente subsidiario del plan de gerencia del proyecto. También se provee nueva información sobre los estimados de los costos de los proyectos, nivelación de recursos, y reportes de avance, reflejando sus influencias en el cronograma del proyecto. [8]

Algunas de las adiciones, eliminaciones o modificaciones de las figuras incluyen elementos que serán tratados más adelante:

- Eliminación del PERT;
- Mejoras en las figuras de PDM y ADM;

- Eliminación de las figuras no vigentes tales como Diseño del Diagrama de Red del Proyecto con las Fechas, diagrama de Gantt y gráfico de Hitos; y
- Se añadieron figuras para el listado de Hitos, cronograma resumido y cronograma detallado.

1.2.2 Gestión de Tiempo del Proyecto.

La Gestión de Tiempo del Proyecto es una de las áreas de conocimientos, relacionadas con la planeación al igual que la gestión del alcance y del costo. Cada una de ellas se divide en varios procesos los que tienen una serie de elementos de entrada a los que se le aplican técnicas y herramientas para obtener una salida, que servirá de entrada a su vez para el área de conocimiento que le sucede. Evidenciándose la estrecha relación entre cada una de las partes con el objetivo de lograr un producto con calidad. Se centrará la atención en el proceso que abarca todo lo relacionado con la Estimación de la duración de Actividades.

Por ello es sumamente importante analizar cada una de las actividades que están vinculadas a las áreas de conocimientos que influyen en la estimación, para tratar de definir los artefactos que marcan la pauta para su buena realización.

En esta área de conocimiento existen procesos que interactúan unos con otros y con los procesos de otras áreas de conocimiento también. Cada proceso puede involucrar el esfuerzo de uno o más individuos basado en las necesidades del proyecto pues cada proceso ocurre al menos una vez en cada fase del proyecto.

En algunos proyectos, especialmente los más pequeños, las secuencias de las actividades, la estimación de sus duraciones, y el desarrollo de la programación están tan estrechamente unidas que se ven como un sólo proceso. Se presentan como procesos distintos porque las herramientas y técnicas para cada una son diferentes tal como se evidencia en la Vista general de la Administración de tiempo de Proyecto dada en el PMBOK. [Ver Anexo 1].

1.2.2.1 Definición de actividades

La definición de actividades involucra el identificar y documentar las actividades específicas que tienen que ser ejecutadas de manera que se puedan producir las entregas y subentregas identificadas en la estructura de desglose de trabajo. Está implícita en este proceso la necesidad de definir las actividades de tal manera que los objetivos del proyecto se puedan cumplir. (Ver Anexo 2.1)

Entradas a la Definición de Actividades

1. Estructura de desglose de trabajo.
2. Declaración del alcance.
3. Información histórica.
4. Restricciones.
5. Suposiciones.

Herramientas y Técnicas para la Definición de las Actividades

1. Descomposición. Involucra subdividir los elementos del proyecto, en componentes más pequeños y manejables de manera que se pueda proveer un mejor control administrativo.
2. Patrones. Una lista de actividades, o una porción de una lista de actividades de un proyecto previo, se usa muchas veces como un patrón para un nuevo proyecto.

Salidas de la Definición de Actividades

1. Lista de actividades. Debe incluir todas las actividades que serán ejecutadas en el proyecto. Las descripciones de cada actividad para asegurar que los miembros del equipo del proyecto entenderán como se deberá de ejecutar el trabajo.
2. Detalle de soporte. Para la lista de actividades deberá ser documentado y organizado de manera que facilite su uso por otros procesos de la

administración del proyecto. Siempre incluirá documentación de todas las suposiciones y restricciones identificadas.

3. Actualizaciones a la estructura de desglose de trabajo. El equipo del proyecto puede identificar entregas, faltantes, o puede determinar que la descripción de la entrega puede necesitar clarificación o corrección. Estas actualizaciones se llaman muchas veces refinamientos y son muy probables cuando el proyecto involucra tecnología nueva o tecnología que no ha sido ensayada.

Al culminar este proceso podemos garantizar que el desarrollo de los proyectos será de forma más ordenada y controlada, así como que se utilicen de mejor manera los recursos asignados al proyecto tanto de capital humano como materiales. En un proyecto deben estar debidamente definidas las actividades, ya que nos permitirá organizar y definir el alcance total del proyecto orientando de la mejor manera en la entrega de los elementos, y poder cumplir con el objetivo del proyecto.

1.2.2.2 Secuencia de Actividades

Involucra identificar y documentar las dependencias entre actividades. Las actividades deben de ser secuenciadas de manera precisa de forma tal que soporten luego el desarrollo de una programación realista y alcanzable. (Ver Anexo 2.2)

Entradas a la Secuencia de Actividades

1. Lista de actividades.
2. Descripción del producto.
3. Dependencias mandatorias.
4. Dependencias discrecionales.
5. Dependencias externas.
6. Restricciones.
7. Suposiciones.

Técnicas y Herramientas para la Secuencia de Actividades.

1. Método de diagrama de precedencia (PDM). Es un método de construir una red de diagrama de proyecto usando nodos para representar las actividades y conectándolos con flechas que muestran las dependencias. Esta técnica también se llama actividad - sobre – nodo (activity - on - node, AON) y es el método usado por la mayoría de paquetes de software de administración de proyectos. incluye cuatro tipos de dependencias o de relaciones de precedencia:
    a. Fin a comienzo
    b. Fin a fin
    c. Comienzo a comienzo
    d. Comienzo a fin
2. Método de diagramación con flechas. (Arrow diagramming method ADM). Este es un método para construir diagramas de red usando flechas para representar las actividades y conectándolas con nodos para mostrar las dependencias. ADM utiliza únicamente dependencias fin a comienzo y puede requerir el uso de actividades ficticias para poder definir todas las relaciones lógicas de manera correcta.
3. Método de diagramación condicional. Las técnicas de diagramación tales como: GERT (técnica de evaluación y repaso gráfica (Graphical Evaluation and Review Techique)) y modelos de Sistemas Dinámicos permiten el uso de actividades no secuenciales tales como loops, ramales condicionales o probabilísticas que no son permitidas por PDM ni por ADM.
4. Patrones de red. Redes estandarizadas pueden ser usadas para acelerar la preparación de diagramas de red de proyectos.

Salidas de la Secuencia de Actividades

1. Diagrama de red del proyecto. Es una figura esquemática de las actividades del proyecto y sus relaciones lógicas. El diagrama deberá estar acompañado de una descripción que resuma y describa la lógica usada para las secuencias de las actividades. El diagrama de red de proyecto muchas veces se llama

incorrectamente diagrama PERT (técnica de evaluación y repaso de programa (Program Evaluation and Review Technique)). Muy poco usado hoy en día.

2. Actualización a la lista de actividades. La preparación de la red de diagrama de proyecto puede revelar instancias en las que una actividad deberá ser dividida o de otra manera redefinida de manera que se pueda diagramar la relación de lógica correctas.

1.2.2.3 Estimación de la Duración de las Actividades

Es esta la nueva actividad que incorpora el PMBOK en su última versión. Involucra estimar el número de períodos de trabajo que probablemente más se necesitará para completar cada actividad identificada. La persona o grupo del equipo del proyecto que esté más familiarizado con la naturaleza de una actividad específica deberá estimar o al menos aprobar la duración de la actividad.

La duración completa del proyecto también puede ser estimada usando herramientas y técnicas aquí presentadas, pero es calculada de manera apropiada como la salida del desarrollo de la programación. (Ver Anexo 2.3)

Entradas a la estimación de la Duración de las Actividades

1. Lista de actividades: Debe incluir todas las actividades que serán ejecutadas en el proyecto. La lista de actividades debe incluir descripciones de cada actividad para asegurar que los miembros del equipo del proyecto entenderán cómo se deberá de ejecutar el trabajo.
2. Restricciones: Son factores que van a limitar las opciones del equipo del proyecto.
3. Suposiciones: Son factores que, para los procesos de planeación, serán consideradas como verdaderas, reales, o ciertas. Estas generalmente involucran algún grado de riesgo y serán normalmente una salida del proceso de identificación de riesgos.
4. Requerimientos de recursos: La duración de la mayoría de las actividades se verá influenciada significativamente por los recursos asignada a ella.

5. Capacidades de recursos: La evaluación de la mayoría de las actividades se verá influenciada significativamente por las capacidades de los recursos humanos y materiales asignados a ella.
6. Información histórica: De la duración más probable de muchas categorías de actividades, está muchas veces disponible de una o de más de las siguientes fuentes:

- Archivos de proyecto.
- Bases de datos de estimación comerciales.
- Conocimiento del equipo de proyecto.

Herramientas y Técnicas para la Estimación de la Duración de las Actividades.

1. Opinión experta: Las duraciones son muchas veces difíciles de estimar porque hay un número de factores que las pueden influenciar. La opinión experta guiada por información histórica deberá ser usada cuando sea posible. Si tal experiencia no está disponible, los estimativos son inherentemente inciertos y riesgosos.
2. Estimación análoga: También llamada estimación de arriba hacia abajo, precisa usar duraciones reales de una actividad previa y similar como base para la estimación de la duración futura de una actividad. Es usada frecuentemente para estimar la duración de proyectos cuando hay una cantidad limitada de proyecto. La estimación análoga es una forma de opinión experta. Es más fiable cuando:
    a. la actividad previa es similar de hecho y no sólo en apariencia
    b. los individuos que preparan los estimativos tienen la experiencia necesaria.
3. Simulación: Involucra calcular múltiples duraciones con diferentes juegos de suposiciones.

Salidas de la Estimación de la Duración de las Actividades.

1. Estimación de la duración de la actividad. Son evaluaciones cuantitativas del número de períodos de trabajo más probable que se requerirá para completar una actividad. La estimación de la duración de las actividades siempre deberá incluir alguna indicación del rango de posibles resultados.
2. Bases de estimación. Las suposiciones hechas en el desarrollo de los estimativos deberán estar documentados.
3. Actualizaciones a la lista de actividades. La preparación de la red de diagrama de proyecto puede revelar instancias en las que una actividad deberá ser dividida o de otra manera redefinida de manera que se pueda diagramar las relaciones lógicas correctas.

Hay que tener en cuenta que la administración del tiempo del proyecto recae principalmente en que los requerimientos de recursos estén disponibles con anticipación. De allí en adelante, el equipo del proyecto, en base a su capacidad, conocimiento e información histórica se encarga de definir las actividades del proyecto, establecer su secuencia, y estimar su duración de cumplimiento, lo que ayudará a conocer previamente qué es lo que se tiene que hacer, en que orden y en que tiempo se debe cumplir, logrando así empezar y terminar con pie derecho lo que te propongas conseguir.

1.2.2.4 Desarrollo de la Programación

La tarea principal de esta etapa consiste en analizar las secuencias de las actividades, las duraciones de las actividades, y los requerimientos de recursos para establecer la programación del proyecto. (Ver Anexo 2.4)

Entradas al Desarrollo de la Programación

1. Diagrama de red del proyecto
2. Estimación de la duración de las actividades
3. Requerimientos de recursos
4. Descripción del administrador de recursos
5. Calendarios

6. Restricciones
7. Suposiciones
8. Holguras

Herramientas y Técnicas para el Desarrollo de la Programación

1. Análisis matemático: Requiere calcular las fechas teóricas tempranas y tardías para todas las actividades sin tener en cuenta cualquier limitación del pool de recursos disponibles. Las técnicas más comunes conocidas son:
    a. Método de la Ruta Crítica.
    b. Técnica de Evaluación y Revisión de Programas.
    c. Método de Evaluación y Revisión Gráfica.
2. Compresión de datos: Se busca el poder acortar la duración del proyecto.
3. Simulaciones: Juego de suposiciones.
4. Heurísticas de nivelación de recursos: El nivelar los recursos es limitar estos para optimizar.
5. Software de administración de proyectos: Producto que genera en forma automática el análisis matemático con presentación de informes y optimizar los diversos procesos de la programación.

Salidas al Desarrollo de la Programación

1. Programación del proyecto. Requiere fechas de inicio y de terminación planeadas para cada detalle de actividad. Suele presentarse generalmente de forma gráfica usando uno o más de los formatos presentados a continuación:
    - Diagramas de red de proyecto
    - Gráficas de barras, que también se conocen como diagramas de Gant.
    - Gráficas de hitos.
    - Diagramas de red de proyectos en escalas de tiempo

2. Detalle de soporte: Incluye al menos documentación de todas las restricciones y suposiciones identificadas.
3. Plan de manejo de la programación: Define como se manejarán los cambios a la programación.
4. Actualizaciones a los requerimientos de recursos. Las nivelaciones de recursos y actualizaciones a la lista de actividades pueden tener un efecto significativo sobre las estimaciones preliminares de los requerimientos de recursos.

Hasta ahora hemos visto como la metodología PMBOK nos permite desarrollar proyectos documentados de forma controlada y ordenada. También existe una relación entre los niveles de recursos y la duración del proyecto, así como el correcto uso de los mismos, en algunos casos existen herramientas de software sencillas para realizar la programación.

1.2.2.5 Control de la Programación

El control de la programación se preocupa por: (Ver Anexo 2.5)

(a) influenciar los factores que crean cambios en la programación para asegurar que tales cambios sean beneficiosos

(b) determinar que la programación ha sido cambiada

(c) administrar los cambios actuales cuándo y cómo ocurren.

Entradas al Control de la Programación

1. Cronograma del proyecto.
2. Reportes de desempeño
3. Requisiciones de cambio.
4. Plan de manejo de la programación.

Herramientas y Técnicas para el Control de la Programación

1. Sistema de control de cambios a la programación. Define los procedimientos por medio de los cuales la programación del proyecto puede ser cambiada. Este incluye el papeleo, el sistema de seguimiento y los niveles de aprobación

necesarios para autorizar tales cambios. El sistema de control a la programación deberá estar integrado de manera íntima con el sistema general de control de cambios.

2. Medición de desempeño. Una parte importante del control de la programación es decidir si la varianza de programación requiere acción correctiva.
3. Planeación adicional. Pocos proyectos se desarrollan exactamente de acuerdo a su plan; pueden requerir nuevas o revisadas duraciones de actividades, secuencias de actividades modificadas, o análisis de programaciones alternas.
4. Software de administración de proyectos. Es hacer un seguimiento de fechas programadas versus fechas reales y de pronosticar los efectos de los cambios de programación, reales o potenciales, hacen de esta herramienta un recurso útil para el control de la programación.

Salidas del Control de la Programación

1. Actualizaciones a la programación. Es cualquier cambio en la información que se usa para administrar el proyecto. Las revisiones son una categoría especial de actualizaciones de la programación, cambios a las fechas programadas de inicio y finalización en la programación de proyecto aprobada.
2. Acción correctiva. En el campo de la administración del tiempo muchas veces requiere expeditar: acción especial que se toma para asegurar la terminación de una actividad a tiempo o con el menor retraso posible.
3. Lecciones aprendidas. Las causas de varianza, el razonamiento detrás de las acciones correctivas escogidas, y otros tipos de lecciones aprendidas del control de la programación, deberán ser documentadas para que puedan ser parte de las bases de datos históricas.

Todo trabajo estructurado permite ejecutar nuevas tareas con mayor factibilidad y facilidad, por sobre todo cuando consideramos que en las empresas nadie es eterno, esto ayudará incluso a que la documentación fortalezca a nuevas generaciones de

empleados, para que con su experiencia externa y la experiencia interna documentada ayuden a ser eficaces en las labores encomendadas.

1.3 Estimación de software

¿Qué es la estimación?

Estimar consiste en determinar el valor de una variable desconocida a partir de otras conocidas, o de una pequeña cantidad de valores conocidos de esa misma variable. Es importante pensar en una predicción como en un rango más que como un simple número. Una estimación o predicción no es un objetivo, sino una valoración probabilística. El valor que se obtiene de una estimación es el centro del rango. [9]

La estimación de proyectos acompaña a cualquier ingeniería y la informática no es una excepción. Generalmente en la actualidad cuando hablamos de estimación de un proyecto podemos hacer referencia tres grandes áreas: estimación de tiempo, de recursos y de costo, que aunque tratemos de verlas a cada una de ellas de manera individual, la relación entre ellas es muy estrecha. La estimación de recursos, costo y tiempo es un esfuerzo que requiere:

- experiencia,
- acceso a buena información histórica (métricas)
- valor para comprometerse con predicciones cuantitativas cuando la información cualitativa es todo lo que existe.

La estimación cuantitativa implica cierta incertidumbre, pues se da a través de información histórica no exacta y variabilidad en los requisitos. Por otra parte los nuevos enfoques de ingeniería asumen una visión iterativa que permite reexaminar las estimaciones (conforme tenemos más información) y modificarlas cuando el cliente cambia los requisitos.

Luego de medir lo que quiere el usuario se debe analizar lo que le costará a la institución desarrollar la aplicación. Para realizar este proceso hace falta experiencia

en valoraciones. Esta experiencia puede gestionarse de dos formas diferentes, individual y de empresa:

- La experiencia individual es la que aporta un individuo de la organización, tras acumular muchas experiencias en su mente.
- La experiencia de empresa se basa en la información que ésta ha ido acumulando en ficheros históricos sobre valoraciones realizadas y costes reales de desarrollos realizados.

Ésta última forma de experiencia es más deseable que la primera ya que permite un mayor cúmulo de información, más proyectos. También es menos dependiente de las personas, con lo que la institución será más estable a posibles pérdidas de personal. Además está más estructurada ya que se pueden recoger todas las medidas que los diferentes líderes de proyecto estimen necesarias, es decir, se podría recoger información sobre herramientas usadas, grado de experiencia al aplicarse, etc...

Después de identificar cada una de las fases del proyecto y conocidas las tareas a realizar se deberá programar (planificar), el proceso de desarrollo y de esta planificación obtendremos una estimación económica del coste.

Tras ubicar el proceso de estimación dentro de la gestión del proyecto, pasamos a ver cómo se realiza realmente la estimación. El proceso para crear una planificación de desarrollo exacta consta de tres pasos:

- Estimar el tamaño del producto (generalmente a través del número de líneas de código o puntos de función).
- Estimar el esfuerzo (personas-mes).
- Estimar la duración (meses).

Estos tres pasos se pueden englobar dentro de un paso más general:

- Dar un intervalo de estimación e ir refinando periódicamente ese intervalo, para ir aumentando la precisión a medida que se avanza el proyecto.

Las secciones siguientes describen con detalle cada uno de estos pasos.

1.3.1 Estimación del tamaño

El tamaño de un producto software es un indicador de la amplitud y profundidad del conjunto de prestaciones que incorpora, así como de la complejidad y dificultad del programa.

Hasta hace algunos años no se vislumbraban métodos estándar para estimar el tamaño de un proyecto de software. Actualmente se identifican los siguientes grupos de técnicas para estimar el tamaño de un producto software:

- Estimación algorítmica: se construye un modelo paramétrico basado en información histórica sobre los costes y, habitualmente, sobre el tamaño. Los modelos empleados pueden ser:
    - Empíricos: se construyen únicamente a partir de los datos históricos, mediante regresión
    - Teóricos: se derivan de hipótesis teóricas acerca del comportamiento de los proyectos
    - Simuladores: Están basados en modelos dinámicos. Permiten simular el comportamiento del proyecto a lo largo del tiempo.
- Estimación heurística: se incluyen aquí técnicas heurísticas como reglas de inducción, técnicas fuzzy (lógica difusa), redes neuronales, razonamiento basado en casos y, en los últimos años, los algoritmos de computación genética.
- Estimación por analogía.
- Juicio de expertos.

Roger Pressman en la última edición de su emblemático libro "Ingeniería de software, un enfoque práctico" plantea que para lograr estimaciones del tamaño del proyecto confiables hay varias opciones:

1. Retrasar la estimación para después
2. Basar la estimación en proyectos similares ya terminados

3. Emplear **técnicas de descomposición** para generar estimaciones de costo y esfuerzo
4. Utilizar uno o más **modelos empíricos** en la estimación de costo y esfuerzo.

La primera opción no es práctica, porque las estimaciones se deben dar por adelantado. La segunda opción funciona si el proyecto es muy similar a previos y se mantienen otras condiciones como equipo, cliente, condiciones de mercado, fechas límite, etc. Las **técnicas de descomposición** (enfoque divide y vencerás) dividen el proyecto en funciones principales y actividades de ingeniería relacionadas que permite estimar en forma escalonada, es decir, las estimaciones se hacen sobre cada componente en que se descompone el software o sobre tareas de bajo nivel en que se descomponen las tareas pues las estimaciones de bajo nivel se combinan para producir una estimación del proyecto completo, o sea, el coste total del proyecto es el resultado de sumar las estimaciones de todos los componentes en los que se ha dividido el proyecto. [6]

A continuación se muestra un análisis más exhaustivo de la última opción pues es en esta categoría donde clasifican los dos métodos más utilizados mundialmente para estimar el tamaño de un software.

1.3.1.1 Principales modelos empíricos de estimación de tamaño

La palabra empírico viene de el griego *empeirikos* = experimentado y por lo general se refiere a una decisión hecha según la experiencia en vez de teoría. Por lo que se puede afirmar que cualquier predicción se debe basar en un modelo empírico, que constituya su punto de partida al analizar datos para establecer un modelo numérico. Dichos modelos de estimación son útiles para complementar las técnicas de descomposición.

Los datos empíricos que soportan la mayoría de los modelos de estimación se obtienen de una muestra limitada de proyectos. Es por eso que estos modelos de estimación no son adecuados para toda clase de software y en todos los entornos de desarrollo. Por consiguiente, los resultados obtenidos de dichos modelos se deben utilizar con prudencia.

Hay dos métodos empíricos por excelencia a los que ha de hacerse referencia cuando se habla de la estimación del tamaño de un software: el enfoque encausado hacia el análisis de la cantidad de Líneas de Código (LCD) y el que está orientado a los Puntos de Función (PF).

Líneas de código

La métrica del software más frecuentemente adoptada en los inicios de esta centuria era la de las **líneas de código** (Low, Graham and Jeffery, Ross 1990). A continuación mostramos 11 variantes de definiciones analizadas de este concepto, separadas en dos grupos: a nivel de programa y a nivel de proyecto.

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.
- Contar todas las líneas del proyecto, más código temporal.
- Contar todas las líneas del proyecto, código temporal, y código de soporte.

Es importante tener en cuenta estas últimas cuando no es nuevo todo el código.

La definición de líneas de código que se adopta en cualquier organización particular depende de donde vaya a ser utilizado el software. Habiendo decidido que la variante a

nivel de proyecto se adoptará, el analista debe decidir también qué variante a nivel de programa se ha de tener en cuenta.

La estimación a posteriori del tamaño del sistema utilizando líneas de código, debería ser totalmente consistente, debido a que solo puede haber una respuesta, ya que la definición de líneas de código no cambia. Sin embargo una estimación a priori del esfuerzo necesario para el desarrollo del sistema, requiere una estimación del tamaño del sistema, que se ha de realizar en las etapas iniciales del ciclo de vida del proyecto: especificación de requerimientos o especificación de diseño (Figura 8). Esta estimación a priori del tamaño del sistema, está normalmente basada en la experiencia pasada, con proyectos similares, de la persona que está realizando la estimación.

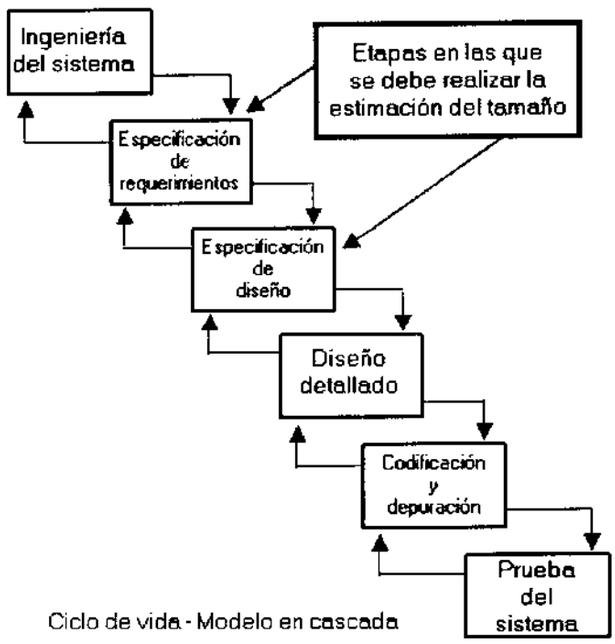


Figura 3: Estimación del tamaño a priori

Expresar el software en términos de líneas de código tiene varios problemas como se mostrará a continuación. El primero es saber realmente qué se entiende por línea de código pues son varios los criterios que aparecen documentados al carecer de una definición universal. Otra dificultad que tiene, es que como métrica para medir el tamaño del sistema depende del lenguaje de desarrollo. Consecuentemente no es

posible comparar directamente la productividad de proyectos realizados en diferentes lenguajes, usando líneas de código como medida del tamaño del sistema. Por otra parte se tiene que en la actualidad, las herramientas de desarrollo proveen la capacidad de disminuir substancialmente el esfuerzo de la codificación, pues la tendencia actual ya no es codificar sino generar código. Además se ha de tener en cuenta que el estimar en fases tempranas del desarrollo de software dificulta el saber la cantidad de líneas de código que tendrá la aplicación.

Existe otra métrica para estimar el tamaño de un software cuyo enfoque tiene características que permiten superar los principales problemas de utilizar el método de las líneas de código. Si se tiene en cuenta que los puntos de función como son una medida de la funcionalidad entonces deben ser independientes del lenguaje, herramientas o metodologías utilizadas en la implementación; que también los puntos de función pueden ser estimados a partir de la especificación de requisitos o especificaciones del diseño, haciendo posible de este modo la estimación del esfuerzo de desarrollo en etapas tempranas del mismo. Como los puntos de función están íntimamente relacionados con la declaración de requisitos, cualquier modificación a ésta, puede ser reflejada sin mayor dificultad en una re estimación; y que además los puntos de función están basados en una visión externa del usuario del sistema, se han de preguntar qué son los puntos de función que ventajas substanciales muestra por sobre las líneas de código, para fines de estimación temprana del tamaño del software.

Puntos de Función

A.J. Albrecht introdujo el concepto de puntos de función en 1979 para medir la funcionalidad proporcionada por el software, basándose en el diseño lógico del sistema. Este método constituye la técnica algorítmica de estimación de tamaño más conocida. Utiliza un modelo paramétrico (lista de parámetros) orientado hacia las aplicaciones de gestión que toma el enfoque de la historia organizacional para estimar.

Es una medida de la complejidad basada en las características globales de la aplicación.

Los objetivos de este método son:

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

Un punto de función es una medida sintética cuyo valor se determina por medio de los siguientes elementos:

- Entradas: Pantallas, formularios, cuadros de diálogo, controles o mensajes, a través de los cuales un usuario final o cualquier otro programa pueda añadir, borrar o cambiar datos de un programa.
- Salidas: Pantallas, informes, gráficos o mensajes que el programa genera para el usuario final o cualquier otro programa. Esto incluye cualquier salida que tenga formato diferente o requiera un procesamiento diferente a otros tipos de salida.
- Consultas: Una consulta está definida como una entrada interactiva que resulta de la generación de algún tipo de respuesta en forma de salida interactiva.
- Archivos lógicos internos: Los principales grupos lógicos de datos de usuarios finales o información de control que están completamente controlados por el programa. Un archivo lógico podría constar de un único archivo plano o de una sola tabla en una base de datos relacional.
- Archivos de interfaz externos: Archivos controlados por otros programas, con los que el programa va a interactuar. Esto incluye cada uno de los principales grupos de datos lógicos o información de control que entre o salga en el programa.

En la práctica el análisis mediante puntos funcionales resulta ser mucho más complicado que el mero recuento de los cinco tipos de atributos. Al aplicar este método

se han de introducir diversas ponderaciones que toman en cuenta la complejidad (reducida, normal o elevada) de cada elemento.

Los pasos a seguir son:

1) Calcular los Puntos Función sin ajustar:

1.1) Contar el número de funciones de usuario (basado en contar el número de elementos de los 5 tipos diferentes recién mencionados).

1.2) Determinar el nivel de complejidad (baja, media, alta) de cada función de usuario. Para ello se tienen en cuenta el número de tipos de elementos de datos y el número de tipos de archivos (o de elementos de tipo registro) referenciados.

1.3) Aplicar pesos de complejidad. Aplicar pesos según el nivel complejidad. La suma para todas las funciones de usuario equivale al número de PF sin ajustar.

2) Ajustar lo anterior para tener en cuenta la "complejidad del proceso". Para ello, se valora el grado de influencia de 14 factores diferentes.

Estos 14 factores se relacionan a continuación:

1. **Comunicaciones de datos**: Este factor es concerniente a la transmisión de datos o información de control, enviados o recibidos mediante algún sistema de comunicaciones.

2. **Procesamiento distribuido**: El procesamiento distribuido es concerniente a si una aplicación es monolítica y se ejecuta en un único procesador, o si la aplicación consiste en código independiente ejecutándose en procesadores distintos y persiguiendo un fin común.

3. **Objetivos de rendimiento**: Hace referencia a la relevancia del rendimiento de la aplicación.

4. **Configuración de uso intensivo**: Indica si el sistema se va a utilizar en un entorno operativo que está siendo utilizado de manera intensa.

5. **Tasas de transacción rápidas**: Tendrá una puntuación de 5 si el volumen de transacciones es suficientemente alto como para requerir un esfuerzo de desarrollo especial, para conseguir la productividad deseada.

6. **Entrada de datos en línea**: Este factor tendría una puntuación de 0 si son interactivas menos del 15 por ciento de las transacciones, y tendrá una puntuación de 5 si más del 50 por ciento de las transacciones son interactivas.

7. **Amigabilidad en el diseño**: Determina si las entradas de datos interactivas requieren que las transacciones de entrada se lleven a cabo sobre múltiples pantallas o variadas operaciones.

8. **Actualización de datos en línea**: Este factor tendrá puntuación máxima si las actualizaciones en línea son obligatorias y especialmente dificultosas, quizá debido a la necesidad de realizar copias de seguridad, o de proteger los datos contra cambios accidentales.

9. **Procesamiento complejo**: Este factor se puntuará con 5 si se requieren gran cantidad de decisiones lógicas, complicados procedimientos matemáticos o difícil manejo de excepciones.

10. **Reusabilidad**: Indica si gran parte de la funcionalidad del proyecto, está pensada para un uso intensivo por otras aplicaciones.

11. **Facilidad de instalación**: Un valor de 5 denota que la instalación del sistema es tan importante, que requiere un esfuerzo especial el desarrollar el software necesario para realizarla. Se guardan los datos.

13. **Multiplicidad de emplazamientos:** Una puntuación máxima indicaría que el sistema se ha diseñado para soportar múltiples instalaciones en diferentes organizaciones.

14. **Versatilidad**: Determina si la aplicación se ha realizado para facilitar los cambios y para ser utilizada por el usuario.

Los Puntos de Función fueron diseñados originariamente para ser aplicados a sistemas de información de gestión, es por ello que se puso énfasis en la dimensión de datos, excluyendo las dimensiones funcionales y de control.

En resumen, a partir de los 5 parámetros iniciales se calculan los **Puntos de Función sin ajustar**. A posteriori se aplica un **factor de ajuste** que resulta de valoraciones subjetivas efectuadas a la aplicación que está siendo medida y a su entorno. La suma de los puntos de función sin ajustar y el factor de ajuste representan los **Puntos de Función ajustados**.

Los métodos que utilizan los puntos de función para estimar el tamaño, no pueden interpretar la información que brindan a través de los puntos de función desajustados que es el resultado final que se obtiene. Problemática resuelta por Jones al tabular la correspondencia, según el lenguaje de programación utilizado, entre los Puntos de Función Ajustados y las Líneas de Código.

Si bien es cierto que los puntos de función de Albrecht, no son aplicables a todos los tipos de software, la mayoría de los métodos, sobre todo los más actuales, incorporan facilidades para distintas aplicaciones y se basan en la técnica de Albrecht, modificando ligeramente algunos aspectos o incorporando nuevos elementos a los mencionados. Entre ellas tenemos:

- Feature Points (puntos de características): Este método fue propuesto por Caper Jones como una alternativa que permitiera obtener puntos de función en software **científicos** y de **ingeniería**. Para evitar confusiones con los Puntos de Función, Jones lo denominó puntos de característica (en inglés feature points). Actualmente es usado con mucho éxito en software de sistemas embebidos, de control de procesos, empotrados y sistemas en tiempo real, pues añade a los elementos que se miden en los puntos de función de Albrecht, una medida de los algoritmos que se emplean, ya que estos pueden tener distintos niveles de complejidad, ampliando con ello, la utilización de esta métrica.

- MK II FPA: propuesto por Charles R. Symons, 1998. Este método es una derivación de los FP de Albrecht, quien considera al sistema que se está analizando, compuesto por cinco tipos de componentes (entradas externas, salidas externas, consultas externas, grupos de datos lógicos internos y externos), mientras que el MK II FPA mira al sistema como una colección de transacciones lógicas discretas (una transacción lógica es definida como una combinación de uno o más tipos de entradas, un procesamiento y uno o más tipos de salidas, correspondientes a un proceso lógicamente único), compuestas cada una de ellas por entrada, proceso y salida. Si se usan herramientas modernas de diseño para el desarrollo del software, y esas herramientas permiten identificar fácilmente las transacciones lógicas, resulta apropiado el uso de este método.
- 3-D Function Point: entre los años 1989 y 1992, Scott Whitmire desarrolló un método para la empresa internacional de aeronavegación Boing. El objetivo fue ampliar su espectro a sistemas con elevada complejidad como los sistemas en tiempo real. El término 3D, se refiere a que considera tres dimensiones en las que puede proyectarse un sistema software; ellas son: datos, funciones y control. Visto de esta forma, resulta atractivo el uso del método, para aquel tipo de software, pero presenta el inconveniente de la necesidad de disponer de mayor cantidad de información acerca del sistema, sobre todo de la complejidad de los algoritmos a implementarse; esta información no siempre está disponible en las primeras etapas del desarrollo.
- Full Function Points (Puntos de Función Completos): esta técnica ha sido desarrollada por un equipo de la Universidad de Québec en Montreal (Canadá), siendo muy eficiente en la medición de puntos de función en sistemas de control, tiempo real y embebidos.

Existen otras métricas para estimar el tamaño de productos de software preparadas para asimilar, según el desarrollo que han tenido las aplicaciones, el paradigma Orientado a Objetos. Entre ellas tenemos:

- Puntos por Casos de Uso:

  Este método se desarrolló en el año 1993 por Gustav Karner para poder finalmente obtener estimaciones de esfuerzo sobre productos de software orientados a objetos.

  Una de las principales limitaciones del método es que no existe una teoría de cómo escribir o estructurar correctamente los casos de uso, por lo que todas las medidas de tamaño y estimación serán afectadas por la rigurosidad de los analistas.

  Es muy importante señalar que el método exige la existencia de un modelo de casos de uso, por lo que la labor deberá ser hecha cuando exista algún entendimiento del dominio del problema o cuando se esté realizando las labores de arquitectura y dimensionamiento del tamaño del sistema. Por lo general, estas condiciones están dadas al término de las actividades de Análisis.

  En términos simples, el método requiere de casos de uso en modo textual y gráfico sólo en términos de mayor claridad, se revisan en detalle los casos de uso seleccionados en la etapa del proyecto que se defina y se realizan los siguientes pasos:

  Cuantificación de características funcionales del Sistema:

  - Clasificación de Actores, obtención del **P**eso de **A**ctores **S**in **A**justar (PASA).
  - Clasificación de los Casos de Uso, obtención del **P**eso de **T**ransacciones **S**in **A**justar (PTSA)
  - Obtención del Peso o **P**untos de **C**asos de **U**so **S**in **A**justar (PCUSA).

  Cuantificación de características no funcionales del Sistema:

  - Clasificación de **F**actores de **C**omplejidad **T**écnica (FCT)
  - Clasificación de **F**actores **A**mbientales (FA)

- Cálculo de Puntos de Casos de Uso Ajustados (PCU)

Existe una relación natural entre los Puntos de Función y los Casos de Uso. Los Puntos de Función permiten estimar el tamaño del software a partir de sus requerimientos, mientras que los Casos de Uso permiten documentar los requerimientos del software.

Puede decirse entonces que la estimación por Puntos de Caso de Uso resulta muy efectiva para estimar el esfuerzo requerido en el desarrollo de los primeros Casos de Uso de un sistema, si se sigue una aproximación iterativa como el Proceso Unificado de Rational. En este tipo de aproximación, los primeros Casos de Uso a desarrollar son sobre los que recaen la mayor parte de la arquitectura del software y los que a su vez ayudan a mitigar los riesgos más significativos. Fuera de este contexto, el método tiende a sobredimensionar el esfuerzo requerido por lo cual no se recomienda para estimar el esfuerzo global de un proyecto.

- Puntos Objeto:

Diseñado por Kaufman y Kumar en 1993. Es apropiado para las aplicaciones con componentes y para estimar esfuerzos en las etapas de prototipación. Se consideran los elementos siguientes (que denomina objetos, aunque no tienen relación con el paradigma de orientación a objetos): pantallas, informes, y componentes de 3ª generación que necesitará el software que se mide. Los elementos (objetos) se ponderan según su complejidad (se consideran tres niveles: básico, intermedio y avanzado) dándoles un peso estándar para cada nivel y se agregan los valores dando un resultado que son los puntos objeto.

1.3.2 Estimación del esfuerzo

El esfuerzo es un indicador de la cantidad de trabajo necesario para realizar un proyecto o alguno de los ítems de un proyecto. En productos software podemos considerar equivalente estimar el esfuerzo y el coste, ya que existe una relación directa

entre ambos. El esfuerzo es la magnitud de coste de elaboración de un proyecto, y se expresa mediante unidades tales como personas-mes o personas-año.

Al principio, el coste del software constituía un pequeño porcentaje del coste total de los sistemas informáticos basados en computador. Un error considerable en las estimaciones del coste del software tenía relativamente poco impacto. Hoy en día, el software es el elemento más caro de la mayoría de los sistemas informáticos. Un gran error en la estimación del coste puede ser lo que marque la diferencia entre beneficios y pérdidas. Sobrestimar el esfuerzo puede también afectar a la competitividad de la compañía así como provocar perdida de beneficios, pues por ejemplo podría contratarse personal en exceso para la realización del proyecto.

No existe una forma simple de calcular el esfuerzo requerido para desarrollar un sistema informático. Las estimaciones iniciales se hacen bajo la base de la definición de requisitos de usuario de alto nivel. Pero en una primera etapa del proyecto es difícil producir una estimación precisa de los costos de desarrollo del sistema. Por ese motivo la estimación se utiliza para definir si el presupuesto del proyecto y el producto se ajustan para que las cifras del presupuesto se cumplan.

Varios autores destacan tres puntos principales a tratar dentro de la estimación del esfuerzo: proceso de predicción, marco de trabajo para seleccionar magnitudes de predicción, y métodos. El proceso de predicción y el marco de trabajo de selección, destacan la importancia de las métricas, puesto que ellas proporcionan los datos en los cuales se fundamentan las predicciones. El proceso de predicción y el marco de trabajo de selección también enfatizan la necesidad de reutilizar experiencia a fin de mejorar la precisión de la estimación.

1.3.2.1 Métodos predictivos

Según el propósito de la predicción, sus puntos de vista, el tiempo en la cual es realizada, el entorno dentro del que se realiza y la experiencia mediante la cual podemos hacerla; así será la influencia que tenga en la selección de los métodos predictivos. En la estimación del esfuerzo, el marco de trabajo identifica cuatro clases de métodos de predicción: los empíricos, los analógicos, los teóricos y los heurísticos.

Como fue expuesto anteriormente el modelo empírico es el punto de partida para cada método de predicción pues analizan datos para establecer un modelo numérico del relacionante entre medidas de los atributos dentro del modelo empírico.

Los métodos de predicción analógicos, o por analogía, utilizan medidas de los atributos del modelo empírico a fin de caracterizar el caso actual, para el que se realiza la predicción. Las medidas conocidas para el caso actual son usadas para buscar un conjunto de datos que identifiquen casos análogos. La predicción se hace interpolando desde uno o varios casos análogos al caso actual.

Los métodos de predicción teóricos proponen un modelo numérico basado también en el modelo empírico. Los modelos teóricos deben ser validados empíricamente, por comparación con los datos actuales de las medidas.

Los métodos heurísticos por su parte, suelen usarse como extensiones de otros métodos. Las heurísticas son reglas empíricas, desarrolladas mediante experiencia, que obtienen conocimiento acerca de relaciones entre atributos del modelo empírico. Las heurísticas se pueden utilizar para ajustar predicciones realizadas con otros métodos.

A continuación se analizan en detalle algunos de estos métodos.

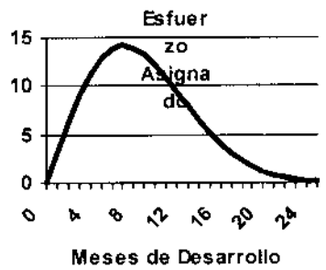
- Métodos basados exclusivamente en la experiencia:
  - *Juicio experto*
    - Puro: un experto estudia las especificaciones y hace su estimación, se basa fundamentalmente en sus conocimientos, si desaparece el experto, la empresa deja de estimar. Método propuesto por Gibson [10] (página 59)
    - Delphi: es una propuesta de O'Conell [11], consiste en un grupo de personas que son informadas y tratan de adivinar lo que costará el desarrollo tanto en esfuerzo, como su duración, teniendo en cuenta que las estimaciones en grupo suelen ser mejores que las individuales. (página 128)

- *Analogía* King da una visión detallada [12] (página 86), consiste en comparar las especificaciones de un proyecto, con las de otros proyectos, teniendo en cuenta factores como el tamaño, los usuarios, la complejidad u otros factores técnicos tales como el sistema operativo, el hardware, el personal del proyecto, etc.
- *Distribución de la utilización de recursos en el ciclo de vida como base de la estimación* como propone King [12] (página 86). Usualmente las instituciones tienen una estructura de costes similar entre proyectos. Si en un proyecto ya se han realizado algunas fases, es de esperar que los costes se distribuyan de manera proporcional.

- Método basado exclusivamente en los recursos: consiste en ver cuánto personal y durante cuánto tiempo se dispone de él. Teniendo como guía la Ley de Parkinson: "El trabajo se expande hasta consumir todos los recursos disponibles".
- Método basado exclusivamente en el mercado: Lo importante es conseguir el contrato. El precio se fija en función de lo que creemos que esta dispuesto a pagar el cliente. El esfuerzo estimado depende del presupuesto del cliente y no de la funcionalidad del software. Si se usa en conjunción con otros métodos puede ser aceptable, para ajustar la oferta. Peligro si es el único método utilizado.
- Métodos basados en los componentes del producto a desarrollar o proceso de desarrollo:
    - Top-Down
        - Se descompone el proyecto en unidades lo menores posibles.
        - Se estima cada unidad y se calcula el coste total.
    - Bottom-up
        - Se ve todo el proyecto, se descompone en grandes bloques o fases.
        - Se estima el coste de cada componente.

- Métodos algorítmicos: Se desarrolla un modelo utilizando información histórica de costos que relaciona alguna métrica de software (por lo general, su tamaño) con el costo del proyecto. Se hace una estimación de esa métrica y el modelo predice el esfuerzo requerido:

    - Putnam: Relaciona cantidad de personas-mes y la duración del proyecto.



      Y=2Kate-at², donde

      Y = Personas-mes en cada punto,

      K = Esfuerzo total del proyecto,

      (Área bajo la curva)

      a = Cte. asociada a la aceleración

      de entrada de personas en el proyecto,

      t = instante del tiempo.

1.3.2.2 Procesos de estimación del coste del software

La mayoría de los procesos de estimación del esfuerzo describen como aplicar un método simple de predicción, el cual está basado en uno o más modelos algorítmicos.

Proceso de Bailey y Basili.

Bailey y Basili propusieron el "meta modelo de Bailey-Basili" como un proceso para estimar el esfuerzo. El proceso tiene tres pasos principales:

- Calibrar el modelo de esfuerzo: El modelo de esfuerzo se calibra usando una regresión por mínimos cuadrados, sobre un conjunto de datos locales para calcular los coeficientes del modelo.
- Estimar los límites superior o inferior del modelo de predicción: Se trata de calcular el error estándar de estimación (standard error of the estimate SEE). El SEE se puede utilizar para calcular los límites superior e inferior del intervalo de confianza construido para el modelo, asumiendo una distribución normal.

- Ajustar las predicciones del modelo para tener en cuenta la significación de los atributos conductores del coste: o sea, el efecto de los conductores del coste sobre el esfuerzo se estima calculando un factor de ajuste de esfuerzo y aplicándolo a la estimación de esfuerzo inicial. El factor de ajuste para un punto de la regresión original es el número mediante el cual el valor efectivo ha de ser multiplicado para obtener el valor predicho. Un modelo de ajuste de esfuerzo se calibra utilizando regresión lineal múltiple para determinar los coeficientes conductores del coste, para tres atributos conductores del coste.

La estimación final se realiza utilizando ambos modelos: el primero para realizar una estimación inicial del esfuerzo, y el segundo para ajustar la estimación, teniendo en cuenta los controladores del coste.

Proceso de Boehm.

Bohem (1981) propuso un proceso de siete pasos para estimar el coste del software:

- Establecer objetivos: El primer paso es determinar nuestros objetivos en la estimación del coste. Para ello se ha de evitar gastar tiempo en recopilar información y realizar estimaciones que no sean relevantes para las decisiones que se han de tomar.
- Planificar recursos y datos requeridos: En el paso siguiente, se debe planificar la actividad de estimación, a fin de evitar la tentación de preparar una estimación demasiado rápida.
- Fijar los requerimientos software: En este tercer paso se determina si las especificaciones en las que está basada la estimación son económicamente viables. Es necesario saber qué se está intentando construir a fin de estimarlo todo con precisión.
- Detallar tanto como sea posible: El cuarto paso indica que hacer una estimación con mucho detalle mejora su precisión.
- Utilizar varias técnicas y fuentes independientes: Bohem clasifica las técnicas de estimación como modelos algorítmicos, opiniones de los expertos, analogía,

Parkinson, price-to-win, top-down y botton-up. Este paso indica que se ha de usar más de una técnica para realizar una estimación del coste. El uso de una combinación de técnicas permite evitar los puntos débiles de cada técnica, así como tomar lo mejor de cada una.

- Comparar e iterar estimaciones: Una vez realizada la estimación utilizando dos o más técnicas, se debe comparar los resultados. Si son inconsistentes, el siguiente paso es investigar las diferencias. El objetivo es converger en una estimación lo más realista posible, mas que establecer un compromiso arbitrario entre las estimaciones iniciales.
- Continuación: En el paso final en el proceso de estimación del coste continúa la estimación realizada en pasos anteriores. Se deben usar los resultados de la comparación entre la estimación y el valor real para mejorar la técnica de estimación y los modelos. Se debería re-estimar el coste cuando el alcance del proyecto cambia.

Proceso de DeMarco.

DeMarco (1982) propone un proceso para desarrollar y aplicar modelos de coste basados en datos recolectados localmente. Él defendía el uso de modelos de coste de factores simples derivados de la regresión lineal. Un modelo de coste de factores simples predice el esfuerzo para todo o parte del proyecto basándose en una variable independiente. El esfuerzo estimado obtenido desde un modelo de coste de factores simples se ha de ajustar aplicando un factor de corrección. Los factores de corrección, se derivan en el mismo modo propuesto por Bailey y Basili.

El modelo de DeMarco soporta estimación botton-up y top-down. Él sugiere dividir el proyecto en componentes de coste, tales como: esfuerzo de diseño, esfuerzo de implementación y esfuerzo de integración. Cada componente de coste es estimado mediante uno o más modelos basados en la medida, los cuales están disponibles en el momento de realizar la estimación.

DeMarco también defendía el uso de modelos sensibles al tiempo, los cuales relacionan el esfuerzo total del proyecto con la duración. Sugiere que se use el modelo

COCOMO de planificación, para este propósito. A continuación se indican las actividades llevadas a cabo para utilizar este modelo:

- Seleccionar el coste del componente a modelar: El primer paso en el proceso implica seleccionar los componentes sobre los que se va a realizar una estimación de coste. El coste de los componentes está basado en una descomposición del sistema de desarrollo en actividades como especificación, diseño, implementación y prueba.
- Seleccionar las medidas desde las cuales predecir los componentes y el coste total: Una vez que los componentes han sido seleccionados, el siguiente paso es seleccionar las medidas desde la cual el esfuerzo para completar esos componentes pueda ser predicho.
- Desarrollar los modelos de coste basados en un simple valor, basándose en los datos existentes: Una vez que se han seleccionado las métricas, los modelos de regresión, mediante los que se predice el esfuerzo, deben ser desarrollados.
- Estimar el coste y el error estándar de la estimación para cada componente del modelo: Cuando el valor de las métricas utilizadas en el modelo de coste basado en un factor simple puede ser estimado o medido, el modelo se utiliza para predecir el esfuerzo de los componentes. Si se utiliza una estimación del valor de las métricas de entrada, el esfuerzo debería ser re-estimado cuando el valor real de estas métricas ha sido calculado.
- Ajustar las estimaciones basadas en diferencias entre el proyecto actual y proyectos pasados: DeMarco indica que la estimación del esfuerzo debe ser ajustada mediante una evaluación subjetiva de las diferencias entre el proyecto en curso y los proyectos pasados.
- Calcular el coste total a partir de los costes de los componentes: Una vez estimado el coste de todos los componentes, se calcula el coste total como suma de estos.

- Calcular el error de estimación global: El error en la estimación total se calcula sumando el error estándar de la estimación de cada componente aislado.
- Utilizar un modelo de coste sensible al tiempo para restringir la estimación total del esfuerzo del proyecto: El paso final en el proceso de DeMarco utiliza la estimación del esfuerzo como entrada a un modelo sensible al tiempo, tal como COCOMO de planificación, para estimar la duración del proyecto.

Proceso de Heemstra.

Heemstra describe un proceso general de estimación de costes. Su proceso asume el uso de modelos de esfuerzo dependientes del tamaño, y el uso de atributos conductores del coste, unidos a estos modelos para realizar ajustes de productividad. Este proceso se divide en siete pasos:

- Crear una base de datos de proyectos finalizados: El paso inicial en el proceso de estimación es la recolección de datos locales para validar y calibrar un modelo.
- Validar el modelo de estimación: El entorno en el que ha sido desarrollado el modelo del coste y los proyectos finalizados en los que está basado, pueden diferir del entorno en el que el modelo es usado. Heemstra advierte, por consiguiente, que antes de utilizar un modelo precalibrado por primera vez en una organización, este debería ser validado, evaluando su precisión sobre datos de proyectos ya finalizados en la organización.
- Calibrar el modelo de estimación: Si en el paso previo se indica que el modelo no es válido para el entorno en el cual va a ser utilizado, el modelo necesita ser recalibrado usando datos de proyectos finalizados en el entorno actual.
- Estimar el tamaño: En este paso, se calcula el tamaño del software partiendo de las características propias del software a desarrollar.
- Estimar el esfuerzo y la productividad: Una vez realizada la estimación del tamaño, el siguiente paso convierte la estimación del tamaño en estimación de esfuerzo. Los atributos conductores del coste con influencia en el esfuerzo del

desarrollo del software son evaluados y utilizados para ajustar la estimación del esfuerzo.

- Distribuir el esfuerzo a lo largo de las fases de realización del proyecto: En este paso del esfuerzo total y la duración del proyecto son distribuidas a lo largo de las fases de desarrollo del software.
- Analizar la sensibilidad de la estimación y los riesgos asociados: En este paso se evalúa la sensibilidad del coste estimado para valores de los atributos conductores del software, y se determinan los riesgos asociados con la estimación de costes del proyecto.

Proceso de Arifoglu.

El proceso de estimación de Arifoglu consta de cuatro pasos:

- Estimación del tamaño: Este paso es equivalente a la estimación de tamaño del proceso de Heemstra.
- Estimación del esfuerzo y duración: Este paso convierte la estimación del tamaño en estimación del esfuerzo y estimación de la duración. Arifoglu sugiere el uso del modelo COCOMO en este paso.
- Distribución del esfuerzo y la duración durante el ciclo de vida: Una vez que el esfuerzo total y la duración del proyecto ha sido estimado, han de ser distribuidos a lo largo del ciclo de vida.
- Reflejar el esfuerzo en el calendario: El paso final del proceso de Arifoglu es convertir el número de días de trabajo requeridos para completar el proyecto en número de días transcurridos en el calendario. Arifoglu es partidario del uso del modelo de Esterling, para realizar esta conversión. Esterling propone un modelo para estimar el porcentaje de día gastado trabajando directamente en una tarea.

Este paso realiza la predicción de la duración del proyecto usando el modelo COCOMO u otro similar.

PROBE.

Humphrey (1995) describe un proceso de estimación basada en proxys (proxy-based estimation PROBE) como parte de su proceso de software personal (PSP). El método PROBE defiende el uso de medidas seleccionadas personalmente y modelos de regresión basadas en datos personales para estimar el tamaño de un producto software. Un proxy es una medida de tamaño que puede ser utilizada para estimar la longitud de un proyecto medida en líneas de código. La estimación de líneas de código se utiliza para predecir el esfuerzo individual para desarrollar el esfuerzo. Los límites superior e inferior del intervalo de predicción deben ser también estimados.

Se utilizan modelos de regresión simples para estimar las líneas de código desde las medidas de tamaño basadas en proxys y las horas de trabajo desde las líneas de código. Se realiza una vuelta atrás desde cada estimación, comparando el valor estimado con el real. La vuelta atrás se introduce para mejorar el rendimiento de la estimación individual. La filosofía del proceso personal del software es mejorar las facultades de los individuos. Esto limita la aplicabilidad de PROBE a equipos de trabajo.

Wideband Delphi.

Es posible que la precisión de un método de estimación pueda ser mejorada mediante la opinión y la estimación de un grupo de personas. Boehm (1981) describe la técnica Wideband Delphi para combinar las opiniones de los expertos y realizar una estimación de tamaño.

A cada experto se le proporciona una especificación y un formulario de estimación. En una reunión los expertos discuten los asuntos de la estimación. Cada experto rellena su formulario de forma anónima. En esta ronda, se proporciona a los expertos un sumario de las estimaciones realizadas, posteriormente se mantiene otra reunión para discutir las estimaciones de la ronda anterior. Se celebrarán mas rodas hasta que las estimaciones converjan satisfactoriamente.

Wideband Delphi es similar al Delphi original, desarrollado por RAND Corporation. No obstante, antes de hacer cada ronda de estimaciones anónimas, los expertos discuten

sobre las estimaciones anteriores. En la técnica original no había interacción entre los expertos.

En la mayoría de los procesos de estimación mencionados anteriormente se menciona la utilización de COCOMO. En próximos epígrafes haremos referencia a este método de estimación para calcular el esfuerzo y la duración de un proyecto de software.

1.3.2.3 Modelos de estimación de esfuerzo

De acuerdo al método de predicción en el que se basa, existen varias clasificaciones para los modelos de estimación de esfuerzo: modelos empíricos paramétricos y no paramétricos, modelos analógicos y modelos teóricos. Los más comunes son los empíricos paramétricos, que por su parte utilizan fórmulas derivadas empíricamente para predecir el esfuerzo en función de las LDC o los PF. Su fórmula general es:

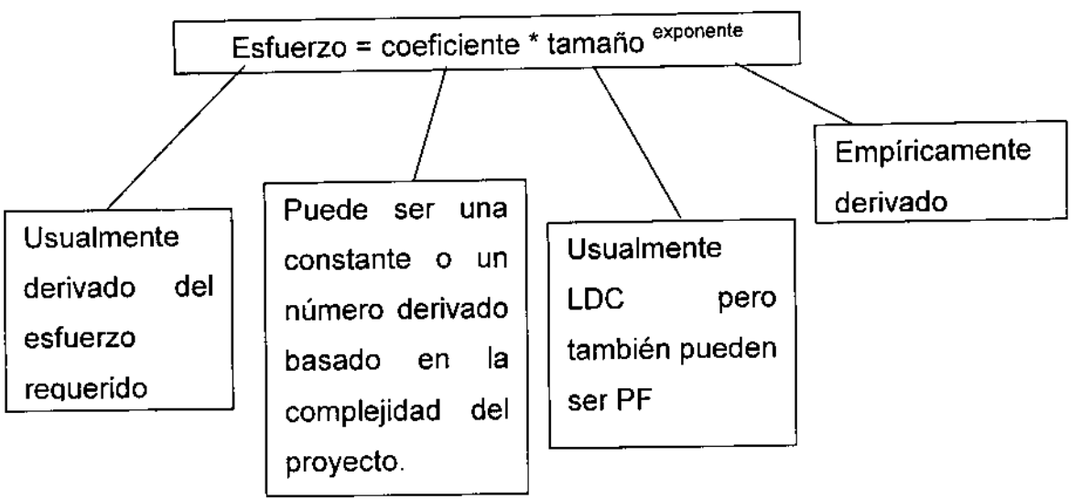


Figura 4: Fórmula general de estimación de esfuerzo.

Algunas estimaciones de esfuerzo por modelos empíricos son:

Orientados a LDC

$E= 5.2*(KLDC)^{0.91}$	Modelo Walston-Felix
$E= 5.5+0.73*(KLDC)^{1.16}$	Modelo Bailey-Basisli
$E= 3.2*(KLDC)^{1.05}$	Modelo de Bohem

$E= 5.288*(KLDC)^{1.047}$	Doty KLOC>9
Orientados a PF	
$E= -13.39 + 0.054\ PF$	Modelo de Albretch y Gaffney
$E= 60.62*7.728*10^{-8}PF^3$	Modelo de Kemerer
$E= 585.7+15.12 * PF$	Modelo de Matson, Barnett y Mellichamp

Tabla 1: Estimaciones de esfuerzo por modelos empíricos.

WALSTON-FELIX

Este estudio es probablemente el primer intento sistemático para coleccionar datos de proyectos de software bajo condiciones semi-controladas, y se desarrolló a comienzos de los años setenta, en IBM. Informes detallados de 60 proyectos diferentes fueron recogidos en diversos momentos del desarrollo. Los datos recolectados incluyeron esfuerzos de desarrollo, recursos computacionales, datos de completación de las fases, errores, número de módulos, grado de uso de prácticas modernas de programación, páginas de documentación y lenguajes usados, entre otros.

DOTY

Doty procede de Doty Associates, la empresa contratista en la que se llevó acabo el estudio encargado por el Rome Air Development Center (RADC) del Data and Análisis Center for Software (DACS) del Deparment of Defense (DoD) de los Estados Unidos de América, que permitió el desarrollo de este modelo. Es un modelo no lineal publicado en 1877.

Este modelo estima el esfuerzo, costes y tiempo de desarrollo para un proyecto software determinado. Para elegir los valores de las constantes de la ecuación de cálculo, el modelo Doty distingue cuatro tipos de programas o módulos de programas. El tamaño del sistema propuesto se estima por comparación con sistemas análogos. Una de las conclusiones introducidas por este modelo es la de que el tiempo que se requiere para escribir un número dado de instrucciones en un lenguaje de alto nivel es el mismo que para escribirlas en un lenguaje ensamblador, pero como el lenguaje de

alto nivel es más pequeño, el resultado es que se incrementa la productividad. Además de que el lenguaje de alto nivel es superior en claridad y por lo tanto facilita el mantenimiento.

Por otra parte deben destacarse los modelos teóricos, es decir, aquellos que se derivan de hipótesis teóricas acerca del comportamiento de los proyectos. Entre ellos sobresale SLIM.

SLIM

Surge en 1978 como solución a un requerimiento de la marina de EEUU para proveer un método para estimar esfuerzo y tiempo. Fue desarrollado por Putnam y lo llamó modelo SLIM (Software LIfecycleModel). Este modelo se utiliza para estimar proyectos con más de 70.000 LOC, aunque puede ser ajustado para proyectos más pequeños. El modelo asume que el esfuerzo para proyectos de desarrollo de software se distribuye de forma similar a una colección de curvas de Rayleigh, una para cada actividad del desarrollo. (Ver Anexo 8)

Este modelo ha recibido críticas por el hecho de que las curvas de Norden-Rayleigh no consideran la fase de especificación de requisitos, por lo que tampoco se tiene en cuenta esta fase del desarrollo en la correspondiente ecuación propuesta por Putnam. Algunos técnicos consideran difícil utilizar este modelo en entornos de desarrollo pequeños. Esta consideración tiene su origen en que los datos recopilados para su creación han sido tomados en entornos de desarrollo grandes.

Putnam utiliza observaciones empíricas sobre niveles de productividad para derivar su *ecuación de software* a partir de la fórmula básica de la curva de Rayleigh, basándose en que la cantidad de trabajo para desarrollar cualquier software se puede concebir como el producto del esfuerzo realizado en un período de tiempo. Así, se puede escribir:

$$L = C * K^{1/3} t_d^{4/3}, \text{ donde:}$$

- L: el tamaño del producto (en LOC) representa una cierta medida de su funcionalidad.

- C: factor tecnológico que incluye 14 conductores de costos y puede tomar hasta 20 valores distintos.
- K: representa el esfuerzo del trabajo humano, medido en personas-mes o personas-año.
- t_d: representa el tiempo que dura el trabajo, es decir, el tiempo transcurrido para la entrega, es medido en meses o en años. Es el punto donde la curva alcanza un máximo.

Se puede decir que en el modelo de Putnam al disminuir la duración aumenta el esfuerzo, pero aumentando la duración disminuye el esfuerzo. De esta forma el tamaño del producto es factor determinante del esfuerzo requerido para construir el producto. Por ello SLIM, como la mayoría de los modelos sugiere que el esfuerzo requerido es aproximadamente proporcional al tamaño, pero incluye un ajuste inverso a la economía de escala, es decir, proyectos más grandes son menos productivos que los más pequeños.

1.3.3 Estimación de la duración

El resultado es una estimación de la duración en unidades temporales: días, semanas, meses. Los métodos más habituales para calcular la duración de un proyecto software a partir de la estimación del esfuerzo son:

- utilización de datos anteriores de la organización,
- utilización de tablas de estimación para convertir desde líneas de código a esfuerzo y duración
- utilización de funciones de equivalencia semiempíricas del tipo

  duración = función del esfuerzo

  que incluyen diversos parámetros cuyos valores se determinan empíricamente.
- utilización de software de gestión de proyectos que permita realizar una planificación de la duración optimizando la utilización de los recursos disponibles

1.3.3.1 PERT

El método PERT (Program Evaluation and Review Technique) fue desarrollado por la Armada de los Estados Unidos de América, en 1957, para controlar los tiempos de ejecución de las diversas actividades que integraban los proyectos espaciales, por la necesidad de terminar cada una de ellas dentro de los intervalos de tiempo disponibles. Fue utilizado originalmente por el control de tiempos del proyecto Polaris.

El objetivo principal de los sistemas tipo PERT consiste en ayudar en la planeación y el control, por lo que no implica mucha optimización directa. Este se desglosa en algunos objetivos más específicos, teniendo siempre en cuenta el determinar la probabilidad de cumplir con fechas de entrega específicas. También identifica aquellas actividades que son más probables que se conviertan en cuellos de botella y señala, por ende, en qué puntos debe hacerse el mayor esfuerzo para no tener retrasos. Otro de sus objetivos secundarios es evaluar el efecto de los cambios del programa y el efecto de desviarse de lo programado.

En resumen, PERT supone que el tiempo para realizar cada una de las actividades es una variable aleatoria descrita por una distribución de probabilidad. Es en particular apropiado cuando se maneja mucha incertidumbre al predecir los tiempos de las actividades y cuando es importante controlar de una manera efectiva la programación del proyecto; como sucede con los proyectos de investigación y desarrollo.

1.3.3.2 CMP

El método CPM (Crítical Path Method) fue desarrollado también en 1957 en los Estados Unidos de América, por un centro de investigación de operaciones para la firma Dupont y Remington Rand, buscando el control y la optimización de los costos de operación mediante la planeación adecuada de las actividades componentes del proyecto. No solamente se llama camino crítico al método sino también a la serie de actividades contadas desde la iniciación del proyecto hasta su terminación, que no tienen flexibilidad en su tiempo de ejecución, por lo que cualquier retraso que sufriera alguna de las actividades de la serie provocaría un retraso en todo el proyecto. Desde otro punto de vista, camino crítico es la serie de actividades que indica la duración total

del proyecto. Cada una de las actividades se representa por una flecha que empieza en un evento y termina en otro.

El objetivo fundamental del CPM es determinar el trueque entre tiempo y costo que debe emplearse en cada actividad para cumplir con el tiempo de terminación del proyecto que se programó a un costo mínimo. A diferencia de PERT el CPM asigna la misma importancia al tiempo y al costo.

CPM supone que los tiempos de las actividades son determinísticos (es decir, se pueden predecir de manera confiable sin incertidumbre significativa) y se puede variar cambiando el nivel de recursos utilizados. Este método resulta muy apropiado cuando se pueden predecir bien los tiempos de las actividades (pudiera ser quizá con base en la experiencia), cuando estos tiempos se pueden ajustar con facilidad, al igual que cuando es importante planear una combinación apropiada entre el tiempo y el costo del proyecto.

En la actualidad, las diferencias entre las versiones actuales de PERT y CPM no son tan marcadas como se han descrito. Muchas versiones de PERT permiten emplear una sola estimación (la más probable) para cada actividad y omiten así la investigación probabilística.

Conclusiones

A lo largo de todo este Capítulo se ha hecho un amplio análisis de los métodos y procesos de estimación. Desglosándolos para una mayor organización en los que corresponden a la estimación del tamaño, del coste y del tiempo. Aunque es palpable el hecho de la gran interrelación que hay entre ellos, creando a veces una dependencia descrita en el proceso que ha de aplicarse a un proyecto de software en específico.

Por otra parte se describió en detalle la propuesta hecha por el PMI para la Gestión del Tiempo de un proyecto de software, centrándose el resto del Capítulo en detallar técnicas y herramientas vinculadas a la 3era actividad: la estimación de la duración de las actividades.

Capítulo 2. Teorías de Boehm y Humphrey.

Introducción

Dos de los puntos de vista más especializados en los temas de estimación son los propuestos por Barry Boehm y Watts Humphrey. El hecho de comentar de manera general sólo sus procesos en el Capítulo anterior, sirve de preámbulo al análisis más detallado de sus teorías en este Capítulo.

2.1 Barry Boehm. Su teoría.

Barry Boehm, licenciado en matemáticas en la Universidad de California, Los Ángeles en 1964, se desempeñó por muchos años en diferentes roles, desde simple Programador-Analista en General Dynamics entre 1955 y 1959, pasando por científico principal del grupo de sistemas de defensa en Thompson Ramo Wooldridge Inc (TRW), entre 1973 y 1989 hasta servir entre 1989 y 1992, en el departamento de defensa de los Estados Unidos como director de la oficina de las ciencias y tecnología de la información de Defense Advanced Research Projects Agency (DARPA), y como director de software, Defense Research & Engineering, en la oficina de informática. Actualmente trabaja en la AIAA, en ACM, en IEEE, y es miembro de la academia nacional de ingeniería.

Sus intereses actuales en la investigación incluyen los campos vinculados con 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.

Sus contribuciones en estos temas 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.

2.1.1 COCOMO

2.1.1.1 Breve historia

El modelo original COCOMO se publicó por primera vez en 1981 por Barry Boehm y reflejaba las prácticas en desarrollo de software de aquel momento. Durante los años 80, el modelo se continuó perfeccionando y consolidando, siendo el modelo de estimación de costos más ampliamente utilizado en el mundo.

En la década y media siguiente las técnicas de desarrollo software cambiaron drásticamente, un giro total desde los mainframe hacia los sistemas en tiempo real. La ventaja competitiva dependía cada vez más del desarrollo de productos inteligentes y a la medida, y de la habilidad de desarrollar y adaptar más rápidamente estos productos y servicios que el resto de los competidores. Una nueva generación de procesos y productos software cambiaron la forma de desarrollarlos, mejorando su calidad y reduciendo riesgo, coste y tiempo: procesos evolutivos, colaborativos y guiados por los riesgos; desarrollo rápido, uso de paquetes y reutilización de software de trayectoria rápida; organizaciones orientadas a la madurez de sus procesos, etc.

La reducción de los costes del hardware y la comodidad de soluciones que ofrece el software han influido indirectamente en los costes del desarrollo de sistemas y dan aún más importancia a unos cálculos de coste-beneficio que el convenzan a los directivos financieros de la ventaja comercial que dan las inversiones en software, así como a la selección de componentes para la construcción y evolución del ciclo de vida de los sistemas, junto a otras cuestiones como la concurrencia de productos y procesos o la capacidad para analizar sus rendimientos costes, tiempos, funciones, resultados y calidades. [la medición del softw, Francisco Sanchis]

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), IRUS at UC Irvine 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. Por tanto, COCOMO II es un modelo que permite estimar el coste, esfuerzo y tiempo cuando se planifica una nueva actividad de desarrollo software. Está asociado a los ciclos de vida de software modernos. Para evitar confusión el modelo COCOMO original fue redesignado con el nombre COCOMO' 81.

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 qué efectos provoca el ajuste de requerimientos y recursos sobre la estimación de costos y tiempos. Existe una nomenclatura para distinguir el modelo teórico, de la implementación, es decir, se denomina COCOMO II al modelo y USC COCOMOII a la herramienta de software. Se han desarrollado varios release de software y cada uno de ellos está identificado por el año calendario y el número de la versión, aunque la USC libera de manera oficial solo una versión al año.

Éste método es el preferido en la actualidad para la estimación del esfuerzo cuando no se tiene información histórica a la cual recurrir.

2.1.1.2 COCOMO 81

Este fue presentado por Barry Boehm 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 los 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.

Por otra parte Boehm presenta una jerarquía de modelos de estimación según el nivel de detalle empleado en su utilización:

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

• **Intermedio**. En éste 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.

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

Modelo Básico:

	Orgánico	Semiempotrado	Empotrado
Esfuerzo estimado	$E_D=2,4(KLDC)^{1,05}$ h-m	$E_D=3,0(KLDC)^{1,12}$ h-m	$E_D=3,6(KLDC)^{1,20}$ h-m
Tiempo de desarrollo	$T_D=2,5(E_D)^{0,38}$ m	$T_D=2,5(E_D)^{0,35}$ m	$T_D=2,5(E_D)^{0,32}$ m
Productividad	$PR = LDC / E_D$		

h=hombre, m=mes, h-m=hombres-mes

Tabla 2: Esfuerzo para el modelo básico.

Modelo Intermedio:

	Orgánico	Semiempotrado	Empotrado
Ecuaciones del esfuerzo nominal	$E_N=3,2(KLDC)^{1,05}$	$E_N=3,0(KLDC)^{1,12}$	$E_N=2,8(KLDC)^{1,20}$

Atributos	Valor					
	Muy bajo	Bajo	Nominal	Alto	Muy alto	Extra alto
Atributos del producto						
Fiabilidad	,75	,88	1,00	1,15	1,40	
Tamaño base de datos		,94	1,00	1,08	1,16	
Complejidad	,70	,85	1,00	1,15	1,30	1,65
Atributos del computador						
Restricciones de tiempo de ejecución			1,00	1,11	1,30	1,66
Restricciones de memoria virtual			1,00	1,06	1,21	1,56
Volatilidad de la máquina virtual		,87	1,00	1,15	1,30	
Tiempo de respuesta		,87	1,00	1,07	1,15	
Atributos del personal						
Capacidad de análisis	1,46	1,19	1,00	,86	,71	
Experiencia en la aplicación	1,29	1,13	1,00	,91	,82	
Calidad de los programadores	1,42	1,17	1,00	,86	,70	
Experiencia en la máquina virtual	1,21	1,10	1,00	,90		
Experiencia en el lenguaje	1,14	1,07	1,00	,95		
Atributos del proyecto						
Técnicas actualizadas de programación	1,24	1,10	1,00	,91	,82	
Utilización de herramientas software	1,24	1,10	1,00	,91	,83	
Restricciones de tiempo de desarrollo	1,23	1,08	1,00	1,04	1,10	

Factores multiplicadores del esfuerzo de desarrollo de software

Tabla 3: Esfuerzo para el modelo intermedio.

Modelo Avanzado:

Este modelo presenta principalmente dos mejoras respecto al anterior modelo:

- Los factores correspondientes a los atributos son sensibles a la fase sobre la que se realizan las estimaciones, puesto que aspectos tales como experiencia en la aplicación, utilización de herramientas software, etc., tienen mayor influencia en unas fases que en otras.
- Establece una jerarquía de tres niveles de productos, de forma que:
  - los aspectos que presentan gran variación a bajo nivel, se consideran a nivel módulo.
  - los que presentan pocas variaciones, a nivel de subsistema
  - los restantes se consideran a nivel sistema.

2.1.1.3 COCOMO II

Definición del modelo, fundamentos y estrategia

Los objetivos principales que se tuvieron en cuenta para construir el modelo COCOMO II fueron:

- 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.
- Construir una base de datos de proyectos de software que permitiera la calibración continua del modelo, y así incrementar la precisión en la estimación.
- Implementar una herramienta de software que soportara el modelo.
- 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.

Los cuatro elementos principales de la estrategia de COCOMO II son:

1. Preservar las habilidades y apertura del modelo original, para que continúe siendo un modelo abierto y público (algoritmos, parametrizaciones, etc.)
2. Adaptar la estructura de COCOMO II a los futuros sectores del mercado software.
3. Adaptar las entradas y salidas de los submodelos de COCOMO II al nivel de información disponible en cada etapa.
4. Permitir a los submodelos de COCOMO II ajustarse a la medida dependiendo de la estrategia utilizada en cada proyecto particular (atender a las distintas calibraciones de los submodelos que se utilicen para obtener mayor fiabilidad en la estimación global).

Un fundamento importante es el considerar la granularidad del modelo de estimación de coste, teniendo en cuenta la información disponible que sirve de soporte al modelo, entendiendo que en las primeras etapas del proyecto software se conocen muy pocas cosas sobre el tamaño del producto a ser desarrollado, la naturaleza de la plataforma

objetivo, la naturaleza del personal involucrado en el proyecto, o los detalles específicos del proceso que se utilizará. [13]

Tal como muestra la tabla expuesta en el Anexo 3 [14], COCOMO II está compuesto por tres modelos denominados: Composición de Aplicación, Diseño Temprano y Post-Arquitectura. Más adelante se detallará cada uno de ellos, los que surgen en respuesta a la diversidad del mercado actual y futuro de desarrollo de software.

La siguiente figura resume el modelo de mercado de actuales y futuras prácticas de software que se usan para guiar el desarrollo de COCOMO II. Incluye en la plataforma superior un sector dedicado a la programación para usuarios finales a la que pertenecerían alrededor de 55 millones de personas en Estados Unidos en el año 2005. A un nivel más bajo, un sector de infraestructura, al que pertenecerían en la misma fecha aproximadamente 0.75 millones de personas y 3 sectores intermedios que incluyen el desarrollo de generadores de aplicaciones y ayudas para la composición (0.6 millones), desarrollo de sistemas mediante la composición de aplicaciones (0.7 millones) y sistemas de integración a gran escala y/o sistemas de software embebidos (0.7 millones) [15].

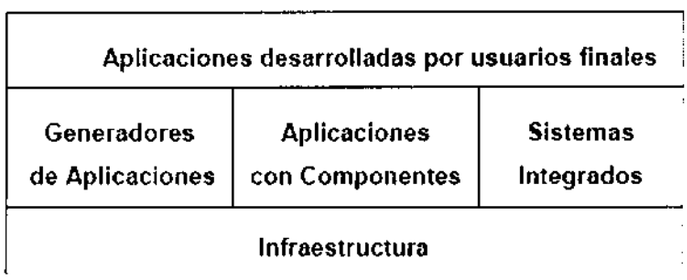


Figura 5: Distribución del Mercado de Software

- 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 componentes típicas, que son constructores de interfases gráficas, administradores de bases de datos, buscadores inteligentes de datos, componentes de dominio-específico.
- 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.
- 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 que forman COCOMO II se adaptan tanto a las necesidades de los diferentes sectores descriptos, 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.

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 emplea en desarrollos de software durante la etapa de prototipación.

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, lo que concuerda con el uso de Puntos Función, para estimar tamaño y el uso de un número reducido de factores de costo.

El modelo ***Post-Arquitectura*** se aplica en la etapa de desarrollo propiamente dicho, después que se define la arquitectura del sistema, y en la etapa de mantenimiento. Este modelo utiliza:

- Puntos Función y/o Líneas de Código Fuente para estimar tamaño, con modificadores que contemplan el reuso, con y sin traducción automática, y el "desperdicio" (breakage)
- 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.

A continuación de especifican cada uno de estos modelos y su aplicación:

Modelo de Composición de Aplicación

Este modelo está dirigido a aplicaciones que están demasiado diversificadas como para crear rápidamente una herramienta específica dentro de un dominio. Los sistemas basados en componentes son los generadores de interfaces de usuario (GUI), gestores de objetos o bases de datos, procesamiento distribuido o de transacciones, manejadores hipermedia, pequeños buscadores de datos, y componentes de un dominio específico tales como dominios financieros, médicos, o paquetes de control de procesos industriales. Estos sistemas se caracterizan por llevar asociados esfuerzos de prototipado, uso de ICASE (CASE integrado) para obtener una composición rápida asistida por la computadora, herramientas de desarrollo software, y un largo número de componentes de aplicaciones e infraestructuras.

Estudios experimentales realizados demuestran que tanto el uso de Puntos Función como de Puntos Objeto, ambos descritos anteriormente, dan unas estimaciones muy próximas, aunque vale señalar que el método de los Puntos Objeto resulta más fácil de utilizar, pues 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, razones que justifican la selección de este último método como el punto de partida para el modelo de estimación de Composición de Aplicaciones.

Modelos Diseño Temprano y Post-Arquitectura

Los modelos de Diseño Temprano y Post-Arquitectura se basan en la misma filosofía a la hora de proporcionar una estimación. 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:

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

Se expresa el esfuerzo en términos de Personas Mes (PM). Una persona mes es la cantidad de tiempo que una persona dedica a trabajar sobre el proyecto de desarrollo software durante un mes. Las entradas son el tamaño del desarrollo software, una constante A, y un factor de escala B. El tamaño se da en miles de líneas de código fuente (KSLOC). Pudiéndose estimar también utilizando Puntos Función Desajustados (UFP), y convertirlos a SLOC dividiendo por mil. El factor de escala (o exponencial) B por su parte, cuenta la relativa economía, positiva o negativa, de la escala encontrada en proyectos software según cambie el tamaño de éste. La constante A es usada para capturar los efectos multiplicativos sobre el esfuerzo con proyectos que incrementan su tamaño.

La variable **Size**, se calcula de la siguiente forma

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

donde **BRAK** es el porcentaje de Rotura para ajustar el tamaño eficaz del producto que utiliza COCOMO II. Viene del término *Breakage*, que hace referencia a la volatilidad de los requerimientos de un proyecto. El factor BRAK no se usa en el Modelo de Composición de Aplicaciones donde se espera un cierto grado de iteración en el producto y se incluye en la calibración de datos.

Esta variable nos da el tamaño de una aplicación que se mide en unidades de líneas de código fuente (KSLOC). Al igual que en la versión inicial del COCOMO, este valor se deriva de la medida de módulos software que constituirán el

programa de aplicación, sin embargo, en la nueva versión COCOMO II puede estimarse también a partir de Puntos de Función sin ajustar convirtiendo a SLOC y luego dividiendo por 1000.

Si se opta por utilizar directamente el valor del número de líneas de código, la meta es medir la cantidad de trabajo intelectual que se emplea en el desarrollo del programa, pero las dificultades aparecen al intentar definir medidas consistentes en diferentes lenguajes. Si por el contrario se opta por utilizar los Puntos de Función sin Ajustar para determinar el tamaño del proyecto, éstos deben convertirse en líneas de código fuente para evaluar la relativamente concisa implementación por Puntos de Función. COCOMO II realiza esto tanto en el Modelo de Diseño Temprano como en el de Post-Arquitectura usando tablas que traducen Puntos de Función sin ajustar al equivalente SLOC.

La variable **B** es el factor exponencial utilizado para considerar los gastos y ahorros relativos de escala encontrados en proyectos software de distinto tamaño. Se obtiene:

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

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.

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

Si **B > 1.0**: El proyecto presenta gastos de escala. Esto se debe normalmente a dos factores principales: el crecimiento del gasto en comunicaciones y el gasto en crecimiento de la integración de un gran sistema. Los proyectos más grandes tendrán más personal y por lo tanto más vías de comunicación interpersonales

produciendo gasto. Integrar un producto pequeño como parte de uno más grande requiere no sólo el esfuerzo de desarrollar el producto pequeño sino también el gasto adicional en esfuerzo para diseñar, mantener, integrar y probar sus interfaces con el resto del producto.

El exponente B se obtiene mediante los denominados drivers de escala. La selección de drivers de escala se basa en la razón de que ellos son un recurso significante de variación exponencial en un esfuerzo ó variación de la productividad del proyecto. Cada driver de escala tiene un rango de niveles de valores desde Muy Bajo hasta Extra Alto. Cada nivel de valores tiene un peso, SF, y el valor específico del peso se llama factor de escala. Un factor de escala de un proyecto, SFj se calcula sumando todos los factores y se usa para determinar el exponente de escala, **B**. Los 5 factores de escala están relacionados con la Precedencia, la Flexibilidad en el desarrollo, la Arquitectura/Resolución de riesgo, la Cohesión del Equipo y la Madurez del Proceso. Ver Anexo 3.

Ajuste mediante drivers de coste

Los drivers de coste se usan para capturar características del desarrollo del software que afectan al esfuerzo para completar el proyecto. Estos drivers tienen un nivel de medida que expresa su impacto en el esfuerzo de desarrollo. Sus valores pueden ir desde Extra Bajo hasta Extra Alto. Cada uno de estos drivers tiene un peso asociado, para su análisis cuantitativo, este peso se llama multiplicador de esfuerzo (EM). Los EM se usan para ajustar el esfuerzo Persona-Mes nominal.

$$PM_{estimado} = PM_{nominal} \times \prod_{i=1}^{17} EM_i$$

La fórmula anterior da el esfuerzo estimado al ponderar teniendo en cuenta dichos multiplicadores de esfuerzo. Téngase en cuenta además que los valores a tomar por la *i* están en correspondencia con el Modelo Post-Arquitectura. En el caso del Modelo de Diseño Temprano, serían 7 los multiplicadores de esfuerzo.

Los Multiplicadores de Esfuerzo se clasifican en cuatro drivers o áreas:

- Producto
  - Confianza Software Requerida (RELY).
  - Tamaño de Base de Datos (DATA)
  - Complejidad del Producto (CPLX).
  - Reutilización Requerida (RUSE).
  - Documentación relacionada con las necesidades del ciclo de vida (DOCU).
- Plataforma
  - Tiempo de Ejecución Necesitado (TIME).
  - Volatilidad de la plataforma (PVOL).
- Personal
  - Capacidad de los Analistas (ACAP).
  - Capacidad del Programador (PCAP).
  - Experiencia en las Aplicaciones (AEXP).
  - Experiencia en la Plataforma (PEXP).
  - Experiencia con Herramientas y Lenguajes (LTEX).
  - Continuidad del Personal (PCON).
- Proyecto
  - Uso de herramientas software ITOOL).
  - Desarrollo Multisitio (SITE).
  - Planificación de Desarrollo Requerida (SCED).

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

Figura 6: Multiplicadores de esfuerzo para los Modelos de Diseño Temprano y el Post-Arquitectura.

El último Multiplicador de esfuerzo que tratamos, relacionado con el driver de Proyecto (SCED), está directamente relacionado con la medición de las restricciones de horario impuestas al equipo de proyecto que desarrolla el software. Los valores se definen en términos de porcentaje de aceleración ó alargamiento sobre el calendario respecto de un calendario nominal para un proyecto que requiere una cantidad de esfuerzo dada.

Tiempo de desarrollo

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

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

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, es la estimación de meses-persona (negado porque excluye el estimador de esfuerzo SCED)
- B, es la suma de los factores de escala del proyecto y
- SCED % es el porcentaje de compresión/expansión en el multiplicador de esfuerzo SCED.

Las tres etapas del modelo COCOMO II permiten la estimación de rangos probables de valores de salida usando distintas relaciones de exactitud de coste y medida. Una vez que se calcula el valor de esfuerzo más probable, E, del modelo elegido: Composición de Aplicaciones, Diseño anticipado o Post-Arquitectura, se calculan un conjunto de estimaciones optimistas y pesimistas que representan una desviación estándar alrededor del valor más probable

Para finalizar se indican algunos de los aspectos a favor y en contra de la utilización del modelo COCOMO II:

2.2. Watts Humphrey. Su teoría.

2.2.1 Watts Humphrey

Watts S. Humphrey fundó el Proceso de Software Personal (PSP) en el Instituto de Ingeniería de Software en la Universidad Carnegie-Mellon. Desde 1959 y hasta 1986 estuvo asociado a IBM Corporation donde era gerente de programación. Sus publicaciones incluyen numerosos artículos técnicos y nueve libros. Los tres más recientes son "TSP: Entrenar a los equipos del desarrollo" (2006), "TSP: Conducir un equipo del desarrollo" (2005), y "PSP: Un proceso de mejora personal para ingenieros de software" (2005). Se licencia en física en la Universidad de Chicago, luego realiza un masters en Física en el instituto de tecnología de Illinois, y otro masters en Administración de Negocios en la Universidad de Chicago

En el 2003 recibió la medalla nacional de la tecnología que es el honor más alto que concede Estados Unidos a los principales innovadores de América por sus contribuciones en la comunidad de ingenieros de software. Ha recibido otros premios y condecoraciones y goza de gran prestigio internacional.

2.2.2 Personal Software Process

2.2.2.1 Breve historia

Después de la segunda guerra mundial, la estrategia de calidad en la mayoría de las organizaciones industriales se basaba casi por completo en las pruebas. Las empresas establecieron departamentos especiales de la calidad para encontrar y arreglar problemas después de la producción de los productos.

Aunque la mayoría de las organizaciones industriales ahora han adoptado principios modernos de calidad, la comunidad del software ha continuado confiando en la prueba como el método principal de la administración de la calidad. En 1978 Michael Fagan introdujo la primera medida principal para garantizar la calidad de software, las inspecciones de software.

Usando inspecciones, las organizaciones han mejorado substancialmente la calidad del software. Una medida significativa en la mejora de calidad del software fue tomada con la introducción del modelo de capacidad de madurez (CMM) en 1987. El enfoque principal de CMM estaba en el sistema que administraba la ayuda que se le proporcionaba a los ingenieros de desarrollo. CMM ha tenido un efecto positivo en el funcionamiento de las organizaciones del software.

Hubo además otra medida significativa en la mejora de calidad del software fue tomada con la esencia del proceso personal del software (PSP) ya que PSP amplia el proceso de mejora a la gente que realiza el trabajo de desarrollo de software.

PSP se concentra en las prácticas de trabajo de los ingenieros en una forma individual. El principio detrás de PSP es ése, 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.

Así mismo 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. [SEI; 2000]

2.2.2.2 ¿Cómo fue desarrollado PSP?

Después de que Watts S. Humphrey condujera el desarrollo inicial de CMM para software, se decidió a aplicar los principios de CMM a los programas pequeños. Mientras que los principios de CMM se aplicaron a tales grupos, cada vez se volvía más necesaria la asesoría para saber que hacer. Fue entonces cuando Humphrey decidió personalmente utilizar los principios de CMM para desarrollar programas modulares para ver si dicho enfoque podría funcionar para convencer a los ingenieros de software a que adoptaran tales prácticas.

En el desarrollo de estos programas modulares, Humphrey utilizó personalmente todas las prácticas de CMM para tratar de ir subiendo hasta llegar al nivel 5. Poco después el Instituto de la Ingeniería de Software (SEI), lo hizo un colaborador, permitiéndole trabajar tiempo completo en la investigación detallada de PSP.

De la experiencia de sus investigaciones, concluyó que los principios de la administración de procesos que desarrollados por W. Edwards Deming y J.M. Juran en las décadas del 70 y del 80, centrada en mejorar la forma en la que la gente hacía sus trabajos y desarrollaban sus procesos, eran totalmente aplicables tanto al trabajo de los ingenieros de software de manera individual como a ingenieros enfocados al trabajo en equipo. Como resultado obtuvo entonces el Proceso en equipo de software (TSP).

El diseño de PSP se basa en principios de planeación y de calidad, defendiendo la idea de que 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 introducen y eliminan de cada proyecto y al finalizar, medir los diferentes tamaños de los productos que llegan a producir. En fin, 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. Finalmente, deben analizar los resultados de cada trabajo y utilizar estos resultados para mejorar sus procesos personales.

2.2.2.3 Estimar duración de actividades con PSP

La exactitud, la estimación y el plan son influenciados en su mayoría por el conocimiento general que tienen los ingenieros respecto al trabajo que se hará. Por ello 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. Sin embargo, para los ingenieros que desarrollan software de manera individual, esta correlación tiene resultados generalmente altos.

El objetivo es realizar mejores estimaciones, cada vez más eficientes, precisas y que sirvan para tener un modelo de comparación con datos reales para que al final se generen los mejores resultados finales.

Por lo tanto, PSP comienza a estimar los tamaños de los productos que los ingenieros desarrollan personalmente. PSP se basa en el tamaño y en los datos de la productividad de cada ingeniero y con estos datos estima el tiempo requerido para hacer el trabajo. Teniendo en cuenta que el tamaño del programa también será expresado en cantidad de Líneas de Código.

Para gestionar el tiempo si estamos empleando los principios de PSP, se debe:

- hacer planes realistas
- intentar seguir el plan
- controlar el uso del tiempo
- determinar errores y cómo corregirlos

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 (Ver Anexo 14), 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.

PSP realiza las estimaciones, tanto del tamaño del programa como de los recursos del mismo, con un método que se creó para estos fines y que tiene por nombre *PROBE* (PROxy Based Estimating) por sus siglas en inglés, traducido al español se entiende como Estimación basada en la evaluación, método que describimos con anterioridad. Esta evaluación de aplica a todos los objetos que se encuentran en el diseño conceptual.

Cada vez que se realiza alguna modificación a cualquier programa, es necesario darle un buen seguimiento a cada cambio que se le realice desde el programa original. Estos datos se utilizan para determinar el volumen específico del producto desarrollado, también determinan la productividad de los desarrolladores y por último la calidad del producto. En aras de facilitar la recogida de estos datos, PSP provee de varias plantillas con un formato específico para ir recepcionando todos y cada uno de los cambios en el sistema. Muchas son estas plantillas, pero para llevar el control de la duración de las tareas de un proyecto de software hay 4 que tienen un papel primordial, sobre todo por la relación que hay entre cada una de ellas. Estos documentos están relacionados con actividades especificas definidas en PSP, pues para registrar las actividades que se van realizando se debe ir llenando el Registro del Control del Tiempo, de la misma forma que para llevar un control de la cantidad de Líneas de Código hasta un momento determinado debe hacerse en la plantilla que recoge los datos referentes al tamaño del Programa, con los datos de los documentos anteriores al finalizar la semana pueden resumirse los tiempos en el Resumen semanal de actividades y en dependencia de la información que se obtenga de estos registros semanales se obtendrá entonces un Presupuesto semanal del tiempo.

Como habíamos dicho anteriormente el propósito de este formato es el de registrar el tiempo empleado en cada fase del proyecto. Al mismo tiempo que todo su contenido será utilizados para complementar el resumen del plan del proyecto. Como información general lo único que se necesita es registrar el tiempo total que se emplea en el proyecto; este tiempo debe estar registrado en minutos y por último se debe procurar ser lo más preciso posible.

Para poder llevar un buen control del tiempo debemos medirlo en minutos, usar la ya descrita tabla de registro de tiempos, gestionar las interrupciones que tengamos y llevar el control de las tareas finalizadas.

Es muy importante que tengamos recogido todo el proceso de gestión de interrupciones son hechos inevitables y frecuentes de los cuales nos interesa controlar su frecuencia y duración.

Para poder hacer planificaciones debemos llegar a entender cómo aprovechamos el Tiempo. Los resúmenes semanales que obtenemos tras la unión de todos los registros de control del tiempo en la semana, permiten observar fácilmente nuestra dedicación a las distintas actividades y nos permiten también actualizar las medias, máximos, y mínimos acumulados.

Conclusiones

Se detallaron los fundamentos de cada uno de estos puntos de vista, destacando sus cimientos y sus peculiaridades. Sin lugar a dudas, en la actualidad siguen existiendo inconvenientes y limitaciones para las estimaciones, pero más allá de esto y a pesar de las diferencias en los criterios ambos autores han recorrido un importante camino, logrando la madurez necesaria en sus modelos para conseguir estimaciones de gran precisión.

Es más extendido el uso de COCOMO II y el resto de sus versiones.

Capítulo 3

Introducción

Tras el estudio y análisis realizado al tratamiento que dan Barry Boehm y Watts Humphrey en sus teorías para determinar la duración de un proyecto de software, podemos concluir que sí existen varios puntos de contacto. Cada uno de ellos nos da la posibilidad de establecer un nexo entre las dos teorías lo que quizás tras su aplicación, nos llevaría a obtener mejores resultados en estos procesos de estimación.

3.1 Líneas de Código

El primero de los contactos está dado por el vínculo que tienen ambos puntos de vista con este método para estimar tamaño en aplicaciones de software. En los dos casos se demostró la plena dependencia del tiempo de duración con el tamaño de los proyectos. Tanto COCOMO como COCOMO II, aun cuando utilicen otros métodos para estimar tales como Puntos de Función, Puntos Objeto, etc. las ecuaciones formuladas para determinar el esfuerzo dependen del tamaño, por ende para determinar el esfuerzo nominal en dichos modelos los puntos función no ajustados tienen que ser convertidos a líneas de código fuente considerando el lenguaje de implementación (assembler, lenguajes de alto nivel, lenguajes de cuarta generación, etc.) gracias al aporte hecho por Jones, al tabular la conversión.

Al aplicar la teoría de Humphrey no podemos, a menos que estemos trabajando con una persona experta en la realización de una actividad determinada, hacer una estimación temprana del tiempo de duración, pues se basa en la experiencia acumulada.

Sin embargo podríamos combinar ambos métodos: si estamos ante un experto en la realización de aplicaciones específicas, podemos estimar gracias a su basta experiencia la cantidad de líneas de código que tendría el sistema a implementar sin tener que aplicar ninguna otra técnica para estimar tamaño y tener de esta manera otro valor con el cual comparar el resultado de nuestras habituales formas de hacer estimaciones. De la misma forma podríamos estudiar el comportamiento del método de

PROBE, planteado por Humphrey introduciéndole la cantidad de líneas de código obtenidas tras la aplicación de algún otro método y comparar entonces resultados.

3.2 Factor de escala PMAT. Multiplicador de esfuerzo PREX

En COCOMO se utilizan distintos factores de escala para ponderar los valores de las variables vinculadas en aras de lograr que en sus valores estén implícitos la incidencia de todos los factores que puedan tener alguna relación directa e indirecta. Dos de ellos están directamente vinculados a los procesos de calidad de software que de manera tan exhaustiva recoge Humphrey en su propuesta.

PMAT, indica la Madurez del Proceso. El procedimiento que se sigue para determinar su valor es que se obtiene a través del Modelo de Madurez de Capacidad del Instituto de Ingeniería del Software (CMM). 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 1 CMM (Mitad inferior)
- Nivel 1 CMM (Mitad superior)
- Nivel 2 CMM
- Nivel 3 CMM
- Nivel 4 CMM
- Nivel 5 CMM

Áreas de Proceso Principales

La segunda está organizada en base a 18 áreas de proceso principales (KPA´s) en el modelo de Madurez de Capacidad SEI. El procedimiento para determinar en este caso el PMAT es decidir el porcentaje de conformidad para cada uno de los KPA´s. Si el proyecto ha sufrido una valoración CMM reciente entonces se usa el porcentaje de conformidad para la KPA global (basada en datos de valoración de la conformidad

práctica principal). Si no se ha hecho una valoración entonces se usan los niveles de conformidad para las metas de los KPA´s para poner el nivel de conformidad. El nivel basado en meta (objetivo) de conformidad se determina haciendo una media basada en juicio de las metas de cada área de proceso principal.

Por su parte PREX, hace referencia a la Experiencia Personal. Es uno de los parámetros de coste del Diseño Temprano que combina los tres parámetros de coste del modelo Post-Arquitectura (experiencia en la aplicación (AEXP), experiencia en la plataforma (PEXP), y experiencia en las herramientas y lenguajes (LTEX)). Cada uno de estos indicadores recibe influencia directa de la aplicación de los puntos de vista defendidos por Humphrey.

Podría hacerse un estudio mas profundo además teniendo en cuenta la incidencia directa que ejerce sobre otros multiplicadores de esfuerzos para delimitar pautas de integración entre estos dos métodos por la importancia que se le debe conferir a obtener productos con mayor calidad partiendo de calibraciones realistas en un entorno en particular como es la Universidad.

3.3 Reutilización de código

De la misma forma que al llevar el control de todas las actividades que realizamos al aplicar PSP tenemos en cuenta la posibilidad de reutilizar código y que esto nos implica una reducción en el tiempo a emplear para resolver una tarea en específico, COCOMO también da un tratamiento a esta posibilidad. Lo hace usando un modelo de estimación no lineal para calcular las LDCF equivalentes a nuevo desarrollo (ESLOC) a través de:

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

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

donde

ASLOC = cantidad de LDCF adaptadas de software existente,

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, y

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

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

Siendo:

- DM = % de modificación del diseño,

- CM = % de modificación del código,

- IM = % del esfuerzo de integración original requerido para integrar el software reutilizado.

En este punto los valores obtenidos por PSP tendrían una incidencia directa sobre las variables ASLOC, AA y UNFM.

Conclusiones

Se muestran tres posibles variantes de integración de estos métodos en aras de obtener resultados más precisos al estimar la duración de un proyecto de software. Hay un aspecto que no puede dejar de tenerse en cuenta y es que el estrecho vínculo entre estas dos teorías tiene su base fundamental en la calibración de las variables que se utilizan para ajustar los valores obtenidos según los distintos factores que inciden en el resultado.

Otras vías de integración pueden existir si se redefinen los parámetros a calibrar en función de los intereses de una organización específica, ya que ha de tenerse en cuenta que las medidas obtenidas, fundamentalmente al aplicar COCOMO II son el resultado de haber trabajado con factores inherentes al campo de acción de Boehm.

Conclusiones

Tras haber realizado un estudio exhaustivo de los procesos de estimación de software como parte de la planificación de la gestión de proyectos, haberse revisado varios métodos y analizado diferentes enfoques y estudios, se puede concluir que, la mayoría de los especialistas se rigen y sugieren el proceso ideado por Barry Boehm, aunque se reconoce la importancia de aplicar la teoría de Humphrey en aras de obtener sistemas con calidad e ir formando mejores especialistas.

Por otra parte son las más legendarias: los Puntos de Función y las Líneas de Código las técnicas más utilizadas aún para estimar, a pesar de estar cogiendo auge otras como Puntos de Características, MK II, Puntos de Objeto y Puntos de Casos de Uso que están ideadas para garantizar la estimación en los actuales modelos de desarrollo de software.

No se puede dejar de mencionar que sí existen elementos que garantizan la integración entre las dos teorías analizadas: la de Barry Boehm y la de Watts Humphrey, y que además puede establecerse dicha integración, pues existen algunos factores de escala y multiplicadores de esfuerzo utilizados en el método de Boehm sobre los que el componente fundamental de PSP, la experiencia adquirida tras estimar continuamente basados en datos propios recogidos en un histórico, ejerce una influencia directa, que puede ser empleada además en las ecuaciones planteadas por Boehm para valorar la reutilización de código. La palabra de orden en este caso es calibración, ya que al evaluar varios de los parámetros expuestos al aplicar COCOMO II se abren las puertas para hacer uso de los resultados obtenidos por el método de Humphrey.

De la misma forma pueden establecerse nexos en la estimación del esfuerzo al intercambiar la forma de obtener el tamaño de la aplicación, dada en LOC.

Recomendaciones

Las recomendaciones van dirigidas a demostrar la validez de los planteamientos expuestos, así como su vinculación para futuros estudios con la Universidad de Ciencias Informáticas:

Se recomienda evaluar en un proyecto real cada uno de los posibles elementos a ser integrados entre ambos métodos a fin de validar sus resultados.

Otros trabajos a los que podría dar lugar el estudio realizado conllevarían a proponer parámetros específicos inherentes a las características de la Universidad de las Ciencias Informáticas para aplicar la teoría de Boehm tratando de ajustar la función a un conjunto de datos locales en aras de obtener una calibración más precisa, introduciendo, siempre que sea posible, resultados obtenidos tras el uso de Humphrey.

Hacer extensivo el estudio de los procesos de estimación vinculados al resto de los elementos de esta área de conocimiento del PMBOK: costos y recursos en busca de integraciones entre las técnicas y métodos existentes.

Referencias Bibliográficas

1. Navarro, A., Planificación de Proyectos de Software. p. 110.
2. Palacio, J., Origen de la gestión de proyectos. 2006.
3. Montesa, J.O., El proyecto: una forma de organizar el trabajo. 2000.
4. PMI, Project Management Body of Knowledge. 2004.
5. Soláns, M., PMI, una referencia a la gestión de proyectos. 2005.
6. Pressman, R.S., Ingeniería de Software. Un enfoque práctico. 5ta edición ed. 2002.
7. Argentina, C.P. (2006) Áreas de proceso PMBOK 2004. Volume,
8. Proyectics, "Los cambios en las Áreas de conociemiento se enfocan en nuevos procesos y un mejor alineamiento." 2004.
9. Cerrillo, D., Estimación del Software. 2000.
10. Gibson, R., Managing Computer Projects. Avoiding the Pitfalls. 1992.
11. O´Conell, F., How to Run Successful projects. 1994.
12. King, D., Project Management Made Simple. A guide to successful of computer Systems projects. 1992.
13. De la Fuente Moya, A., COCOMO v2. 2000.
14. Moreno Capuchino, A.M., Estimación de proyectos de software.
15. Boehm, B., COCOMO II Overview. 2000.
16. Humphrey, W.S., "Introducción al Proceso de Software Personal" 2001.

Bibliografía

"Administración y Gestión de Proyectos de Software" Dpto. Cs. e Ingeniería de la Computación Universidad Nacional del Sur. 2005

Barato José. "Representación Formal PMBOK Guide". 2004

Boehm, Barry. "Some Future Trends and Implications for Systems and Software Engineering Processes." 2006

Boehm, Barry; Jain, Apurva. "An Initial Theory of Value-Based Software Engineering". Febrero 2005

Boehm, Barry; Lane, Jo Ann. Software-Intensive Systems of Systems. Mayo 2006.

Boehm, Barry. "Some Future Trends and Implications for Systems and Software Engineering Processes", Julio 2005.

Boehm, Barry; Colbert, Ed; Wu, Dan; Chen, Yue; "Cost Estimation for Secure Software & Systems", 2005

Boehm, Barry; Valerdi, Ricardo; Lane, Jo Ann; Brown, A. Winsor. "COCOMO Suite Methodology and Evolution"

Cerrillo, David. "Planificación y Gestión de Sistemas de Información. Estimación del software". Mayo 1999.

"Certificación PMP Project Management Professional". Instituto Argentino de Administración de Proyectos (IAAP). Project Management Institute. 2005

Cao, José Ignacio. "Principios para un método de estimación de proyectos de software basado en los escenarios principales." 2006

Cuadrado Gallego, J.; Sicilia, Miguel Ángel; Garre Rubio, Miguel. "Segmentación Recursiva de Proyectos Software para la Estimación del Esfuerzo de Desarrollo Software". Dpto de Ciencias de la Computación, Universidad de Alcalá, España. 2004
De la Fuente Moya, Antonio. "C O C O M O v 2. Modelo de Estimación de Costes para proyectos software".Escuela Superior de Informática. Universidad de Castilla-La Mancha. Mayo 2000.

Dolado Cosín, J. Javier, Luis Fernández Sanz, "Medición para la gestión en la Ingeniería del Software", Ra-Ma, 2000

"Estimación de proyectos de software. COCOMO 81." Marzo 2005.

Ferreira, Mateus; García, Félix; Ruiz, Francisco; Bertoa, Manuel F.; Calero, Coral; Vallecillo, Antonio; Piattini, Mario; Mora, Beatriz. "Medición del Software. Ontología y Metamodelo". Noviembre 2006

"Gestión de Proyectos de Software". Universidad Rey Juan Carlos. 2002

Global Tecnologies & Business Consulting. "Project Management para Ejecutivos". Abril 2006

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

Gómez Sánchez, Rubén. "Impacto del PMBOK". Julio 2002

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

Grupo de Investigacion en Gestión y Evaluación de Programas y Proyectos. "Reseña del PMBOK Guide- Una Guía de los Fundamentos de la Dirección de Proyectos". Universidad del Valle. Junio 2005

J. Navón. "Estimaciones." 2005

M.E.Manso. "Calidad de Software. Teoría de Medición". 2002

Moreno S.-Capuchino, Ana Mª. "Estimación de Proyectos Software".

Ogáyar, Diego Jódar; Casals Castells, Anna. "Project Manager Handbook. Manual del Jefe de Proyectos". 2004

Ordieres Meré, J.B; Torralba Martínez, J. Mª; Chiner Dasi, M. "Estimación del presupuesto del proyecto de software". 2004

Pow-Sang Portillo, José Antonio; Imbert Paredes, Ricardo. "Estimación y Planificación de Proyectos Software con Ciclo de Vida Iterativo-Incremental y empleo de Casos de Uso". 2004.

Pressman, Roger. "Ingeniería de Software, un enfoque práctico."

Projectics. "Tercera Edición de la Guía para el PMBOK®. Los Cambios en las Áreas de Conocimiento se enfocan en Nuevos Procesos y en un mejor Alineamiento". Octubre 2004.

Raffo David, Harrison Warren, Vandeville Joseph. "Coordinating Models and Metrics to Manage Software Projects". 2000

Ruiz-Bertol, Fran J.; Dolado, Javier. "Gestión Activa de Eventos en Proyectos Software" Dpto. Lenguajes y Sistemas Informáticos, Universidad del País Vasco, España. 2004

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

Soláns, Manuel. "PMI, una referencia en la Gestión de Proyectos." Febrero 2005
Palacio, Juan. "Origen de la gestión de proyectos". 2006.

Glosario de Términos

Adaptabilidad. Facilidad con la que un sistema o un componente puede modificarse para corregir errores, mejorar su rendimiento u otros atributos, o adaptarse a cambios del entorno. Ver también: **escalabilidad**.

Aplicación de software. Software diseñado para satisfacer las necesidades de un usuario. Contrasta con: **software de soporte**; **software de sistema**.

Ciclo de vida. Período de tiempo que comienza con la concepción del producto de software y termina cuando el producto esta disponible para su uso.

Normalmente, el ciclo de vida del software incluye las fases de concepto, requisitos, diseño, implementación, prueba, instalación, verificación, validación, operación y mantenimiento, y, en ocasiones, retirada. Nota: Esta fases pueden superponerse o realizarse iterativamente.

CMM. Siglas de "Capability Maturity Model", modelo desarrollado por SEI (Software Engineering Institute) en 1990, para la evaluación y mejora de los procesos. El primer modelo desarrollado para evaluar y mejorar los procesos fue el SW-CMM, por lo que muchas veces se hace referencia a el coloquialmente como "CMM". En la actualidad los modelos de evaluación y mejora desarrollados y mantenidos por SEI son: P-CMM (People Capability Maturity Model), SA-CMM (Software Acquisition Capability Maturity Model).

Con la aparición en 2001 de los modelos *CMMI*, SEI ha dejado de mantener desde finales de 2004 los siguientes modelos CMM, por haberse integrado en los nuevos CMMI: SW-CMM (Capability Maturity Model for Software), SE-CMM (Systems Engineering Capability Maturity Model), IPD-CMM (Integrated Product Development Maturity Model).

CMMI Siglas de "Capability Maturity Model Integration", modelos desarrollados por SEI que integran varias disciplinas: Desarrollo de software, Ingeniería de sistemas, Integración de productos y procesos de desarrollo.

COCOMO. (Construvtive Cost Model) Modelo constructivo de costes, desarrollado por B.W. Boehm en 1981, y expuesto en su libro "Software Engineering Economics". Es una jerarquía de modelos de estimación de costes que incluye los sub-modelos: básico, intermedio y detallado.

Compatibilidad. (1)Preparación de dos o más componentes o sistemas para llevar a cabo sus funciones mientras comparten el mismo entorno de hardware o software. (2) Capacidad de dos o más sistemas o componentes para intercambiar información.

Componente. Una de las partes que forman un sistema. Un componente puede ser hardware, software, y puede a su vez subdividirse en otros componentes.

CPM. (Critical Path Method) Método para el control y la optimización de los costes de operación mediante la planificación adecuada de las actividades que componen un proyecto. Fue desarrollado en 1957 en los Estados Unidos por un centro de investigación de operaciones para la firma Dupont y Remington Rand. Actualmente se utilizan sus principios en combinación con los del método *PERT* en lo que se conoce como *PERT/CPM*

Descripción del sistema. Documento orientado al cliente que describe las características del sistema desde el punto de vista del usuario final. El documento se utiliza para coordinar conjuntamente los objetivos del sistema del usuario, cliente, desarrollador e intermediarios.

Diseño. (1) Proceso de definición de la arquitectura, componentes, interfaces y otras características de un sistema o de un componente. (2) El resultado de este proceso.

Diseño de arquitectura. (1) Proceso que define una colección de componentes de software y hardware junto con sus interfaces, para definir el marco de desarrollo de un sistema. Ver también: **diseño funcional**. (2) El resultado del proceso (1).

Diseño detallado.(1) Proceso de definición y ampliación del diseño preliminar de un sistema o de un componente hasta un grado de detalle suficiente para llevar a cabo la implementación. (2) El resultado del proceso (1)

Diseño preliminar (1) Proceso de análisis de las alternativas de diseño y definición de la arquitectura, componentes, interfaces, estimación de tiempo y tamaño de un sistema o de un componente. Ver también: **Diseño detallado**. (2) El resultado del proceso (1).

Disponibilidad. El grado con el que se mide la accesibilidad de un sistema o de un componente cuando es necesario su uso. Suele expresarse en términos de probabilidad. Ver también: **tolerancia a errores**, **tolerancia a fallos**, **robustez**.

Escalabilidad. Facilidad con la que un sistema o un componente puede modificarse para
aumentar su capacidad funcional o de almacenamiento. Ver también: **adaptabilidad**.

Especificación de requisitos de software. Documentación de requisitos fundamentales (necesarios, esenciales e indispensables) de funcionalidades, rendimiento, restricciones y atributos del software, y sus interfaces externas.

Estimación por analogía. Modelo de estimación de costes y recursos, basado en la comparación con proyectos de ámbitos y características similares, de los que se conocen sus costes reales por haberse terminado.

Flexibilidad. Facilidad con la que un sistema o un componente puede modificarse para ser empleado con aplicaciones o en entornos distintos para los que fue construido.

Gestión de configuración. Disciplina que aplica la dirección y supervisión técnica y administrativa para: identificar y documentar las características funcionales y fisicas de un elemento de configuración, controlar cambios, registrar cambios procesados, registrar el estado de la implementación, informar y verificar la conformidad con los requisitos especificados.

Gestión de procesos. Dirección, control y coordinación del trabajo realizado para desarrollar o producir un servicio.

Implementación. (1) Proceso de transformación de un diseño en componentes de hardware, software o de ambos. Ver también: **codificación**. (2) El resultado del proceso (1).

Ingeniería del software. (1) Aplicación de procesos sistemáticos y disciplinados para el desarrollo, operación y mantenimiento de software. (2) El estudio de la aplicación (1).

Interfaz. (1) Característica común en la información enviada. (2) Componente de hardware o software que conecta dos o más componentes con el propósito
de transmitir información entre ellos. (3) Conexión de dos o más componentes con el propósito de transmitir información entre ellos. (4) Empleado en la conexión (2)

Interfaz de usuario. Interfaz que permite la comunicación entre un usuario y un sistema, o los componentes de un sistema.

Línea de base. Conjunto de elementos de configuración, formalmente revisados y aprobados (para su uso interno o para entregar al cliente), que constituyen la base para el desarrollo posterior, y que sólo puede modificarse a través de procedimientos de cambio formales.

Modelo de ciclo de vida. Representación del ciclo de vida del software.

Nivel de integridad: Grado de daño que puede producir un fallo en un sistema. El estándar IEEE 1012-1998 define cuatro niveles de integridad para sistemas de software siendo el grado 1 el propio de sistemas cuyo fallo produce daños de escasa relevancia, y el 4 el que implica pérdidas de vida o graves pérdidas económicas o sociales.

OO. (Orientación por Objetos) Enfoque para el desarrollo de sistemas de software que representa el dominio de aplicación de forma natural y directa basándose en los objetos que se implican en dicho dominio. Emplea diversos métodos para representar de forma abstracta los objetos, definiendo su estructura, comportamiento, agrupaciones, estados, etc. Las estrategias de orientación por objetos han desarrollado metodologías tanto para requisitos, como para análisis, diseño y programación. *OOD* (Diseño orientado por objetos), *OOP* (Programación orientada por objetos).

OOP (Object-Oriented Programming) Programación orientada por objetos. Método de implementación de los programas que los organiza como grupos cooperativos de objetos, cada uno de los cuales representa instancias de una clase, que a su vez forman parte de una jerarquía a través de relaciones de herencia. v. *OO*.

PERT. (Program Evaluation and Review Technique) Método para el control de los tiempos de ejecución de diversas actividades integrantes de proyectos. Fue desarrollado en 1957 por la armada de los Estados Unidos. Actualmente se utilizan sus principios en combinación con los del método *CPM* en lo que se conoce como *PERT/CPM*

PERT/CPM. Método para el control de la ejecución de proyectos. Combina principios de los métodos PERT y CPM. Su desarrollo en un proyecto resulta útil para: conocer la

probabilidad de cumplimiento de fechas, identificar las actividades con mayor potencial para retrasar el proyecto y evaluar las consecuencias de una desviación.

Plan de proyecto. Documento que describe el enfoque técnico y de gestión que seguirá un proyecto. Generalmente, el plan describe el trabajo a realizar, los recursos necesarios, los métodos a utilizar, los procesos a seguir, los programas a cumplir y la forma en la que se organiza el proyecto.

Producto de software. (1) Conjunto de programas, procedimiento y opcionalmente documentación asociada que se entrega al usuario como resultado. (2) Uno de los elementos de (1).

Programa de ordenador. Combinación de instrucciones informáticas y definiciones de datos que permiten a un ordenador llevar a cabo tareas de control o de manipulación de información. Ver también **software**.

Prototipado. Técnica de desarrollo consistente en la construcción de una versión preliminar de parte o de todo un sistema, para evaluar su viabilidad, funcionalidad, tiempos de respuesta, etc.

Prototipo. Versión preliminar de un sistema que sirve de modelo para fases posteriores.

Puntos de función. Modelo de estimación basado en la perspectiva de la funcionalidad, sin contemplar detalles de la codificación. Se basa en una combinación de características del sistema de software: entradas del usuario, salidas (presentadas) al usuario, consultas del usuario, archivos usados por el sistema e interfaces externos.

Rational Unified Process (RUP). Proceso de Ingeniería del Software que proporciona un enfoque disciplinado para asignar tareas y responsabilidades en las organizaciones de desarrollo de software. Se trata de un proceso integrado en un producto, desarrollado y mantenido por Racional Software, e integrado en su conjunto de herramientas de desarrollo. Se encuentra disponible a través de IBM.

Requisito. (1) Condición o facultad que necesita un usuario para resolver un problema. (2) Condición o facultad que debe poseer un sistema o un componente de un sistema para satisfacer una especificación, estándar, condición de contrato u otra formalidad impuesta documentalmente.

(3) Documento que recoge (1) o (2).

Robustez. El grado de capacidad que presenta un sistema o un componente para funcionar correctamente frente a entradas de información erróneas, o carga de trabajo elevada. Ver también: **tolerancia a errores**; **tolerancia a fallos**.

RUP. v. Racional Unified Process

SEI. (Software Engineering Institute) Fundación federal norteamericana para la investigación y desarrollo, cofinanciada por el Departamento de Defensa de los Estados Unidos y dependiente de la Universidad Carnegie Mellon.

Sistema. Conjunto de procesos, hardware, software, instalaciones y personas necesarios para realizar un trabajo o cumplir un objetivo.

Sistema de software. Conjunto de programas de ordenador, procedimientos y opcionalmente la documentación y datos asociados, necesarios para el funcionamiento de un sistema.

SLIM. (Software Lifecycle Management) Metodologías para estimaciones de duración, costes, control de proyectos y gestión de métricas. Desarrolladas por la comercial QSM.

Software. Los programas de ordenador, procedimientos, y opcionalmente la documentación y los datos asociados que forman parte de un sistema.

Software de sistema. Software diseñado para facilitar o permitir la operación y el mantenimiento de un sistema informático; por ejemplo los sistemas operativos. Contrasta con **aplicación de software**.

Subsistema. *Sistema* subordinado a otro mayor.

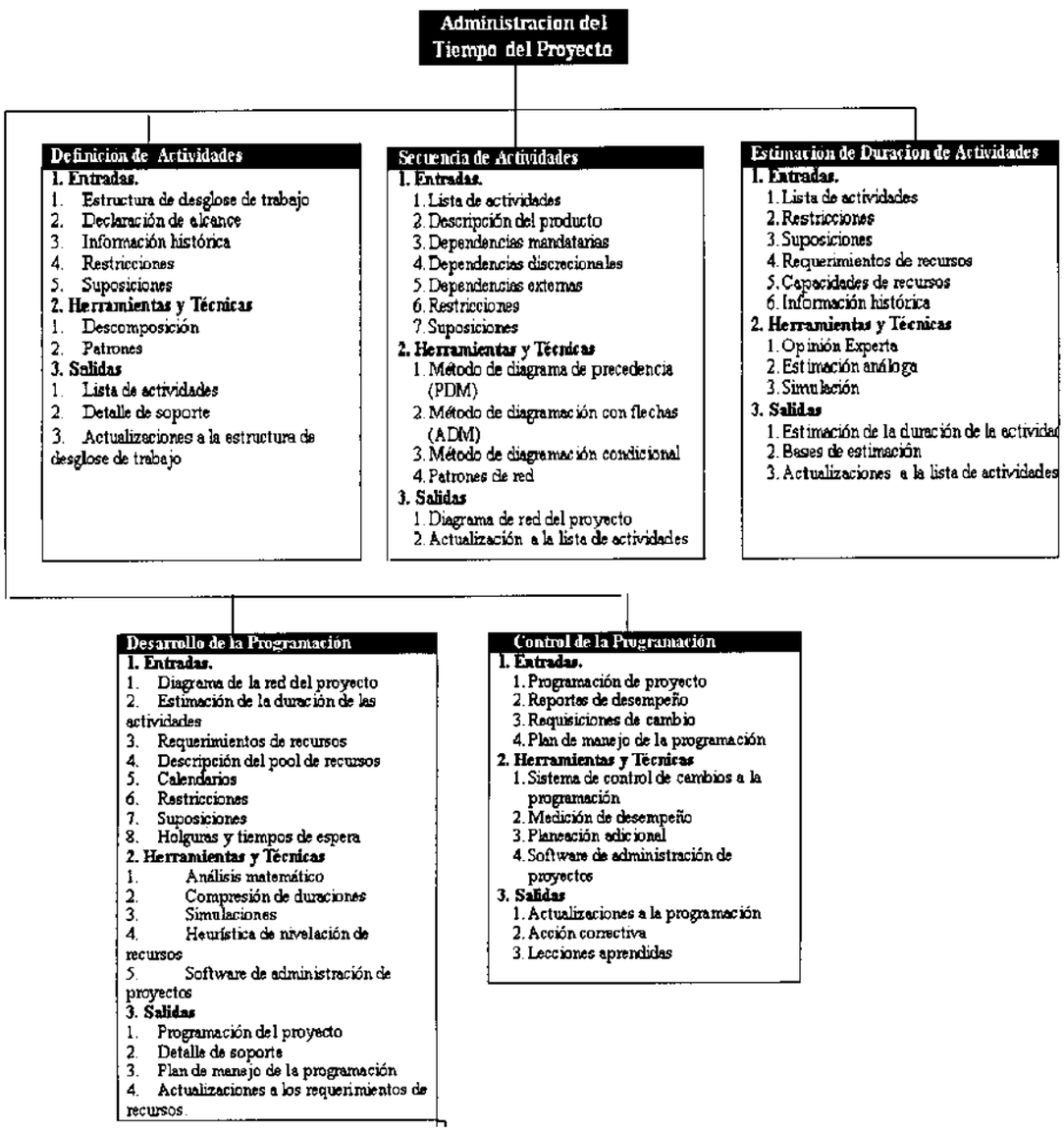
Tolerancia a errores. Preparación de un sistema o de un componente para continuar su estado normal de operación, a pesar de la presencia de entradas erróneas. Ver también: **tolerancia a fallos, robustez.**

Tolerancia a fallos. Preparación de un sistema o de un componente para continuar su estado normal de operación, a pesar de la aparición de errores de hardware o de software. Ver también: **tolerancia a errores**, **robustez**.

Anexos.

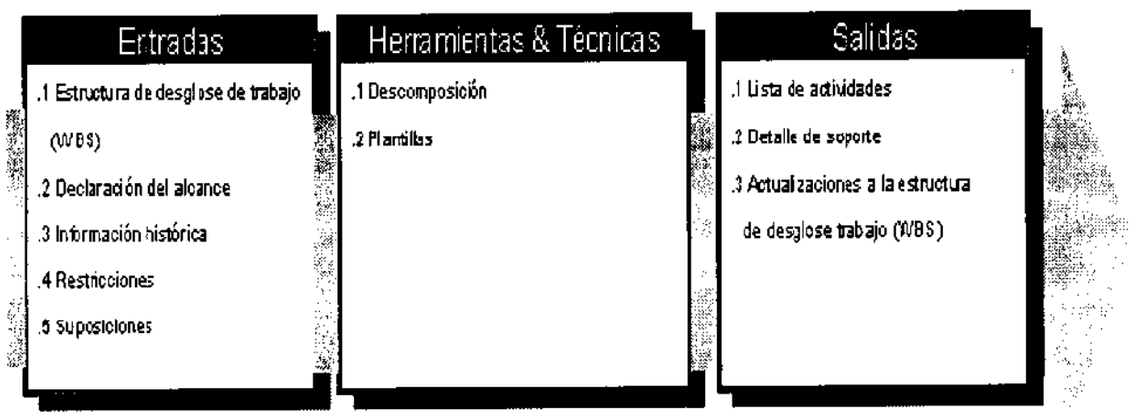
Anexo 1: Vista general de la Administración de tiempo de Proyecto según el PMBOK.

Vista General de la Administracion del Tiempo del Proyecto

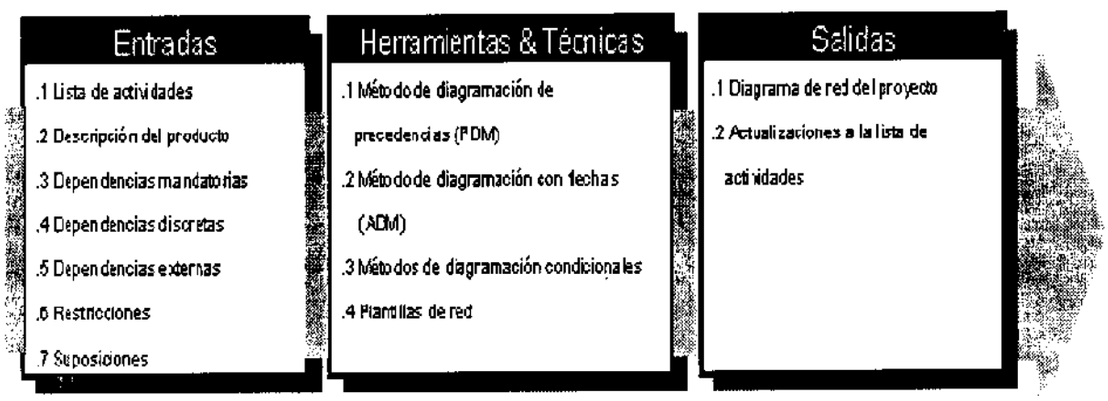


Anexo 2: Entradas, herramientas y técnicas sugeridas en el PMBOK para obtener las salidas sugeridas por el PMI.

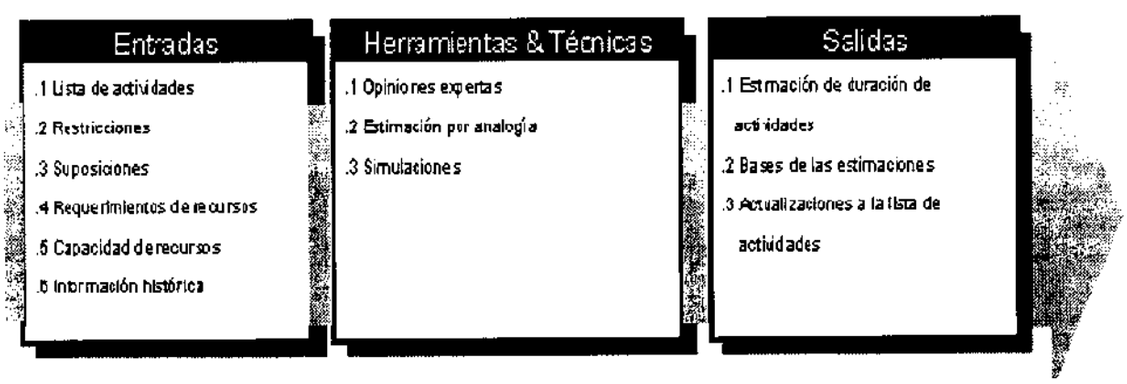
Anexo 2.1: Elementos de Definición de Actividades.



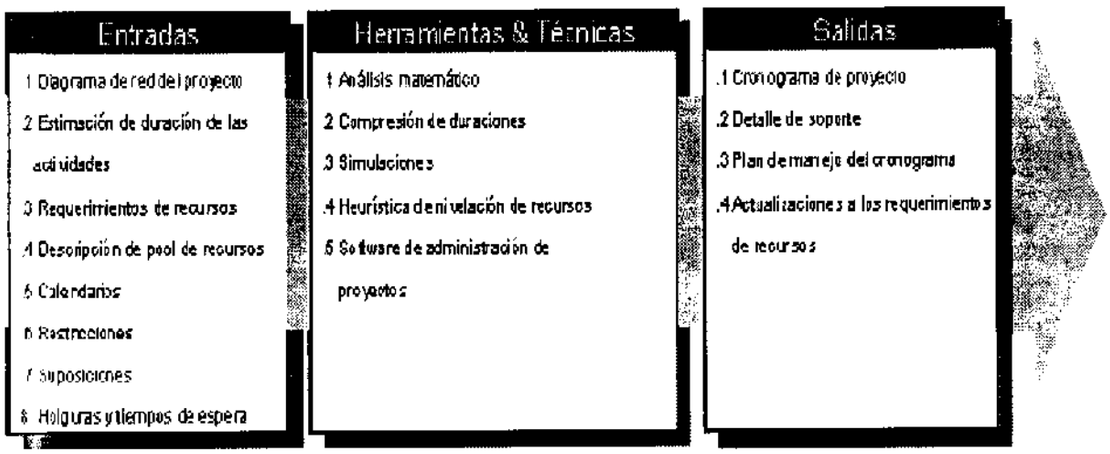
Anexo 2.2: Elementos de Secuencia de Actividades.



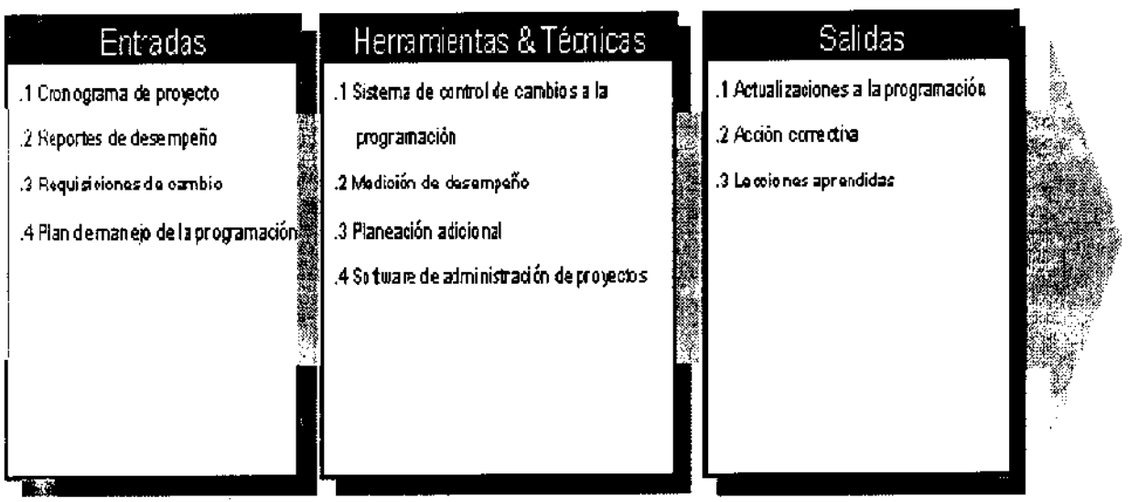
Anexo 2.3: Elementos de Estimación de la duración de Actividades.



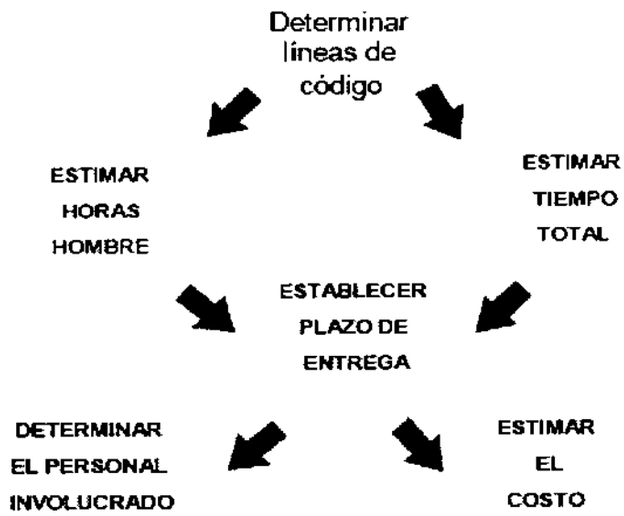
Anexo 2.4: Elementos de Desarrollo de la Programación.



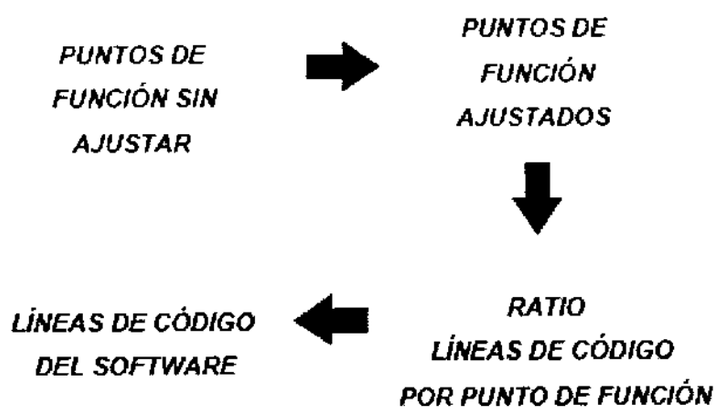
Anexo 2.5: Elementos de Control de la Programación.



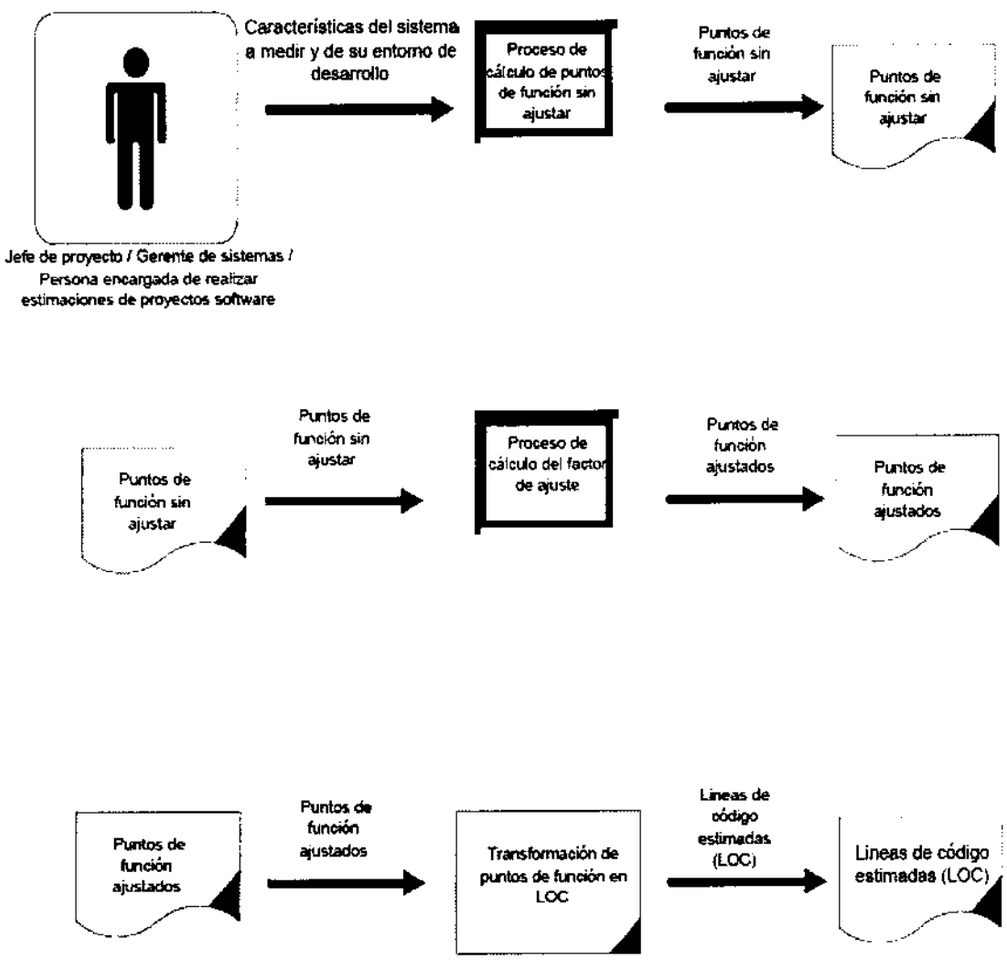
Anexo 3.1: Estimación de proyectos por el Método COCOMO



Anexo 3.2: Estimación de proyectos por el Método Puntos de Función



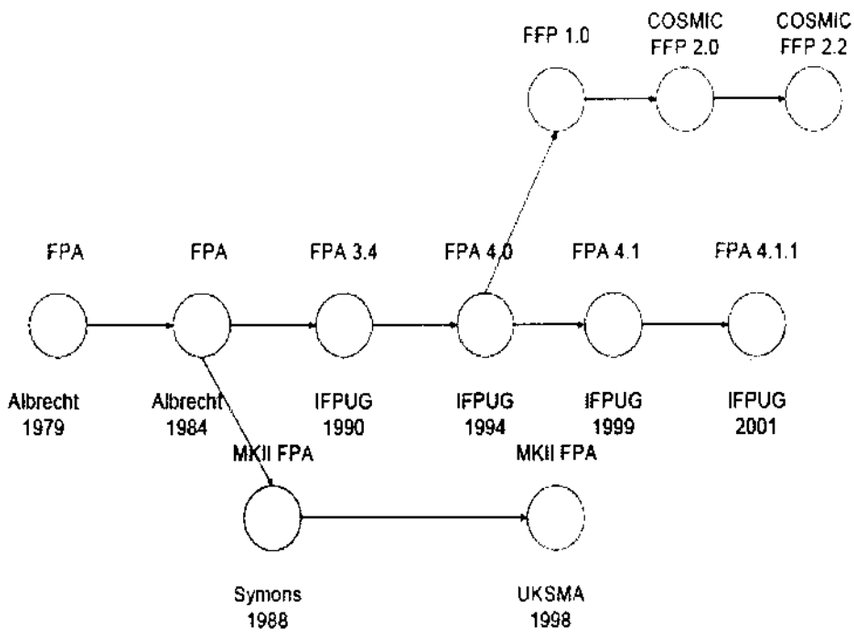
Anexo 4: Proceso de cálculo y obtención de puntos de función y su posible transformación a líneas de código



Anexo 5: Conversión de Puntos de Función a Líneas de Código

Lenguaje (o entorno de programación)	LDC/PF
4GL	40
Ada 83	71
Ada 95	49
APL	32
BASIC - compilado	91
BASIC - interpretado	128
BASIC ANSI/Quick/Turbo	64
C	128
C++	29
Clipper	19
Cobol ANSI 85	91
Delphi 1	29
Ensamblador	320
Ensamblador (Macro)	213
Forth	64
Fortran 77	105
FoxPro 2.5	34
Generador de Informes	80
Hoja de Cálculo	6
Java	53
Modula 2	80
Oracle	40
Oracle 2000	23
Paradox	36
Pascal	91
Pascal Turbo 5	49
Power Builder	16
Prolog	64
Visual Basic 3	32
Visual C++	34
Visual Cobol	20

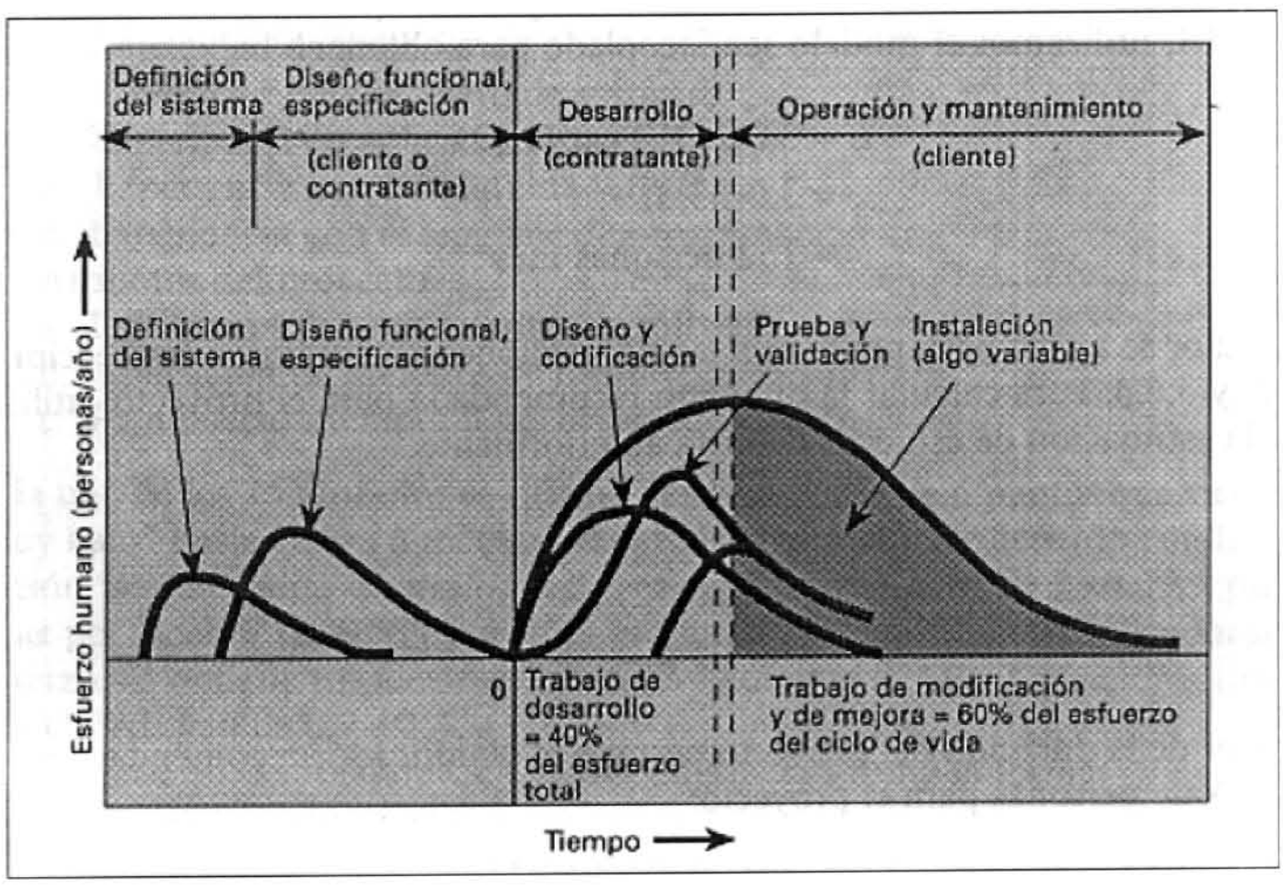
Anexo 6: Evolución de los métodos de estimación basados en Puntos de Función



Anexo 7: Comparativa entre algunos métodos de estimación

MÉTODO	VENTAJAS	DESVENTAJAS
Algorítmico	Objetivo, repetible, eficiente, fórmula analizable Bueno para el análisis de sensibilidad Objetivamente calibrado para la experiencia	Entradas subjetivas No se adapta a circunstancias excepcionales Asume la historia para predecir nuevas aplicaciones
Juicio de expertos	Asegura la representatividad de las interacciones y circunstancias excepcionales que pueden afectar el juicio	No es mejor que los participantes Sesgado, recuerdos incompletos Representativo de la experiencia
Analogía	Basado en la experiencia	No es mejor que los participantes Sesgado, recuerdos incompletos Representativo de la experiencia
Ley de Parkinson	Fuertemente relacionado con la experiencia	Refuerza malas prácticas
Precio para ganar	A menudo gana los contratos	Produce grandes excesos Uso poco ético de información falsa
Top-Donw	Se enfoca a nivel del sistema Uso eficiente de los recursos	Menos detallado y estable que otros métodos Pasa por alto la complejidad técnica
Bottom-Up	Base más detallada Más estable que el top-down Soportado por compromisos individuales dados por las personas que estiman su propio trabajo	Puede pasar por alto las complejidades y los costos a nivel del sistema Requiere más esfuerzo que la mayoría de los otros métodos
Puntos de Función	Objetivo, repetible, entrada objetivas	Basado en la historia Debe ser calibrado Se enfoca en las externalidades de la aplicación

Anexo 8: Curvas de Rayleigh para Modelo SLIM



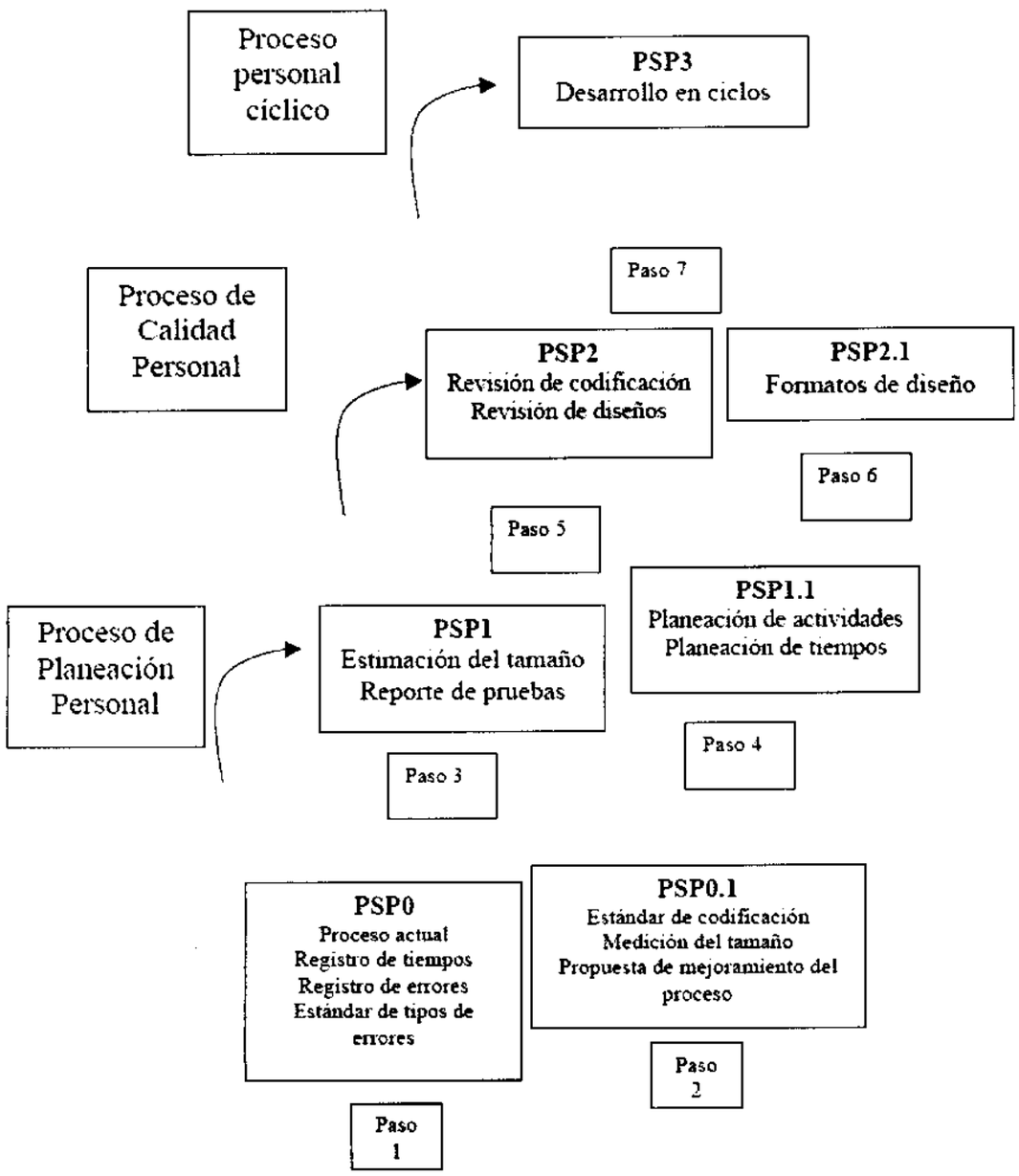
Anexo 9: Comparativa entre COCOMO 81 y COCOMO II

	COCOMO 81	COCOMO II
ESTRUCTURA DEL MODELO	Un modelo único que asume que se ha comenzado con unos requisitos asignados para el software	Tres modelos que asumen que se progresa a lo largo de un desarrollo de tipo espiral para consolidar los requisitos y la arquitectura, y reducir el riesgo.
FORMA MATEMÁTICA DE LA ECUACIÓN DEL ESFUERZO	Esfuerzo = A (q) $(SIZE)^{exponente}$	Esfuerzo = A (q) $(SIZE)^{exponente}$
EXPONENTE	Constante fija seleccionada como una función de modo: ▪ Orgánico = 1.05 ▪ Semi-libre = 1.12 ▪ Rígido = 1.20	Variable establecida en función de una medida de cinco factores de escala: ▪ PREC Precedencia ▪ FLEX Flexibilidad de desarrollo ▪ RESL Resolución de Arquitectura / Riesgos ▪ TEAM Cohesión del equipo ▪ PMAT Madurez del proceso
MEDIDA	Líneas de código fuente (con extensiones para puntos de función)	Puntos objeto, Puntos de función ó líneas de código fuente.
DRIVERS DE COSTE (c_i)	15 drivers, cada uno de los cuales debe ser estimado: ▪ RELY Fiabilidad ▪ DATA Tamaño Base de datos ▪ CPLX Complejidad ▪ TIME Restricción tiempo de ejecucion ▪ STOR Restricción de almacenamiento principal ▪ VIRT Volatilidad máquina virtual ▪ TURN Tiempo de respuesta ▪ ACAP Capacidad del analista ▪ PCAP Capacidad programador ▪ AEXP Experiencia aplicaciones ▪ VEXP Experiencia máquina virtual ▪ LEXP Experiencia lenguaje ▪ TOOL Uso de herramientas software ▪ MODP Uso de Técnicas modernas de programación ▪ SCED Planificación requerida	17 drivers, cada uno de los cuales debe ser estimado: ▪ RELY Fiabilidad ▪ DATA Tamaño Base de datos ▪ CPLX Complejidad ▪ RUSE Reutilización requerida ▪ DOCU Documentacion ▪ TIME Restricción tiempo de ejecución ▪ STOR Restricción de almacenamiento principal ▪ PVOL Volatilidad plataforma ▪ ACAP Capacidad del analista ▪ PCAP Capacidad programador ▪ AEXP Experiencia aplicaciones ▪ PEXP Experiencia plataforma ▪ LTEX Experiencia lenguaje y herramienta ▪ PCON Continuidad del personal ▪ TOOL Uso de herramientas software ▪ SITE Desarrollo Multi-lugar ▪ SCED Planificación requerida
OTRAS DIFERENCIAS DEL MODELO	Modelo basado en: ▪ Fórmula de reutilización lineal ▪ Asunción de requisitos razonablemente estables	Tiene muchas otras mejoras que incluyen: ▪ Fórmula de reutilización No-lineal ▪ Modelo de reutilización que considera esfuerzo necesario para entender y asimilar ▪ Medidas de rotura que se usan para abordar la volatilidad de requisitos ▪ Caracteristicas de autocalibracion

Anexo 10: Factores de Escala utilizados en COCOMO II

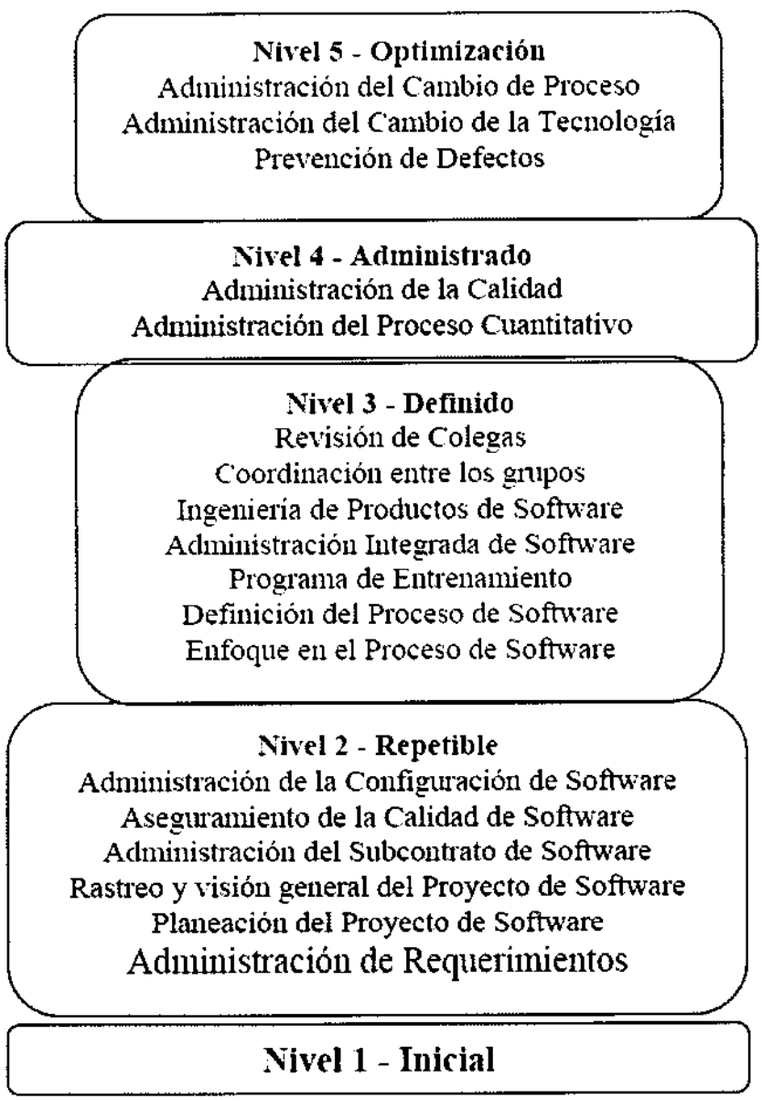
Factor de Escala Wj	Muy Bajo	Bajo	Normal	Alto	Muy Alto	Extra
Precedencia **PREC**	Completamente sin precedentes	Ampliamente sin precedentes	Algún precedente	Generalmente Familiar	Ampliamente Familiar	Completamente Familiar
Flexibilidad en el desarrollo **FLEX**	Rigurosa	Relajación Ocasional	Alguna Relajación	Conformidad en general	Alguna Conformidad	Metas generales
Arquitectura/ Resolución de riesgo **RESL**	Poca (20%)	Alguna (40%)	Siempre (60%)	Generalmente 75%)	Principalmente (90%)	Completo (100%)
Cohesión de equipo **TEAM**	Interacciones dificiles	Interacciones con alguna dificultad	Interacciones básicamente cooperativas	Ampliamente Cooperativas	Altamente Cooperativas	Interacciones Sin Fisuras
Madurez del proceso **PMAT**	Desarrollado más adelante					

Anexo 11: Evolución del Proceso Personal de Software

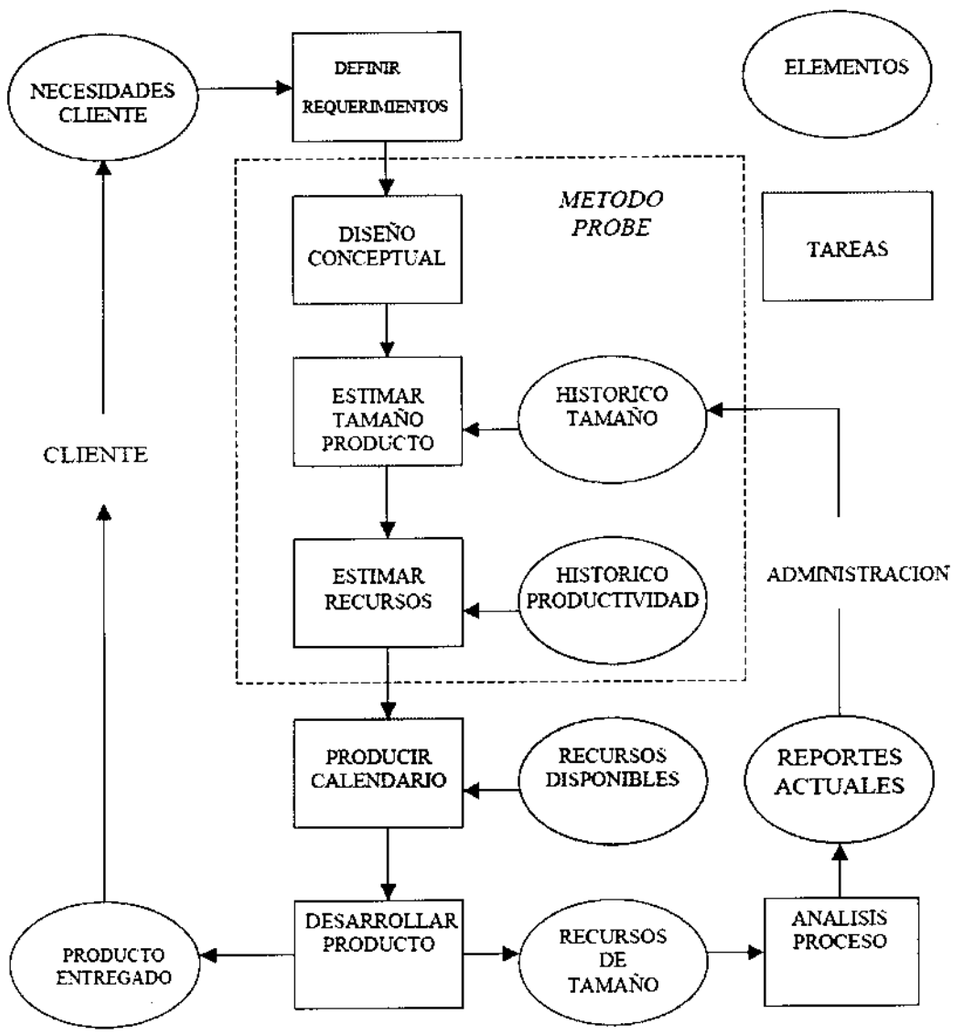


Anexo 12: Vínculos PSP con CMM

Elementos del PSP en el CMM



Anexo 13: Marco de planeación de proyectos según Humphrey.



Anexo 14. Formato del Registro de tiempo

Formato del Registro de Tiempo

Estudiante _____ Fecha _____

Instructor _____ Programa # _____

Fecha	Inicio	Trmino	Tiempo de Interrupción	Tiempo Delta	Fase	Comentarios

Anexo 15: Resumen semanal de actividades. [16]

Nombre **Fecha**

1	Tarea									Total
2	Fecha									
3	D									
4	L									
5	M									
6	Mi									
7	J									
8	V									
9	S									
10	Totales									

11 **Tiempos y Medias del Periodo** **Número de Semanas** (número anterior +1): _____

12 Resumen de las semanas anteriores

13	Totales									
14	Med.									
15	Máx.									
16	Mín.									

17 Resumen incluyendo la última semana

18	Total									
19	Avg.									
20	Max.									
21	Min.									

Anexo 16: Formulario para estimar el tamaño de un programa. [16]

Estudiante _____ Fecha _____
Profesor _____ Clase _____

Programa	LOC	Func. anteriores	Funciones estimadas	Mín.	Media	Máx.
Estimado						

Commentarios: