Estimación del Esfuerzo Software: Factores vinculados a la aplicación a desarrollar

Joseba Esteban López
José Javier Dolado Cosín

Departamento de Lenguajes y Sistemas Informáticos
U.P.V./E.H.U.

Resumen. Durante la historia del software se han desarrollado multitud de métodos de estimación del esfuerzo de desarrollo en proyectos software. Cada uno de estos métodos basa su estimación en diferentes aspectos, también denominados *factores*. Este artículo tiene como objetivo listar y catalogar los principales factores utilizados en la estimación del esfuerzo software y que están relacionados con la aplicación a desarrollar . Para esto se han revisado los métodos de estimación más mencionados en la literatura, como son COCOMO, SLIM o analogía.

1. Introducción

La estimación del coste de desarrollo software, y por tanto la exactitud de esta, se ha convertido en un aspecto crucial tanto para la compañía desarrolladora del proyecto como para el cliente, el cual espera que el coste del software coincida lo más posible con la estimación. Una infravaloración en la estimación del coste software puede suponer que la planificación del proyecto acordado acabe sufriendo más problemas de los esperados inicialmente y una disminución de la calidad o retrasos en la fecha de entrega. Por el contrario, una sobrevaloración puede suponer un exceso de recursos reservados al proyecto, lo que conlleva un presupuesto excesivo del proyecto y, por tanto, una reducción de la competitividad de la compañía ante la asignación de nuevos proyectos. Una estimación lo suficientemente exacta permite a la compañía desarrolladora de software una mejor planificación de los proyectos que maneja, así como una mejor asignación de los recursos necesarios para cada proyecto. Además permite valorar el impacto ante cambios en el plan de desarrollo del proyecto [6]. En consecuencia, la exactitud de las estimaciones toma un cariz cada día más relevante dentro de la Ingeniería del Software.

Los modelos de estimación del coste de desarrollo software se basan en un conjunto de variables. Estas variables se denominan *factores*, como el coste, el esfuerzo o el tamaño entre otros. Cada modelo basa sus estimaciones en un conjunto propio de factores que pueden estar relacionados entre sí. La naturaleza de las relaciones causales entre los factores implica cierta incertidumbre: no son deterministas. Esto quiere decir que, suponiendo que exista una relación entre el esfuerzo de desarrollo y la calidad del software, no es necesariamente cierto que aumentando el esfuerzo se consiga una mejoría en la calidad, aunque si es probable que suceda. [8]

Hasta la fecha no hay un completo entendimiento sobre las relaciones causales entre factores ni de su influencia en el resultado final del proyecto software [4]. Las causas que impiden un mejor discernimiento sobre estas relaciones pueden deberse a que los métodos de estimación fallan en la representación de las relaciones causales y su incertidumbre. Asimismo se ve afectado por la gran cantidad de datos necesarios y la dificultad de obtenerlos [8]. Esta falta de información se ve acentuada con las estimaciones en las etapas iniciales del proyecto software debido a que la aplicación a desarrollar no está definida del todo. Por otro lado,

según va avanzando el desarrollo del proyecto, se conocen más datos sobre la aplicación y las relaciones entre los factores, y por tanto la exactitud de la estimación aumenta [5].

Los factores se pueden agrupar en factores del grupo de desarrollo, de la aplicación a desarrollar o relacionados con metodología de desarrollo. Este artículo muestra los factores relacionados con la aplicación a desarrollar más relevantes en la estimación del esfuerzo software. Comienza con una breve descripción de los métodos de estimación analizados. A continuación se analiza qué factores vinculados a la aplicación a desarrollar son utilizados por los métodos de estimación y cómo son valorados.

2. Métodos de estimación analizados

La estimación del coste software es el proceso de predecir el esfuerzo requerido para el desarrollo de un sistema software. La mayor parte del coste de desarrollo software es debido al esfuerzo humano [10], por lo que la mayoría de los métodos de estimación proporcionan sus estimaciones de esfuerzo en términos de persona-mes de los programadores, analistas y gestores de proyecto. La estimación del esfuerzo puede ir acompañada de otras unidades, como son la duración del proyecto (en días en el calendario) y el coste (basado en el coste medio en unidades de moneda de la plantilla implicada en el desarrollo). [6]

Históricamente la estimación del coste de desarrollo software ha supuesto la parte más compleja del desarrollo de software. Las principales causas de esta complejidad son:

- Ausencia de bases de datos históricas adecuadas de medidas de coste
- Los factores implicados en el desarrollo software están interrelacionados y sus relaciones no están bien entendidas
- Falta de práctica y habilidad del personal encargado de realizar las estimaciones

En este apartado se muestran los principales métodos de estimación sobre los que se ha basado el estudio de los factores de estimación del esfuerzo software. De cada uno de los métodos se incluye una breve descripción. Los métodos están clasificados en dos grupos: métodos algorítmicos y métodos no algorítmicos.

2.1. Métodos de estimación no algorítmicos

En este apartado se exponen brevemente los métodos de estimación que no se basan en modelos matemáticos. Estos incluyen la estimación experta y el uso de analogías entre otros.

Estimación experta El método de estimación más extendido es el juicio experto. Está basado en la experiencia del experto y su conocimiento de las normas dominantes de la industria como base para la estimación del coste software. Se puede considerar [5] como experto a aquel que cuenta con experiencia como supervisor o gestor, o tiene experiencia haciendo estimaciones. Los expertos suelen hacer sus estimaciones basándose en su experiencia y en analogías de proyectos anteriores.

A pesar de ser el método más extendido, las estimaciones basadas en el juicio experto no están libres de problemas, ya que este enfoque no es ni repetible ni explícito. Además de que contar con un experto experimentado para cualquier nuevo proyecto es muy complicado, sus estimaciones están sujetas a un alto grado de inconsistencia. Según el estudio realizado por Stein G. y Magne J. en , dos estimaciones de un experto sobre el mismo proyecto pueden variar de media hasta un 71 %, con una mediana del 50 %. Un intento de minimizar la inconsistencia de las estimaciones de los expertos, consiste en consultar a varios expertos. Como ejemplos de técnicas de consenso entre expertos están Delphi y PERT [6]. Por otro lado, los gestores de los proyectos software intentan maximizar la productividad y minimizar

el coste, y así hacer coincidir las estimaciones con el objetivo fijado con el cliente. Por tanto, los expertos basan sus estimaciones en estas manipulaciones de los presupuestos de proyectos anteriores de la compañía, con lo que su experiencia queda en entredicho.

Los factores que manejan los expertos a la hora realizar sus estimaciones no son muy conocidos. Por tanto, en este artículo no se hablará de este método en las siguientes secciones donde se analizan los factores empleados por los métodos de estimación.

Uso de analogías La utilización de analogías a la hora de estimar el coste de desarrollo software, consiste en comparar el nuevo proyecto a estimar con el coste real de proyectos anteriores de la compañía. La comparación entre proyectos se puede hacer comparando el proyecto en su totalidad o por partes. Comparar los proyectos basándose en su descomposición tiene la ventaja de ser un proceso más detallado que si se comparan proyectos en su totalidad. Por contra, comparar los proyectos en su totalidad lleva implícita la ventaja de incluir todos los componentes relativos al coste, incluso los componentes de los que no se tiene constancia.

La ventaja de este método radica en que las estimaciones se hacen en base al coste real de los proyectos de la compañía. Esto implica que la compañía debe disponer de una cantidad de datos de proyectos anteriores suficiente. Aún así, no queda muy claro que proyectos anteriores sean representativos de los nuevos proyectos. [6]

Una implementación de este método es la herramienta ANGEL [9] que se basa en la minimización de la distancia euclídea según las variables consideradas y que soporta la recolección, almacenaje e identificación de los proyectos más similares a la hora de estimar el esfuerzo.

Otros métodos En este apartado se comentan métodos menos extendidos dentro de los métodos de estimación no algorítmicos.

Por un lado encontramos el principio de Parkinson. Este principio determina el coste de desarrollo basandose en la máxima de que el trabajo se expande hasta completar todo el volumen disponible. Por ejemplo, si un proyecto debe estar terminado en 12 meses y se dispone de 5 personas, el esfuerzo quedará determinado como 60 personas/mes. [6]

Price-To-Win es un método consistente en determinar el coste del proyecto en función del precio que está dispuesto a pagar el cliente. Con este método es habitual que el encargado de realizar la estimación se vea obligado a ajustarla en función de lo que quiere pagar el cliente y, así, conseguir que le asignen el proyecto a la compañía. La aplicación de este método puede derivar en retrasos en el desarrollo o en que la plantilla se vea obligada a hacer horas extra. Al igual que el método de Parkinson, no se trata de un método recomendable a la hora de estimar el esfuerzo.[6]

Los enfoques Bottom-up y Top-down generan estimaciones basándose en la descomposición del software en componentes. Con Bottom-up se estima por separado cada componente del sistema. Posteriormente se unen todas las estimaciones para generar la estimación del proyecto en su totalidad. El único requerimiento para este enfoque radica en conocer perfectamente cómo se descompone el sistema, algo complicado en las etapas iniciales del desarrollo. Por el contrario, con el enfoque Top-down primero se realiza una estimación global del proyecto, utilizando el método de estimación que se prefiera, y a continuación se divide en componentes. Este enfoque resulta más útil en fases tempranas del desarrollo [6]. Ambos enfoques no son métodos de estimación como tales, sino estrategias para facilitar la tarea de estimar el esfuerzo de desarrollo software.

2.2. Métodos de estimación algorítmicos

Los métodos de estimación algorítmicos realizan sus estimaciones de coste en función de un conjunto de variables y parámetros. Estas variables están consideradas como los principales factores de coste. Los diferentes métodos de estimación algorítmicos se diferencian tanto por los factores que emplean en su modelo, como por la forma de la función que utilizan. Dependiendo del tipo de formula matemática que utiliza un modelo, se puede clasificar en modelos lineales, multiplicativos o con función de rendimiento (Power Function Models). Los métodos algorítmos se dividen en dos grupos: analíticos y empíricos [6].

Métodos de estimación algorítmicos empíricos Los métodos de estimación empíricos se basan en la experiencia a la hora de realizar una formulación matemática que modele el esfuerzo de desarrollo software. En este apartado se describe brevemente el método de estimación COCOMO.

COCOMO (COnstructive COst MOdel) Este método de estimación fue ideado por B. W. Boehm [1] a principios de la década de los 80. Está basado en el método de Walston-Felix [10]. Dado que en este artículo se estudian los factores relacionados con el esfuerzo y la aplicación a desarrollar, de ahora en adelante se considerará COCOMO y sus factores.

Se trata de un método de estimación basado en una función de rendimiento de la forma: $Effort = a \times S^b \times m(X)$

Donde S es el tamaño del código en miles de líneas (KLOC). Las otras dos variables, a y b, son coeficientes dependientes de la complejidad del software. Con $m(X)$ se representa un multiplicador dependiente de 15 factores. En COCOMO los factores de coste se dividen en cuatro grupos: factores de producto (fiabilidad requerida, tamaño de la base de datos y complejidad del producto), factores de computación (restricciones de tiempo de ejecución, de almacenaje, inestabilidad de la máquina virtual y de tiempo de respuesta del equipo), factores de la plantilla (capacidad del analista, capacidad del programador, experiencia con la aplicación a desarrollar, con el lenguaje de desarrollo y la máquina virtual) y factores del proyecto (plan de desarrollo requerido, herramienta de desarrollo y la utilización de prácticas modernas de programación).

Este método basa la calibración de sus coeficientes en 63 proyectos finalizados, para los que consigue resultados moderados, aunque no se puede concluir que sea válido para todos los casos.

El modelo de COCOMO reflejaba las prácticas de desarrollo de software de la época de los 80. En la década y media siguiente estas técnicas cambiaron radicalmente, por lo que la aplicación del modelo original empezó a resultar problemática y se acabó por desarrollar un nuevo modelo: COCOMO II [2]. En lo referente a este estudio, los factores que afectan a la estimación de esfuerzo software, se produce un cambio entre el modelo viejo y el más reciente. Del COCOMO original se desecharon varios factores (VIRT, TURN, VEXP, LEXP y MODP) y se aportaron otros nuevos (DOCU, RUSE, PVOL, PEXP, LEXT, PCON y SITE). En este artículo se tratan los factores de ambos modelos y, por tanto, no se hace diferencia entre versiones.

Métodos de estimación algorítmicos analíticos Los métodos de estimación analíticos se basan en una comprensión del problema, descomponiéndolo para así comprender mejor su comportamiento. A a partir de aquí se pueden desarrollar ecuaciones matemáticas que modelen el problema de estimar el esfuerzo de desarrollo software. Este artículo expone brevemente dos métodos de estimación algorítmicos analíticos: SOFTCOST y SLIM.

SOFTCOST [3] El modelo SOFTCOST asume una relación lineal entre el esfuerzo y el tamaño de la aplicación mediante esta fórmula matemática:

$E = S/P_1$ con $P_1 = P_0 A_1 A_2$

Donde E es el esfuerzo, S es el tamaño estimado en KLOC y P_1 es la productividad media en KLOC/Personas-Mes. Para definir P_1 se utilizan varios factores. P_0 es una constante, A_1 engloba 6 factores de ajuste de productividad combinados mediante una fórmula matemática. A_2, al igual que A_1, es una combinación de otros 29 factores.

El método SOFTCOST engloba un total de 68 parámetros. La implementación del SOFTCOST es interactiva y recopila la información de una serie de 47 preguntas con las que deduce los valores de los factores. Como salida ofrece el esfuerzo y la duración del desarrollo software descompuesto en una estructura estándar. También ofrece una estimación del nivel de la plantilla, tamaño en páginas de la documentación y requerimientos del procesador.

SLIM Este método de estimación se apoya en el modelo teórico de Norden-Rayleigh sobre la forma de la curva de los desarrollos de los proyectos y se supone que es aplicable a proyectos con un tamaño superior a 70.000 líneas de código. El método de estimación SLIM se basa en el punto de vista de que la cantidad de trabajo asociado a cualquier producto se puede ver como una proporción del producto entre en esfuerzo realizado y tiempo necesario para conseguirlo:

$$Producto = Parmetro\,de\,Productividad \cdot \left(\frac{Esfuerzo}{B}\right)^{\frac{1}{3}} \cdot Tiempo^{\frac{4}{3}}$$

Donde *Producto* representa cierta medida sobre la funcionalidad del mismo y se suele medir en SLOC. El *Esfuerzo* viene medido en personas-año o personas-mes. El *Tiempo* representa la cantidad de tiempo empleada en el desarrollo y se mide en meses o años. El *Parámetro de Productividad* (PP) es una constante que engloba los factores del entorno. PP expresa la productividad de la compañía. Esta constante permite igualar las otras tres variables. De manera que la cantidad de *Producto* con el mismo *Esfuerzo* y *Tiempo* dedicado puede variar dependiendo del entorno de trabajo. El parámetro B representa la destreza en función del tamaño del sistema. Las potencias de la fórmula matemática corresponden a sucesivas validaciones. La forma que se muestra en este artículo la adquirió a mediados de os 80, cuando el método se validó con 750 sistemas.

Este método de estimación merece mención aparte en cuanto a los factores en los que se basa, que es el objetivo de análisis de este artículo. Aparte del tamaño de la aplicación a desarrollar, en este método se tienen en cuenta el conjunto de factores relacionados con el entorno (*Parámetro de Productividad - PP*) y la tasa de contratación de personal (*Manpower Build Parameter - MBP*), pero no de manera explícita o desglosada. Ambas variables se definen a partir de proyectos ya realizados. Dentro de un mismo proyecto ya finalizado, tanto *PP* como *MBP* se comportan como una constante. Así, una vez establecidos los parámetros de esfuerzo, tiempo y tamaño de proyectos anteriores, se pueden definir ambos parámetros. *PP* se comporta como un parámetro de proporcionalidad dentro la fórmula de SLIM. De esta manera se establece a grado de productividad asignada a la compañía en un momento dado. El parámetro *MBP* se establece de manera análoga.

El *Parámetro de Productividad* constituye una macromedida del entorno general de desarrollo. *PP* engloba el conjunto de factores que afectan a toda la organización, como la gestión del proyecto, la utilización de buenos requerimientos, diseños, codificaciones, inspecciones y pruebas. También incluye aspectos como el nivel del lenguaje de programación, el estado de la tecnología, la experiencia de los miembros del grupo y la complejidad de la aplicación. Es por esto que no se puede determinar qué valor o peso se asigna a cada uno de los aspectos que engloba. Se obtuvieron una serie de valores representativos de *PP* a partir de la base de datos QSM. A cada valor representativo de *PP* se le asignó un índice (*PI*). De esta manera, una compañía obtendrá un valor bajo de *PI* cuando trabaje en entornos elementales con herramientas inadecuadas, o cuando el producto tenga un alto grado de complejidad (como

microcódigo o firmware). Los valores altos de *PI* se asocian a compañías con un buen entorno y personal experimentado, o cuando se trate de productos de baja complejidad que se comprenden bien.

El parámetro *MBP* se obtiene por calibrado de proyectos anteriores. Representa el estilo de gestión de la empresa o la tasa de contratación de personal. Se calcularon seis valores representativos del parámetro *MBP* a partir de la base de datos QSM. A cada valor representativo se le asignó un índice (*MBI*). Estos índices representan desde una tasa de contratación lenta (*MBI* =1) a una tasa extremadamente rápida (*MBI* = 6). Con este parámetro se ha observado que acumular personal moderadamente reduce el esfuerzo de desarrollo, en comparación con hacerlo más intensamente. En otras palabras, se puede reducir el tiempo de desarrollo aumentando la tasa de contratación (*MBI*), y por consiguiente el esfuerzo, pero sólo se consigue con un gasto elevado en comparación con la reducción en el tiempo de desarrollo.

3. Factores de la aplicación a desarrollar

A la hora de estimar el esfuerzo empleado en el desarrollo de un software, hay que estimar el esfuerzo de una aplicación determinada, con una plantilla concreta y un método de desarrollo. Tanto la aplicación a desarrollar como la plantilla y el método de desarrollo empleado cuentan con unas características propias, denominados factores. Los atributos o factores relacionados con el producto software tienen gran importancia dentro de la ingeniería del software. Esta idea se suele resumir mediante el lema "Una buena estructura interna supone una buena calidad externa". Con lo cual, es evidente la importancia de estos factores en el desarrollo del software y el esfuerzo dedicado. [7]

En esta sección se exponen los factores más habituales en la estimación del esfuerzo software dentro del grupo de factores propios de la aplicación a desarrollar. Se da una visión de cada factor, con qué otros factores está relacionado y cómo es tratado por cada uno de los métodos de estimación expuestos en la sección anterior.

3.1. Tamaño (*Size*)

El factor de tamaño, o *size*, se suele medir en líneas de código (LOC), en miles de líneas de código (KLOC) o en líneas de código fuente (SLOC). También se puede medir mediante puntos de función [7] o teniendo en cuenta el espacio de almacenamiento (bytes o cualquiera de sus múltiplos).

Comúnmente se ha considerado el factor del tamaño como el más relacionado con el esfuerzo software [7], aunque contabilizar líneas de código no es un proceso claro. Entre el código que se desarrolla se pueden encontrar líneas en blanco o comentarios. Estas líneas no están relacionadas directamente con el esfuerzo de desarrollo, aunque sí influyen en la legibilidad del código, útiles a la hora de modificar una aplicación existente. Hay que tener en cuenta que no todas las líneas de código que se escriben, y que por tanto suponen un esfuerzo, llegan a ser entregadas y que la utilización de generadores automáticos (pantallas, formularios o interfaces de usuario) generan miles de líneas de código con un esfuerzo mucho menor. Ante una aplicación orientada a objetos, establecer el tamaño de la misma en líneas de código no apropiado. En consecuencia, se puede decir que el tamaño en relación al esfuerzo es significativo pero no determinante.

COCOMO basa su estimación del esfuerzo en la posibilidad de conocer el tamaño, medido en KLOC o puntos de función, del software a realizar. En la ecuación del modelo COCOMO se puede ver una proporción del esfuerzo software y el tamaño de este, ajustando con los coeficientes (a y b) y la función $m(X)$, que considera el peso, en relación al coste, del resto de atributos del proyecto software. Es decir, traslada la estimación del esfuerzo a la estimación

del tamaño. En consecuencia, el factor del tamaño del software adquiere un carácter mucho más determinante en la estimación del esfuerzo que el resto de factores.

El método de estimación SLIM también otorga gran relevancia al tamaño (denominado *producto* o *tamaño*) del software a la hora de estimar el esfuerzo. El tamaño del software en SLIM es proporcional al producto del la "productividad" por el esfuerzo y el tiempo. Se mide en líneas de código fuente (SLOC).

3.2. Lenguaje de desarrollo

Con este factor se tiene en cuenta el lenguaje que se utiliza a la hora de desarrollar la aplicación software. En la actualidad existen multitud de lenguajes. Cada lenguaje permite desarrollar un tipo de software de manera óptima dependiendo de su nivel de abstracción (desde lenguaje máquina a BASIC), según su forma de ejecución (compilados o interpretados) o el paradigma de programación donde podemos encontrar lenguajes imperativos (como C#, Java o Perl), lenguajes funcionales (como Lisp), lenguajes lógicos (Prolog) o lenguajes orientados a objetos (como Java, Visual Basic .Net, PHP o Ada entre otros). Unos son más adecuados que otros a la hora de realizar ciertas aplicaciones. Por ejemplo, para desarrollar una aplicación para un dispositivo móvil se puede utilizar un lenguaje de nivel bajo como ensamblador, pero el esfuerzo empleado sería muy alto. Por el contrario, si se utiliza un lenguaje de alto nivel u orientado a objetos como Java, parte del esfuerzo de desarrollo se emplearía en adaptar la funcionalidad de los diferentes paquetes a un dispositivo sin una capacidad de procesado elevada. En cambio, si se utiliza un lenguaje adecuado, como Java Mobile, el esfuerzo se minimiza ya que los paquetes de este lenguaje están preparados para este tipo de dispositivos. Aparte de este aspecto, el factor del lenguaje de desarrollo debe tener en cuenta la complejidad intrínseca de cada lenguaje (programar en un lenguaje de alto nivel es más sencillo e intuitivo que uno de nivel bajo).

Este factor lo tienen en cuenta varios métodos de estimación. El método de estimación SLIM lo denomina *NLENG* y lo incluye dentro del parámetro de Productividad calibrándolo con los proyectos ya realizados.

El método de B.W. Boehm (COCOMO) no tiene en cuenta el lenguaje de desarrollo, al menos directamente. En este método se considera que el lenguaje de desarrollo y la experiencia de los desarrolladores afectan de manera conjunta al esfuerzo. Cuanta más experiencia acumulan los desarrolladores, programan de manera más segura y con menor número de errores, reduciendo el esfuerzo.

3.3. Herramientas de desarrollo

Aparte de lo adecuado que sea el lenguaje de programación con la aplicación que se quiere desarrollar, la herramienta de desarrollo también juega un papel importante en el esfuerzo. Desarrollar un sistema complejo con una herramienta elemental, multiplica el esfuerzo dedicado o lo convierte en una tarea casi imposible. Por el contrario, desarrollar una aplicación muy simple con una herramienta muy compleja, aunque potente, requiere un esfuerzo extra dedicado al manejo de la herramienta. Por tanto, emplear la herramienta adecuada para cada tipo de aplicación implica mejorar la productividad y ajustar el esfuerzo de desarrollo. Cada herramienta de desarrollo lleva implícito un periodo de adaptación por parte del grupo de desarrolladores para aprender a utilizar dicha herramienta y sacarle todo el partido. En consecuencia, un cambio de herramienta en un proyecto conlleva un retraso y un aumento del esfuerzo.

Dentro de los factores que se tienen en cuenta en COCOMO, *TOOL* es el factor con el que se plasma el uso adecuado de las herramientas de desarrollo. Los valores de *TOOL* varían desde "muy bajo" cuando sólo se utilizan herramientas básicas, a "muy alto" cuando se utilizan herramientas específicas.

3.4. Fiabilidad del software

La fiabilidad del software se define como la probabilidad de ejecutar un software y que no cause ningún fallo en el sistema durante el tiempo especificado y bajo unas condiciones específicas. También se puede ver como la probabilidad de que exista un fallo en el software y sea ejecutado. Un fallo en el sistema a realizar puede suponer desde un problema de fácil solución y sin importancia, hasta la pérdida de vidas humanas (por ejemplo en el software de control aéreo).

Existen diversas técnicas para mejorar la fiabilidad del software. Con la prevención de fallos se intentan corregir los fallos durante la etapa de desarrollo. Una vez el producto software ha sido desarrollado se pueden descubrir sus posibles fallos con la aplicación de técnicas de detección de fallos. Con las técnicas de tolerancia a fallos el objetivo es proporcionar una respuesta controlada ante fallos no detectados.

Se puede ver que este factor está estrechamente ligado a la fase de pruebas dentro del desarrollo del software. Por tanto, cuanto mayor sea la necesidad de fiabilidad del software a desarrollar, más concisa será la fase de pruebas y, consecuentemente, el esfuerzo necesario para el desarrollo software se verá incrementado.

Este factor se tiene en cuenta en el método de estimación COCOMO. Se denomina *RELY* e indica las posibles consecuencias para el usuario en el caso de que existan defectos en el producto. El factor puede tomar cinco posibles valores, desde "muy bajo" a "muy alto". Un valor "muy bajo" indica que el efecto de un posible fallo sólo tendría como consecuencia el inconveniente de solucionarlo. Con un valor "bajo" las consecuencias supondrían una perdida fácilmente recuperable para los usuarios. Cuando la perdida para los usuarios es moderada y permite salvar la situación sin demasiada dificultad, *RELY* tomaría el valor "nominal". Ante un valor "alto" las consecuencias del fallo pueden suponer una perdida financiera o una molestia a un gran número de usuarios. El valor más extremo, "muy alto", supondría la perdida de vidas humanas o grandes perdidas financieras.

3.5. Almacenamiento del software

Las necesidades de almacenamiento requeridas por el software, principalmente el tamaño de la base de datos, pueden suponer esfuerzo extra en el desarrollo del software. Hoy en día el tamaño del software no suele ser un aspecto tan preocupante como lo fue a mediados de los 70. Actualmente el precio por megabyte de almacenaje se ha reducido considerablemente, además de que se han mejorado las técnicas de compresión. Aun así, ante un software que tenga una restricción de espacio (como pueden ser los sistemas para dispositivos móviles o empotrados), ajustar el tamaño a ese espacio puede suponer un incremento del esfuerzo importante.

Hoy en día el espacio necesario para un sistema o una base de datos no muy grande no supone un gran inconveniente. No ocurre lo mismo con las grandes bases de datos. Si el software a realizar necesita apoyarse en una base de datos de un tamaño importante (por ejemplo una aplicación bancaria), el esfuerzo se puede ver incrementado por la necesidad gestionar la infraestructura de almacenaje necesaria. Por norma general, en proyectos de tamaño medio, el espacio de almacenamiento tanto del código como de los datos, no suele suponer ningún esfuerzo extra.

COCOMO tiene en cuenta el aspecto del almacenamiento del software mediante dos factores: *DATA* y *STOR*. Con la medida *DATA* se captura cuanto se ve influenciado el desarrollo del producto software por unos requerimientos de almacenamiento elevados. Este factor es importante en cuanto al esfuerzo de generar datos que posteriormente se utilizaran para probar el sistema, introducir nuevos datos en la base de datos o adaptarlos. *DATA* viene definido por el coeficiente:

$\frac{D}{P} = \frac{TBD}{TP}$

Donde *TBD* representa el tamaño de la base de datos (la cantidad de datos a ser articulada y almacenada en memoria secundaria hasta la entrega del producto software) y *TP* el tamaño del código en SLOC. El valor del factor se discretiza en cuatro segmentos (0-10, 10-100, 100-1000. +1000) que determinan los valores de *DATA*, que son "bajo", "nominal", "alto" o " muy alto".

Parte del almacenamiento principal del software se utiliza para guardar el código. Si el software a realizar tiene como requisito la obligación de correr en un volumen menor al del almacenamiento principal, el esfuerzo de programación se verá incrementado. Este echo lo captura COCOMO mediante el factor *STOR*, asignándole un valor desde "nominal", cuando la reducción del almacenamiento principal es del 50 %, hasta "extremadamente alto", cuando la reducción es del 95 %.

3.6. Complejidad del software

A la hora de establecer el esfuerzo asociado a la realización o mejora de un determinado software, resulta esencial conocer la dificultad intrínseca al propio software. Evidentemente, abstrayéndose de las ventajas o dificultades que presente el lenguaje con el que se programe o la aplicación de desarrollo, una aplicación sencilla siempre tendrá asociado un esfuerzo de desarrollo menor que otra más compleja de realizar.

La complejidad o dificultad asociada a un software depende del dominio del problema (gran cantidad de requisitos incluso contradictorios, dificultad en la comunicación con el cliente), de los impedimentos en la gestión del proceso de desarrollo (manejar grandes cantidades de código, la necesidad de gestionar un gran equipo de desarrollo, gran esfuerzo en el análisis y diseño, gestionar un ciclo de vida largo, etc.), el nivel de detalle exigido o los límites de la capacidad humana. Por otro lado, la complejidad en el desarrollo de una aplicación se reduce si la plantilla asociada al proyecto cuenta con la experiencia suficiente en el desarrollo de aplicaciones similares.

El método de estimación COCOMO captura la complejidad del software mediante el factor: *CPLX*. Con *CPLX* se indica la complejidad de cada módulo y que, una vez determinada la complejidad de todos los módulos, se utiliza para determinar la complejidad compuesta del sistema. *CPLX* toma valores desde "bajo", cuando se trata de módulos compuestos por expresiones matemáticas simples, hasta "extremadamente alto" para módulos con muchos recursos de planificación. Con la segunda versión de este método, el factor *CPLX* se ha mantenido pero ha cambiado la escala de valores. La complejidad pasa a decidirse evaluando cinco tareas que caracterizan el software a desarrollar. Las tareas a evaluar son las operaciones de control, operaciones computacionales, operaciones dependientes de dispositivos, operaciones de gestión de datos y operaciones de gestión de interfaz de usuario. Cada una de estas tareas se valora de manera subjetiva mediante una escala de seis valores que van desde "muy bajo" a "extremadamente alto".

3.7. Factores de plataforma

En esta sección se exponen los factores que capturan el esfuerzo extra de ajustar la aplicación a desarrollar al entorno donde se va a implementar o ejecutar. El esfuerzo de desarrollo de una aplicación que se ejecute en modo local (sin acceso a la red) será menor que si ese mismo sistema accediese a la red. La misma diferencia se puede encontrar con el acceso a una base de datos así como la adaptación del software a uno o varios sistemas operativos. También influye el echo de tener que adaptar el sistema al hardware de la máquina, ya sea por estar limitada en cuanto a su capacidad de computo (por ejemplo un dispositivo móvil), como por la necesidad de gestionar un hardware específico (por ejemplo un dispositivo haptico para medicina). El esfuerzo de desarrollo también se puede ver afectado por los

requisitos de la máquina en la que se desarrolla el software: el tiempo de respuesta. Así, cuanto mayor sea el tiempo de respuesta, más alto será el esfuerzo humano en la fase de desarrollo. Un ejemplo que ilustra este aspecto puede verse en el desarrollo de aplicaciones de minería de datos. Si se va a ejecutar el software en una máquina con poca capacidad de computo, que tarde en mostrar los resultados, el esfuerzo del desarrollador será mucho mayor que si dispusiera de una máquina más potente que le permitiese visualizar los resultados de la ejecución más rápidamente.

En el método de estimación COCOMO se tienen en cuenta estos aspectos mediante los factores *TURN*, *VIRT*, *PVOL* y *TIME*. El factor *TURN* cuantifica el tiempo de respuesta del ordenador desde el punto de vista del programador. *TURN* puede variar desde "bajo" para un sistema interactivo, a "muy alto" cuando el tiempo medio de respuesta es de más de 12 horas. Este factor desaparece en la nueva versión de COCOMO II debido a que la mayoría de los desarrolladores tienen acceso casi inmediato a los recursos computacionales de su estación de trabajo. Aún así, este factor ha sido tenido en cuenta en este estudio dado que, si la aplicación con la se desarrolla el software tiene un tiempo de respuesta alto, el esfuerzo puede verse incrementado. Igualmente, si el propio sistema a desarrollar tiene un tiempo de respuesta significativamente largo, la fase de pruebas se verá afectada aumentado su duración. Por tanto, actualmente, este factor sólo es significativo para el esfuerzo en los casos en que el tiempo de respuesta sea notablemente alto.

TIME es el factor con el que cuantifica la restricción del tiempo de ejecución del sistema a desarrollar. Se expresan en términos de porcentaje de tiempo de ejecución disponible que se espera que sea utilizado por la aplicación. Este factor puede ser considerado como un factor complejidad, ya que se basa en el echo de que para un programador siempre será más exigente desarrollar una aplicación con una restricción en el tiempo de ejecución. Los valores van desde "nominal" (menos del 50 % de recursos de tiempo de ejecución utilizados) hasta "extra alto" (95 % del recurso de tiempo de ejecución consumido).

La volatilidad de la plataforma (hardware y software) en el método COCOMO se refleja mediante dos factores: *PVOL* y *VIRT*. Ambos hacen referencia a lo posibles cambios que se pueden producir tanto en la plataforma de desarrollo (*VIRT*) como en la que se ejecutará el sistema a desarrollar (*PVOL*). Los valores que pueden tomar van desde "bajo", donde cada 12 meses se produce un cambio importante, a "muy alto", donde se produce un cambio importante cada dos semanas. En la segunda versión del método el factor *VIRT* desaparece, con lo que no se tiene en cuenta la volatilidad de la plataforma de desarrollo.

3.8. Requerimientos no funcionales

Con este apartado se engloban aquellas características que se exigen a la aplicación a desarrollar, como la posibilidad de reutilización o una documentación exhaustiva.

El método de B. W. Boehm utiliza el driver de coste *RUSE* para contabilizar el esfuerzo extra necesario para el desarrollo de componentes pensados para ser reutilizados en otros proyectos. La mayoría de este esfuerzo se atribuye en la creación de un diseño del software más genérico, una documentación más detallada y pruebas más extensas, así se asegura que los componentes están listos para utilizarse en otras aplicaciones. *RUSE* puede tomar cinco valores que van desde "bajo" cuando el componente no se va a reutilizar, a "extremadamente alto" cuando se va a reutilizar a lo largo de múltiples líneas de producto. El nivel de documentación requerida, COCOMO lo analiza con el driver de coste *DOCU*. Este factor se evalúa en términos de adecuación de la documentación del proyecto a las necesidades del ciclo de vida del mismo. *DOCU* puede tomar valores desde "muy bajo" cuando la documentación deja muchas necesidades del ciclo de vida sin cubrir, hasta "muy alto" cuando la documentación resulta excesiva para las necesidades del ciclo de vida.

4. Conclusiones

Este artículo ha estudiado los factores más importantes relacionados con la "aplicación a desarrollar". Uno de los principales inconvenientes que se han presentado a la hora de hacer este estudio ha sido la falta de información en cuanto a este grupo de factores, su grado de influencia en el esfuerzo de desarrollo y las relaciones con el resto de factores.

Teniendo en cuenta el grado en el que estos factores inciden en el esfuerzo, consideramos que los que afectan de manera más directa al esfuerzo son la fiabilidad y la complejidad. Esto es debido a que ante unos valores altos de estos factores, el esfuerzo de desarrollo aumenta significativamente tanto en la fase de desarrollo como en las fases de análisis y pruebas. Estos dos factores pueden ver reducido sus efectos sobre el esfuerzo. Si la compañía desarrolladora cuenta con una gran experiencia en proyectos de similar dificultad y fiabilidad requerida, es lógico pensar que el esfuerzo de desarrollo se verá reducido aún manteniéndose alto.

En este sentido los factores del lenguaje y la herramienta de desarrollo así como los requisitos no funcionales, afectan al esfuerzo de igual manera, viendo reducida su relevancia sobre el esfuerzo si el equipo de desarrollo cuenta con experiencia en proyectos similares o con la herramienta y el lenguaje de desarrollo empleados. En este caso, con experiencia con la herramienta y el lenguaje de desarrollo, el esfuerzo tendría una relación más directa con el tamaño de la aplicación a desarrollar. Es por esto, que el factor del tamaño tiene una relación significativa con respecto al esfuerzo, pero no determinante. Si además se tiene en cuenta los problemas de medición del tamaño del software, tanto en líneas de código como con los puntos de función, y los resultados no muy buenos que se han obtenido por parte de los métodos de estimación basados en este factor (como COCOMO, SOFTCOST y SLIM), el factor del tamaño no debería de tener tanta importancia en relación al esfuerzo software.

Por último, los factores de plataforma y de almacenamiento, tanto del código como de la base de datos, son los que tienen una relación menos directa con el esfuerzo software. Hoy en día el problema de almacenamiento y la de las restricciones de computo no suponen un inconveniente que requiera de un esfuerzo extra. En casos extremos, como puede ser el desarrollo de una aplicación para un sistema móvil con restricciones de computo importantes y plataforma (pantalla reducida, teclado, etc.), el esfuerzo no se verá incrementado siempre que se utilicen las herramientas y el lenguaje de desarrollo apropiados para la plataforma. Ante un sistema que requiera una base de datos muy grande se debe tener en cuenta el factor del tamaño, ya que incide en el esfuerzo de manera significativa tanto en la gestión de la base de datos como en la introducción de datos para las pruebas.

En este artículo no se han comentado otros grupos de factores como los factores humanos o los relacionados con la metodología de desarrollo. Estos factores pueden tener la misma importancia que los factores comentados en este estudio y quedan como trabajo futuro.

Referencias

1. Barry W. Boehm. *Software Engineering Economics*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1981.
2. Barry W. Boehm, Chris Abts, A. Winsor Brown, Sunita Chulani, Bradford K. Clark, Ellis Horowitz, Ray Madachy, Donald J. Reifer, and Bert Steece. *Software Cost Estimation with Cocomo II*. Prentice Hall PTR, August 2000.
3. S. D. Conte, H. E. Dunsmore, and Y. E. Shen. *Software engineering metrics and models*. Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA, 1986.
4. J. J. Dolado. On the problem of the software cost function. *Information and Software Technology*, 43(1):61–72, 2001.
5. Stein Grimstad and Magne Jorgensen. Inconsistency of expert judgment-based estimates of software development effort. *Journal of Systems and Software*, 80(11):1770–1777, nov 2007.
6. Zhang Fan Hareton Leung. Software cost estimation.

7. Jose Javier Dolado Cosin Luis Fernandez Sanz. *Medicion para la Gestion en Ingenieria del Software*. feb 2000.
8. Emilia Mendes. The use of a bayesian network for web effort estimation. In *ICWE 2007, Web Engineering, 7th International Conference, Como, Italy, July 16-20, 2007, Proceedings*, volume 4607 of *Lecture Notes in Computer Science*, pages 90–104. Springer, 2007.
9. M. Shepperd, C. Schofield, and B. Kitchenham. Effort estimation using analogy. In *Software Engineering, 1996, Proceedings of the 18th International Conference on*, pages 170–178, 1996.
10. Claude E. Walston and Charles P. Felix. A method of programming measurement and estimation. *j-IBM-SYS-J*, 16(1):54–73, 1977. See letters \citeHalstead:1977:FLC,Walston:1977:FAR.