RISI **1**(1), 73-82 (2004)
Rev. investig. sist. inform.
Facultad de Ingeniería de Sistemas e Informática
Universidad Nacional Mayor de San Marcos
ISSN: 1815-0268 (impreso)

ESTUDIO DE TÉCNICAS BASADAS EN PUNTOS DE FUNCIÓN PARA LA ESTIMACIÓN DEL ESFUERZO EN PROYECTOS SOFTWARE

José Pow-Sang*

RESUMEN

La estimación de costos y esfuerzos sigue siendo una de las tareas más difíciles en la gestión de un proyecto de software. En la actualidad existen técnicas que permiten realizar esta labor aunque, lamentablemente, aún no hay técnicas maduras específicas para enfoques de desarrollo como la orientación a objetos o como los sistemas expertos. A este problema se suma la escasa información proporcionada por las técnicas de estimación existentes para su aplicación a ciclos de vida de desarrollo de software diferente al de cascada, como, por ejemplo, los ciclos de vida iterativo-incrementales o en espiral.

El presente artículo muestra un estudio comparativo de tres técnicas que se podrían emplear para la estimación del esfuerzo de cada una de las iteraciones de la construcción de software. Las técnicas se basan en puntos de función y COCOMO II. Una de estas técnicas ha sido probada en proyectos desarrollados por una persona con resultados satisfactorios.

Este estudio utiliza los resultados obtenidos en dos proyectos, uno formado por un equipo de 10 y otro de 11 alumnos de pregrado. Las técnicas se han aplicado al esfuerzo real utilizado por cada uno de los equipos de desarrollo (esfuerzo en horas-hombre) para determinar cuál de ellas pudo haber sido la más adecuada para realizar la estimación del esfuerzo de cada una de las iteraciones.

Palabras clave: Estimación de esfuerzo, métricas de software, puntos de función, COCOMO, casos de uso.

A STUDY OF FUNTION POINT BASED TECHNIQUES FOR EFFORT ESTIMATION IN SOFTWARE PROYECTS

ABSTRACT

Effort and cost estimation is still one of the hardest tasks in software project management. At the moment, there are many techniques to do this job, but there are not enough mature techniques for object-oriented or expert systems projects. Besides this problem, there is not much information about how to use those techniques in non-waterfall software lifecycles such as iterative or spiral lifecycles projects.

This article shows a comparative study of three techniques they could be used for effort estimation in each of all iterations in software construction. These techniques are based on function points and COCOMO II. One of these techniques has been tested in projects developed by one person and the results were satisfactory.

This study shows the results of two projects, one of them had 10 persons and the other 11 persons. The techniques have been applied to the real effort obtained in those projects. The purpose of this work is to conclude which technique will be the best alternative for effort estimation in each of all iterations.

Key words: Effort estimation, software metrics, function points, COCOMO II, use cases.

* Pontificia Universidad Católica del Perú, Sección Ingeniería de Informática, Lima-Perú.
E-mail: jpow@pucp.edu.pe

1. INTRODUCCIÓN

La creciente complejidad de los desarrollos software que provocó la denominada "crisis del software" se ha tratado de abordar mediante el planteamiento de nuevos métodos, metodologías, técnicas y paradigmas para minimizar su impacto. El alcance de dichas propuestas no se limita exclusivamente a actividades relacionadas con el desarrollo en sí de los sistemas, sino que abarca también las actividades de gestión de los mismos. Una de estas actividades es la de la estimación de los proyectos software.

A pesar de que la estimación de proyectos continúa siendo una tarea muy compleja, en muchas ocasiones dejada al albur de la pericia del experto estimador, en las últimas décadas se han desarrollado algunas técnicas para la estimación del esfuerzo de proyectos software completos, tales como puntos de función [19] y COCOMO II [4]. Aún así, estas técnicas -si bien se postulan como independientes de la tecnología final de desarrollo- fueron concebidas para su aplicación en sistemas basados en el paradigma estructurado con un ciclo de vida clásico o en cascada de Royce [15], y aún es difícil emplearlas en desarrollos orientados a objetos y ciclos de vida iterativo-incrementales, tan en boga en los últimos años. Incluso, parece interesante que éstas técnicas de estimación exploten para sus propósitos la información proporcionada por prácticas muy extendidas últimamente, como, por ejemplo, la de los casos de uso.

Teniendo en cuenta la problemática planteada, el presente artículo muestra un estudio realizado a dos equipos de desarrollo de 10 y 11 personas que han utilizado un modelo iterativo-incremental para el desarrollo de un sistema software. De acuerdo a estos resultados, se puede concluir que técnica es la más adecuada para realizar la estimación del esfuerzo para cada iteración.

Este documento se ha estructurado de la siguiente manera: la sección 2 muestra un breve resumen de las técnicas de estimación y su relación con los casos de uso; la sección 3 muestra las tres técnicas motivo del estudio; la sección 4 presenta los resultados obtenidos al aplicar dichas técnicas; y finalmente, se presentan las conclusiones obtenidas de este trabajo.

II. LAS TÉCNICAS DE ESTIMACIÓN Y LOS CASOS DE USO

Esta sección muestra un breve resumen de las técnicas de puntos de función y COCOMO II, y su relación con los casos de uso. Además, se da una visión general del trabajo que se ha encontrado con relación al tema.

2.1 La técnica de puntos de función

La técnica de puntos de función [19] fue introducida por Albrecht [1] y su propósito es medir el software cualificando la funcionalidad que proporciona externamente, basándose en el diseño lógico del sistema. Los objetivos de los puntos de función son:

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

Proporcionar un factor de normalización para la comparación de distintos software.

El análisis de los puntos de función se desarrolla considerando cinco parámetros básicos externos del sistema:

1. Entrada (EI, del inglés *External Input*).
2. Salida (EO, del inglés *External Output*).
3. Consultas (EQ, del inglés *External Query*).
4. Ficheros Lógicos Internos (ILF, del inglés *Internal Logic File*).
5. Ficheros Lógicos Externos (EIF, del inglés External Interface File).

Con estos parámetros, se determinan los puntos de función sin ajustar (PFsA). A este valor, se le aplica un factor de ajuste obtenido en base a unas valoraciones subjetivas sobre la aplicación y su entorno; es decir, las características generales del sistema.

2.2 COCOMO II y los puntos de función

COCOMO II, propuesto y desarrollado por Barry Boehm [4], es uno de los modelos de estimación de costos mejor documentados y utilizados. El modelo permite determinar el esfuerzo y tiempo que se requiere en un proyecto de software a partir de una medida del tamaño del mismo expresada en el número de líneas de código que se estimen generar para la creación del producto software.

Debido a la complejidad de los proyectos de software, el modelo original fue modificado, denominándose al modelo actual COCOMO II. El nuevo modelo permite estimar el esfuerzo y tiempo de un proyecto de software en dos etapas diferentes: diseño temprano y post-arquitectura. La tabla 1 muestra los drivers de coste del modelo de post-arquitectura que permitirán determinar el factor de ajuste "EAF" que se multiplicará al esfuerzo nominal para determinar el esfuerzo del proyecto.

Tabla N.º 1: Drivers de Coste para el modelo de post-arquitectura de COCOMO II

Driver de Coste (Post-arquitectura)	Descripción
RELY	Fiabilidad requerida del software
DATA	Tamaño de la base de datos
CPLX	Complejidad del producto
RUSE	Reusabilidad requerida
DOCU	Documentación de acuerdo a las necesidades del ciclo de vida
TIME	Restricción de tiempo de restricción
STOR	Restricción de almacenamiento principal
PVOL	Volatilidad de la plataforma
ACAP	Capacidad de analistas
PCAP	Capacidad de programadores
PCON	Continuidad del personal
AEXP	Experiencia en aplicaciones
PEXP	Experiencia de plataforma
LTEX	Experiencia de lenguajes y herramientas
TOOL	Uso de herramientas de software
SITE	Desarrollo en múltiples lugares

El nuevo modelo permite determinar el esfuerzo y tiempo de un proyecto de software a partir de los puntos de función sin ajustar, lo cual supone una gran ventaja, dado que en la mayoría de los casos es difícil determinar el número de líneas de código de que constará un nuevo desarrollo, en especial cuando se tiene poca o ninguna experiencia previa en proyectos de software. Esto hace que ambos modelos, Puntos de función y COCOMO II, sean perfectamente compatibles y complementario.

2.3. Los casos de uso y la técnica de estimación de puntos de función

Los casos de uso fueron introducidos en 1987 como una herramienta de la técnica *Objetory* [16]. Su utilización en los procesos de Ingeniería de Software fue propuesta por Ivar Jacobson y publicada en su libro *Object Oriented Software Engineering* [7]. Actualmente, su empleo se ha extendido aún más, debido a su inclusión en UML [10], por lo que su uso en el desarrollo de software orientado a objetos se ha vuelto altamente recomendable, y obligatorio.

David Longstreet, en uno de sus artículos [8], señala que el análisis de puntos de función se puede aplicar de manera sencilla con los casos de uso mejorando la calidad de los documentos de requerimientos y, a la vez, mejorando la estimación del proyecto de software. El aplicar la técnica de puntos de función permite verificar y validar el contenido de un documento de especificación de Reruisitos de software.

Lo interesante de la aplicación de ambas técnicas es que se puede actualizar el conteo de los puntos de función cada vez que los casos de uso cambien y, de esta manera, determinar el impacto que un caso de uso específico puede producir en la estimación del desarrollo de todo el proyecto.

Thomas Fetcke [5] propone una forma para utilizar la técnica de puntos de función con la metodología OOSE de Jacobson [7]. La relación que muestra se basa en los casos de uso y en el diagrama de clases de análisis propuestos por dicha metodología. En su propuesta no especifica el tipo de ciclo de vida que se debe seguir con su técnica, por lo que se podría inferir que se debería usar el modelo ciclo de vida en cascada.

2.4. Propuesta para estimación y planificación de iteraciones

En [11] se propuso una técnica para estimar y planificar las iteraciones en base a puntos de función y COCOMO II. Esta técnica ha sido probada en proyectos cuyo desarrollo es realizado por una persona con resultados satisfactorios.

La técnica en mención propone dos pasos principales: en primer lugar, determinar los casos de uso a realizar por iteración y, posteriormente, estimar el esfuerzo para cada iteración. A continuación se detallan las actividades que se siguen en cada uno de los pasos mencionados.

2.4.1 Paso 1: determinar los casos de uso a realizar por iteración

El objetivo de esta tarea es determinar los casos de uso que se deberán implementar en cada iteración. Para ello, se deberá haber realizado previamente la especificación de todos los casos de uso del software a construir y que deberán estar en el documento de especificación de requisitos de software (para la especificación de casos de uso, se puede revisar [2][3][12] y para el documento de especificación de requisitos de software, [6][14]).

Para esta actividad se propone la utilización de un nuevo diagrama al que se ha denominado "diagrama de precedencias", en el cual se muestran gráficamente las precondiciones de cada uno de los casos de uso que se incluyen en sus respectivas especificaciones.

La idea de dicho diagrama fue tomada de Doug Rosengberg [17] quien propone la realización de un diagrama similar al propuesto, especificando las relaciones "precede" e "invoca" (invoke en inglés) para determinar los requerimientos de usuario. La Figura N.º 1, muestra un ejemplo de un diagrama de precedencias.

El diagrama se utilizará para saber qué caso de uso necesita de alguna funcionalidad o información que es administrada o implementada por otro caso de uso. Esto permitirá conocer qué caso de uso debe programarse antes que otro, de manera que lo necesario para que se pueda implementar un caso de uso ya haya sido desarrollado en una iteración previa.

Siguiendo la idea del párrafo anterior, los casos de uso que se encuentran a la izquierda del diagrama, se implementarán antes de los que se encuentran a la derecha; es decir, en el ejemplo propuesto en la Figura N.º 1, el "Caso de uso A" se deberá implementar antes que "Caso de uso C".

Es importante resaltar que en este diagrama no se consideran los casos de uso incluidos y extendidos, ya que éstos se pueden considerar como parte de aquellos que les hacen referencia.

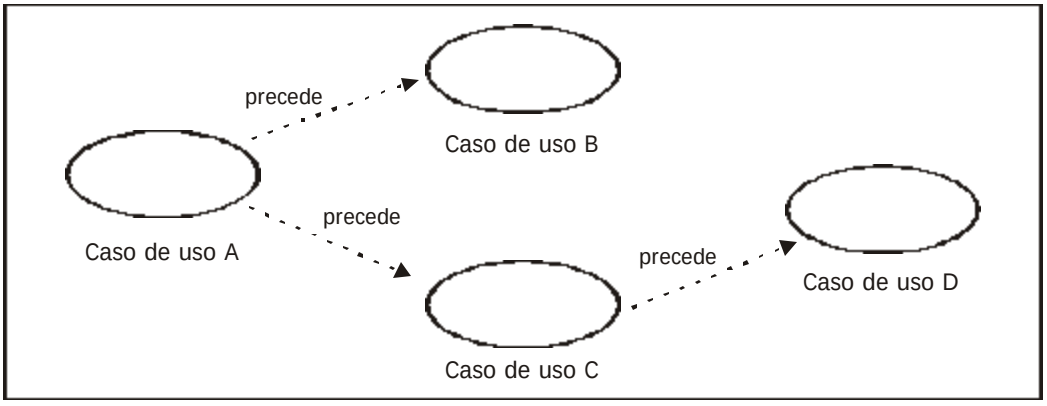

Figura N.º 1. Ejemplo de diagrama de precedencias

2.4.2 Paso 2: estimación del esfuerzo para cada iteración

La siguiente actividad, después de determinar qué caso de uso se realizará en cada iteración, consiste en determinar el esfuerzo que tomará cada iteración. Para ello, se propone la utilización de las técnicas de puntos de función y COCOMO II.

La primera actividad consiste en determinar los puntos de función sin ajustar para cada iteración. El criterio seguido, para adaptar la técnica de puntos de función para ciclos de vida iterativos-incrementales, consiste en considerar que el resultado del cálculo de los puntos de función sin ajustar para todo el proyecto y sin considerar iteraciones, deberá ser igual a la suma de los puntos de función sin ajustar (PFsA) calculados para cada iteración por separado. La fórmula 1 muestra la hipótesis planteada.

$$PFsA\_Total = \sum_{i=1}^{n} PFsA(i)$$

Donde:
PFsA_Total = puntos de función sin ajustar de todo el proyecto, calculado sin considerar iteraciones.
PFsA(i) = puntos de función sin ajustar de la iteración "i".

Fórmula 1: Equivalencia de los PFsA totales versus los PFsA de las iteraciones.

Uno de los aportes más importantes de esta propuesta consiste en determinar los puntos de función sin ajustar correspondientes a los ficheros lógicos internos (ILF) y ficheros de interfaz externa (EIF) para cada caso de uso. Para ello, esta técnica propone utilizar la siguiente fórmula (fórmula 2).

$$PFsA\_F(j) = \sum_{i=1}^{n} \frac{1}{TCU(i)} xPeso(i)$$

Donde:
PFsA_F = punto de función sin ajustar para un caso de uso "j" debido a los ILF/EIF.
TCU(i) = número total de casos de uso que utiliza un ILF/ EIF "i".
Peso(i)= peso debido a la complejidad del ILF/EIF "i".
i= ILF/ EIF utilizado en el caso de uso "j".
j= caso de uso en cuestión.

Fórmula 2. Cálculo de PFsA debido a ILF/EIF por cada caso de uso

Con los resultados obtenidos con la fórmula anterior y sabiendo qué caso de uso se desarrollará en cada iteración, se determinan los puntos de función sin ajustar debido a los ficheros por iteración. A esto se le añadirán los PFsA correspondientes a las transacciones. Para ello, se utiliza la fórmula 3.

Luego, utilizando COCOMO II y con los puntos de función sin ajustar correspondientes a cada iteración, se obtiene el esfuerzo en horas-hombre de cada una de las iteraciones y, con este valor, se puede determinar el tiempo y personas necesarias para acabar la iteración y por ende el tiempo de todo el proyecto.

Es importante resaltar que el contexto del proyecto puede cambiar al pasar de una iteración a otra (conocimiento de la plataforma de desarrollo, integración del equipo de desarrollo, etc.), por lo que podría ser necesario reestimar de nuevo el esfuerzo requerido para las iteraciones siguientes, revisando ficheros (ILF/EIF) y transacciones (EI, EO y EQ) en caso de cambio de requisitos, y recalculando *drivers de coste* propuestos por COCOMO II en caso de cambios contextuales u organizacionales.

$$TPFsA\_1(i) = \sum_{j=1}^{n} [PFsA\_F(j)] + \sum_{j=1}^{n} [PFsA\_Trans(j)]$$

Donde:
i= iteración específica
TPFsA(i)_1= total de puntos de función sin ajustar para una iteración "i".
PFsA_F(j) = punto de función sin ajustar para un caso de uso "j" debido a ILF/EIFs.
PFsA_Trans(j) = punto de función sin ajustar para un caso de uso "j" debido a las transacciones (EI,EO,EQ).
j= caso de uso a implementar en una iteración "i".

Fórmula 3. Cálculo de PFsA para cada iteración para la técnica 1.

III. TÉCNICAS PARA ESTIMAR EL ESFUERZO DE LAS ITERACIONES DE CONSTRUCCIÓN

La principal dificultad para estimar el esfuerzo necesario para el proyecto se encuentra en el cálculo de puntos de función sin ajustar para cada una de las iteraciones. A continuación, se muestran las posibles técnicas que se podrían utilizar para este fin:

3.1 Técnica 1

Esta técnica es la propuesta en [11] y es explicada brevemente en la sección 2.4 de este artículo. Para ello, se utiliza las fórmulas 2 y 3 para calcular los puntos de función sin ajustar para cada una de las iteraciones.

3.2 Técnica 2

Esta técnica es una variación a la técnica 1 y corresponde a la hipótesis en las que las sólo se deben considerar los puntos de función sin ajustar de los ILFs para las entradas externas (EI), ya que es en este tipo de transacciones en las que se deben crear las estructuras de la base de datos. En las otras transacciones (EO, EQ), básicamente, sólo se realizan consultas de la información almacenada y cuya estructura ya ha sido creada.

Para esta técnica, se utiliza la fórmula que se muestra a continuación.

$$PFsA\_F\_EI(j) = \sum_{i=1}^{n} \frac{1}{TCU\_EI(i)} xPeso(i)$$

Donde:
PFsA_F_EI = punto de función sin ajustar para un caso de uso "j" debido a los ILF/EIF de los EI.
TCU_EI(i) = número total de casos de uso que contienen un EI que utiliza un ILF/ EIF "i".
Peso(i)= peso debido a la complejidad del ILF/EIF "i".
i= ILF/ EIF utilizado en el caso de uso "j".
j= caso de uso en cuestión.

Fórmula 4. Cálculo de PFsA debido a ILF/EIF que usan los EI por cada caso de uso.

A continuación, según fórmula 5, se obtienen los puntos de función sin ajustar para cada iteración debido a los ILFs o EIFs, según la fórmula 5, que se muestra a continuación:

$$TPFsA\_2(i) = \sum_{j=1}^{n} [PFsA\_F\_EI(j)] + \sum_{j=1}^{n} [PFsA\_Trans(j)]$$

Donde:
i= iteración específica
TPFsA_2(i)=a total de puntos de función sin ajustar para una iteración "i".
PFsA_F(j) = punto de función sin ajustar para un caso de uso "j" debido a ILF/EIFs.
PFsA_Trans(j) = punto de función sin ajustar para un caso de uso "j" debido a las transacciones (EI,EO,EQ).
j= caso de uso a implementar en una iteración "i".

Fórmula 5. Cálculo de PFsA para cada iteración para la técnica 2.

3.3 Técnica 3

Esta técnica se relaciona a la hipótesis en la que no es necesario considerar los puntos de función sin ajustar de los ficheros (ILF, EIF) y sólo se deben utilizar los puntos de función sin ajustar relacionados a las transacciones. A continuación se muestra la fórmula utilizada para esta técnica.

$$TPFsA\_3(i) = \sum_{j=1}^{n} [PFsA\_Trans(j)]$$

Donde:
i= iteración específica
TPFsA_3(i)= total de puntos de función sin ajustar para una iteración "i".
PFsA_Trans(j) = punto de función sin ajustar para un caso de uso "j" debido a las transacciones (EI,EO,EQ).
j= caso de uso a implementar en una iteración "i".

Fórmula 6. Cálculo de PFsA para cada iteración para la técnica 3.

4. CARACTERÍSTICAS DEL PROYECTO MATERIA DEL ESTUDIO

El proyecto de software se realizó dentro de una asignatura del cuarto año de la carrera de Ingeniería Informática. La duración total correspondía a un semestre académico (14 semanas). A continuación se muestran las características del proyecto y de los integrantes de los equipos:

- Características del proyecto
  - Metodología de desarrollo: Rational Unified Process [14].
  - Número de integrantes por grupo: 10 u 11.
  - Tema: Sistema para Cadena de Restaurantes de Cómida Rápida (fast food).

  - Semanas por iteración en construcción: 2 a 3 semanas.
- Características Tecnológicas:
  - Herramienta de desarrollo: Delphi 5.0.
  - Plataforma de desarrollo: Microsoft Windows 2000.
  - Base de datos: Microsoft SQL Server 2000.
- Habilidades y conocimientos del equipo:
  - Conocimiento de lenguajes de programación: Java, C#, Pascal y C.
  - Conocimiento de base de datos: Oracle 8.
  - Conocimiento de análisis y diseño: orientado y estructurado a objetos.

Siguiendo la metodología RUP[14], previo a la fase de construcción de software, cada grupo completó la especificación de requisitos del software utilizando casos de uso. Luego, se determinó la arquitectura del sistema, la cual se basó en el modelo cliente/servidor de dos capas de tipo "solución mixta" [18], y se realizó un prototipo para validarlo.

Para planificar qué caso de uso se debería implementar en cada iteración de la fase de construcción, se utilizó un diagrama de precedencias de casos de uso propuesto en [11] (vea sección 2.4 de este artículo), en el cual se muestran gráficamente las precondiciones de cada uno de los casos de uso que se incluyen en sus respectivas especificaciones.

Para la fase de construcción, se indicó que la repartición de la carga de trabajo por integrante se debería realizar utilizando los puntos de función sin ajustar relacionados a las transacciones y que no consideraran los puntos correspondientes a los ficheros (ILF,EIF). A los estudiantes no se les explicó ninguna de las técnicas mostradas en la sección anterior (sección 3 de este artículo) para evitar algún posible sesgo en los tiempos utilizados para el desarrollo del software.

V. RESULTADOS

Para entender los resultados obtenidos en el estudio es importante considerar lo siguiente:

Sólo dos integrantes del equipo 2 conocían Delphi 5.0 antes de iniciar el proyecto. La mayoría de los integrantes tuvieron que aprender la herramienta en el transcurso del desarrollo del proyecto.

Los integrantes de los equipos habían trabajado de manera conjunta en proyectos de asignaturas de semestres anteriores, por lo que el contexto correspondiente a las relaciones interpersonales fue la misma en todas las iteraciones.

De acuerdo a lo indicado anteriormente, se pudo observar que el único cambio que se produjo en el contexto de cada una de las iteraciones fue el conocimiento de la herramienta de programación.

5.1 Factor de ajuste "EAF" de COCOMO II

Según lo conversado con los equipos, luego de finalizado el proyecto y utilizando los *driver de coste* de COCOMO II, se pudo determinar el factor de ajuste EAF de cada iteración, los cuales se muestran en la tabla N.º 2.

Para ambos equipos, se consideró a todos los *drivers de coste* con valor "nominal", a excepción de los drivers AEXP y LTEX, los cuales se relacionan a la experiencia de desarrollo de aplicaciones, de la herramienta y del lenguaje de programación.

De la tabla N.º 2, se puede observar lo siguiente:

Para el equipo 1, la primera iteración debería tomar mayor esfuerzo x PFsA que la segunda y tercera iteración. La segunda y tercera iteración deberían tomar el mismo esfuerzo x PfsA.

Para el equipo 2, la primera y segunda iteración deberían tomar el mismo esfuerzo x PFsA. La tercera iteración debería tomar menor esfuerzo x PFsA que la primera y la segunda.

Tabla N.º 2: Factor de ajuste "EAF" de COCOMO II por iteración y equipo

Iteración de Construcción	Factor "EAF" del equipo 1	Factor "EAF" del equipo 2
1	1.46	1.20
2	0.80	1.20
3	0.80	0.80

5.2 Resultados del equipo 1

La tabla N.º 3 muestra el resultado del equipo 1. La primera columna muestra el número de iteración; la segunda, el esfuerzo real utilizado en horas-hombre; la tercera a la quinta, los puntos de función sin ajustar por técnica y la sexta a la octava, el esfuerzo real en horas-hombre por punto de función sin ajustar de cada una de las técnicas.

De la tabla anterior, se puede observar lo siguiente:

En las tres técnicas, el esfuerzo utilizado por punto de función es menor en la última iteración que en las dos primeras iteraciones, lo cual no difiere a lo observado en la tabla N.º 2.

De acuerdo la tabla N.º 2, la primera y segunda iteración deberían tener el mismo esfuerzo por PFsA, lo cual no se observa en ninguna de las tres técnicas. Sin embargo, la técnica 1 es la que tiene menor diferencia entre ambas iteraciones (sólo 0,02), en cambio las técnicas 2 y 3 tienen mucha mayor diferencia, incluso la primera iteración necesita de un mayor esfuerzo que en la segunda.

Tabla N.º 3. Resultados para el equipo 1

Iter. Const.	Esfuerzo Real (horas)	TPFsA_1 (Téc. 1)	TPFsA_2 (Téc. 2)	TPFsA_3 (Téc. 3)	Esf_Real por PFsA (Téc. 1)	Esf_Real por PFsA (Téc. 2)	Esf_Real por PFsA (Téc. 3)
1	465	114.05	116.50	78.00	4.08	3.99	5.96
2	715	282.32	290.25	187.00	2.53	2.46	3.82
3	334	129.63	119.25	72.00	2.58	2.80	4.64

De la tabla anterior, se puede observar lo siguiente:

En las tres técnicas, el esfuerzo utilizado por punto de función es mayor en la primera iteración, lo cual no difiere a lo observado en la tabla N.º 2.

De acuerdo la tabla N.º 2, la segunda y tercera iteración deberían tener el mismo esfuerzo por PFsA, lo cual no se observa en ninguna de las tres técnicas. Sin embargo la técnica 1 es la que tiene menor diferencia entre ambas iteraciones (sólo 0,01), en cambio las técnicas 2 y 3 tienen mucha mayor diferencia (más de 0,3).

Según las observaciones hechas en la tabla N.º 3, podemos decir que la técnica 1 podría dar una mayor precisión para el cálculo del esfuerzo de cada una de las iteraciones.

5.3 Resultados del Equipo 2

La tabla N.º 4 muestra el resultado del equipo 2 de manera similar a la tabla 3

Según las observaciones hechas en la tabla 4, podemos decir que la técnica 1 podría dar una mayor precisión para el cálculo del esfuerzo de cada una de las iteraciones.

5.4 Coeficiente de correlación lineal para los resultados de ambos equipos

La tabla N.º 5 muestra el coeficiente de correlación lineal entre el factor de ajuste EAF mostrados en la tabla N.º 2 y el esfuerzo x PFsA de las técnicas 1, 2 y 3 de las tablas N.º 3 y 4.

De la tabla N.º 5, se puede observar lo siguiente:

Tabla N.º 4: Resultados para el equipo 2

Iter. Const.	Esfuerzo Real (horas)	TPFsA_1 (Téc. 1)	TPFsA_2 (Téc. 2)	TPFsA_3 (Téc. 3)	Esf_Real por PFsA (Téc. 1)	Esf_Real por PFsA (Téc. 2)	Esf_Real por PFsA (Téc. 3)
1	191	100.13	121.33	70.00	1.91	1.57	2.73
2	131	69.14	58.90	47.00	1.89	2.22	2.79
3	133	97.71	86.80	63.00	1.36	1.53	2.11

Tabla N.º 5. Coeficiente de correlación de regresión lineal de EAF vs esfuerzo por PFsA

Equipo	Coeficiente Correlación Técnica 1	Coeficiente Correlación Técnica 2	Coeficiente Correlación Técnica 3
1	0.9997	0.9777	0.9259
2	0.9998	0.5461	0.9969

En las técnicas 1 y 2, se observa una alta correlación entre el factor de ajuste (EAF) de COCOMO II y el esfuerzo por PFsA.

En la técnica 2, se observa una alta correlación entre el factor de ajuste (EAF) de COCOMO II y el esfuerzo por PFsA sólo en el equipo de trabajo 1, mas no en el equipo de trabajo 2.

La técnica 1 muestra una mayor correlación entre el factor de ajuste (EAF) de COCOMO II y el esfuerzo por PFsA que las otras técnicas.

De los resultados obtenidos, se puede observar que la técnica 1 ofrece una mayor precisión para determinar el esfuerzo x PFsA para cada iteración que las otras técnicas. Por consiguiente, se puede afirmar que la técnica 1 es la más adecuada para realizar las estimaciones de esfuerzo de las iteraciones de proyectos software.

VI. CONCLUSIONES

La mayoría de las aproximaciones actuales para estimar proyectos, aún definiéndose como independientes de la tecnología y modelos de ciclo de vida, tienen un carácter fuertemente influido por los ciclos de vida en cascada. A pesar de que son válidas para otras aproximaciones, por ejemplo como los ciclos de vida iterativos-incrementales, en general no ofrecen ninguna guía para acometer dicha adaptación.

El presente trabajo muestra que la propuesta dada en [11] se podría utilizar para proyectos con ciclos de vida iterativos-incrementales y con casos de uso, en equipos de desarrollo en los que participan más de una persona. En términos generales, a partir de los casos de uso identificados para cada iteración, se calcula la proporción de utilización de puntos de función sin ajustar por caso de uso, tal y como se detalla en la técnica 1 (vea la sección 3). A continuación se obtienen los puntos de función sin ajustar por cada caso de uso debidos a las transacciones (EI, EO, EQ). Por último, se determinan los puntos de función sin ajustar por cada iteración y, a partir de ellos, y mediante la aplicación del método de COCOMO II, se obtiene el esfuerzo también de cada iteración.

En un futuro, una vez que la técnica esté estabilizada, será aplicada a nuevos desarrollos de características más amplias, como equipos de desarrollo más numerosos o aplicaciones más complejas. También, se determinará la viabilidad de modificar la técnica de puntos de función de manera que se puedan reemplazar los ficheros lógicos internos y ficheros de interfaz externa por clases y diagramas de clases, para que dicha técnica pueda adaptarse completamente al enfoque orientado a objetos.

VII. BIBLIOGRAFÍA

1. Albrecht, A. J. *Measuring Application Development Productivity, IBM Applications Development Symposium*, Monterey, CA, USA, 1979.
2. Bittner, K. *Use Case Modeling, Addison-Wesley.* USA, 2003.
3. Bittner, K. *Why Use Cases Are Not Functions*. http://www.therationaledge.com, USA, 2000.
4. Boehm, B. *COCOMO II Model Definition Manual*, http://sunset.usc.edu/research/COCOMOII, USA, 1999.
5. Fetcke, T., Bran, A., Nguyen, T. *Mapping the OO-Jacobson Approach into Function Point Analysis*. Proceedings of TOOLS-23'97, IEEE, USA, 1997.
6. *IEEE Computer Society, IEEE Std 830-1998, Recommended Practice for Software Requirements Specifications, The Institute of Electrical and Electronics Engineers*. USA, 1998.
7. Jacobson, I. *Object-Oriented Software Engineering. A Use Case Driven Approach, Addison-Wesley*. USA, 1992.
8. *Longstreet, D.,Use Case and Function Points*, http://www.softwaremetrics.com/, Longstreet Consulting Inc, USA, 2001.

9. *Ministerio de Administraciones Públicas, Métrica Versión 3*. http://www.map.es, España, 2000.

10. *Object Management Group, OMG Unified Modeling Language*. http://www.uml.org, USA, 1999.

11. Pow-Sang, J., Imbert R. *Estimación y Planificación de Proyectos Software con Ciclo de Vida Iterativo-Incremental y empleo de Casos de Uso*. Proceedings IDEAS 2004, ISBN 9972-9876-1-2, Arequipa-Perú, 2004.

12. Pow-Sang, J. *La Especificación de Requisitos con Casos de Uso: Buenas y Malas Prácticas*. II Simposio Internacional de Sistemas de Información e Ing. de Software en la Sociedad del Conocimiento-SISOFT 2003, Pontificia Universidad Católica del Perú, Lima-Perú, 2003.

13. Pow-Sang, J. *GESPROMET, Sistema para la Gestión de Proyectos de Software utilizando MÉTRICA Versión 3*. Tesis de Máster en Ingeniería del Software, Universidad Politécnica de Madrid, España, 2002.

14. *Rational Software, Rational Unified Process version* 2001A.04.00.13, USA, 2001.

15. Royce, W. W. *Managing the Development of Large Software Systems: Concepts and Techniques*. Proceedings WESCON, 1970.

16. Rumbaugh, I. Jacobson, and G. Booch, *Unified ModellingLanguage Reference Manual*. Addison Wesley, 1997.

17. Rosenberg, D., Scott, K. *Use Case Driven Object Modeling with UML*. Addison-Wesley, Massachusets, USA, 1999.

18. Sadoski, D. *Two Tier Software Architectures*, http://www.sei.cmu.edu/str/descriptions/twotier.html, SEI, USA, 2004.

19. *The International Function Point User Group (IFPUG)*. Function Point Counting Practices Manual-Release 4.1, USA, 1999.