ISSN: 1698-2029 Volumen 6 Número 1

JULIO 2009

Revista de

Procesos y Métricas

de las tecnologías de la información

CONTENIDOS

AEMES asociación española de métricas del software

Revista de Procesos y Métricas de las Tecnologías de la Información

Volumen 6 Número 1

Revista fundada por la Asociación Española de Métricas del Software (AEMES)

<http://www.aemes.org>

Editores
Dr. D. José Carrillo Verdún, AEMES
Dr. D. Juan J. Cuadrado-Gallego, Universidad de Alcalá

Consejo Editorial
D. Jesús Campo Prieto, E-work-LINE
D. Ramiro Carballo, Gesein
D. José L. Lucero, IEE
D. Emilio del Moral, ALI
D. Luis Redondo López, MTP
Dña. Cecilia Rigoni, Asesora de AEMES
D. Edmundo Tovar Caro, UPM
D. Ángel Sánchez Díaz, EVERIS

Comité Científico
Dr. José Antonio Gutiérrez. UAH
Dr. Eladio Domínguez Murillo. UNIZAR
Dr. José Javier Martínez, UAH
Dr. José María Gutiérrez, UAH
Dr. Roberto Barchino Plata, UAH
Dr. Luis de Marcos Ortega, UAH
Dr. José Ramón Hilera. UAH
Dr. Edmundo Tovar Caro. UPM
Dr. José Antonio Calvo-Manzano. UPM
Dr. Tomás San Feliú. UPM
Dr. Oscar Pastor. UPV

ISSN: 1698-2029
Nº Depósito: M23879-2006

Maquetación
Luis de Marcos
Juan Ortega Pérez

Impresión
Imprenta UAH

Asociación Española de MÉtricas del Software (AEMES)

1. Propósito de la Sociedad

La Asociación Española de Métricas de Sistemas Informáticos (AEMES) es una asociación sin ánimo de lucro y con plena capacidad de obrar y personalidad jurídica y patrimonial y con un funcionamiento plenamente democrático, tal y como exige el artículo 7.1 g) de la ley orgánica 1/2002 de 22 de marzo.

La Asociación AEMES tiene por finalidad última contribuir a la difusión de los métodos y técnicas relacionados con la gestión cuantitativa de las Tecnologías de la Información y en particular con aquellos aspectos relacionados con la mejora del proceso de desarrollo del software y su gestión económica, en las empresas e instituciones que las desarrollan o utilizan, promoviendo el uso de indicadores, métricas y cuadro de mando en las mismas.

Con este fin dirige sus actuaciones a:

a) Promover, coordinar y desarrollar actividades relacionadas con las medidas de los procesos de las Tecnologías de la Información.

b) Favorecer el intercambio de información y experiencia entre los profesionales relacionados con este tema.

c) Relacionarse con otras organizaciones internacionales afines.

d) Difundir información al público en general sobre la gestión económica de las Tecnologías de la Información.

e) Crear comités y grupos de trabajo especializados en estos temas.

f) Canalizar la formación de especialistas en este campo, facilitando el acceso a titulaciones específicas y en concreto las promovidas por el International Function Points Users Group.

g) Canalizar las peticiones de certificaciones de puntos función y de otras métricas del software relativas a aplicaciones solicitadas por empresas.

h) Homologar y mantener un registro de profesionales, material docente y otros que puedan ser utilizados para los fines de la asociación.

2. Asociados

La asociación se compone de dos clases de asociados: Miembros de Número y Miembros Honoríficos.

Los Miembros de Número son las personas físicas o entidades que participan regularmente en las actividades de la asociación. La asociación está abierta, sin discriminación, a todas las personas físicas o entidades, sean estas organizaciones públicas o privadas, que están interesadas en promover los fines y actividades de la asociación.

Los miembros de Número ingresan en la asociación previa solicitud dirigida a la junta directiva (El formulario se encuentra las páginas finales de esta revista).

3. Beneficios de la Asociación

Los miembros de Número gozan de la plenitud de derecho en orden a participar en los Órganos de Gobierno de la asociación, tanto en la Asamblea General como en la Junta Directiva, siempre con sujeción a lo previsto en los estatutos y de acuerdo con las directrices y normas fijadas por la Junta Directiva.

Los miembros de Número tienen derecho a participar en las actividades y actos sociales en la forma, en que cada caso, disponga la Junta Directiva.

Los miembros de Número tienen derecho a recibir sin coste alguno las publicaciones periódicas realizadas por la asociación.

4. Publicaciones de la Asociación

Boletín de la Asociación Española de MÉtricas de Sistemas informáticos.
Publicación de periodicidad trimestral cuyo contenido está centrado fundamentalmente en la actividad interna de la asociación. Se describen también nuevos recursos como libros o herramientas software de interés para los asociados. También se comentan aquellos eventos de especial relevancia relacionados con la gestión de los Procesos de las Tecnologías de la Información y las Comunicaciones.

Revista de Procesos y Métricas de las Tecnologías de la Información.
Publicación de periodicidad cuatrimestral cuyo contenido está formado por artículos científicos revisados por pares y enfocados en las áreas de interés de la asociación.

APLICACIÓN DE UN ENFOQUE DE FACTORÍA DE SOFTWARE EN LA GENERACIÓN DE APLICACIONES A PARTIR DE BASES DE DATOS

José R. Hilera, José J. Martínez, José A. Gutiérrez, Luis Fernández, Deborah Rodríguez, Luis de Marcos, Carmen Pagés
Departamento de Ciencias de la Computación, Universidad de Alcalá, Alcalá de Henares (Madrid)
jose.hilera@uah.es

Resumen: Se presenta un caso real de aplicación del enfoque de factoría de software, a través del diseño y construcción de mecanismos de extensión para un entorno integrado de desarrollo de software (IDE), con funcionalidades de asistente para la producción sistemática de aplicaciones de acceso a bases de datos, que contengan una capa de interfaz gráfica de usuario (formularios de petición de datos), diferenciada de una segunda capa constituida por objetos generados de forma sistemática a partir de la base de datos que utilizará la aplicación, con un código robusto que incorpora una variedad de métodos de inserción, modificación, borrado, etc. El enfoque se ha aplicado con éxito en una organización de desarrollo para la producción de software comercial con Visual Studio, para la plataforma .NET y base de datos Postgres.

Palabras clave: Software factory, IDE extensible, Visual Studio.

1. INTRODUCCIÓN[1]

La evolución del desarrollo de software está ligada a la consecución de métodos y procesos de desarrollo que permitan lograr la eficiencia y efectividad alcanzada en la manufactura y fabricación de los productos industriales más maduros. De hecho, el concepto de Ingeniería de Software surgió del afán de aplicar el enfoque ingenieril exitoso en otros campos al desarrollo y mantenimiento de aplicaciones. De acuerdo con [1], la idea de industrialización de la producción de software se remonta al concepto acuñado de fábrica de software de R.W. Bemer, de General Electric, a finales de la década de los sesenta del siglo XX, y con una temprana puesta en práctica con la experiencia de Hitachi en esa época. Originalmente se contemplaba exclusivamente una combinación de herramientas automatizadas estandarizadas, con interfaz de usuario amigable y base histórica de proyectos para control de gestión: el principal propósito era reducir la variabilidad de productividad en los programadores. Posteriormente, en 1969, McIllroy añade la necesidad de que se potencie la reutilización y abarca distintos grados de aproximación al ideal de fábrica de software (*software factory*). El concepto de fábrica de software ha evolucionado adoptando propuestas de modelos específicas (pero con elementos comunes de proceso, herramientas y recursos, tecnologías y gestión) en las tres principales áreas tecnológicas, como son EE.UU., Europa y Japón [2]. Por supuesto, se pueden encontrar a lo largo de la historia dos enfoques según la amplitud del concepto adoptado [3]: por una parte los entornos productivos de Ingeniería de Software y por otra los enfoques completos de fabricación de software [4].

Actualmente, para el término de *software factory* se encuentran diversas acepciones; por una parte, la multinacional Microsoft ha incorporado el término en el contexto de soluciones para la plataforma .NET, y ha

[1] Artículo recibido el 20 de Julio de 2008

promocionado un esquema de construcción rápida de código mediante ensamblaje de componentes que ha tratado de arropar con un marco conceptual limitado. Así, en [5] se define una fábrica de software como un enfoque de producción de software que configura herramientas, lenguajes de modelado de dominio específico (*DSL: Domain Specific Lenguage*) y procesos para automatizar el desarrollo y mantenimiento de variantes de productos arquetípicos adaptando, ensamblando y configurando componentes.

Por otra parte, sobre todo en estos primeros años del siglo XXI, han proliferado múltiples iniciativas de fábricas de software consideradas como centros de trabajo, promovidos por compañías multinacionales, centrados en un tipo de desarrollo concreto y situados en lugares propicios al ahorro de costes (de mano de obra, de instalación, etc.); principalmente centrados en el diseño detallado, generación de código y prueba mediante equipos de desarrollo permanentes y habituados a unas mismas rutinas y procesos de trabajo. El objetivo evidente de estas experiencias es la eficiencia y el ahorro de costes. Dependiendo de su ubicación han dado lugar a la difusión de términos como *off-shore*, *near-shore*, *close-shore*, etc. Desde este punto de vista, definiciones como "entorno de alta productividad para la construcción de software, consiguiendo soluciones competitivas en coste y tiempo para el cliente, con unos resultados de calidad y fiabilidad elevados siguiendo estándares de producción de software", "una unidad de servicio donde cada grupo de especialistas asume la responsabilidad para una etapa especifica del ciclo de software: definición, especificación, desarrollo, prueba e implementación" [6], etc., se han aplicado a este tipo de implantaciones que se caracterizan por considerables niveles de organización, procesos maduros, disciplina de gestión y automatización mediante herramientas [7].

Es dentro de esta última tendencia de trabajo especializado centrado en la generación de código para desarrollo de productos especializados mediante la automatización y el uso de herramientas adaptadas a la problemática específica, donde se enmarca la solución presentada en este trabajo. En nuestro caso, se trata de adaptar una filosofía de fábrica de software a una PYME española con una veintena de desarrolladores, no vinculada a una multinacional, en la que se persigue la producción sistemática de aplicaciones con acceso a bases de datos. Para reforzar el esquema de trabajo inspirado en las *software factories* implantado en la compañía se precisa la creación de un proceso específico y, especialmente, de herramientas de ayuda integradas en los entornos tecnológicos utilizados y reforzadas por el uso de componentes, modelos y plantillas como se recomienda en [8][5].

Para la descripción de la experiencia llevada a cabo, en el siguiente apartado del artículo se presenta el modelo de ciclo de vida y los procesos de desarrollo propuestos para implementar el esquema de factoría de software; a continuación se describen las herramientas que se han diseñado e integrado en el entorno de desarrollo Visual Studio para dar soporte a cada uno de los proceso del ciclo de vida, y que han permitido implementar el enfoque propuesto en un caso real de producción de software, en una empresa dedicada al desarrollo de software con tecnología .Net. Finalmente se analiza el resultado de una encuesta realizada a los desarrolladores sobre la utilidad real de la solución creada.

2. DEFINICIÓN DE LOS PROCESOS DEL CICLO DE VIDA DE DESARROLLO

En la organización de desarrollo, para la aplicación de un esquema de factoría de software, se ha establecido una secuencia de cuatro procesos o fases a llevar a cabo en la generación de cualquier aplicación a partir

de una Base de Datos, como se muestra en la figura 1. La automatización de cada uno de ellos se ha realizado a través de componentes que se han integrado en el entorno de desarrollo Visual Studio, constituyendo cada uno de los elementos de la cadena de producción de software. Los procesos definidos y sus actividades asociadas, son los siguientes:

- **Proceso 1, Definición de la base de datos**: El objetivo de este primer proceso es la obtención de una descripción de la estructura de la base de datos que utilizará la aplicación que se va a desarrollar. Dicha descripción se representará en forma de archivo que podrá ser reutilizado en otras aplicaciones que vaya a utilizar la misma base de datos. El archivo de definición de la base de datos se puede crear nuevo o bien obtenerse de forma automática a partir de una base de datos real ya existente.

- **Proceso 2, Generación de componentes de datos:** Se deben generar las clases que hagan de intermediaras entre la aplicación y la base de datos. Estas clases tiene unos elementos básicos comunes a todas las aplicaciones a generar siguiendo la metodología propuesta, independientemente de la base de datos particular que utilice cada aplicación. Sobre estos elementos comunes se añadirán aquellos derivados de la descripción de la base de datos particular, creada en el proceso anterior. Se pueden crear diferentes clases intermedias a partir de una misma base de datos si la aplicación final así lo requiere. Cada una de estas clases se obtienen a partir de las tablas, columnas y relaciones de las tablas que se desee incluir.

- **Proceso 3, Generación de componentes de Interfaz de Usuario:** Se trata de crear componentes gráficos a partir de los componentes de datos generados en el proceso anterior. Existirá una serie de componentes básicos ya disponibles para todos los proyectos, pero el desarrollador puede crear otros más avanzados combinándolos. En este proceso también se pueden asociar diferentes estilos predefinidos a los componentes gráficos, con el objetivo de adaptarlos a la imagen corporativa actual del cliente para el que se desarrolla la aplicación, o a la imagen que pueda tener en el futuro.

- **Proceso 4, Composición de la aplicación:** El último proceso consiste en integrar en una aplicación los componentes generados, implementando mecanismos de apoyo que hagan este trabajo fácil e intuitivo, guiando al desarrollador paso a paso en la composición, hasta tener la aplicación terminada. El código fuente ha sido generado de forma automática hasta este punto del desarrollo, y una vez realizada la composición el programador puede completar la aplicación con código adicional. Una de las opciones que tiene el desarrollador en esta fase, en relación al aspecto final de la aplicación, es que puede cambiar y probar diferentes estilos globales de los componentes añadidos a la misma.

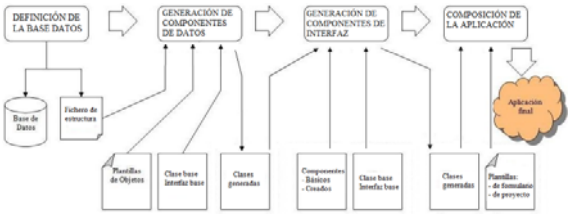

Figura 1. Procesos de la factoría de software

3. IMPLEMENTACIÓN DE LOS PROCESOS

La factoría de software se ha implementado en la plataforma .Net, concretamente en el entorno integrado de desarrollo Visual Studio [9], aprovechando las facilidades de extensibilidad que ofrece, a través de la creación de complementos (*Add-ins*) y asistentes (*Wizards*) que permiten apoyar y dirigir al

desarrollador en su trabajo a lo largo de los cuatro procesos del ciclo de vida propuesto. Las herramientas que se han creado para cada uno de los proceso son las que se describen a continuación.

3.1 Proceso 1: Definición de la base de datos

Este proceso se ha implementando a través de un complemento añadido al entorno Visual Studio, que se ha denominado *DB Designer* (figura 2), que permite al desarrollador crear, actualizar y comparar ficheros de descripción de estructura de base de datos, a los que llamamos archivos *DataBase* (con extensión *.dat*). Además, controla y registra información sobre el autor de las nuevas actualizaciones y las acciones llevadas a cabo (figura 3).

Figura 2. Integración del complemento *DB Designer* en Visual Studio.

Algunas de las funciones que ofrece el complemento son:

- Crear, modificar y comparar ficheros de estructura de BD
- Generar un fichero de estructura a partir de un BD existente
- Gestionar un repositorio local de ficheros de estructura del desarrollador.
- Actualizar un repositorio remoto corporativo, a partir de un fichero local.
- Ver historial de acciones realizadas por el desarrollador
- Conectarse a un servidor de base de datos (y abrir una base de datos)
- Diseñar gráficamente tablas, columnas, claves y relaciones
- Añadir observaciones a la base de datos.
- Generar un script de creación de base de datos

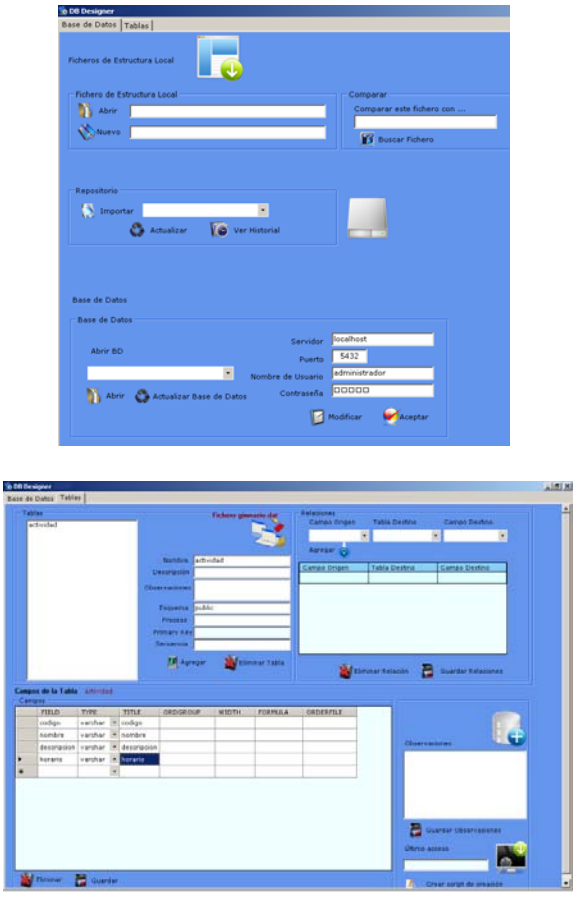

Figura 3. Complemento para la definición de una Base de Datos.

Un fichero de estructura gestionado por esta herramienta se genera a partir de un objeto de una clase, que denominamos *DataBase*, programada con tecnología .NET, que contiene la información de descripción de la estructura de la Base de Datos y la de conexión y acceso (figura 4).

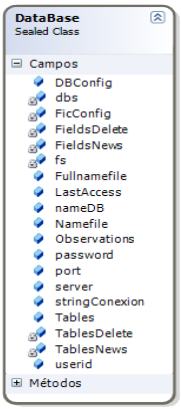

Figura 4. Clase a partir de la que se genera un fichero de estructura.

Si una misma base de datos se utiliza en varias aplicaciones, este archivo se reutilizará en el desarrollo de las todas ellas, sin necesidad de tener que crearlo en cada caso, registrándolo en un repositorio corporativo compartido por todos los desarrolladores, y que deberá estar siempre actualizado, recogiendo la última versión de la estructura de la base de datos que describe.

3.2 Proceso 2: Generación de componentes de datos

La implementación de este proceso se ha llevado a cabo a través de la integración de un asistente en el entorno de desarrollo Visual Studio, al que se ha denominado *DbRegWiz*, que se ofrece al desarrollador en forma de una nueva plantilla llamada *DbReg*. Cuando un desarrollador va a realizar una nueva aplicación, debe crear en este punto un nuevo proyecto de programación, y empezar agregando al proyecto un nuevo elemento de tipo *DbReg*. Al igual que el entorno de desarrollo le permite añadir a un proyecto clases, formularios, archivos XML, diagramas de clases, etc.; en este caso, el primer elemento que todo desarrollador que vaya a producir una aplicación con acceso a una base de datos debe agregar a su proyecto debe ser de este tipo, lo que desencadena la ejecución del asistente *DbRegWiz*, cuyo objetivo es generar de forma automática, a partir de un archivo de definición de estructura de la base de datos, las clases del programa para el acceso a los datos.

DbRegWiz es un asistente que consta de una única ventana (ver figura 5), en la que el desarrollador selecciona un archivo de definición de estructura de una base de datos y, a partir de la base de datos elegida se genera de forma automática el código fuente (en C#) de una nueva clase para el proyecto, derivada de una clase abstracta (*PkDbReg*) que, a su vez, implementa una interfaz básica (*IReg*), ambas previamente implantadas en la factoría en forma de librería de clases. El asistente solicita un nombre para la clase, y una vez finalizado, la herramienta la generará a partir de una plantilla y de la información seleccionada (tablas y columnas a los que se quiere acceder). En la figura 6, se muestra un ejemplo de una clase denominada *propietario*, generada a partir de la selección de cinco columnas de una tabla PROPIETARIOS incluida en una base de datos.

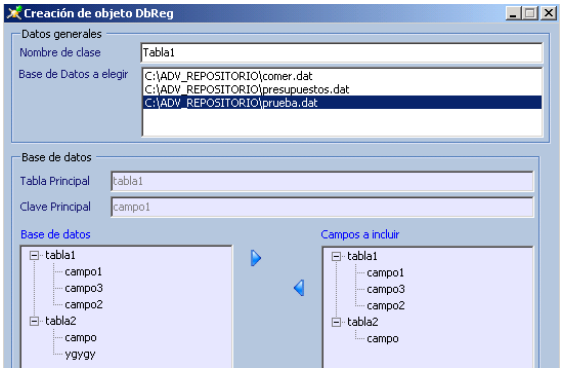

Figura 5. Asistente de generación de componentes de datos.

La ventaja de utilizar este asistente en todos los proyectos es evidente: por una parte se consigue que el código fuente que se genera en todos ellos sea estándar, y muy robusto, ya que ha sido suficientemente probado. Además facilita el mantenimiento de las

aplicaciones, ya que todas tienen una estructura y código fuente común. Y por otra parte, el programador no tiene que escribir código, evitando los consiguientes errores.

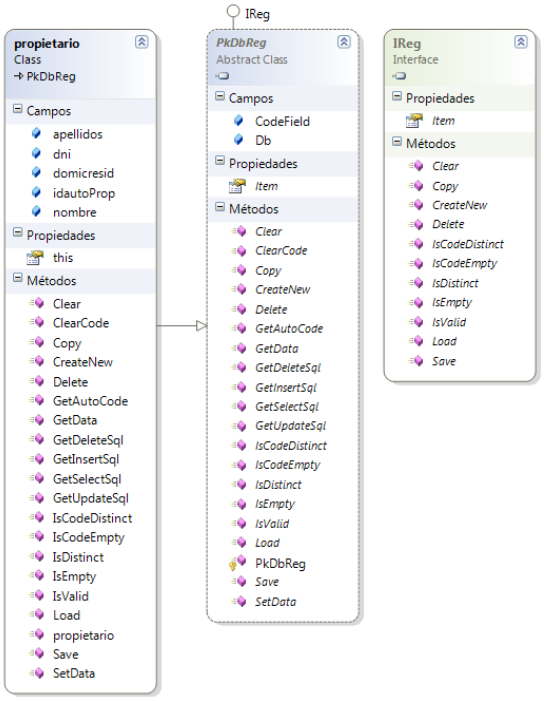

Figura 6. Ejemplo de clase generada de forma automática.

3.3 Proceso 3: Generación de componentes de interfaz de usuario

Una vez generadas las clases de acceso a la base de datos, el siguiente proceso consiste en preparar los componentes de la interfaz de usuario desde los que se visualizarán o manipularán los datos. Para implementar un mecanismo que ayude al desarrollador en esta fase, es necesario ofrecer una herramienta que permita asociar los componentes gráficos a los datos, y otra que permita asociar a estos componentes estilos comunes adaptados a la imagen corporativa de la empresa cliente.
Para ello, se han creado dos herramientas diferentes:

- Editor de estilos (*ToolEditor*): Se ha implementado como un complemento de Visual Studio para ser mostrado en la barra de herramientas de este entorno cuando el desarrollador esté creando una aplicación con interfaz gráfica, para que la asignación de estilos a los elementos gráficos simplemente consista en seleccionar dicho elemento y elegir en la barra de herramientas el estilo a asignarle. La ventaja de incorporar esta herramienta en la factoría consiste básicamente en ofrecer a los desarrolladores un número limitado de estilos que puedan aplicar, garantizando que todas las aplicaciones y las ventanas de una misma aplicación tendrán un aspecto similar.

- Diseñador de componentes gráficos de datos (*EdMaker*): Se ha implementado como un asistente. Cuando un desarrollador se encuentre en esta fase y tenga que diseñar los componentes gráficos de datos, debe agregar al proyecto un nuevo elemento de tipo *EdMaker*, lo que desencadena la ejecución del asistente, cuyo objetivo es obtener de forma automática, a partir de las clases *DbReg* generadas en la fase anterior, un nuevo componente gráfico (o *Control*, en la terminología de la plataforma .NET), para los datos procesados por esas clases. El asistente permite ir seleccionando los elementos de datos que se convertirán en elementos gráficos (cajas de texto, cajas de lista, etc.) en el interior del componente a generar (figura 7). Los elementos gráficos son de carácter más avanzado a los equivalentes que ofrece Visual Studio, porque se basan en ellos pero los extienden con propiedades y métodos para poder aplicarles los estilos comentados anteriormente.

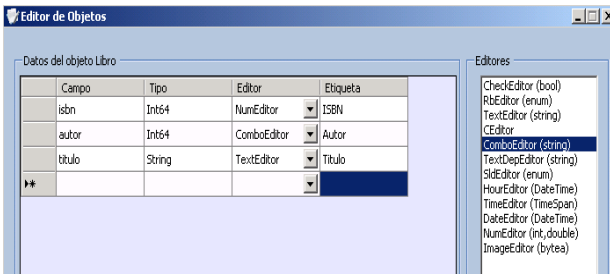

Figura 7. Diseñador de componentes gráficos de datos

El resultado final de este tercer proceso es, por cada componente gráfico de datos generado, un objeto contenedor, con cada uno de los objetos que representan a cada uno de los elementos gráficos del componente, además de una serie de botones para permitir la modificación de los datos. Por ejemplo, en la figura 8, se muestra (a la izquierda) un componente gráfico con cinco cajas de texto asociadas a cinco campos de una base de datos, y tres botones para insertar y borrar datos, y para cancelar la operación. En la misma figura (a la derecha) se representa el diagrama de clases del componente de ejemplo; puede observarse que, tanto el componente contenedor (en el ejemplo, clase *CEditor*), como los objetos que contiene (en el ejemplo, de la clase *TextEditor*), implementan una interfaz denominada *IEditor*, que permite que a todos ellos se les pueda aplicar los estilos ofrecidos en tiempo de diseño, por el editor de estilos.

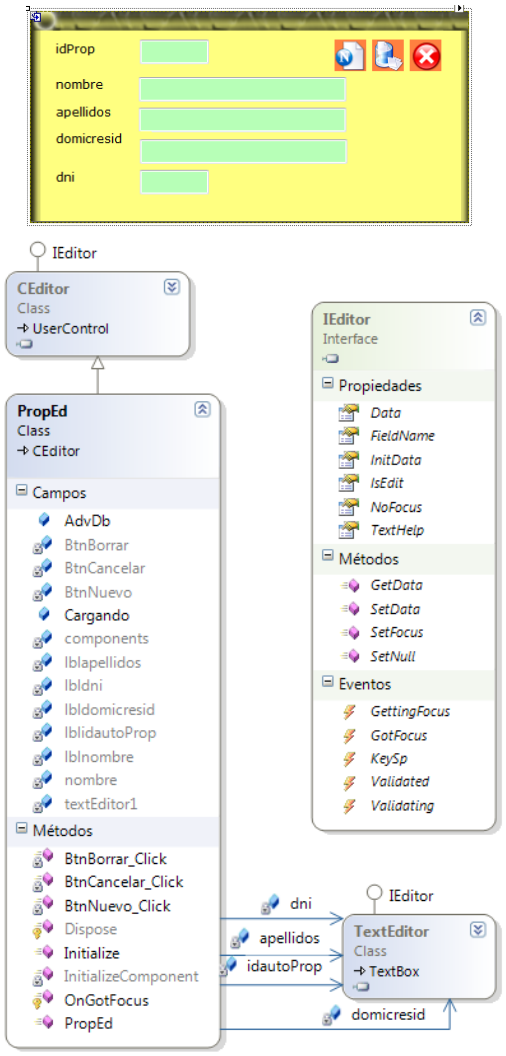

Figura 8. Ejemplo de componente gráfico de acceso a datos

3.2 Proceso 4: Composición de la aplicación

Para la composición de la aplicación final, se deben diseñar y construir las ventanas o formularios que la forman. El desarrollador deberá añadir a las ventanas correspondientes los componentes gráficos de acceso a datos generados en la fase anterior. Si se han seguido los procesos previos establecidos, todavía no ha sido necesario escribir ninguna línea de código fuente, y se ha obtenido una

aplicación lista para su compilación y ejecución contra una base de datos.
Por supuesto, el desarrollador puede ampliar y añadir nuevos elementos a la aplicación. En el caso de ser elementos gráficos, se recomienda utilizar un repertorio de elementos editables que se han integrado en el cuadro de herramientas del entorno de desarrollo (figura 9), cuyo comportamiento en similar a los que ya existen en Visual Studio, pero con la posibilidad de asignarles estilos. Por ello durante este proceso, también se puede utilizar el complemento de edición de estilos (*ToolEditor*) descrito en el apartado anterior.

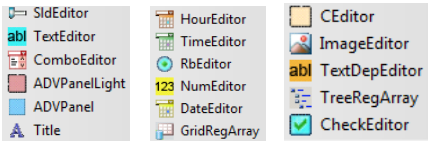

Figura 9. Elementos de interfaz gráfica de usuario, que soportan “estilos”

4. APLICACIÓN EN UNA ORGANIZACIÓN DE DESARROLLO

La solución presentada en este trabajo es fruto de la colaboración de la Universidad de Alcalá con la empresa de desarrollo de software SOLUCIONES INFORMÁTICAS ADV S.A., interesada en aplicar un enfoque de factoría de software para la generación de código a una importante parte de sus desarrollos comerciales de software de gestión. El equipo de investigación y desarrollo de la UAH realizó un proceso de reuniones y elaboración de propuestas junto a los responsables de la empresa para fijar el esquema de ciclo de vida de generación de código presentado en el apartado 2. Dadas las características del software a desarrollar y el entorno de la empresa, se procedió a determinar el rango más reducido posible de actividades a involucrar en la automatización que hemos presentado. ADV es una PYME dedicada al desarrollo de software desde hace 20 años, (www.siadv.com) que gestiona más de medio millón de líneas de código fuente propias, y que recientemente se ha instalado en el parque científico-tecnológico de la Universidad de Alcalá.
Como evaluación de los resultados tangibles en la empresa se ha procedido a analizar el impacto favorable de la solución sobre el personal al cargo de su uso en dicha empresa mediante cuestionarios anónimos. La muestra incluye a los programadores y los jefes de proyecto/desarrollo (con rangos de experiencia desde los 2 hasta los 23 años); aunque se trata aún de datos limitados, puesto que falta por completarse la extensión de la herramienta a toda la organización. El resumen de los resultados obtenidos es el siguiente:

- El 100% de los desarrolladores indica que reduce el esfuerzo de generación de código: en promedio se estima en un 53% dicha reducción.
- Un 50% considera que mejora mucho la fiabilidad del código, mientras que el resto reduce esa percepción a un incremento ligero de la misma. Por otra parte, el 70% considera que existe un gran incremento en la facilidad de mantenimiento del código frente al resto que sugiere una ligera mejora en este aspecto.
- La mitad de los encuestados informa de la necesidad de dedicar un gran esfuerzo a la adaptación y el aprendizaje del entorno, mientras que la otra mitad indica un esfuerzo medio. No obstante, el 100% de los encuestados indica que compensa el esfuerzo de adaptación, evaluado en algunos casos en hasta 5 días y medio de dedicación.

Aunque se trata de datos iniciales, consideramos que los beneficios de la solución planteada han sido confirmados por los desarrolladores al integrarla en su rutina diaria.

5. CONCLUSIONES

En este trabajo se ha aplicado el enfoque de factoría de software mediante la creación de herramientas que automatizan la creación de artefactos software que se integran en aplicaciones .Net con acceso a bases de datos. El valor fundamental de estas herramientas es que han quedado integradas de forma natural en Visual Studio, el entorno que utilizan habitualmente los desarrolladores que trabajan con tecnología .Net. Este entorno ofrece mecanismos de extensión que han facilitado la integración. Aunque el proyecto se ha realizado en este entorno, la idea general de la propuesta es aplicable en cualquier otro contexto (Java, Web, etc.), y la dificultad de implementar los procesos del ciclo de vida de desarrollo establecido dependerá de las facilidades de extensión que ofrezcan los entornos de desarrollo.

El principal objetivo que se ha perseguido con este trabajo ha sido ofrecer a una organización que desarrolle de manera habitual de aplicaciones con acceso a bases de datos, un método sistemático para obtener en todas ellas un código fuente robusto de fácil mantenimiento, al ser común en todas las aplicaciones, y reduciendo al mínimo el código que deba ser escrito por los programadores. El método propuesto y las herramientas desarrolladas se han aplicado con éxito en el desarrollo de aplicaciones comerciales por parte de una organización de desarrollo.

6. AGRADECIMIENTOS

Este trabajo ha sido financiado a través del proyecto de investigación UAH 50/2007: “Estudio e implantación del enfoque de Factoría del Software en un entorno industrial de Ingeniería del Software basada en componentes y servicios reutilizables”, realizado por la empresa SOLUCIONES INFORMÁTICAS ADV S.A. y la Universidad de Alcalá.

7. REFERENCIAS

[1] Chao L., Huaizhang L., Mingshu L.: A software factory model based on ISO9000 and CMM for Chinese small organizations. In: Asia-Pacific Conference on Quality Software 2001-, pp. 288-292. IEEE Press, New York (2001)

[2] Lim, N.K., Ang, J.S.K., Pavri. F.N.: Diffusing software-based innovation with a software factory approach for software development. In: Proceedings of the 2000 IEEE International Conference on Management of Innovation and Technology - ICMIT 2000, pp. 549-555. IEEE Press, New York (2000)

[3] Cantone, G.: Software factory: modeling the improvement. In: Third International Conference on Factory 2000 Competitive Performance Through Advanced Technology (Conf. Publ. No. 359), pp. 549-555. IEEE Press, New York (1992)

[4] Matsumoto, Y.: A Software Factory: An Overall Approach to Software Production in: P. Freeman (Ed.), Tutorial: Software Reusability, Computer Society Press, Washington, DC, pp. 155-178 (1987)

[5] Greenfield, J., Short, K.: Software factories. Assembling applications with patterns, models, frameworks and tools. Wiley, Indianapolis (2004)

[6] Escribano, C.: 12 años de fábrica de software: lecciones aprendidas, In: First Software Factories International Conference, www.calidaddelsoftware.com (2007)

[7] Rodríguez, A.: Software Factory: desarrollo de calidad para procesos de desarrollo a medida , In: First Software Factories International Conference, www.calidaddelsoftware.com (2007)

[8] Fernstrom, C.; Narfelt, K.-H.; Ohlsson, L.: Software factory principles, architecture, and experiments, IEEE Software, 9-2, 36 – 44 (1992)

[9] Microsoft: Visual Studio Developer Center. msdn.microsoft.com/es-es/vstudio/ (2008)

MÉTODOS PÚBLICOS DE ESTIMACIÓN DE ESFUERZO PARA PROYECTOS SOFTWARE

Borja Martín-Herrera, Pablo Rodríguez-Soria, José-Luis Cuadrado, Miguel-Ángel Herranz
CuBIT, Laboratorio de Medición del Software, Despacho N335, Tel.918856956
Departamento de Ciencias de la Computación, Universidad de Alcalá de Henares
borja.martin@uah.es, pablo.rsoria@uah.es

Abstract: Este artículo presenta una revisión de los métodos de estimación de proyectos software que han sido desarrollados a lo largo de la historia de la ingeniería del software, centrados en los modelos matemáticos de estimación de esfuerzo No-Lineales. Estos métodos y modelos han sido clasificados partiendo de un nuevo criterio introducido y están todos ellos basados específicamente en modelos públicos. Para cada modelo se especifican sus principales características, elementos y ecuaciones que nos permitan mostrar en su conjunto el funcionamiento y las herramientas de cada uno de estos métodos de estimación de esfuerzo para proyectos software.

Palabras Clave: Ingeniería del Software, estimación de esfuerzo, Tamaño del proyecto software, proceso de planificación de proyectos software.

1. INTRODUCCIÓN

Desde los primeros desarrollos informáticos hasta los actuales un problema fundamental ha sido el cumplimiento de unos plazos de entrega dentro de unos costes establecidos, así como el poder realizar un seguimiento y control de la evolución de los proyectos. Por lo que el establecimiento de unos métodos que permitiesen obtener estos objetivos de una forma lo más realista y exacta posible ha sido un factor cada vez más importante para la Ingeniería Informática en su conjunto y dichos métodos se han fundamentado en conocimientos adquiridos por distintas disciplinas de esta ciencia, desde la ingeniería del software hasta la ingeniería artificial.

Además de producir cada vez mejores resultados en los objetivos originales, la continua evolución que los métodos de estimación han experimentado, ha permitido también obtener otros beneficios como el perfeccionamiento de los análisis de riesgos de los proyectos o la posibilidad de realizar análisis cuantitativos sobre la eficacia de distintas propuestas de cambio de los procesos de construcción de software.

Desde los años sesenta hasta hoy en día se han publicado un gran número de modelos y se han propuesto también distintas clasificaciones de los mismos en base a distintos criterios. Una de las más conocidas y referenciadas en la literatura es la propuesta por Conte, Dunsmore y Shen [6] que comprende cuatro tipos de modelos:

1. Históricos / Experimentales

Se refiere a aquellos modelos basados en un conjunto de ecuaciones propuestas por un experto.

2. Estadísticos

Agrupa a los modelos que utilizan un análisis de regresión para establecer la relación entre el esfuerzo y los conductores de costes. Se diferencian dos tipos en función de las ecuaciones utilizadas:

a. Lineales. Los algoritmos son ecuaciones lineales.

b. No Lineales. Los algoritmos son ecuaciones no lineales.

3. Teóricos

Están basados en teorías sobre cómo los seres humanos se comunican entre si, sobre cómo funciona la mente humana durante el proceso de programación o sobre las leyes matemáticas que sigue el proceso de construcción de un producto software.

4. Compuestos

Incorporan una combinación de ecuaciones analíticas, ajuste estadístico de datos e ideas de expertos.

El resto del artículo está compuesto por los siguientes apartados; La Sección 2 muestra una nueva clasificación propuesta para los modelos de estimación de esfuerzo siguiendo las líneas de modelos previos. La Sección 3 describe los modelos matemáticos públicos y No-Lineales de estimación de esfuerzo los cuales vamos a analizar en orden cronológico. Finalmente, las conclusiones y trabajos futuros son dadas en la Sección 4.

2. NUEVA CLASIFICACIÓN PROPUESTA

Aquí vamos a proponer una nueva clasificación de los modelos de estimación [7], que aunque toma algunas ideas de las propuestas anteriormente, presenta un nuevo enfoque según el cual dichos métodos pueden ser estructurados en cuatro grupos:

1. Modelos Matemáticos Paramétricos

Se incluirían en este grupo todos aquellos modelos que utilicen ecuaciones matemáticas para realizar las estimaciones. No se introduce la diferencia de si han sido generadas por expertos porque se supone que la intervención de expertos es imprescindible en el desarrollo de la ecuación. Tampoco se diferencia entre modelos tabulares, multiplicativos, lineales y no lineales. Lo motivos son, que por una parte las ecuaciones multiplicativas y lineales se consideran una caso particular de las no lineales y por otra los modelos tabulares o los que utilizaban ecuaciones lineales están ya en desuso y todos los nuevos métodos publicados utilizan ecuaciones no lineales.

Solamente se introduce la diferenciación entre Modelos Públicos y Modelos Propietarios por considerarla muy importante, ya que sobre los primeros será posible un trabajo de investigación y estudio, así como una adecuación a un entorno local por parte de los investigadores o de los usuarios; y en el caso de los segundos solo será posible conocer de forma general su funcionamiento o método de empleo y procedimiento para adquirirlos o alquilarlos.

2. Estimación basada en la Experiencia de Expertos

Los modelos clasificados dentro de este método son aquellos que proporcionan una manera de extraer de la forma más objetiva posible los conocimientos de expertos para realizar las estimaciones. Normalmente lo hacen a través de un conjunto de pasos o mediante la utilización de cuestionarios.

3. Técnicas Orientadas al Aprendizaje

Este método se basa en el empleo de datos recogidos en proyectos anteriores para realizar las estimaciones y en este sentido es análogo a los Modelos Matemáticos Paramétricos, los cuales utilizan también los datos de proyectos anteriores para determinar sus parámetros. Pero la diferencia que presentan los modelos clasificados en esta metodología, aparte de no estar fundamentados en ecuaciones matemáticas, es que no pretenden ser de aplicación universal sino que su objetivo es realizar estimaciones precisas para aquellos proyectos del mismo tipo de los que se han empleado para aprender.

4. Modelos Dinámicos

Estos modelos asumen que los factores de costes de un proyecto software cambian a lo largo del tiempo y realizan estimaciones y simulaciones de dichas variaciones. Por lo tanto el fundamen-

to de este método es muy diferente al de los otros tres, los cuales tienen una visión estática del sistema.
Esta revisión se va a centrar especialmente en los modelos matemáticos paramétricos de estimación libres, debido a la amplia difusión de su utilización y su importancia a la hora del análisis de sus modelos.

3. MODELOS MATEMÁTICOS PARAMÉTRICOS

Los modelos matemáticos paramétricos se fundamentan en el desarrollo de unas ecuaciones matemáticas que permitan, de la forma más universal posible, obtener, a partir de la introducción de los valores de un conjunto de variables independientes como el tamaño del producto software a desarrollar o la experiencia de los usuarios; el valor de otro conjunto de variables dependientes, como el esfuerzo o el tiempo necesarios para desarrollar dicho producto. Estas ecuaciones, que comenzaron siendo lineales en los primeros modelos que se desarrollaron y que en los modelos más actuales son no lineales, son ajustadas a través de un conjunto de parámetros, que, a su vez son los que determinan el ámbito de aplicación del modelo. El objetivo último de todos los modelos está en obtener estimaciones precisas utilizando los parámetros genéricos suministrados por el propio modelo, pero la experiencia ha demostrado que para obtener estimaciones precisas éstos deben ajustarse para el entorno de aplicación del modelo; y esto es válido, en general, para todos los métodos de estimación.

El desarrollo de los modelos matemáticos paramétricos comenzó a mediados de los años 60 y desde entonces se han desarrollado gran cantidad de modelos hasta hoy en día. Esta evolución en los modelos ha ido ligada al progreso que se ha producido en el conjunto de la informática, con el cambio constante que ha tenido lugar en el hardware y el software y sobre todo en la manera de producir este último.

Los modelos matemáticos paramétricos pueden clasificarse en públicos y propietarios. Los primeros modelos que se produjeron solían ser públicos ya que sus ecuaciones, parámetros, etc., eran de libre difusión. Pero a medida que este campo tomó importancia, fue generando intereses comerciales, de tal forma que entre los modelos más utilizados en los últimos tiempos, una gran parte son propietarios o patentados por las compañías que los comercializan. Aunque estos modelos patentados actúan como una caja negra y solamente se conocen sus resultados, la mayor parte de ellos suelen ser el resultado de la evolución de algún modelo libre previo por lo que es de suponer que sus bases fundamentales sean las ecuaciones matemáticas en las que estaba basado el modelo original, por esta razón se clasifican en este apartado.

Como ya se ha comentado anteriormente, en este artículo vamos a centrar nuestro análisis únicamente en los modelos públicos ya que resultan más idóneos para su estudio y la mayoría de los modelos privados o patentados están basados en éstos, todos ellos en orden cronológico de desarrollo.

3.1 GRC

GCR procede de General Research Corporation, la empresa que lo desarrolló. Su primera publicación se remonta al año 1974 por Taback [11].

Este modelo calcula el costo del sistema en dólares directamente a partir del número de instrucciones, sin hacer un cálculo previo del esfuerzo.
Está basado en la siguiente ecuación para su funcionamiento:

Esfuerzo: $c = 0.232(s)^{1.43}$

Donde *c* representa el coste medido en dólares, *s* el tamaño del sistema en SLOC[2].

[2] SLOC – *Source Lines of Code* - Líneas Código Fuente

3.2 Doty

Doty procede de Doty Associates, la empresa contratista en la que se llevó a cabo el estudio encargado por el *Rome Air Development Center* (RADC) del *Data and Analysis Center for Software* (DACS) del *Department of Defense* (DoD) de los Estados Unidos de América, que permitió el desarrollo de este modelo.

La primera publicación fue realizada por Herz en el año 1997 [8].
Este modelo estima el esfuerzo, costes y tiempo de desarrollo para un proyecto software determinado. Para elegir los valores de las constantes de la ecuación de cálculo, el modelo de Doty distingue cuatro tipos de programas o módulos de programas:

1. On-line o Interactivo
2. Tiempo real o control
3. Cálculo
4. Sistemas operativos

El tamaño del sistema propuesto se estima por comparación con sistemas análogos. Una de las conclusiones introducidas por este modelo es la de que el tiempo que se requiere para escribir un número dado de instrucciones en un lenguaje de alto nivel es el mismo que para escribirlas en un lenguaje ensamblador, pero como el lenguaje de alto nivel es más pequeño, normalmente en una relación de 3 a 7, el resultado es que se incrementa la productividad. Además de que el lenguaje de alto nivel es superior en claridad y por lo tanto facilita el mantenimiento.
Este modelo está basado en las siguientes ecuaciones de esfuerzo:

a. *On-line:* $e = 4.495(s)^{1.068}$
b. *Tiempo Real:* $e = 4.573(s)^{1.228}$
c. *Cálculo:* $e = 2.895(s)^{0.784}$
d. *Sistemas Operativos:* $e = 12.039(s)^{0.719}$

Donde *e* representa el Esfuerzo medido en MM[3] y *s* el tamaño en SLOC.

[3] MM – *Men per Month* – Hombres al mes

3.3 Aerospace

El nombre oficial para este modelo es Modelo Aerospace. Su primera publicación consta del año 1997 y fue realizada por James [9].
Este modelo estima de forma separada el esfuerzo para los programas de tiempo real y el resto. Se basa en las siguientes ecuaciones:

a. *Tiempo Real:* $e = 0.057\ (s)^{0.94}$
b. Resto de programas: $e = 2.012\ (s)^{0.404}$

Donde *e* representa el Esfuerzo medido en MM (Man Months – Hombres Mes; De aquí en adelante MM) y *s* el tamaño en SLOC (Source Lines of Code – Líneas de Código Fuente; De aquí en adelante SLOC).

3.4 IBM – FSD – Waltson-Felix

IBM-FSD procede de International Bussines Machines Federal Systems Division, la empresa en la que se llevó a cabo el estudio que permitió el desarrollo de este modelo. También conocido como Modelo de Waltson-Felix.
Su primera publicación fue realizada gracias a Waltson y Felix en el año 1997 [13].
Aunque este modelo determina el esfuerzo en función de las líneas de código producidas; del análisis de la base de datos utilizada por los autores se proponen otra serie de relaciones: productividad frente a porcentaje de nuevo código desarrollado, documentación desarrollada frente a código desarrollado, duración frente a código desarrollado, duración frente a esfuerzo, equipo de desarrollo frente a esfuerzo, costo frente a código desarrollado, o costo frente a esfuerzo.
Este modelo consta de las siguientes ecuaciones de esfuerzo y tiempo de desarrollo:

1. Esfuerzo: $e = 5.2(s)^{0.91}$

Donde *e* representa el esfuerzo en MM, *s* el tamaño final en miles de SLOC.

2. Tiempo de desarrollo: $t = 2.47(e)^{0.35}$

Donde *t* representa el tiempo de desarrollo en meses y *e* el esfuerzo medido en MM.

3.5 Bailey-Basili

El nombre oficial de este modelo es Modelo de Baylei-Basili. Su primera publicación se remonta al año 1982 realizada por Bailey y Baisili [2].
Más que un modelo de estimación en si mismo, Bailey y Basili presentan un método de construcción de un modelo local de estimación. El proceso de generación del modelo consiste en tres pasos:

1. Calcular el Esfuerzo a partir de la ecuación inicial proporcionada por el modelo.

2. Determinar qué conjunto de factores diferencian al proyecto que se ha estimado y que podrían explicar las variaciones entre los datos medidos en el proyecto real y los valores estimados obtenidos mediante la ecuación inicial. Bailey y Basili identificaron cerca de 100 atributos dependientes del entorno local de desarrollo como posibles causantes de la variación entre el esfuerzo medido y el estimado, aunque al trabajar solamente con 18 conjuntos de datos no podían considerar matemáticamente tantos atributos. Para resolver el problema propusieron la utilización de distintas técnicas como la experiencia de expertos, o las matrices de correlación, para seleccionar los más influyentes; de esta forma obtuvieron finalmente 21 atributos. Además agruparon dichos atributos siguiendo la lógica de que el grupo tuviera un impacto positivo o negativo sobre el esfuerzo y fuera fácilmente explicable. Los tres grupos obtenidos fueron:

   1. Metodología Total (METH)
      1. Diagramas de árbol.
      2. Diseño Top – Down.
      3. Formalismos de Diseño.
      4. Lectura de código.
      5. Jefe del grupo de programadores.
      6. Planes formales de pruebas.
      7. Unidades de desarrollo.
      8. Planes formales de formación.

   2. Complejidad Acumulada (CMPLX)
      1. Complejidad de interfaz de usuario.
      2. Cambios del cliente iniciado el diseño.
      3. Complejidad de proceso de la aplicación.
      4. Complejidad del flujo de datos del programa.
      5. Complejidad de comunicación interna.
      6. Complejidad de comunicación externa.
      7. Complejidad de la base de datos.

   3. Experiencia Acumulada (CEXP)
      1. Aptitud del programador.
      2. Experiencia del programador con la máquina.
      3. Experiencia del programador con el lenguaje.
      4. Experiencia del programador con la aplicación.
      5. Trabajos previos del equipo trabajando juntos.

      Se evalúan cada uno de estos grupos en un rango del 1 al 5.

3. Utilizar el modelo para predecir nuevos proyectos.

La ecuación inicial o relación básica entre esfuerzo y tamaño se determinó, como se ha dicho más arriba, utilizando 18 conjuntos de datos procedentes del SEL (Software Engineering Laboratory – Laboratorio de Ingeniería del Software) de la NASA (National Agency Space Administration – Agencia Nacional para la Administración del Espacio).
El funcionamiento de este modelo se basa en la siguiente ecuación de esfuerzo:

Esfuerzo: $e = 3.5 + 0.73s^{1.16} \prod_{i=1}^{3} xi$

Donde *e* representa el Esfuerzo medido en MM, *s* el tamaño en SLOC, y *xi* el valor de cada grupo de atributos.

3.6 COCOMO

COCOMO procede de COnstructive COst MOdel, y consta de 2 versiones; La primera es llamada COCOMO 81, se toma de los dos últimos dígitos del año de publicación y se añaden cuando aparece la nueva versión COCOMO II. COCOMO 81 fue publicado por primera vez por Boehm en 1981. [3]
Fue desarrollado por Bary Boehm en TRW, basándose en el análisis de 63 proyectos software completados. Este modelo está compuesto de tres submodelos:

1. Básico.
2. Intermedio.
3. Detallado.

El modelo COCOMO 81 se basa en las siguientes ecuaciones de esfuerzo y tiempo de desarrollo:

1. *Esfuerzo:* $e = \sum_{i=1}^{n} \left[a(s)^b \prod_{ij=1}^{15} x_{ij}\right]$

Donde *e* representa el Esfuerzo medido en MM, *s* el tamaño en SLOC, *a* y *b* son constantes, y x_{ij} el valor de conductor de costes *j* para la fase *i*, donde *i* puede valer 1 para el modelo intermedio y de 1 a 4 para el modelo detallado.

2. *Tiempo de desarrollo:* $t = c(e)^d$

Donde t representa el tiempo en meses, e representa el esfuerzo medido en MM, y c y d son constantes.
La segunda versión, COCOMO II, fue publicada en 1995 por Boehm [4] y consta de las siguientes herramientas:
1. COCOMO II. 1999.0. Desarrollada por el equipo de la Universidad del Sur de California que ha desarrollado el modelo teórico. Es gratuita.
ftp: ftp://ftp.usc.edu/pub/soft_engineering/COCOMOII/cocomo99.0/c990windows.exe

2. COSTAR. Desarrollada por Costar. Es una herramienta interactiva que permite realizar seguimientos de proyectos, así como realizar experimentos del tipo ¿Qué sucedería si?
Web: http://www.softstarsystems.com
Este modelo está compuesto por tres submodelos:

1. Composición de Aplicaciones.
Este submodelo es usado para estimar el esfuerzo y el tiempo de desarrollo en proyectos que utilizan herramientas CASE (Computer Aided Software Engineering – Ingeniería del Software Asistida por Ordenador) para desarrollo rápido de aplicaciones. Estos productos aunque son muy diversos, son lo suficientemente simples como para ser rápidamente construidos a partir de componentes interoperables. Componentes típicos son los constructores del GUI (Graphical User Interface – Interfaz Gráfico de Usuario), gestores de bases de datos o de objetos, o procesos de transacción, etc. así como componentes de dominios específicos como paquetes de control médico o industrial.

Este modelo utiliza como variable de medida del tamaño del producto los Puntos Objeto [1; 10]. Los puntos objeto son básicamente una cuenta de las pantallas, los informes y los módulos, desarrollados en un lenguaje de tercera generación, en la aplicación. Cada cuenta es ponderada mediante un factor de complejidad de tres niveles, simple, medio y complejo. Solo existe un calibrado de este modelo [KAUffman y Kumar, 1993] debido a la falta de datos.

2. Diseño Inicial.
Este submodelo tiene en cuenta la exploración de diferentes arquitecturas del sistema y conceptos de operación. Normalmente no es suficiente para hacer estimaciones de precisión.

Utiliza como tamaño del producto los Puntos de Función o Las Líneas de Código, cuando están disponibles.

Introduce un conjunto de 5 Factores de Escala. A cada uno de los cuales se les asigna un valor que será un número real correspondiente al rango en el que se sitúe el factor para el proyecto que se está desarrollando. Hay seis rangos: Muy Bajo, Bajo, Nominal, Alto, Muy Alto y Extra Alto. Los factores de escala son los siguientes:

1. Precedentes (PREC).

Tiene en cuenta la experiencia de la organización en el desarrollo de este tipo de aplicaciones.

2. Flexibilidad en el desarrollo (FLEX).

Tiene en cuenta la rigidez de los requisitos y de las restricciones. Junto con la anterior sustituyen a los modos de desarrollo de COCOMO 81.

3. Arquitectura / Solución de riesgos (RESL).

Tiene en cuenta las medidas tomadas para la eliminación de riesgos.

4. Cohesión del equipo (TEAM).

Refleja las dificultades de cohesión y sincronización de los implicados en el proyecto, debido a su diversa procedencia y objetivos. Estos son: usuarios, clientes, desarrolladores, equipo de mantenimiento, etc.

5. Madurez de procesos (PMAT).

Este factor tiene en cuenta el nivel de madurez de procesos de la organización, para lo cual utiliza la escala de cinco niveles del CMM (Capability Maturity Model) desarrollado por el SEI (Software Engineering Institute) de la Universidad Carnegie Mellon.

Además este submodelo utiliza 7 Multiplicadores de Esfuerzo a los que se les asigna un número real teniendo en cuenta el rango, de seis niveles (Muy Bajo, Bajo, Nominal, Alto, Muy Alto y Extra Alto), en que se encuentre el multiplicador para el proyecto estudiado. La tabla de valores de los multiplicadores de esfuerzo se calibra uniendo los multiplicadores de esfuerzo calibrados para el modelo Post – Arquitectura [USC-CSE 1997]. Dichos conductores de coste son:

1. Capacidad del personal (PERS).
2. Fiabilidad y complejidad del producto (RELY).
3. Reutilización requerida (RUSE).
4. Dificultad de la plataforma (PDIF).
5. Experiencia del personal (PREX).
6. Facilidades (FCIL).
7. Calendario (SCED).

3. Post – Arquitectura.

Este submodelo puede ser utilizado cuando se ha completado el diseño de alto nivel y se dispone de información detallada sobre el modelo y, como su nombre sugiere, la arquitectura del software está bien definida y establecida. Permite realizar estimaciones para el conjunto del ciclo de vida de desarrollo y es una extensión del modelo Intermedio de COCOMO 81. Utiliza Puntos de Función y/o Líneas de Código Fuente como entrada para el parámetro del tamaño del producto. Utiliza un conjunto de cinco factores de escala, iguales en su fundamento conceptual a los vistos anteriormente para el submodelo de diseño inicial, y diecisiete multiplicadores de esfuerzo, cada uno con un rango de seis niveles igual al visto para el submodelo de diseño inicial. Este submodelo ha sido calibrado sobre una base de datos de 161 proyectos recopilados de la industria comercial, aeroespacial, gubernamental y de organizaciones sin ánimo de lucro; utilizando una aproximación bayesiana [5]. Los conductores de costes se agrupan en las siguientes cuatro categorías:

1. Producto.
    1. Fiabilidad requerida del software (RELY)
    2. Tamaño de la base de datos (DATA)
    3. Complejidad del producto (CPLX)
    4. Reutilización requerida (RUSE)
    5. Documentación desarrollada (DOCU)

2. Plataforma.
    6. Restricciones en el tiempo de ejecución (TIME)
    7. Restricciones en el almacén principal (STOR)
    8. Volatilidad de la plataforma (PVOL)

3. Personal.
    9. Aptitud de los analistas (ACAP)
    10. Aptitud de los programadores (PCAP)
    11. Experiencia en el desarrollo de aplicaciones similares (AEXP)
    12. Experiencia con la plataforma de desarrollo (PEXP)
    13. Experiencia con el lenguaje y la herramienta (LEXP)
    14. Continuidad del personal (PCON)

4. Proyecto.
    15. Utilización de herramientas software (TOOL)
    16. Desarrollo en múltiples localizaciones (SITE)
    17. Tiempo necesario para el desarrollo (SCED)

Las ecuaciones que utiliza este modelo para su funcionamiento son las siguientes:

1. *Esfuerzo:* $e = a.(s)^{(b+c\ \Sigma_{i=1}^{5} yi)} . (\prod_{j=1}^{n} x_j)$

Donde *e* representa el Esfuerzo medido en MM, *s* el tamaño en SLOC, *a*, *b* y *c* son constantes, y x_j el valor de conductor de costes *j*, *n* vale 7 en el submodelo diseño preliminar y 17 en post – arquitectura y y_i es el factor de escala *i*.

2. *Tiempo de desarrollo:*

$t = (e)^{0.33+0.2(b-1.01)} . \left(\frac{SCED\ \%}{100}\right)$

Donde *t* representa el tiempo en meses, *e* representa el esfuerzo medido en MM, *b* es una constante y *SCED* es el multiplicador de esfuerzo Tiempo necesario para el desarrollo.

3.7 COPMO

COPMO procede de COoperative Programming MOdel. Su primera publicación la realizaron Thebaut y Shen en el año 1984 [12].
Este modelo utiliza como variables independientes el tamaño del producto a desarrollar y la medida del tamaño del equipo que desarrolla el producto.
Está basado en la siguiente ecuación de esfuerzo:

Esfuerzo: $e = a + b.s + c(n)^d$

Donde *e* representa el Esfuerzo medido en MM, *s* el tamaño en miles de SLOC, *a, b, c* y *d* son constantes, y *n* es el tamaño medio del equipo de desarrollo.

4. CONCLUSIONES

En este artículo, hemos presentado una revisión de los métodos de estimación de esfuerzo para proyectos software que han sido desarrollados a lo largo de la historia de la ingeniería del software, centrados fundamentalmente en los modelos de estimación de esfuerzo públicos y No-Lineales. Con esta revisión, hemos intentado

mostrar cómo estos modelos de estimación han evolucionado desde sus primeras publicaciones y las bases matemáticas que han seguido para su implementación.
Teniendo una idea clara de cómo estos modelos de estimación han sido desarrollados e implementados, entonces podemos llegar a entender de una forma más adecuada cómo se están desarrollando los modelos actuales y las líneas que podemos seguir para estudiar los siguientes.
Este artículo no pretende presentar nuevos modelos ni líneas a seguir para desarrollarlos, simplemente se desea mostrar de una forma cronológica y partiendo de una nueva clasificación propuesta, modelos públicos No-Lineales de estimación de esfuerzo para proyectos software. Por consiguiente, como líneas futuras de investigación, propondremos un estudio de los modelos propietarios o patentados que han supuesto un gran progreso tanto técnico como económico en la industria de planificación de procesos y esfuerzo de proyectos software.

5. AGRADECIMIENTOS

Los autores agradecen sus expertas indicaciones al profesor Dr. Juan José Cuadrado-Gallego, Director del Laboratorio de Investigación en Medición de Software y Gestión de Proyectos CuBIT, así como a la Universidad de Alcalá de Henares por el apoyo mostrado en esta investigación bajo el programa *Ph.DC. de apoyo a la investigación*.

6. REFERENCIAS

[1] Banker, R., Kauffman, R. and Kumar, R.: An Empirical Test of Object-Based Output Measurement Metrics in a Computer Aided Software Engineering (CASE) Environment, En *Journal of Management Information System*, 1994.

[2] Baylei, J. and Basili, V.: A Meta-model for Software Development Resource Expenditures, En *Proceedings of the Fifth International Conference on Software Engineering*, (1981), pp. 107-116.

[3] Boehm, B.: *Software Engineering Economics*, Editorial Prentice Hall, 1981.

[4] Boehm, B., Clark, B., Horowitz, E., Madachy, R., Selby, R. and Westland, C.: Cost Model for Future Software Life Cycle Processes: COCOMO 2.0, In *Annals of Software Engineering Special Volume on Software Process and Product Measurement*, Eds. J.D. Arthur, S.M. Henry and J.C. Baltzer, Edit. AG Science Publishers, Amsterdam (Holland), (1995), Vol. 1.

[5] Chulani, S., Boehm, B. and Steece, B.: Calibrating Software Cost Models Using Bayesian Analysis, En *Technical Report USC-CSE-98-508*, (1998).

[6] Conte, S. D., Dunsmore, H. E., Shen, V. Y.: *Software Engineering Metrics and Models*. Benjamin / Cumming Co., Inc. Menlo Park. 1986.

[7] Cuadrado-Gallego, J. J.: Métodos de Estimación de Proyectos Software, In *UC3M-TR-CS-2000-01* Vol. 1, Ed. Dep.Informática UC3M, (Spain), 2000.

[8] Herd, J., Postak, J., Russel, W. and Stewart, K.: Software Cost Estimation Study – Study Results, En *Final Technical Report, RADC-TR-77-220*, Ed. Doty Associates, Inc., 1977.

[9] James, T. Jr.: Software Cost Estimating Methodology. En *IEE Proceedings of National Aerospace Electronic Conference*, (1997), pp 22-28.

[10] Kauffman, R. and Kumar, R.*: Modelling Estimation Expertise in Object Based ICASE Environments*, Editorial Stern School of Business Report, New York University, January 1993.

[11] Taback, R., and Ditimore, J.: Estimation Computer Requirements and Software Development Costs, In *RM-1873*, Eds. General Research Corporation, 1974.

[12] Thebaut, S. y Shen, V.: An Analytic Resource Model for Large-Scale Software Development, En *Inf. Proc. Management*, 20 (1-2), (1984).

[13] Walston, C. and Felix, C.: (1) A Method of programming Measurement and Estimation, En *IBM System Journal*, 16, (1), (1977, pp. 54-73. (2) Author´s Response, In *IBM System Journal*, 16 (4), (1977), pp. 422-42

EVALUACIÓN DE CÓDIGO MEDIANTE MÚLTIPLES INTERVALOS DE MÉTRICAS

Carlos López, Yania Crespo, Esperanza Manso, Raúl Marticorena
Universidad de Burgos, Universidad de Valladolid
{clopezno,rmartico}@ubu.es, {yania,manso}@infor.uva.es

Abstract: Code evolution and maintenance include measurement activities to detect possible flaws and to suggest improvements. An approach to detect anomalies on code entities is based on metric values outside their thresholds. From the 90s, there are plenty of papers that propose general threshold values for a set of metrics. One of the criticisms outlined in the literature to this approach is the difficulty of using these results to other contexts. Threshold values may be influenced by multiple variables, including the nature of the problem solved: exception, graphical interface, model and test. This information can be introduced by the user in the measurement process, making it semiautomatic. To verify this hypothesis, this paper describes a case study where the measurement process is guided by the inspector/assessor, who classifies code entities to measure depending on their nature. The case study result suggests some metric thresholds for different kind of problems and relative to a specific organization.

Resumen: La evolución y mantenimiento del código incorporan en su proceso actividades de medición para detectar posibles defectos y proponer mejoras. Una manera de detectar anomalías sobre entidades de código se basa en comprobar que el valor de una métrica está fuera de un intervalo de valores recomendados. En este sentido, desde la década de los 90, existen multitud de trabajos que proponen de manera empírica intervalos recomendables para un conjunto de las mismas. Una de las críticas expuesta en la literatura a este planteamiento es la dificultad de transportar estos resultados a otros contextos. La determinación de los valores que comprenden el intervalo puede estar condicionada por múltiples variables, entre ellas la relacionada con la naturaleza del problema que resuelve: excepciones, interfaces gráficas, modelo, controladores y pruebas. Esta información puede ser incorporada por el usuario en el proceso de medición, transformándolo en semiautomático. Para corroborar esta hipótesis, en este trabajo se define un caso de estudio donde el proceso de medición es guiado por el inspector/evaluador, que clasifica las entidades de código a medir dependiendo su naturaleza. Como resultado del caso de estudio se proponen unos intervalos relativos a las métricas para los diferentes tipos y relativos a una organización concreta.

Palabras Clave: Métricas de Código, Intervalos Relativos, Experimento, Proceso de Medición, Herramientas de Medición

1. INTRODUCCIÓN

Desde la década de los 90 las métricas del software y el proceso de medición asociado han captado la atención de la comunidad de la ingeniería del software como medio de cuantificar y controlar la calidad del software [1-4]. La medición es una actividad que se encuentra englobada dentro de un proceso (ver Figura 1), que consiste en derivar un valor numérico para algún atributo de un producto o proceso software.

En este proceso, la detección de entidades defectuosas se basa en la identificación de medidas anómalas sobre las entidades. De manera pragmática la identificación consiste en comprobar si una medida está dentro de unos valores recomendados. De una manera general, este proceso de medición puede aplicarse sobre múltiples productos resultado del proceso de desarrollo del software, dando lugar a modelos de calidad de producto software como el propuesto en la norma ISO 9126 [5].

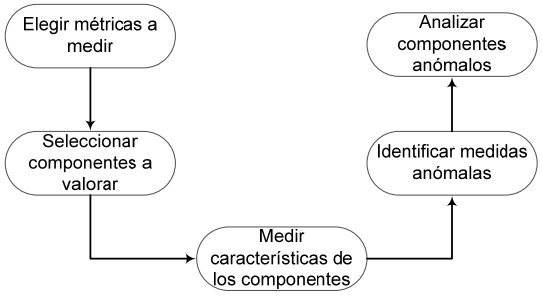

Figura 1 Proceso de medición

Desde un punto de vista más concreto el proceso se puede aplicar sobre el código ya que

es un producto en constante evolución y mantenimiento [6]. La aplicación de métricas sobre él, con su posterior interpretación de valores, es una de las técnicas que puede ayudar a evaluar la calidad del mismo. La evaluación de código mediante métricas no es nueva. Como consecuencia de ello, en la literatura existe un gran número de definiciones de métricas agrupadas por criterios diferentes dependiendo del autor. Por ejemplo, en el paradigma de la orientación a objetos algunos conjuntos de métricas bien conocidos sobre diferentes entidades de código son:

- Sobre clases: Chidamber y Kemerer [7], Lorenz y Kid [8].
- Sobre subsistemas: Robert Martin [9], Brito e Abreu [10].
- Sobre métodos: MCabe[11].
- Otros recogidos en [12].

Pero en la literatura también existen muchas críticas, resultado de su aplicación. Una de ellas es que los valores utilizados para identificar medidas sobre entidades anómalas obtenidas a través de experimentos empíricos están restringidos al contexto de medición, siendo limitada su utilización en otros contextos. Aunque somos conscientes que es muy difícil acotar por completo la heterogeneidad de los diferentes contextos, este trabajo pretende avanzar en este aspecto. Para ello se propone incorporar nuevas tareas en el proceso de medición, basadas en el conocimiento del inspector/evaluador respecto a la identificación de entidades en función de la naturaleza del problema que resuelven. Bajo esta premisa y en un entorno de la orientación a objetos, una selección de criterios de clasificación se puede basar en la utilización de algunos estereotipos sobre clasificadores de UML. Por ejemplo, los estereotipos sobre clases de análisis [13] [14]: entidad, controladores, límites. Además el criterio de clasificación límite se desglosa a su vez en: interfaces de usuario, interfaces de sistema e interfaz de dispositivo. Otros estereotipos interesantes en los clasificadores son los obtenidos como resultados de algunas tareas del proceso de desarrollo como los relacionados con excepciones y pruebas. Esto provoca un desglose de tareas en la actividad de medición obteniendo diferentes intervalos de validación para cada tipo de estereotipo considerado. En la Figura 2 se adapta el nuevo proceso de medición propuesto y se enmarca con un rectángulo el desglose de tareas.

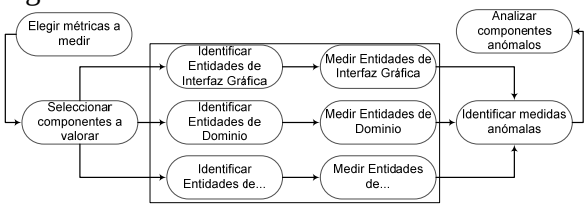

Figura 2 Proceso de medición propuesto

El objetivo del trabajo es corroborar de manera empírica la necesidad de múltiples intervalos de validación de métricas dependiendo de la naturaleza del problema que resuelve la entidad de código medida. Como resultado del caso de estudio se proponen unos intervalos de validación para un conjunto de métricas. En lo que sigue el artículo se estructura de la manera siguiente. En la sección 2 se introducen las nociones básicas asociadas al proceso de experimentación en ingeniería del software. El grueso del trabajo se presenta en la sección 3 que describe el caso de estudio utilizando las etapas identificas previamente en el proceso de experimentación: definición, planificación, operación, análisis, validez. Por último, en la sección 4 se concluye y proponen líneas futuras de actuación.

2. EXPERIMENTACIÓN EN INGENIERÍA DEL SOFTWARE

Cuando se plantea un estudio empírico siempre hay un conjunto de elementos del mundo real del que se necesita conocer cierto comportamiento que se expresa a través de una hipótesis. Ésta se quiere aceptar o rechazar tomando como base los resultados observables y medibles. En [15] se presentan tres tipos de estrategias empíricas que pueden

ayudar para realizar esta tarea: experimentos, casos de estudio y encuestas. Para este trabajo se enmarca dentro de la categoría de casos de estudio.

Los casos de estudio son estudios observacionales, esto es, investigan fenómenos en el contexto real cotidiano conforme se desarrollan. En la ingeniería del software se suelen utilizar para estudiar fenómenos a gran escala, como el desarrollo completo de procesos, monitorización de proyectos, o para evaluar métodos o herramientas. Su propósito está orientado normalmente a analizar un determinado atributo o establecer relaciones entre diferentes atributos.

Otro aspecto importante para realizar experimentación en ingeniería del software es la elección de un proceso que asista al desarrollo del caso de estudio. El proceso experimental que se ha seguido es el propuesto en [16], que consta de seis etapas principales (ver Figura 3), y sirven como plantilla base para presentar y estructurar el desarrollo del mismo.

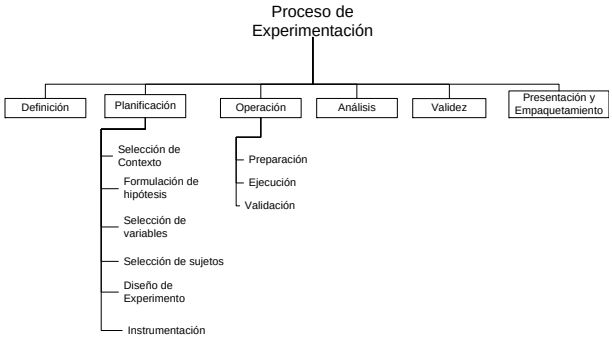

Figura 3 Visión general del proceso experimental

3. CASO DE ESTUDIO

Este apartado presenta a los resultados obtenidos al aplicar el de proceso experimental elegido [16]. Cada uno de los subapartados se corresponde con una etapa del proceso.

3.1 Definición

Analizar códigos fuentes mediante métricas

- Con el propósito de comprobar si sus valores recomendados varían dependiendo de la naturaleza del problema que resuelve la entidad medida. La naturaleza del problema se define a partir de algunos estereotipos de UML sobre clasificadores.
- Con respecto a los factores de calidad de código relacionados con de tamaño, documentación, estructura y complejidad.
- Desde el punto de vista de mejorar la identificación de entidades anómalas en el proceso de medición.
- En el contexto de una asignatura de proyecto fin de carrera de Ingeniería Informática.

Se busca obtener evidencias empíricas que en la evaluación de sistemas heterogéneos con un solo intervalo de interpretación puede dar lugar a muchos resultados erróneos.

3.2 Planificación

Si la definición establece por qué se va a realizar el experimento, la planificación establece como se llevará a cabo. Esta etapa se divide en seis pasos que se corresponden con cada una de las siguientes secciones.

3.2.1 Selección de Contexto

En el 5º curso de Ingeniería Informática en la Universidad de Burgos (UBU) existe una asignatura de Sistemas Informáticos conocida comúnmente como proyecto final de carrera. En ella, los alumnos deben presentar un proyecto obtenido fruto de un proceso de desarrollo. Como responsables de la evaluación de los mismos se establecieron unos criterios de evaluación donde se incluyó uno para valorar la calidad del código entregado. Para ello se utilizan métricas de código recogidas automáticamente a través de herramientas y su interpretación a través de intervalos generales y otros relativos a la Universidad de Burgos. Como criterio cuantificable se utiliza un indicador de cobertura de métricas que se obtiene contando el número de los valores de las métricas de un proyecto que están dentro de los intervalos recomendados y se divide entre

número total de métricas consideradas. Este indicador sirve como criterio de conformidad equivalente al concepto de compliance definido en los modelos de calidad de producto ISO 9126.

En la Figura 4 se muestra una evaluación de un conjunto de proyectos (filas de la tabla) utilizando dos indicadores de cobertura uno relativo a la propia organización UBU y otro general basado en los intervalos recomendados por la herramienta. Además se muestran los valores ordenados respecto al indicador general (columna Cobertura Tool). Un histórico de los datos de métricas de distintos trabajos puede obtenerse en (ver http://pisuerga.inf.ubu.es/lsi/Asignaturas/SI/MetricSist.html#Coberturas).

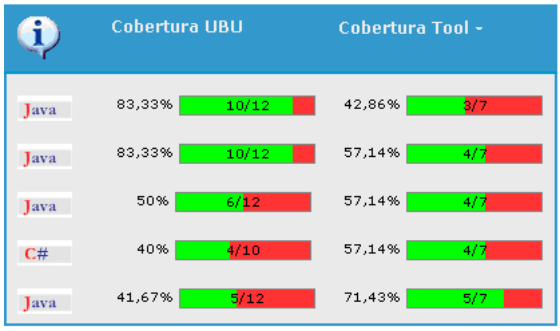

Figura 4 Evaluación de proyectos

En este proceso de evaluación, aparecieron algunos casos en los cuales se obtuvieron resultados contradictorios entre las coberturas de las métricas y los criterios subjetivos de los miembros del tribunal de evaluación.

La experiencia adquirida es que la interpretación de un valor de una métrica utilizando un sólo intervalo general o relativo a la propia organización no es suficientemente fiable y puede ser mejorada si se incorpora al proceso de medición información de la naturaleza del problema que resuelve el proyecto medido.

Hasta donde alcanza el conocimiento de los autores no existen clasificaciones de proyectos basadas en la naturaleza del problema. Pero cuando se realiza una rápida inspección de código tanto los desarrolladores como los evaluadores de una organización sí conocen la naturaleza del problema al que se enfrentan distintas entidades de código que componen el proyecto. Así, a través de las inspecciones realizadas basadas en la estructura código, se pueden distinguir las capas y subsistemas del proyecto: interfaz gráfico, modelo, controladores, pruebas. Además dentro de las capas y subsistema existen unas clases especiales llamadas excepción para el tratamiento de errores.

3.2.2 Formulación de hipótesis

H_0: Todos los grupos diferentes de entidades orientadas a objeto respecto de la naturaleza del problema que resuelven (e_1 excepciones, e_2 interfaz gráfico, e_3 modelo, e_4 controladores, e_5 pruebas) se comportan igual con respecto a las métricas de código.

El refinamiento de esta hipótesis se basa en la formulación de hipótesis estadísticas formuladas como hipótesis nula (H_0) frente a su alternativa (H_1), la cuál esta implícitamente aceptada como la negación de H_0 ($H_0 = \neg H_1$). Sobre cada una de las métricas de código consideradas (m) se refina la hipótesis basada en la media teórica (μ) de cada grupo de clases (e_i)

$$H_0: \mu^m_{e1}=\mu^m_{e2}=\mu^m_{e3}=\mu^m_{e4}=\mu^m_{e5}$$

La decisión de aceptar una u otra hipótesis en función de los resultados observados implica una serie de riesgos que medidos en términos de probabilidad son α y β. Si se elige H_1 pero la real es H_0 se comete un error que en términos de probabilidad se denomina α el valor considerado para α es de 0.01.

3.2.3 Selección de sujetos

El proceso de muestreo seguido ha sido aleatorio respecto a los proyectos presentados en la asignatura. La datos se corresponde con los valores de las métricas de código de 30 proyectos presentados en la asignatura de Sistemas Informáticos de 5º de Ingeniería Informática de la Universidad de Burgos desde el curso 2003-04 (http://pisuerga.inf.ubu.es/lsi/Asignaturas/SI/

HistoricoSist.html), 2 desarrollados por profesores de la misma Universidad y 2 de carácter industrial, Java Development Kit 1.5.0, JUnit 4.0. Estos cuatro últimos sirven para tener una referencia de comparación con el trabajo realizado por los alumnos y como criterio de validación externo del caso de estudio.
Tomando como referencia las líneas de código (LOC) en la Tabla 1 se recoge el tamaño de los códigos analizados dependiendo de la naturaleza del problema considerada.

Naturaleza del problema	LOC.
e_1 excepciones	8.711
e_2 interfaz gráfico	645.293
e_3 modelo	154.661
e_4 controladores	182.243
e_5 pruebas	93.730

Tabla 1 Tamaño de la muestra

3.2.4 Diseño del experimento

El diseño del experimento va a determinar cómo se estudian los datos obtenidos. Esto está muy relacionado con los tipos de análisis estadísticos que se pueden llevar a cabo.
Por cada proyecto se deben identificar las entidades de cada categoría (e_1 excepciones, e_2 interfaz gráfico, e_3 modelo, e_4 controladores, e_5 pruebas) y posteriormente calcular sus medias aritméticas. En la Tabla 2 se muestra un ejemplo con los resultados de la medición de una métrica de complejidad sobre un mismo proyecto y respecto a todas las categorías consideradas. La categoría “global” mostrada en la primera fila de la columna cuya cabecera es “Naturaleza” hace referencia al cálculo de la métrica sobre todo el proyecto sin hacer ningún tipo de clasificación. Se considera un diseño del caso de estudio intrasujeto, es decir todos los proyectos tienen el mismo tratamiento.

Nombre Proyecto	Naturaleza	Complej.
2005_refactorizaciones_xmi	global	2,12
2005_refactorizaciones_xmi	excepción	1
2005_refactorizaciones_xmi	interfaz	1,9
2005_refactorizaciones_xmi	modelo	1,95
2005_refactorizaciones_xmi	controlador	1,89
2005_refactorizaciones_xmi	prueba	2,59

Tabla 2 Medición de un proyecto

Como técnica de contraste de hipótesis estadística se ha elegido el Análisis de Varianza en sus variantes ANOVA de una vía o ANOVA no paramétrico de Kruskal-Wallis, que afecta simultáneamente a los valores medios o esperados (media aritmética, mediana) de *k* poblaciones (variables aleatorias). En el caso de de ANOVA de una vía, los valores de la muestra deben seguir una distribución normal y tener idénticas varianzas, en caso de no cumplirse se aplicará la versión de Kruskal-Wallis.
En este estudio interviene una variable aleatoria Y (métrica de código), denominada variable observable o variable respuesta, que se analiza en *k* situaciones experimentales (e_1 excepciones, e_2 interfaz gráfico, e_3 modelo, e_4 controladores, e_5 pruebas), las cuales definen el llamado factor o vía *k* niveles de tratamiento (Naturaleza). Esta técnica deberá ser aplicada sobre cada una de las métricas de código consideradas. Si al aplicar el contraste de análisis de varianza el resultado de p-valor < 0,01 se rechaza la hipótesis de igualdad de un único intervalo de validación H_0, aceptando la necesidad de múltiples intervalos dependiendo de naturaleza del problema que resuelven.
Para determinar la variante de análisis de varianza, una vía o Kruskal-Wallis, se evaluarán

las precondiciones de ANOVA de una vía utilizando los siguientes contrastes:

- Respecto a la estructura de probabilidad normal con el contraste de Shapiro-Wilk.
- Respecto a la homocedastidad se empleará el test de homogenidad de varianzas de Barlett.

Si alguno de ellos falla, es decir se obtiene un resultado p-valor <= 0,05 se aplicará el test de Kruskal Wallis.

Si se aplica el contraste de ANOVA de una vía y se rechaza la hipótesis de igualdad nula de medias H_0 se procederá a la realización de contrastes de medias dos a dos ($m_ie_2 - m_ie_1$, $m_ie_3 - m_ie_1$, $m_ie_4 - m_ie_1$,…, $m_ie_5 - m_ie_4$, donde m_i es la métrica de código tratada y e_j las situaciones experimentales que definen el factor guía naturaleza del problema). El resultado es un mapa de relaciones de las k situaciones consideradas. El test recomendado para llevar a cabo comparaciones múltiples es el de Tukey por proporcionar intervalos de confianza de menor longitud.

Por último, independientemente de la versión de análisis de varianza empleado y en caso de aceptar H_1, se aplicarán los estadísticos percentil 25 y 75 sobre cada una de las métricas consideradas para cada uno de los tipos de problemas considerados (e_1 excepciones, e_2 interfaz gráfico, e_3 modelo, e_4 controladores, e_5 pruebas). Estos valores se pueden utilizar para estimar los valores de los intervalos válidos de una métrica [4].

En el siguiente diagrama de actividad se muestra a modo guía los diferentes test aplicados y las decisiones basadas en p-valor (α) para llevar a cabo el caso de estudio.

3.2.5 Instrumentación

Esta sección se ha desglosado en dos, una dedicada a herramientas para medir métricas de código y otra que muestra una relación entre métricas y propiedades de diseño.

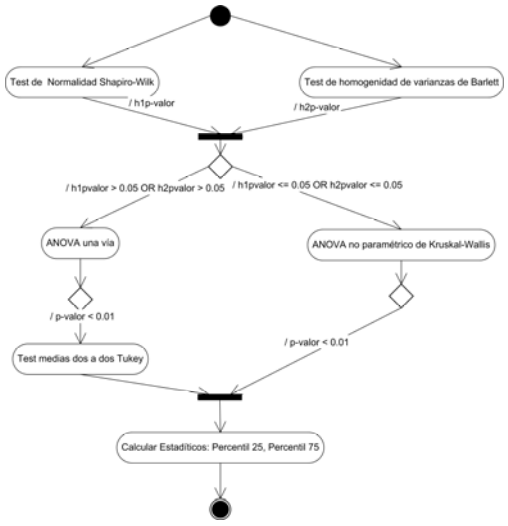

Figura 5 Diseño de del experimento

3.2.5.1 Métricas de código y herramientas

La actividad de medición necesita de herramientas que automaticen el cálculo de valores de las métricas para una determinada entidad de código. Además los resultados deben ser exportables para poder ser analizados con otras herramientas de análisis estadísticos de datos.

Para ayudar a la elección de herramientas en la Tabla 3 se muestra el resultado de la evaluación de un conjunto de herramientas respecto a las siguientes características:

C1.Lenguaje de programación sobre el que trabajan.

C2.Entrada: ficheros binarios o fuentes (binarios/fuentes/ambos).

C3.Número de métricas que calcula.

C4. Formato de exportación de resultados (html/txt/xml/xls).

C5. Indicadores gráficos o técnicas de agrupación y filtrado para analizar los resultados (Si/No)

Una de las críticas realizadas a la experimentación con métricas de código es la falta de precisión en su definición. Esto se pone de manifiesto en las herramientas en las diferentes reglas de cálculo aplicadas con la consecuente diferencia de valores obtenidos en el

cálculo de la misma métrica sobre la misma entidad de código.

Herramientas	C1	C2	C3	C4	C5
Dependency Finder	java	binarios	33	html,txt,xml	No
RefactorIt	java	fuentes	25	html,txt xml	Si
JDepend	java	binarios	9	html,txt,xml	No
Eclipse Metrics - v1.3.6	java	fuentes	25	xml	No
NDepend	.NET	ambos	66	html, txt, xml, xls	Si
SourceMonitor	java, C#, C++, VB	fuentes	14	txt, xml,	Si

Tabla 3 Herramientas de medida de código

Aunque las definiciones de muchas de las métricas de código se mantienen independientes del lenguaje de programación, en la práctica muchas herramientas o componentes sólo trabajan sobre un único lenguaje de programación (ver columna C1 de la Tabla 3).

En el contexto de medición que se plantea en este trabajo se debe permitir evaluar proyectos de diferentes lenguajes de programación y la exportación de resultados en formatos de texto para el posterior análisis de los mismos. Bajo estas premisas la herramienta seleccionada en medición es SourceMonitor [17]. En la Tabla 4 se presentan las métricas de subsistema proporcionadas por la herramienta y los intervalos recomendados sobre algunas de ellas.

Descripción	Identificador	MinValor	MaxValor
Nombre proyecto	M0		
Líneas de código	J0		
Nº de sentencias	J1		
% de sentencia condicionales	J2		
Nº de llamadas a métodos	J3		
% Líneas de comentarios	J4	8	20
Nº de clases e interfaces	J5		
Nº de métodos por clase	J6	4	16
Media de sentencias por método	J7	6	12
Máxima complejidad	J10	2	8
Máxima profundidad de bloques	J12	3	7
Media de profundidad de bloques	J13	1	2,2
Media de Complejidad	J14	2	4

Tabla 4 Métricas de la herramienta SourceMonitor

3.2.5.2 Caracterización de subsistemas mediante métricas

Independientemente de las convenciones particulares sobre las métricas de código, cada vez que se analiza o se habla del código de un sistema software, se quiere obtener una impresión del tamaño y complejidad del mismo. Algunos trabajos expresan el tamaño de un sistema en términos de líneas de código, número de clases e incluso cantidad de megabytes del código fuente.
Estos números no son nada más que valores de alguna métrica básica. Desafortunadamente después de obtener un conjunto de valores de manera aislada todavía se tiene problemas para caracterizar el sistema o entidad evaluada. La caracterización a través de métricas tiene que servir para reflejar la bondad de los principales aspectos de diseño de como son: tamaño, complejidad, acoplamiento, herencia, documentación, etc.

Para proporcionar una caracterización de este tipo a partir de las métricas definidas en la Tabla 4, este caso de estudio supone la relación entre características y métricas que se muestra en la Tabla 5.

DOCUMENTACIÓN	% de líneas de comentarios	J4
ESTRUCTURA	Nº sentencias por	J1/J
	Nº de métodos por	J6
	Nº sentencias por método	J7
COMPLEJIDAD	% de sentencia condi-	J2
	Media de profundidad de bloques	J13
	Media de complejidad	J14

Tabla 5 Caracterización de código mediante métricas

3.3 Operación

Por cada proyecto, el tiempo empleado para el cálculo de métricas e identificación de tipos de entidades ha estado comprendido entre 15 y 20 minutos.

Para identificar los diferentes tipos de las entidades de código, se ha utilizado como técnica la convención de nombres de las entidades. Además por cada tipo de entidad se han considerado dos criterios de identificación: identificación por agrupamiento (agrupación o individual) y el tipo de conocimiento necesario para la identificación (genérico o específico del proyecto).

La identificación por agrupación se basa en la identificación de capas del proyecto, se considera que todas las entidades que están en una misma capa son del mismo tipo. Por ejemplo, la identificación de entidades de tipo e_2 interfaz gráfico de un proyecto, puede realizarse con la identificación de la capa de interfaz gráfica, cuya convención de nombres suele ser “interfaz”, “gui”, “form”, etc. El conocimiento necesario basado en convención de nombres para identificar entidades puede ser genérico respecto a convenciones de diseño o del lenguaje de programación, o bien por conocimiento específico de los requisitos del proyecto. Por ejemplo, si en un proyecto de cálculo de métricas aparece una capa llamada “métricas”, en una primera inspección pertenecerá a los tipos e_3 modelo o e_4 controladores. La Tabla 6 muestra un resumen de los criterios de identificación expuestos dependientes del tipo de entidad.

En la inspección realizada, la identificación de ciertas entidades no se ha podido etiquetar y no se han considerado en el caso de estudio. El porcentaje de etiquetado ha sido 76,6 %, siendo la unidad de porcentaje la línea de código.

Otra característica operacional del caso de estudio ha sido que no en todos los proyectos se han podido identificar entidades de los 5 tipos propuestos, el porcentaje de datos desconocido respecto al total ha sido del 27,9%.

	e_1	**e_2**	**e_3**	**e_4**	**e_5**
Agrupación	No	Si	Si	Si	Si
Conocimiento Genérico	Si	Si	Ambos	Ambos	Si
Convenciones de nombres	exception	interfaz gui forms ui report	core model	control fachada manager handler	test debug dummy

Tabla 6 Identificación de tipo de entidades

3.4 Análisis

A partir del caso de estudio llevado a cabo y aplicando los contrastes de hipótesis estadísticos especificados en la sección 3.2.4, se concluye de manera general en rechazar la hipótesis de partida H_0. Por tanto, se acepta como cierta la hipótesis H_1, que indica que los grupos diferentes de entidades orientadas a obje-

to respecto de la naturaleza del problema que resuelven (e_1 excepciones, e_2 interfaz gráfico, e_3 modelo, e_4 controladores, e_5 pruebas) se comportan de manera distinta respecto a métricas de código. En la Tabla 7 se muestra un resumen de los contrastes de hipótesis llevados a cabo (filas) y de sus resultados al ser aplicados sobre los datos de medición de cada de una de las métricas consideradas (columnas). El valor de cada celda se corresponde con la probabilidad de error α. Si se analizan los resultados del contraste de hipótesis correspondiente al análisis de varianza, en una de sus dos variantes, se aprecia que todas las métricas consideradas excepto la el porcentaje de sentencias condicionales (J2), presentan un resultado de rechazar la hipótesis nula H_0. Por otro lado, excepto el conjunto de datos formado por los valores de la métrica de porcentaje de líneas de comentarios (J4), el resto de conjunto de datos no cumplen alguna de las dos precondiciones para aplicar la variante ANOVA una vía, normalidad y homecedasticidad.

Respecto al conjunto de datos correspondientes a la métrica porcentaje de líneas comentarios (J4), en la Figura 6 se han calculado los intervalos de confianza de Tukey comparando dos a dos los valores de la media de J4 respecto a tipos de entidades considerados. El análisis de la salida lleva a las siguientes conclusiones:

- El porcentaje de líneas de comentario es mayor en e_1 excepciones que en el resto de tipos considerados.
- El porcentaje de líneas de comentario respecto a e_2 interfaz gráfico es mayor que el de e_5 pruebas, y es menor que e_3 modelo y e_4 controladores.
- El porcentaje de líneas de comentario respecto a e_3 modelo es mayor que el de e_4 controladores y e_5 pruebas.
- El porcentaje de líneas de comentario respecto a e_4 controladores es mayor que el de e_5 pruebas.

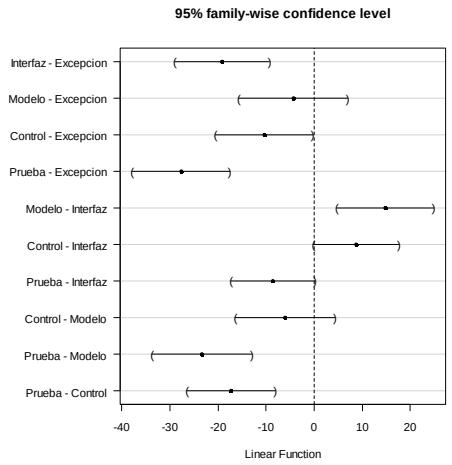

Figura 6 Intervalos de confianza de Tukey para métrica de porcentaje de documentación J4

Con estas evidencias empíricas este apartado concluye proponiendo unos intervalos válidos de métricas para cada uno de los tipos de entidades considerados (ver Tabla 8). La obtención de los extremos de los intervalos se adquiere calculando los estadísticos primer cuartil Q1 y tercer cuartil Q3 para cada tipo considerado.

Se ha utilizado R, como herramienta software para realizar los contrastes de hipótesis propuestos [18].

4. CONCLUSIONES YLÍNEAS DE TRABAJO FUTURAS

En este trabajo se propone una variación en el proceso de medición incorporando conocimiento del inspector/evaluador sobre la clasificación de entidades del programa dependiendo de su naturaleza (e_1 excepciones, e_2 interfaz gráfico, e_3 modelo, e_4 controladores, e_5 pruebas). A partir de dicha clasificación se han propuestos por cada una de sus categorías un intervalo de validación para un conjunto de métricas.

Se ha realizado un caso de estudio para corroborar la hipótesis de que la naturaleza del problema que resuelve la entidad de código medida influye en los valores de sus medidas.

En este sentido, somos conscientes que la restricción multilenguaje marcada por el contexto específico de trabajo en el que se ha llevado a cabo el caso de estudio, ha determinado en exceso el conjunto de métricas seleccionadas y la subjetiva relación con características de diseño: documentación, estructura y complejidad. Es por ello que para mejorar la validez del experimento se podrían realizar réplicas aplicando otros conjuntos de métricas. El caso de estudio aquí definido necesita de múltiples habilidades personales y técnicas en el uso de herramientas. Una de las líneas futuras de trabajo deseable en las herramientas de medida es la integración nuevas funcionalidades que reduzcan esta complejidad. Así se podría incorporar funcionalidad para clasificar las entidades según su naturaleza, mantenimiento de histórico para calcular intervalos de medidas, y poder realizar una evaluación multintervalo en función de su naturaleza.

	DOC	ESTRUCTURA			COMPLEJIDAD		
	J4	**J1 DIV J5**	**J6**	**J7**	**J2**	**J13**	**J14**
Normalidad Shapiro Wilk	0,4281	2,69 e-09	1,152 e-08	0,0007163	2,969 e-06	0.7094	2,459e-06
Igualdad de varianzas Barlett	0,1348	0,0004395	0,0004337	1,162 e-05	1,405 e-11	0.0002161	3,955e-06
ANOVA una vía	5,338 e-12	-	-	-	-	-	-
ANOVA Kruskal Wallis	-	0,0003056	0,0001013	4,947e-06	0,02264	4,934e-05	0,0001728

Tabla 7 Resultado de los contrastes estadísticos para cada una de las métricas

		DOC	ESTRUCTURA			COMPLEJIDAD		
		J4	**J1 DIV J5**	**J6**	**J7**	**J2**	**J13**	**J14**
e_1 excepciones	**Q1**	36.900	4.75000	1.750	1.0000	0.000	0.8450	1.0000
	Q3	56.900	25.06250	4.0000	5.785	29.500	2.2150	2.670
e_2 interfaz gráfico	**Q1**	23.450	44.16648	3.960	5.3250	8.875	1.8350	1.8750
	Q3	35.050	66.94631	7.4050	9.360	13.750	2.6450	2.815
e_3 modelo	**Q1**	41.900	33.11111	6.520	2.6750	7.050	1.4050	1.3950
	Q3	46.000	62.18954	9.2600	4.090	14.000	1.9100	2.020
e_4 controladores	**Q1**	32.050	36.19753	3.075	5.6450	10.300	1.8000	1.9700
	Q3	44.750	110.15374	11.1600	9.765	16.800	2.3550	2.935
e_5 pruebas	**Q1**	10.725	48.62500	5.545	4.5675	1.375	1.4775	1.0825
	Q3	26.875	96.30572	9.2775	9.130	8.575	1.8075	1.960

Tabla 8 Intervalos de valores recomendables de las métricas según la naturaleza de la entidad

	DOC	ESTRUCTURA			COMPLEJIDAD		
	J4	**J1 DIV J5**	**J6**	**J7**	**J2**	**J13**	**J14**
Normalidad Shapiro Wilk	0.02964	0.07905	6.875e-06	0.0474	0.1954	0.3288	0.08075
Igualdad de varianzas Barlett	0.3078	0.3823	0.005803	0.9698	0.8199	0.4119	0.2739
ANOVA una vía	0.7536	0.8609		0.09338	0.3621	0.5753	0.4707
ANOVA Kruskal Wallis			0.1330				

Tabla 9 Resultado de los contrastes estadísticos para cada una de las métricas para la validación externa

5. AGRADECIMIENTOS

Este trabajo ha sido financiado por el Ministerio de Ciencia e Innovación TIN2008-05675 cuyo título de proyecto es Detección de oportunidades de refactorización para la corrección de defectos de diseño y la evolución de sistemas mediante la generación y ejecución de planes de refactorización.

6. REFERENCIAS

[1] Pressman, R.S., Ingeniería del software: un enfoque práctico 6ª ed. 2005, McGraw-Hill Interamericana.
[2] Sommerville, I., Ingeniería del software. 7ª ed. 2005, Pearson Educación.
[3] IEEE, C.S., Guide to the Software Engineering Body of Knowledge: 2004 Edition - SWEBOK. 2005.
[4] Fenton, N.E. and S.L. Pfleeger, Software Metrics: A Rigorous and Practical Approach. 2nd ed. 1998: Course Technology. 656.
[5] ISO/IEC, Software engineering -- Product quality Part 1: Quality model. 2001.
[6] Marín, B., N. Condori-Fernández, and O. Pastor, Calidad en modelos conceptuales. Un análisis multdimensional de modelos cuantitativos basados en ISO 9126., in Revista de Procesos y MÉtricas. 2007.
[7] Chidamber, S.R. and C.F. Kemerer, A Metrics Suite for Object Oriented Design. IEEE Transactions on Software Engineering, 1994. 20: p. 476-493.
[8] Lorenz, M. and J. Kidd, Object-oriented software metrics: a practical guide, ed. I. Prentice-Hall. 1994, Upper Saddle River, NJ, USA.
[9] Martin, R., OO Design Quality Metrics. An Analysis of Dependencies. 1994.
[10] Brito e Abreu, F. and R. Carapuça, Candidate metrics for object-oriented software within a taxonomy framework. Journal of Systems and Software, 1994. 26(1): p. 10.
[11] McCabe, T., A Complexity Measure. IEEE Transactions on Software Engineering, 1976. 2: p. 308-320.
[12] Piattini Velthuis, M.G., Calidad en el desarrollo y mantenimiento del software / Mario G. Piattini Velthuis, Félix O. García Rubio, ed. F.O. García Rubio. 2002, Paracuellos de Jarama (Madrid) :: Ra-Ma. 310 p. ; 24 cm.
[13] Jacobson, I., G. Booch, and J. Rumbaugh, El proceso Unificado de Desarrollo del Software. 2000: Addison Wesley. 435 p. ; 25 cm.
[14] Arlow, J. and I. Neustadt, Uml 2 And The Unified Process: Practical Object-oriented Analysis And Design. 2005: Addison-Wesley Object Technology Series. 569.
[15] Dolado, J.J. and L. Fernández, Medición para la Gestión en la Ingeniería del Software. 2000: Ra-Ma.

16] Wohlin, C., et al., Experimentation in Software Engineering: An Introduction. 2000: Kluwer Academic Publishers.
[17] Campwood and Software, SourceMonitor. 2007.
[18] OPEN-SOURCE, R. 1997 - 2003.

Llamada a la participación Revista Procesos y Métricas

Uno de los principales objetivos de esta revista es que aquellas entidades y organizaciones interesadas **participen** en ella. Y por ello le instamos tanto a usted como a su institución para que realicen contribuciones, o envíen sus comentarios y sugerencias. Actualmente estamos abiertos a la recepción de trabajos para el próximo número que cubra los siguientes tópicos:

- Artículos académicos.
- Casos prácticos o casos de éxito (success histories) de aplicación de métricas y procesos de tecnologías de la información en organizaciones
- Artículos de difusión que traten conceptos teóricos, básicos, novedosos, etc… explicados de forma amena, o que traten aspectos relacionados con la difusión y empleo de ciertas técnicas, métricas o procesos.
- Información sobre noticias, eventos, etc…

La revista se edita en formato electrónico y papel (ambas con ISSN propio), y es accesible a través de su web: http://www.aemes.fi.upm.es/rpm/rpm.php. Consulte la 'Guía para Autores' publicada en este número o visite la web para más información sobre el formato de las contribuciones.
Esperamos sus contribuciones en: rpm@aemes.org

ASOCIACIÓN ESPAÑOLA DE MÉTRICAS DE SISTEMAS INFORMÁTICOS AEMES
Facultad de Informática de la UPM. Campus de Montegancedo. 28660-Boadilla del Monte (Madrid)
Teléfono: 91 336 66 08. Fax: 91 336 74 12. E-mail: admon@aemes.org

FORMULARIO DE INSCRIPCIÓN A AEMES

Información Personal
Apellidos: Nombre:
Teléfono: e-mail:
Dirección Personal:
Empleo:

Inscripción Institucional o Empresarial
Institución o Empresa: C.I.F.:
Dirección:

Datos Bancarios
Número de Cuenta:
Titular:

Autorizo a AEMES a cargar a la cuenta arriba indicada la cuota Anual de inscripción que asciende a 360 € + la cuota de inscripción que asciende a 150 €

Fecha y Firma

FORMULARIO DE INSCRIPCIÓN A RPM (Revista de Procesos y Métricas)

Información Personal
Apellidos: Nombre:
Teléfono: e-mail:
Dirección Personal:
Empleo:

Inscripción Institucional o Empresarial
Institución o Empresa: C.I.F.:
Dirección:

Datos Bancarios
Número de Cuenta:
Titular:

Autorizo a AEMES a cargar a la cuenta arriba indicada la cuota Anual de subscripción a RPM que asciende a 60 € + Gastos de Envío

Fecha y Firma

Guía para Autores

Se recomienda a los autores enviar los artículos electrónicamente utilizando la dirección de correo electrónico rpm@aemes.org . Por favor dirigir los artículos al Editor de la Revista de Procesos y Métricas de las Tecnologías de la Información o a la Asociación Española de Métricas de Sistemas Informáticos. El artículo debe ser enviado para el proceso de revisión en formato Microsoft Word o PDF.
En el caso de envíos de artículos en papel, se deben enviar tres copias al Editor de la Revista de Procesos y Métricas de las Tecnologías de la Información o a la Asociación Española de Métricas de Sistemas Informáticos. Facultad de Informática - Universidad Politécnica de Madrid, Campus de Montegancedo, Boadilla del Monte. Madrid - 28660.
Los artículos se deberán enviar sin indicar en el documento en el que se describa el trabajo presentado los autores del mismo. Para cada artículo enviado se deberá enviar en un documento adjunto el nombre y la filiación completa (incluida dirección, teléfono y correo electrónico) de los autores del artículo, y se indicará cual de ellos se deberá considerar como autor de contacto a efectos de comunicación.
El envío de un artículo implica que el trabajo descrito no ha sido publicado previamente (excepto en el caso de una tesis académica), que no se encuentra en ningún otro proceso de revisión, que su publicación es aceptada por todos los autores y por las autoridades responsables de la institución donde se ha llevado a cabo el trabajo y que en el caso de que el artículo sea aceptado para su publicación, el artículo no será publicado en ninguna otra publicación en la misma forma, ni en Español ni en ningún otro idioma, sin el consentimiento de la Asociación Española de Métricas del Software.
Una vez recibido un artículo se enviará al autor de contacto, por correo ordinario, una carta de recepción del artículo, tanto si este ha sido enviado por correo electrónico como si lo ha sido por correo ordinario.
Todos los artículos recibidos para ser considerados para su publicación serán sometidos a un proceso de revisión. La revisión será realizada por tres expertos independientes. Para asegurar un proceso de revisión lo más correcto posible los nombres de los autores y los revisores permanecerán confidenciales.
Una vez revisado un artículo se enviará por correo ordinario una carta con los resultados de la revisión, tanto si este ha sido enviado por correo electrónico como si lo ha sido por correo ordinario. En el caso de que el artículo haya sido rechazado se adjuntarán las valoraciones de los revisores.
El proceso de revisión está libre de costes para los autores.
Una vez que un artículo haya sido aceptado, se solicitará a los autores que transfieran los derechos de autor del artículo a la Asociación Española de Métricas de Sistemas Informáticos. Recibida la transferencia, se solicitará a los autores el envío de una versión del artículo lista para publicación que se deberá enviar en formato Microsoft Word.
La publicación de un artículo en la revista está libre de costes para los autores.

Guía para la preparación de manuscritos
El texto deberá estar escrito en un correcto castellano (Uso Español) o en Inglés (Uso Británico). Excepto el abstract que deberá estar escrito en un correcto Inglés (Uso Británico).
Abstract y Resumen. Se requiere un abstract en inglés con un máximo de 200 palabras. El abstract deberá reflejar de una forma concisa el propósito de la investigación, los principales y resultados y las conclusiones más importantes. No debe contener citaciones. Se debe presentar a continuación del abstract en inglés una traducción del mismo al castellano bajo el epígrafe Resumen.
Palabras clave. Inmediatamente después del Resumen se proporcionarán un conjunto de 5 palabras clave evitando términos en plural y compuestos, tampoco se deben usar acrónimos o abreviaturas a no ser que sean de un uso ampliamente aceptado en el campo del artículo. Estas palabras claves serán utilizadas a efectos de indexación.
Subdivisión del artículo. Después del Abstract y el Resumen, que no llevarán numeración, se debe dividir el artículo en secciones numeradas, comenzando en 1 y aumentando consecutivamente. Las subsecciones se numerarán 1.1 (1.1.1, 1.1.2, etc.), 1.2, etc. No se deben incluir subdivisiones por debajo del tercer nivel (1.1.1). Cada sección o subsección debe tener un título breve que aparecerá en una línea separada.
Apéndices. Si hay más de un apéndice, se deben identificar como A, B, etc. Las ecuaciones en los apéndices tendrán una numeración separada: (Eq. A.1), (Eq. A.2), etc.
Agradecimientos. Se deben situar antes de las referencias, en una sección separada.
Tablas. Se deben numerar las tablas consecutivamente de acuerdo con su orden de aparición en el texto. Se deben poner títulos a las tablas debajo de las mismas.
Figuras. Se deben numerar las figuras consecutivamente de acuerdo con su orden de aparición en el texto. Se deben poner títulos a las figuras debajo de las mismas.
Referencias. Se debe verificar que cada referencia citada en el texto se encuentra también en la lista de referencias y viceversa. Los trabajos no publicados o en proceso de revisión no pueden ser citados.
-*Citaciones en el texto*: Un solo autor. El primer apellido del autor, seguido de una coma y la primera inicial, seguida de un punto, a continuación, tras una coma, el año de publicación. Todo entre corchetes. Dos o más autores. Los nombres de los autores, siguiendo el formato de un solo autor, separados por puntos y comas y el año de publicación. Lista. Las listas deberán ser ordenadas, primero de forma alfabética y luego, si fuera necesario, de forma cronológica. Si hay más de una referencia del mismo autor en el mismo año deben ser identificadas por las letras “a”, “b”, etc., situadas después del año de su publicación.
-*Referencias*. Véase Volumen 1 Número 1 de esta publicación. Apartado 2.8.2.
Formato
-*Tamaño de la Página*: Deberá ser Carta (21,6 cm de ancho por 27,9 de largo). Las páginas irán sin numeración.
-*Tipo de Letra*: Deberá ser Times New Roman
-*Tamaño y Formato de la letra y el texto*: Título: 18 Negrita. Texto Centrado. Título de Sección: 14 Negrita. Alineación Izquierda. Espaciado Anterior y Posterior 12. Título de Subsección: 12 Negrita. Alineación Izquierda. Espaciado Anterior y Posterior 6. Título de Sub-Subsección: 12 Normal. Alineación Izquierda. Espaciado Anterior y Posterior 6. Texto: 12 Normal. Justificado. Espaciado Anterior y Posterior 0. Sangría en Primera Línea 1
-*Interlineado*: 1 Línea
-*Columnas*: 2. Todo el texto excepto el título, datos de los autores, abstract y resumen debe presentarse a 2 columnas

Socios Institucionales

ALCAMPO S.A.
ALI
ASOCIACIÓN TÉCNICA DE CAJAS DE AHORROS
ATOS ORIGIN
BANCO DE ESPAÑA
BBVA
CA IT MANAGEMENT SOLUTIONS SPAIN
CAELUM INFORMATION & QUALITY TECHNOLOGIES
CARE TECHNOLOGIES S.A.
CAST SOFTWARE ESPAÑA
C.E.C.A.
CENTRO DE CÁLCULO DE ALAVA, S.A.
COMPUWARE, S.A.
CORITEL, S.L.
DELOITTE, S.L.
EL CORTE INGLÉS
ENDESA SERVICIOS
EUROPEAN SOFTWARE INSTITUTE
EWORK-LINE TECHNOLOGY SERVICES, S.A.
GAS NATURAL INFORMÁTICA
GESEIN
IBERIA
ICM
IEE
INDRA SISTEMAS
INDRA SOFTWARE LABS, S.L.
INSA
IT·DEUSTO, S.L.
LA CAIXA
LEDA CONSULTING, S.L.
MTP
NESS PRO SPAIN
ONCE
OPTIMYTH SOFTWARE TECHNOLOGIES
SADIEL
SOFTWARE AG, ESPAÑA, S.A.
SOGETI ESPAÑA, S.L.
TECNOLOGÍA Y CALIDAD DE SOFTWARE, S.A.
TELEFÓNICA DE ESPAÑA, S.A.U.
UNIVERSIDAD CARLOS III
UNIVERSIDAD DE ALCALÁ DE HENARES
UNIVERSIDAD DE DEUSTO
UNIVERSIDAD OBERTA DE CATALUNYA
UNIVERSIDAD POLITÉCNICA DE MADRID
UNIVERSIDAD POLITÉCNICA DE VALENCIA


AEMES
Asociación Española
de Métricas de Sistemas
Informáticos
(N.I.F. G-81 76 73 86)