Estimación y Planificación de Proyectos Software con Ciclo de Vida Iterativo-Incremental y empleo de Casos de Uso

José Antonio Pow-Sang Portillo
Pontificia Universidad Católica del Perú
Sección Ing. Informática
Perú
japowsang@pucp.edu.pe

Ricardo Imbert Paredes
Universidad Politécnica de Madrid
Facultad de Informática
España
rimbert@fi.upm.es

Resumen

La estimación de costos y esfuerzos sigue siendo una de las tareas más difíciles en la gestión de un proyecto de software. Esta actividad es realizada por el jefe de proyecto, quien es responsable de hacer dichas estimaciones lo más precisas posible. 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 los sistemas expertos. A ello se suma el problema de la escasa información proporcionada por las técnicas de estimación existentes para su aplicación a ciclos de vida de desarrollo de software diferentes al de cascada, como, por ejemplo, los ciclos de vida iterativo-incrementales o en espiral.

El presente artículo presenta una propuesta para estimar y planificar proyectos software cuyo ciclo de vida sea el iterativo-incremental. Dichas actividades se realizarán a partir de la especificación de los casos de uso del sistema, la cual se ha debido realizar previamente.

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 [17] 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 [14], 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 propone una técnica para estimar y planificar las iteraciones en proyectos orientados a objetos, basada en los casos de uso de los mismos, la técnica de puntos de función y el método de COCOMO II.

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, plantea una propuesta para la

estimación y planificación de las iteraciones; y, finalmente, se presentan las conclusiones obtenidas de este trabajo.

2. 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 respecto al tema.

2.1 La técnica de Puntos de Función

La técnica de Puntos de Función [17] 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 de 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, 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 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 complementarios.

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 [15] . 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, si no 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 Requisitos 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.

3. Propuesta para Estimación y Planificación de Iteraciones

La propuesta para determinar la estimación y planificación del esfuerzo de un desarrollo que se apoya en un ciclo de vida iterativo-incremental comprende 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.

3.1 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][11] y para el documento de especificación de requisitos de software, [6][13]).

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 Rosenberg [16] 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 1, muestra un ejemplo de un diagrama de precedencias.

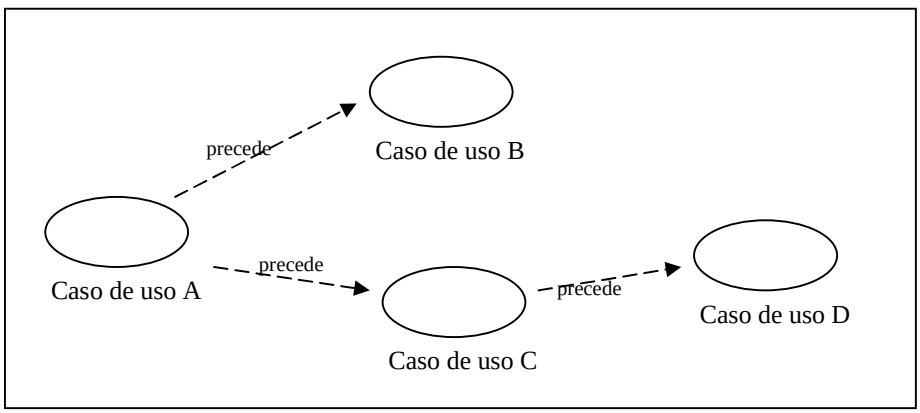

Figura 1. Ejemplo de Diagrama de Precedencias

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 1, “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.

3.2 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.

3.2.1 Paso 2.a: Determinar los puntos de función sin ajustar

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

A continuación se detallan las tareas a realizar en este paso.

a) Determinar los puntos de función sin ajustar para ILF/EIF por caso de uso.

Para esta tarea, inicialmente, se determinan los ficheros, sin importar si son lógicos internos (ILF) o de interfaz externos (EIF), que serán utilizados por cada caso de uso y, a continuación, se determinan los DETs y RETs de cada ILF y EIF. Para esto, se deberá seguir los pasos que especifica la técnica de puntos de función [17].

Posteriormente, se determinan los puntos de función sin ajustar correspondientes a un fichero lógico interno o externo para cada caso de uso. Para ello, se utiliza 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

A continuación, se obtienen los puntos de función sin ajustar para cada iteración debido a los ILFs o EIFs, según la fórmula 3, que se muestra a continuación:

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

Donde:

i= iteración específica
TPFsA(i)= 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.
j= caso de uso a implementar en una iteración "i".

Fórmula 3. Cálculo de PFsA por cada iteración

b) Determinar los puntos de función sin ajustar para las transacciones de cada iteración

Para cada iteración se realiza, por cada caso de uso, lo siguiente:

- Se determinan las entradas externas (EI), salidas externas (EO) o consultas externas (EQ) de cada caso de uso, utilizando para ello las reglas de la técnica de puntos de función [17].
- Se determinan los DETs, FTRs para cada EI, EO y EQ utilizando para ello las reglas especificadas en la técnica de puntos de función.
- Con los datos obtenidos anteriormente, se calculan los puntos de función sin ajustar para cada EI, EO y EQ.

Como resultado de los dos pasos anteriores se obtiene el número de puntos de función sin ajustar para cada iteración.

3.2.2 Paso 2.b Determinación del esfuerzo utilizando COCOMO II

Para realizar este paso, en primer lugar es preciso estimar cada uno de los *drivers de coste* propuestos por el método COCOMO II (RELY, DATA, etc), correspondientes a cada iteración.

A partir de estos valores y los puntos de función sin ajustar correspondientes a cada iteración, se obtiene el esfuerzo en meses 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* en caso de cambios contextuales u organizacionales.

4. Conclusiones y trabajo futuro.

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, como los ciclos de vida iterativos-incrementales, por ejemplo, en general no ofrecen ninguna guía para acometer dicha adaptación.

El presente trabajo muestra una propuesta para determinar el esfuerzo de proyectos software que emplean ciclos de vida iterativos-incrementales y con casos de uso. 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. 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.

Esta técnica propuesta ha sido usada en proyectos de software en los que participa una sola persona con resultados esperanzadores: los desajustes entre esfuerzos estimados y esfuerzo real nunca sobrepasó el 10%. Esto ha animado a que la propuesta haya sido aplicada en nuevos proyectos, con el fin de refinar el proceso.

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.

5. 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., *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.

[12] 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.

[13] Rational Software, *Rational Unified Process version 2001A.04.00.13*, USA, 2001.

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

[15] Rumbaugh, I. Jacobson, and G. Booch, Unified ModellingLanguage Reference Manual, Addison Wesley, 1997.

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

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