Universidad de las Ciencias Informáticas

Facultad 2

Título

Evaluación de posibles elementos a considerar en la integración de las teorías de Boehm y Humphrey para la estimación de la duración de un proyecto de software para aplicaciones de realidad virtual.

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autores: Yunior Álvarez Caballero

Yorkeidis Cuenca Vilche

Tutor: Mcs. Yadira Ruiz Constantens

Junio de 2008

DECLARACIÓN DE AUTORÍA

Declaro que soy el único autor de este trabajo y autorizo a la Facultad 2 de la Universidad de las Ciencias Informáticas (UCI) a hacer uso del mismo en su beneficio.
Para que así conste firmo la presente a los __ días del mes de ____del año ____.

_____ _____

Firma del Autor **Firma del Autor**

Yunior Alvarez Caballero Yorkeidis Cuenca Vilche

_____

Firma del Tutor

Yadira Ruiz Constanten

"No hay más que asomarse a las puertas de las tecnologías y a la ciencia contemporánea para preguntarnos si es posible vivir y conocer ese mundo de futuro sin un enorme caudal de proposición y conocimientos. "

Fidel Castro

Dedicatoria

A mi madre, a mi tía Dalia, a mi primo Erisbel, a mi prima Daymis, a mi tío Jose Manuel y a Rolando por tener tanta confianza en mi y apoyarme siempre.

Y muy en especial a mi abuela Ada.

Yunior

A mis padres Maricela y Raidel, a mis hermanos Diannelis y Yuder, por ser las personas más importantes de mi vida.

A mi novia por estar siempre a mi lado y darme su apoyo en los momentos difíciles, por ser la mejor novia del mundo y por todo el cariño que me ha dado.

A todos mis amigos por estar siempre en contacto conmigo y apoyarme en los momentos difíciles.

En especial a mi compañero de tesis Yunior por caminar junto a mí en este largo y difícil camino.

A toda mi familia.

Yorkeidis

Agradecimientos

A nuestra tutora Yadira por la ayuda prestada, por el tiempo dedicado, por confiar en nosotros desde el principio y por todos los conocimientos que nos ha transmitido.

A Pedro Luis Basulto por estar siempre dispuesto a ayudarnos y brindarnos su amistad incondicional.

A nuestros compañeros de Aula.

A nuestro líder de todos los tiempos Fidel Castro y a la revolución Cubana por formarnos y darnos la oportunidad de estar en esta Universidad de excelencia.

A todos muchas gracias…

Resumen

Este trabajo aparece como respuesta a ciertos problemas que se están dando en la estimación de los proyectos en la Universidad de las Ciencias Informáticas, donde generalmente se estima la duración de un proyecto solo una vez durante su proceso de desarrollo, y además empleando un solo método, lo que impide establecer comparaciones que puedan asegurar la confiabilidad de las estimaciones realizadas, de ahí que se presenta como problema: Cuáles son los resultados que ofrece evaluar los elementos de la integración de las teorías de Boehm y Humphrey para la estimación de la duración de proyectos de software para las aplicaciones de realidad virtual en la Universidad de las Ciencias Informática?, para lo cual se propuso evaluar los elementos de integración de las teorías de Boehm y Humphrey para una mejor estimación de la duración de un proyecto de software para aplicaciones de Realidad Virtual. Se espera que al aplicar la propuesta que se presenta se obtengan mejores resultados en la estimación de la duración de los proyectos de Realidad Virtual de la Universidad, que conllevaría a lograr productos de mejor calidad y aumentar la productividad en la Universidad.

Palabras claves

- Estimación
- Esfuerzo
- Tiempo
- Integración
- Realidad virtual

Índice General

Índice de figuras

Índice de tablas

Introducción

La planificación en general en todos los campos que se aplica reduce la incertidumbre porque ayuda a prevenir los cambios, considerar los impactos de estos y desarrollar las respuestas apropiadas para cada caso. También se puede decir que la planificación es un proceso de preparación de decisiones referentes al futuro con lo que se condicionan y posibilitan futuras decisiones. Este proceso se basa en un pronóstico, previamente elaborado según métodos específicos.

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 (anónimo, 2003).

Se encuentran 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.

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 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 (Concepción, 2005).

El objetivo de la planificación se logra mediante un proceso de descubrimiento de la información que lleve a estimaciones razonables.

El Autor Bennatan sugiere cuatro categorías de recursos de software que se deberían tener en cuenta a medida que se avanza con la planificación:

- Componentes ya desarrollados.
- Componentes ya experimentados.
- Componentes con experiencia Parcial.
- Componentes nuevos.

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.

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

- Deje la estimación para más adelante (obviamente se puede realizar una estimación al cien por cien fiable después de haber terminado el proyecto.
- Base las estimaciones en proyectos similares ya terminados.
- Utilice técnicas de descomposición relativamente sencillas para generar las estimaciones de costos y esfuerzo del proyecto.
- Desarrolle un modelo empírico para él cálculo de costos y esfuerzos del Software.

Desdichadamente la primera opción, aunque atractiva no es práctica.

La Segunda opción puede funcionar razonablemente bien si el proyecto actual es bastante similar a los esfuerzos pasados y si otras influencias del proyecto son similares. Las opciones restantes son métodos viables para la estimación del proyecto de software. Desde el punto de vista ideal, se deben aplicar conjuntamente las técnicas indicadas usando cada una de ellas como comprobación de las otras (Ochoa, 2006).

Antes de hacer una estimación, el planificador del proyecto debe comprender el ámbito del software a construir y generar una estimación de su tamaño.

En la Universidad de las ciencias informática la mayoría de las veces se realiza la estimación en los proyectos sólo una vez en todo su proceso de desarrollo, y además empleando un solo método, lo que frena la adquisición de experiencia y conocimiento por parte de la Universidad en esta importante tarea para la realización de un proyecto con calidad. La utilización de un solo método trae como consecuencia la imposibilidad de poder comparar los resultados que se obtengan entre los métodos para poder asegurar la confiabilidad de las estimaciones realizadas. Este método que se utiliza en la gran mayoría de los casos es el de estimación por puntos de casos de uso que plantea que se puede predecir ciertos parámetros de un sistema a partir de sus requisitos, expresados en los casos de usos, a pesar de ser el COCOMO2 el más utilizado en el mundo.

Con este método aparecen otros errores que no dependen sólo de la persona encargada de realizar la estimación, sino además de las que hicieron la descripción detallada de los casos de uso, que son quienes determinan la cantidad de transacciones existentes. Hay que destacar también que el proceso de estimación la mayoría de las veces no lo realiza una persona especializada en esta tarea, lo que trae como consecuencia que los resultados no sean los más exactos a la realidad.

También se tiene que debido a que la Universidad tiene poco tiempo de existencia y es el primer centro en entrar en el mundo de la creación de software tiene poca experiencia en este campo. Tampoco se cuenta en general con la cantidad de información histórica necesaria para realizar una estimación lo más exacta posible aunque existen áreas de desarrollo en las que se encuentra cierta acumulación de experiencia y a pesar de esto no se utiliza el método de Humphrey.

Esta investigación pretende evaluar en proyectos de Realidad Virtual de la Universidad una metodología para la estimación de tiempo de duración de proyectos basada en los elementos de integración entre los métodos Boehm y Humphrey.

Después de dar a conocer la situación problémica surge como **problema científico** ¿Cuáles son los resultados que ofrece evaluar los elementos de la integración de las teorías de Boehm y Humphrey para la estimación de la duración de proyectos de software para las aplicaciones de realidad virtual en la Universidad de las Ciencias Informática?

Definiéndose en la investigación como **objeto de estudio** proyectos de software para aplicaciones de realidad virtual.

Se tiene como **campo de acción** la estimación de la duración de un proyecto de software para aplicaciones de realidad virtual 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 realidad virtual en la Universidad de las Ciencias Informáticas.

La **idea a defender** por este trabajo investigativo es la que se expone a continuación: La integración de los puntos de contacto que tienen los métodos de Humphrey y Boehm provee una estimación más precisa de la duración de Proyectos de Realidad Virtual.

Concretándose los **objetivos específicos** en:

- Caracterizar las métricas de estimación.
- Caracterizar los modelos de estimación.
- Validar la posible integración entre los elementos comunes de las teorías de Boehm y Humphrey en los proyectos de realidad virtual.

Las **tareas investigativas** para llevar a cabo estos objetivos son:

- Análisis de métricas de software vinculadas con la estimación para la duración de un proyecto.
- Análisis del método de puntos por casos de usos.
- Analizar las teorías de estimación planteadas por Boehm y Humphrey.
- Analizar los elementos de integración entre las teorías de Boehm y Humphrey en la duración de proyectos de software.
- Validar los puntos de integración entre las teorías de Boehm y Humphrey para la estimación de tiempo en los proyectos de Realidad Virtual.

Para el cumplimiento de estas tareas se utilizaron varios métodos de investigación:

Métodos teóricos

Histórico-lógico: En el principio de la investigación se desarrolla un estudio profundo sobre la situación que se analiza; revisando de forma crítica cada uno de los documentos para detallar la importancia de los diferentes métodos desarrollados para la estimación de tiempo que se llevan a cabo en la actualidad.

El analítico sintético: Es el método empleado para el procesamiento de la documentación.

El Hipotético deductivo: Permite a partir del problema concreto plantear los objetivos específicos e idea a defender que son resueltos a través de la investigación.

Métodos empíricos

Entrevista: Se realizó una entrevista al inicio de la investigación para detallar el proceso de estimación en los proyectos de Realidad Virtual en la Universidad de las Ciencias Informáticas, fundamentalmente en la facultad #5, y a partir de la cual se trazaron los lineamientos para la investigación.

Encuesta: Se realizaron encuestas con el fin de obtener y analizar los resultados de la investigación.

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

Capítulo 1: Principales conceptos. La gestión y la estimación de proyecto. Modelos y métricas de estimación.

Capítulo 2: Caracterización del método por Puntos de Casos de Uso y sus principales limitaciones. Teorías de Boehm y Humphrey. Análisis de los puntos de integración.

Capítulo 3: Características de los proyectos de Realidad Virtual. Situación de los proyectos de Realidad Virtual en la Universidad de las Ciencias Informáticas en cuanto a la estimación. Validación de los puntos de integración en este tipo de proyecto mediante el juicio de expertos.

Capítulo 1: Fundamentación Teórica

1.1. Introducción

La planificación tiene una gran importancia ya que disminuye el grado de incertidumbre y aumenta en gran medida las posibilidades de éxito de un proyecto de software que no es más que la satisfacción de las necesidades del cliente. En el presente capítulo se abordaran los principales conceptos de la Gestión de proyecto planificación de proyectos y dentro de esta más específicamente la estimación. También se abordaran con una breve introducción en el modelo COCOMO II, en el Proceso Personal del Software y del método PROBE. Dentro del método de Estimación Basado en Proxy se explicara acerca del método estadístico de Regresión Lineal el cual este usa para ajustar sus datos.

1.2. Antecedentes de la Gestión de proyectos y la Estimación

El desarrollo de productos, la prestación de servicios, o la organización de la propia empresa son trabajos que pueden tomar la forma de proyectos o de operaciones, según los casos. En todos estos casos se necesita llevar a cabo el sobre la cantidad de recursos y la cantidad de personas que se necesitan para poder realizar el mismo con éxito.

En ambos casos se comparten tres características comunes:

- Los realizan personas.
- Para su ejecución se dispone de recursos limitados.
- Se llevan a cabo siguiendo una estrategia de actuación.

Cada proyecto tiene objetivos y características propias y únicas. Algunos necesitan el trabajo de una sola persona, y otros el de cientos de ellas; pueden durar unos días o varios años. Es necesario conocer de antemano un aproximado del tiempo que tardara en realizarse y de los posibles riesgos que puede correr este para tomar precauciones y estar preparados para enfrentar los peores casos.

Los proyectos existen desde siempre. Cualquier trabajo para desarrollar algo único es un proyecto, pero la gestión de proyectos es una disciplina relativamente reciente que comenzó a forjarse en los años sesenta.

En los años 50, el desarrollo de grandes proyectos militares requería la coordinación del trabajo conjunto de equipos y disciplinas diferentes en la construcción de sistemas únicos. Bernard Schriever, arquitecto de desarrollo de misiles balísticos Polares es considerado el padre de la gestión de proyectos, porque desarrolló el concepto de "concurrencia" integrando todos los elementos del plan de desarrollo en un solo programa y presupuesto, ejecutándolos en paralelo y no secuencialmente. Consiguió de esta forma reducir considerablemente los tiempos de ejecución de los proyectos (Palacio, 2004).

El desarrollo de sistemas complejos que requerían el trabajo conjunto y sincronizado de varias disciplinas o ingenierías hizo evidente en los años 60 la necesidad de desarrollar métodos de organización y de trabajo para evitar los problemas que se repetían con frecuencia en los proyectos: (Palacio, 2004)

- Desbordamiento de agendas.
- Desbordamiento de costes.
- Calidad o utilidad del resultado obtenido.

Para dar respuesta a esta necesidad, a partir de los años 60 surgieron organizaciones que han desarrollado el cuerpo de conocimientos y las prácticas necesarias para gestionar esos trabajos con las mejores garantías de previsibilidad y calidad de los resultados. Ese cuerpo de conocimientos se ha ido desarrollando y configurando como el currículo de una nueva profesión garante del éxito de los proyectos: La gestión de proyectos. Las organizaciones más relevantes en esta línea son:

- Internacional Project Managenet Association (IPMA), fundada en 1965
- Project Management Institute (PMI) constituido en 1965
- Prince2, que comenzó a trabajar en 1989.

El principal objetivo de esta es conseguir que el desarrollo se lleve a cabo según lo "previsto"; y basa el éxito del proyecto en los tres puntos apuntados: agendas, costes y calidad.

Dentro de la Gestión de proyecto existe una situación muy problemática en el proceso de estimación de software. Ya en los años 70 se comenzó a hablar del proceso de estimación del software. Sin embargo, y desafortunadamente, el arte y la ciencia de la estimación están hoy en día en su infancia. La industria del software sigue fuera de control, con costes y tiempos desmedidos. 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.

1.3. Gestión de proyectos.

Esta es la disciplina de organizar y administrar recursos de manera tal que se pueda culminar todo el trabajo requerido en el proyecto dentro del alcance, el tiempo, y coste definidos. Un proyecto es un esfuerzo temporal, único y progresivo, emprendido para crear un producto o un servicio también único. Se tiene que la clave del éxito en la gestión del desarrollo del software es:

- Una adecuada gestión del proyecto de desarrollo del software.
- Una adecuada gestión del proceso de software.

¿Qué se entiende por gestión del proyecto de desarrollo del software?

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

¿Qué es entonces, la gestión del proceso del software?

La gestión del proceso es el conjunto de técnicas y actividades que permiten una adecuada gestión de los procesos personales de los constructores y de los productos que participan en el proyecto.

También se tiene que un proyecto es:

Una acción en la que recursos humanos, financieros y materiales se organizan de una nueva forma para acometer un trabajo único. En este trabajo, dadas unas especificaciones y dentro de unos límites de costes y tiempo, se intenta conseguir un cambio beneficioso dirigido por unos objetivos cualitativos y

cuantitativos. El hecho de que sea un trabajo único para producir un cambio beneficioso significa que un proyecto lleva un alto grado de incertidumbre y riesgo, y por lo tanto su éxito dependerá en gran medida de una adecuada gestión (Adriana Gómez, 1994).

1.3.1. Tareas críticas dentro de la Gestión de proyectos.

Hay tres tareas que son críticas dentro de la gestión de proyectos que se deben cumplir si se desea que el proyecto termine correctamente. Estas tareas son:

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

Estimación de duración, coste y esfuerzo

Esta es la primera tarea en la gestión de proyecto. La estimación es el proceso de predicción de personal, del esfuerzo, de los costes y de la planificación que se requerirá para realizar todas las actividades y construir todos los productos asociados con el proyecto (anónimo, 2005).

Uno de los parámetros críticos de la estimación es determinar su exactitud. La estimación puede realizarse a partir de datos históricos o con herramientas. Curiosamente, en la actualidad, está ocurriendo algo sorprendente y es que algunas herramientas actualmente existentes proporcionan una estimación más exacta que la obtenida por la empresa a partir de sus datos históricos.

Planificación

La planificación cumple dos propósitos principales en las organizaciones: el protector y el afirmativo. El propósito protector consiste en minimizar el riesgo reduciendo la incertidumbre que rodea al mundo de los negocios y definiendo las consecuencias de una acción administrativa determinada. El propósito afirmativo de la planificación consiste en elevar el nivel de éxito organizacional.

La planificación de un proyecto se define como el proceso de selección de una estrategia para la producción de unos productos finales dados, así como la definición de las actividades a realizar para conseguir ese objetivo, la concurrencia y solapamiento de dichas actividades. También debe asignar recursos a las actividades anteriores en función del plan establecido. La estimación y la planificación son actividades relacionadas pero difieren en su alcance y propósito. La estimación normalmente está orientada al proyecto en su conjunto, mientras que la planificación esta dirigida a los individuos. Obviamente, debe existir una fuerte correlación entre la estimación realizada y las tareas específicas a realizar día a día por el equipo de proyecto. La estimación se realizará al principio del proyecto, precisamente para pedir presupuesto, saber cuánta gente se necesita etc., la planificación es la etapa donde se asigna exactamente quién hace qué y en cuánto tiempo. Una diferencia de técnica entre las herramientas de planificación y estimación es que estas últimas son normalmente sistemas expertos, construidos a partir de las reglas derivadas de miles de proyectos. Las herramientas para la planificación, en cambio, no son sistemas expertos, sino herramientas para ser utilizadas por personas expertas. La razón para esta diferencia es que las herramientas de estimación están basadas en miles de proyectos y pueden llegar a alcanzar una gran exactitud, pero el trabajo día a día de las personas que participan en el proyecto con sus conocimientos, planes de vacaciones e interrupciones requieren un director de proyectos expertos y casi con ajustes diarios de la planificación. Se puede sacar una conclusión y es que las herramientas de planificación dan los mejores resultados con los mejores líderes. En cambio, las herramientas de estimación pueden aumentar y mejorar las capacidades de los nuevos jefes de proyecto o de los expertos cuando tienen que planificar proyectos en los que no existe una experiencia previa.

Seguimiento

El seguimiento es la recolección de datos y su almacenamiento, sobre tiempos, recursos, costes, e hitos asociados con un proyecto, para el análisis y estudio de la evolución real de dicho proyecto, comparando el progreso real con el planificado. Desafortunadamente, el seguimiento de los proyectos de desarrollo de software no ha sido todo lo correcto que cabría esperar. Muchos sistemas son inexactos y entre el 35% y el 75% de los costes reales del software no son registrados.

Las mayores omisiones en el seguimiento son debidos a:

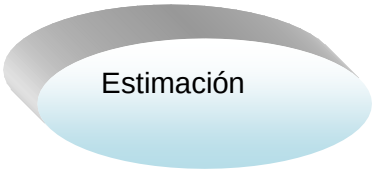
- Tiempo extra de profesionales no registrado.

- Coste de viajes y reuniones.
- Esfuerzo de los directivos.
- Esfuerzo de los usuarios en tareas técnicas, como escritura de manuales, pruebas o participación en revisiones.
- Desarrollos iníciales antes de comenzar el proyecto.

Existen muchas más, pero éstos son una muestra de la situación, y de la poca exactitud del seguimiento.

Los conceptos anteriores pueden dar la falsa impresión de que cada uno de los procesos descritos es independiente, discreto y de que se aplican una sola vez. El hecho es que éste no es el caso. Cuando se tiene una estimación inicial sobre el proyecto que se va a desarrollar, se debe definir una planificación para el proyecto siempre dentro del marco de esa estimación, es decir, la salida del proceso de estimación debe ser una de las entradas del proceso de planificación. Una vez realizada la planificación se comienza el seguimiento del proyecto. Por lo tanto, las entradas del proceso de seguimiento serán la estimación y planificación del proyecto, además de los datos reales recogidos mientras evoluciona el proyecto. Durante la realización del proceso de seguimiento se puede producir una replanificación si se aparta del plan original. Una fuerte desviación durante el seguimiento puede ser la consecuencia, por ejemplo, de un cambio en la naturaleza del proyecto. En ese caso, se necesita una reestimación y replanificación.

La gestión del desarrollo del software es ineficaz a causa de que dicho desarrollo es extremadamente complejo, disponiéndose de pocas medidas para guiar y evaluar el proceso. De esta manera sin una estimación eficaz y exacta, la planificación y el seguimiento eficaces son prácticamente imposibles de conseguir.



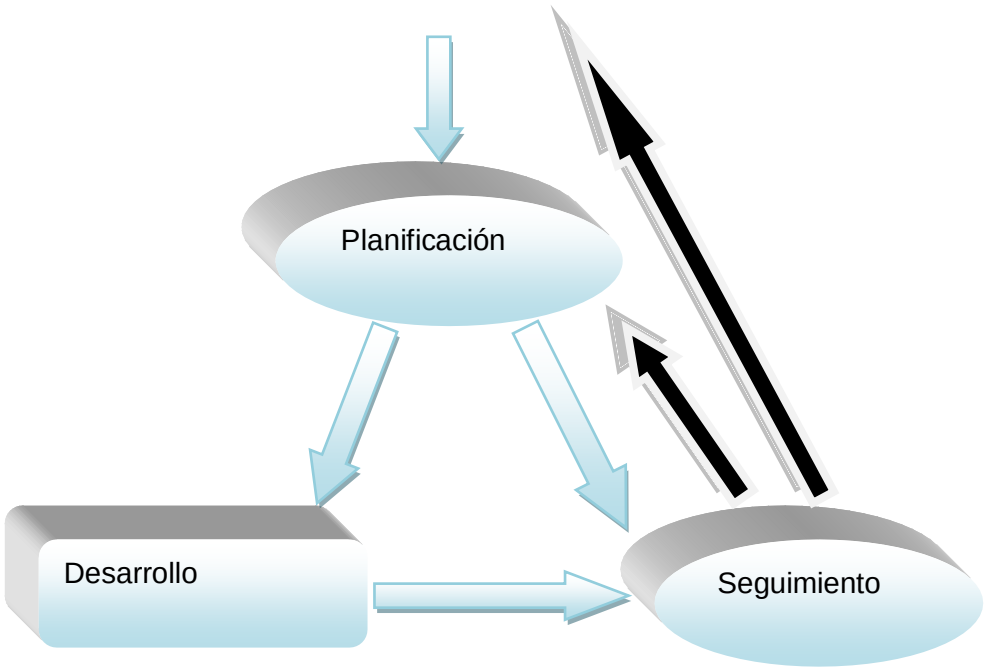


Figura 1: Relación entre las tareas de Planificación, Estimación y Seguimiento

1.4. Estimación de Software.

La estimación vista desde el punto de vista de un diccionario, una estimación es un conjunto aproximado de valores para algo que ha de ser hecho. En el mundo del desarrollo de software, Larry Putnam ha apuntado que la gestión del desarrollo de software considera la estimación como una actividad que permite obtener, principalmente, respuestas aproximadas a las siguientes preguntas: (anónimo, 2005)

¿Cuánto costará?

¿Cuánto tiempo llevará hacerlo?

La respuesta a estas dos preguntas constituye el quid del tema que aquí se contempla. Sin embargo, y como se puede prever ésta no es tan sencilla.

Las razones por lo que se hace tan difícil dar respuesta a estas preguntas son:

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

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

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

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

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

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

- La rapidez con la que cambia la tecnología de la información y las metodologías de desarrollo de software son problemas para la estabilización del proceso de estimación. Por ejemplo, son difíciles de predecir la influencia de nuevos bancos de pruebas, lenguajes de cuarta y quinta generación, estrategias de prototipo, y de técnicas y herramientas novedosas en general.

- Un estimador puede no tener mucha experiencia en estimar desarrollos, especialmente de proyectos largos.

- Existe una tendencia aparente de los desarrolladores hacia la subestimación. Un estimador suele elegir una porción de software que debería tomar para luego extrapolarlo al resto del sistema, normalmente se ignoran los aspectos no lineales del desarrollo de software, por ejemplo, la coordinación y la gestión.

- El estimador estima el tiempo que le llevaría ejecutar personalmente una tarea, ignorando el hecho de que, a menudo, una parte del trabajo la realiza personal menos experimentado, con un índice de productividad menor.

- Existen malas interpretaciones en las relaciones lineales entre la capacidad requerida por unidad de tiempo y el tiempo disponible. Esto significa que el software desarrollado por 25 personas en dos años podrá ser llevado a cabo por 50 personas en un año. Esta interpretación es errónea.

- El estimador tiende a reducir en algún grado las estimaciones para hacer más aceptable la oferta.

- Influyen un gran número de factores en el esfuerzo y duración de un desarrollo de software. Estos factores se llaman “drivers de coste” o disparadores de coste. Ejemplos de estos disparadores son el tamaño y complejidad del software, el compromiso y participación del usuario, la experiencia del equipo de desarrollo. En general estos disparadores de coste son difíciles de determinar.

Algunos ejemplos de disparadores de coste son:

- El producto software que se tiene que desarrollar.
- El significado de la producción.
- El personal de producción.
- La organización de producción.
- Usuario/organización del usuario.

Los drivers de costo actúan directamente sobre los resultados que se obtienen a la hora de estimar, pero la mayor dificultad radica en determinar cuáles son los disparadores de coste más importantes en cada situación específica, cuáles son sus valores y cómo influyen en el esfuerzo y la duración de cada proyecto. Se tiene mucha dificultad a la hora de determinar cuando un desarrollador es experimentado y cuando no. Es muy difícil tomar una unidad de medida para aplicarla a los disparadores de costo, generalmente se usa: mucho, moderado, poco.

1.5. Métricas de software

Cuando se lleva a cabo un proyecto de software es preciso calcular las estimaciones de este en cuanto a costo y esfuerzo, esto se hace aplicando métricas de software para poder adquirir información la cual se utilizará para mejorar los procesos de gestión y el producto que se obtiene mediante estos.

La medición es algo que se utiliza mucho en el mundo de la ingeniería pero existen interrogantes con respecto a:

¿Cuáles son las métricas apropiadas para el proceso y para el producto?

¿Cómo se deben utilizar los datos que se recopilan?

¿Es bueno usar medidas para comparar personas, procesos o productos?

Estas son interrogantes que surgen debido a que los proyectos de software no se han medido antes por sus características de ser únicos. La medición es indispensable en el mundo de la ingeniería. Se mide

consumo, dimensiones físicas, potencia, fuerza, voltaje, entre otras características. Desafortunadamente en el mundo de la ingeniería del software esto se sale de lo normal ya que se hace sumamente difícil decir qué se va a medir y cómo se van a evaluar esas medidas. Existen muchas razones por las cuales se debe medir un producto entre ellas se encuentran:

- Para poder obtener el grado de calidad que tiene el producto.
- Para poder medir el grado de productividad de los desarrolladores del software.
- Para poder determinar los beneficios en cuanto a productividad y calidad, nivel de exactitud y efectividad de nuevos métodos y herramientas de la ingeniería de software.
- Para establecer una línea base de la estimación.
- Para justificar el uso de nuevas herramientas de ingeniería de software.

A continuación se tienen dos definiciones de modelo. Una es más sencilla e informal y la otra es formal:

Modelo: es el proceso por el cual se asignan números o símbolos a atributos de entidades del mundo real para que lo describan de acuerdo a reglas definidas claramente. Se puede definir una entidad y un atributo de la siguiente manera:

Entidad: es un objeto o un evento del mundo real.

Atributo: es una propiedad de una entidad.

Se miden atributos de las entidades. Los números y símbolos utilizados son abstracciones que usan para reflejar la percepción mundo real y preservan las relaciones que se observan entre las entidades.

Modelo: es una abstracción de la realidad, que permite abstraer detalles y visualizar una entidad o concepto desde una perspectiva particular. El modelo del mapeo debería suplementarse con un modelo el dominio del mapeo, es decir con un modelo de cómo se relacionan las entidades con sus atributos. Ejemplo: para medir la longitud de un programa se necesita un modelo del programa. En el proceso de

ediciones existe un peligro: focalizar demasiado en el sistema matemático formal y no lo suficiente en el empírico.

Los tipos de medidas en el mundo del software pueden agruparse en dos categorías: medidas directas y medidas indirectas. Las **medidas directas** están en el proceso de gestión de software como el costo, el esfuerzo, las líneas de código fuente, velocidad de ejecución, el tamaño en memoria y en la cantidad de defectos en un periodo de tiempo determinado. En las **medidas indirectas** se encuentra la funcionalidad, calidad, complejidad, eficiencia, fiabilidad, facilidad de mantenimiento, entre otros aspectos, Productividad de programadores (LOC/unidad de tiempo: conflictiva), Densidad de defectos (nro. de defectos/tamaño), Eficiencia en detección de errores (nro. defectos detectados/nro. defectos) (S, 2004).

Hay diferentes formas en las que pueden ser utilizadas las Métricas de Software, algunas de las cuales constituyen una especialidad por si solas. La más consolidada de las áreas en el estudio de las métricas es la correspondiente a las técnicas de estimación de costes y tamaño. Existen distintos paquetes en el mercado que proporcionan estimación del tamaño del software a desarrollar, coste de desarrollo del sistema y duración del proyecto de desarrollo o mejora del software. Estos paquetes están basado en modelos de estimación y el más conocido es el COCOMO, desarrollado por Barry Boehm en 1981.

Existe un cierto número de métricas que intentan cuantificar el tamaño del software. La métrica más utilizada, líneas de código, tiene el inconveniente obvio de que sus valores no pueden ser medidos hasta que el proceso de codificación ha finalizado. Los Puntos de Función, y los Bang de DeMarco tienen la ventaja de ser medibles durante los primeros pasos del desarrollo.

1.5.1. Líneas de Código

Líneas de Código: La medida más utilizada de la longitud del código fuente de un programa es el Número de Líneas de Código (Lines of Code en ingles, abreviado LOC). Sin embargo, esta métrica puede calcularse de muchas maneras. Estas diferencias afectan al tratamiento de las líneas en blanco y las líneas de comentarios, las sentencias no ejecutables, las instrucciones múltiples por línea y las múltiples líneas por instrucción. Además, deberían contarse las líneas reusables de código.

La definición más común es la siguiente:

Una línea de código es cualquier línea de un texto de un programa que no es un comentario o línea en blanco, sin tener en cuenta el número de instrucciones o parte de instrucciones en la línea. Esta definición incluye todas las líneas que contienen cabeceras de programas, declaraciones e instrucciones ejecutables y no ejecutables. Esta medida se suele representar por NCLOC (No Comentary Lines of Code).

Sin embargo, en algunos casos, por ejemplo cuando se desea conocer qué capacidad de almacenamiento se necesita para el código fuente o cuántas páginas se van a imprimir, esta medida debe incluir los comentarios.

Como puede verse no es una medida que refleje la longitud real de un programa. Su justificación está en el uso que se ha hecho de ella en ciertos modelos para determinar el esfuerzo desde el punto de vista de evaluar la productividad. Sin embargo, si se quiere conocer la longitud real del programa esta sería:

$$LOC = NCLOC + CLOC$$

Donde CLOC (Comentary Lines of Code(es el número de líneas de comentarios).

Una medida indirecta de la densidad de comentarios seria:

$$CLOC / LOC$$

En general, es interesante obtener ambas medidas (NCLOC Y LOC) ya que expresan diferentes conceptos.

Cuando se intenta utilizar esta medida (líneas de código) en términos de productividad surgen dos problemas:

No se tiene en cuenta el concepto de reutilización.

No se tiene en cuenta el concepto de costes fijos ni tareas que se desarrollan que no producen instrucciones.

Por ello, no debe ser utilizada esta medida directamente en la estimación de esfuerzo o productividad.

Cuando se esté buscando la noción pura de longitud existen dos alternativas aceptables si se quiere utilizar bajo el concepto de ratio:

- Medir la longitud en términos de número de bytes de almacenamiento requerido para contener el texto del programa.
- Medir la longitud en términos de número de caracteres en el texto del programa. (CHAR, del inglés Character)

Si se conoce el número medio de caracteres por línea de texto, NL; el número de líneas sería:

$$LOC = CHAR/NL$$

1.5.2. Puntos de Función

Realizada por Allan Albercht en 1979 y revisada a continuación en 1983, esta técnica está basada (orientada) en la teoría de la "ciencia del software" desarrollada por Halstead, la cual está orientada al análisis del proceso de construcción de programas y se basa en la medida del número de "unidades sintácticas básicas"

(Operadores y operandos)

No se fija en el número de LDC sino en su funcionalidad.

La finalidad de la técnica de los puntos función es estimar el tamaño de un producto software y el esfuerzo asociado a su desarrollo, expresado éste en horas trabajadas por punto función, en las etapas previas a su desarrollo. Los estudios realizados sobre la utilización de este método reflejan la bondad del mismo y la existencia de un elevado grado de correlación entre el número de líneas de código (LDC) y la estimación total de los puntos función.

Etapas del método:

Contar las funciones de usuario.

Ajustar el modelo en función de la complejidad del proceso.

Contar las funciones de usuario. En la etapa primera, se definen cinco tipos de funciones de usuario:

- Entradas (al sistema): entradas de usuario que proporcionan al sistema diferentes datos orientados a la aplicación.
- Salidas: salidas de usuario que le proporcionan a éste información sobre la aplicación.
- Consultas: peticiones de usuario que como resultado obtienen algún tipo de respuesta en forma de salida.
- Ficheros lógicos internos o archivos maestros: número de archivos lógicos maestros (agrupación lógica de datos).
- Ficheros o interfaces externos: interfaces legibles (archivos de datos en cinta o disco) utilizados para transmitir información a otros sistemas.

Ajustar el modelo en función de la complejidad del proceso. El recuento de las funciones de usuario se realiza tras una clasificación previa de éstas en tres niveles de complejidad:

- Simple
- Medio
- Complejo

1.6. Modelos de Estimación

Uno de los modelos de estimación es el COCOMO. En el año 1981 Barry Boehm publica el modelo COCOMO, acorde a las prácticas de desarrollo de software de aquel momento. Durante la década de los 80, el modelo se continuó perfeccionando y consolidando, siendo el modelo de estimación de costos más ampliamente utilizado en el mundo. Al aparecer las computadoras personales y generalizarse su uso, surgieron algunas implementaciones. Varias empresas comenzaron a comercializar herramientas de estimación computarizadas. En el año 1983 se introduce el lenguaje de programación Ada (American

National Standard Institute) para reducir los costos de desarrollo de grandes sistemas. Algunos aspectos de Ada provocaron un gran impacto en los costos de desarrollo y mantenimiento, así Barry Boehm y Walker Royce definieron un modelo revisado, llamado Ada COCOMO (Giraldo, 2006).

También se tiene que el SEI (del inglés, Software Engineering Institute) propone desde hace algunos años un método para la estimación del esfuerzo llamado COCOMO II. Éste método está basado en ecuaciones matemáticas que permiten calcular el esfuerzo a partir de ciertas métricas de tamaño estimado, como el Análisis de Puntos de Función y las líneas de código fuente (en inglés SLOC, Source Line Of Code).

COCOMO II es un modelo que permite estimar el coste, esfuerzo y tiempo cuando se planifica una nueva actividad de desarrollo software. Está asociado a los ciclos de vida modernos. El modelo original COCOMO ha tenido mucho éxito pero no puede emplearse con las prácticas de desarrollo software más recientes tan bien como con las prácticas tradicionales. COCOMO II apunta hacia los proyectos software de los 90 y de la primera década del 2000, y continuará evolucionando durante los próximos años.

Dentro de los modelos de Estimación se tiene el de puntos por Casos de Usos, Este método se desarrollo en el año 1993 por Gustav Karner para poder finalmente obtener estimaciones de esfuerzo sobre productos de software orientados a objetos y posteriormente refinado por muchos otros autores. Se trata de un método de estimación del tiempo de desarrollo de un proyecto mediante la asignación de "pesos" a un cierto número de factores que lo afectan, para finalmente, contabilizar el tiempo total estimado para el proyecto a partir de esos factores. Una de las principales limitaciones del método es que no existe una teoría de cómo escribir o estructurar correctamente los casos de uso, por lo que todas las medidas de tamaño y estimación serán afectadas por la rigurosidad de los analistas.

1.7. Proceso Personal del Software (PSP)

PSP fue creado por Watts Humphrey y se dio a conocer a partir de 1995. Este enseña los pasos que se deben seguir para medir, planificar y analizar un trabajo, y ajustar este proceso a otros trabajos posteriores.

PSP proporciona los métodos para hacer este trabajo. Concretamente esta metodología debe ser efectiva aproximadamente en el nivel 2, el cual se define como sigue:

Se establecen los procesos de gestión del proyecto, para hacer seguimientos del coste, de la planificación y de la funcionalidad. Para repetir éxitos anteriores en proyectos con aplicaciones similares se aplica la disciplina necesaria para el proceso.

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 postmortem, 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.

Los pasos que se necesitan para llevar a cabo un trabajo utilizando PSP son:

- Medición Personal (PSP0)
- Planificación Personal (PSP1)
- Calidad Personal (PSP2)
- Proceso Personal Cíclico (PSP3)
- Medición Personal (PSP0): Gestión y seguimiento del Tiempo

Se debe aprender como llenar los formularios del PSP y anotar los datos del trabajo personal.

Gestión del Tiempo

La mejor forma de gestionar el tiempo es comprender cómo se utiliza, lo que exige que se sepan catalogar las actividades, anotar el tiempo empleado en cada actividad y almacenar esos datos en un lugar apropiado.

Libro de Ingeniería

Para poder llevar acabo la tarea propuesta se necesita una agenda o libro para seguir el rastro al tiempo. También Se utilizará para otras cosas como almacenar asignaciones, anotar los compromisos, apuntar notas y como un libro de trabajo para diseñar ideas y cálculos.

Seguimiento del Tiempo

Se necesita almacenar en un formulario especial la fecha y la duración de las actividades que se realizan en cada jornada y de las interrupciones que se presentan. El manejo de las interrupciones es muy importante, mediante su identificación es posible eliminarlas para realizar un trabajo más eficaz y de más calidad.

Planificación Personal (PSP1)

Se debe realizar una planificación personal del trabajo. Este paso proporciona una estimación del tamaño y de los recursos utilizados.

Planificación del Período de Tiempo

Conviene hacer un Resumen de Actividad Semanal. Es un nuevo reporte que se hace con las actividades de una semana.

Planificación del Producto

Es otro reporte para almacenar los datos referentes al tiempo estimado en realizar un proyecto. Es un registro de planificación que contiene datos referentes al producto.

Relación entre el Tamaño del Programa y el Tiempo de Realización

Como las tareas varían en tamaño y complejidad es útil tener una manera de comprobar sus tamaños. Basándose en la experiencia se puede estimar el tiempo de realización, empleando otro formulario para ello. Se puede estimar el tiempo de realización de un programa, examinando los requisitos del programa que va a desarrollarse (numero de ciclos, listas de datos, cálculos, etc.) y después hacer una clasificación por tamaño del nuevo programa.

Gestión de Compromisos

Debe realizarse una lista de compromisos donde se anotará la fecha de cada compromiso y la cantidad de tiempo que probablemente se utilizará. Luego se establecerá una calendarización empleando Mapas de Gant.

Bases para realizar un Seguimiento del Trabajo

Proporciona una definición de cada tarea, una estimación del tiempo, de los recursos requeridos y un marco de gestión de revisión y control. Se hace un Resumen de Planificación del Proyecto.

Calidad Personal (PSP2)

Para realizar un buen trabajo se deben tener en cuenta los defectos que los programadores introducen en el código del programa. Estos defectos, en la mayoría de los casos, se deben a errores de tipos, por descuido o de nombres, pero en cualquier caso se deben prevenir y corregir.

Tipos de Defectos

Los defectos se clasifican para ver que categorías causan la mayoría de los problemas y poder así prevenirlos y almacenarlos mejor. Se tienen errores de documentación, sintaxis, construcción, asignaciones, interfaz, chequeo, datos, función, sistema y entornos.

Registro de Almacenamiento de Defectos. Los datos referentes a los defectos deben almacenarse, deben almacenarse en un apartado especial del Resumen de Planificación del Proyecto.

Métodos para Encontrar y Arreglar Defectos.

Se puede utilizar el propio compilador. Se deben realizar pruebas con los datos apropiados. Y desde luego repasando el código fuente. Se puede eliminar los defectos empleando pequeños prototipos de funciones poco familiares o procedimientos antes que se usen en un programa.

Estimación de defectos.

Se deben estimar los defectos tomando en consideración el número de líneas de código del programa.

Costo de Calidad

Para obtener software libre de defectos, se necesitará tomar en cuenta el tiempo empleado en encontrarlos, por lo que para evaluar el Costo de la Calidad, se deben considerar los costos del fracaso, de apreciación y de prevención.

Proceso Cíclico (PSP3)

PSP3 es el último paso del PSP. Hasta ahora se ha visto un proceso lineal para construir programas pequeños. Este último paso incluye métodos para uso individual que se utilizan para desarrollar programas a gran escala.

Ventajas de este proceso

- Se logra aumentar la productividad de los programadores en forma personal.
- Es posible mejorar la forma de administrar el tiempo.
- Se logra realizar una buena planificación del trabajo.
- Se consigue una manera más eficiente de programar.
- Se puede evaluar la productividad de cada persona en particular.
- Es posible evaluar el costo de la calidad que se obtenga.

En el nivel 2 de PSP se comienza a aplicar la herramienta PROBE. PROxy Based Estimating. Emplea objetos como la base para estimar el tamaño de los productos. Los ingenieros hacen referencia a datos históricos de los tamaños de objetos similares que han desarrollado previamente y usan regresión linear para determinar el tamaño estimado del producto final. Este método de regresión lineal se abordara en el siguiente epígrafe.

1.8. Regresión lineal

Este método lo utiliza PROBE como método estadístico sólido para ajustar el tamaño estimado del proxy. Este realiza la estimación, basado en los datos de una muestra, el valor de una variable y correspondiente a un valor dado de otra variable x. Ello se puede hacer estimando el valor de y mediante la recta de mínimos cuadrados que ajuste los datos. La recta resultante se llama recta de regresión de y sobre x, ya que y se estima a partir de x. La regresión lineal es un conocido método para expresar una asociación como una formula lineal. Cada par de atributos se expresa como un punto de datos (xi,yi), y entonces se calcula la línea que mejor se ajusta entre la nube de puntos. La meta es expresar el atributo y en términos del atributo x, en una ecuación de la forma:

$$y=a+bx$$

Se tiene en la figura 3 un ejemplo de una recta de regresión del esfuerzo de realización de un proyecto sobre el tamaño del mismo. La teoría detrás de la regresión lineal, es dibujar una línea desde cada punto verticalmente hacia arriba o hacia debajo de la línea que denota la tendencia de los puntos. En cierto sentido la longitud de esta línea representa la discrepancia entre los datos y la línea. la intención es conseguir que esta discrepancia sea la mínima. De este modo la recta del mejor ajuste, es la línea que minimiza estas distancias.

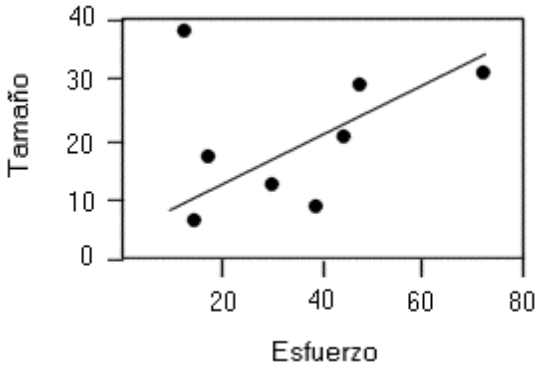


Figura 2: Recta de regresión de y sobre x

Los cálculos matemáticos necesarios para determinar los coeficientes a y b, de esta línea del mejor ajuste son directos. La discrepancia de cada punto se denomina residual, y la expresión para generar la línea de regresión lineal minimiza la suma de los cuadrados de los residuales. Se puede expresar el residual para un punto dado como:

$$r_i = y_i - a - bx_i$$

Minimizando la suma de los cuadrados de los residuales se obtienen las ecuaciones siguientes para a y para b:

$$b = \frac{\sum(x_i - m_x)(y_i - m_y)}{\sum(x_i - m_x)^2}$$

$$a = m_y - bm_x$$

La técnica de los mínimos cuadrados (least-squares LS), no hace suposiciones acerca de la normalidad de cada atributo. Sin embargo para realizar cualquier test estadístico relacionando los parámetros de regresión (por ejemplo, se puede querer determinar si el valor de b es significativamente diferente de cero), si sería necesario asumir que los valores de los residuales están normalmente distribuidos. Esta técnica se ha de utilizar con cuidado cuando existen numerosos datos atípicos, pues estos datos pueden distorsionar la estimación de a y b. Es importante mantener en la mente que la naturaleza lineal de la regresión, únicamente alude a la forma lineal de los coeficientes de los parámetros. Se pueden realizar transformaciones sobre las variables, para permitir modelado lineal.

1.1. Conclusión

En este capítulo se encuentran los principales conceptos relacionados con la estimación, la definición de gestión de proyecto, planificación, estimación, entre otros conceptos fundamentales que ayudan a comenzar a entender este mundo tan complicado y poco valorado a la hora de realizar proyectos de software. También se habla acerca de cómo surge la gestión de proyectos y de cuándo y por quién fueron creados algunos modelos de estimación, entre ellos esta COCOMO II. En el próximo capítulo se tratará como tema principal los puntos de integración entre las teorías de Boehm y Humphrey.

Capítulo 2: Puntos de Casos de Uso. Puntos de contacto entre las teorías de Boehm y Humphrey.

2.1. Introducción

En este capítulo se analizará el método de puntos por Casos de Uso y los puntos de contactos entre las teorías de Boehm y Humphrey planteados en la investigación realizada por MSc. Yadira Ruiz Contasten. Esto ayudará a realizar comparaciones entre los distintos modelos de estimación con el objetivo de determinar las ventajas y desventajas que brindan para su aplicación en este trabajo de tesis. Todo esto ayudará a entender mejor cómo aplicar esos puntos de contactos entre COCOMO II y PROBE en los proyectos de realidad virtual para evaluar los resultados del mismo y así lograr una mayor calidad en las aplicaciones de Realidad Virtual que se realizan en la Universidad.

2.2. Puntos de Caso de Uso

Este método de estimación de esfuerzo de un proyecto de desarrollo de software se realiza a partir de los casos de uso. El método utiliza los actores y casos de uso identificados para calcular el esfuerzo que costará desarrollarlos. A los casos de uso se les asigna una complejidad basada en transacciones, que son pares de pasos acción-usuario->respuesta-sistema de los escenarios de los casos de uso. A los actores se les asigna una complejidad basada en el tipo de actor, es decir, si son interfaces con usuarios o si son interfaces con otros sistemas (api o protocolo). También se utilizan factores de entorno y de complejidad técnica para afinar el resultado. Una vez asignada complejidad a actores y casos de uso y establecidos los factores técnicos y de entorno, se calculan los puntos de caso de uso no ajustados o UUCP, el TCF (factor de complejidad técnica) y el EF (factor del entorno). Con ellos, se calculan los puntos de caso de uso o UCP, que finalmente se traducen a esfuerzo en horas-hombre con un sencillo cálculo. La estimación por Puntos de Caso de Uso resulta muy efectiva para estimar el esfuerzo requerido en el desarrollo de los primeros Casos de Uso de un sistema, si se sigue una aproximación iterativa como el Proceso Unificado de Rational. En éste tipo de aproximación, los primeros Casos de Uso a desarrollar son los que ejercitan la mayor parte de la arquitectura del software y los que a su vez ayudan a mitigar los riesgos más significativos (iteraciones de Elaboración en el Proceso Unificado). Fuera de éste contexto, el método tiende a sobredimensionar el esfuerzo requerido por lo cual no se recomienda para estimar el

esfuerzo global de un proyecto. Se tiene que actualmente la metodología de análisis, diseño y POO es la más utilizada en el ámbito del desarrollo de software y dentro de esta la que más se ha adoptado es el Lenguaje Unificado de Modelado (UML). La unidad de trabajo y control en UML es el Caso de Uso por lo que entonces debería buscarse una forma de asignar unidades de medida a los casos de uso para poder estimarlos. Sin embargo hay diferentes estilos de agrupar escenarios en casos de uso, y por otra parte la complejidad de cada escenario puede tener grandes variaciones, haciendo entonces del caso de uso una unidad no especialmente apta para la estimación (Cao, 2004).

Los pasos a seguir para realizar la estimación después que se tienen los casos de uso y su descripción son:

- Calcular puntos de casos de uso sin ajustar.
- Calcular factor de complejidad técnica.
- Factor de ambiente.

2.2.1. Puntos de Casos de Uso sin ajustar (UUCP)

Como ya es sabido cuando se utilizan los casos de uso para realizar la especificación de requerimientos de sistemas deben identificarse los actores y los casos de uso y luego realizar una descripción de los mismos. Con esta información se va a calcular los Puntos de Casos de Uso sin ajustar a partir de la siguiente ecuación:

$$UUCP = UAW + UUCW$$

Donde:

UAW -> Factor de peso de actores sin ajustar.

UUCW -> Factor de peso de Casos de Uso sin ajustar.

Factor de peso de actores sin ajustar (UAW)

Se debe realizar un catastro de todos los actores del sistema y deben ser clasificados como Simple, Promedio y Complejo, de acuerdo al siguiente criterio:

- Actor Simple: Se trata de otro sistema interactuando a través de una interfaz de programación definida y conocida (API).
- Actor Promedio: Es otro sistema interactuando a través de un protocolo (como TCP/IP).
- Actor Complejo: se trata de una persona interactuando con el sistema a través de una interfaz gráfica de usuario (GUI) o página Web.

Junto a la cuenta y clasificación de los actores se debe asociar un factor de peso de a cuerdo a la siguiente tabla:

Tipo de actor	Descripción	Factor de Peso
Simple	Otro Sistema que interactúa con el sistema a desarrollar mediante una interfaz de programación (API, Aplication Programming Interface).	1
Medio	Otro sistema que interactúa con el sistema mediante un protocolo o una interfaz basada en texto.	2
Complejo	Una persona que interactúa con el sistema mediante una interfaz gráfica.	3

Tabla 1 Asignación del peso al actor

Finalmente, se cuentan los actores de acuerdo a su clasificación o grado de complejidad, multiplicando cada subtotal por su factor de complejidad y sumando cada producto obteniéndose el peso de los actores sin ajustar (UAW).

Factor de peso de Casos de Uso sin ajustar (UUCW)

Al igual que con los actores para calcular este Factor se cuentan la cantidad de Casos de uso y se les asignan un nivel de complejidad. Esta complejidad que se le asigna a cada CU depende de la cantidad de transacciones que tengan. Una transacción no es más que una secuencia de actividades atómicas. También se puede decir que se define una transacción como un evento que ocurre entre un actor y el sistema a ser modelado. El método de puntos por Casos de Uso clasifica los casos de uso en simple, medio y complejo con factores de peso 5, 10 y 15 respectivamente. La clasificación se hace en base al número de transacciones que contiene el caso de uso, 1 a 3 para simple, 4 a 7 para promedio y 8 ó más para complejos. Los criterios para asignar esta complejidad se muestran en la siguiente tabla:

Tipo de Caso de Uso	Descripción	Factor de Peso
Simple	El Caso de Uso contiene de 1 a 3 transacciones	5
Medio	El Caso de Uso contiene de 4 a 7 transacciones	10
Complejo	El Caso de Uso contiene 8 o más transacciones	15

Tabla 2 Asignación del peso a los CU

Según ciertos estudios que se han realizado esta clasificación de los casos de uso es totalmente inadecuada a los efectos de realizar una estimación. El principal motivo de esto es que un caso de uso no tiene un tamaño determinado. En un ensayo para Rational Software, John Smith [SMITH, 1999] considera como normal un promedio de 30 escenarios por caso de uso, tratando de ser conservador en el número. Un caso de uso de este tipo podría llegar a tener 30 transacciones. Si a cada escenario le correspondiera una transacción, este caso de uso tendría igual factor de peso que uno de 8 transacciones. Es decir el rango superior, 8 a infinito, da igual valor para 8 transacciones que para infinitas, lo cual es un inconveniente. Los inconvenientes persisten si se mejora la escala asignando puntos de caso de uso proporcionalmente al número de transacciones. En efecto la complejidad de un escenario puede ser tanto como la complejidad de otros 10, ó 20 ó el número que se quiera elegir y lo mismo vale para las transacciones. Por otra parte la agrupación de los requerimientos en casos de uso puede ser totalmente arbitraria haciendo que el método dé valores muy distintos para el mismo sistema. En ese sentido se ha dado el caso de un trabajo sobre el método de puntos por Casos de Uso en donde para hacer un programa de altas, bajas, modificaciones y consultas resultó un esfuerzo de programación de 4 meses

utilizando el lenguaje C++. Para un programa de altas, bajas, modificaciones y consultas, cuatro meses es a todas luces excesivo. En ese caso el método fue bien aplicado, sin embargo se consideró un caso de uso para el alta, otro para la baja, otro para la modificación y otro para la consulta.

Hoy en día se puede decir, como explica Bittner [BITTNER, 2001] que semejante división es un error, que alta, baja, modificación y consulta no conforman cuatro casos de uso. Sin embargo en el libro Writing Effective Use Cases [COCKBURN, 2000], quizás ya fuera de época, se puede leer que la opción de hacer un caso de uso para cada una de esas funciones, alta, baja, modificación y consulta, es una alternativa válida. Por otra parte hay quien, con criterio razonable, establece un caso de uso con el alta, y otro con la consulta, siendo baja y modificación extensiones de esta. Es decir, en algo tan simple y común como un programa de mantenimiento se puede considerar uno, dos o cuatro casos de uso, haciendo que la estimación llegue hasta cuadruplicarse según la separación que se haga. En conclusión, se debería entonces tener en cuenta el número de transacciones o escenarios del mismo para determinar su peso. Sin embargo teniendo en cuenta esto no resolvería el problema de la complejidad que pueda tener el conjunto de transacciones o escenarios contenidos en cada caso de uso, es decir no es simplemente un problema de cantidad sino también de complejidad.

Entonces se tiene que el factor de peso de los Casos de Uso sin ajustar (UUCW) es la suma de todos los pesos de cada caso de uso y finalmente los Puntos de Casos de Uso sin ajustar se calculan con la siguiente formula:

$$UUCP = UAW + UUCW$$

2.2.1. Cálculo de Puntos de Casos de Uso ajustados (UCP).

Después que se ha Calculado los Puntos de Casos de Uso sin ajustar hay que ajustarlos mediante la siguiente ecuación:

$$UCP = UUCP \times TCF \times EF$$

UCP: Puntos de Casos de Uso ajustados

UUCP: Puntos de Casos de Uso sin ajustar

TCF: Factor de complejidad técnica

EF: Factor de ambiente

Factor de complejidad técnica (TCF)

Este coeficiente se calcula mediante la cuantificación de un conjunto de factores que determinan la complejidad técnica del sistema. Cada uno de los factores se cuantifica con un valor de 0 a 5, donde 0 significa un aporte irrelevante y 5 un aporte muy importante. En la siguiente tabla se muestra el significado y el peso de cada uno de éstos factores:

Factor	Descripción	Peso
T1	Sistema distribuido	2
T2	Objetivos de performance o tiempo de respuesta	1
T3	Eficiencia del usuario final	1
T4	Procesamiento interno complejo	1
T5	El código reutilizable	1
T6	Facilidad de instalación	0.5
T7	Facilidad de uso	0.5
T8	Portabilidad	2
T9	Facilidad de cambio	
T10	Concurrencia	1
T11	Incluye objetivos especiales de seguridad	1
T12	Provee acceso directo a terceras partes	1
T13	Se requieren facilidades especiales de entrenamiento a usuarios	1

Tabla 3 Peso de los Factores de Complejidad Técnica

El primer factor es Sistema distribuido el cual no es más que aquel en el que dos o más máquinas colaboran para la obtención de un resultado. En todo sistema distribuido se establecen una o varias comunicaciones siguiendo un protocolo prefijado mediante un esquema cliente-servidor. A este factor se le asigna un valor de 0 a 5 en dependencia si el software va a ser centralizado o distribuido.

Otro factor es el de portabilidad sobre el cual existen varias definiciones:

Una definición sencilla:

Se dice que un programa informático (o una aplicación, o un sistema operativo) es portable cuando se puede hacer funcionar fácilmente en otro sistema distinto a aquel en el que fue diseñado.

Una definición más extensa y detallada del mismo enfoque:

La portabilidad de un software se define como su dependencia de la plataforma en la que corre. La portabilidad es mayor cuanto menor es su dependencia del software de plataforma.

Si un software puede ser compilado en plataformas diversas (x86, IA64, amd64, etc), dicho software se dice que es multiplataforma.

En algunos casos el software es "independiente" de la plataforma y puede ejecutarse en plataformas diversas sin necesidad de ser compilado específicamente para cada una de ellas, a este tipo de software se le llama interpretado, por que necesita de un intérprete para ser ejecutado en las diferentes plataformas.

El código reutilizable lo se puede definir como el código o la funcionalidad que serán probablemente útiles en uno o más proyectos futuros. En la asignación de un valor a este factor se tiene que si se va utilizar código reutilizable solo dentro del proyecto como buena práctica de programación entonces se le asigna valor 3, sino se utiliza ninguno entonces, y en el caso de que se valla a utilizar código reutilizable de una biblioteca se le asigna valor 5.

La facilidad de uso se conoce más como usabilidad acerca de la cual se puede encontrar varias definiciones. La ISO ofrece dos definiciones:

ISO/IEC 9126

"La usabilidad se refiere a la capacidad de un software de ser comprendido, aprendido, usado y ser atractivo para el usuario, en condiciones específicas de uso"

Esta definición hace énfasis en los atributos internos y externos del producto, los cuales contribuyen a su funcionalidad y eficiencia. La usabilidad depende no sólo del producto sino también del usuario. Por ello un producto no es en ningún caso intrínsecamente usable, sólo tendrá la capacidad de ser usado en un contexto particular y por usuarios particulares. La usabilidad no puede ser valorada estudiando un producto de manera aislada (Bevan, 1994).

ISO/IEC 9241

"Usabilidad es la eficiencia y satisfacción con la que un producto permite alcanzar objetivos específicos a usuarios específicos en un contexto de uso específico"

Es una definición centrada en el concepto de calidad en el uso, es decir, se refiere a cómo el usuario realiza tareas específicas en escenarios específicos con efectividad.

El Factor de complejidad técnica se calcula mediante la siguiente ecuación:

$$TCF = 0.6 + 0.01 \times \Sigma (Pesoi \times Valor\ asignadoi)$$

Factor de ambiente

El tiempo de desarrollo de un proyecto depende en gran medida de las habilidades y la experiencia que tenga el equipo que va a construir el sistema. Estos parámetros son los que se miden para calcular el factor de ambiente. Estos parámetros se calculan asignándole un grado de importancia de 0 a 5.

Factor	Descripción	Peso
E1	Familiaridad con el modelo de proyecto utilizado	1.5
E2	Experiencia en la aplicación	0.5

E3	Experiencia en orientación a objetos	1
E4	Capacidad del analista líder	0.5
E5	Motivación	1
E6	Estabilidad de los requerimientos	2
E7	Personal Part-time	-1
E8	Dificultad del lenguaje de programación	-1

Tabla 4: Peso de los Factores de Ambiente

Para los factores E1 al E4, un valor asignado de 0 significa sin experiencia, 3 experiencia media y 5 amplia experiencia (experto).

- Para el factor E5, 0 significa sin motivación para el proyecto, 3 motivación media y 5 alta motivación.
- Para el factor E6, 0 significa requerimientos extremadamente inestables, 3 estabilidad media y 5 requerimientos estables sin posibilidad de cambios.
- Para el factor E7, 0 significa que no hay personal part-time (es decir todos son full-time), 3 significa mitad y mitad, y 5 significa que todo el personal es part-time (nadie es full-time).
- Para el factor E8, 0 significa que el lenguaje de programación es fácil de usar, 3 medio y 5 que el lenguaje es extremadamente difícil.

El Factor de ambiente se calcula mediante la siguiente ecuación:

$$EF = 1.4 - 0.03 \times \Sigma (Peso_i \times Valor\ asignado_i)$$

Un ejemplo de un caso real de cómo quedaría una tabla con los valores asignados a cada factor técnico (TCF) sería:

Factor	Descripción	Peso	Valor asignado	Comentario

T1	Sistema distribuido	2	0	El sistema es centralizado
T2	Objetivos de performance o tiempo de respuesta	1	1	La velocidad es limitada por las entradas provistas por el usuario
T3	Eficiencia del usuario final	1	1	Escasas restricciones de eficiencia
T4	Procesamiento interno complejo	1	1	No hay cálculos complejos
T5	El código debe ser reutilizable	1	0	No se requiere que el código sea reutilizable
T6	Facilidad de instalación	0.5	1	Escasos requerimientos de facilidad de requerimientos de facilidad de instalación
T7	Facilidad de uso	0.5	3	Normal
T8	Portabilidad	2	0	No se requiere que el sistema sea portable
T9	Facilidad de cambio		3	Se requiere un costo moderado de mantenimiento
T10	Concurrencia	1	0	No hay concurrencia
T11	Incluye objetivos especiales de seguridad	1	3	Seguridad normal
T12	Provee acceso directo a terceras partes	1	5	Los usuarios web tienen acceso directo
T13	Se requieren facilidades especiales de entrenamiento a usuarios	1	1	Pocos usuarios internos, sistemas fácil de usar

Tabla 5: Factor de complejidad Técnica (TCF)

Un ejemplo de un caso real de cómo quedaría una tabla con los valores asignados a cada factor de ambiente (EF) sería:

Factor	Descripción	Peso	Valor asignado	Comentario
E1	Familiaridad con el modelo de proyecto utilizado	1.5	4	El grupo está bastante familiarizado con el modelo
E2	Experiencia en la aplicación	0.5	4	La mayoría del grupo ha trabajado mucho tiempo en esta aplicación
E3	Experiencia en orientación a objetos	1	4	La mayoría del grupo programa en objetos
E4	Capacidad del analista líder	0.5	5	Se encontró a un especialista
E5	Motivación	1	5	El grupo esta altamente motivado
E6	Estabilidad de los requerimientos	2	2	Se esperan cambios
E7	Personal Part-time	-1	0	Todo el grupo es full-time
E8	Dificultad del lenguaje de programación	-1	3	Se usará lenguaje C++

Tabla 6: Factor de ambiente (EF)

2.2.1. Calcular la estimación del esfuerzo

Originalmente se propuso que fueran 20 horas-hombre por cada punto de Caso de Uso. Pero sugirieron un refinamiento basado en factores ambientales. Estos plantean lo siguiente:

- Se cuentan cuantos resultados de la multiplicación del peso por el valor asignado desde el factor E1 hasta el factor E6 son menor o igual que 2 es decir menores que el nivel promedio.

- Se cuentan cuantos resultados de la multiplicación del peso por el valor asignado de E7 y E8 son mayores que 3 es decir mayores que el valor promedio.

- Después que se hace esto si da un total de 2 o menos entonces se escoge como factor de conversión el de 20 horas-hombre.
- Después que se hace esto si da un total de 4 o menor entonces se escoge como factor de conversión el de 28 horas-hombre.
- Después que se hace esto si da un total de 5 o mayor entonces se escoge como factor de conversión el de 36 horas-hombre aunque lo más recomendable efectuar cambios en el proyecto ya que el riesgo de fracaso es muy alto.

Un ejemplo de cómo quedaría el esfuerzo partiendo de una cantidad de puntos por casos de uso ajustados igual a 14.52 multiplicándolo por el factor de conversión 20 horas-hombre seria:

$$E = UCP * FC = 14.52 * 20 = 290 \text{ horas-hombre}$$

2.3. Elementos a considerar en la integración de las teorías de Boehm y Humphrey

En este epígrafe se abordara el tema de la teoría de Boehm centrándose en COCOMO 2 y también tratando la teoría de Humphrey y particularmente en su método PROBE. Uno de los modelos de estimación es el COCOMO. En el año 1981 Barry Boehm publica el modelo COCOMO, acorde a las prácticas de desarrollo de software de aquel momento. Durante la década de los 80, el modelo se continuó perfeccionando y consolidando, siendo el modelo de estimación de costos más ampliamente utilizado en el mundo. Al aparecer las computadoras personales y generalizarse su uso, surgieron algunas implementaciones. Varias empresas comenzaron a comercializar herramientas de estimación computarizadas. En el año 1983 se introduce el lenguaje de programación Ada (American National Standard Institute) para reducir los costos de desarrollo de grandes sistemas. Algunos aspectos de Ada provocaron un gran impacto en los costos de desarrollo y mantenimiento, así Barry Boehm y Walker Royce definieron un modelo revisado, llamado Ada COCOMO. En los 90, las técnicas de desarrollo de software cambiaron dramáticamente, surgieron la necesidad de rehusar software existente, la construcción de sistemas usando librerías, etc. Estos cambios comenzaron a generar problemas en la aplicación del modelo COCOMO. La solución fue reinventar el modelo. Después de algunos años y de un

esfuerzo combinado de USC-CSE (University of Southern California-Center For Software Engineering), IRUS at UC Irvine y organizaciones privadas, aparece COCOMO II (Adriana Gómez, 1994).

El PSP fue elaborado por Watts Humphrey, creador del CMM, y se encuentra descrito en su libro "A Discipline for Software Engineering", publicado por primera vez en 1995. El PSP así como el CMM (Capability Maturity Model) tiene como finalidad mejorar los procesos de desarrollo de software. El PSP se destina a una actuación individual, mientras que el CMM abarca un ámbito más amplio: el mejoramiento de la capacidad de toda una organización, en el desarrollo de aplicaciones con bajo número de errores. La mejora de las estimaciones - que la mayor parte de los proyectos se realice, en el tiempo calculado - es una de las principales metas del PSP. El planeamiento y el seguimiento de cronogramas, así como el compromiso personal del ingeniero de software con la calidad, también son objetivos del método. En 1997 un estudio realizado por varios ingenieros de software reveló que el PSP puede proporcionar una mejora del 75% en la agudeza de estimación de esfuerzo y del 150% en la calidad de la estimación de tamaño. Además, con la utilización del método, el número de sobreestimaciones y de subestimaciones quedó más equilibrado. Por otra parte la calidad del Producto aumentó en 2,5 veces. La productividad personal aumentó bastante. No en función del número de líneas de código escritas por programador, más con relación al producto, resultando en un ciclo de desarrollo de mejor calidad, con menos errores y por lo tanto, con menos tiempo empleado en la corrección de los mismos.

2.3.1. Líneas de código

Con respecto a esta métrica de software se tiene una definición aceptable pero no estándar que plantea que toda línea de un programa que no es un comentario, o línea en blanco, independientemente del número de instrucciones o fragmentos de instrucciones en ella. Esto incluye específicamente las líneas de encabezamiento de un programa, declaraciones, e instrucciones ejecutables y no ejecutables. El principal problema asociado a la métrica líneas de código es que no todas las líneas son equivalentes en su dificultad de codificación. El significado de línea de código varía de un lenguaje a otro, pero también dentro de un mismo lenguaje de programación. Las líneas de código son utilizadas como unidad de medida para estimar tamaño en COCOMO2 y en PROBE. En modelo COCOMO2 se mide el tamaño en KSLOC (miles de líneas de código fuente). Existen dos vías actualmente para calcular estas KSLOC:

- Una es Calculando los puntos de función desajustados, estos mediante un factor de conversión que depende del lenguaje que se valla a utilizar se convierte a SLOC y después se divide por 1000 y así se obtienen las KSLOC.

- Los KSLOC también se derivan de la medida de módulos software que constituirán el programa de aplicación.

En este modelo después que se obtiene el tamaño dado en miles de líneas de código se utiliza un porcentaje de rotura breakage (BRAK) para ajustar el tamaño eficaz del producto. La rotura refleja la volatilidad de los requisitos en un proyecto. Es el porcentaje de código desperdiciado debido a la volatilidad de los requisitos. Un proyecto a medida que avanza debido a cambios que surgen en los requisitos es necesario suprimir código, este código aunque no forma parte del tamaño final se debe tomar en cuenta.

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

Si se opta por utilizar directamente el valor del número de líneas de código, la meta es medir la cantidad de trabajo intelectual que se emplea en el desarrollo del programa, pero las dificultades aparecen al intentar definir medidas consistentes en diferentes lenguajes. Si se opta por utilizar los Puntos de Función sin Ajustar para determinar el tamaño del proyecto, éstos deben convertirse en líneas de código fuente en el lenguaje de implementación (ensamblador, lenguajes de alto nivel, lenguajes de cuarta generación, etc.) para evaluar la relativamente concisa implementación por Puntos de Función (ver tabla 7.6). COCOMO II realiza esto tanto en el Modelo de Diseño Anticipado como en el de Post-Arquitectura usando tablas que traducen Puntos de Función sin ajustar al equivalente SLOC. A continuación se muestra la tabla de conversión.

Lenguaje	SLOC / UFP
Ada	71

Al Shell	49
APL	32
Assembly	320
Assembly (Macro)	213
ANSI/Quick/Turbo Basic	64
Basic – Compiled	91
Basic Interpreted	128
C	128
C++	29
Visual Basic	32
ANSI Cobol 85	91
Fortran 77	105
Forth	64
Jovial	105
Lisp	64
Modula 2	80
Pascal	91
Prolog	64
Report Generator	80
Spreadsheet	6

Tabla 7: Tabla de factores de conversión

El método PROBE utiliza como unidad de medida las líneas de código LOC. En este método no se toman en cuenta las líneas en blanco o las líneas de comentarios. Si existe una línea que contenga código y comentario, entonces ésta sí se toma en cuenta como una línea de código. Este es un método de estimación basado en Proxy que le permite utilizar cualquier elemento que usted elija siempre que cumpla con las condiciones de un buen proxy. Un proxy es una característica del programa que es fácilmente visualizable en etapas tempranas del desarrollo. Las características de un buen proxy son:

- La cuenta o medida del proxy debe tener una alta correlación con el esfuerzo necesario para construir el programa.
- El proxy debe poder contarse o medirse en forma automática sobre el producto terminado.
- Debe ser fácil de visualizar al comienzo del proyecto.
- Debe ser adaptable a necesidades específicas.
- Debe adaptarse a variaciones de implantación.

Para poder aplicar este método con lo primero que se debe contar es con información histórica sobre el tamaño. Después se debe realizar el diseño conceptual el cual es necesario para:

- Relacionar los requisitos con el producto.
- Definir los elementos del producto que producirán las funciones deseadas.
- Estimar el tamaño de lo que se construirá.
- Diseños comprendidos, los diseños conceptuales pueden hacerse rápidamente.

Después de haber realizado el diseño conceptual se identifican las partes del producto que usted piensa desarrollar. El segundo paso es estimar el tamaño de las partes. Esto se hace determinando los tipos de las partes y luego juzgando la cantidad de elementos para cada una de las partes. Los tamaños relativos de los elementos se determinan a partir de datos históricos y también se juzga cuáles de estos objetos se agregaran a la biblioteca. Los elementos no son más que los métodos. Se juzga el tamaño relativo de cada parte y después se multiplica por el número de elementos, y de este modo se obtiene el tamaño de las partes añadidas. El tamaño total del programa consiste en:

- El código nuevo desarrollado.
- Código reutilizado procedente de una biblioteca.
- El código base proveniente de versiones anteriores.

El código nuevo desarrollado esta compuesto por:

- El código añadido a la base (BA).
- Nuevos objetos desarrollados (NO).
- El código base q a sido cambiado es decir código modificado (M).

Lo primero que se calcula con todos estos datos es el tamaño del Proxy (E) con la siguiente ecuación:

$$E = BA + NO + M$$

Después que se calcula el tamaño estimado del Proxy según la cantidad de datos históricos que se tiene, se utiliza el método A, B, C o D. Si se ha realizado una buena práctica del PSP entonces se debe utilizar el método A que es el que normalmente se utiliza. Cuando se selecciona el método entonces se calcula la correlación (r) que en caso de r >= 0.7 se puede utilizar el método. La correlación se calcula mediante la siguiente ecuación:

$$r = \frac{\mathrm{n} * \mathrm{Sum}(\mathrm{x} * \mathrm{y}) - \mathrm{Sum}(\mathrm{x}) * \mathrm{Sum}(\mathrm{y})}{\sqrt{(n * Sum(x^2) - Sum(x)^2) * (n * Sum(y^2) - Sum(y)^2)}} =$$

Una vez que se haya calculado r, con el tamaño estimado del Proxy (E) se calcula el tamaño proyectado del programa, el cual se calcula con la siguiente ecuación:

$$P = \beta 0_{tamaño} + \beta 1_{tamaño} * E$$

Para realizar una estimación más precisa se necesita un método de estadística sólido. La línea de tendencia de los datos se calcula con un método llamado de regresión lineal. Este produce una línea que ajusta con precisión los datos. Esta línea de tendencia o de regresión se calcula mediante la ecuación anterior. 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.

En la ecuación anterior β0 y β1 son los parámetros de regresión y se calculan a partir de sus datos históricos. El valor β1 es el promedio de tiempo de desarrollo necesario por elemento de la base de datos y β0 es el tiempo general. Estos parámetros se calculan de la siguiente forma:

$$\beta_1 = \frac{\sum_{i=1}^{n} x_i\, y_i - n x_{avg} y_{avg}}{\sum_{i=1}^{n} x_i^2 - n x_{avg}^2}$$

$$\beta_0 = y_{avg} - \beta_1 x_{avg}$$

Se comienza con n valores de x e y, donde x es el tamaño estimado del proxy (S) e y es el tamaño añadido y modificado actual o el tiempo actual de desarrollo para los n programas.

Una posible integración entre las teorías de Boehm y Humphrey con respecto a este punto de contacto es el cálculo del tamaño mediante el método PROBE como se explica anteriormente y utilizarlo en COCOMO 2 para calcular esfuerzo. La ecuación básica para calcular el esfuerzo en COCOMO, tanto en el modelo de diseño inicial como en el modelo de post-arquitectura es:

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

Esta ecuación calcula el esfuerzo nominal para un proyecto de un tamaño dado expresado en Meses persona (MM). La constante A, se usa para cortar los efectos multiplicativos de esfuerzo en proyectos de tamaño incremental. El tamaño que se calcula por PROBE se da en SLOC y este se divide entre 1000 para llevarlo a KSLOC por que esa es la unidad de medida con la cual trabaja COCOMO II. Exactamente en esta integración se pretende sustituir Size por P la cual se lleva a KSLOC. Los modelos de estimación de coste del software a menudo tienen un factor exponencial para considerar los gastos y ahorros relativos de escala encontrados en proyectos software de distinto tamaño. El exponente B se usa para capturar estos efectos. La ecuación para calcular B es la siguiente:

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

Después de calcular B se tiene los siguientes criterios:

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

- Si B = 1.0. Los ahorros y gastos de escala están equilibrados. Este modelo lineal se usa a menudo para la estimación de coste de proyectos pequeños.

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

El exponente B se obtiene mediante los denominados drivers de escala. La selección de drivers de escala se basa en la razón de que ellos son un recurso significante de variación exponencial en un esfuerzo ó variación de la productividad del proyecto. Cada driver de escala tiene un rango de niveles de valores desde Muy Bajo hasta Extra Alto. Cada nivel de valores tiene un peso, SF, y el valor específico del peso se llama factor de escala. Un factor de escala de un proyecto, SFj, se calcula sumando todos los factores y se usa para determinar el exponente de escala, B. La SFj esta compuesto por cinco factores de escala los cuales son:

- (PREC) (FLEX). Precedencia y Flexibilidad de desarrollo. Estos dos factores de escala capturan en gran parte las diferencias entre los modos Orgánico, Semilibre y Rígido del modelo original COCOMO.

- (RESL) Arquitectura/Resolución de Riesgos. Este factor combina dos factores de medida de AdaCOCOMO “Minuciosidad del diseño por revisión del diseño del producto (PDR)” y “Eliminación de riesgos por PDR”.

- (TEAM). Cohesión del Equipo. El factor de escala de Cohesión del Equipo explica los recursos de turbulencia y entropía del proyecto debido a dificultades en la sincronización de los implicados en el proyecto, usuarios, clientes, desarrolladores, los que lo mantienen, etc... Estas dificultades pueden aparecer por las diferentes culturas y objetivos de los implicados; dificultades en conciliar objetivos y la falta de experiencia y de familiaridad de los implicados en trabajar como un equipo.

- (PMAT). Madurez del proceso. Este factor se analizará mas adelante como un punto de contacto entre COCOMO 2 y PROBE.

EMi corresponde a los factores de costo que tienen un efecto multiplicativo sobre el esfuerzo, llamados Multiplicadores de Esfuerzo (Effort Multipliers). Cada factor se puede clasificar en seis niveles diferentes que expresan el impacto del multiplicador sobre el esfuerzo de desarrollo. Esta escala varía desde un nivel Extra Bajo hasta un nivel Extra Alto. Cada nivel tiene un peso asociado. El peso promedio o nominal es 1.0. Si el factor provoca un efecto nocivo en el esfuerzo de un proyecto, el valor del multiplicador correspondiente será mayor que 1.0, caso contrario el multiplicador será inferior a 1.0.

En el modelo de Diseño Inicial un reducido conjunto de parámetros de coste es utilizado. Los parámetros de coste del modelo de Diseño Inicial se obtienen por combinación de los parámetros de coste del modelo Post-Arquitectura como muestra la siguiente tabla:

Parámetro de Coste Diseño Inicial	Combinación Equivalente Arquitectura
RCPX	RELY, DATA, CPLX, DOCU
RUSE	RUSE
PDIF	TIME, STOR, PVOL
PERS	ACAP, PCAP, PCON
PREX	AEXP, PEXP, LTEX
FCIL	TOOL, SITE
SCED	SCED

Tabla 8: Multiplicadores de esfuerzo Diseño Inicial y Post-Arquitectura

En el modelo de diseño inicial hay 7 drivers y en el post-arquitectura 17. La ecuación de esfuerzo es la misma para ambos modelos. La escala de ratios del modelo de Diseño Inicial siempre tiene un total nominal igual a la suma de los ratios nominales de los elementos del modelo Post-Arquitectura con los que se han formado. Estos drivers son los siguientes:

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

- (RUSE) Reutilización Requerida. Este driver de coste del modelo de Diseño Anticipado es el mismo que su homólogo de Post-Arquitectura.

- (PDIF) Dificultad de la Plataforma. Este driver de coste del Diseño Anticipado combina los 3 drivers de coste de Post-Arquitectura siguientes: Tiempo de Ejecución (TIME), Restricciones de Almacenamiento (STOR) y Volatilidad de la Plataforma (PVOL).

- (PREX) Experiencia Personal. Este driver de coste de Diseño Anticipado combina los 3 drivers de coste de Post-Arquitectura siguientes: Experiencia (AEXP), Experiencia en la Plataforma (PEXP) y Experiencia en el Lenguaje y Herramientas (LTEX). Este Multiplicador de esfuerzo lo se tratará mas adelante como un punto de contacto.

- (FCIL) Facilidades Este driver de coste de Diseño Anticipado combina los 2 drivers de coste de Post-Arquitectura siguientes: Uso de Herramienta Software (TOOL) y Desarrollo MultiLugar (SITE).

- (SCED) Planificación Temporal El driver de coste de Diseño Anticipado es el mismo que su homólogo de Post-Arquitectura.

2.3.2. Reutilización de código

En la Ingeniería de Software, el reuso del software sigue siendo uno de los principales temas de investigación, como también una de las principales técnicas para incrementar la calidad de los desarrolladores de Software. En estos últimos tiempos, se presentan muchos anuncios acerca de “la solución definitiva” a todos los problemas de la Ingeniería de Software. Se plantea que la reutilización es la

única vía para desarrollar Sistemas de Información que tengan los niveles de calidad exigidos, dentro de las restricciones existentes de tiempo y presupuesto.

En un principio el concepto de reutilización se concibe como la combinación de componentes de código almacenados en una biblioteca, y usados cuando estos fueron necesarios. En la actualidad este concepto ha cambiado sustancialmente, provocando la necesidad de clasificar los trabajos relacionados con la reutilización según el objeto y el método de reutilización usado. Si se toma por reutilización como un criterio de clasificación simple que combina la fase de desarrollo y/o el nivel de abstracción en el que el conocimiento se produce o se reutiliza, se tendrá tres niveles:

- Reutilización de código
- Reutilización de diseños
- Reutilización de especificaciones

Todas estas conducen a un solo nivel de abstracción o a una sola etapa, que es el ciclo de vida del software. La reutilización del código, es la más común y extendida, la cual se presenta en la fase de implementación, lo que se puede reutilizar en esta fase son: código fuente, código objeto, bibliotecas estándar, etc.

Con respecto a la reutilización de código se tiene que según el resultado de un análisis que se hizo con 3000 módulos en un laboratorio de la NASA sobre como influye este factor en los proyectos de software se pudo observar que el coste de este se obtiene mediante una función no lineal y esto sucede debido a dos factores principales:

- No se comienza desde el principio. Existe un coste alrededor del 5% debido a la valoración, selección y asimilación que debe hacerse del componente reutilizable.
- Pequeñas modificaciones producen desproporcionadas reacciones en el coste. Esto es debido principalmente a dos factores: el coste de comprender el software que va a ser modificado, y el relativo coste asociado a testear el interface.

Estudios realizados determinan que el 47% del esfuerzo de mantenimiento del software está directamente relacionado con la comprensión del software que se modifica. También se demuestra que si se modifican k de m módulos, el número N que indica los interfaces de módulos que son chequeadas es N = k * (m-k) + k *x (k-1) / 2. Relación que se muestra en la siguiente figura, donde la curva demuestra la mencionada relación no lineal.

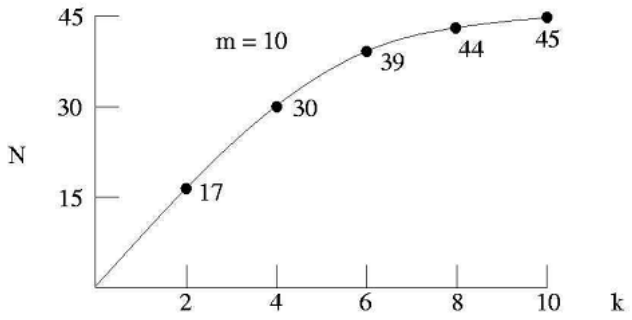


Figura 1: Efectos no lineales del reuso

En los modelos de Diseño Anticipado y Post-Arquitectura se pueden incluir consideraciones especiales cuando se prevea reutilización del código que compondrá la aplicación que se está estimando. La inclusión de características de reutilización conlleva describir el parámetro Size como sigue:

$$Size = KSLOC + KASLOC \times \left[\frac{100 - AT}{100} \right] \times (AAM)$$

Dónde la variable KSLOC ha sido explicada anteriormente, y representa el número de líneas de código a desarrollar desde cero; la variable KASLOC representa miles de líneas de código fuente adaptadas; el valor AT representa el porcentaje de traducción automatizada y por último la variable AAM representa un Multiplicador de Ajuste para la Adaptación:

$$AAM[ESLOC] = \frac{ASLOC \left[AA + AAF\,(1 + 0.02.(SU)(UNFM)) \right]}{100} \quad \textbf{AAF<=0.5}$$

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

$$AAF = 0.4(DM) + 0.3(CM) + 0.3(IM)$$

El tratamiento que hace COCOMO II del software reutilizado utiliza un modelo de estimación no lineal. Esto implica que hay que estimar la cantidad de software que se va a adaptar, ASLOC y tres parámetros de grado de modificación: El porcentaje de diseño modificado (DM), el porcentaje de código modificado (CM) y el porcentaje de esfuerzo inicial de integración requerido para la integración del software reutilizado (IM).

El cálculo de ESLOC se basa en una cantidad intermedia, el Factor de Ajuste de Adaptación (AAF). Las cantidades de adaptación DM, CM, IM se usan para calcular AAF, donde:

DM: Porcentaje de diseño modificado. El porcentaje de diseño de software que es modificado para adaptarlo a los nuevos objetivos y al entorno. (Esto es necesariamente una cantidad subjetiva).

CM: Porcentaje de código modificado. El porcentaje de código software adaptado que es modificado para adaptarlo a los nuevos objetivos y al entorno.

IM: Porcentaje de integración requerida para software modificado. El porcentaje de esfuerzo requerido para integrar el software adaptado dentro de la totalidad del producto y comprobar el producto resultante comparado con la cantidad de esfuerzo normal de integración y pruebas para software de un tamaño similar.

El incremento que supone la comprensión del software (SU) se obtiene de la siguiente tabla; y expresado cuantitativamente como un porcentaje.

	Muy Bajo	Bajo	Nominal	Alto	Muy Alto
Estructura	Muy baja cohesión, alto acoplamiento, definición poco clara del código.	Moderada baja cohesión, alto acoplamiento.	Razonablemente bien estructurado; algunas áreas resultan débiles.	Alta cohesión, bajo acoplamiento.	Enormemente modular, Información ocultada mediante estrucuras de datos y control.
Claridad de la Aplicación	No existe una clara identidad entre las visiones del programa y de la aplicación.	Existe alguna correlación entre el programa y la aplicación.	Moderada correlación entre el programa y la aplicación.	Buena correlación entre ambos.	Existe una clara relación entre las visiones globales de ambos (programa y aplicación).
Descripción implícita	Código obtuso; la documentación o no existe o resulta oscura u obsoleta.	Existen algunos comentarios en el código y alguna documentación útil.	Un moderado nivel de comentarios en el código, y documentación.	Bien comentado el código, documentación útil, aunque débilmente en algunas áreas.	Código autodescriptivo, documentación actualizada, bien organizada y con un diseño racional.
Incremento de SU en ESLOC	50	40	30	20	10

Tabla 9: Escala según se Incrementa la Compresión del Software (SU)

El otro incremento no lineal sobre la reusabilidad tiene que ver con el grado de Valoración y Asimilación (AA) necesarias para determinar si un cierto módulo software completamente reutilizable es apropiado para la aplicación, e integrar su descripción dentro de la descripción global del producto. La Tabla 10 siguiente proporciona una escala para este porcentaje:

Incremento de AA	Nivel de esfuerzo AA
0	Ninguno.
2	Una básica búsqueda modular y de documentación.
4	Alguna Evaluación y Chequeo modular, documentación.
6	Una considerable Evaluación y Chequeo modular, documentación.
8	Una gran Evaluación y Chequeo modular, documentación.

Tabla 10: Escala según el incremento de Valoración y Asimilación (AA)

La cantidad de esfuerzo necesario para modificar el software existente es una función no sólo de la cantidad de modificación (AAF) y la comprensibilidad del software existente (SU) sino también de lo relativamente desconocido que es el software para el programador (UNFM). El parámetro UNFM se aplica multiplicativamente al incremento de esfuerzo en comprensión del software. Si el programador trabaja con el Software todos lo días, el multiplicador 0.0 no añadirá incremento de comprensión del software. Si el programador no ha visto nunca antes el software, el multiplicador 1.0 añadirá el mayor incremento de esfuerzo de comprensión del software. Los valores de UNFM están en la tabla siguiente:

Incremento UNFM	Nivel de Desconocimiento
0.0	Completamente Conocido.
0.2	Bastante Conocido
0.4	Algo Conocido
0.6	Considerablemente Desconocido
0.8	Bastante Desconocido
1.0	Completamente Desconocido

Tabla 11: Escala de valoración para el Desconocimiento del Programador

En PROBE se plantea con respecto a la reutilización que si usted puede encontrar partes disponibles que proporcionen las 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. Para cualquier parte de la biblioteca de reutilización, tome nota de sus nombres y tamaños bajo Partes Reutilizadas.

Este método considera dos tipos de partes reutilizadas. En primer lugar están las partes tomadas de una biblioteca de reutilización. En segundo, las nuevas piezas reutilizables, son algunas de las partes añadidas que ya han sido estimadas. Se identifican estas partes como nuevas partes reutilizables si planea utilizarlas en el desarrollo de otros programas. Es una buena práctica construir una biblioteca de partes útil cuando se trabaja en un nuevo entorno de aplicación.

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.

2.3.3. PMAT(Proceso de madurez)

El PMAT es uno de los factores que se utilizan para calcular el factor de escala B. El procedimiento para determinar el mismo se basa en el Modelo de CMM propuesto por el Software Engineering Institute. Existen dos formas de calcularlo:

La primera captura el nivel de madurez de la organización, resultado de la evaluación según CMM y asignándole el valor correspondiente según la siguiente tabla.

Nivel de CMM	PMAT
1 – Mitad inferior	Muy bajo
1 – Mitad superior	Bajo
2	Nominal
3	Alto
4	Muy Alto
5	Extra Alto

Tabla 12: Factor PMAT de acuerdo al nivel de CMM

La segunda está basada en las dieciocho Áreas de Procesos Claves (KPAs) del modelo del SEI (CMM). 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: ver

.

- Casi siempre: cuando los objetivos son consistentemente alcanzables y bien establecidos en procedimientos operativos estándar.
- Frecuentemente: cuando los objetivos son alcanzables con relativa frecuencia, pero en algunas ocasiones son emitidas bajo circunstancias difíciles (entre el 60% y 90% de las veces).
- Sobre la mitad: cuando los objetivos son alcanzables sobre la mitad de las veces (entre el 45% y 60% de las veces).
- Ocasionalmente: cuando los objetivos son alcanzados algunas veces, pero poco frecuentes (entre el 10% y 40% de las veces).
- Raramente (si acaso): cuando los objetivos raramente son alcanzados (menos del 10% de las veces).
- No se aplica: cuando se tiene el conocimiento requerido sobre el proyecto u organización y el KPA, pero se tiene un sentimiento de que los KPAs circunstancialmente no se pueden aplicar.
- Se desconoce: cuando no se tiene certeza de cómo responderán los KPAs.

Después que se asigna un porciento a cada KPAs se aplica la siguiente ecuación:

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

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. Mediante estos niveles se le asignan distintos valores al a PMAT.

Inicial o Nivel 1 CMM. Este es el nivel en donde están todas las empresas que no tienen procesos. Los presupuestos se disparan, no es posible entregar el proyecto en fechas, te tienes que quedar durante noches y fines de semana para terminar un proyecto. No hay control sobre el estado del proyecto, el desarrollo del proyecto es completamente opaco, no sabes lo que pasa en él.

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

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.

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

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

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.

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.

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

- Innovación organizacional
- Análisis y resolución de las causas

PSP trabaja con 12 KAPs de las 18 KAPs de CMM. Una posible integración entre las teorías de Boehm y Humphrey con respecto a este punto de contacto seria aplicar PSP con el objetivo de obtener mejores por cientos y así un mejor valor de PMAT, es decir una mayor productividad.

2.4. PREX

Este driver de coste de Diseño Anticipado combina los 3 drivers de coste de Post-Arquitectura siguientes: Experiencia en Aplicaciones (AEXP), Experiencia en la Plataforma (PEXP) y Experiencia en el Lenguaje y Herramientas (LTEX). La tabla siguiente se le 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 Año	4 Año	6 Año

Tabla 13: Niveles de medida PREX

AEXP (Experiencia en las Aplicaciones) es un driver que depende del nivel de experiencia en aplicaciones del equipo de proyecto al desarrollar sistemas ó subsistemas software. Los valores se definen en términos de nivel de experiencia del equipo de proyecto en este tipo de aplicaciones. Un valor muy bajo para experiencia en aplicaciones es menor que 2 meses. Un valor muy alto es por experiencia de 6 años ó más. En la siguiente tabla se verá estos rangos de valores según la experiencia del equipo:

Muy Bajo	Bajo	Nominal	Alto	Muy Alto	Extra Alto
AEXP ≤ 2 meses	6 meses	1 año	3 año	6 año	

Tabla 14: Niveles de medida AEXP

PEXP (Experiencia en la Plataforma) este driver mide el nivel de experiencia del equipo en la o las plataformas que se desarrolla el proyecto. El modelo post-Arquitectura amplía la influencia en productividad de PEXP, reconociendo la importancia de entender el uso de plataformas más poderosas, incluyendo más interfaces gráficos de usuario, redes y capacidades de middleware distribuido. En la

siguiente tabla se muestra el rango de valores que puede tomar este driver según la experiencia del equipo.

Muy Bajo	Bajo	Nominal	Alto	Muy Alto	Extra Alto
PEXP 2 meses	6 meses	1 año	3 año	6 año	

Tabla 15: Niveles de medida PEXP

LTEX (Experiencia en la Herramienta y en el Lenguaje). Este es un driver que mide el nivel de experiencia en el lenguaje de programación y en la herramienta software del equipo de proyecto que desarrolla el sistema ó subsistema software. El desarrollo de software incluye el uso de herramientas que realizan representación de requisitos y diseño, análisis, gestión de la configuración, origen de los documentos, gestión de librería, estilo de programa y estructura, verificación de consistencia, etc...Además de la experiencia programando en un lenguaje específico, las herramientas que dan soporte también influyen en el tiempo de desarrollo. La siguiente tabla muestra cómo se le asignan los valores a este driver según la experiencia del equipo.

Muy Bajo	Bajo	Alto	Muy Alto	Extra Alto
LTEX 2 meses	6 meses	1 año	3 año	6 año

Tabla 16: Niveles de medida LTEX

En este punto de integración se pretende aplicando PSP lograr una mejor asignación de los ratios a estos factores de costo. Esto se apoya en los registro de los programadores los cuales pueden brindar una mejor evaluación de la experiencia y habilidad de estos.

2.5. Conclusiones

En el capítulo que concluye se explicó de forma detallada el método por Puntos de Casos de Usos y se hizo énfasis en todos los problemas que presenta, para de esta forma poder entender mejor la Situación Problémica de la Universidad. También en esta parte se tiene como tema central los puntos de integración. Con respecto a estos se realiza un análisis de cómo los utiliza cada teoría y de que forma se podrían utilizar para obtener un mejor resultado a la hora de realizar estimaciones más confiables y precisas. En el próximo capítulo se validará esta integración a través de juicios de expertos.

Capítulo 3: Validación de los puntos de integración entre las teorías de Boehm y Humphrey en aplicaciones de Realidad Virtual en la UCI

3.1. Introducción

En este capítulo se realizaran varias entrevistas a la mayoría de los proyectos de realidad virtual con el objetivo de obtener una visión acerca de cómo se realizan las estimaciones, que métodos utilizan para calcular el esfuerzo y por qué los utilizan. Después teniendo en cuenta esta situación se realizará una evaluación de la integración de las teorías de Boehm y Humphrey en los puntos de contacto mediante los juicios expertos. También se hablará de los beneficios que aportaría el uso de esta integración en esta Universidad.

3.2. Realidad Virtual

El concepto realidad virtual ha estado en general asociado a experimentos con computadoras de alta performance vinculadas a dispositivos de Entrada y Salida (E/S) no convencionales y más recientemente a la industria de los video-juegos. El ambiente científico-tecnológico no ha utilizado demasiado esta herramienta aún, pero el interés está creciendo sensiblemente en los últimos tiempos. Algunas de las aplicaciones que hoy se observan en la literatura son la exploración de datos tridimensionales, tales como estructuras de moléculas, resultados de simulaciones computacionales o análisis de estructuras cristalinas. Y también por supuesto el tema que interesa en este trabajo, los simuladores orientados al entrenamiento, donde el mejor ejemplo son los simuladores de vuelo a nivel internacional y en la Universidad un simulador de autos.

De hecho puede decirse que los simuladores de vuelo fueron los responsables del nacimiento del concepto realidad virtual y sin duda los principales impulsores de los desarrollos en computación gráfica tridimensional. Fue en la década del ’70 cuando las Fuerzas Armadas Americanas invirtieron enormes sumas de dinero para implementar un sistema que permitiera entrenar a los pilotos de cazabombarderos con el mayor realismo posible. Gracias a estos esfuerzos se optimizaron los procedimientos para renderizar escenarios tridimensionales, dando origen a lo que son hoy los lenguajes de graficación 3D como OpenGL.

Hoy OpenGL es completamente implementado en hardware en placas gráficas accesibles a minicomputadoras, por lo que sin duda las aplicaciones de realidad virtual verán un crecimiento notable en el futuro cercano. Algunos ejemplos interesantes son las aplicaciones en medicina, que van desde el diagnóstico por imágenes tridimensionales, hasta intervenciones quirúrgicas virtuales. En la Universidad se está desarrollando un simulador quirúrgico con el objetivo de entrenar en la especialidad médica de mínimo acceso y se enmarca en la solución de automatizar el sistema de salud cubano.

A finales de los 80's, los gráficos por computadora entraron en una nueva época. No era sólo que las soluciones tridimensionales (3D) comenzaran a reemplazar los enfoques bidimensionales y de dibujo de líneas (2D), sino que también existía la necesidad de un espacio de trabajo totalmente interactivo generado a través de la tecnología. A partir de principios de los años 90's, estas soluciones se han visto enriquecidas con sensaciones del mundo real a través de estímulos visuales, auditivos y de otro tipo que afectan al usuario de manera interactiva. Esto es en esencia lo que se llama Realidad Virtual.

Tecnológicamente hablando, la Realidad Virtual ha sido definida de varias maneras específicas, por ejemplo, como una combinación de la potencia de una computadora sofisticada de alta velocidad, con imágenes, sonidos y otros efectos. Otras definiciones son:

1. Un entorno en tres dimensiones sintetizado por computadora en el que varios participantes acoplados de forma adecuada pueden atraer y manipular elementos físicos simulados en el entorno y, de alguna manera, relacionarse con las presentaciones de otras personas pasadas, presentes o ficticias o con criaturas inventadas.

2. Un sistema interactivo computarizado tan rápido e intuitivo que la computadora desaparece de la mente del usuario, dejando como real el entorno generado por la computadora, por lo que puede ser un mundo de animación en el que se puede adentrar.

Aunque todas estas definiciones son válidas la definición más sencilla y la mas general es: "La Realidad Virtual es aquella forma de trabajo donde el hombre puede interactuar totalmente con la computadora, generando ésta espacios virtuales (ambientes virtuales) donde el humano puede desempeñar sus labores y donde el humano se comunica con la computadora a través de efectores o dispositivos de interacción".

La Realidad Virtual explota todas las técnicas de reproducción de imágenes y las extiende, usándolas dentro del entorno en el que el usuario puede examinar, manipular e interactuar con los objetos expuestos. Un mundo virtual es un modelo matemático que describe un "espacio tridimensional", dentro de este "espacio" están contenidos objetos que pueden representar cualquier cosa, desde una simple entidad geométrica, por ejemplo un cubo o una esfera, hasta una forma compleja, como puede ser un desarrollo arquitectónico, un nuevo estado físico de la materia ó el modelo de una estructura genética. Se trata, en definitiva, de un paso mas allá de lo que sería la simulación por computador, tratándose realmente de la simulación interactiva, dinámica y en tiempo real de un sistema.

3.3. Características y requisitos de los especialistas para la validación de estos puntos de integración

Las características y los requisitos que se tuvieron en cuenta para seleccionar los especialistas a los cuales se entrevistó con el objetivo de validar la forma en que se propone la integración en cada uno de estos puntos de integración entre las teorías de Boehm y Humphrey fueron:

Como aspecto principal debían tener como mínimo 3 años de experiencia vinculados continuamente a las estimaciones de proyectos y la planificación.

Otro requisito que debían cumplir fue tener conocimientos acerca de PSP y COCOMO II.

Haber realizado investigaciones vinculadas a temas que se encuentran dentro de la Gestión de proyectos como son la estimación, la planificación y la calidad de software.

3.4. Primeros resultados de las entrevistas

Con esta entrevista se pretende conocer la situación de los proyectos de Realidad Virtual en cuanto a los métodos de estimación que utilizan. Otro objetivo que se persigue es conocer cuál es el nivel de conocimiento y cuál es la opinión de las personas encargadas de realizar las estimaciones en estos proyectos con respecto a ciertos factores que se han tomado en cuenta para realizar esta integración. Ver Anexo 2

Los Resultados de las primeras entrevistas son:

- Con respecto a la primera pregunta de la entrevista el 100% de los proyectos entrevistados han realizado estimaciones. También la han tenido en cuenta a la hora de realizar la planificación y opinan que es necesaria para realizar una planificación que se ajuste a la realidad del tiempo que puede demorar realizar alguna tarea por un programador u otro integrante del proyecto. El no ajustarse a la realidad puede traer como consecuencia que se pida, por ejemplo, a un programador que realice un determinado componente en un tiempo de 15 días cuando en realidad necesita 20 o más y traer como consecuencia mala calidad del producto.

- Con el resultado de la segunda pregunta se obtiene que el 90 % de los proyectos entrevistados trabaja con métodos empíricos, se basan en su experiencia para realizar las estimaciones. Algunos proyectos se encuentran investigando otros métodos de estimación para su uso en el cálculo las estimaciones y en un 10% trabajan con COCOMO II.

- Las estimaciones en un 90% de los casos las estimaciones las realizan los líderes de proyectos. En un 10% lo realiza un equipo de trabajo que se dedica específicamente a la estimación y planificación del proyecto. El 100% cree que la estimación es un elemento clave para realizar la planificación y que esta se debe aplicar al menos dos veces en el tiempo de desarrollo del proyecto, una en la fase de inicio y después que se halla definido la arquitectura. Los entrevistados plantean que la realización de estimaciones a medida que avanza el proyecto les sirve para medir el estado en que se encuentra el proyecto y de esta forma poder tomar las medidas necesarias y las precauciones para que el resultado final sea un producto que satisfaga las necesidades del el cliente y al mismo tiempo que se le asignen los recursos necesarios es decir ni más ni menos para lograr un software de calidad.

- Un 60% de los entrevistados tenía como mínimo 2 años trabajando en ese tipo de proyectos y sólo unos pocos se considera expertos. Otros plantean que es casi imposible decir que alguien es un experto en este tipo de aplicaciones debido a que las aplicaciones de Realidad Virtual es algo muy extenso y abarca muchas líneas de trabajo por lo que opinan que no es posible alcanzar la categoría de experto en este campo en general. En una línea de trabajo especifica como puede ser la creación de simuladores, un 70% plantea que en 4 o 5 años se puede llegar ser un experto trabajando constantemente e investigando en profundidad y un 30% cree que en 3 o 4 años.

- Con respecto a la valoración de la experiencia el 100% respondió que es un factor clave a la hora de hacer una estimación lo más exacta posible. El líder del proyecto necesita de experiencia para poder guiar de la forma más adecuada a los integrantes del equipo y de esta forma poder lograr una mayor motivación en estos. El encargado de realizar las estimaciones y la panificación que en la mayoría de los casos en los proyectos de este tipo en la Universidad la realiza el líder y la experiencia influye notablemente en los resultados que obtiene.

Estas entrevistas se les aplicaron a los encargados de realizar las estimaciones y la planificación en los distintos proyectos de Realidad Virtual en la Universidad con el objetivo de conocer la situación de los mismos. También con estas entrevistas se puede apreciar y demostrar que la utilización de esta integración entre las teorías de Boehm y Humphrey que se propone tendría como resultados grandes beneficios en cuanto a la calidad y eficiencia de las aplicaciones de Realidad Virtual.

3.5. Líneas de código

COCOMO 2 es un modelo de estimación que no cuenta con una forma propia de calcular el tamaño del software. Este utiliza los puntos de función sin ajustar y una vez que los se obtienen se multiplican por un factor de conversión que depende del lenguaje de programación en el cual se va a escribir el código de la aplicación. Los puntos de función de un sistema software se calculan teniendo en cuenta:

- Entradas al sistema
- Salidas del sistema
- Consultas
- Grupos de datos lógicos del sistema
- Grupos de datos lógicos que no son del sistema pero que el sistema usa.

Debe computarse cuántas ocurrencias de cada parámetro contiene un sistema, calificándolos según su complejidad en alta, media y baja. Cada parámetro, para una complejidad dada tiene un determinado peso, ese peso son los puntos de función asignados a ese parámetro. Luego de este proceso los puntos de función se ajustan de acuerdo a las características generales del sistema.

A los efectos de una estimación temprana los puntos de función tienen un inconveniente. En efecto, los puntos de función implican que uno conoce los grupos de datos que utilizará el sistema y además asume que ese conocimiento es lo suficientemente amplio como para calificar esos grupos en complejidades baja, media y alta.

En general en el período de preventa no se dispone de esos datos, especialmente de los datos internos al sistema. Ese es el primer problema, casi insuperable. Por otra parte, si se dispusiera de esa información es complicado en tiempo y costo hacer el análisis necesario para calcular los puntos de función.

El método de puntos de función de Albretch afirma que es independiente de la tecnología pero existen incógnitas en cuanto a esto.

Hasta aquí todo parece bien, pero surge una pregunta, ¿por qué la función de entrada de datos es equivalente funcionalmente a la de consulta? ¿Qué criterio se utilizó para establecer esa equivalencia entre las funciones y todas las demás que conforman el método?

Albretch estudió 24 proyectos de aplicaciones de negocios con un rango de tamaño desde 3000 a 318.000 líneas de código desarrolladas en DMS, PL/1 y COBOL. Se puede asumir, dado que no hay una explicación, que en esos proyectos Albretch encontró una relación entre entrada y consulta que le permite decir que la cantidad de líneas de código utilizadas para una entrada de complejidad media es igual a la cantidad de líneas de código de una consulta media. Sin embargo se tiene que reconocer que esa igualdad de líneas de código se da para los lenguajes analizados y no necesariamente para cualquiera.

En efecto, si un lenguaje tiene facilidades para programar una consulta entonces esa relación ya no existe. Se ve entonces que el tamaño, “independiente de la tecnología” del método de puntos de función ya no es tan independiente de la tecnología, y por lo tanto no sería tamaño según la definición clásica, sino esfuerzo normalizado para una o más tecnologías.

En este punto de contacto después de apreciar lo planteado anteriormente se puede llegar a la conclusión de que la forma de calcular el tamaño por puntos de función desajustados no es la más confiable ni la más eficiente.

La única forma de definir puramente y estrictamente tamaño, independientemente de cualquier tecnología, es enumerar la cantidad de funciones de un mismo tipo que tiene un sistema, por ejemplo decir que el tamaño es 20 altas simples, 30 modificaciones complicadas, 12 listados simples , etc. , pero al tratar de establecer una equivalencia entre un tipo función y otra, necesariamente se acude a nociones relacionadas con el esfuerzo de construcción de las mismas (codificación, diseño, análisis, etc.), ya que las únicas unidades de medida que tiene en común un alta con una modificación o un listado son las horas que se tarda en construirlas o también el tamaño físico (sean bytes o líneas de código o páginas de documentación) que tienen en una determinada tecnología o método.

El método PROBE cumple con estas características ya que este después de definir el diseño conceptual lo divide en partes. Después le asigna a cada parte un número de métodos y un tamaño relativo a cada método. Este 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. Este método le guía en el uso de información histórica para hacer estimaciones y utiliza una vía de sonido para ajustar los datos.

Con respecto a este punto de integración en los proyectos de Realidad Virtual las entrevistas arrojaron los siguientes resultados:

- Con respecto a la definición de línea de código se tiene que un 70% de los entrevistados opina que la definición de esta es líneas de código más definiciones de datos y el otro 30% opina que son líneas de código, definiciones de datos y comentarios.

- También se tiene que como variante de líneas de código que se deben tener en cuenta a nivel de proyectos para calcular el esfuerzo según los entrevistados se tiene que un 80% considera que son líneas nuevas, modificadas y reutilizadas. el otro 20% consideró que eran las nuevas y modificadas solamente.

- Después de haberles explicado a los expertos acerca de las ventajas que tiene el método PROBE y la forma en que calcula el tamaño se les preguntó si esa propuesta podría traer buenos resultados en cuanto a la exactitud y precisión de las estimaciones a lo que un 90% respondió que si se hacia un buen uso del PSP y PROBE entonces si se podría obtener buenos resultados y el 10% respondió que no sabia. Los que respondieron afirmativamente plantean en su mayoría que al

apoyarse este método en datos históricos para realizar las estimaciones se convierte en un método bastante confiable.

3.6. Reutilización

En este punto de integración después de haber hecho un estudio profundo de cómo se utiliza la reutilización en las dos teorías se puede llegar a la conclusión de que el Modelo de Rehúso planteado en el modelo de COCOMO II es el más ideal y abarcador cuando se trata de mantenimiento de software. ¿Por qué se plantea esto?

En [Parikh and Zvegintzov 1983] se indica que el 47% del esfuerzo en el mantenimiento de software está relacionado con la tarea que implica entender el software que va a ser modificado. Además, en [Gerlich and Denskat 1994] se muestra que existe un efecto no lineal en el costo del rehúso debido al chequeo de interfaces que debe realizarse durante el desarrollo del software modificado. Estos inconvenientes pueden reducirse si el software está apropiadamente estructurado.

La variable que mide este factor tan importante como es el esfuerzo que se realiza comprendiendo el software que se va a reutilizar o dar mantenimiento es SU (Comprensión del Software). Ver Tabla 9.

Mientras más bajo es el nivel de esta variable mayor es el esfuerzo que hay que realizar para darle mantenimiento al software. Aplicando PSP en los proyectos de Realidad Virtual se puede obtener un valor muy alto en algunas características de este parámetro y de este modo disminuir esfuerzo de un modo considerable. También se tiene en este punto de integración que mediante PROBE y PSP se podría obtener un mejor porciento de código a modificar (CM). Por estas razones se plantea que mediante esta integración se obtendría resultados más precisos y se emplearía menos esfuerzo en el mantenimiento de software.

Hay que mencionar con respecto a este punto de integración que el modelo de COCOMO II utiliza el Modelo de reutilización cuando la cantidad de código fuente base añadido ó modificado es menor ó igual que el 20% del nuevo código que se está desarrollando. En caso de que esto no se cumpla entonces se calcula el tamaño por puntos de función desajustados donde no se tiene en cuenta ni el código reutilizado ni el código modificado entre otros parámetros, por lo que es mejor utilizar en estos casos el método PROBE.

Los resultados de las entrevistas realizadas con respecto a este punto de integración son:

- Al 83% de los experto cuando se les preguntó si utilizando PSP se puede lograr que el programador escriba el código de una forma más descriptiva respondieron que si. Debido a que el PSP promueve las buenas prácticas de programación y le exige al ingeniero utilizar un estándar de codificación de forma tal que hace su código más comprensible para otras personas.

- Al 100% de los experto cuando se les preguntó que si aplicando el proceso personal del software se podría obtener un código más documentado respondió afirmativamente, argumentando que esta práctica obliga a cada ingeniero a obtener datos bastantes detallados acerca de su trabajo como son los posibles errores que este comete más a menudo.

- Se les preguntó que cual era el porciento mínimo de reutilización de código en cada módulo que realizaban y la mayoría respondió aproximadamente entre un 30% y un 40%.

- Con respecto a este punto de integración teniendo en cuenta las características en que los dos métodos trabajan con la reutilización de código la opinión del 70% de los entrevistados opinan que en los casos que no se aplican el modelo de rehúso de COCOMO II el método PROBE es más abarcador ya que tiene en cuenta código modificado, código reutilizado y código adicionado.

- Después de haberle explicado a los entrevistados acerca de las dos vías en que COCOMO II calcula el porcentaje del código modificado debido a la confiabilidad del método PROBE teniendo en cuenta sus características a la hora de estimar el número de líneas de código ya vistas en el epígrafe anterior se obtuvo como resultado de sus evaluaciones que un 66% opinó que el método PROBE es el más indicado para obtener este porcentaje de una forma más exacta. El otro 34% consideró que no era el más indicado ya que este método tomaba demasiado tiempo y esfuerzo para aplicarlo.

3.7. Proceso de Madurez (PMAT)

En este punto de integración para su validación se aplicó el método de expertos y dentro de este el método de agrupados individuales. Se consultaron a varios expertos en estos temas, como son PSP y CMM. Como tarea principal a validar se plantea la interrogante que si al aplicar PSP en los proyectos de

Realidad Virtual se podría mejorar el porciento de los 12 aéreas de procesos de este que están entre los 18 del CMM y por lo cual obtener un mejor valor de PMAT. El resultado de esto ayudaría marcadamente en la obtención de un valor de B menor que 1 lo cual indicaría que el proyecto es productivo.

Para comenzar a investigar este punto de integración se realizaron entrevistas en los proyectos de Realidad Virtual para conocer si aplicaban el PSP, hasta que punto lo aplicaban y cuál era el nivel de conocimiento y la importancia que le daban al mismo. Los resultados de las mismas son:

- Con respecto a PSP todos conocen sus características pero un 90% de estos nunca lo ha aplicado y el otro 10% lo ha aplicado muy poco. Un 100% de los entrevistados plantean que el PSP es importante para obtener software de calidad ya que este ayuda al ingeniero de software ha disminuir sus errores previendo los mismos antes de terminar el trabajo. También plantean que el buen uso de este puede disminuir el costo y el esfuerzo que se emplea en la creación software.

- Con respecto a los factores de escala 80% cree que tiene gran influencia en el costo y la duración del desarrollo del software y que no le agregarían más ninguno al modelo COCOMO II ya que lo ven bastante abarcador en esta parte.

- Después de explicar a cada experto como se realizaría esta integración en este punto se obtuvo que el 100% de estos considera que la posibilidad de éxito en el cumplimiento del objetivo de este punto de integración es total ya que el PSP ayuda a entender cómo se debe de hacer para alcanzar los niveles del CMM. También plantean que la aplicación del PSP es la manera más rápida que existe para alcanzar los niveles del CMM.

3.8. Experiencia Personal (PREX)

Con este multiplicador de esfuerzo en el modelo COCOMO II se pretende medir la habilidad por medio de la experiencia del equipo de trabajo en cuanto al tipo de aplicación, de plataforma y herramienta y lenguaje de programación. Los drivers que miden esto son AEXP, LTEX y PEXP. A estos se les asignan un ratio desde muy bajo que serian 2 meses hasta muy alto que serian 6 años o más. Con este punto de integración se pretende que aplicando PSP se pueda asignar un valor más real a estos driver de coste apoyándose en los datos históricos del Proceso Personal Software. Se puede observar en la Universidad que existen programadores que a pesar de llevar menos tiempo que otros trabajando en un tipo de

lenguaje o de aplicación tienen mayor habilidad por lo que a la hora de medir la habilidad de estos el PSP sería la forma de probar esta habilidad y capacidad mediante sus métricas de calidad.

Después de explicarle a los expertos acerca de este punto de integración respondió afirmativamente un 83% de estos justificando que apoyarse en los registros que hacen los programadores se les puede asignar un mejor ratio de experiencia.

3.9. Conclusiones

En este capítulo que termina se tratan las definiciones de Realidad Virtual. Las características de los proyectos de Realidad Virtual en la Universidad con respecto a las estimaciones y la planificación. Se realizaron entrevistas a los responsables en estos proyectos de realizar las estimaciones y la planificación para conocer la situación de los mismos. También se han validado los puntos de integración en este tipo de proyecto mediante los juicios de expertos.

Conclusiones

En el trabajo que concluye se han realizado varias tareas investigativas para darle cumplimiento a tres objetivos específicos muy importantes para llegar a darle cumplimiento a este trabajo.

- Se tiene que en el primer objetivo específico que es estudiar las métricas en la estimación se realizó el análisis de varias de estas. Primeramente las líneas de código porque tienen el mayor peso de esta integración. También se tratan los puntos de función puesto que el modelo COCOMO II utiliza los puntos de función desajustados para calcular tamaño y otro fue los puntos de casos de usos debido a que es el más usado en la Universidad.

- Otro objetivo específico fue estudiar los modelos de estimación, para esto se analizaron las teorías de Boehm y de Humphrey y dentro de estas se hizo énfasis en el modelo COCOMO II y PROBE. Se analizaron de forma profunda los puntos de contactos, planteando las posibles formas en que se podrían utilizar estos para la integración de estas teorías y justificando las ventajas que tendría esta integración en los proyectos de Realidad Virtual.

- Para el cumplimiento del ultimo objetivo y más importante de este trabajo que fue la validación de la integración de las teorías de Boehm y Humphrey se aplicó el juicio de expertos y dentro de este específicamente el método de agrupados individuales donde se dividieron los puntos de integración en una o más tareas para determinar su posibilidad de éxito o fracaso por los expertos.

Después de haber realizado todas estas actividades se pudo demostrar que los resultados de esta integración serian de gran beneficio para llevar al máximo nivel la exactitud de las estimaciones y por lo tanto la planificación en los proyectos de Realidad Virtual con lo que se obtendrían aplicaciones de mayor calidad, terminadas en tiempo, más flexibles a los cambios y con un mayor control sobre su desarrollo.

Recomendaciones

- Aplicar una herramienta para comenzar a registrar los datos históricos necesarios en aras de utilizar PSP en los proyectos de Realidad Virtual.
- Emplear el método puntos de función desajustados y después ajustar este tamaño mediante los parámetros de regresión lineal para calcular el tamaño a partir de datos históricos y así comenzar a comparar estos resultados con los obtenidos en la recomendación anterior.
- Crear en cada proyecto una BD para tener un registro de los resultados de las estimaciones, así como su comportamiento e influencia en la planificación y los resultados reales del proyecto.

Referencias bibliográficas

Adriana Gómez, María del C. López, Silvina Migani, Alejandra Otazú. 1994. www.alarcos.inf-cr.uclm.es. [En línea] 1994. http://alarcos.inf-cr.uclm.es/doc/pgsi/doc/teo/8/cocomo2-apuntes.pdf.

anónimo. 2005. *Apuntes de la asignatura de empresa y gestion de proyectos.* 2005.

—. 2005. Estimacion de proyectos de software. [En línea] 2005. [Citado el: 16 de diciembre de 2007.] http://w3.sel.inf.uc3m.es.

—. 2003. Planificacion. *Gestion de Proyectos.* [En línea] 2003. [Citado el: 10 de diciembre de 2007.] http://www.getec.etsit.upm.es/docencia/gproyectos/planificacion/planificacion.htm.

Cao, Jose Ignacio. 2004. Principios para un metodo de estimacion de proyectos de software basado en los escenario principales. [En línea] 2004. [Citado el: 15 de enero de 2008.] http://www.itba.edu.ar/capis/epg-tesis-y-tf/cao-trabajofinaldeespecialidad.pdf.

Concepción, Pedro. 2005. Planificación de Proyectos de Software. [En línea] 2005. [Citado el: 10 de diciembre de 2007.] http://www.getec.etsit.upm.es/articulos/gproyectos/art4.htm.

Giraldo, Otoniel Perez. 2006. Métricas, Estimación y Planificación en Proyectos. [En línea] 2006. [Citado el: 15 de enero de 2008.] http://www.willydev.net/descargas/willydev_planeasoftware.pdf.

Gómez, Adriana, y otros. 2003. *COCOMO -UN MODELO DE ESTIMACION DE PROYECTOS DE SOFTWARE.* 2003.

Ochoa, Francisco Romero. 2006. Estimacion de costos. [En línea] 2006. [Citado el: 10 de diciembre de 2007.] http://boards4.melodysoft.com/app?ID=2005AISI0405&msg=51&DOC=21.

Palacio, Juan. 2004. Origen de gestion de proyectos. [En línea] 2004. [Citado el: 15 de diciembre de 2007.] http://www.navegapolis.net/files/s/NST-001_01.pdf.

S, Ana Mª Moreno. 2004. ESTIMACIÓN DE PROYECTOS SOFTWARE. [En línea] 2004. [Citado el: 20 de diciembre de 2007.] http://www.colosia.com.mx/docentes/archivos/Tema1gespro.pdf.

Bibliografía

1. . **Rancán, Claudio Jorge.** TRABAJO FINAL ESPECIALIDAD EN CONTROL Y GESTION DE SOFTWARE. [En línea] julio de 2003. [Citado el: 5 de diciembre de 2007.] http://www.itba.edu.ar/capis/epg-tesis-y-tf/rancan-trabajofinaldeespecialidad.pdf.

2. **Giraldo, Otoniel Perez.** Métricas, Estimación y Planificación en Proyectos de Software. [En línea] [Citado el: 6 de diciembre de 2007.] http://www.willydev.net/InsiteCreation/v1.0/descargas/willydev_planeasoftware.pdf .

3. Estimación de Tamaño -Parte 2. [En línea] [Citado el: 10 de diciembre de 2007.] http://hornet.ls.fi.upm.es/DSP/Lecciones/Lec04-20071120.pdf.

4. **Wesley, Addison.** Proceso Software Persona. [En línea] 2001. [Citado el: 10 de diciembre de 2007.] http://lsi.ugr.es/~ig1/docis/respsp.pdfl.

5. Gramajo, E., García-Martínez, R., Rossi, B., Claverie, E. y Britos, P. [En línea] [Citado el: 15 de diciembre de 2007.] http://www.itba.edu.ar/capis/webcapis/RGMITBA/articulosrgm/R-ITBA-22-estimacion.pdf.

6. **Peralta, Mario.** ESTIMACIÓN DEL ESFUERZO BASADA EN CASOS DE USO. [En línea] [Citado el: 16 de diciembre de 2007.] http://www.itba.edu.ar/capis/rtis/rtis-6-1/estimacion-del-esfuerzo-basada-en-casos-de-usos.pdf.

7. Métodos de Estimación de Tamaño Funcional Software Aplicación a Enfoques de desarrollo Hichem Labdelaoui. [En línea] [Citado el: 15 de enero de 2008.] http://is.ls.fi.upm.es/doctorado/Trabajos20022003/Labdelaoui2.pdf.

8. **Hernández-Pozas, Olivia.** Esfuerzo Individual al Trabajar en Grupos. [En línea] [Citado el: 15 de enero de 2008.] http://www.mty.itesm.mx/rectoria/dda/rieee/pdf-05/23(EGADE).OliviaHdz..pdf.

9. **Varas, Marcela.** Una Experiencia con la Estimación del Tamaño del Software. [En línea] 28 de septiembre de 2002. [Citado el: 20 de enero de 2008.] http://www.inf.udec.cl/revista/ediciones/edicion1/mvaras.PDF.

10. **Navarro, Antonio.** Plaificación de proyectos de software. [En línea] [Citado el: 21 de enero de 2008.] http://www.fdi.ucm.es/profesor/anavarro/5._Planificacion_de_proyectos_de_software.pdf.

11. Administracion y Gestion de Proyectos. [En línea] [Citado el: 21 de enero de 2008.] http://faea.uncoma.edu.ar/materias/gestion/#Overview.

12. MODELOS EMPÍRICOS DE ESTIMACIÓN. [En línea] 15 de septiembre de 2001. http://www.angelfire.com/my/jimena/ingsoft/guia4.htm.

13. **MORENO, ANA Mª.** ESTIMACIÓN DE PROYECTOS SOFTWARE. [En línea] [Citado el: 15 de enero de 2008.] http://trevinca.ei.uvigo.es/~cfajardo/Nueva_carpeta/presentaciones/cocomo2k.pdf.

14. Estimación del Coste. *Proyectos Software.* [En línea] [Citado el: 20 de enero de 2008.] http://www.monografias.com/trabajos27/estimacion-coste/estimacion-coste.shtml.

15. Gestión de Proyectos. [En línea] [Citado el: 22 de enero de 2008.] http://www.getec.etsit.upm.es/docencia/gproyectos/gproyectos.htm.

16. **Mateus Ferreira, Félix García, Francisco Rui y Manuel F.** Ontología y Metamodelo. *Medición del Software.* [En línea] [Citado el: 27 de enero de 2008.] http://www.esi.uclm.es:8080/tsi/informes/UCLM-TSI-001.pdf.

17. **Palacio, Juan.** Origen de la gestión de proyectos. [En línea] 2006. [Citado el: 27 de enero de 2008.] http://www.navegapolis.net/files/s/NST-001_01.pdf.

18. **Mendoza, Luis Eduardo.** SISTEMAS DE INFORMACIÓN II TEORÍA CONTENIDO:PLANIFICACIÓN DE LA IMPLEMENTACIÓN TÉCNICAS DE ESTIMACIÓN. [En línea] [Citado el: 25 de enero de 2008.] http://www.lisi.usb.ve/prof/lmendoza/Documentos.

19. LA MEDICIÓN DEL SOFTWARE MÉTRICAS. [En línea] [Citado el: 2 de febrero de 2008.] http://www.oei.eui.upm.es/Asignaturas/PInformaticos/ficheros/transparencias/TEMA_2.pdf.

20. **Rubio, Sergio Eduardo Durán.** Una métrica estándar para establecer el tamaño del software. *Puntos por Función.* [En línea] junio de 2003. [Citado el: 18 de febrero de 2008.] http://www.inegi.gob.mx/inegi/contenidos/espanol/prensa/Contenidos/Articulos/tecnologia/puntosxfuncion.pdf.

21. **Vandeville, W. Harrison y J.** Coordinating Models and Metrics to Manage Software Projects. [En línea] [Citado el: 11 de febrero de 2008.] http://www.sba.pdx.edu/faculty/davidr/draccess/WEB/publications/papersWordformat/2paper/models_metrics.pdf.

22. Utilidades de los Puntos de Función. [En línea] CalidaddelSoftware.com, 2008. [Citado el: 1 de marzo de 2008.] http://www.calidaddelsoftware.com/modules.php.

23. **Gracia, Por Joaquin.** CMM – CMMI. [En línea] 14 de agosto de 2005. [Citado el: 1 de marzo de 2008.] http://www.ingenierosoftware.com/calidad/cmm-cmmi.php.

24. CAPITULO 2. PROCESO PERSONAL DE SOFTWARE. [En línea] [Citado el: 2 de marzo de 2008.] http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/pelaez_r_jj/capítulo 2.pdf.

25. Modelo de Madurez de la Capacidad del Software. [En línea] 2004. [Citado el: 3 de marzo de 2008.] http://www.cii-murcia.es/informas/ene05/articulos/CMM.pdf.

Glosario de siglas y términos

Calidad: Es el conjunto de características de un producto que satisfacen a 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.

Calidad del software: Concordancia con los requisitos funcionales y de rendimiento explícitamente establecidos con los estándares de desarrollo

CMM: Modelo de Madurez de Capacidad

COCOMO: Modelo de estimación de costos

Estándar: en informática, conjunto de especificaciones técnicas utilizadas para unificar el desarrollo de hardware o de software

Estimación: Aprecio y valor que se da y en que se tasa y considera algo

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 importante de la ingeniería que crea y mantiene las aplicaciones de software.

Interfaz: Punto en el que se establece una conexión entre dos elementos, que les permite trabajar juntos

IPMA: Internacional Project Managenet Association

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

LOC (LDC): Líneas de código.

Métrica: Una forma de medir una escala, definidas para realizar mediciones de uno o varios atributos en un software.

Medición: es el acto de determinar una medida.

Método: Procedimiento que se aplica para llegar a un fin determinado

NCLOC: No Comentary Lines of Code.

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

Producto: cualquier software que será construido a petición de otros

PMAT: Es el factor mediante el cual se calcula la Madurez del Proceso.

PREX: Experiencia Personal.

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

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

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.

PMI: Project Management Institute

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

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.

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.

Anexos

Anexo 1: Nivel de cumplimiento de los objetivos de cada KPA

Áreas de Procesos Claves	Casi Siempre (90%)	A menudo (60-90%)	La mitad de las veces (40-60%)	Ocasión almente (10-40%)	Casi nunca (<10%)	No se aplica	No se conoce
Administración de Requerimientos							
Planificación del Proyecto de Software							
Seguimiento y supervisión del Proyecto de Software							
Administración de Subcontratos							
Aseguramiento de la Calidad							
Administración de la Configuración							
Objetivo del Proceso de Organización							
Definición del Proceso de Organización							

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

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. ¿Qué 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? ¿Por qué?

2- Detallar impactos de PSP (experiencia).

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

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

c. ¿Qué 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. ¿Qué 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 Languaje JCL).
___ Líneas de código y líneas físicas visualizadas en una pantalla.
___ Líneas de código determinadas por delimitadores lógicos.

Las variantes a nivel de proyecto son:

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

b. ¿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 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?

Anexo 3: Encuesta

Líneas de Código

¿Cree usted que utilizando PROBE se puede obtener un valor más exacto del tamaño en los proyectos de Realidad Virtual? ¿Por qué?

Reutilización

¿Cree usted que la reutilización es mejor utilizarla como lo hace COCOMO II o PROBE? ¿Por que?

¿Cree usted que aplicando PSP se puede obtener un código más descriptivo?

¿Cree usted que aplicando PSP se puede obtener un código más documentado?

¿Con los datos históricos de PSP se pueden obtener porcientos más exactos del código a modificar?

PMAT

¿Aplicando PSP se puede obtener un mejor porciento en las 12 áreas de procesos de CMM que coinciden entre ambos?

PREX

¿Se pueden asignar mejores ratios a los multiplicadores de esfuerzo que miden la experiencia en COCOMO II apoyándose en los registros del PSP?