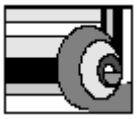

INSTITUTO POLITÉCNICO NACIONAL
CENTRO DE INVESTIGACIÓN EN COMPUTACIÓN

No. ___ **Serie:** Azul **Fecha:** Octubre 2005

Estimación del Esfuerzo de Desarrollo en Proyectos de *Software* a Nivel Personal con Base en Sistemas de Lógica Difusa

Cuauhtémoc López Martín[1]
Cornelio Yánez Márquez[2]
Agustín Gutiérrez Tornés[3]

RESUMEN

Se plantea la necesidad que existe en torno a la estimación del esfuerzo de desarrollo en proyectos de *software* comenzando su práctica a nivel personal. Se describen los tres tipos generales de técnicas de estimación existentes y se fundamenta la necesidad de comparación de resultados entre más de una de ellas. Se hace énfasis en la necesidad que existe por experimentar dentro del ámbito de la ingeniería de *software* y específicamente dentro del ramo de la estimación. Se justifica la necesidad de utilizar alternativas de modelos de estimación y se propone el uso de sistemas de lógica difusa para ello. Asimismo, se describe un estudio de caso demostrando que un sistema de lógica difusa puede representar una alternativa para estimar el esfuerzo de desarrollo.

Palabras clave: Estimación del esfuerzo de desarrollo, estudio empírico, lógica difusa.

[1] Estudiante del Doctorado en Computación, CIC
[2] Profesor del Centro de Investigación en Computación
[3] Coordinador de Sistemas, Banamex S.A

“Este reporte contiene información desarrollada por el Centro de Investigación en Computación del Instituto Politécnico Nacional a partir de datos y documentos con derechos de propiedad y por lo tanto su uso queda restringido a las aplicaciones que explícitamente se convenga.

La aplicación no convenida exime al Centro de su responsabilidad técnica y da lugar a las consecuencias legales que para tal efecto se determinen.

Información adicional sobre este reporte podrá obtenerse recurriendo a la Unidad de Publicaciones y Reportes Técnicos del Centro de Investigación en Computación del I.P.N. Av. Juan de Dios Bátiz s/n, teléfono 729-60-00 ext. 56403, 56608 y 56610”.

INDICE

Índice de Tablas y Figuras

INTRODUCCIÓN

De acuerdo con el Programa Nacional de la Industria del Software, el 98% de las empresas mexicanas (catalogadas como *emergentes*) no tienen procesos formales para registrar, dar seguimiento y controlar aspectos mensurables durante el proceso de desarrollo de sus productos (Secretaría de Economía, 2002), lo que tácitamente indica que, entre otras prácticas, existe una necesidad por la estimación del esfuerzo de desarrollo, actividad que debería comenzar desde un nivel personal desarrollando pequeños programas para que los desarrolladores puedan integrarse con menor problema a equipos de trabajo y posteriormente a organizaciones altamente maduras en sus procesos.

El CMM (*Capability Maturity Model*, Modelo de Madurez de la Capacidad) es la mejor descripción disponible de metas, métodos y prácticas necesarias para la industria de la ingeniería de *software*; por ello, las empresas necesitan comprender el CMM y saber como aplicar sus principios (Humphrey W., 2002). Los desarrolladores primero deberían conocer dichos principios individualmente. La pregunta sería, ¿cómo comprender el CMM a nivel personal dentro de un entorno de investigación? La respuesta es otra tecnología (además del CMM y TSP, *Team Software Process*) creada por el Instituto de Ingeniería de *Software* (SEI) de la Universidad de Carnegie Mellon: el PSP (*Personal Software Process*, Proceso de *Software* Personal). Doce de las dieciocho áreas de proceso clave (*Key Process Areas*) del CMM están al menos parcialmente tratadas por el PSP. El proceso del PSP incluye planeación, desarrollo (diseño del algoritmo, revisión de diseño, codificación, revisión de código, compilación, y pruebas), además de la etapa final denominada *postmortem*. Por lo que puede deducirse del presente párrafo, que las prácticas del PSP podrían coadyuvar en la experimentación dentro de laboratorios de investigación en los que se desee comparar técnicas de estimación a nivel personal.

En 1968, Alfred M. Pietrasanta del Instituto de Investigación de Sistemas de IBM mencionó lo siguiente: "*Quien espera una rápida y fácil solución para el problema de la polifacética estimación se va a decepcionar*" (Jorgensen). Esta predicción sigue siendo una realidad ahora como lo era entonces. Incluso se ha mencionado que el tiempo de desarrollo no puede ser objetivamente predicho (Lewis J.P., 2001):), pero al mismo tiempo es una realidad que existe la necesidad de ello, por lo que encontrar alternativas cada vez más precisas, se mantiene como una ocupación constante en investigadores.

La precisión y predicción oportunas tanto del esfuerzo de desarrollo como de la calendarización requeridas para construir y/o mantener un sistema de *software* es una de las actividades más críticas en la administración de proyectos de *software* (Idri A. *et al*, 2003; Jorgensen M. *et al*, 2000). Por ello, la estimación del esfuerzo de desarrollo ha sido identificada como uno de los tres grandes desafíos para la ciencia computacional (Brooks F. P. Jr., 2003).

Las técnicas de estimación se usan para los siguientes propósitos dentro de las empresas: (a) presupuestación, (b) administración de riesgos, (c) planeación y control del proyecto y (d) análisis de mejoramiento de inversión (Boehm B. *et al*, 1998).

Dentro de las empresas, las consecuencias de un esfuerzo deficientemente estimado son, entre otras (a) deficiencia en la calidad de los productos, (b) clientes insatisfechos y (c) desarrolladores frustrados (Jorgensen M. *et al*, 2000).

Este informe comienza en su primera parte con un marco teórico que define las tres principales clasificaciones de técnicas para estimar el esfuerzo de desarrollo de *software*: Juicio Experto, Modelos Algorítmicos y Aprendizaje de Máquina (la tercera de ellas más reciente en su aparición en comparación con las otras dos); posteriormente plantea la necesidad de comparar los resultados de más de una técnica y expone los criterios que son utilizados por la mayoría de los investigadores de esta área para comparar la precisión de técnicas. De igual forma, expone la necesidad de experimentar dentro del ámbito de la ingeniería de *software* y específicamente dentro de la estimación. Continúa describiendo las peculiaridades de la medición del *software* y su rol dentro de la estimación; finaliza describiendo los tópicos estadísticos utilizados dentro de la estimación del esfuerzo y mencionando la razón por la que la lógica difusa puede representar una alternativa para la estimación.

Una segunda parte está dedicada al análisis de artículos que involucran a dos áreas del aprendizaje de máquina: lógica difusa y redes neuronales. Análisis que concluye con la ausencia de algún estudio empírico tal, que involucrara (a) replicación en su experimento ni (b) experimentos a nivel personal. Asimismo, se deduce que, dada la dificultad por encontrar datos reales en nuestro país, en primera instancia las redes neuronales quedarían descartadas mientras no se cuente con una ingente cantidad de datos que son necesarios para su entrenamiento.

La tercera parte está dedicada a la exposición de lineamientos que un experimento para estimar el esfuerzo de desarrollo de *software* a nivel personal, debería tener. Se describe qué características debería poseer la población de la cual se tomaría la muestra, qué criterio se establecería para la selección de la muestra, de qué serie de etapas debería consistir la administración del experimento y cuales serían los mecanismos tales que redujeran el sesgo en la investigación.

Una cuarta parte está dedicada a un estudio de caso (no replicado) dedicado a la estimación del esfuerzo de módulos de programas a nivel personal con base en sistemas de lógica difusa. Este experimento incluye la descripción de la fuente de datos y la generación de las ecuaciones estadísticas correspondientes. Con base en los resultados de correlación, se definen las reglas difusas y por último se exponen los resultados del experimento de los que se puede concluir que la lógica difusa puede ser una alternativa más para la estimación del esfuerzo de desarrollo de *software* a nivel personal.

1. MARCO TEÓRICO

1.1 Técnicas de Estimación

Diversas técnicas de estimación han sido propuestas e investigadas durantelos pasados 30 años (Mendes E. *et al*, 2002; Briand L.C. *et al*, 2000). Las investigaciones están dirigidas a (1) determinar cual técnica tiene la mayor precisión para predecir el esfuerzo y (2) proponer nuevas o combinadas técnicas que pudieran proveer mejores estimaciones. Estas técnicas se encuentran dentro de las siguientes tres categorías (Mendes E. *et al*, 2002):

1) **Juicio Experto**. Técnica ampliamente utilizada para derivar estimaciones basadas en la experiencia de expertos en proyectos similares. El significado de derivar una estimación de este tipo no es explícita y por lo tanto es no repetible. Sin embargo, aunque siempre es difícil de cuantificar, el juicio experto puede ser una herramienta de estimación efectiva por si misma o bien ajustando un factor dentro de un modelo algorítmico. Aunque la estimación experta es comúnmente usada, es probable que no sea por su precisión; de hecho, parece ser tan imprecisa como el uso de modelos formales. Sin embargo, el juicio experto puede ser especialmente útil y frecuentemente la única opción para compañías con ya sea deficiente experiencia documentada de proyectos desarrollados o bien por tener recursos limitados para el proceso de estimación (Molokken K. y Jorgensen M., 2004). El término *estimación experta* no esta claro y cubre un amplio rango de enfoques de estimación. Una característica común es, sin embargo, que los procesos *intuitivos* constituyen la mayor parte de la estimación (Jorgensen M. *et al*, 2000).
2) **Modelos algorítmicos**. A la fecha son los más populares en la literatura; intentan representar la relación entre el esfuerzo y una o más características del proyecto. El principal conductor de costo en tales modelos es usualmente tomado del tamaño del *software* (líneas de código, por ejemplo). Los modelos algorítmicos necesitan ser ajustados a circunstancias locales.
3) **Aprendizaje de máquina**. Estas técnicas han sido usadas en años recientes como complemento o alternativa al juicio experto y modelos algorítmicos. Entre estos aparecen las redes neuronales y la lógica difusa.

La ingeniería de *software* se preocupa por construir sistemas de *software* y productos dentro de las limitaciones de tiempo, recursos, tecnología, calidad y consideraciones del negocio. Los modelos, procedimientos y técnicas de estimación son componentes esenciales de la disciplina de la ingeniería de *software*. Conforme a los cambios presentados a través del tiempo, las técnicas de estimación deben por necesidad, igualmente cambiar (Boehm B. W. y Fairley R. E, 2000).

La estimación se puede ver desde tres perspectivas diferentes (Ahmed M. A. *et al*, 2005): (1) el problema de estimación (¿qué etapa del proceso se estimará: desarrollo, mantenimiento, etc.?), (2) el problema particular (¿qué producto se estimará?) y (3) la técnica de estimación a usar dentro de las tres mencionadas categorías generales.

1.2 Necesidad de Comparar Técnicas

La experiencia ha mostrado que para una situación determinada, no existe una mejor técnica de predicción sobre las otras. Algunos investigadores han encontrado que la estimación por analogía genera mejores resultados que la regresión, mientras que otros han reportado resultados opuestos (Idri, A. *et al*, 2003). Por lo tanto, ningún método o modelo debería preferirse sobre los otros. La clave está entonces en usar una variedad de métodos y herramientas y luego investigar las razones del porqué las estimaciones provistas por una, podrían diferir significativamente de aquellas provistas por otra (Boehm B. *et al*, 1998). Es decir, una importante directriz es combinar métodos de estimación (Jorgensen),

1.2.1 Criterios de evaluación para comparación de técnicas

Un criterio común para la evaluación de modelos de estimación es la Magnitud Relativa del Error (MRE) la cual se define como sigue (Briand L.C. *et al*, 1998):

$$MRE_i = \frac{|\ \text{Esfuerzo Real}_i - \text{Esfuerzo Estimado}_i\ |}{\text{Esfuerzo Real}_i}$$

El valor de la MRE se calcula para cada observación (programa) *i* cuyo esfuerzo es estimado. Una agregación del MRE sobre múltiples observaciones (*N*), se puede obtener a través de la Media de MRE (MMRE) como se muestra a continuación:

$$MMRE = \frac{1}{N}\sum_{i}^{N} MRE_i$$

Un criterio complementario es el nivel de predicción *l*. Pred(*l*) = k/N, donde *k* es el número de observaciones donde MRE es menor o igual a *l*, y *N* es número total de observaciones (programas desarrollados). De esta forma, Pred(20) da el porcentaje de proyectos que fueron estimados con un MRE menor o igual a 0.20.

En general, la precisión de una técnica de estimación es proporcional a Pred(*l*) e inversamente proporcional a la MMRE.

1.3 La importancia de la experimentación

Para hacer una investigación académica relevante, los investigadores deberían tratar sus teorías con practicantes en situaciones reales dentro de organizaciones reales (Avison D. *et al*, 1999).

En los pasados tres decenios, diversos estudios han llevado a cabo evaluaciones comparativas de la precisión de estimaciones con diferentes técnicas. Una principal razón de tener conclusiones tangibles mínimas, es que los estudios de evaluación son raramente replicados y son usualmente no redactados en una forma tal, que permita la comparación de resultados. Es por ello que solo se puede contar con una confianza limitada en las conclusiones a partir de estudios de uno a la vez. Como en cualquier otro experimento o campo empírico, la replicación es la clave para establecer la validez y generalización de resultados (Briand L.C. *et al*, 2000).

Un importante objetivo en la mayoría de las investigaciones empíricas dentro de la ingeniería de *software*, es la transferencia de la investigación a aplicaciones industriales. Dos importantes obstáculos para esta transferencia son la deficiencia en el control de variables en estudios de caso y la deficiencia en el realismo de experimentos controlados (Sjoberg D. *et al*, 2002).

Los investigadores prueban predicciones teóricas contra la realidad. Una comunidad gradualmente acepta una teoría si todos conocen hechos dentro de su dominio que puedan ser inferidos a partir de la teoría, y si esta ha superado numerosas pruebas experimentales y correctamente predice nuevos fenómenos. Los experimentos se ayudan con inducción: derivando teorías a partir de la observación.

Pudieran existir creencias falsas alrededor de la experimentación dentro del ámbito de la ingeniería de *software*. Por ejemplo que "El método científico tradicional no es aplicable", al respecto existe una refutación: "Para comprender los procesos de información, los científicos de la computación deben observar fenómenos, formular explicaciones y probarlas, luego entonces este es el método científico". El hecho de que dentro del campo de la ciencia computacional el objeto de estudio es información más que energía o materia, no hace la diferencia en la aplicabilidad del tradicional método científico. Las ciencias tradicionales prueban teorías y las exploran de manera iterativa debido a que las observaciones ayudan a formular nuevas teorías que puedan ser validadas más tarde, luego entonces un importante requerimiento para cualquier experimento es, sin duda, la repetitividad, que asegura que los resultados puedan ser verificados independientemente y de esta manera elevar el grado de confianza en los resultados. Esto ayuda a eliminar errores, engaños y fraudes (Tichy W. F, 1998).

1.4 Medición del *software*

La aplicación más común de las métricas de *software* es desarrollar modelos que predigan el esfuerzo que será requerido para completar ciertas etapas de desarrollo de un sistema de *software* (Gray A. R. y MacDonell S. G., 1997).

Hay dos amplios usos en la medición del *software*: evaluación y predicción. La medición es definida como un proceso por el cual números o símbolos son asignados a atributos de entidades en el mundo real de tal forma que los describen de acuerdo a reglas claramente definidas (Fenton N., 1994).

Desde el principio de la historia del desarrollo de *software*, fue de interés, como actividad dominante, el tamaño del código fuente. De esta manera Halstead, Chrysler y Boehm emplearon tanto análisis teórico como empírico para construir modelos iniciales para el tamaño de código fuente. A pesar de la disponibilidad de un amplio rango de medidas para el tamaño del producto software, las líneas de código fuente permanecen a favor en muchos modelos (MacDonell S.G., 2003; Mendes E. *et al*, 2002).

1.5 El rol de la estadística

1.5.1 Correlación y Coeficiente de Determinación

La **correlación** (***r***) es el grado en que dos conjuntos de datos están relacionados. El valor de la correlación *r* varía de -1.0 a +1.0. El coeficiente de correlación entre líneas de código y esfuerzo, por ejemplo, puede ser calculado utilizando la siguiente ecuación:

$$r = \frac{n\left[\sum(LOC \cdot E)\right] - \left[\left(\sum LOC\right)\left(\sum E\right)\right]}{\sqrt{\left[n\left(\sum LOC^2\right) - \left(\sum LOC\right)^2\right]\left[n\left(\sum E^2\right) - \left(\sum E\right)^2\right]}}$$

Donde *n* es el número de pares de observaciones, *LOC* son las líneas de código y *E* es el esfuerzo de desarrollo.

El **coeficiente de determinación** (***r²***) es la proporción de la variación en los valores observados en la variable de respuesta que es explicada por la regresión. El valor de ***r²*** es obtenido usando la siguiente ecuación:

$$r^2 = \frac{SCR}{SCT}$$

Donde:

Suma de Cuadrados de la regresión (SCR): es la variación en los valores observados de la variable de respuesta que es explicada por la regresión:

$$SCR = \sum(y' - y_{media})^2$$

Suma de Cuadrados Totales (SCT): es la variación en los valores observados de la variable de respuesta:

$$SCT = \sum(y - y_{media})^2$$

1.5.2 Regresión Lineal Simple y Múltiple

La mayoría de los métodos usados para desarrollar modelos de predicción son aquellos derivados a partir de la estadística inferencial. Entre sus ventajas está la relativa simplicidad en su formulación y su teoría de la probabilidad. Una línea recta bajo el modelo de mínimos cuadrados intenta encontrar la línea que minimice el error en la relación entre variable dependiente e independiente. La estructura de esta línea está normalmente expresada en forma de una ecuación.

La regresión lineal simple considera la relación que existe entre una variable dependiente y una independiente. Cualquier forma de regresión lineal es generalmente precedida por el uso de gráficas de dispersión y análisis de correlación para primero intuitivamente, y luego cuantitativamente, determinar el potencial de las relaciones que pudieran existir en los datos.

Cuando dos conjuntos de datos están fuertemente relacionados, es posible usar un procedimiento de regresión lineal para modelar esta relación. La ecuación de regresión lineal usando mínimos cuadrados y teniendo al esfuerzo como variable dependiente y a las líneas de código como independiente, es la siguiente: $E' = a + b(LOC)$

Donde:

$$b = \frac{n\left[\sum (LOC \cdot E)\right] - \left(\sum LOC\right)\left(\sum E\right)}{n\left(\sum LOC^2\right) - \left(\sum LOC\right)^2}$$

$$a = \frac{\sum E}{n} - b\frac{\sum LOC}{n}$$

Por otra parte, una ecuación lineal con más de una variable independiente se le denomina múltiple. Por ejemplo, una con tres variables independientes podría ser expresada como:

$$y = b_0 + b_1x_1 + b_2x_2 + b_3x_3$$

Donde b_0, b_1, b_2 y b_3 son constantes; x_1, x_2 y x_3 son variables independientes, y y es la variable dependiente.

Los valores de b_0, b_1, b_2 y b_3 de la ecuación de regresión múltiple pueden ser obtenidos resolviendo el siguiente sistema de ecuaciones lineales:

$$\begin{aligned}
\sum y &= nb_0 + b_1(\sum x_1) + b_2(\sum x_2) + b_3(\sum x_3) \\
\sum x_1y &= b_0(\sum x_1) + b_1(\sum x_1^2) + b_2(\sum x_1x_2) + b_3(\sum x_1x_3) \\
\sum x_2y &= b_0(\sum x_2) + b_1(\sum x_1x_2) + b_2(\sum x_2^2) + b_3(\sum x_2x_3) \\
\sum x_3y &= b_0(\sum x_3) + b_1(\sum x_1x_3) + b_2(\sum x_2x_3) + b_3(\sum x_3^2)
\end{aligned}$$

1.6 Lógica Difusa

Recientes técnicas computacionales no algorítmicas para estimar esfuerzo fueron vistos en los noventas. Los investigadores pusieron su atención en un conjunto de enfoques basados en *sof-computing*. Dentro de estos enfoques, la lógica difusa posee características tanto de representación lingüística que puede representar imprecisión en entradas y salidas, como proveyendo un enfoque basado en el conocimiento experto; con base en estas características entonces se construye el modelo de estimación (Ahmed M. A *et al*, 2005).

Una desventaja en los modelos estadísticos es la forma por la que ellos disminuyen la comprensión de cómo las variables, interacciones y transformaciones son añadidas. Este problema puede ser al menos parcialmente superado con el uso de la lógica difusa, la cual fue desarrollada por la insatisfacción de la lógica clásica de *todo o nada*. La afirmación central subyacente de este enfoque difuso es que las entidades en el mundo real simplemente no pertenecen a una sola categoría. Por ejemplo, un proyecto no es solo *pequeño*, *mediano* o *grande*, sino podría de hecho ser un *gran* proyecto en su mayoría pero también *mediano* en su minoría. Esto puede ser representado con un grado de pertenencia en una categoría lingüística particular (MacDonell S. G., 1996).

Todas las técnicas de estimación tienen una limitación importante, la cual se muestra cuando proyectos de *software* son descritos usando datos categóricos (en escala nominal u ordinal) tales como *pequeño, mediano, promedio* o *alto.* Un enfoque más comprensivo para tratar

con valores lingüísticos es precisamente usando la teoría de la lógica difusa (Idri, A. *et al*, 2002b).

Desde su origen por Zadeh en 1965, la lógica difusa ha sido objeto de importantes investigaciones. A principio de los noventas, la lógica difusa se mantuvo firme en términos de sus fundamentos teóricos y aplicaciones en diversos campos en los cuales estuvo siendo usado, tales como la robótica, medicina y procesamiento de imágenes. Un conjunto difuso es un conjunto con una función de membresía *m* en el intervalo real [0, 1]. Esta definición extiende aquella del conjunto clásico donde la función de membresía esta en el par {0, 1}. Los conjuntos difusos pueden ser efectivamente usados para representar valores lingüísticos tales como *bajo*, *mediano* o *complejo*. La representación por un conjunto difuso tiene las siguientes ventajas: (a) es más general, (2) imita la forma en la que la mente de los humanos interpretan los valores lingüísticos y (3) la transición de un valor lingüístico a un valor lingüístico continuo es gradual más que abrupto (Idri, A. *et al*, 2003).

Hay un número de formas para la *defuzificación* de datos que podrían ser aplicadas al problema de la estimación del esfuerzo. Una de ellas es construir un sistema de inducción de reglas reemplazando los hechos *crisp* con entradas difusas, luego una máquina de inferencia usa las reglas para corresponder la entradas a la salida difusa, la cual puede ser traducida a partir de un valor *crisp* (Schofield C., 1998).

2. TRABAJO RELACIONADO

Pocos estudios han sido replicados y raramente las técnicas son investigadas con conjuntos de datos más allá de aquellos que fueron hace tiempo ya propuestos. En general, las evaluaciones de la precisión de la predicción de esfuerzo no solo revelan las fortalezas y debilidades de los métodos de estimación, sino también las ventajas y limitaciones de los conjuntos de datos usados.

Dos muestras de artículos relacionados con aprendizaje de máquina involucrando redes neuronales y lógica difusa en la estimación del esfuerzo de desarrollo de *software* fueron analizadas:

Una primera muestra de artículos ha sido revisada con la finalidad de encontrar investigación empírica replicada sobre estimación del esfuerzo de desarrollo a nivel personal basada en lógica difusa. Ningún artículo involucrando replicación fue encontrado en (a) Musflek *et al*, 2000 (b) Huang X. *et al*, 2004 (c) Idri, A. *et al*, 2002b, (d) Gray, A. y MacDonell, 1997 y 1999 (e) Braz, M. y Vergilio, S., 2004 (f) Ahmed, M. A. *et al*, 2005 y (g) Xu Z. y Khoshgoftaar, T. M., 2004.

Asimismo el trabajo (López-Martín Cuauhtémoc *et al*, 2005b) considera lógica difusa y prácticas a nivel personal (sugeridas por el PSP), pero no dentro de un experimento replicado.

Una muestra más ha sido revisada con la finalidad de encontrar investigación empírica replicada sobre estimación del esfuerzo de desarrollo a nivel personal basada en redes neuronales. Ningún artículo involucrando replicación ni a nivel personal fue encontrado en (a) Idri A. *et al*, 2002a (b) Hughes, 1993 (c) Srinivasan y Fisher, 1995 (d) Samson *et al*, 1993, (e) Wittig y Finnie, 1997 y (f) Verkatachalam, 1993.

El trabajo (López-Martín Cuauhtémoc *et al*, 2005a) involucra una red neuronal de regresión generalizada y prácticas a nivel personal (sugeridas por el PSP), pero no dentro de un

experimento replicado. Además, en este artículo se usaron 41 módulos de programas como entrada para el entrenamiento de la red, lo que representa un número reducido para una red neuronal. Es por esta razón que, en primera instancia, las redes neuronales quedan descartadas en tanto no se logre integrar la cantidad de datos requerida para su entrenamiento.

Un artículo adicional de Briand L.C. (2000) involucra replicación, sin embargo, no usa lógica difusa ni redes neuronales para estimar el esfuerzo.

3. DISEÑO EXPERIMENTAL

Un experimento para la estimación del esfuerzo de desarrollo de *software,* debería considerar las siguientes seis directrices: (1) contexto experimental, (2) diseño experimental, (3) conducción del experimento y recolección de datos, (4) análisis, (5) presentación de resultados e (6) interpretación de resultados (Kitchenham B. A. *et al*, 2002).

Los estudios empíricos tienen una amplia variedad de tipos, haciendo uso de una variedad de diseños experimentales. Uno de ellos es el *estudio de proyectos replicados*. Estudios de este tipo emplean múltiples personas (o equipos de personas), todos ellos trabajando en el mismo proyecto o aplicación (Seaman C.B., 1999).

Es importante mencionar qué características deberían tener los integrantes que participarían en el estudio replicado, cual sería el criterio para la selección de la muestra, qué proceso se seguiría durante el experimento y cuales serían los mecanismos que reducirían un posible sesgo durante el experimento. En las siguientes secciones se detallan estos aspectos.

3.1 Población del experimento

La población a ser estudiada debería estar integrada de personas con al menos un año de experiencia en el desarrollo de *software* dentro de sus empresas (fuera de su formación académica) así como al menos un año de experiencia en el lenguaje de programación que utilice dentro del experimento (un año en el lenguaje de programación es el indicado como *nominal* por Boehm [Boehm B., 1981]).

3.2 Criterio para la selección de la muestra de la población

Seleccionar atributos (métricas) que describan adecuadamente a los programas es una tarea compleja. Los atributos deben ser relevantes para la estimación el esfuerzo. El problema es cómo identificar atributos que exhiban una significante relación con el esfuerzo. La solución adoptada por investigadores de la estimación es probar la correlación entre el esfuerzo y atributos (Idri, A., 2001). Un valor de $|r| > 0.5$ podría ser considerado como aceptable.

Un estudio puede no ser aleatorio cuando las muestras son pequeñas en los experimentos de *software* e incluso porque algunas veces la asignación aleatoria es algunas veces no factible (Kitchenham B. A. *et al*, 2002).

3.3 Proceso para administrar los tratamientos

Un proceso que se podría seguir es el sugerido por el PSP, que incluye las fases de planeación, desarrollo (diseño, revisión de diseño, codificación, revisión de código, compilación, pruebas) y postmortem (Humphrey W., 2002).

Cada persona debería desarrollar los mismos programas y la misma cantidad de ellos, preferentemente no más de un programa por día. Los desarrolladores deberían tener supervisión constante y disipar sus dudas inmediatamente por parte de quien dirige el experimento.

Los desarrolladores deberán usar los siguientes estándares y cuadernos de registro sugeridos por el PSP: estándar de codificación, estándar de cuantificación de código, estándar de tipos de defectos, sumario del plan del proyecto, cuaderno de registro de tiempos, cuaderno de registro de defectos, lista de revisión de código, lista de revisión de diseño, reporte de pruebas y propuesta de mejoramiento de proceso.

Una herramienta de software (como MATLAB) podría ser utilizada para elaborar los sistemas de lógica difusa. Incluyendo por ejemplo, como características, el tipo mamdani o sugeno, método *and*: *min*, método *or*: *max*; implicación: *min*, agregación: *max*, y *defuzyficación*: *centroide*.

3.4 Métodos para reducir el sesgo

Para reducir el sesgo en resultados, todos los desarrolladores deberían tener al menos un mínimo de experiencia en el desarrollo de *software* empresarial y una mínima experiencia en el lenguaje de programación en el que codificarían sus programas dentro del experimento. Ellos deberían seguir el mismo proceso de desarrollo siguiendo asimismo prácticas comunes reflejadas en el respeto a estándares y registro constante de datos. De igual forma, los integrantes deberían ser constantemente asesorados mientras se encuentran dentro del entorno del experimento. Por otra parte, un conjunto de programas debería ser común a todos ellos y cada desarrollador debería elegir su propio lenguaje de programación.

4. ESTUDIO DE CASO

El uso de sistemas difusos para estimar el esfuerzo de desarrollo a nivel personal es mostrado en un estudio de caso (López-Martín Cuauhtémoc *et al*, 2005b) publicado por el IEEE Computer Society Press. Se comienza por la descripción del origen de los datos, se calcula la correlación de ellos, se calcula la ecuación de regresión múltiple y su coeficiente de determinación, posteriormente se formulan las reglas difusas y las funciones de membresía, por último se comparan los resultados del modelo estadístico contra el difuso y se concluye que un sistema difuso con base en tres métricas de entrada (complejidad de McCabe, acoplamiento de Dhama y líneas de código) y una de salida (esfuerzo de desarrollo), puede ser una alternativa para la estimación del esfuerzo en proyectos de *software* a nivel personal.

4.1 Recolección de datos

Los diez programas sugeridos por Humphrey fueron usados para obtener los datos de prueba. Después del diseño de alto nivel de cada programa, cada uno de los requerimientos de los módulos fueron completamente comprendidos y su algoritmo codificado, compilado y probado.

Las métricas registradas por cada módulo fueron (a) número de líneas de código (*lines of code*, LOC), (b) Complejidad de Mccabe (*McCabe complexity*, MC), (c) Acoplamiento de Dhama (*Dhama coupling*, DC) y el tiempo de desarrollo (*development time*, DT). La tabla 1 muestra la aceptable correlación (con |r | > 0.5, ver sección 1.5.1) entre el tiempo de desarrollo (DT) y las tres métricas MC, DC y LOC (0.7078, -0.7051 y 0.5827, respectivamente) registradas de los módulos.

Pair	r	Pair	r
MC – DC	-0.3860	DT – MC	0.7078
MC – LOC	0.7653	DT – DC	-0.7051
DC – LOC	-0.4346	DT – LOC	0.5827

Tabla 1. Correlación de métricas

41 módulos distribuidos dentro de los diez programas conformaron la muestra. Todos los programas fueron codificados en Pascal, por lo que los tipos de módulos fueron procedimientos (*procedure*) o funciones (*function*). Las estadísticas resultantes son descritas en la tabla 2.

4.2 Ecuación de regresión múltiple

Resolviendo el sistema de ecuaciones lineales usando las tres variables independientes (ver sección 1.5.2) y datos de la Tabla 2, la ecuación de regresión lineal múltiple es la siguiente:

$$DT' = 17.3097 + 2.06268*MC - 32.9405*DC - 0.0499692*LOC$$

4.3 Coeficiente de determinación (r^2)

Calculando r^2 (ver sección 1.5.2) resulta:

$$\mathbf{r^2} = \frac{RSS}{TSS} = \frac{\sum(y' - y_{avg})^2}{\sum(y - y_{avg})^2} = \frac{389.73}{539.51} = 0.7223$$

Este alto porcentaje (72.23 %), es la variación en el tiempo de desarrollo observado por módulo, que puede ser explicada por la regresión. Por lo que las métricas de MC, DC y LOC son todas útiles para predecir el tiempo de desarrollo de *software* por módulo.

4.4 Reglas difusas

El término difuso usualmente se refiere a las técnicas y algoritmos para construir modelos difusos a partir de datos. Hay dos enfoques principales para obtener un modelo difuso a partir de datos (Zhiwei Xu Z. y Khoshgoftaar T.M., 2004):

1) El conocimiento experto en modo verbal que es traducido a un conjunto de reglas si-entonces. Un cierto modelo puede ser creado y los parámetros de esta estructura, tales como funciones de membresía y pesos de las reglas, pueden establecerse usando datos de entrada y salida.

	Module Description	MC	DC	LOC	DT
1.	Calculates t value	1	0.25	4	13
2.	Inserts a new element in a linked list	1	0.25	10	13
3.	Calculates a value according to normal distribution equation	1	0.333	4	9
4.	Calculates the variante	2	0.083	10	15
5.	Generates range square root	2	0.111	23	15
6.	Determines both minimum and maximum values from a sorted linked list	2	0.125	9	15
7.	Turns each linked list value into its *z* value	2	0.125	9	16
8.	Copies a list of values from a file to an array	2	0.125	14	16
9.	Determines the parity of a number	2	0.167	7	16
10.	Defines segment limits	2	0.167	8	18
11.	From two lists (*X* and *Y*), returns the product of all x_i and y_i values	2	0.167	10	15
12.	Calculates a sum from a vector and its average	2	0.167	10	15
13.	Calculates *q* value	2	0.167	10	18
14.	Generates the sum of vector components	2	0.2	10	13
15.	Calculates the sum of a vector values square	2	0.2	10	14
16.	Calculates the average of the linked list values	2	0.2	10	15
17.	Counts the number of lines of code including blanks and comments	2	0.2	15	13
18.	Prints values non zero of a linked list	2	0.25	10	12
19.	Stores values into a matrix	2	0.25	10	12
20.	Generates range square root	3	0.083	17	22
21.	Returns the number of elements in a linked list	3	0.125	11	19
22.	Calculates the sum of odd segments (Simpsons´s formula)	3	0.125	15	18
23.	Calculates the sum of pair segments (Simpsons´s formula)	3	0.125	15	19
24.	Generates the standard deviation of the linked list values	3	0.143	13	21
25.	Returns the sum of square roots of a list values	3	0.143	14	20
26.	Prints a matriz	3	0.143	14	21
27.	Calculates the sum of odd segments (Simpsons´s formula)	3	0.143	15	19
28.	Calculates the sum of pair segments (Simpsons´s formula)	3	0.143	15	20
29.	Calculates the average of linked list values	3	0.167	13	15
30.	Returns the sum of a list of values	3	0.167	14	13
31.	Generates the standard deviation of linked list values	3	0.2	18	19
32.	Prints a linked list	3	0.25	9	13
33.	Calculates gamma value (G)	3	0.25	12	12
34.	Calculates the average of vector components	3	0.25	17	12
35.	Calculates the range standard deviation	4	0.077	16	21
36.	Calculates beta1 value	4	0.077	31	21
37.	Returns the product between values of two vectors and the number of these pairs	4	0.111	16	19
38.	Counts commented lines	4	0.2	24	18
39.	Reduces final matrix (according to Gauss method)	5	0.143	22	24
40.	Reduces a matrix (according to Gauss method)	5	0.143	22	25
41.	Counts blank lines	5	0.2	22	18

Tabla 2. Descripción de módulos y sus métricas, MC: *McCabe Complexity*, DC: *Dhama Coupling*, LOC: *Lines of Code*, DT: *Development Time* (*minutos*)

2) Sin conocimiento previo acerca de un sistema bajo estudio es inicialmente usado para formular las reglas, y un modelo difuso se construye a partir de datos basados en un cierto algoritmo. Se espera que las reglas extraídas y las funciones de membresía puedan explicar el comportamiento del sistema. Un experto puede modificar las reglas o suministrar nuevas basadas en la propia experiencia.

Este estudio de caso se basa en el primer enfoque. Las reglas difusas son formuladas en la correlación de pares de datos. A partir de estos valores de *r* (tabla1), las siguientes seis reglas son derivadas:

1. Si *complejidad es baja (low)* y *tamaño(LOC) es pequeño (small)* entonces *el tiempo de desarrollo* (*DT)* es *bajo (low)*
2. Si *complejidad es promedio (average)* y *tamaño(LOC) es medio(medium)* entonces *el tiempo de desarrollo* (*DT) es promedio(average)*
3. Si *complejidad es alta (high)* y *tamaño (LOC) es grande (big)* entonces *el tiempo de desarrollo* (*DT) es alto (high)*
4. Si *el acoplamiento es bajo (low)* entonces *el tiempo de desarrollo* (*DT) es bajo (low)*
5. Si *el acoplamiento es promedio (average)* entonces *el tiempo de desarrollo* (*DT) es promedio(average)*
6. Si *el acoplamiento es alto (high)* entonces *el tiempo de desarrollo* (*DT) es alto (high)*

Las funciones de membresía (FM) de entrada y salida se muestran en la Tabla 3. Todas ellas son triangulares y sus parámetros escalares (a, b, c) son definidos como sigue:

$$FM(x) = 0 \text{ if } x < a$$
$$FM(x) = 1 \text{ if } x = b$$
$$FM(x) = 0 \text{ if } x > c$$

Inputs

Nombre de variable	Rango	Función de Membresía	Parametros		
			a	b	c
Complejidad de McCabe (MC)	1 – 7	Low	1	2	3
		Average	2	4	5
		High	4	6	7
Acoplamiento de Dhama (DC)	0 – 0.4	Low	0.24	0.31	0.40
		Average	0.05	0.17	0.33
		High	0.00	0.12	0.21
Líneas de código (LOC)	1 – 40	Small	3	8	15
		Medium	10	16	25
		Big	21	28	40

Output

Nombre de variable	Rango	Función de Membresía	Parametros		
			a	b	c
Tiempo de Desarrollo (DT)	1 – 27	Low	6.6	9.0	11.8
		Average	8.1	12.8	18.6
		High	14.0	20.0	27.0

Tabla 3. Características de las funciones de membresía

Las gráficas de las funciones de membresía correspondientes a la tabla anterior se muestran en la figuras 1(a), 1(b), 1(c) y 1(d).

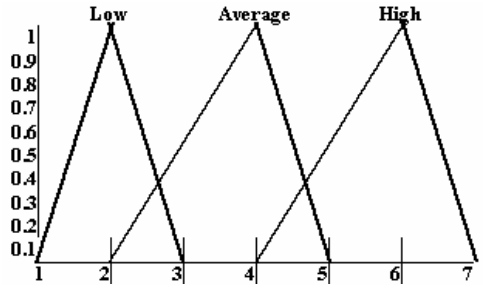

Figura 1(a). Gráfica de la complejidad de McCabe (*entrada*)

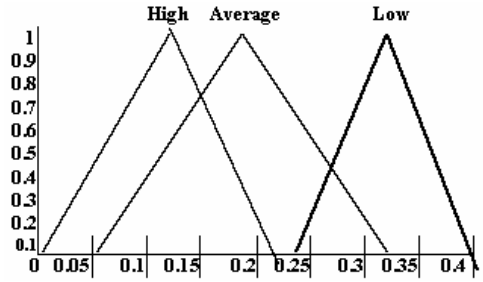

Figura 1(b). Gráfica del acoplamiento de Dhama (*entrada*)

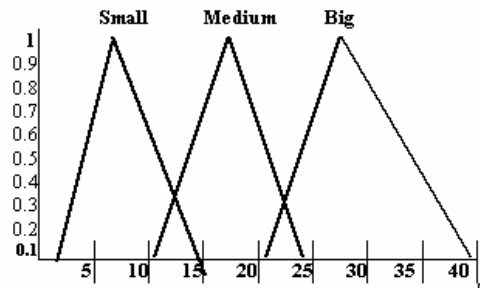

Figura 1(c). Gráfica de tamaño del módulo en líneas de código (*entrada*)

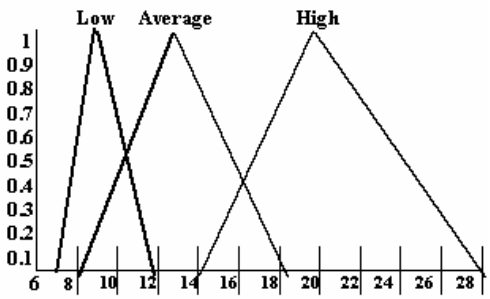

Figura 1(d). Gráfica del tiempo de desarrollo (*salida*)

4.5 Resultados experimentales

Tanto la ecuación de regresión múltiple y las reglas difusas descritas en la sección anterior fueron aplicadas al mismo conjunto de datos de entrada. Los resultados de la MMRE se describen en la tabla 4, mientras que los valores de Pred(20) son los siguientes:

	Multiple Regression	**Fuzzy Logic**
Pred(20)	0.9024	0.9268

La tabla 4 resalta en **negrita** aquellos valores de MRE que no mostraron desviación alguna. Asimismo, cada MRE cuyo valor es mayor a 0.20 se representa en *cursiva*. El comportamiento similar de los resultados de los dos modelos con respecto de los resultados reales es mostrado en la figura 2.

Como se observa en estos datos de 41 módulos generados dentro de diez programas, el valor de la MMRE aplicando lógica difusa es ligeramente mayor que la MMRE aplicando regresión lineal múltiple; mientras que el valor de Pred(20) aplicando lógica difusa resulta ligeramente mayor que el valor de Pred(20) aplicando regresión múltiple. Asimismo, seis de los 41 MRE resultan iguales a cero cuando un sistema de lógica difusa es aplicado

Módulo	DT Real	Regresión Múltiple		Sistema Difuso	
		DT´	MRE_i	DT´	MRE_i
1	13	10.9374	0.1587	13	**0.0000**
2	13	10.6376	0.1817	13	**0.0000**
3	9	8.2033	0.0885	9.15	0.0167
4	15	18.2013	*0.2134*	16.8	0.1200
5	15	16.6294	0.1086	17.8	0.1867
6	15	16.8678	0.1245	16.3	0.0867
7	16	16.8678	0.0542	16.3	0.0188
8	16	16.6179	0.0386	17.3	0.0813
9	16	15.5842	0.0260	15.6	0.0250
10	18	15.5342	0.1370	15.5	0.1389
11	15	15.4343	0.0290	15.6	0.0400
12	15	15.4343	0.0290	15.6	0.0400
13	18	15.4343	0.1425	15.6	0.1333
14	13	14.3473	0.1036	14	0.0769
15	14	14.3473	0.0248	14	**0.0000**
16	15	14.3473	0.0435	14	0.0667
17	13	14.0974	0.0844	15.1	0.1615
18	12	12.7002	0.0584	12	**0.0000**
19	12	12.7002	0.0584	12	**0.0000**
20	22	19.9142	0.0948	17.6	0.2000
21	19	18.8305	0.0089	17.6	0.0737
22	18	18.6306	0.0350	17.6	0.0222
23	19	18.6306	0.0194	17.6	0.0737
24	21	18.1376	0.1363	17.3	0.1762
25	20	18.0877	0.0956	17.3	0.1350
26	21	18.0877	0.1387	17.3	0.1762
27	19	18.0377	0.0506	17.2	0.0947
28	20	18.0377	0.0981	17.3	0.1350
29	15	17.3471	0.1565	16.7	0.1133
30	13	17.2971	*0.3305*	16.7	*0.2846*
31	19	16.0102	0.1574	15.2	0.2000
32	13	14.8129	0.1395	13	**0.0000**
33	12	14.6630	*0.2219*	13	0.0833
34	12	14.4131	*0.2011*	13	0.0833
35	21	22.2245	0.0583	17	0.1905
36	21	21.4750	0.0226	18.8	0.1048
37	19	21.1045	0.1108	17.2	0.0947
38	18	17.7731	0.0126	15.2	0.1556
39	24	21.8133	0.0911	17.3	*0.2792*
40	25	21.8133	0.1275	17.3	*0.3080*
41	18	19.9357	0.1075	15.2	0.1556

MMRE	**0.1005**		**0.1057**

Tabla 4. Comparación de las MMRE entre modelos de estimación

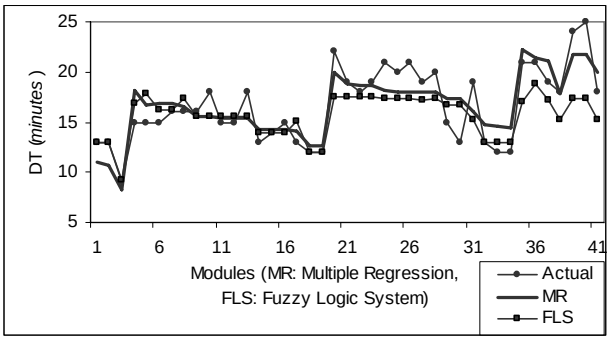

Figura 2. Comparación gráfica entre estimaciones de los modelos

CONCLUSIONES

En 1968, el comité de ciencia de la OTAN, se reunió para plantear los problemas en proyectos de *software* entonces existentes: los proyectos no terminaban dentro del lapso estimado, excedían los costos originalmente estimados y los requerimientos inicialmente especificados eran incompletos. Nació entonces el concepto de ingeniería de *software*, disciplina tecnológica y administrativa destinada a definir, desarrollar y mantener *software* con la finalidad de cumplir en tiempo, costo y calidad predeterminados.

La necesidad de estimar el esfuerzo de desarrollo (del cual los costos son derivados) se acentúo y comenzaron a surgir alternativas para ello. Además del juicio experto y los modelos algorítmicos, las alternativas que ofrece la inteligencia artificial han tomado auge.

El presente informe planteó la necesidad que existe en torno a la estimación del esfuerzo de desarrollo en proyectos de *software* comenzando su práctica a nivel personal. Se describieron los tres tipos generales de técnicas de estimación existentes y se fundamentó la necesidad de la comparación de resultados entre más de una de ellas. Se hizo énfasis en la necesidad que existe por experimentar dentro del ámbito de la ingeniería de *software* y específicamente dentro del ramo de la estimación. Se justificó la necesidad de utilizar alternativas de modelos de estimación y se propuso el uso de sistemas de lógica difusa para ello. Asimismo, se describió un estudio de caso ya publicado demostrando que un sistema de lógica difusa puede representar una alternativa para estimar el esfuerzo de desarrollo.

La necesidad de experimentar dentro del ámbito de la ingeniería de *software* se mantiene latente enfocando los esfuerzos a experimentos replicados en los que participen diversos desarrolladores a la vez. Con resultados de estos diseños experimentales, podrán comprobarse las teorías alrededor del uso de otras alternativas para estimar. Dentro de ellas el uso de sistemas de lógica difusa.

Al igual que el ser humano requiere de buenos hábitos para integrarse a grupos extra profesionales (deporte, comités, etc.), como profesionista del desarrollo de *software*, tiene la necesidad de llevar a cabo, en primera instancia, buenas prácticas a nivel personal, mismas que son requeridas para que puedan aspirar a pertenecer a organizaciones desarrolladoras de *software* de alta madurez.

FUENTES DE INFORMACIÓN

[1] Ahmed M. A, Saliu M.O., AlGhamdi J. (2005): Adaptive fuzzy logic-based framework for software development effort prediction. *Information and Software Technology, Vol. 47,* No. 1, pp. 31-48

[2] Avison D., Lau F., Myers M., Nielsen P.A. (1999): Action Research. *Communications of the ACM, Vol. 42, No. 1*

[3] Boehm B. W., Fairley R. E. (2000): Software Estimation *Perspectives IEEE Software* November/December, pp 22-26

[4] Boehm B., Abts Ch., Chulani S. (1998): Software Development Cost Estimation Approaches – A Survey. Chulani, Ph. D. Report

[5] Boehm B. (1981): *Software Engineering Economics*, Prentice Hall

[6] Braz, M., Vergilio S. (2004): Using Fuzzy Theory for Effort Estimation of Object-Oriented Software. *Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence*, ICTAI

[7] Briand L.C., Emam K.E., Surmann D., Wieczorek I. (1998): An Assesment and Comparison of Common Software Cost Estimation Modeling Techniques. *ISERN*

[8] Briand L.C., Langley T., Wieczorek I. (2000): A replicated Assessment and Comparison of Common Software Cost Modeling Techniques, *IEEE International Conference on Software Engineering (ICSE)*, Limerick, Ireland

[9] Brooks F. P. Jr. (2003): Three Great Challenges for Half-Century-Old Computer Science, *Journal of the ACM, Vol. 50,* No. 1 pp. 25-26, January

[10] Fenton N. (1994): Software Measurement A Necessary Scientific Basis, *IEEE Transactions on software engineering, Vol. 20, No. 3*, March, pp. 199-206

[11] Gray A. R., MacDonell S. G. (1997): Applications of Fuzzy Logic to Software Metric Models for Development Effort Estimation, *Proceedings of NAFIPS*

[12] Gray A.R., MacDonell S.G. (1999): Fuzzy Logic for Software Metric Models throughout the development Life-Cycle, *Proceedings of the Annual Meeting of the North American Fuzzy Information Processing Society - NAFIPS*. New York NY, USA, IEEE Computer Society Press (1999) 258-262.

[13] Huang X., Ren J. and Capretz L.F. (2004): A Neuro-Fuzzy Tool for Software Estimation, *Proceedings of the 20th IEEE International Conference on Software Maintenance, p. 520*, 2004

[14] Hughes R.T. (1996): An Evaluation of Machine Learning Techniques for Software Effort Estimation, University of Brighton

[15] Humphrey W. (2002): *A Discipline for Software Engineering* Addison Wesley

[16] Idri A., Khoshgoftaar T., Abran A. (2002a): Can Neural Networks be easily be interpreted in software Cost Estimation. World Congress on Computational Intelligence, Honolulu, Hawaii

[17] Idri, A., Abran, A., Khoshgoftaar T. M. (2003): Computational Intelligence in Empirical Software Engineering, *First USA-Morocco Workshop on Information Technology , ENSIAS, Rabat*

[18] Idri, A., Abran, A., Khoshgoftaar T. (2002b): Estimating Software Project Effort by Analogy Based on Linguistic Values, *Proceedings of the Eight IEEE Symposium on Software Metrics (METRIC)*

[19] Idri, A., Abran, A., Khoshgoftaar, T. (2001): Fuzzy Analogy: a New Approach for Software Cost Estimation, *International Workshop on Software Measurement (IWSM´01)*, Montréal, Québec, Canada, August 28-29
[20] Jonson R. A. (1997): *Probabilidad y Estadística para Ingenieros* Prentice Hall
[21] Jorgensen M. (2004): A Review of Studies on Expert Estimation of Software Development Effort, *Journal of Systems and Software* 70 (1-2): 37-60, 2004
[22] Jorgensen M., Kirkeboen G., Sjoberg D., Anda B., Brathall L. (2000): Human Judgement in Effort Estimation of Software Projects, *International Conference on Software Engineering*, Limerick, Ireland
[23] Jorgensen M., Practical Guidelines for Better Software Effort Estimation, Simula Research Laboratory. Available (2005): www.simula.no/departments/engineering/.artifacts/ieee_jorgensen_guidelines5.pdf
[24] Kitchenham B. A., Pfleeger S. L., Pickard L.M., Jones P. W., Hoaglin D. C., Emam K.E. Rosenberg J. (2002): Preliminary Guidelines for Empirical Research in Software Engineering, *IEEE Transactions on Software Engineering, Vol. 28, No. 8,* August 2002
[25] Lewis J.P. (2001): Large Limits to Software Estimation, *ACM Software Engineering Notes Vol 26, No. 4,* July, pages 54-59
[26] López-Martín Cuauhtémoc, Leboeuf J., Olivares J. (2005a): Software Development Effort Estimation Using a General Regression Neural Network: A Case Study, 12º Congreso Internacional de Investigación en Ciencias Computacionales, Monterrey, México, November
[27] López-Martín Cuauhtémoc, Leboeuf J., Yáñez Cornelio (2005b): Software Development Effort Estimation Using Fuzzy Logic: A Case Study, *Encuentro Internacional de Ciencias de la Computación, IEEE Computer Society Press,* September
[28] MacDonell S. G., Gray A. R. (1996): Alternatives to Regression Models for Estimating Software Projects, *Proceedings of the IFPUG Fall Conference*, Dallas TX
[29] MacDonell S.G. (2003): Software source code sizing using fuzzy logic modelling. *Elsevier Science*
[30] Mendes E., Mosley N. (2002): Watson I., A Comparison of Case-Based Reasoning Approaches to Web Hypermedia project Cost Estimation, *ACM*
[31] Molokken K., Jorgensen M. (2004): Group Processes in Software Effort Estimation. *Empirical Software Engineering*
[32] Musflek P., Pedrycz W., Succi G. (2000): Reformat M., Software Cost Estimation with Fuzzy Models, *Applied Computing Review, Vol. 8, No. 2,* pages 24-29.
[33] Park R. E. (1992): Software Size Measurement: A Framework for Counting Source Statements, *SEI, Carnegie Mellon University,* September
[34] Samson B., Ellison D., Dugard P. (1993): Software Cost Estimation using an Albus Perceptron. 8th International COCOMO Estimation Meeting, Pittsburgh
[35] Schofield C. (1998): Non-Algorithmic Effort Estimation Techniques, *ESERG*, TR98-01

[36] Seaman C.B. (1999):, Qualitative Methods in Empirical Studies of Software Engineering, *IEEE Transactions on Software Engineering, Vol. 25, No. 4*, July/August, pages 557-572
[37] Secretaría de Economía, (2002): *Programa para el Desarrollo de la Industria del Software*, June
[38] Simon, H. A. (1987): Making management decisions: The role of intuition and emotion, *Acad. Management Exec.1:* 57-63
[39] Sjoberg D., Anda B., Arisholm E., Dybå T., Jorgensen M., Karahasanovic A., Koren E.F., Vokác M. (2002): Conducting Realistic Experiments in Software Engineering, *Proceedings of the International Symposium on Empirical Software Engineering (ISESE'02)*
[40] Srinivasan K, Fisher D. (1995): Machine Learnig Aproaches to Estimating Software Development Effort- IEEE Transaction on Software Engineering, Vol. 21, no. 2, pp.126-136
[41] Tichy W. F. (1998): Should Coºmputer Scientists Experiment More? *IEEE Computer*, pp. 32-40
[42] Verkatachalam, (1993): Software Cost Estimation Using Artificial Neural Networks, IJCNN-IEEE
[43] Wittig G., Finnie G. (1997): Estimating Software Development Effort with connectionist Models. Information and Software Technology, vol. 39, pp. 469-476
[44] Zhiwei Xu Z., Khoshgoftaar T.M. (2004): Identification of fuzzy models of software cost estimation. *Elsevier Fuzzy Sets and Systems. Volume 145, Issue 1*, July, Pages 141-163