Universidad de las Ciencias Informáticas
Facultad 2

Título: Evaluación de posibles elementos a considerar en la integración de los métodos de Boehm y Humphrey para la estimación de la duración de un proyecto de software para aplicaciones de gestión.

Trabajo de Diploma para optar por el título de

Ingeniero Informático

Autor(es): Robin Triana Marrero

Dayron Agüero Jimenez

Tutor: MSc. Yadira Ruíz Constanten

Cuidad de la Habana

Junio, 2008

DECLARACIÓN DE AUTORÍA

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

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

<nombre autor>	**<nombre tutor>**
_____	_____
Firma del Autor	Firma del Tutor

AGRADECIMIENTOS

A nuestra tutora que nos ayudó en todo momento.

A nuestros familiares, por querernos y apoyarnos siempre.

A nuestras novias por estar a nuestro lado y apoyarnos hasta en los momentos más duros.

A nuestros amigos, que se han preocupado por nosotros y que han estado siempre allí cuando los hemos necesitado.

A nuestros suegros y cuñadas por el apoyo que nos brindaron.

A todas aquellas personas que de una forma u otra formaron parte de nuestro desarrollo como futuros profesionales.

A la Revolución Cubana y a nuestro Comandante en Jefe Fidel Castro por haber creado un proyecto de tal magnitud y darnos la posibilidad de forjarnos como profesionales.

A todos Gracias.

DEDICATORIA

A mis abuelos por el apoyo y la confianza que me han dado y por ser las personas más especiales que me dio esta vida.

A mi tía Lydia por quererme tanto y estar siempre cerca tan de mí.

A mi mamá por preocuparse tanto por mí.

A mi papá por todo el apoyo que me brindó.

A mi tía tata por estar siempre tan cerca, aunque esta lejos.

A mis hermanos y primos por tener la dicha de contar con ellos.

Dayron

A mi mamá por estar siempre muy cerca de mí,

A mi papá por preocuparse tanto por mí,

A mis abuelos por estar siempre atentos a mí,

A mis hermanas porque las quiero mucho.

A todos los que supieron animarme a seguir adelante y a no perder las fuerzas.

Los quiero a todos.

Robin

RESUMEN

La estimación de productos de software cada vez impone más retos a nivel mundial, debido a que es necesario tener el control con toda la exactitud posible o con bastante precisión de los recursos, el costo y el tiempo apropiados para desarrollar los mismos. Los proyectos de software que se desarrollan internacionalmente por lo general presentan grandes retrasos y sobrecostes, muchos no se basan en planificaciones realistas, por tanto la calidad y funcionalidad del producto se comprometen para cumplir el calendario. La investigación centra sus esfuerzos en el estudio del comportamiento del proceso de desarrollo de Proyectos de Aplicaciones de Gestión en la Universidad de las Ciencias Informáticas (UCI), donde se detectó una serie de problemas en la práctica de la Disciplina Ingeniería y Gestión de Software, específicamente en el área de la estimación del tiempo de duración.

En la actualidad existen muchas teorías, modelos y métodos de estimación, pero aún así continúan existiendo problemas a la hora de estimar duración de proyectos. La UCI no se excluye de todo esto, al ser una organización joven, prácticamente la pionera de la producción de software en Cuba, no tiene experiencia en temas de esta índole, por lo que aplica varias teorías para desarrollar este proceso y no sigue una metodología estándar a la hora de realizarlo. Esta investigación expone algunos modelos de estimación, amplia acerca de las métricas del software, hace énfasis en las teorías de Barry Boehm y Watts Humphrey, así como desglosa y explica los puntos de contacto existentes entre las mismas provenientes de la investigación que se tomó como precedente.

Con el objetivo de dar solución a los problemas planteados, se propone una integración entre las teorías de Boehm y Humphrey para la estimación del tiempo de duración del proceso de desarrollo de Aplicaciones de Gestión.

PALABRAS CLAVES

Estimación, duración, planificación, integración, Gestión de Proyectos, Aplicaciones de Gestión.

TABLA DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

Desde la antigüedad el hombre ha necesitado organizarse y planificarse para lograr cumplir sus objetivos con éxito. Con toda esta necesidad de planificación surgieron teorías relacionadas con la obtención de metodologías para lograr productos específicos, este es el caso del desarrollo de aplicaciones de software con calidad.

La vía más usual para la producción de software es la realización de proyectos, estos pueden surgir de distintas circunstancias y situaciones en dependencia de las necesidades que tenga una determinada organización y los beneficios que pueda aportar el software a la misma; todo proyecto es único en cuanto a sus características y los objetivos a alcanzar, por lo que es necesario una gestión apropiada que se adapte a las características del mismo. Puede decirse que un proyecto es un conjunto de etapas, actividades y tareas para alcanzar un objetivo que implica un trabajo no inmediato, en un plazo relativamente largo.

En la actualidad los proyectos tienden a ser crecientes en complejidad tanto técnica como de gestión. En realidad las dos dimensiones tienen un efecto sinérgico importante, en la medida en que la complejidad crece, suele llevar acompañada una presión adicional sobre la gestión en términos de recursos limitados, plazos recortados por una fecha de entrega fija y dilaciones en las negociaciones previas a la adjudicación, normalmente en un entorno de riesgo intrínseco y que debe ser tenido en cuenta.

Existen estudios centrados en el análisis de fallos en proyectos reales que muestran que una parte relevante provienen de aspectos relacionados con la organización y gestión (37%) y con aspectos de planificación y control de los mismos (15%) [Selin, 1994].

Mucho se ha hablado, escrito y discutido sobre gestión de proyectos en general y de proyectos informáticos en particular, la tendencia hace unos años se centraba en el análisis, diseño, estilo de codificación, estimación de costes, etc. Se generaban montones y montones de diagramas, gráficos, montañas de papel y megas de información. Se planificaba, se daban unos plazos y todo entraba en una caja negra de la que al final salía el producto. Un producto que en muchas ocasiones no satisfacía al cliente, en el que los plazos se cumplían tarde, mal

o nunca, rara vez con una calidad aceptable y donde los costes se disparaban en todos los casos.

En materia de tecnología de hardware y software, los últimos cincuenta años han estado regidos por cambios significativos. En lo que respecta al hardware, una constante evolución en su arquitectura, aumentos tanto de capacidad de almacenamiento como de velocidad en el procesamiento y una amplia variedad de opciones de entrada y salida, entre otras razones, han acelerado y acrecentado su uso en diferentes tipos de sistemas basados en computadoras. El software también ha tenido un giro importante en la última mitad del siglo veinte. En sus inicios (durante las décadas del cincuenta y sesenta), su desarrollo estaba destinado a unas pocas personas, no había una planificación previa y era considerado más un arte que una disciplina [Pressman, 1998]; generalmente quien programaba un sistema, lo operaba y mantenía. A mediados de los setenta, y conforme crecía la demanda de software, se comenzó a pensar en éste como un producto que debía ser desarrollado y distribuido a una cantidad significativa de usuarios. Al igual que el hardware, su uso ha ido creciendo y se ha expandido a diferentes sectores entre los que se encuentran: la administración de información en tareas de oficina, el transporte, la medicina, la educación, la aeronavegación, la industria, el entretenimiento, entre otros.

En el principio el costo del Software constituía un pequeño porcentaje del costo total de los sistemas basados en Computadoras. Hoy en día el Software es el elemento más caro de la mayoría de los sistemas informáticos.

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.

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

proyecto. Además las estimaciones deberían definir los escenarios del mejor caso, y peor caso, de modo que los resultados del proyecto pueden limitarse.

La estimación y la planificación son actividades relacionadas pero difieren en su alcance y propósito. La estimación normalmente está orientada al proyecto en su conjunto, mientras que la planificación esta dirigida a los individuos. Obviamente, debe existir una fuerte correlación entre la estimación realizada y las tareas específicas a realizar día a día por el equipo de proyecto.

Ana Sánchez Capuchino define la estimación como “el proceso que proporciona un valor a un conjunto de variables. Para la realización de un trabajo, dentro de un rango aceptable de tolerancia.” [Capuchino, 1996].

Las variables a estimar dentro del marco de un proyecto de desarrollo de software pueden ser muchas. Estas van desde la estimación de las horas necesarias para realizar tareas tales como análisis, diseño, gestión, desarrollo, etc., hasta la estimación de índices tales como cantidad de defectos por unidad de medida, casos de prueba por caso de uso etc.

Es necesario enfatizar en la importancia de las estimaciones en la gestión de proyectos. Cualquier cronograma de tareas pendientes implica estimaciones. Cualquier intención de calcular tiempos, recursos o costos a futuro implica estimaciones. Sin estimaciones no podría haber gestión de proyectos por que el proyecto es, como su nombre lo indica, una proyección.

Es real que la estimación es solo una aproximación, pero justamente la idea es identificar los mecanismos suficientes para que esa aproximación resulte lo menos distante posible de la realidad. La actividad de estimación no se hace una sola vez en el proyecto. A medida que se cuenta con más datos se hacen estimaciones más precisas que permiten una mejor planificación de lo que resta del proyecto.

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

Actualmente es un desafío constante para las empresas de desarrollo de software convertirse en un entidad seria que logre confianza en sus clientes, para ellos el objetivo de cumplir con los plazos establecidos, comprender las necesidades de los clientes y ser prudentes en identificar sus propios límites, se convierte en una de las principales herramientas competitivas y es el factor de diferenciación de la competencia.

Además de lo expresado anteriormente se está atravesando a nivel mundial una crisis debido a la gran demanda de profesionales de sistemas, tornándose esto en uno de los rubros mejores pagos en el mundo. Esto permite identificar claramente que los desvíos producidos en los proyectos producto de una errónea estimación son causa de grandes pérdidas para las empresas, es decir, se torna extremadamente oneroso afrontar los costos de una mala planificación debido a los altos sueldos que poseen los desarrolladores. Se suma a la problemática mencionada la desmotivación que los desarrolladores adquieren en aquellos casos donde son presionados para llegar a las fechas establecidas.

Un gran error en la estimación del costo puede ser lo que marque la diferencia entre beneficios y pérdidas, la estimación del costo y del esfuerzo del software nunca será una ciencia exacta, son demasiadas las variables: humanas, técnicas, de entorno, políticas, que pueden afectar el costo final del software y el esfuerzo aplicado para desarrollarlo.

Al estimar se debe 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, servir de fundamento para los cronogramas, la asignación de personal, la gerencia de proyectos y definición de estructura y evitar problemas como la renegociación de contratos, sobre tiempos e incrementos de los costos de los usuarios o de los costos de los proyectos.

La correcta estimación temprana de un proyecto de software es una tarea difícil o casi imposible. Esta situación es conocida y es por eso que cada vez más las empresas de desarrollo intentan vender el análisis y diseño separado de la codificación, a fin de que esta última pueda ser estimada sobre bases más firmes. En particular el Proceso Unificado de Desarrollo de Software sostiene que sólo al final de la fase de Elaboración se está en

condiciones de hacer una propuesta económica firme, lo cual implica haber consumido del 25% al 30% de los recursos del proyecto al llegar a ese punto [Ivar JACOBSON, 2000].

Independientemente de lo que resulta más correcto, lo cierto es que el mercado exige cerrar un precio y una fecha de entrega antes de comenzar los trabajos. En ese contexto la única información existente es una evaluación, no muy detallada, de la funcionalidad del sistema.
Las razones por las cuales se debe aplicar un método de estimación de esfuerzo preciso son altamente contundentes. El actual mercado altamente demandante exige siempre buena calidad, bajos costos y plazos cortos. La vertiginosa velocidad que los negocios han adquirido nivel mundial y también local no resisten proyectos que se extiendan demasiado en el tiempo y que requieran de grandes inversiones.

En la Universidad de las Ciencias Informáticas (UCI) por ejemplo, las estimaciones las hace generalmente el líder o el planificador del proyecto, basándose en la experiencia y en la buena información histórica que tenga acerca de proyectos similares o que hayan trabajado sobre la misma línea de producción. Debido a que la UCI es una institución recién creada y prácticamente la pionera en el campo de la fabricación de productos de software en el país, no se tiene experiencia ni suficiente información histórica para hacer una buena estimación, además no se puede confiar en las métricas hechas con anterioridad en el centro, ya que en la Universidad de las Ciencias Informáticas (UCI), es algo muy usual que solo se estime la duración de un proyecto una sola vez en etapas muy tempranas del mismo, donde existe una evidente falta de claridad que no se podría encontrar hasta ya bien avanzado el proyecto.

En la mayoría de los casos, a la hora de hacer las estimaciones en la Universidad de las Ciencias Informáticas se utiliza un solo método de estimación, careciendo así de la posibilidad de comparación entre los resultados que se pudieran obtener si se emplearan otros métodos, para así lograr una estimación más exacta.

En el caso de la estimación por Puntos de Casos de Uso, que viene a ser el método más usado en la Universidad y 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, aparecen entonces otros problemas, porque ya no depende sólo de la persona

encargada de realizar la estimación, que en algunos casos no es la más indicada ni la más calificada para hacerlo, 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 se deba incluir o no algún caso de uso que en el primer instante no se consideraba como crítico y después se decide modificar su clasificación.

La investigación está basada en proyectos que estén desarrollándose o que se hayan terminado recientemente en la Universidad de las Ciencias Informáticas que estén dirigidos a la Gestión, empleando como punto de partida la investigación realizada con el título: "Elementos a considerar en la integración de los métodos de Boehm y Humphrey para la estimación de la duración de un proyecto de software", donde se enuncian los puntos de contacto existentes entre ambas teorías.

Por tanto el **Problema científico** se puede formular de la siguiente manera**:** ¿Cuáles son los resultados que ofrece evaluar los elementos de integración de los métodos de Boehm y Humphrey para la estimación de la duración de un proyecto de software para aplicaciones de gestión?

Definiéndose en la investigación como **Objeto de estudio** los proyectos de software para aplicaciones de Gestión en la Universidad de las Ciencias Informáticas.

Tomando como **Campo de acción** la estimación de la duración de un proyecto de software para aplicaciones de Gestión en la Universidad de las Ciencias Informáticas.

Con el fin de solucionar el problema planteado se tiene como **Objetivo general** evaluar los elementos de integración de los métodos de Boehm y Humphrey para la estimación de la duración de un proyecto de software para aplicaciones de Gestión.

La **Idea a defender** por este trabajo investigativo es la que expone a continuación: La integración de los puntos de contacto que tienen los métodos de Humphrey y Boehm provee una estimación más exacta de la duración de las aplicaciones de Gestión.

Concretándose los ***Objetivos específicos*** en:

1. Estudiar las métricas y los modelos de estimación.

2. Validar la integración entre los elementos comunes de las teorías de Boehm y Humphrey en los proyectos de aplicaciones de Gestión.

Para cumplir el objetivo de la investigación, se llevaron a cabo las siguientes ***Tareas investigativas***:

1. Analizar y estudiar las métricas de software vinculadas con la medición y estimación de un proyecto.

2. Analizar las teorías de estimación propuestas por Boehm y Humphrey.

3. Realizar un estudio significativo del método PROBE.

4. Estudiar los elementos de integración entre las teorías de Boehm y Humphrey en la duración de proyectos.

5. Evaluar la integración entre los elementos comunes de las teorías de Boehm y Humphrey en los proyectos de aplicaciones de Gestión.

Los **Métodos teóricos** que se utilizan en el proceso de investigación son: el analítico sintético, pues a partir de un estudio detallado de las teorías, tendencias y documentos relacionados con el tema, se pueden resumir los elementos más importantes y de mayor utilidad, para el desarrollo del trabajo, y en su momento, proponer una solución más acertada. El histórico lógico pues estudia toda la trayectoria, evolución y desarrollo de los fenómenos tratados. El hipotético deductivo ya que permite exponer los objetivos específicos dada la problemática en cuestión.

Además se pondrá en práctica el ***Método empírico*** de ***entrevista*** que permite obtener información cualitativa sobre el fenómeno que se investiga. Se usó la entrevista para la recogida de información sobre cómo se realiza el proceso de estimación de duración de aplicaciones de Gestión en la UCI y para recoger las opiniones de la situación actual sobre el tema investigado.

Con el propósito de organizar y darle una estructura al trabajo se ha decidido dividirlo en 3 capítulos.

Capítulo 1: Aborda temas como: ¿Qué es un proyecto de software?, ¿Qué es un proceso de Gestión de software? ; ¿Qué es la planificación?; ¿Qué es la estimación?; ¿Qué es un software de Gestión? y se hará una breve referencia a las teorías de Boehm y Humphrey.

Capítulo 2: Recoge un estudio más detallado del tema de la investigación. En el mismo se abordan temas como el Proceso Personal de Software (PSP), cómo surge, los principios en los que esta basado, etc. Además también se tratan tópicos como: el método PROBE y los Puntos de Contacto que arrojó la investigación que se toma como precedente.

Capítulo 3: Tiene como objetivo enunciar los argumentos utilizados para validar la integración entre los elementos comunes de las teorías de Boehm y Humphrey en los proyectos de aplicaciones de Gestión en la UCI. Se plantea una entrevista a algunos especialistas seleccionados así como se expone los requisitos que deberían reunir los mismos para poder ser entrevistados, se enuncian además contenidos como: los conocimientos y la forma de estimar y planificar en los proyectos de Gestión en la universidad, también se hablará del impacto del Proceso Personal del Software, los Factores de Escala y las líneas y la reutilización de código en la UCI.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

Introducción

Cada día las personas se hacen más dependientes de la tecnología y el desarrollo de la misma ha facilitado en gran medida actividades que en otra época hubieran sido de gran envergadura para el hombre. Los proyectos de software son una parte importante de todo este desarrollo, en el transcurso de este capítulo se abordarán algunos conceptos que aclararán interrogantes como: ¿Qué es un proyecto de software?, así como algunos antecedentes de los mismos; ¿Qué es un proceso de Gestión de software? ; ¿Qué es la planificación?; ¿Qué es la estimación?; ¿Qué es un software de Gestión?; así como algunas de las métricas usadas para medir el tamaño a la duración de los mismos y se hará una breve referencia a las teorías de Boehm y Humphrey.

1.1 Proyectos de Software

Para ingresar en el tema cabe plantear la siguiente interrogante: ¿Qué es un Proyecto de Software? A lo largo de la historia de la ingeniería informática se han adoptado un sin número de definiciones para esta interrogante entre los que se destacan definiciones no muy lejanas a las utilizadas en otros campos del conocimiento como las siguientes:

“Conjunto de actividades, planificadas, ejecutadas y supervisadas que, con recursos finitos, tiene por objeto crear un servicio o producto único”. [Ajenjo, 2000]

“Combinación de todos los recursos necesarios, reunidos en una organización temporal, para la transformación de una idea en realidad”. [Castillo, 1999]

“Trabajo en el que se define un principio y un fin (el tiempo), se especifica un resultado deseado (alcance), cumpliendo con unos requisitos de calidad (acabado) y coste (presupuesto)”. [Courter, 2000]

Desde estos puntos de vistas conceptuales los autores identifican las siguientes características que hacen a un proyecto distinto a cualquier otra actividad, dígase proceso o tarea:

- ✓ Actividades planificadas, ejecutadas y supervisadas

- ✓ Disponibilidad limitada de recursos
- ✓ Limitado en el tiempo
- ✓ Con un resultado único

1.1.1 Antecedentes.

La industria del software, a diferencia de otras industrias, tiene muy poco tiempo de existir. Lo que ha llamado la atención del mercado hacia ella han sido dos factores esenciales: la velocidad con que ha crecido y su alcance. Desde su inicio existieron personas en distintos campos que vieron el avance que para ellos representaba hacer uso de software especializado que les permitiera automatizar procesos o acelerarlos. Durante la década del 50 aun no se tenía ninguna metodología. La computación era incipiente. No se diferenciaba bien entre el software y el hardware. Esta situación perduró hasta comenzada la siguiente década.
Al haber tanta demanda en cuanto al campo se iniciaron muchas investigaciones en la rama de software y hardware. En la década del 60 ocurre la primera gran crisis de la gestión de proyectos de software: **la crisis del OS/360.** Con la nueva línea de computadoras de IBM, la línea /360 planteó, por primera vez, la realización de un paquete de programación de tamaño mediano**.** Hasta este momento puede decirse que todos los proyectos eran pequeños o -si bien eran de tamaño mediano- se habían hecho en ambientes universitarios sin preocuparse de plazos y costos. Con el Sistema Operativo de /360 IBM desborda todos los plazos y costos imaginables. Con el tiempo los costos se redujeron y el software se convirtió en un negocio rentable. De este primer gran atraso (que no fue el único atraso en la entrega de software) nace la gestión de proyectos. Al haber tanto interés, muchas personas empezaron a desarrollar y ahí nacieron las primeras grandes empresas de software. La década del 70 se caracteriza por la realización de los grandes estudios empíricos. La difusión de las computadoras y la aparición de las mini computadoras hacen que se disponga de muchos proyectos medianos y grandes. Con todo este material empírico se hacen gran cantidad de estudios. De estos estudios se hacen modelos y metodologías. Estos son algunos de los resultados de este período que se muestran ordenados de manera cronológica:

Farr – Zagorski, 1965	Aerospace, 1977
SDC, 1966	Daly, 1977

Aron, 1969	Doty/RADC, 1977
Naval Air Development Center, 1971	Kustanowits, 1977
GRC, 1974	Walston – Félix – (fin de los ‘70s)
Tecolote Research Incorporated , 1974	Putnam, 1977
Wolverton, 1974	Albrecht, 1979- 1984 (function points)
ESD, 1975	Boehm, 1980 (COCOMO)

Tabla 1: Ejemplos de modelos y metodologías en la década del 70.

La década del 80 es el período de confrontación de los modelos y metodologías con los grandes proyectos de software. Es contradictorio, pero la difusión de las computadoras personales hace que los proyectos sean más complejos y exigentes. Esto trajo consigo un problema natural en el proceso: al haber tantos desarrolladores en distintos países y para distintas aplicaciones, empezó a haber diversidad de estilos así como la calidad de del producto final variaba mucho entre producto y producto. En este marco se hizo necesario un estándar que permitiera a los consumidores de software decidir si el producto que estaban recibiendo era de calidad y si cumplía ciertos requisitos de funcionalidad.

La década del 90, fue la época de la normalización de las metodologías. Ya se ha separado lo útil de lo teórico, lo bueno de lo malo y se procede a normalizar las metodologías de modo que obtengan resultados consistentes y comparables.

1.2 ¿Proceso de Gestión de Proyectos de Software?

Con el planteamiento de las características de proyectos expuestas hasta ahora, se puede dar entrada a la respuesta de la interrogante que le da nombre a este epígrafe. Para los autores es la actividad encaminada a garantizar el cumplimiento de las tareas expuestas en el Proceso de Desarrollo de Software para darle cumplimiento a un Proyecto de este tipo. Esta definición es muy sencilla, por lo que los autores considerando, que no satisfaga las necesidades de lectores experimentados en el tema, se dan a la tarea de abordar conceptos dados por prestigiosas organizaciones, como los que le sigue a continuación.

“Es la aplicación de conocimientos, aptitudes, herramientas y técnicas a las actividades del proyecto, encaminados a satisfacer o colmar las necesidades y expectativas de las entidades y organizaciones involucradas en un Proyecto”. [Heredia, 2003]

“Es la planificación, organización, seguimiento y control de todos los aspectos de un proyecto, así como la motivación de todos aquellos implicados en el mismo, para alcanzar los objetivos del proyecto de una forma segura, y satisfaciendo las especificaciones definidas de plazo, coste y rendimiento. Ello también incluye el conjunto de tareas de liderazgo, organización y dirección técnica del proyecto, necesarias para su correcto desarrollo”. [Brand, 1996]

Como bien se plantea en los conceptos anteriores la Gestión de Proyectos tiene como finalidad principal la planificación, el seguimiento y control de las actividades de los recursos humanos y materiales que intervienen en el desarrollo de un Sistema. Los autores quedan complacidos con esta afirmación, que corrobora la definición inicial dada por ellos.

1.3 Planificación de Proyectos de Software.

La planificación de un proyecto debe afrontarse de manera adecuada para que al final del mismo se pueda hablar de éxito. No se trata de una etapa independiente abordable en un momento concreto del ciclo del proyecto. Es decir, no se puede hablar de un antes y un después al proceso de planificación puesto que según avance el proyecto será necesario modificar tareas, reasignar recursos, etc. Se debe tener claro que si bien sí se puede hablar de una "etapa de planificación", llamada así porque aglutina la mayor parte de los esfuerzos para planificar todas las variables que se darán cita, cada vez que se intenta prever un comportamiento futuro y se toman las medidas necesarias se está planificando.

Encontrándose dos grandes fases en las que la planificación cobra el máximo protagonismo. La primera es necesaria para estudiar y establecer la viabilidad de un proyecto, ya sea interno o externo a la organización. Hay que hacer los correspondientes estudios técnicos, de mercado, financieros, de rentabilidad... así como una estimación de los recursos necesarios y los costes generados. Todo ello constituye el elemento fundamental en el que se apoya el cliente (que puede ser la propia organización en el caso de proyectos internos) para decidir sobre la realización o no del proyecto.

La segunda fase importante de planificación tiene lugar una vez se ha decidido ejecutar el proyecto. Ahora es el momento de realizar una planificación detallada punto por punto. Uno de los errores más importantes y graves en gestión de proyectos es querer arrancar con excesiva premura la obra, sin haber prestado la atención debida a una serie de tareas previas de preparación, organización y planificación que son imprescindibles para garantizar la calidad de la gestión y el éxito posterior.

Planificar es armonizar dos tipos de elementos muy diferentes entre sí:

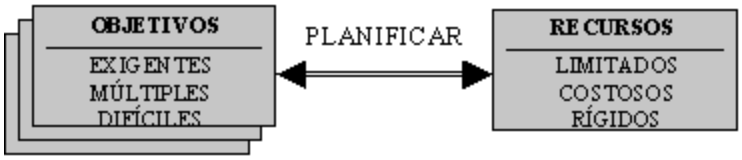


Figura 1: Armonización de elementos al planificar.

Al hilo de lo señalado al principio, la planificación de los proyectos debe estar afectada de un notable grado de agilidad y dinamismo: no es razonable planificar un proyecto y pensar que esa planificación es ya definitiva e inmutable. En casi todos los casos, la realidad no coincide exactamente con lo previsto, por lo que es necesario ir haciendo ajustes periódicos. La planificación es una herramienta para la gestión y la toma de decisiones, no para imaginar en un primer momento una evolución que posteriormente el tiempo se encargará de demostrar que estaba equivocada.

Aunque existen técnicas de planificación muy avanzadas y elaboradas, la adecuada planificación se basa, ante todo, en una actitud de anticipación que no es sino una evidente manifestación del sentido común.

Los procesos básicos de planificación se pueden resumir en el siguiente cuadro:

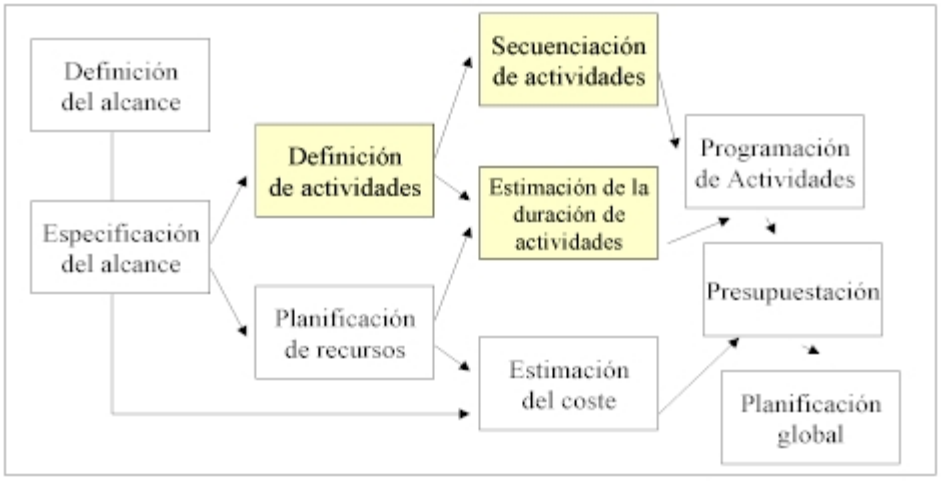


Figura 2: Procesos básicos de la planificación.

La figura anterior muestra la secuencia de procesos para hacer una planificación lo más eficiente posible, resaltando los pasos más importantes: definición de actividades, secuenciación de actividades y estimación de la duración de las mismas, operaciones vitales cuando se esta planificando un proyecto de software.

La planificación y el seguimiento son dos actividades principales del Proceso de Gestión y cabría preguntarse: ¿Qué actividad le da soporte a ellas? La respuesta a esta interrogante se encontró en el trabajo de Ana M. Moreno donde identifica una tercera actividad dentro del proceso de gestión, como otras de las fundamentales: la estimación.

Una estimación o predicción no es un objetivo, es una predicción del valor de una variable, que es igualmente probable que esté por encima o por debajo del resultado real. De modo que para indicar su calidad se ha de definir un rango, representado como una tripleta: el valor más probable (que es, la mediana de la distribución), más los límites superior e inferior del valor.

1.4 La Estimación.

La estimación es la primera actividad dentro del proceso de gestión proyectos. En este marco puede definirse la estimación como: “el proceso que proporciona un valor a un conjunto de variables para la realización de un trabajo, dentro de un rango aceptable de tolerancia.” [Garcia, 2000]

En términos estadísticos a estos límites inferior y superior se les denomina intervalos de confianza. La calidad de la estimación está dada por la efectividad de la predicción o sea que el valor real de la variable debe de estar dentro del rango de confianza de la predicción.

Las variables a estimar dentro del marco de un proyecto de desarrollo de software pueden ser muchas. Estas van desde la estimación de la duración del proyecto, el esfuerzo y el costo para realizar un proyecto, etc., hasta la estimación de índices de calidad tales como cantidad de defectos por unidad de medida, casos de prueba por caso de uso, etc. [Cao, 2006]

A continuación varios autores citarán algunas consideraciones que han tenido sobre el tema:

“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”. [Pressman, 2002]

“Estimar es echar un vistazo al futuro y aceptar resignados cierto grado de incertidumbre. Aunque la estimación, es más un arte que una Ciencia, es una actividad importante que no debe llevarse a cabo de forma descuidada”. [Concepción, 2005]

“La estimación es una actividad que se debe realizar desde las primeras etapas en que se plantea la idea del proyecto informático de construcción de software”. [Carla Basurto, 2005]

Sin lugar a dudas es la estimación una actividad de gran importancia, en el proceso de gestión, cuando se necesita saber cuánto costará en esfuerzo, materiales o tiempo llevar a cabo un proyecto informático.

1.4.1 La Estimación en la Planificación.

El objetivo de la planificación del proyecto de software es proporcionar un marco de trabajo que permita al gestor hacer estimaciones razonables de recursos, costos y planificación temporal. Estas estimaciones se hacen dentro de un marco de tiempo limitado al comienzo de un proyecto de software, y deberían actualizarse regularmente a medida que progresa el proyecto. Además las estimaciones deberían definir los escenarios del mejor caso, y peor caso, de modo que los resultados del proyecto pueden limitarse.

“La estimación y calendarización (planificación) del proyecto se llevan a cabo de forma conjunta”. [Sommerville, 2002]. Los autores viendo esta relación han llegado a considerar a la planificación como la actividad organizacional que le da uso a los valores de las variables estimadas.

Con estos planteamientos se ha de concluir que la planificación y la estimación son actividades relacionadas, pero difieren en su alcance y propósito. La estimación normalmente está orientada al proyecto en su conjunto, mientras que la planificación está dirigida a los recursos humanos y materiales.

Las etapas de estimación y planificación se harán de forma continua a todo lo largo del proceso de desarrollo de software y al inicio de cada etapa. La estimación, de forma general, se realizará al principio del proyecto, precisamente para saber cuánta gente se necesita o cuánto durará una etapa en el proceso. La planificación definirá exactamente quién hace qué y en cuánto tiempo.

1.4.2 Detalles para dar comienzo a un Proceso de Estimación de Proyectos de Software.

La estimación es un proceso continuo, a medida que el proyecto avanza, más se conoce de él, y por lo tanto, más parámetros están disponibles para introducir en un modelo de estimación.

Todo proceso de estimación esta divido en dos etapas, una primera etapa es donde se realizan los estudios de la viabilidad del proyecto o sea se establece el ámbito del software, en cual se toman en cuenta elementos como: los datos a procesar, la función, el rendimiento, las restricciones, las interfaces y la fiabilidad. Es donde se requiere solo de los datos descriptivos necesarios para determinar el tiempo y el esfuerzo que llevará realizar un proyecto. Es donde se define si es viable para la organización continuar adelante en el proyecto. El ámbito se define como un prerrequisito para la estimación.

El segundo momento de la estimación es mucho más específico, va orientado a determinar etapas dentro del proceso de desarrollo y no solo genera datos para el estudio de viabilidad sino los suficientes como para establecer una planificación detallada de las etapas del proceso

de desarrollo del proyecto. Es donde se establecen los valores necesarios de los recursos que se van a utilizar en el proceso.

Los recursos simulan una pirámide donde las Herramientas (Hardware y Software), son la base que proporcionará la infraestructura de soporte al esfuerzo de desarrollo, en segundo nivel de la pirámide se encuentran los Componentes Reutilizables que son, como su nombre lo indica, todo aquello que pueda volverse a utilizar; en la parte más alta de la pirámide se encuentra como el recurso primario, las personas (RR-HH).

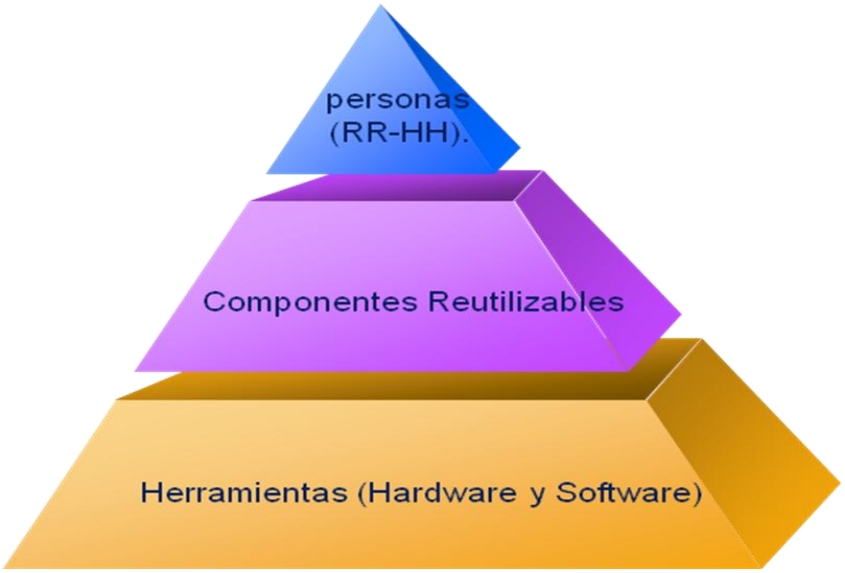


Figura 3: Recursos para dar comienzo a un Proceso de Estimación.

Debido a la diferencia que hay entre los procesos de producción industrial y los procesos de producción de software en cuanto a que los últimos generan productos intangibles y que para ello se requiere de comunicación y coordinación intensivas entre los implicados en el proceso, los recursos humanos ocupan la cúspide de la pirámide de recursos.

El entorno es donde se apoya el proyecto de Software, llamado a menudo entorno de Ingeniería de Software, incorpora el Hardware y el Software a utilizar dentro del proceso. Es lo que proporcionará la plataforma que se requiere para producir el producto, que son el resultado de la buena práctica de la Ingeniería del Software.

1.4.3 Estimación de la duración.

Se trata de evaluar el número de períodos estimados necesarios para completar el proyecto. Los datos para la estimación de duraciones son: los recursos asignados al proyecto, la capacidad (productividad) de dichos recursos, información histórica (proyectos similares, bases de datos comerciales, conocimientos y experiencia del equipo de proyecto).

Las técnicas para la estimación de duración de proyectos son: la Asesoría especializada, basada en experiencia en la gestión de proyectos en el sector; la Estimación por analogía, basada en información histórica de duraciones reales de proyectos anteriores similares; la Simulación, cálculo de múltiples duraciones basadas en distintas hipótesis.

Este análisis podría extenderse a cualquier combinación de diferentes lenguajes de implementación del diseño con resultados similares. Por tanto se podría decir que el tiempo total de duración de un proyecto esta relacionado con las características propias del mismo y no depende directamente del lenguaje de implementación del diseño así como tampoco de la calidad de personal a utilizar, sino que esta última es una consecuencia directa proveniente de los dos valores: Meses Hombre Totales MMF y Tiempo Total del Proyecto TDEV, que son calculados por el método que esté empleando para estimar la duración del proyecto.

1.4.4 Estimación del Tamaño.

Para realizar la planificación de un proyecto software, es necesario poseer una estimación certera del esfuerzo necesario para el desarrollo lo más temprano posible, idealmente, con sólo la etapa de especificación de requisitos cubierta. Se tiene que a esta altura del desarrollo del software, difícilmente se puede realizar una estimación certera de la cantidad de líneas de código que tendrá la aplicación, ya que en este nivel no tiene por qué estar decidida la herramienta de desarrollo. Sólo se podría entrar a realizar una estimación certera en los comienzos de la etapa de construcción, con un diseño acabado.

De manera general, todos los métodos de estimación utilizan datos históricos de proyectos pasados para estimar el tamaño del nuevo software a desarrollar, ver el siguiente gráfico. [Estimating of Software Size - Part II, 2003]

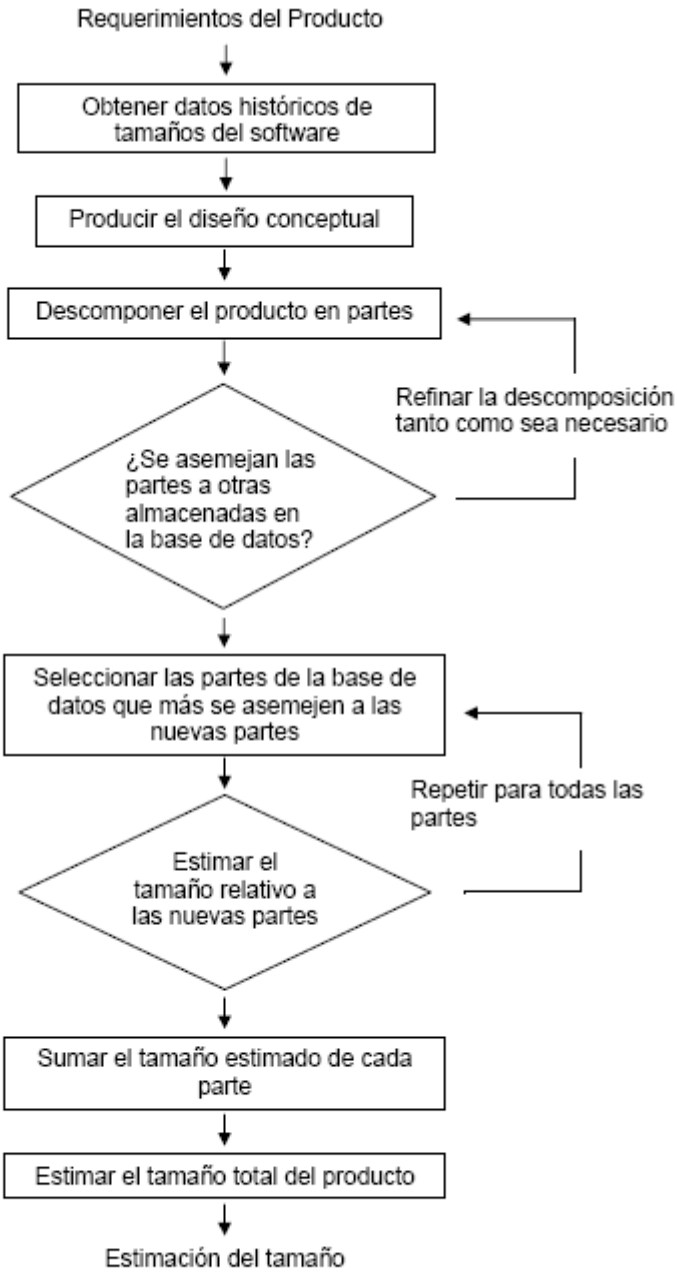


Figura 4: Vista general de los procedimientos de estimación del tamaño del software. Descomposición del software y comparación con datos históricos.

1.4.5 Técnicas comúnmente utilizadas.

Para la estimación, existen cuatro técnicas básicas y comunes [Garcia, 2000]:

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

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

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

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

Atributos comunes de las estimaciones.

Se han desarrollado varias técnicas de estimación para el desarrollo de software, aunque cada una tiene sus puntos fuertes y sus puntos débiles, todas tienen en común los siguientes atributos:

- Se han de establecer de antemano el **ámbito del proyecto**.
- Como base para la realización de estimaciones se usan **métricas del software** de proyectos pasados.
- El **proyecto se desglosa en partes** más pequeñas que se estiman individualmente.

1.4.6 Métodos de estimación.

Jerzy Nawrocki, famoso científico polaco que se destaca por sus estudios en el área de las métricas de estimación, propone la aplicación de los siguientes métodos de estimación del tamaño del software según la fase del proyecto:

Después del análisis de los requerimientos:

- Puntos de Función

Los Puntos de Función miden la aplicación desde una perspectiva del usuario, dejando de lado los detalles de codificación. Es una técnica totalmente independiente de todas las consideraciones de lenguaje y ha sido aplicada en más de 250 lenguajes diferentes. Se supone que el Análisis por Puntos de Función evalúa con fiabilidad: el valor comercial de un sistema para el usuario, el tamaño del proyecto, el coste y el tiempo de desarrollo, la calidad y productividad del programador, el esfuerzo de adaptación, modificación y mantenimiento, la posibilidad de desarrollo propio y los beneficios de implementación.

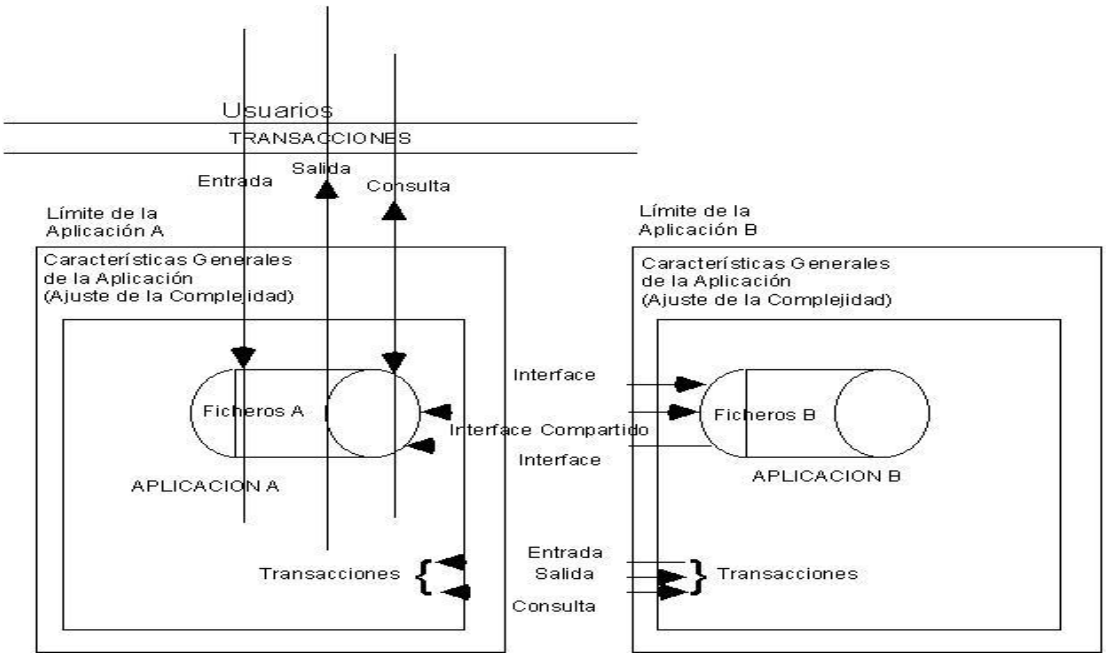


Figura 5: Relaciones entre Usuarios, Aplicaciones y Funciones.

Un Punto de Función se define como una función comercial de usuario final. De esta manera un programa que tenga “x” PF’s entrega “x” funciones al usuario final. El mejor modo de trabajo es la interacción analista-usuario.

El proceso requiere dos etapas fundamentales:

1. Se identifican las funciones disponibles para el usuario y se organizan en cinco grupos (mejor en este orden): salidas, consultas, entradas, ficheros, interfaces. Después se clasifica y pondera cada función por su nivel de complejidad (simple, media, compleja).

2. Se ajusta este total de acuerdo con unas características del entorno (Ver **Figura 5**).

Después de un diseño de alto nivel:

- Método Wideband-Delphi

El método **Wideband-Delphi** es una técnica de estimación estructurada en grupo. Deriva de un método creado en los 40. Se basa fundamentalmente en que varios expertos, tras crear estimaciones individuales, se reúnen para ponerse de acuerdo en una estimación. Es una técnica de estimación de la duración de una tarea basada en el consenso, muy útil para proyectos de software. Se trata de un proceso de iteración de estimaciones anónimas moderadas por un coordinador, con la finalidad de alcanzar un consenso final en la estimación.

Un estudio del método original decía que no se obtenían mejores resultados que con estimaciones individuales debido a las presiones políticas que se podían ejercer sobre el grupo. Así que Boehm y Farquhar crean en los 70 el Wideband-Delphi como mejora del original.

- Método Difuso de Putnam

El método de estimación de Putnam es un modelo multivariable dinámico que asume una distribución específica del esfuerzo a lo largo de la vida de un proyecto de desarrollo de software. El modelo se ha obtenido a partir de distribuciones de mano de obra en grandes proyectos (esfuerzo total de 30 personas-año o más). Sin embargo, se puede extrapolar a proyectos más pequeños.

Este enfoque utiliza las técnicas aproximadas de razonamiento que son la piedra angular de la lógica difusa. Para aplicar este enfoque, el planificador debe identificar el tipo de aplicación, establecer su magnitud en una escala cuantitativa y refinar la magnitud dentro del rango original. Aunque se puede utilizar la experiencia personal, el planificador también debería tener acceso a una base de datos histórica de proyectos para que las estimaciones se puedan comparar con la experiencia real [Pressman, 2002].

• Método de Componentes Estándar

El método de estimación del tamaño del software basado en componentes estándar, es descrito por Putnam como una vía para hacer estimaciones progresivamente más refinadas utilizando datos históricos.

Se comienza obteniendo datos de tamaños de varios niveles de abstracción del programa, por ejemplo, subsistemas, módulos y pantallas. Luego se estima que cantidad de cada uno de estos componentes se necesitará para el nuevo programa. Se estima la mayor y la menor cantidad para cada tipo de los componentes candidatos. Estas estimaciones se combinan en una función probabilística de tres puntos, como por ejemplo:

Valor estimado = [menor valor + 4*(valor más probable) + mayor valor] /6

En esta ecuación, el valor más probable suele tomarse como el promedio del mayor y el menor valor estimado.

Después de un diseño de bajo nivel:

• Método PROBE

A través de este método, los Ingenieros usan el tamaño relativo de un Proxy (medida o estimador, de la unidad de tamaño) para hacer una estimación inicial y utilizan los datos históricos para convertir el tamaño relativo del Proxy a LOC (líneas de código). Este método es propuesto por Watts S. Humphrey para la estimación de la duración de un proyecto de software, el mismo será abordado con mayor profundidad durante el desarrollo del trabajo.

1.4.7 Modelos de estimaciones.

Los modelos de estimación son el resultado de dos elementos: *una idea*, a priori, acerca del proceso de fabricación del software y una *ecuación empírica* ajustada a partir de casos de estudio reales. Los modelos de estimación pueden ser clasificados según diferentes criterios.

Según el tipo de resultados que se obtienen, se pueden clasificar en:

- Modelos estáticos
- Modelos dinámicos

Los modelos estáticos dan estimaciones globales o promedio. Los modelos dinámicos, complementarios de los estáticos, permiten realizar estimaciones a lo largo del proyecto.

Según la cantidad de variables básicas que tienen en cuenta se pueden clasificar en:

- Modelos de una variable.
- Modelos de varias variables.

Las opciones abiertas significan un compromiso entre la sencillez (una variable) y la precisión (varias variables). Según las situaciones interesará más uno u otro.

Según el tipo de ecuaciones empíricas que se ajusten, se pueden clasificar en:

- Modelos lineales.
- Modelos potenciales o exponenciales.
- Otros modelos

Otros modelos de estimación son: los analógicos, teóricos, heurísticos y empíricos. Estos van a estar clasificados según el método de predicción en el que están basados fundamentalmente, ya que un modelo puede estar basado en más de uno de estos métodos. Se hará una subclasificación de los modelos empíricos en paramétricos y no paramétricos. Un modelo paramétrico tiene una fórmula funcional explícita, relacionando una variable dependiente con una o más variables independientes. Otras subclasificaciones pudieran ser la de modelos análogos, modelos teóricos y los heurísticos. A continuación se abordarán algunos de estos brevemente.

Modelos analógicos.

El modelo analógico de estimación de costes, se basa en comparar uno o más proyectos finalizados en un dominio como medio para producir una nueva estimación. La estimación se lleva a cabo analizando los datos recolectados de estos proyectos finalizados y contrastándolos con el nuevo proyecto, evaluando similitudes. Como el esfuerzo de desarrollo se conoce de antemano, se utiliza como base de la estimación del nuevo proyecto. La estimación por analogía parece sencilla y directa, sin embargo hay una serie de problemas que han de ser determinados. La manera de establecer las características del proyecto es uno

de estos problemas, puesto que al principio del proceso de estimación hay restricciones en cuanto a la información disponible. Ejemplo de esto pueden ser el dominio de la aplicación, el proceso de desarrollo del software, etc. Otro problema común es la manera de contrastar varios proyectos a la vez. Siempre puede haber proyectos disidentes que hagan desviar la estimación.

Modelos teóricos.

Los modelos teóricos son más complejos desde el punto de vista de lograr su validez. Su pertinencia para los sistemas reales deberá evaluarse mediante la experimentación. Estos modelos se construyen con bloques conceptuales básicos como definiciones, axiomas, hipótesis, principios, etc., seguidos de una derivación analítica a partir de estos puntos básicos de partida. Como todos los elementos de las teorías, son construcciones de la imaginación humana.

En esta sección se hablará fundamentalmente de un modelo teórico realizado en 1991 por Abdel-Hamid, cuyas características principales se exponen a continuación. La realimentación dinámica de la relación entre equipo de producción, desarrollo del software, planificación y control, se modela mediante un lenguaje de simulación. Los escenarios de simulación de gestión de proyectos, pueden ser ejecutados para investigar los efectos de las decisiones y políticas de gestión. El modelo trabaja partiendo de una estimación inicial para el esfuerzo total y luego explora qué influencia tienen sobre el esfuerzo total, así como hace suposiciones acerca de las iteraciones y la realimentación entre decisiones y proyectos.

Un escenario explorado por Abdel-Hamind y Madnick considera las circunstancias en las que se añade más gente al final del proyecto para permitir una finalización más rápida de este. La ley de Brooks establece que: “Añadir más personal al final del proyecto hace que este se retrase más”, por otra parte, el modelo de Abdel-Hamid y Madnick indica que añadir más personal al final del proyecto hace que este sea más costoso, pero no necesariamente más lento. El punto de desarrollo en el cual se añade personal extra y la experiencia del personal a añadir, son factores claves para el avance o retraso del proyecto. Abdel-Hamind y Madnick demostraron mediante su modelo cómo subestimaciones y sobrestimaciones del esfuerzo pueden ser determinantes para disminuir la productividad e incrementar el esfuerzo total.

Como las estimaciones para proyectos nuevos se basan en proyectos pasados, ellos afirman que su modelo puede ser utilizado para explorar si para un proyecto finalizado se realizó el mínimo esfuerzo o no, y esto pudiera utilizarse como factor de corrección para estimaciones futuras.

Modelos Empíricos.

Un modelo empírico de estimación para el software de computadora utiliza fórmulas derivadas empíricamente para predecir los datos que se requieren en el paso de planificación del proyecto de software. Los datos empíricos que soportan la mayoría de los modelos se obtienen de una muestra de proyectos limitada. Por esta razón, un mismo modelo de estimación no es adecuado para todas las clases de software ni para todos los entornos de desarrollo. Por lo tanto, los resultados obtenidos de los modelos deben utilizarse de forma sensata.

Los Modelos de recursos consisten en una o varias ecuaciones obtenidas empíricamente que predicen el esfuerzo (en personas/mes), la duración del proyecto (en meses cronológicos) y otros datos relativos al proyecto. Se describen a nivel mundial cuatro clases de modelos de recursos: modelos univariable estáticos, modelos multivariables estáticos, modelos multivariables dinámicos y modelos teóricos.
Un univariable estático toma la Forma:

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

Donde el recurso podría ser el esfuerzo, la duración del proyecto, la cantidad de personal o las líneas requeridas de documentación del software. Las constates c_1 y c_2 se derivan de los datos recopilados de anteriores proyectos. La característica estimada puede ser la cantidad de líneas de código fuente, el esfuerzo (si ya está estimado) u otra característica del software. Se pueden derivar modelos con la forma recién descrita para un entorno local si hay suficientes datos históricos disponibles. La versión básica del modelo de coste constructivo o COCOMO 81 es un ejemplo de modelo univariable estático.

Los modelos multivariables estáticos, como sus análogos univariable estáticos, usan los datos históricos para obtener relaciones empíricas. Un modelo típico de esta categoría toma la forma:

$$\text{Recurso} = c_{11}e_1 + c_{21}e_2 + \ldots$$

Donde e_i es la característica i-ésima del software y ci_1, ci_2 son constantes obtenidas empíricamente para la característica i-ésima.

Un modelo multivariable dinámico proyecta los requisitos de recursos como una función del tiempo. Si se obtiene empíricamente el modelo, los recursos se definen en una serie de pasos consecutivos en el tiempo que asigna cierto porcentaje de esfuerzo (o de otro recurso) a cada etapa del proceso de ingeniería del software. Cada paso puede ser además subdividido en tareas. El enfoque teórico de la modelización multivariable dinámica incluye una "curva continua de utilización del recurso" como hipótesis y, a partir de ella, obtiene ecuaciones que modelan el comportamiento del recurso.

Herramientas Automáticas de Estimación.

Los modelos empíricos de estimación descritos en la sección anterior se pueden implementar con software. Las herramientas automáticas de estimación permiten al planificador estimar costes y esfuerzos, así como llevar a cabo análisis del tipo "qué pasa si" con importantes variables del proyecto, tales como la fecha de entrega o la selección de personal. Aunque existen muchas herramientas automáticas de estimación, todas exhiben las mismas características generales y todas requieren una o más de las siguientes clases de datos:

1. Una estimación cuantitativa del tamaño del proyecto (por ejemplo. en líneas de código) o de la funcionalidad (datos sobre puntos de función)
2. Características cualitativas del proyecto, tales como la complejidad, la fiabilidad requerida o el grado crítico del negocio.
3. Alguna descripción del personal de desarrollo y/o del entorno de desarrollo.

A partir de estos datos, el modelo implementado por la herramienta automática de estimación proporciona estimaciones del esfuerzo requerido para llevar a cabo el proyecto, de los costes, de la carga de personal y en algunos casos, de la agenda de desarrollo y del riesgo asociado.

1.4.8 Teorías de Boehm y Humphrey.

Barry Boehm se graduó de B.A. en Harvard, en 1957 recibió su grado de B.A., y sus grados de M.S. y de Ph.D. Se licenció de matemáticas en la Universidad de California, Los Ángeles en 1961 y 1964 respectivamente. Fue Programador-Analista en General Dynamics entre 1955 y 1959. De 1959 a 1973 trabajó en la Corporación Rand culminando como jefe del departamento de las ciencias de la información.

Entre sus distinciones incluyen conferencias en la Academia de Ciencias de URSS (1970), la concesión de los sistemas de información del AIAA en 1979, el premio de J.D. Warnier por su honorabilidad en ciencias de la información en 1984, la concesión de ISPA Freiman para análisis paramétricos en 1988, el consentimiento del logro del curso de la vida de ASQC (1994), y de la concesión distinguida ACM de la investigación en la tecnología de dotación lógica (1997).

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

El método COCOMO II permite determinar los valores de las siguientes dos variables:

- meses/hombre a aplicar al proyecto
- meses totales del proyecto (dependiendo de factores tales como los atributos de fiabilidad requerida del software, tamaño de la base de datos, complejidad del producto, limitaciones en el tiempo de ejecución, limitaciones de memoria principal, volatilidad de la máquina virtual, frecuencia de cambio en el modelo de explotación del ordenador, capacitación de los analistas, experiencia en aplicaciones, capacitación de los

programadores, experiencia en la máquina virtual, experiencia en el lenguaje de programación, prácticas modernas de programación, uso de herramientas para el desarrollo del software y limitaciones en la planificación).

En la **(Figura 6)** se presenta un esquema de estimación que proporciona además de las horas-hombre a emplear en el tiempo total del proyecto (basándose para ello en el conocimiento previo de la cantidad de sentencias de código del proyecto) lo que permite determinar el plazo de entrega. Mostrándose, además, cómo a partir de estos dos valores (horas-hombre y tiempo total), simplemente por el cociente de ambos, se obtiene la cantidad de recursos (personas) para llevarlo a cabo. A partir de allí se puede elaborar el costo mediante la aplicación de ratios, de igual forma que en las metodologías tradicionales.

Figura 6: Estimación de proyectos por método COCOMO II.

Debe tenerse en cuenta que la duración total del proyecto es un valor teórico y que puede disminuirse incrementando los recursos (personas) a emplear aunque el impacto, en razón de lo expresado anteriormente, será menor (puede alcanzar a un 20% menos) que el esfuerzo aplicado a tal efecto.

Por otra parte Watts S. Humphrey fundó el Proceso de Software Programa del Instituto de Ingeniería de Software (SEI) en la Universidad Carnegie Mellon. Es miembro del Instituto y es

un científico investigador en lo personal. De 1959 a 1986 se asoció con la Corporación IBM, donde fue director de la programación.

Sus publicaciones incluyen numerosos documentos técnicos y nueve libros. Sus tres libros más recientes son TSP: Desarrollo de Equipos (2006), TSP: Dirigir y un Equipo de Desarrollo (2005), y PSP: Un Mejoramiento del Proceso de Software de Ingenieros (2005).

En 1991 ocupó el cargo de examinador en la Junta de Malcolm Baldrige National Quality Award. Está en posesión de un título de licenciatura en física por la Universidad de Chicago, una maestría en física del Instituto de Tecnología de Illinois, y una maestría en administración de empresas por la Universidad de Chicago.

1.5 Métricas del Software.

La medición es muy común en el mundo de la ingeniería. Se miden potencia de consumo, pesos, dimensiones físicas, temperaturas, voltajes y señales de ruidos, por mencionar algunos aspectos. Desgraciadamente la medición se aleja de lo común en el mundo de la ingeniería del software. Se encuentran dificultades en ponerse de acuerdo sobre qué medir y cómo se van a evaluar las medidas.

En la mayoría de los desafíos técnicos, las métricas ayudan a entender tanto el proceso técnico que se utiliza para desarrollar un producto, como el propio producto.

En principio, podría parecer que la necesidad de la medición es algo evidente. Después de todo es lo que permite cuantificar y por consiguiente gestionar de forma más efectiva. Pero la realidad puede ser muy diferente. Frecuentemente la medición conduce a una gran controversia y discusión.

1. ¿Cuáles son las métricas apropiadas para el proceso y para el producto?
2. ¿Cómo se deben utilizar los datos que se recopilan?
3. ¿Es bueno usar medidas para comparar personas, procesos o productos?

Estas preguntas y otras tantas docenas de ellas siempre surgen cuando se intenta medir algo que no se ha medido en el pasado.

Hay varias razones por lo que se hace necesario medir un producto, dentro de las que se encuentran indicar la calidad del producto, evaluar la productividad de la gente que desarrolla el producto, evaluar los beneficios en términos de productividad y de calidad, derivados del uso de nuevos métodos y herramientas de la ingeniería de software, establecer una línea base para la estimación y ayudar a justificar el uso de nuevas herramientas o de formación adicional.

Hay varias razones para medir un producto.

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

Las métricas del software están relacionadas con el desarrollo del software como funcionalidad, complejidad y eficiencia.

MÉTRICAS TÉCNICAS: Se centran en las características de software por ejemplo: la complejidad lógica, el grado de modularidad. Miden la estructura del sistema y cómo esta hecho.

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

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

MÉTRICAS ORIENTADAS A LA PERSONA: Proporcionan medidas e información sobre la forma que la gente desarrolla el software de computadoras, y sobre todo, el punto de vista humano de la efectividad de las herramientas y métodos. Son las medidas que voy a hacer del personal que hará el sistema.

MÉTRICAS ORIENTADAS AL TAMAÑO: Es para saber en qué tiempo voy a terminar el software y cuántas personas voy a necesitar. Son medidas directas al software y al proceso por el cual se desarrolla.

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

La Lic. Sánchez Capuchino [Capuchino, 1996] define las características que debe tener una buena métrica:

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

1.6 Aplicaciones de Gestión.

Se entiende como aplicación de gestión aquella que se diseña para sustituir uno o varios procedimientos, tanto comerciales como administrativos, que habitualmente realiza una persona en una empresa o institución de forma presencial, por una serie de pantallas en un ordenador, que permitan realizar al cliente los mismos procedimientos de forma no presencial.

Los nuevos retos en las aplicaciones de gestión.

El cambio de milenio se presentó como la frontera en la que desaparecen viejos convencionalismos presentes en la aplicaciones de contabilidad desde los años 80’s y se van a confirmar los nuevos conceptos que han despuntado en las aplicaciones de gestión de negocios empresariales (ERP) que han emergido en los 90’s.

En un mundo de imagen en el que triunfa el entorno visual, las aplicaciones de software que son gráficas representan el presente y el futuro. Las antiguas pantallas que se utilizaban con cientos de combinaciones de teclas de mandato para realizar diferentes tareas están obsoletas. En aquellas aplicaciones grandes, rígidas y pesadas la flexibilidad la ponía el usuario.

Ahora cuando un cliente desea conocer información, lo primero que hace es visualizar una imagen en la pantalla de un ordenador. Ya no tiene que consultar inmensos listados para localizar la información. Con las nuevas tecnologías en pocos segundos consigue la información que desea y con mayor fiabilidad que antes.

Un nuevo paso es la incorporación de las tecnologías multimedia en los procesos de gestión. La presentación visual se impone en todos los entornos. Se sigue haciendo el mismo trabajo que antes, pero ahora de forma incomparablemente más eficaz. Los incrementos significativos de productividad que además incorporan una reducción de costes son los motores del cambio hacia nuevas aplicaciones de gestión.

Otro aspecto que preocupa al usuario de las aplicaciones de gestión es la facilidad de uso. Las aplicaciones de software emergentes deben disponer de un ambiente pensado para y por el usuario. Por ejemplo: realizar las tareas en la administración de una empresa debe ser tan fácil como utilizar la hoja de cálculo o el procesador de texto. El objetivo para el usuario es trabajar de forma fácil y homogénea. En este sentido las aplicaciones de gestión tienen muchos desafíos pendientes.

Un aspecto que preocupa es la flexibilidad de la aplicación de gestión. En un mundo cambiante, en el que la información fluye con mayor rapidez de un extremo a otro, hay que tomar decisiones en función de la información disponible. Y esto requiere de gran capacidad de análisis.

Hoy con las nuevas capacidades de los ordenadores y el nuevo software que es capaz de aprovechar su potencia al máximo, se puede tener la información al instante.

Las aplicaciones de gestión basadas en la arquitectura cliente/servidor permiten mayor acceso y más rápido a la información. El usuario tiene un ordenador en su mesa, al que llegan los datos que necesita y que posteriormente puede analizar en su propio ordenador con su propio criterio de análisis. Las nuevas tecnologías han aportado nuevos conceptos que proporcionan mayor productividad e independencia como son:

- drill-down (taladrar dentro): es una de las herramientas más significativas y útiles del sistema, ya que permite al usuario desglosar cualquier dato de un informe hasta el comprobante original, para conocer al detalle de cómo fue calculado. Se puede llegar a visualizar hasta la operación que hace, si parte del cálculo o generó el dato.

- drill-around (taladrar alrededor): con esta herramienta se puede navegar de forma intuitiva por toda la información histórica relacionada con cada archivo y se puede obtener información rápido y fácilmente.

- OLAP (procesamiento analítico en línea): cuyo objetivo es agilizar la consulta de grandes cantidades de datos. Para ello utiliza estructuras multidimensionales (o Cubos OLAP) que contienen datos resumidos de grandes Bases de Datos o Sistemas Transaccionales (OLTP). Se usa en informes de negocios de ventas, marketing, informes de dirección, minería de datos y áreas similares.

Otro criterio es la fiabilidad de la información. Nadie en un entorno de gestión esta dispuesto a perder la información. La información es un activo precioso y sólo pensar en la idea de perder los datos de los clientes hace temblar. Por tanto las nuevas aplicaciones deben incorporar sistemas que garanticen la seguridad y coherencia de la información.

Otro factor importante es la integración de la información. La información de administración y gestión de una empresa, no debe estar separada del resto de la información, ya que es beneficiosa para otras áreas de la empresa y viceversa.

Finalmente, la internacionalización va a ser muy importante. En el país es necesario disponer de aplicaciones de gestión sin límites nacionales, que permitan realizar las operaciones de

negocio, allá donde se presenten. La globalización pide aplicaciones multi-lenguaje, multi-divisa, etc.

Conclusiones:

Durante este capítulo se han enunciado conceptos de proyectos de software, así como algunos antecedentes de los mismos, se ha hablado también de la planificación en estos proyectos, de la estimación dentro de la planificación, de los detalles necesarios para dar comienzo a un proyecto de software, de la estimación de la duración y del tamaño así como algunas técnicas comúnmente utilizadas para estimar, de algunos métodos y modelos de estimación de aplicaciones de software y por último se habló de métricas, de las aplicaciones de Gestión y de los nuevos retos que estas enfrentan hoy en día.

CAPÍTULO 2:

Introducción

Watts Humphrey plantea métodos para organizar, planificar y estimar la duración y el tamaño de las tareas de un proyecto de software, algunos de los cuales fueron mencionados en el capítulo anterior de forma general. Al ser algunos de sus criterios los más reconocidos a la hora de estimar duración de aplicaciones de software a nivel internacional, se ha dedicado este capítulo a profundizar en las teorías de este científico. También se hará referencia a los puntos de contacto resultantes entre las teorías de Boehm y Humphrey provenientes de la investigación hecha por la MSc. Yadira Ruíz Constanten que sirve como precedente a esta.

Entre los temas a tratar estarán: el PSP, la obra cumbre de Humphrey, cómo surge, características, principios, el flujo de procesos del Proceso Personal del Software (PSP), etc.; el método de Estimación Basado en Proxy (PROBE), que será abordado de forma más profunda, prácticamente en su totalidad y por último el estudio más detallado de los puntos de contacto obtenidos en la investigación previamente mencionada, los cuales son: factor de escala PMAT, Multiplicador de esfuerzo PREX, la Reutilización de Código y las Líneas de Código, así como las formas de uso de los mismos.

2.1 Proceso Personal del Software (PSP).

‘’PSP esta diseñado para su uso con cualquier lenguaje de programación o método de diseño y puede ser usado para la mayor parte de aspectos de trabajo incluyendo requerimientos de trabajo, correr pruebas, definir procesos y reparar defectos” [Mendoza, 1997].

PSP está conformado por siete procesos de desarrollo de software, divididos en cuatro niveles:

Nivel 0 – Bases del proceso personal (0.0 y 0.1). Esta es una introducción al proceso. Incluye ciertos conceptos básicos sobre métricas y planeación. En este nivel los desarrolladores hacen tres códigos que serán evaluados. Estos son libres de estilo y forma aunque deben de contener seis puntos que son: planeación, diseño, codificación, compilación, pruebas y resultados.

Nivel 1 – Administración del proceso personal (1.0 y 1.1). En este nivel los dos procesos se enfocan a las técnicas personales de administración de proyectos, incluyendo estimación de tiempo y esfuerzos, formulario de la calendarización y métodos de rastreos de tiempo. Este nivel usa un método llamado PROBE (Proxy Based Estimating) que hace uso de proxys y datos históricos para solucionar los estimados de tiempo y esfuerzo en líneas de código.

Nivel 2 – Administración personal de calidad (2.0 y 2.1). Este nivel añade procesos de calidad al proyecto: diseño personal y revisión de códigos, diseños de notaciones, verificación de técnicas y medidas para administrar un proceso y la calidad del producto. La meta en este nivel es erradicar cualquier error en el código antes de compilarlo. Esto se logra mediante la revisión del diseño y la del código. La idea de esta técnica no es crear un nuevo método de diseño, sino que el programador examine y documente el diseño desde otra perspectiva. PSP muestra cuatro enfoques para el diseño: operacional, funcional, estado y lógico.

Nivel 3 – Proceso personal cíclico (3.0). Este es el último de los niveles de procesos y busca enfocar a las personas a proyectos más grandes sin que esto fomente la pérdida de calidad y productividad en su trabajo. Para esto usan un modelo de ciclo de vida de ciclos, para descomponer el proyecto en partes a ser desarrolladas y después integrarlas. En el nivel 3, los ingenieros descomponen su proyecto en varios ciclos PSP 2.1 y luego integran y prueban la salida de cada ciclo. Debido a que ya se domina el proceso 2.1, los costos de integración y pruebas son minimizados.

La estructura del proceso del PSP conceptualmente comienza con los requerimientos, el primer paso en el proceso PSP es la planeación. Hay un guión de planeación que orienta este trabajo y un sumario del plan para registrar los datos de la planeación. Mientras los ingenieros están siguiendo el guión para hacer el trabajo, registran sus datos de tiempo y defectos en los formularios para tiempo y defectos. Al final del trabajo, durante la fase post-mortem (PM), ellos hacen un resumen de los datos de tiempo y defectos de los formularios, y de las medidas del tamaño del programa e incluyen esos datos en el formulario del resumen del plan. Cuando el trabajo está terminado ellos entregan el producto terminado junto con el formulario del plan completado.

2.1.1 Cómo surge el Proceso Personal del Software (PSP).

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. No fue sino hasta los años 70 y los años 80 que W. Edwards Deming y J.M. Juran convencieron a la industria estadounidense que se centrara en mejorar la forma en la que la gente hacía sus trabajos y desarrollaban sus procesos.

La estrategia tradicional que había de "prueba y arregla" ahora es reconocida como costosa, que desperdicia tiempo y que además es ineficaz para el trabajo de la ingeniería y de la fabricación.

Usando inspecciones, las organizaciones han mejorado considerablemente 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 les proporcionaba a los ingenieros de desarrollo. CMM ha tenido un efecto positivo en el funcionamiento de las organizaciones del software.

Mientras que los principios de CMM se aplicaron a pequeños 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.

Esto dio lugar a que, en el desarrollo de estos programas modulares, Humphrey utilizara personalmente todas las prácticas de CMM para que el mismo subiera poco a poco hasta llegar al nivel 5. Poco después comenzó a trabajar en el proyecto a tiempo completo en abril de 1989, el Instituto de la Ingeniería de Software (SEI) hizo a Humphrey su colaborador, permitiéndole trabajar tiempo completo en la investigación detallada de PSP.
Durante los siguientes tres años, Humphrey desarrolló un total de 62 programas y definió cerca de 15 versiones de PSP. Utilizó los siguientes lenguajes de programación: PASCAL y

C++, para desarrollar cerca de 25.000 líneas de código que ayudarían a darle la forma final a PSP.

De esta experiencia, él concluyó que los principios de la administración de procesos que desarrolló Deming y de Juran eran totalmente aplicables tanto al trabajo de los ingenieros de software de manera individual como a ingenieros enfocados al trabajo en equipo, el resultado de ello fue la aparición del Proceso en Equipo de Software (TSP).

2.1.2 Principios de PSP.

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 a que utilicen constantemente prácticas sanas de ingeniería de software.

Asimismo, les enseña 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, cómo analizar los resultados de cada trabajo, y cómo utilizar los resultados para mejorar el proceso del proyecto siguiente.

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.

Por lo tanto, PSP comienza a estimar los tamaños de los productos que los ingenieros desarrollan personalmente, basándose 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.

El diseño de PSP se basa en los siguientes principios de planeación y de calidad:

- ✓ Cada ingeniero es esencialmente diferente; para ser más precisos, los ingenieros deben planear su trabajo y basar sus planes en sus propios datos personales.

- ✓ Para mejorar constantemente su funcionamiento, los ingenieros deben utilizar personalmente procesos bien definidos y medidos.
- ✓ Para desarrollar productos de calidad, los ingenieros deben sentirse personalmente comprometidos con la calidad de sus productos.
- ✓ Cuesta menos encontrar y arreglar errores en la etapa inicial del proyecto que encontrarlos en las etapas subsecuentes.
- ✓ Es más eficiente prevenir defectos que encontrarlos y arreglarlos.
- ✓ La manera correcta de hacer las cosas es siempre la manera más rápida y más barata de hacer un trabajo.

Para hacer un trabajo de ingeniería de software de la manera correcta, los ingenieros deben planear de la mejor manera su trabajo antes de comenzarlo y deben utilizar un proceso bien definido para realizar de la mejor manera la planeación del trabajo.

Para desarrollar software de alta calidad, cada componente individual también debe de contar con la más alta calidad posible. La estrategia total de PSP es cerciorarse de que todos los componentes individuales se desarrollen con la más alta calidad. PSP logra esto proporcionando un marco de proceso personal ya definido que el programador puede utilizar. Este marco es:

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

Por otra parte para que los desarrolladores lleguen a entender su funcionamiento de manera personal, deben medir el tiempo que pasan en cada proceso, los defectos que inyectan y

remueven de cada proyecto y finalmente medir los diferentes tamaños de los productos que llegan a producir.

Para producir constantemente productos de calidad, los ingenieros deben planear, medir y rastrear constantemente la calidad del producto y deben centrarse en la calidad desde el principio de un trabajo. Finalmente, deben analizar los resultados de cada trabajo y utilizar estos resultados para mejorar sus procesos personales.

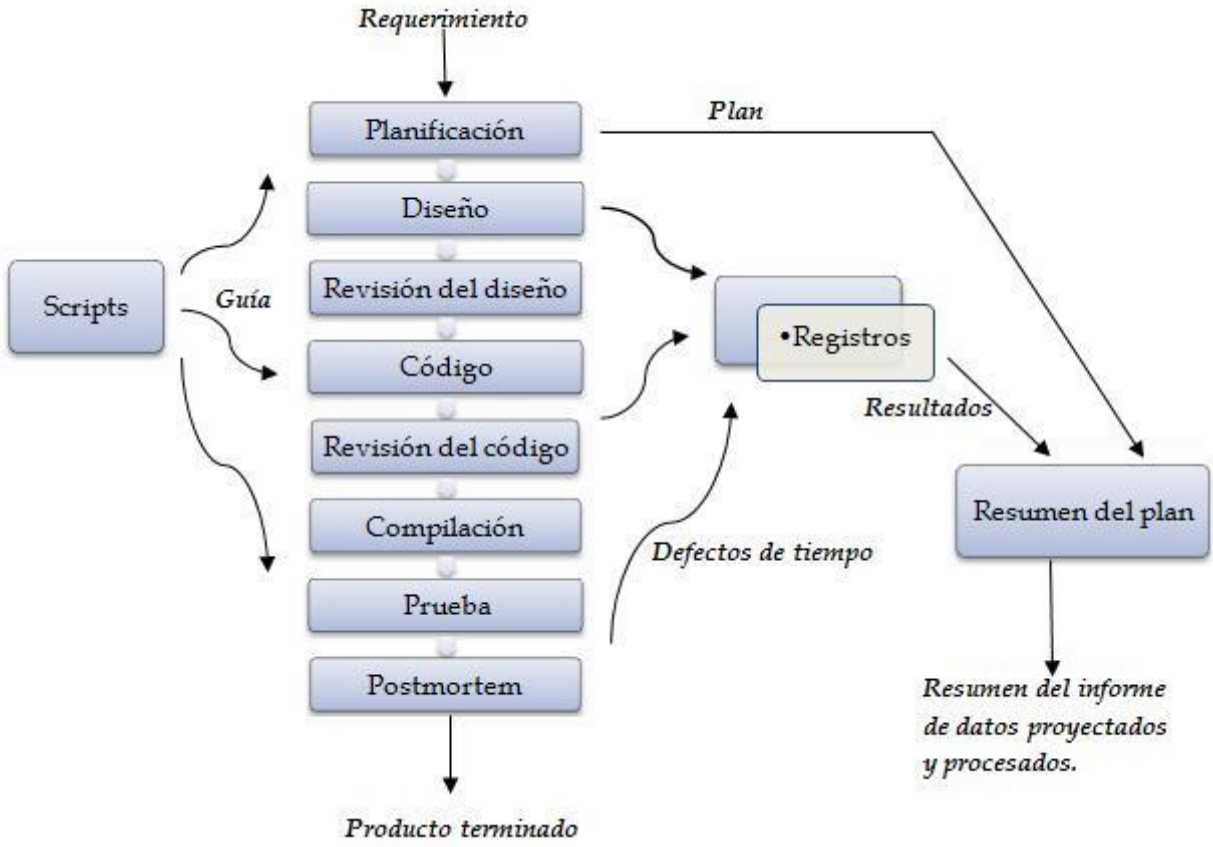


Figura 7: Flujo del Proceso PSP.

La estructura del proceso del PSP se muestra conceptualmente en la **Figura 7**. Comenzando con los requerimientos, el primer paso en el proceso PSP es la planeación. Hay un guión de planeación que guía este trabajo y un sumario del plan para registrar los datos de la planeación. Mientras los ingenieros están siguiendo el guión para hacer el trabajo, registran sus datos de tiempo y defectos en los formularios para tiempo y defectos. Al final del trabajo, durante la fase post-mortem (PM), ellos hacen un resumen de los datos de tiempo y defectos

de los formularios, y de las medidas del tamaño del programa e incluyen esos datos en el formulario del resumen del plan. Cuando el trabajo está terminado ellos entregan el producto terminado junto con el formulario del plan completado.

2.2 Método PROBE (Proxy-Based Estimating).

Un proxy es un sustituto, medida o estimador, de la unidad de tamaño, que proviene de elementos del producto; por ejemplo clases u objetos y eventualmente métodos. Otros ejemplos de proxys son las pantallas, archivos, guiones, y puntos de función. El método de estimación basado en Proxy (PROBE) le permite utilizar cualquier elemento que usted elija como proxy, siempre y cuando reúna los criterios para ser un buen proxy.

Los criterios de un buen indicador son las siguientes:

- ✓ La medida de tamaño del proxy debe relacionarse estrechamente con el esfuerzo necesario para desarrollar el producto.
- ✓ El proxy contenido de un producto debe ser automáticamente contable.
- ✓ El Proxy debe ser fácil de visualizar en el inicio de un proyecto.
- ✓ El proxy debe ser personalizable a las necesidades de cada proyecto y el desarrollador.
- ✓ El proxy debe ser sensible a las variaciones de aplicación que afectan al desarrollo, costo o esfuerzo.

El objetivo principal del proceso es producir estimaciones exactas. Para hacer esto, usted debe comenzar con los mejores requisitos que pueda tener y obtener buenos datos históricos. Debido a que en tempranas estimaciones probablemente tenga grandes errores, también es necesario analizar la manera de juzgar y mejorar la precisión de su estimación. El método PROBE muestra cómo obtener la estimación de los datos, la forma de utilizar los datos para hacer estimaciones y cómo medir y mejorar la precisión de sus estimaciones. En el método las medidas de tamaño de los objetos son normalizadas. Los objetos se organizan en categorías, para cada una de las cuales se definen rangos de tamaño.

En la siguiente figura se muestran los pasos que componen el método PROBE.

Figura 8: Gráfico de los pasos del método PROBE.

Este método cuenta también con 4 procedimientos para estimar según la cantidad de datos que usted tenga, ellos son: A, B, C, y D. El método A debería ser su primera elección, pero este requiere al menos tres, y de preferencia cuatro, puntos de datos de tamaño estimado de proxy (S) y el tamaño real añadido y modificado que se correlaciona con un r > = 0,7; donde r es el factor de correlación entre los puntos de datos. Si usted no puede usar el método A, trate de utilizar el método B. Este método usa el plan de tamaño agregado y modificado actual y el tamaño agregado y modificado. Una vez más, debería tener por lo menos tres, y preferiblemente cuatro, puntos de datos que se correlacionan con una r > = 0,7. Si los datos no son los adecuados para los métodos A y B, utilice entonces el método C si tiene por lo menos algunos datos sobre el plan y tamaño modificado y añadido. Si no dispone de todos los datos, debe utilizar el método D. Con el método D, usted no está realmente haciendo una proyección, sino que simplemente esta adivinando un valor para entrar como el tamaño agregado y modificado en el plan o el tiempo de desarrollo en el Resumen del Plan de Proyecto.

2.2.1 Realizar un Diseño Conceptual.

Para que la estimación del tamaño refleje adecuadamente el producto a construir se debe iniciar con un diseño conceptual. Este diseño establece una aproximación preliminar de diseño y nombra los objetos del producto y sus funciones. No se pretende realizar un diseño completo pero si postular los objetos que se necesitaran y las funciones que realizarán.

Los desarrolladores tienen que estar conscientes de que deben realizar un diseño conceptual que se adapte lo más fácilmente posible al proyecto. Esta conjetura es esencial, porque cada vez que los ingenieros tengan dudas respecto al proyecto, bastaría mirar este diseño para tener bien plantadas las bases de lo que se está creando.

Cuando se están estimando productos relativamente pequeños se puede producir un diseño conceptual directamente. Para productos más grandes será necesaria una técnica de diseño que permita subdividir el producto. Si alguna o todas las partes resultantes son aún demasiado grandes para permitir un diseño conceptual suficientemente detallado, entonces será necesario refinar la partición aun más. Se continua el proceso de refinamiento hasta alcanzar un nivel en el cual se puedan describir las funciones del producto en términos de objetos, estos últimos similares a los objetos almacenados en la base de datos histórica.

2.2.2 Estimación del Tamaño de las Partes.

Después de completar el diseño conceptual, se identifican las partes del producto que usted piensa desarrollar. Se estima el tamaño de las partes del diseño conceptual. Haga esto determinando los tipos de las partes y luego juzgando la cantidad de elementos (métodos) para cada una de las partes y los tamaños relativos de los elementos. Los elementos pueden ser muy pequeños (MP), pequeños (P), medianos (M), grandes (G), o muy grandes (MG). Si una parte tiene elementos de diferentes tipos, estime cada combinación parte-elemento como una parte separada.

Rango de Tamaño	**Fronteras en LOC**	**LOC por Método**
MP	-5.68	
		6
		6
P	7.16	
		8.333
		10.333
		12.333
		16.400

M	20.0	
		20.500
		21.750
		22.250
		23.000
		28.333
		29.000
G	32.84	
MG	45.68	
		55.800

Tabla 2: Tamaños de objetos organizados en rangos.

La idea consiste en calcular el promedio de LOC por métodos para cada objeto, se hace corresponder este valor con la frontera del rango de tamaño Medio, y luego se le resta y adiciona sucesivamente el valor de la desviación estándar, para obtener los valores por debajo y por encima de la frontera media respectivamente.

2.2.3 Determinar el tipo de objeto y su tamaño.

Las categorías de tamaños de los objetos son necesarias para establecer las bases de estimación de los nuevos objetos que componen los productos planificados. Debido a que existe un interés en la medida del tamaño relativo de los objetos basado en la consideración de su complejidad funcional, es útil normalizar el tamaño de los objetos, dividiendo la cantidad total de LOC de los objetos entre el número de métodos de cada objeto.

En el diseño conceptual son nombrados los objetos y se les asigna una categoría. El próximo paso consiste en encontrar los objetos de la base de datos histórica que más se asemejen a los objetos del diseño. Es entonces que cada objeto se estima comparándose su tamaño con el tamaño de los objetos de su misma categoría en la base de datos.

En la tabla que se muestra a continuación el rango de tamaño de los objetos es de 18 a 558 LOC con una proporción de alrededor de 30 a 1. Los rangos de tamaños son más útiles si son razonablemente pequeños.

Nombre del Objeto	Número de Métodos	LOC del Objeto	LOC por Método
each_line	3	31	10.333
each_char	3	18	6.000
list_clump	4	87	21.750
character	3	87	29.000
single_character	3	25	8.333
string_read	3	18	6.000
list_clp	4	89	22.250
char	3	85	28.333
single_char	3	37	12.333
converter	10	558	55.800
string_manager	4	82	20.500
string_builder	5	82	16.400
tring_decrementer	10	230	23.000

Tabla 3: Tabla de tamaño de objetos.

Debido a que existe un interés en la medida del tamaño relativo de los objetos basado en la consideración de su complejidad funcional, es útil normalizar el tamaño de los objetos, dividiendo la cantidad total de LOC de los objetos entre el número de métodos de cada objeto. De esta manera, si se tiene un objeto complejo con un único método, se podrá distinguir claramente de un objeto simple con muchos métodos.

2.2.4 Estimación del Tamaño de las Partes Reutilizadas y Adiciones de Base.

El próximo paso consiste en estimar el tamaño de cualquieras otras partes del programa (base, suprimida, reutilizadas, y así sucesivamente). Por ejemplo, si usted puede encontrar partes disponibles que proporcionen funciones necesarias por su diseño conceptual, es

posible que pueda volver a utilizarlas. Siempre que estas partes trabajen según lo previsto y sean de calidad adecuada, la reutilización de partes disponibles puede ahorrar un tiempo considerable.

PROBE considera dos tipos de partes reutilizadas. En primer lugar están las partes tomadas de una biblioteca de reutilización. Mientras que por otra parte las nuevas piezas reutilizables, las cuales son algunas de las partes añadidas que ya han sido estimadas. Identifique estas partes como nuevas partes reutilizables si planea utilizarlas en el desarrollo de otros programas. Es una buena práctica construir una biblioteca de partes útiles cuando se trabaja en un nuevo entorno de aplicación.

Aunque de vez en cuando los desarrolladores llegan a desarrollar programas enteramente nuevos, la mayor parte de este trabajo es la mejora de los programas existentes. Aquí, el tamaño del programa base es el tamaño total del programa sin modificar antes el desarrollo. Las adiciones de Base son para que realice mejoras a la base del programa. Si sabe lo suficiente sobre la base de producto como para hacerlo, estime las adiciones de base de la misma manera que la estimación de partes adicionadas.

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

La parte final de la etapa 3 es añadir todas las estimaciones de las partes para obtener el tamaño estimado del proxy (S), que se indicará en la Plantilla de Estimación de tamaño la cual se muestra a continuación. E es el número utilizado por PROBE para hacer las proyecciones de tamaño y de tiempo. La cantidad de LOC Objeto estimadas (E) para el nuevo programa se calculan con la formula:

$$E = BA + NO + M.$$

Los términos Adiciones Base (BA), Nuevos Objetos (NO), y LOC Modificadas se definen teniendo como referencia el formulario de datos presentado.

BA: LOC que han sido añadidas al programa de partida.

NO: LOC para nuevos objetos que se desarrollan.

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

Estudiante:		Fecha:		
Instructor:		# programa:		
Programa Base				LOC
Tamaño Base (B)				
LOC Eliminadas (D)				
LOC Modificadas (M)				
LOC Proyectadas				LOC
Adiciones Base	Tipo	Método	Tamaño	
Total de Adiciones Base (BA)				
Nuevos Objetos (NO)	Tipo	Método	Tamaño	LOC (Nuevo Reuso*)
Total de Nuevos Objetos (NO)				
Objetos Reutilizados				LOC
…..				…..

Total de Reutilizados (R)		LOC
LOC Proyectadas	P = BA + NO	
Parámetro de regresión	β0 Tamaño y Tiempo	
Parámetro de regresión	β1 Tamaño y Tiempo	
Tamaño proyectado añadido y modificado (P)	P = β0 + β1*(E)	
Tamaño total estimado (T)	T = P + B – D – M + R	
Estimado total de nuevos reutilizables (NR)	suma de *LOC	
Estimado total de tiempo de desarrollo	tiempo=β0+β1*E	
Intervalo de Predicción	Rango	
Intervalo de Predicción Superior	IPS = N + Rango	
Intervalo de Predicción Inferior	IPI = N - Rango	
Porciento del Intervalo de Predicción	%	

Tabla 4: Formulario de datos para el método PROBE.

Los datos de entrada se describen a continuación:

Programa Base. Al mismo tiempo que se estiman los objetos también se determina el tamaño del programa base (código existente) que se está mejorando y cualquier cambio que se le realice.

Objetos Reutilizados. Para cada objeto que se planee reutilizar se anota su nombre y tamaño en la sección de objetos reutilizados.

Nuevos de Reuso. Son objetos que se van a desarrollar pero que se piensan reutilizar en el futuro, por lo que serán añadidos a la librería de reutilización.

2.2.5 El procedimiento de estimación de tamaño.

En el paso 4 de PROBE, con el tamaño estimado de proxy E, se puede calcular el tamaño proyectado de programa P y el tiempo total estimado de desarrollo. Si, por ejemplo, los datos históricos muestran que sus programas acabados fueron en general alrededor del 25% más largos de lo estimado, podría añadir un 25% para cada factor agregado de estimación.

El método PROBE esencialmente lo hace pero de una manera estadísticamente sólida. Utiliza para ello el método de regresión lineal. Aunque estos cálculos son un poco más complejos que tomando promedios simples, utilizan los datos históricos para producir una estimación estadísticamente sólida. Los parámetros β_0 y β_1 se utilizan en la siguiente ecuación para calcular el tamaño proyectado añadido y modificado:

Tamaño proyectado Añadido y Modificado (P) = $\beta_0 + \beta_1 * E$

Cuando dos conjuntos de datos están estrechamente relacionados, puede utilizar el método de regresión lineal para representar esa relación. El estimado de tamaño de la parte y el tamaño actual añadido y modificado son a menudo estrechamente correlacionados. Esto quiere decir que la regresión lineal es adecuada. Los parámetros β_0 y β_1 se calculan a partir de sus datos históricos.

$$\beta_1 = \frac{\sum_{i=1}^{n} x_i * y_i - n * x_{prom} * y_{prom}}{\sum_{i=1}^{n} x_i^2 - n(x_{prom})^2}$$

$$\beta_0 = y_{prom} - \beta_1 * x_{prom}$$

Los programas terminados suelen contener algo más que las partes que se especifican en el diseño conceptual. Por ejemplo, es probable que tengan un código de declaración y encabezado que no está incluido en las estimaciones de las partes. Para darse cuenta de este y cualquier otro código, se debe utilizar un factor que se basa en su experiencia histórica. Afortunadamente, el método PROBE cuenta con este factor a la hora en que se calculan los parámetros β_0 y β_1.

2.2.6 El procedimiento de estimación de tiempo.

En el paso 5 de PROBE, una vez que haya obtenido los valores de β de tiempo y el tamaño estimado proxy (E) úselos para calcular el tiempo estimado de desarrollo, utilizando la siguiente ecuación:

$$Tiempo = \beta_0 + \beta_1 * \mathrm{E}$$

A continuación, introduzca el tamaño y las estimaciones de tiempo en los espacios del plan en el Resumen de Plan de Proyecto.

2.2.7 El intervalo de Predicción.

La calidad de una estimación hecha con el método PROBE es directamente proporcional a la calidad de los datos utilizados. También depende del grado en que estos datos se corresponden a la manera en que se intentará desarrollar el próximo programa. Si la base de datos históricos tiene variaciones bruscas, descontroladas; el intervalo de predicción será grande.

Si se cambia el método de diseño, se construye un programa mucho mayor, o se desarrolla una nueva clase de aplicación, los datos históricos no representarán con precisión lo que se intenta hacer. Se tiene que reconocer que el intervalo de predicción no es un buen indicador del rango de error de la estimación realizada.

Los cálculos finales en la Plantilla de Estimación de Tamaño en PROBE son para el intervalo de predicción. El intervalo de predicción es una gama determinada estadísticamente alrededor de su tamaño o la estimación del tiempo, en el que el valor real es probable que caiga. Para un intervalo de predicción del 70%, era de esperar que el tamaño real y los valores de tiempo cayeran fuera de este rango alrededor de 30% del tiempo.

Una vez hecha una estimación se necesita medir su calidad. El intervalo de predicción se calcula utilizando datos históricos y la distribución probabilística t del Estudiante. Este intervalo da el rango alrededor de la estimación en el que el tamaño actual del programa se hallará probablemente.

La fórmula del intervalo de predicción es:

$$Rango = t(\alpha/2, n-2)\sigma\sqrt{1+\frac{1}{n}+\frac{(x_k - x_{prom})^2}{\sum_{i=1}^{n}(x_i - x_{prom})^2}}$$

Xi: número de LOC Objeto estimadas para cada programa en la base de datos históricos.

Xprom: es el promedio de LOC Objeto estimadas para cada programa en la base de datos históricos.

n: cantidad de programas almacenados en la base de datos históricos.

Xk: LOC Objetos estimadas en el nuevo programa.

α/2: % utilizado para el intervalo de predicción.

σ: desviación estándar de los datos alrededor de la línea de regresión.

$$Varianza = \sigma^2 = (\frac{1}{n-2})\sum_{1-1}^{n}(y_i - \beta_0 - \beta_1 * x_i)^2$$

$$DesvEst = \sqrt{Varianza} = \sigma$$

2.3 Puntos de Contacto entre las Teorías de Boehm y Humphrey.

Después de haber realizado la investigación y tener conocimiento del tema, se pueden abordar los puntos de contacto obtenidos en el trabajo de la MSc. Yadira Ruíz Constanten entre las teorías de Boehm y Humphrey para la estimación de la duración de proyectos de software y que fueron enunciados en la introducción de este capítulo.

Al analizar estos puntos de contacto se persigue la intención de plantear una posible integración entre los métodos que diseñaron estos científicos para estimar proyectos de software.

2.3.1 Líneas de Código.

La definición de línea de código, aunque es básica para muchas métricas del software, es ambigua. El significado de línea de código varía de un lenguaje a otro, pero también dentro de un mismo lenguaje de programación. En el lenguaje de programación C, por ejemplo, una línea de código puede ser, una instrucción acabada en un salto de línea, una instrucción acabada en un punto y coma o cualquier línea del programa que acabe en un salto de línea.

Habitualmente en cada línea se ejecuta una instrucción que tiene que ejecutar el software programado. También es habitual tabular las estructuras de control del programa en cuestión para una lectura más fácil. En ocasiones los programadores hablan del número de líneas de código que tiene cierto programa para hablar de la magnitud o complejidad de este.

En informática la cantidad de líneas de código de una instrucción es un punto bastante útil a la hora de compilar el programa ya que habitualmente los compiladores detectan errores mostrando el número de línea donde se ha encontrado el error que el programador deberá corregir para una compilación satisfactoria.

En la actualidad no existe una definición estándar de lo es una línea de código, ya que diversos actores ofrecen conceptos distintos, en los que algunos por ejemplo incluyen los comentarios y las etiquetas de los lenguajes de programación web a la hora contar las líneas de código, otros no las toman en cuenta.

Definiciones de LDC:
Las variantes a nivel de programas son:

- ✓ Contar solo las líneas de código ejecutable.
- ✓ Líneas de código más definiciones de datos.
- ✓ Líneas de código, definiciones de datos y comentarios.
- ✓ Líneas de código, definiciones de datos, comentarios y lenguaje de control.
- ✓ 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.

En el método PROBE cimentado por Humphrey, las líneas de código juegan un papel importante porque forman la base de la que se parte a la hora de calcular o estimar la duración que puede tener un proyecto de software. Se pueden calcular las líneas de código mediante este método de la siguiente forma:

$$E = BA + NO + M$$

La cantidad de LOC Objeto estimadas (E) para el nuevo programa se calculan con la fórmula planteada, contenido que está más abundado en el epígrafe 2.2.4.

El próximo paso sería el de calcular el tamaño proyectado de programa (P), contenido que se encuentra más explicado en el epígrafe 2.2.5.

Tamaño proyectado Añadido y Modificado $(P) = \beta_0 + \beta_1 * E$

Para entonces determinar el tiempo estimado de duración aplicando COCOMO II.

COCOMO II está compuesto por tres modelos denominados: Composición de Aplicación, Diseño Temprano y Post-Arquitectura. Los tres modelos se adaptan tanto a las necesidades de los diferentes sectores, 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.

Los modelos de Diseño Anticipado 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:

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

Si se sustituye el valor *Size* que representa el tamaño en KSLOC, por (P/1000), donde (P) representa el valor del Tamaño proyectado Añadido y Modificado que se calcula aplicando el método PROBE, se logra una posible integración entre ambas teorías.

Donde 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 y es el esfuerzo en términos de Persona-Mes nominal.

PMnominal se utiliza junto con los drivers de coste para calcular el esfuerzo estimado (PMestimado), 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). He aquí la diferencia entre el modelo de Diseño Anticipado y el Post-Arquitectura.

- ✓ Ecuación para calcular el PMestimado en el Modelo de Diseño Anticipado.

$$\boldsymbol{PM}_{estimado} = \boldsymbol{PM}_{nominal} * \prod_{i=1}^{7} EM_i$$

- ✓ Ecuación para calcular el PMestimado en el Modelo Post-Arquitectura.

$$\boldsymbol{PM}_{estimado} = \boldsymbol{PM}_{nominal} * \prod_{i=1}^{17} EM_i$$

Teniendo todos estos datos se procede a calcular el tiempo en meses (TDEV) 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 con la siguiente ecuación:

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

Donde 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.

2.3.2 Factor de Escala PMAT.

Una organización de software madura, posee habilidad y capacidad en el conjunto de la organización para gestionar procesos de desarrollo y mantenimiento de software. Comunica el proceso de software con precisión tanto al personal existente como a los nuevos empleados y lleva a cabo actividades de trabajo de acuerdo al proceso planificado. Los procesos están documentados y son utilizables y consistentes con la forma en la que el trabajo es hecho en cada momento. Las definiciones de los procesos son actualizadas cuando es necesario, y las mejoras a esos procesos son desarrolladas a través de pilotos y pruebas controlados y análisis de costes / beneficios.

En una organización madura, los gestores monitorizan tanto la calidad de los productos software como los procesos que los producen. Hay una base cuantitativa objetiva para juzgar la calidad del producto y analizar los problemas tanto del producto como del proceso. Las planificaciones y presupuestos están basados en realizaciones históricas y son realistas, siendo normal alcanzar los resultados esperados en cuanto a coste, planificación, funcionalidad y calidad de los productos.

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

El procedimiento para determinar PMAT 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. De esta manera se pueden encontrar dos formas de medir la madurez del proceso.

Formas de medir la madurez del proceso:

1. La primera captura el nivel de madurez de la organización, resultado de la evaluación según CMM y asignándole el valor correspondiente en la **Tabla 5**.

Nivel de CMM	**PMAT**
1 – Mitad inferior	Muy Bajo
1 – Mitad superior	Bajo
2 -- Repetible	Nominal
3 -- Definido	Alto
4 -- Gestionado	Muy Alto
5 -- Optimizado	Extra Alto

Tabla 5: Factor PMAT de acuerdo al nivel de CMM.

El CMM es un modelo de calidad del software que clasifica las empresas en niveles de madurez. Estos niveles sirven para conocer la madurez de los procesos que se realizan para producir software.

Los niveles CMM son 5:

Inicial o Nivel 1 CMM: Este es el nivel en donde están todas las empresas que no tienen procesos. Los presupuestos se disparan y no es posible entregar el proyecto en fechas. No hay control sobre el estado del proyecto, el desarrollo del proyecto es completamente opaco, no sabes lo que pasa en él.

Las organizaciones en este nivel no disponen de un ambiente estable para el desarrollo y mantenimiento de software. Aunque se utilicen técnicas correctas de ingeniería, los esfuerzos se ven minados por falta de planificación. El éxito de los proyectos se basa la mayoría de las

veces en el esfuerzo personal, aunque a menudo se producen fracasos y casi siempre retrasos y sobrecostes. El resultado de los proyectos es impredecible.

Repetible o Nivel 2 CMM: Quiere decir que el éxito de los resultados obtenidos se pueden repetir. La principal diferencia entre este nivel y el anterior es que el proyecto es gestionado y controlado durante el desarrollo del mismo. El desarrollo no es opaco y se puede saber el estado del proyecto en todo momento.

Los procesos que hay que implantar para alcanzar este nivel son:

- ✓ Gestión de requisitos
- ✓ Planificación de proyectos
- ✓ Seguimiento y control de proyectos
- ✓ Gestión de proveedores
- ✓ Aseguramiento de la calidad
- ✓ Gestión de la configuración

En este nivel las organizaciones disponen de unas prácticas institucionalizadas de gestión de proyectos, existen unas métricas básicas y un razonable seguimiento de la calidad. La relación con subcontratistas y clientes está gestionada sistemáticamente.

Definido o Nivel 3 CMM: Resumiéndolo, alcanzar este nivel significa que la forma de desarrollar proyectos (gestión e ingeniería) está definida, por definida se debe entender, que está establecida, documentada y que existen métricas (obtención de datos objetivos) para la consecución de objetivos concretos.

Además de una buena gestión de proyectos, a este nivel las organizaciones disponen de correctos procedimientos de coordinación entre grupos, formación del personal, técnicas de ingeniería más detallada y un nivel más avanzado de métricas en los procesos.

Los procesos que hay que implantar para alcanzar este nivel son:

- ✓ Desarrollo de requisitos
- ✓ Solución Técnica

- ✓ Integración del producto
- ✓ Verificación
- ✓ Validación
- ✓ Desarrollo y mejora de los procesos de la organización
- ✓ Definición de los procesos de la organización
- ✓ Planificación de la formación
- ✓ Gestión de riesgos
- ✓ Análisis y resolución de toma de decisiones

La mayoría de las empresas que llegan al nivel 3 paran aquí, ya que es un nivel que proporciona muchos beneficios y no ven la necesidad de ir más allá porque tienen cubiertas la mayoría de sus necesidades.

Cuantitativamente Gestionado o Nivel 4 CMM: Los proyectos usan objetivos medibles para alcanzar las necesidades de los clientes y la organización. Se usan métricas para gestionar la organización.

Se caracteriza porque las organizaciones disponen de un conjunto de métricas significativas de calidad y productividad, que se usan de modo sistemático para la toma de decisiones y la gestión de riesgos. El software resultante es de alta calidad.

Los procesos que hay que implantar para alcanzar este nivel son:

- ✓ Gestión cuantitativa de proyectos
- ✓ Mejora de los procesos de la organización

Optimizado o Nivel 5 CMM: Los procesos de los proyectos y de la organización están orientados a la mejora de las actividades. Mejoras incrementales e innovadoras de los procesos que mediante métricas son identificadas, evaluadas y puestas en práctica. La organización completa está volcada en la mejora continua de los procesos. Se hace uso intensivo de las métricas y se gestiona el proceso de innovación.

Los procesos que hay que implantar para alcanzar este nivel son:

- ✓ Innovación organizacional

- ✓ Análisis y resolución de las causas

Normalmente las empresas que intentan alcanzar los niveles 4 y 5 lo realizan simultáneamente ya que están muy relacionados. La implantación de un modelo de estas características es un proceso largo y costoso que puede costar varios años de esfuerzo. Aun así el beneficio obtenido para la empresa es mucho mayor que lo invertido.

2. La segunda está basada en las 18 Áreas de Procesos Claves (KPAs) del modelo del SEI.

En el CMM, hay en total 18 áreas de procesos clave y cada una de ellas está asociada con un sólo nivel de madurez. No hay ninguna KPA asociada al nivel 1, el cual es el nivel de partida y no tiene ningún requisito previo. Los otros niveles de madurez tienen de 2 a 7 KPAs cada uno. Lista de áreas de proceso clave:

<u>En el Nivel 2</u>

- ✓ Gestión de requisitos
- ✓ Planeamiento del proyecto de Software
- ✓ Seguimiento y supervisión del proyecto de software
- ✓ Gestión de subcontratos de software
- ✓ Garantía de calidad de software
- ✓ Gestión de configuración del software

<u>En el Nivel 3</u>

- ✓ Enfoque en el proceso de organización
- ✓ Definición del proceso de la organización
- ✓ Programa de entrenamiento
- ✓ Gestión integrada del software
- ✓ Ingeniería del producto de software
- ✓ Coordinación entre grupos
- ✓ Revisión de Pares

<u>En el Nivel 4</u>

- ✓ Gestión cuantitativa del Proceso

- ✓ Gestión de la calidad del software

En el Nivel 5

- ✓ Prevención de defectos
- ✓ Gestión de cambio de tecnología
- ✓ Gestión del cambio del proceso

Se considera que una organización ha alcanzado un nivel dado de madurez cuando ha cumplido con todos los objetivos asociados con cada una de las aéreas de proceso clave desde el nivel 2 hasta el nivel identificado.

El procedimiento para determinar el PMAT es establecer el porcentaje de cumplimiento de cada una de las Áreas evaluando el grado de cumplimiento de las metas correspondientes. Para este procedimiento se emplea la **Tabla 6**.

Áreas de Procesos Claves	**Casi Siempre (90%)**	**A menudo (60-90%)**	**La mitad de las veces (40-60%)**	**Ocasionalmente (10-40%)**	**Casi nunca (<10 %)**	**No se aplica**	**No se conoce**
Administración de Requerimientos							
Planificación del Proyecto de Software							
Seguimiento y supervisión del Proyecto de Software							
Administración de Subcontratos							
Aseguramiento de							

la Calidad							
Administración de la Configuración							
Objetivo del Proceso de Organización							
Definición del Proceso de Organización							
Programa de Entrenamiento							
Ingeniería del Producto							
Administración Integrada de Software							
Coordinación entre Grupos							
Revisión por Pares							
Administración Cuantitativa							
Administración de la Calidad							
Prevención de Defectos							
Administración de las Tecnologías de Cambio							
Administración de los Procesos de							

Cambio							

Tabla 6: Nivel de cumplimiento de los objetivos de cada KPA.

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

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

Una vez que se aplica este modelo evaluativo, se encuentran algunas dificultades que podrían atentar contra la veracidad de los resultados obtenidos, por ejemplo si las personas encargadas de establecer los porcentajes para cada una de las KPA´s, no poseen los conocimientos necesarios para fijar valores que reflejen la situación real de su empresa, o si por el contrario, tienen los conocimientos y la experiencia necesaria para fijar valores reales y se percatan de que se pudiera disminuir el esfuerzo si aumentan los porcentajes de cumplimiento en algunas de las distintas KPA's.

Con el objetivo de eliminar estas posibles deficiencias y otras que pudieran surgir, es que se propone otra integración entre las teorías de Boehm y Humphrey. Partiendo de que PSP es un proceso de aplicación de los principios de CMM para ayudar y guiar a los ingenieros a realizar bien su trabajo y que cubre parcialmente 12 de las 18 KPA's que define CMM, siendo estas capas las que están orientadas a los procesos que puede dársele un enfoque personal (Véase Anexo_1). Se propone entonces la integración de las buenas prácticas que brinda PSP para el trabajo de los ingenieros dentro del equipo de desarrollo, con el objetivo de optimizar el factor PMAT.

2.3.3 Multiplicador de Esfuerzo PREX.

COCOMO II obtiene los datos necesarios para el ajuste del esfuerzo nominal considerando un conjunto de Multiplicadores de Esfuerzo (ME), los cuales representan las características del proyecto y expresan su impacto en el desarrollo total del producto de software. Cada uno de los modelos de estimación (Diseño preliminar y Post arquitectura) tiene un conjunto de

Multiplicadores de esfuerzo, los cuales son acordes con la información que se maneja en cada uno de estos modelos.

Los multiplicadores de esfuerzo son 17, los cuales están relacionados con el producto, con la plataforma de desarrollo, con el personal y con el proyecto en cuestión. COCOMO II sugiere que muchos de los 17 multiplicadores de esfuerzo no pueden estimarse en el diseño temprano, por lo que los reduce a 7 multiplicadores, estos son:

- ✓ *RCPX:* Confiabilidad y complejidad del producto.
- ✓ *RUSE:* Nivel de reutilizabilidad del desarrollo.
- ✓ *PDIF:* Dificultad de uso de la plataforma.
- ✓ *PERS:* Capacidad del personal de desarrollo.
- ✓ *PREX:* Experiencia del personal de desarrollo.
- ✓ *FCIL:* Facilidades de desarrollo.
- ✓ *SCED:* Exigencias sobre el calendario.

(PREX) se basa en la Experiencia Personal, es uno de los parámetros de coste del Diseño Temprano que combina los 3 parámetros de coste del modelo Post-Arquitectura siguientes: Experiencia (APEX), Experiencia en la Plataforma (PLEX) y Experiencia en el Lenguaje y Herramientas (LTEX). La **Tabla 7** asigna valores PREX en el rango correspondiente.

	Extra Bajo	**Muy Bajo**	**Bajo**	**Nominal**	**Alto**	**Muy Alto**	**Extra Alto**
Experiencia en aplicaciones, plataforma, lenguaje y herramienta	3 Meses	5 Meses	9 Meses	1 Año	2 Años	4 Años	6 Años

Tabla 7: Niveles de medida PREX.

El 70% de los costos en el desarrollo de software se atribuye a costos personales, habilidades, experiencia y los hábitos de cada ingeniero determinan los resultados en el

desarrollo de software. Esta relación del ingeniero con los resultados del proceso de desarrollo de software es la premisa en la que se basa el Proceso Software Personal.

Utilizando el PSP se obtienen datos que muestran la efectividad del trabajo y se identifican los puntos fuertes y las debilidades. Además se practican habilidades y métodos que ingenieros del software van a desarrollar durante muchos años de pruebas y errores, poniendo en práctica la experiencia adquirida en el desarrollo de proyectos de software.

Después de examinar todo lo anterior se opina que se debe robustecer el trabajo personal del ingeniero aplicando la PSP que forma parte de la teoría de Humphrey, esperando con ello se definan y mejoren los indicadores personales que agrupa el Driver de Coste PREX, de modo que se contribuya a disminuir el esfuerzo en el proceso de desarrollo y a su vez el tiempo de terminación de los proyectos.

Humphrey plantea que lo creativo y rutinario debe ser tratado separadamente, y una vez que se limite lo que necesita originalidad, se haga lo rutinario eficiente. Así se podrá optimizar el trabajo, y cuando se tenga que hacer una tarea parecida a alguna desarrollada anteriormente, se encontrarán con la grata sorpresa de que ya existe un proceso o por lo menos uno muy parecido con posibilidades reales de adaptarse.

2.3.4 Reutilización de Código.

La reutilización de código se refiere al comportamiento y a las técnicas que garantizan que una parte o la totalidad de un programa informático existente, se pueda emplear en la construcción de otro programa. De esta forma se aprovecha el trabajo anterior, se economiza tiempo, y se reduce la redundancia.

La manera más fácil de reusar código es copiarlo total o parcialmente desde el programa antiguo al programa en desarrollo. Pero es trabajoso mantener múltiples copias del mismo código, por lo que en general se elimina la redundancia dejando el código reusable en un único lugar, y llamándolo desde los diferentes programas. Este proceso se conoce como abstracción. La abstracción puede verse claramente en las bibliotecas de software, en las que se agrupan varias operaciones comunes a cierto dominio para facilitar el desarrollo de programas nuevos. Hay bibliotecas para convertir información entre diferentes formatos

conocidos, acceder a dispositivos de almacenamiento externos, proporcionar una interfaz con otros programas, manipular información de manera conocida (como números, fechas, o cadenas de texto).

Para que el código existente se pueda reusar, debe definir alguna forma de comunicación o interface. Esto se puede dar por llamadas a una subrutina, a un objeto, o a una clase. En resumen, la reutilización de código no sólo permite crear programas menos propensos a errores, sino que es fundamental para obtener diseños flexibles y robustos ante posibles cambios de requerimientos.

Estudios realizados desde finales de la década de los 80 plantean la necesidad de que los métodos que trabajan o tienen alguna vinculación con procesos de estimación deben ser no lineales, basados entre otros aspectos por la posibilidad de manejar el reuso de las líneas de código, pues para ello:

- ✓ Existe un costo base, de alrededor de un 5%, que contempla la evaluación, selección, y asimilación del componente reusable.
- ✓ Pequeñas modificaciones generan desproporcionadamente grandes costos. Esto se debe al esfuerzo por comprender el software a ser modificado, testear y chequear las interfaces.

Además, existe un efecto no lineal en el costo del reuso debido al chequeo de interfaces que debe realizarse durante el desarrollo del software modificado. Estos inconvenientes pueden reducirse si el software está apropiadamente estructurado [Adriana Gómez, 1994].

Estos son elementos que hacen del método COCOMO II el más documentado y utilizado en el mundo, pues usa un modelo no lineal para estimar el tamaño del software cuando éste incluye componentes reusables, permitiendo tener en cuenta si un proyecto de software va a ser construido a partir de componentes existentes. Para ello, reemplaza en la ecuación de estimación de esfuerzo el parámetro KSLOC por el KESLOC, que representa la cantidad equivalente de nuevas líneas de código a desarrollar. Las ESLOC se calculan de la siguiente forma:

$$\boldsymbol{AAF} = 0.4(DM) + 0.3(CM) + 0.3(IM)$$

$$\mathbf{ESLOC} = \frac{\mathrm{ASLOC}[\mathrm{AA} + \mathrm{AAF}(1 + 0.02(\mathrm{SU})(\mathrm{UNFM}))]}{100} \text{ , AAF} \leq 0.5$$

$$\mathbf{ESLOC} = \frac{\mathrm{ASLOC}[\mathrm{AA} + \mathrm{AAF} + (\mathrm{SU})(\mathrm{UNFM})]}{100} \text{ , AAF} > 0.5$$

Donde:

ASLOC: Cantidad de líneas de código fuente del software existente usadas para desarrollar el nuevo producto.

DM: Porcentaje del diseño del software que requiere modificación para alcanzar los objetivos del nuevo software a desarrollar.

CM: Porcentaje del código del software que requiere modificación para lograr los objetivos del nuevo software a desarrollar.

IM: Porcentaje del esfuerzo requerido para integrar y testear el software adaptado al producto global.

SU: Porcentaje de comprensibilidad del software existente. Se determina en función a tres características: estructura, claridad y descriptividad.

AA: Grado de Evaluación y Asimilación. Porcentaje de esfuerzo necesario para determinar si un módulo de software a adaptar es apropiado a la aplicación, como así también para integrar su descripción a la descripción total del producto.

UNFM: Nivel de familiaridad del programador con el software.

Por su parte Humphrey en su método PROBE también propone fórmulas para tener en cuenta la influencia de la reutilización del código, en el tamaño estimado del programa (Véase **Tabla 4**). Además reafirma la idea de que siempre que estas partes trabajen según lo previsto y

sean de calidad adecuada, la reutilización de partes disponibles puede ahorrar un tiempo de desarrollo considerable, todo este contenido esta expuesto de forma más explicita en el epígrafe 2.2.4.

Todo este estudio permitió proponer una integración entre ambas teorías, para ello se pueden tener en cuenta dos opciones, la primera: utilizar el método PROBE para calcular las líneas de código reutilizadas y luego con este valor se aplica COCOMO II para calcular el tamaño y tiempo estimado; la segunda: haciendo el proceso inverso, calcular el tamaño estimado incluyendo las líneas reutilizadas mediante COCOMO II y con este valor calcular el tiempo total estimado aplicando el método PROBE.

Conclusiones:

En el desarrollo de este capítulo se han abordado contenidos como: el Proceso Personal del Software, cómo surge el mismo, los principios en los que esta basado, entre otras cosas; el método PROBE, con los pasos que consta así como las operaciones que se realizan internamente en ellos y demás detalles del mismo; y por último, los puntos de contacto entre las teorías de Boehm y Humphrey, explicando a fondo cada uno de ellos con el objetivo de argumentar la posible integración entre ambas teorías.

CAPÍTULO 3:

Introducción:

En la investigación, hasta el momento, se realizó un estudio de las teorías de Boehm y Humphrey y de los puntos de contacto encontrados entre ambas. Además se estudiaron también las características de los procesos de estimación para aplicaciones de software de Gestión en la UCI. Este capítulo tiene como objetivo validar la integración entre los elementos comunes de las teorías de Boehm y Humphrey en los proyectos de aplicaciones de gestión a través de la propuesta realizada en el Capítulo 2. Con el fin de alcanzar esta meta, se llevó a cabo la selección de algunos especialistas, los cuales fueron entrevistados para obtener algunos datos reales acerca de cómo se desarrolla la estimación de duración de este tipo de proyectos en la UCI, así como para saber la opinión que merece la integración que se plantea en esta investigación a sus criterios y obtener argumentos que validen la misma.

3.1 Aspectos a tener en cuenta para seleccionar a los especialistas.

Como tema fundamental a la hora de escoger los candidatos a ser entrevistados se tuvo en cuenta que al menos llevasen tres o más años vinculados directamente a la producción de software de Gestión y que hubieran obtenido buenos resultados en esta área, tales como: participación en eventos de calidad, ponencias en eventos científicos, participación en seminarios, cátedras, conferencias, talleres, etc., que evidencien conocimiento del tema en cuestión. Además los especialistas que serían entrevistados deben o debieron haber estado vinculados a algún rol dentro de sus proyectos que participe en la realización de los procesos de estimación de la duración de los mismos. Estos especialistas debían tener conocimientos acerca de:

- ✓ Mejora de procesos de software.
- ✓ Planificación de proyectos.
- ✓ Modelos de estimación.
- ✓ Proceso de Software Personal (PSP).
- ✓ Calidad y Gestión del Software.

3.2 Elaboración de la entrevista.

La entrevista confeccionada (Ver Anexo_2) consta de dieciocho preguntas divididas en cuatro temáticas que recogen los elementos fundamentales tratados en la investigación agrupando algunos por puntos de vistas y argumentos, por parte de los especialistas, en determinadas preguntas. Para la elaboración del cuestionario se tuvieron en cuenta las características básicas que debería cumplir la propuesta presentada para su aplicación en la producción de software de Gestión en la UCI.

El documento fue elaborado con preguntas abiertas, las cuales brindan la posibilidad de proporcionar una mayor riqueza en las respuestas que son ofrecidas por los especialistas. También se les dio la posibilidad, además de exhortarlos a hacerlo, de presentar su opinión general acerca de los beneficios o dificultades que pudiera presentar la aplicación de la integración propuesta entre las teorías de Boehm y Humphrey en los proyectos productivos de la UCI.

3.3 Resultados de las entrevistas.

Las entrevistas realizadas condujeron a los siguientes resultados según las temáticas abordadas:

3.3.1 Medir conocimiento y aplicación de estimación de SW.

El 100 % de los especialistas consultados realizan o han realizado estimaciones en los proyectos, teniéndolas en cuenta para la planificación. Todos consideran que la estimación es un factor determinante en la planificación de los proyectos y que existe una fuerte correlación entre la estimación realizada y las tareas específicas a realizar día a día por el equipo de proyecto. El objetivo de la planificación es proporcionar un marco de trabajo que permita al gestor hacer estimaciones razonables de recursos, costos y planificación temporal. Agregan también que la estimación y la planificación son actividades relacionadas pero difieren en su alcance y propósito. La estimación normalmente está orientada al proyecto en su conjunto, mientras que la planificación esta dirigida a los individuos.

Un 80 % de los entendidos que trabajan realizando las estimaciones en un período de tiempo limitado al comienzo de un proyecto de software, opinan que estas deberían actualizarse

regularmente a medida que progresa el mismo debido a que en la fase de inicio es necesario tener una idea acerca de los recursos y el tiempo que se van a emplear en su desarrollo y con ello poder administrar y planificar las actividades del proyecto para optimizar los resultados. Además las estimaciones deberían definir los escenarios del mejor y el peor caso, de modo que los resultados del proyecto puedan limitarse; esto se realiza con el objetivo de tener una cuota inferior y superior para prevenir lo mejor y lo peor que pudiera suceder en el desarrollo del proyecto y así estar más preparados para enfrentar futuros errores. Solo el 20% de los entrevistados realizan el proceso de estimación después de la etapa de Arquitectura, criterio con el cual se esta de acuerdo debido a que en este momento se tiene una mejor idea de lo que realmente se persigue con el desarrollo del proyecto, así como los recursos y el tiempo necesarios para el mismo. Este proceso de estimación en la generalidad de los casos visitados se lleva a cabo por los líderes de proyecto. En la UCI, le ha sido asignada esta responsabilidad a los planificadores quienes poseen los conocimientos necesarios y la mayor experiencia para realizar esta actividad. Este rol, existente en un equipo de desarrollo de software, es el encargado de: aprender de los actores locales y reconocer y respetar sus conocimientos, en vez de imponer ideas; no aplicar un proceso de aprendizaje predeterminado y rígido, sino flexible, explorativo, interactivo e innovador; no investigar más que lo necesario para el objetivo definido, no debe recolectar más datos de lo necesario, ni detallar innecesariamente el análisis; dejar que los actores mismos realicen su investigación, análisis e identificación y presentación de soluciones; identificar diferencias y excepciones, en vez de promediar, ya que la información diversa es muy rica para identificar innovaciones y para mayor comprensión; ser relajado en vez de ser apurado; escuchar en vez de exponer; comprobar en vez de dar respuestas rápidas y superficiales; ser modesto en vez de dominante e involucrar los retirados y callados en vez de trabajar con los que hablen más.

Existe diversidad en cuanto a los métodos empleados para desarrollar el proceso de estimación, luego de censar toda la información recopilada se arribó a que: un 60 % de los especialistas entrevistados vinculados a proyectos, utilizan el método de estimación por Puntos de Casos de Uso, citando que este 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 y que este método permite obtener resultados en un rango aceptable; un 30 %

realiza la estimación por Puntos de Función Desajustados (COCOMO II), siendo este un elemento tomado en consideración en la integración propuesta entre las teorías de Boehm y Humphrey en el trabajo y el 10 % restante utiliza una macro estimación, haciendo estimaciones a grandes rasgos empleando aplicaciones Excel provistas por los clientes con los datos requeridos para hacerlas. (Consúltese **Figura 9**).

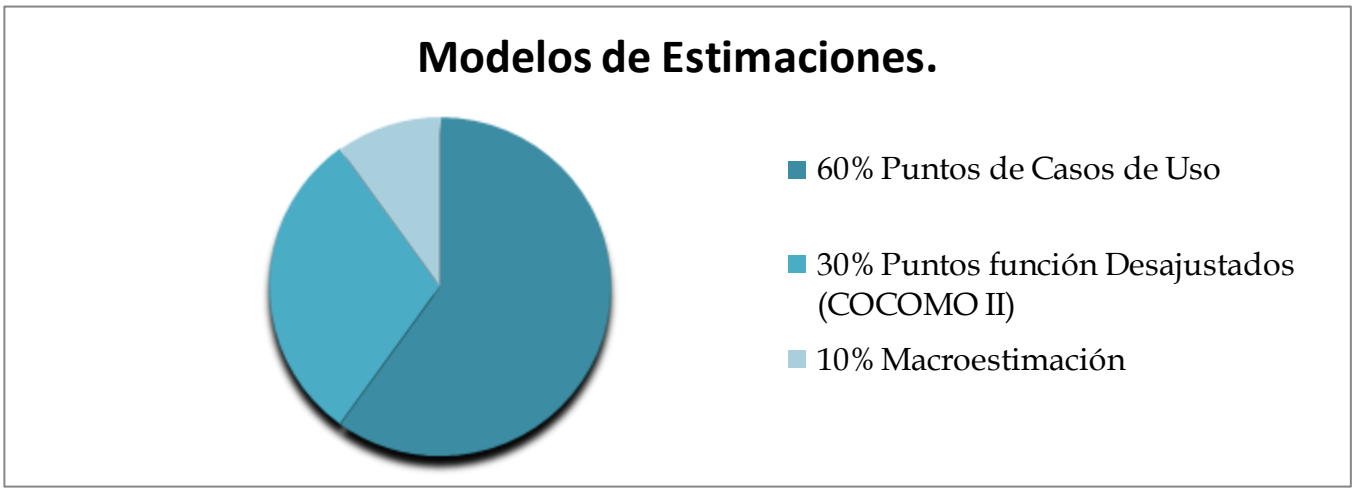


Figura 9: Porcentajes de uso de modelos de estimación en la UCI.

3.3.2 Impactos de PSP (experiencia).

Al preguntar acerca de la experiencia adquirida en aplicaciones de Gestión, si se creían expertos y cuál sería la valoración que le merece la experiencia adquirida, todos coincidieron que no se consideraban expertos en el tema, debido a que muchos opinan que una persona pudiera convertirse en experto en algún rol en específico, pero no experto en aplicaciones de Gestión de forma general, también creen que la experiencia no se adquiere solamente del tiempo que esa persona esté vinculada a un proyecto, sino a la cantidad de proyectos en los cuales logre aplicar estos conocimientos y la facilidad que tenga de adquirir los mismos. Los criterios acerca de la valoración que merece la experiencia adquirida fueron similares según los especialistas entrevistados, todos coinciden en que la experiencia es algo muy útil a la hora de realizar un proyecto de este tipo, añaden que es algo muy importante en la formación profesional del ingeniero y aún más en la aplicaciones de Gestión ya que estas son la base de las demás aplicaciones, o sea, muchas usan este tipo de aplicaciones para gestionar la información necesaria, es un elemento que pudiera reducir ampliamente el margen de errores

y el tiempo de desarrollo del proyecto, además es muy necesaria porque permite llevar la teoría a la práctica.

Con respecto al conocimiento y aplicación del Proceso Personal de Software (PSP) el 100 % considera que este modelo puede ser utilizado con el objetivo de producir software de calidad. Fue diseñado para ayudar a profesionales del software para que utilizaran constantemente prácticas sanas de ingeniería de software, es una herramienta que ayuda a organizar el trabajo, haciendo un uso óptimo del tiempo y de la planificación de las actividades, así como asignar estas actividades a los ingenieros según la capacidad de los mismos. Alegan también que PSP demuestra a los ingenieros la forma de manejar la calidad desde el comienzo y durante todo el ciclo de desarrollo del software. Pese a tener conocimiento de todas esas ventajas que brinda PSP, el 50% de los entrevistados afirman no haber utilizado PSP en sus proyectos y la otra mitad plantea que se han utilizado, casi de forma empírica, algunos elementos básicos de este modelo. Por ejemplo como cada ingeniero es esencialmente distinto a otro, se ha tenido en cuenta a la hora de asignar las tareas y planificar el tiempo de entregas de las mismas; por otra parte para comprometer a los ingenieros o futuros ingenieros con la calidad de lo que desarrollan, se les han asignado módulos en sus respectivos proyectos, responsabilizándolos de esta manera con el desarrollo y calidad de los mismos.

3.3.3 Factores de Escala.

Al tratar el tema de la influencia de los factores de escala y si la integración que se plantea en este sentido la creían factible, el 100% de los entrevistados coincidieron que la integración planteada podía acarrear buenos resultados, opinando que con la aplicación de esta propuesta se pudiera lograr que exista mayor precisión y seguridad en los valores ofrecidos por los especialistas encargados de dar los porcentajes de conformidad para cada una de las KPA’s, lo que representaría un estado de madurez real. Logrando de esta forma que los procesos de software no sean improvisados durante el curso de un proyecto por quienes los ejecutan y que sus gestores no se centren en la resolución inmediata de los problemas, consiguiendo también evitar que se sobrepasen los plazos y los presupuestos de los proyectos basados en estimaciones poco realistas, impidiendo fechas topes agresivas en la

planificación de los proyectos, que pudieran comprometer la funcionalidad y calidad del producto para poder cumplir con el plan de fechas.

Otra de las ventajas que puede traer esta integración es la de aumentar los porcentajes de cumplimiento de cada una de las KPA's, pues en esto se centra PSP, en orientar a los ingenieros a que exista un mejoramiento notable en el proceso de desarrollo, lo que pudiera significar una disminución del esfuerzo necesario para el desarrollo de proyectos y por consiguiente en el tiempo de duración de los mismos. Logrando con ello una base objetiva para poder enjuiciar la calidad del producto o para resolver problemas del producto o el proceso, además de adquirir un mayor entendimiento de cómo los pasos del proceso software afectan a la calidad.

En la Universidad de las Ciencias Informáticas (UCI) no se presta atención en la mayoría de los casos, al cálculo de la madurez del proceso a la hora de construir software de aplicaciones de Gestión. De allí que no se tenga en cuenta los resultados que se puedan obtener y cómo pudieran influir en la calidad del software. En sentido general los menores porcentajes de cumplimiento de las KPA's en aplicaciones de software de Gestión se encontraron en las siguientes áreas de procesos claves:

- ✓ Seguimiento y control de proyectos.
- ✓ Planeación de los proyectos.
- ✓ Manejo integrado del software.
- ✓ Control de calidad.
- ✓ Prevención de defectos.

De esta forma se reafirma la integración planteada ya que estas KPA's antes mencionadas, entran dentro de las 12 áreas de procesos claves que tiene en cuenta PSP.

3.3.4 Líneas de Código y Reutilización de Código.

Al hacer referencia a las líneas de código, al igual que sucede a nivel internacional, no existe una definición estándar de línea de código. Existen dos formas de definir las líneas de código: la primera es a nivel de programas (Véase **Figura 10**), donde:

- ✓ Un 40 % de los entrevistados las define como: líneas de código determinadas por delimitadores lógicos.
- ✓ Un 30 % las toma como: solo las líneas de código ejecutable.
- ✓ Un 20 % las cuenta como: líneas de código, definiciones de datos y comentarios.
- ✓ El 10 % restante las define como: líneas de código y líneas físicas visualizadas en una pantalla.

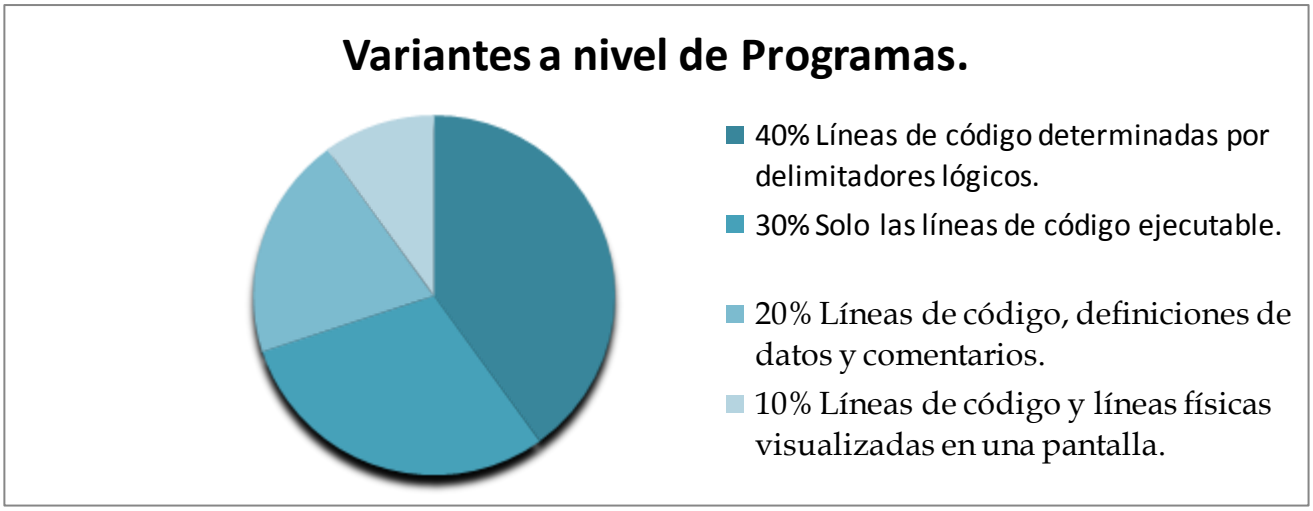


Figura 10: Porcentajes de definiciones de línea de código a nivel de programa en la UCI.

La segunda variante es a nivel de proyecto (Véase **Figura 11**), donde los criterios se dividen en:

- ✓ Un 70 % las define como: líneas nuevas y líneas modificadas.
- ✓ El 30 % restante las toma como: solo las líneas nuevas.

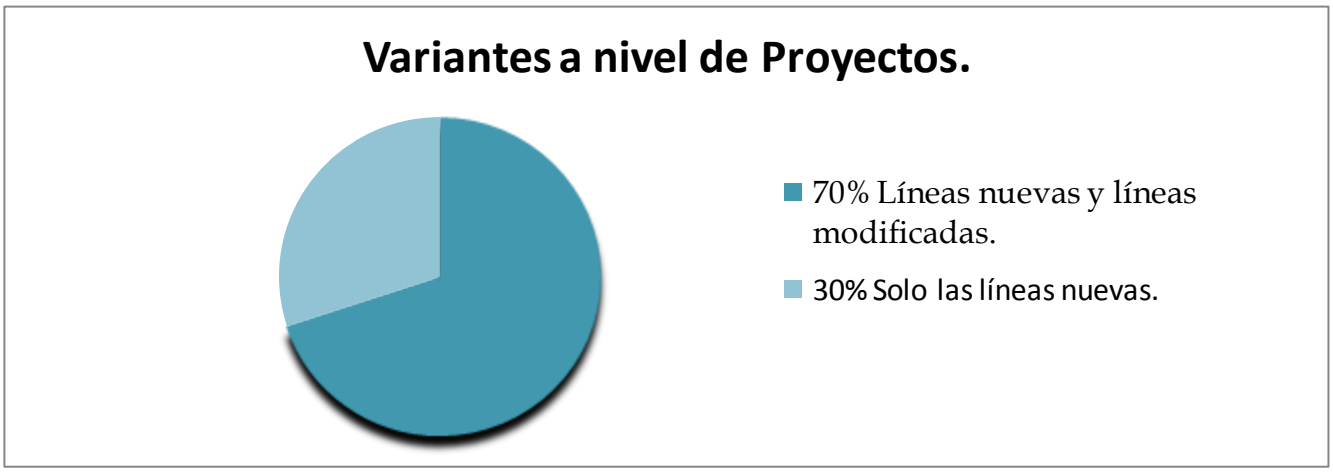


Figura 11: Porcentajes de definiciones de línea de código a nivel de proyectos en la UCI.

Una de las interrogantes abordadas en la entrevista cuestionaba a los especialistas acerca de qué no dejaría de tener en cuenta a la hora de construir un concepto de líneas de código en la UCI, para dar respuesta a esta pregunta se realizó una compilación de todos los criterios emitidos y en resumen salió a relucir lo siguiente: cuantificar, a nivel de proyecto, solamente las nuevas líneas de código generadas y a nivel de programa, contar el código ejecutable, las líneas determinadas por delimitadores lógicos, sin excluir pero dando menos importancia al resto de los elementos, por ejemplo las definiciones de datos, comentarios etc.

Precisamente este es uno de los puntos de contacto entre las teorías de Boehm y Humphrey tomados en cuenta para plantear la posible integración entre ambas. Durante la investigación y como parte de los resultados de la misma se demostró que es posible hacer estimaciones de tiempo calculando el tamaño total estimado del programa (T) expresado en líneas de código mediante el método PROBE, luego con el valor de tamaño calcular el esfuerzo estimado y con este el tiempo total estimado de duración del proyecto empleando para ello COCOMO II. Ante esta vía de integración propuesta los especialistas se mostraron atraídos, aunque un 30 % se negó a expresar su opinión, el 70% restante la consideró aceptada y propuso aplicarla.

La reutilización de código fue otro de los puntos de contacto entre las teorías de Boehm y Humphrey, así como otro de los tópicos atendidos en las entrevistas, con respecto a esto se

pudo llegar a que el 100 % de los especialistas abordados plantean que en sus respectivos proyectos se aplica la reutilización de código, variando solamente el porcentaje de reutilización en dependencia de la experiencia y la forma de llevar a cabo la reutilización, por ejemplo en algunos proyectos cuando terminan un módulo guardan las líneas de código construyendo una especie de biblioteca para cuando algún otro módulo necesite ese código, el mismo esté listo para ser reutilizado, otros usan los frameworks, los cuales traen ya implementados un conjunto de funcionalidades que son reutilizadas con mucha frecuencia, la genialidad de los mismos consiste en que simplifican y aceleran considerablemente el proceso de desarrollo de una aplicación; ya que automatiza algunos de los patrones utilizados para resolver las tareas más comunes, mediante el encapsulamiento de operaciones complejas en instrucciones sencillas. Existe la posibilidad de reutilizar código también gracias a la existencia de librerías y del desarrollo de arquitecturas base, lo que resulta de mucha ayuda para disminuir el esfuerzo y el tiempo de duración del proyecto.

Por otra parte al introducir la propuesta que tenia en cuenta la líneas de código reutilizadas con el objetivo de lograr un uso óptimo de las mismas para lograr estimaciones de tiempo más exactas, se probó que se puede calcular el tamaño total estimado del programa expresado en líneas de código (KESLOC) mediante el modelo COCOMO II y que ya tiene en cuenta las líneas de código reutilizadas, para luego con ese valor de tamaño calcular el tiempo total de duración mediante el método PROBE. En este sentido un 80% de los entrevistados coinciden en que la idea es interesante y proponen la aplicación de la misma con el objetivo de obtener resultados más fiables.

Conclusiones:

Con el fin de lograr la integración propuesta se realizaron búsquedas bibliográficas y entrevistas a conocedores del tema en la UCI: vicedecanos de producción, líderes de proyectos, evaluadores de calidad y estudiantes vinculados a proyectos productivos aplicando el Método DELPHI Cara-Cara para evaluar la integración planteada. En el desarrollo de este capítulo se han mostrado los argumentos dados por los especialistas acerca de cómo medir el conocimiento y cómo aplicar el mismo para realizar estimaciones en los proyectos de software, dar los detalles a la hora de hacer uso de la experiencia adquirida, evaluar la influencia de los factores de escala en las teorías de Boehm y Humphrey, así como abordar

todo lo concerniente a las líneas de código y la reutilización de las mismas en los proyectos de aplicaciones de Gestión en la Universidad.

CONCLUSIONES

Para dar solución al problema planteado se hizo la introducción al tema en cuestión, se abordaron temas como: proyectos de software, de los cuales además se enunciaron algunos antecedentes haciendo un poco de historia del surgimiento y desarrollo de los mismos; proceso de gestión de software; planificación y estimación, las cuales serían el centro del trabajo, en este último tema se profundizó en cuanto a la Estimación en la Planificación, a los detalles para dar comienzo a un Proceso de Estimación de Proyectos de Software, a la Estimación de la Duración, a la Estimación de Tamaño, a las Técnicas comúnmente utilizadas para la estimación, a los Métodos de Estimación donde se evidenciaron los siguientes: Puntos de Función, Método Wideband-Delphi, Método Difuso de Putnam, Método de Componentes Estándar y el Método PROBE; se hizo alusión a los Modelos de Estimación entre los cuales se trataron los analógicos, los teóricos, los empíricos y algunas herramientas automáticas de estimación. Se resumieron las teorías de Boehm y Humphrey, así como las obras cumbres de estos grandes científicos, luego se aborda todo lo concerniente a las Métricas del Software: definiciones, ejemplos, etc. También como parte fundamental del trabajo se hace referencia a las Aplicaciones de Gestión y a los nuevos retos que enfrentan las mismas, debido a que en ellas se debía probar la validez de la integración propuesta.

Se atendieron los temas: el Proceso Personal del Software (PSP), profundizando en el mismo, haciendo énfasis en su definición, sus niveles, su surgimiento, sus principios, las características del marco que proporciona PSP para que todos los componentes individuales se desarrollen con la más alta calidad y el flujo de trabajo de este modelo; el método PROBE fue también objeto de la investigación, el mismo se centra en la estimación basada en Proxy por lo que se expone el concepto de Proxy, así como los criterios que se toman en consideración para declarar a un Proxy como un buen indicador y se explica el objetivo fundamental de este proceso, se enuncian los pasos que componen el método PROBE, explicando el funcionamiento de cada uno de estos pasos individualmente. Se hace referencia también a los 4 procedimientos con que cuenta este método para estimar según la cantidad de datos históricos que usted tenga, detallando los datos necesarios para usar cada uno de ellos.

Los Puntos de Contacto entre las teorías de Boehm y Humphrey arrojados por al investigación que sirve de precedente, formaron parte de este trabajo también, estos son el punto de partida de esta investigación debido a que sugieren que es posible una integración real entre las teorías de estos dos científicos, los mismos son: líneas de código, factor de escala de Madurez del Proceso (PMAT), multiplicador de esfuerzos de Experiencia del Personal de Desarrollo (PREX) y Reutilización de código. Acerca de las ***líneas de código*** se trataron asuntos como algunas de sus definiciones, debido a que no existe un concepto en específico que se use para definirlas y se ilustraron los conceptos más tratados internacionalmente a nivel de programa y a nivel de proyectos. También se expone una vía para calcular el tiempo de duración aplicando la integración entre las teorías tratadas, substituyendo el valor de tamaño en líneas de código calculado mediante el método PROBE en una ecuación de COCOMO II para hallar el tiempo y así fomentar la posible integración planteada.

En cuanto al ***PMAT*** que es uno de los cinco factores de escala que influyen exponencialmente en la productividad y esfuerzo de un proyecto de software y que a su vez es uno de los puntos de contacto entre las ambas teorías, se propone utilizar las buenas prácticas que brinda PSP, con el objetivo de obtener valores de la variable ***PMAT*** más reales o aumentar los porcentajes de algunas de las Áreas de Procesos Claves (KPA's).

Otro punto de contacto es el multiplicador de esfuerzos PREX que se basa en la experiencia personal, es uno de los 7 multiplicadores que sugiere COCOMO II para estimar en el diseño temprano. El Proceso Personal del Software (PSP) de Humphrey también toma como premisa que la experiencia es uno de los factores fundamentales, junto con la habilidad y los hábitos de cada ingeniero, para el desarrollo con éxito del software, por ello se tomó este indicador (experiencia) para continuar argumentando la posible integración propuesta en esta investigación entre las teorías de Boehm y Humphrey. El último punto de contacto es la ***reutilización de código***, detallando en el mismo algunos conceptos y las diferentes formas que se emplean para reutilizar líneas de código. En este epígrafe se plantea utilizar el método PROBE para calcular las líneas de código reutilizadas y luego con este valor aplicar COCOMO II para calcular el tamaño y tiempo estimado.

Se confeccionó una entrevista donde se exponen los criterios de un grupo de especialistas acerca del tema tratado, los mismos argumentan de forma consistente la veracidad y ventajas de la integración propuesta. Dicha entrevista se centró en: medir los conocimientos y la aplicación de estimación de software, donde se exponen los distintos métodos empleados para realizar procesos de estimación en la universidad y la importancia que merecen los mismos a la hora de planificar los proyectos de software; los impactos que ha tenido el PSP, así como la importancia de usar este modelo y la importancia que tiene la experiencia en las aplicaciones de Gestión específicamente; los factores de escala, exponiendo las ventajas que tiene aumentar los porcentajes de las KPAs para lograr una madurez real en los proyectos, también se enunciaron las KPAs que presentan menor porcentaje de cumplimiento en aplicaciones de Gestión, etc.; las líneas de código y la reutilización del mismo en los proyectos de software de Gestión en la UCI, abordando los diferentes conceptos de líneas de código según las dos variantes existentes (a nivel de proyecto y a nivel de programa); también se explica, haciendo uso de las opiniones provistas por los especialistas, cómo se vinculan estos dos puntos de contacto profundizados en capítulos anteriores a un grupo proyectos de Gestión en la universidad, validando y evaluando la integración propuesta entre los elementos las teorías de Boehm y Humphrey para la estimación de duración de los proyectos de aplicaciones de Gestión.

Todo este contenido proporcionó respuestas a las interrogantes que proponía el trabajo, lográndose hacer un análisis de las métricas de software vinculadas con la medición y estimación de un proyecto, de las teorías de estimación propuestas por Boehm y Humphrey, así como realizar un estudio más profundo del método PROBE. También se estudiaron los elementos de integración entre las teorías de Boehm y Humphrey en la duración de proyectos y se aplicó la integración entre estos elementos comunes en los proyectos de aplicaciones de gestión. Además se logró evaluar la integración entre los elementos comunes anteriormente mencionados dando cumplimiento al objetivo principal de este trabajo investigativo.

RECOMENDACIONES

- De este trabajo se extrae la necesidad de guardar registros históricos de la aplicación de modelos de estimación, por tal motivo se propone, como primera recomendación, crear una base datos donde se almacenen estos datos.

- Que todo el personal encargado de llevar a cabo el proceso de planificación y estimación de los recursos tanto materiales como humanos y el tiempo, en los proyectos productivos de aplicaciones de gestión, reciba la capacitación necesaria para realizar dicha tarea.

- Continuar con la investigación para aplicar la integración planteada en este trabajo a los proyectos de aplicaciones de Gestión en la UCI con el objetivo de hacer estimaciones más exactas y así lograr administrar y planificar los recursos, el personal y el tiempo de duración en el desarrollo de proyectos productivos de este tipo.

REFERENCIAS BIBLIOGRÁFICAS

1. **Adriana Gómez, María del C. López, Silvina Migani, Alejandra Otazú. 1994.** www.alarcos.inf-cr.uclm.es. *www.alarcos.inf-cr.uclm.es.* [En línea] 1994. http://alarcos.inf-cr.uclm.es/doc/pgsi/doc/teo/8/cocomo2-apuntes.pdf.
2. **Ajenjo, Alberto Domingo. 2000.** *Dirección y Gestión de Proyectos.* 2000.
3. **Brand, Jaime Pereña. 1996.** *Dirección y Gestión de Proyectos.* 1996.
4. **Cao, Jose Antonio. 2006.** www.itba.edu.ar. *www.itba.edu.ar.* [En línea] 2006. http://www.itba.edu.ar/capis/epg-tesis-y-tf/cao-trabajofinaldeespecialidad.pdf.
5. **Capuchino, Ana Sánchez. 1996.** *Universidad Politécnica de Madrid. Departamento de Lenguajes y Sistemas Informáticos e Ingeniería del Software. Unidad Docente de Ingeniería del Software.* Madrid : s.n., 1996.
6. **Carla Basurto, Abraham Dávila, Karin Melendez. 2005.** www.gidis.inf.pucp.edu.pe. *www.gidis.inf.pucp.edu.pe.* [En línea] 2005. http://gidis.inf.pucp.edu.pe/recursos/cocomoii.pdf.
7. **Castillo, Manuel de Cos. 1999.** *Teoría General del Proyecto, Vol. I y II.* Madrid : Síntesis, 1999.
8. **Concepción, Pedro. 2005.** www.getec.etsit.upm.es. *www.getec.etsit.upm.es.* [En línea] 2005. http://www.getec.etsit.upm.es/articulos/gproyectos/art4.htm.
9. **Courter, Gini. 2000.** www.sorad.ual.es. *www.sorad.ual.es.* [En línea] 2000. http://sorad.ual.es/mitra/documentos/jueves_estrate/Gesti%C3%B3n%20de%20Proyectos.pdf.
10. **2003.** *Estimating of Software Size - Part II 2003.* 2003.
11. **Garcia, Maria N. Moreno. 2000.** www.unab.edu.co. *www.unab.edu.co.* [En línea] 2000. http://www.unab.edu.co/editorialunab/revistas/rcc/pdfs/r31_art3_c.pdf.
12. **Ivar JACOBSON, Grady BOOCH, James RUMBAUGH. 2000.** *El Proceso Unificado de Desarrollo de Software.* 2000.
13. **Mendoza, Luis Eduardo. 1997.** www.prof.usb.ve. *www.prof.usb.ve.* [En línea] 1997. http://prof.usb.ve/lmendoza/Documentos/PS-6117%20(Teor%EDa)/PS6117%20Calidad%20del%20Software.pdf.

14. **Pressman, Roger S. 1998.** www.itba.edu.ar. *www.itba.edu.ar.* [En línea] 1998. http://www.itba.edu.ar/capis/webcapis/proyectodetesisdemagister/Anteproyecto-Ovejero.pdf.
15. **—. 2002.** www.serdis.dis.ulpgc.es. *www.serdis.dis.ulpgc.es.* [En línea] 2002. http://serdis.dis.ulpgc.es/~a013775/asignaturas/ii-is2/Apuntes/UT05.%20Estimaci%C3%93n%20de%20proyectos.pdf.
16. **Heredia. 2003.** *Dirección integrada de proyectos (DIP) Project Management.* Madrid : E.T.S. de Ingenieros Industriales de la Universidad Politécnica de Madrid, 2003.
17. **Selin. 1994.** *Selin & Selin.* Estados Unidos : s.n., 1994.
18. **Sommerville, Ian. 2002.** www.dcc.espe.edu.ec. *www.dcc.espe.edu.ec.* [En línea] 2002. http://www.dcc.espe.edu.ec/archivos/File/Area%20de%20Software/Ingenieria%20de%20Software%20I.pdf.

BIBLIOGRAFÍA

1. **Adriana Gómez, María del C. López, Silvina Migani, Alejandra Otazú. 1994.** www.alarcos.inf-cr.uclm.es. *www.alarcos.inf-cr.uclm.es.* [En línea] 1994. http://alarcos.inf-cr.uclm.es/doc/pgsi/doc/teo/8/cocomo2-apuntes.pdf.
2. **Ajenjo, Alberto Domingo. 2000.** *Dirección y Gestión de Proyectos.* 2000.
3. **Brand, Jaime Pereña. 1996.** *Dirección y Gestión de Proyectos.* 1996.
4. **Cao, Jose Antonio. 2006.** www.itba.edu.ar. *www.itba.edu.ar.* [En línea] 2006. http://www.itba.edu.ar/capis/epg-tesis-y-tf/cao-trabajofinaldeespecialidad.pdf.
5. **Capuchino, Ana Sánchez. 1996.** *Universidad Politécnica de Madrid. Departamento de Lenguajes y Sistemas Informáticos e Ingeniería del Software. Unidad Docente de Ingeniería del Software.* Madrid : s.n., 1996.
6. **Carla Basurto, Abraham Dávila, Karin Melendez. 2005.** www.gidis.inf.pucp.edu.pe. *www.gidis.inf.pucp.edu.pe.* [En línea] 2005. http://gidis.inf.pucp.edu.pe/recursos/cocomoii.pdf.
7. **Castillo, Manuel de Cos. 1999.** *Teoría General del Proyecto, Vol. I y II.* Madrid : Síntesis, 1999.
8. **Concepción, Pedro. 2005.** www.getec.etsit.upm.es. *www.getec.etsit.upm.es.* [En línea] 2005. http://www.getec.etsit.upm.es/articulos/gproyectos/art4.htm.
9. **Courter, Gini. 2000.** www.sorad.ual.es. *www.sorad.ual.es.* [En línea] 2000. http://sorad.ual.es/mitra/documentos/jueves_estrate/Gesti%C3%B3n%20de%20Proyectos.pdf.
10. **2003.** *Estimating of Software Size - Part II 2003.* 2003.
11. **Garcia, Maria N. Moreno. 2000.** www.unab.edu.co. *www.unab.edu.co.* [En línea] 2000. http://www.unab.edu.co/editorialunab/revistas/rcc/pdfs/r31_art3_c.pdf.
12. **Ivar JACOBSON, Grady BOOCH, James RUMBAUGH. 2000.** *El Proceso Unificado de Desarrollo de Software.* 2000.
13. **Mendoza, Luis Eduardo. 1997.** www.prof.usb.ve. *www.prof.usb.ve.* [En línea] 1997. http://prof.usb.ve/lmendoza/Documentos/PS-6117%20(Teor%EDa)/PS6117%20Calidad%20del%20Software.pdf.

14. **Pressman, Roger S. 1998.** www.itba.edu.ar. *www.itba.edu.ar.* [En línea] 1998. http://www.itba.edu.ar/capis/webcapis/proyectodetesisdemagister/Anteproyecto-Ovejero.pdf.
15. **—. 2002.** www.serdis.dis.ulpgc.es. *www.serdis.dis.ulpgc.es.* [En línea] 2002. http://serdis.dis.ulpgc.es/~a013775/asignaturas/ii-is2/Apuntes/UT05.%20Estimaci%C3%93n%20de%20proyectos.pdf.
16. **R., Heredia. 2003.** *Dirección integrada de proyectos (DIP) Project Management.* Madrid : E.T.S. de Ingenieros Industriales de la Universidad Politécnica de Madrid, 2003.
17. **Selin. 1994.** *Selin & Selin.* Estados Unidos : s.n., 1994.
18. **Sommerville, Ian. 2002.** www.dcc.espe.edu.ec. *www.dcc.espe.edu.ec.* [En línea] 2002. http://www.dcc.espe.edu.ec/archivos/File/Area%20de%20Software/Ingenieria%20de%20Software%20I.pdf.
19. *Capítulo 3. PSP 0 y PSP 0.1* Disponible en:http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/pelaez_r_jj/capitulo3.pdf
20. *Estimating of Software Size - Part I* 2003. Disponible en: http://www.engr.sjsu.edu/ahambaba/course/Estimating%20of%20Software%20Size%20-%20Part%20I%20-%20Fall%202003.ppt
21. *Estimating of Software Size - Part II* 2003. Disponible en:http://www.engr.sjsu.edu/ahambaba/course/Estimating%20of%20Software%20Size%20-%20Part%20II%20-%20Fall%202003.ppt
22. *Estimating of Software Size - Part III* 2003. Disponible en: http://www.engr.sjsu.edu/ahambaba/course/Estimating%20of%20Software%20Size%20-%20Part%20III%20-%20Fall%202003.ppt
23. Documento de la Universidad de Guadalajara Disponible en: http://www.willydev.net/Descargas/WillyDEV_PlaneaSoftware.Pdf
24. GROMPONE, J. *Gestión de Proyectos de Software* 1996. Disponible en: http://www.itapebi.com.uy/pdfs/GPS.pdf
25. *Guía de Aprendizaje - Tema 8.Gestión de Costes en Proyectos Software*. Disponible en: http://alarcos.inf-cr.uclm.es/doc/pgsi/doc/teo/8/pgsi-t8.pdf
26. *Métricas en el desarrollo del Software*. Disponible en:

http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/gonzalez_d_h/capitulo4.pdf

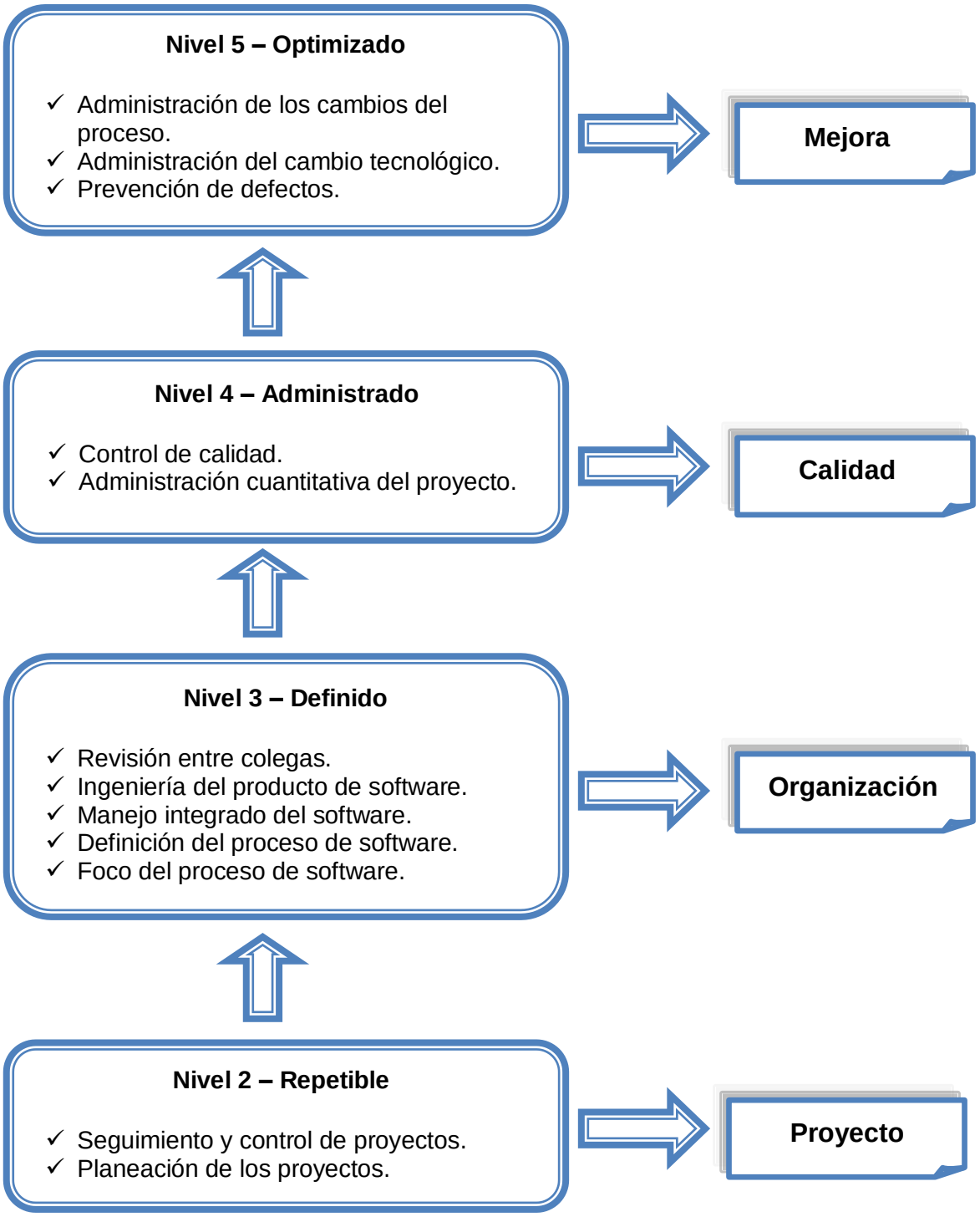
27. El modelo CMM (Modelo de Capacidad y Madurez) Disponible en: http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/gonzalez_d_h/capitulo5.pdf
28. SANDRA PATRICIA FORIGUA PULIDO, O. A. B. G. *Propuesta de un modelo de análisis para estimación del tamaño del software y gestión de costos y riesgos a partir de requerimientos funcionales*, Pontificia Universidad Javeriana. Facultad de Ingeniería. Carrera de Ingeniería de Sistemas. , 2006. Disponible en: http://www.pegasus.javeriana.edu.co/~riesgors/propuesta.pdf
29. WIKIPEDIA Dato, 2007a.
30. *Ingenieria de software* 2007b. Disponible en: http://es.wikipedia.org/wiki/Ingenier%C3%ADa_del_software
Universidad Simón Bolívar, Ingeniería de Software 3, Enero-Marzo 2001
31. *COCOMO II: Una Familia de Modelos de Estimación.* Disponible en: *http://www.ldc.usb.ve/~teruel/ci4713/clases2001/cocomo2.html*
32. *Joaquín Gracia 14 de Agosto de 2005,* CMM – CMMI. http://www.ingenierosoftware.com/calidad/cmm-cmmi.php
33. Richard Basque. El modelo de madurez de capacidades del software del SEI (CMM) Disponible en: http://www.procesix.com/v1/cmm_rb_1.pdf
34. Ing. Yeleny Zulueta Véliz. Introducción de técnicas del Personal Software Process desde los primeros años en la formación del ingeniero informático. Disponible en: http://www.inf.udec.cl/revista/ediciones/edicion14/zulueta.pdf
35. Francisco Sanchis. Dpto. OEI /UPM. LA MEDICIÓN DEL SOFTWARE. Disponible en: http://www.issi.uned.es/CalidadSoftware/material/LA-MEDICION-DEL-SOFTWARE.doc
36. Juan Palacio Bañeres. Junio 2006 Compendio de Ingeniería del Software II. Disponible en: http://www.navegapolis.net/files/cis/CIS_2_04.pdf
37. Guía de los Fundamentos de la Dirección de Proyectos 2004 Project Management Institute, Tercera Edición. Disponible en: http://www.depto103.cl/taller1/documentos/PMBOK-2004.pdf
38. Eduardo Diez .Buenos Aires - 2003. ASEGURAMIENTO DE LA CALIDAD EN LA CONSTRUCCIÓN DE SISTEMAS BASADOS EN EL CONOCIMIENTO UN ENFOQUE

PRÁCTICO. Disponible en: http://www.itba.edu.ar/capis/epg-tesis-y-tf/diez-trabajofinaldeespecialidad.pdf

39. WATTS S. HUMPHREY 2000. Making the Tactical Case for Process Improvement. Disponible en: http://www.sei.cmu.edu/news-at-sei/columns/watts_new/2000/spring/watts-spring00.htm
40. IX Workshop de Investigadores en Ciencias de la Computación (2007). METODOS DE LA INGENIERIA INFORMATICA AVANZADA Disponible en: http://www.itba.edu.ar/capis/webcapis/RGMITBA/comunicacionesrgm/WICC-07-435-437.pdf
41. OLSINA, D. L. Curso: INGENIERÍA DE SOFTWARE, 2005. Disponible en: http://www.frsf.utn.edu.ar
42. FÉLIX GARCÍA, F. R. Gestión Integrada del Modelado y de la Medición del Proceso Software. Disponible en: http://alarcos.inf-cr.uclm.es
43. JAVIER GARCÍA, A. D. A., MANUEL VELASCO. TOP 10 de factores que obstaculizan la mejora de los procesos de verificación y validación en organizaciones intensivas en software, 2006. Disponible en: http://www.ati.es/IMG/pdf/GarciaGuzmanNum2Vol2.pdf
44. CIENTEC. CMMI: MEJORANDO PROCESOS EN FORMA INTEGRADA, 2006. [Disponible en: http://www.cientec.com/analisis/cmmi.asp

ANEXOS

Anexo_1: Elementos de PSP en CMM.



Anexo_2: Entrevista.

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

a. ¿Han realizado estimaciones en el proyecto? ¿Las han tenido en cuenta para hacer la planificación?

b. ¿Que método(s) utilizan para estimar?

c. ¿En que etapas de desarrollo del software realizan las estimaciones?

d. ¿Quienes o quien las realizan?

e. ¿Considera que puede valorarse la estimación como un elemento importante antes de hacer la planificación? Porqué?

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

a. ¿Que tiempo lleva desarrollando este tipo de aplicaciones? ¿Se considera un experto?

b. ¿Cuando considera que una persona es experta en aplicaciones de gestión? ¿Que tiempo puede demorar alcanzar esa categoría?

c. ¿Que valoración merece la experiencia adquirida en el desarrollo de una aplicación de Gestión?

d. ¿Conoce las características del PSP? ¿Cree usted que es importante para obtener software de calidad?

e. ¿Se ha tenido en cuenta las prácticas de PSP en el equipo de desarrollo? En caso afirmativo de detalles de su aplicación.

3- Detallar Teoría de Boehm.

a. Evaluar influencia de los factores de escala. ¿Agregaría otros?

b. ¿Cree que la integración entre las teorías de Boehm y Humphrey, para estimaciones de tiempo en Aplicaciones de Gestión que propone la aplicación de PSP para mejorar los valores de PMAT sea factible? ¿Por qué?

c. ¿En que etapa de desarrollo considera que se requiere de un mayor esfuerzo?

4- Líneas de Código y Reutilización de Código.

a. ¿Que es una línea de línea de código (LDC)?

Las variantes a nivel de programas son:

___ Contar solo las líneas de código ejecutable.
___ Líneas de código más definiciones de datos.
___ Líneas de código, definiciones de datos y comentarios.
___ Líneas de código, definiciones de datos, comentarios y lenguaje de control (Job Control Lenguaje, JCL).
___ Líneas de código y líneas físicas visualizadas en una pantalla.
___ Líneas de código determinadas por delimitadores lógicos.

Las variantes a nivel de proyecto son:

___ Contar solo las líneas nuevas.
___ Contar líneas nuevas y líneas modificadas.
___ Contar líneas nuevas, líneas modificadas y líneas reutilizadas.

b. ¿Si fuera a construir un concepto de líneas de código en la UCI que no dejaría de tener en cuenta?

c. ¿Cree que la integración que se propone entre las teorías de Boehm y Humphrey, para estimaciones de tiempo en Aplicaciones de Gestión teniendo en cuenta las líneas de código sea factible? ¿Por qué?

d. ¿Reutiliza líneas de código? ¿En que porciento reutiliza o ha reutilizado líneas de código en esta u otras aplicaciones de gestión anteriores?

e. ¿Cree que la integración que se propone entre las teorías de Boehm y Humphrey, para estimaciones de tiempo en Aplicaciones de Gestión teniendo en cuenta las líneas de código reutizadas sea factible? ¿Por qué?

GLOSARIO

Aplicación (informática): programa informático diseñado para facilitar al usuario la realización de un determinado tipo de trabajo… Suele resultar una solución informática para la automatización de ciertas tareas complicadas como puede ser la contabilidad o la gestión de un almacén (ENCARTA 2007).

Calidad del software: concordancia con los requisitos funcionales y de rendimiento explícitamente establecidos con los estándares de desarrollo explícitamente documentados y con las características implícitas que se espera de todo software desarrollado profesionalmente" R. S. Pressman (1992).

Calidad: es el conjunto de características de un producto que satisfacen las necesidades de los clientes y en consecuencia hacen satisfactorio el producto.

Ciclo de vida (del proyecto): Un conjunto de fases del proyecto que, generalmente son secuenciales, cuyos nombres y números son determinados por las necesidades de control de la organización u organizaciones involucradas en el proyecto. Un ciclo de vida puede ser documentado con una metodología.

CMM: Capability Maturity Model: Modelo para la mejora o evaluación de los procesos de desarrollo y mantenimiento de sistemas y productos de software. Fue desarrollado por el Instituto de Ingeniería del Software de la Universidad Carnegie Mellon (SEI).

Estándar: en informática, conjunto de especificaciones técnicas utilizadas para unificar el desarrollo de *hardware* o de software (ENCARTA 2007).

Estimación: Aprecio y valor que se da y en que se tasa y considera algo (ENCARTA 2007).

Herramientas: Algo tangible, como una plantilla o un programa de software, utilizado al realizar una actividad para producir un producto o resultado.

Ingeniería de Software: es la rama de la ingeniería que crea y mantiene las aplicaciones de software aplicando tecnologías y prácticas de las ciencias computacionales, incluye el manejo de proyectos y otros campos (WIKIPEDIA 2007c).

Interfaz: punto en el que se establece una conexión entre dos elementos, que les permite trabajar juntos. La interfaz es el medio que permite la interacción entre esos elementos (ENCARTA 2007).

Internet: es un método de interconexión de redes de computadoras implementado en un conjunto de protocolos denominado TCP/IP y garantiza que redes físicas heterogéneas funcionen como una red (lógica) única. De ahí que Internet se conozca comúnmente con el nombre de "red de redes".

KPA's (Key Process Areas): áreas de procesos clave.

LOC (LDC): Líneas de código.

Medición: es el acto de determinar una medida (E-CLASES).

Medida: proporciona una indicación cuantitativa de extensión, cantidad, dimensiones, capacidad y tamaño de algunos atributos de un proceso o producto. Pueden ser directas, por ejemplo, número de líneas de código, número de errores encontrados, etc., o pueden ser indirectas, por ejemplo, funcionalidad, calidad, complejidad, etc. (E-CLASES).

Mejora de proceso (de software): Intenta cambiar la forma en que la gente ejecuta las actividades para satisfacer mejor los objetivos del negocio: Mejorar implica siempre cambiar y se debe definir en términos de objetivos de negocio y se debe manejar del mismo modo.

Método: procedimiento que se aplica para llegar a un fin determinado (SANDRA PATRICIA FORIGUA PULIDO 2006).

Métrica: Una forma de medir y una escala, definidas para realizar mediciones de uno o varios atributos**.** Son un medio para entender, monitorizar, controlar, predecir y probar el desarrollo de software y los proyectos de mantenimiento.

PMAT: Es el factor mediante el cual se calcula la Madurez del Proceso (COCOMO II).

Proceso (de desarrollo) de software: es el conjunto de técnicas y procedimientos que permiten conocer los elementos necesarios para definir un proyecto de software (Proceso de Desarrollo de Software 2007).

Producto: cualquier software que será construido a petición de otros (PRESSMAN 2002).

Programa (informático): conjunto de instrucciones escritas en un lenguaje de programación para su ejecución en un ordenador o computadora. Por lo general, el término implica una entidad auto-contenida, a diferencia de una rutina o una biblioteca (ENCARTA 2007).

PSP (Personal Software Process): Proceso Personal de Software.

Software: todos los componentes intangibles de un ordenador o computadora, es decir, al conjunto de programas y procedimientos necesarios para hacer posible la realización de una

tarea específica, en contraposición a los componentes físicos del sistema (hardware) (WIKIPEDIA 2007g).

Tamaño del software: el tamaño de un producto se software es un indicador de la amplitud y profundidad del conjunto de prestaciones que incorpora, así como de la dificultad y profundidad del programa (Guía de Aprendizaje – Tema 8.Gestión de Costes en Proyectos Software).