Modelo Neuronal de Estimación para el Esfuerzo de Desarrollo en Proyectos de Software (MONEPS)

Mario G. Almache C, Geovanny Raura, Jenny A. Ruiz R., Efraín R. Fonseca C.
Universidad de las Fuerzas Armadas ESPE
Sangolquí, Ecuador
mgalmache@espe.edu.ec

***Resumen*—La estimación temprana del esfuerzo para la construcción de un producto software, es crucial en la previsión del costo y tiempo necesarios para su desarrollo. Los modelos y técnicas para la estimación del esfuerzo presentan como principal inconveniente, la poca precisión en las predicciones realizadas y generalmente se hace una mínima consideración de los aspectos no funcionales del software. Se propone la construcción de un modelo de estimación para el esfuerzo en el desarrollo de software denominado MONEPS, que pretende mejorar la precisión en la estimación del esfuerzo, utilizando una Red Neuronal Artificial (RNA) en Backpropagation, cuya capa de entrada se estructura sobre la base de un conjunto de características y atributos tomados de la norma ISO/IEC 25000 de la calidad del software. La RNA fue entrenada con datos recopilados de aplicaciones desarrolladas en el ámbito académico, de las cuales se conocían sus tiempos de desarrollo y costos asociados. Las estimaciones de tiempo y costo, para dos casos de prueba, muestran más precisión en el modelo neuronal, en comparación con los modelos Cocomo-81 y Cocomo-II. MONEPS ha logrado la convergencia de aspectos funcionales y no funcionales para mejorar la precisión en la estimación del esfuerzo en proyectos de software.**

***Palabras Clave*—Software, Inteligencia Artificial, Redes Neuronales, Estimación del Esfuerzo.**

I. Introducción

La estimación del esfuerzo es un aspecto de particular importancia en la elaboración de los proyectos de software [1]. Las pequeñas y medianas empresas de software se enfrentan al desafío de seleccionar, de una parte, el método de estimación más adecuado para su contexto, y de otra parte, adoptar mejores prácticas que permitan consolidar una base histórica de estimación, con el propósito de disminuir cada vez más la brecha entre los valores estimados y los valores reales [2].

La estimación del esfuerzo para el desarrollo de software es una actividad compleja y de poca precisión en los resultados obtenidos [3], lo que ha ocasionado la aparición de varias herramientas comerciales para la estimación de costos del software, tales como [4]: Cocomo-II [5], CoStar [6], CostModeler [7], CostXpert [8], KnowledgePlan® [9], SEER [10], y SoftCost-R [11]. Por otra parte, existen algunas herramientas de estimación de costos que están tendiendo a la obsolescencia [12], debido principalmente a su temprana aparición; así tenemos: CheckPoint [13], ESTIMACS [14], REVIC [15] y SPQR/20 [16].

El número de herramientas existentes no significa que la problemática de la estimación del esfuerzo en proyectos de software haya sido solventada favorablemente, al contario ésta se ha incrementado, debido principalmente a la falta de mantenimiento de las herramientas y modelos disponibles [17].

En el presente artículo se presenta una propuesta de Modelo Neuronal de Estimación para el Esfuerzo de Desarrollo en Proyectos de Software (MONEPS), utilizando una RNA en Backpropagation, cuya capa de entrada se basa en un conjunto de atributos de naturaleza funcional y no funcional tomados de la norma de calidad del software ISO/IEC 25000 [18]. Esta RNA fue entrenada con datos de aplicaciones desarrolladas en el ámbito académico, de las que se conocían el tiempo empleado para su desarrollo y los costos asociados. La evaluación de MONEPS se llevó a cabo sobre dos aplicaciones específicas, obteniéndose mejoras significativas en las predicciones de tiempo y costo, en relación a los modelos Cocomo-81 [19] y Cocomo-II [20]. Este resultado evidencia la convergencia de los aspectos funcionales y no funcionales para mejorar la precisión en la estimación del esfuerzo en este tipo de aplicaciones.

El documento está organizado de la siguiente manera: En la Sección II se pone de manifiesto la importancia de mejorar la precisión en la estimación de proyectos de software. También se describen las principales categorías de los modelos de estimación para el esfuerzo, enfatizando en las técnicas de inteligencia artificial y en el modelo Cocomo-II. La Sección III presenta la propuesta neuronal para la estimación del esfuerzo en el desarrollo de proyectos de software, y se muestran los resultados obtenidos en la fase de entrenamiento de la RNA. En la Sección IV se muestra el desempeño de MONEPS cuando sus resultados son comparados con aquellos calculados a través de los modelos Cocomo-81 y Cocomo-II. Finalmente, en la Sección V, se mencionan las principales conclusiones de esta investigación y el trabajo futuro.

II. Antecedentes

A. Necesidad de mejorar la precisión en la estimación del esfuerzo

El estado actual del arte evidencia, por una parte, una falta de consenso al momento de seleccionar los atributos para la concepción de los modelos sobre los que se implementan las herramientas de estimación de software; por otra parte, las relaciones causales entre los atributos y el esfuerzo, tampoco están claras, lo cual se aprecia en la insatisfacción de los usuarios respecto a los resultados de estimación obtenidos [21].

Los resultados de la estimación del esfuerzo en software siguen siendo aún muy distantes de los valores reales, lo que conduce a reflexionar acerca del grado de adecuación que se consigue con las herramientas de estimación disponibles, respecto a las necesidades particulares de los diferentes tipos de proyectos [22].

Como se puede notar, la tarea de estimar esfuerzo en software no está exenta de dificultades; los modelos y

herramientas de estimación tradicionales enfrentan algunos problemas, como los que a continuación se exponen.

El principal inconveniente de los modelos predictivos basados en líneas de código (p.e. del tipo Cocomo), es que las líneas de código no pueden medirse hasta que el sistema esté completo [23]. Además, los modelos genéricos como Cocomo fallan a la hora de realizar ajustes de la predicción sin calibración. La necesidad de calibración se confirma en los estudios de Kemerer [24], Kitchenham & Taylor [25] y Low & Jeffery [26]. La poca atención de los aspectos no funcionales del software ha sido otra característica de los modelos predictivos para estimar el esfuerzo [27].

Frente a las dificultades que presentan los modelos tradicionales de estimación, han surgido nuevos enfoques que proponen la utilización de las técnicas de inteligencia artificial para mejorar las predicciones del esfuerzo; así podemos mencionar:

El clustering [28] es utilizado en la estimación del software para salvar el inconveniente de los modelos paramétricos que utilizan una única ecuación que representa a toda una base de datos de proyectos. Mediante el clustering los proyectos de software son divididos en grupos homogéneos, de tal manera que, un nuevo proyecto será asociado al grupo con el que tenga más coincidencias. Este enfoque requiere la disponibilidad de un gran conjunto de datos para establecer los grupos.

Chiu & Huang [29, 30] proponen mejorar la estimación del esfuerzo basados en analogía, con la ayuda de algoritmos genéticos y medidas de distancias (p.e. Euclidea, Manhattan, y Minkowski) entre parejas de proyectos. Este modelo calcula una distancia entre el proyecto de software que se estima y cada uno de los proyectos de software históricos, y a continuación, recupera el proyecto más similar para generar una estimación de esfuerzo. Luego, un algoritmo genético ajusta el esfuerzo reutilizado en función de las distancias de similitud entre pares de los proyectos. Los algoritmos genéticos, en general, requieren muchos recursos de cómputo para procesar las posibles soluciones.

Kumar [31] utilizó redes neuronales para predecir el esfuerzo en software con la ayuda de funciones de transferencia de Morlet y Gaussiana; estas funciones tienen un buen desempeño en RNA con más de dos capas ocultas. Más específicamente se utilizaron dos redes neuronales: WNN (Wavelet Neural Network) y TAWNN (Threshold Accepting Trained Wavelet Neural Network). Datos experimentales mostraron un mejor desempeño en la configuración WNN.

Jodpimai [32] hace una propuesta neuronal para predecir el esfuerzo, basado en la reducción de características del modelo Cocomo-81. La propuesta consta de tres pasos: a) preparación de datos tomados de dominios públicos; b) reducción del número de características, considerando sólo aquellas relevantes; c) transformar el problema de la estimación de esfuerzo del software en un problema de clasificación y aproximación funcional mediante el uso de una red neuronal en cascada. Lamentablemente, el modelo Cocomo-81 no ha podido acoplarse satisfactoriamente a la dinámica del software.

En el 2011, García [33] hace una propuesta neuronal para estimar el tiempo de duración en proyectos de software, recurriendo a un subconjunto de datos obtenido de la organización International Software Benchmarking Standards Group (ISBSG[1]); en dicha propuesta se filtró el conjunto de características disponibles, excluyendo aquellas referentes al cálculo de puntos de función.

En síntesis, las dificultades de los modelos tradicionales que inciden directamente en la poca precisión de las predicciones, requieren enfoques alternativos que permitan disponer de herramientas actuales y útiles para estimar costos y tiempos en el desarrollo de proyectos software. Esto propició la presente propuesta para la estimación del esfuerzo en software desde una perspectiva neuronal.

B. Modelos y técnicas de estimación para el esfuerzo en software

Las técnicas de estimación, pueden ser clasificadas bajo tres categorías principales [34]:

- ***Juicio del experto:*** un estimador de proyectos de software usa su experticia para estimar, basado en información histórica y en proyectos similares. El principal inconveniente de esta técnica es la dificultad de estandarización en los criterios de los expertos [35, 36].
- ***Modelos algorítmicos:*** es la categoría más popular en la literatura. Estos modelos incluyen Cocomo [37], SLIM [38] y SEER-SEM [39]. El factor principal de costo de estos modelos es el tamaño del software, que usualmente está asociado con las líneas de código. Estos modelos usan fórmulas de regresión lineal o también fórmulas de regresión no lineal. La desventaja de estos modelos radica en la necesidad de hacer ajustes a las predicciones, cuando los modelos no están calibrados [40].
- ***Aprendizaje de máquina:*** actualmente estas técnicas están siendo utilizadas en conjunción con los modelos algorítmicos, o como alternativas a éstos. Las técnicas de aprendizaje de máquina pueden incluir: lógica difusa [41], redes neuronales artificiales [42], minería de datos, sistemas neuro-difusos [43] y algoritmos genéticos [44].

La Tabla I muestra un resumen de las principales ventajas e inconvenientes para los enfoques de juicio del experto y modelos algorítmicos, para la estimación del esfuerzo en software.

TABLA I. PRINCIPALES VENTAJAS Y DESVENTAJAS EN DOS ENFOQUES PARA LA ESTIMACIÓN DEL ESFUERZO EN SOFTWARE

Enfoque	Ventajas	Inconvenientes	Aplicación idónea
Modelos algorítmicos	Entradas y parámetros concretos. Objetividad. Eficiencia en cálculos.	No prestan atención a circunstancias excepcionales. Rechazan opiniones subjetivas.	Proyectos con escasas alteraciones accidentales, con equipos de desarrollo estables y productos sencillos.
Juicio del experto	Gran cantidad de opiniones subjetivas. Consideración de circunstancias excepcionales.	Dependencia de los expertos. Posturas de expertos difíciles de adoptar.	Primeras fases de desarrollo del producto.

Por otra parte, varios modelos de estimación que se han publicado desde los años 60 han quedado obsoletos [45], así tenemos los modelos de: Aaron (1969), Wolverton (1974), Walston-Feliz (1977), Doty (1977), Putnam (1978), SLIM (1979), entre otros.

Cabe indicar que, el modelo Cocomo-81 (1981) y su actualización Cocomo-II (1997), han ido evolucionando para

[1] El sitio web de ISBSG está disponible en: http://www.isbsg.org/

constituirse como la base de las herramientas de estimación existentes en la actualidad [46].

En el 2012, P.K. Suri y Pallavi Ranjan [47] hacen un análisis de los métodos de estimación desde la década de los 70s, concluyendo que en los últimos 5 años se introdujeron diversos métodos en aras de incrementar la precisión de los resultados, tras la identificación de algunos problemas teóricos (Kitchenham, 2009). Para ello, la tendencia propone combinar diferentes métodos de estimación y utilizar a la par técnicas de inteligencia artificial, entre las que destacan: lógica difusa, sistemas basados en conocimiento y redes neuronales artificiales [48].

Por lo tanto, se cree conveniente aportar con una propuesta neuronal para estimar el esfuerzo de desarrollo en proyectos de software, basada en el uso de la norma ISO/IEC 2500 para la calidad de software. Esta propuesta no tiene una fórmula explícita con coeficientes preestablecidos para el cálculo del esfuerzo; tampoco precisa de líneas de código ni puntos de función (como es el caso del modelo Boehm [49] y el modelo Albretch [50]) en el proceso de estimación del esfuerzo. En contraposición, MONEPS necesita inputs (p.e. número de requisitos funcionales del proyecto, nivel de seguridad requerido, tipo de lenguaje de programación utilizado, etc.) fundamentados en la norma referida anteriormente, para estimar el costo y tiempo de desarrollo de un producto software. A través de MONEPS se muestra el mecanismo predictivo de las RNA's [51] para mejorar la precisión en la tarea de estimar el tiempo y costo para el desarrollo de software.

III. Propuesta Del Modelo Neuronal De Estimación Para El Esfuerzo De Desarrollo En Proyectos De Software (MONEPS)

A. Topología neuronal para MONEPS

La estructura neuronal seleccionada para MONEPS es de tipo Backpropagation. Estas redes neuronales artificiales (ver Figura 1) presentan algunas características [52, 53, 54], las cuales se indican a continuación:

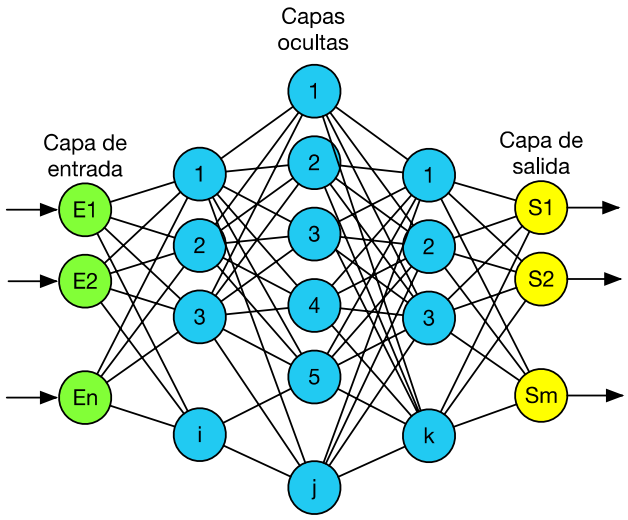

Fig. 1. Ejemplo de una red neuronal en cascada (Backpropagation)

- Aprenden de manera supervisada e inductiva.
- Son suficientes 3 capas (una de entrada, otra oculta, y una de salida) para las tareas de aprendizaje e identificación de patrones.
- No tienen mayor complejidad estructural ni algorítmica (lo que no sucede p.e. con las topologías recursivas de redes).
- Existe una buena disponibilidad de herramientas automatizadas, tanto libres como propietarias, para el diseño y funcionamiento de estas redes.
- Han sido utilizadas y probadas de manera satisfactoria en varios campos de aplicación (p.e. reconocimiento de imágenes, clasificación de patrones, codificación / decodificación de información, etc.).

B. Diseño neuronal de MONEPS

Para definir la topología de entrada del modelo neuronal se consideraron 42 atributos pertenecientes a características del estándar ISO/IEC 25000[2], que describen varios aspectos para la calidad de un producto software. Estos atributos fueron seleccionados en base a que consideran la tendencia actual hacia la mejora en el desarrollo de software [55]. A modo de ejemplo, en la Tabla II se describen cinco de los cuarenta y dos atributos seleccionados.

TABLA II. Tiemplos de atributos que forman parte de la capa de entrada para la RNA utilizada por MONEPS

Atributo/Código	Valores	Descripción
Nivel de seguridad (A#2)	Alto, Medio, Bajo	Indica el nivel de seguridad requerido para la aplicación.
Número de programadores (M#3)	1, 2, 3, ...	Número de integrantes del equipo de desarrollo asignados a esta actividad.
Facilidad de navegación (C#1)	Alta, Media, Baja	Indica la facilidad requerida para navegar en la aplicación.
Lenguaje de Programación (F#1)	Imperativo, Declarativo, Orientado a Objetos, Orientado al Problema	Tipo de lenguaje de programación utilizado.
Disponibilidad de arquitectos de software (M#2)	0, 1, 2, 3, ...	Arquitectos asignados al proyecto.

Por consiguiente, la capa de entrada tendrá en total 42 neuronas, mientras que la capa de salida tendrá solamente 2 neuronas (una para el tiempo y otra para el costo requerido).

En la Figura 2 se muestra un esquema parcial de la RNA utilizada en MONEPS. Como se puede observar en la red, las conexiones unidireccionales van desde la capa de entrada hasta la capa de salida (es decir, no hay ciclos), a través de una capa oculta que tiene 7 neuronas; esta capa posibilita el aprendizaje de la RNA. En la misma figura se muestran algunos atributos de la capa de entrada de la red. Cabe referir que, los atributos pueden ser de naturaleza cuantitativa (p.e. número de programadores asignados) o cualitativa (p.e. experiencia del equipo de desarrollo).

C. Implementación y resultados obtenidos en la fase de entrenamiento de la RNA

El diseño de MONEPS fue implementado en la herramienta JustNN[3]. Durante la fase de entrenamiento de la RNA fueron utilizados 9 proyectos académicos desarrollados por estudiantes pertenecientes a los últimos niveles de la carrera de Ingeniería en Sistemas e Informática de la Universidad de las Fuerzas Armadas ESPE de Ecuador. De cada uno de los proyectos académicos se obtuvo la información correspondiente a los 42 atributos de entrada de

[2] El sitio web de esta norma está disponible en: http://iso25000.com/

[3] Las caracteristicas de JustNN están disponibles en: http://www.justnn.com/

la RNA; así por ejemplo, se disponía de información como: nivel de seguridad, tiempo de respuesta requerido, nivel de escalabilidad de la aplicación, entre otros. Para poder estructurar las dos salidas de la RNA, se utilizó el tiempo de desarrollo y costo de cada proyecto académico referido.

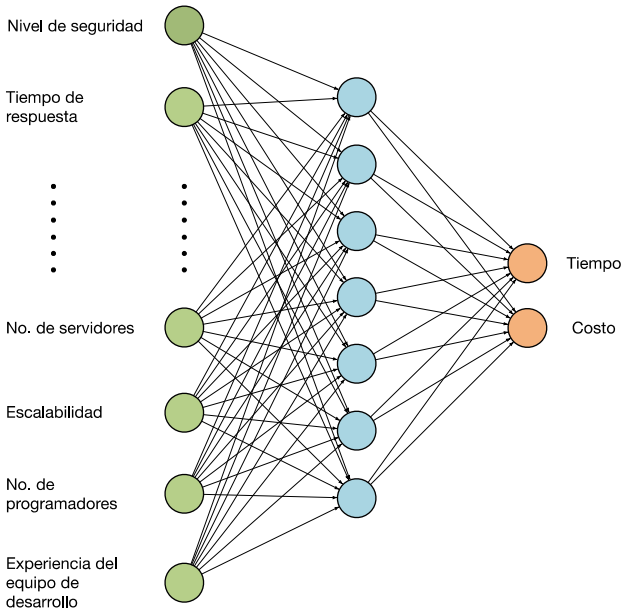

Fig. 2. RNA en Backpropagation usada en MONEPS

El funcionamiento de la RNA es similar al de una caja negra (en este caso con 42 entradas y 2 salidas); es decir, dentro de la caja negra la red aprende a configurar patrones de entrada/salida, actualizando los valores de los denominados "pesos sinápticos" o "conexiones neuronales", lo cual es transparente para el usuario. Luego del entrenamiento de la RNA utilizando la herramienta JustNN, se pudo constatar que la red tuvo un desempeño satisfactorio después de 44 ciclos (o épocas) de aprendizaje (ver Figura 3).

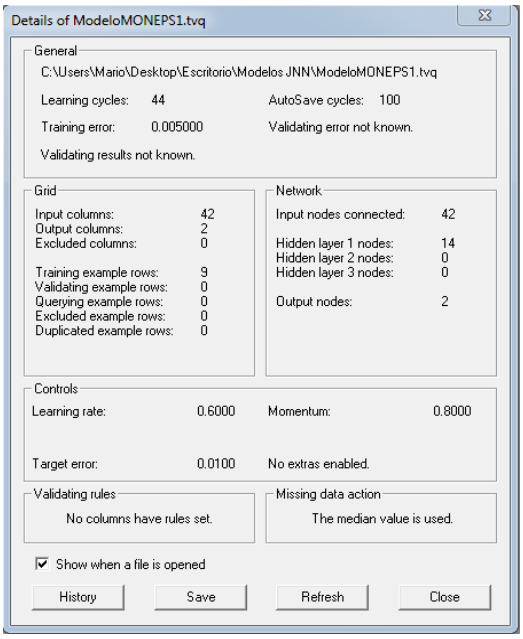

Fig. 3. Resultados obtenidos durante la fase de entrenamiento de la RNA

El error de la red durante la fase de entrenamiento estuvo en el orden del 0.5 %, lo que denota un rendimiento muy aceptable en relación al mínimo error requerido que es del 1.0%. En consecuencia, la red aprendió rápidamente a configurar patrones de comportamiento para tiempos y costos referidos a proyectos de software, lo que se puede ver en la Figura 4.

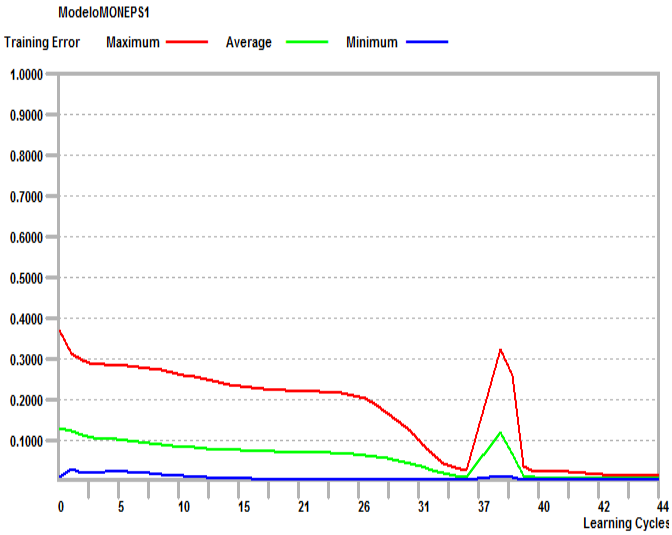

Fig. 4. Variación del error máximo, mínimo y medio durante el entrenamiento de la red

Adicionalmente, la Figura 5 muestra la importancia relativa (en valor porcentual) de cada atributo de entrada en la RNA. Por ejemplo, la *disponibilidad de un arquitecto de software* (atributo M#2) no es muy importante; al contrario, la *facilidad de navegación* (atributo C#1) es muy importante en la determinación del esfuerzo en los proyectos académicos.

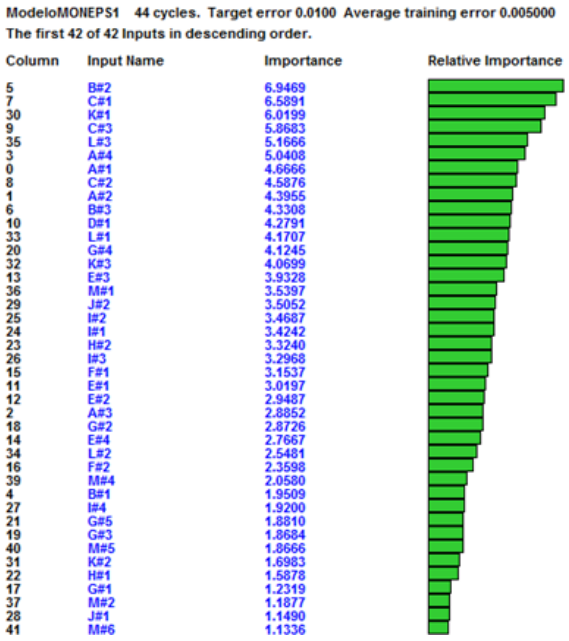

Fig. 5. Importancia relativa de cada atributo en la RNA

IV. Evaluación Del Modelo

El desempeño de la RNA de MONEPS está dado principalmente por su carácter predictivo, es decir, la capacidad de responder ante casos *que no ha visto*. En tal virtud, se estudiaron tres proyectos académicos de software; dos de éstos formaron el *conjunto de prueba* (proyectos académicos de software que no fueron considerados en la fase de entrenamiento de la RNA).

En la fase de evaluación, la RNA fue consultada sobre el tiempo y costo de desarrollo de los tres proyectos académicos referidos, para lo que se ingresaron (por cada proyecto) los 42

datos que precisa la red en su capa de entrada. Los resultados obtenidos de la evaluación se muestran en la Tabla III.

TABLA III. TIEMPO Y COSTO ESTIMADO PARA 2 CASOS DE PROYECTOS QUE LA RED NO HA VISTO. *TAMBIÉN SE PRUEBA CON UN EJEMPLO DE ENTRENAMIENTO

Caso	Tiempo real de duración (meses)	Tiempo estimado (MONEPS)	Costo referencial (USD)	Costo estimado (MONEPS)
1	3.00	3.65	7558.80	8139.45
2	5.00	3.97	8810.00	7273.05
3*	4.00	3.96	7244.00	7797.52

El primer proyecto evaluado corresponde a un simulador para la evaluación de aptitudes de aspirantes para el desarrollo de software, denominado Codesoft. El segundo proyecto es un sistema de facturación. El tercer proyecto se refiere a un sistema para control de fichas odontológicas.

Puede observarse que, los costos y tiempos estimados por MONEPS, son muy cercanos a los costos y tiempos reales. Para una mejor contrastación se ha creído conveniente realizar una comparación de resultados con respecto a los modelos Cocomo-81 y Cocomo-II, lo que se muestra en las Tablas IV y V, respectivamente.

TABLA IV. TIEMPOS ESTIMADOS APLICANDO COCOMO-81

Caso	Tiempo (meses)		
	Real	Estimado por Cocomo-81	Estimado por Moneps
1	3.0	6.85	3.65
2	5.0	7.75	3.97

TABLA V. TIEMPOS Y COSTOS ESTIMADOS EN COCOMO-II

Caso	Estimado por Cocomo-II	Estimado por Moneps	Estimado por Cocomo-II	Estimado por Moneps
1	12.20	3.65	10371.01	8139.45
2	10.20	3.97	12376.34	7273.05

La comparación con los modelos Cocomo-81 y Cocomo-II, permite apreciar que MONEPS mantiene una mejor aproximación al tiempo real de duración y costo. En la Tablas VI y VII se muestra el cálculo del error relativo para Cocomo-II, Cocomo 81 y MONEPS.

V. CONCLUSIONES Y TRABAJO FUTURO

La RNA utilizada por MONEPS aprendió rápidamente a configurar patrones de comportamiento para tiempos y costos referidos a proyectos de software, y por ende, las estimaciones realizadas son bastante cercanas a los costos y tiempos reales. Los resultados arrojados por la propuesta neuronal, en la fase de evaluación, mostraron mejor precisión respecto a los modelos Cocomo-81 y Cocomo-II, en la estimación de costo y tiempo para proyectos académicos de software. Más específicamente, el error relativo promedio en tiempo y costo, fue respectivamente, 10 y 3 veces menor en MONEPS con relación a Cocomo-II, para dos aplicaciones de prueba. Por otra parte, el error relativo promedio para el tiempo fue 4 veces menor en MONEPS con relación a Cocomo-81, para las mismas dos aplicaciones de prueba.

TABLA VI. TABLA VI. ERRORES RELATIVOS EN COCOMO-II Y MONEPS

Caso	Nombre del proyecto	Error relativo para el tiempo		Error relativo para el costo	
		Cocomo-II	Moneps	Cocomo-II	Moneps
1	CODESOFT	306.67%	21.67%	37.20%	7.68%
2	FACTURACIÓN	104.00%	20.60%	40.48%	17.45%

TABLA VII. ERRORES RELATIVOS EN COCOMO-81 Y MONEPS

Caso	Nombre del proyecto	Error relativo para el tiempo	
		Cocomo-81	Moneps
1	CODESOFT	128.33%	21.67%
2	FACTURACIÓN	55.00%	20.60%

La RNA tiene sus entradas fundamentadas en el uso de atributos del estándar ISO/IEC 25000, para la calidad de software. La asociación directa de estos atributos con la capa de entrada de la RNA, ayudó a simplificar la información que alimentó el modelo neuronal. El criterio de diseño para MONEPS, posibilitó la convergencia de aspectos funcionales y no funcionales en la estimación temprana de tiempo y costo para el desarrollo de proyectos de software de tipo académico. Sin embargo, es necesario validar la propuesta neuronal en otros proyectos de software fuera del ámbito académico.

MONEPS es de fácil uso y escalable; se pueden realizar los ajustes necesarios para mejorar el nivel de adecuación y precisión en la naturaleza dinámica del software. Tales ajustes se refieren básicamente a activar/desactivar atributos de entrada, y también a la alimentación de la RNA con datos de nuevas aplicaciones.

Como trabajo futuro se vislumbra la identificación de nuevos atributos críticos y métricas especializadas para los aspectos no funcionales, que podrían incidir notablemente en el nivel de precisión para las estimaciones realizadas por la propuesta neuronal. Asimismo, se pretende más adelante, añadir un componente fuzzy (difuso) para realizar interpretaciones lingüísticas de las entradas y respuestas obtenidas (sistemas neuro-difusos).

AGRADECIMIENTOS

Queremos expresar nuestro más profundo agradecimiento a todas aquellas personas que contribuyeron con su valioso aporte al desarrollo del presente trabajo. En especial a los estudiantes de los últimos niveles pertenecientes a la Carrera de Ingeniería en Sistemas e Informática en la Universidad de las Fuerzas Armadas ESPE. Agradecemos también los importantes comentarios y opiniones de los distinguidos docentes de la mencionada Carrera, en especial a los Ingenieros: Carlos Montenegro, Danilo Martínez, Raúl Córdova y Henry Coral.

REFERENCIAS

[1] Diaz Villegas, J. E., & Robiolo, G. (2014). Método de estimación de costos de un producto de software web. In XLIII Jornadas Argentinas de Informática e Investigación Operativa (43JAIIO)-XV Simposio Argentino de Ingeniería de Software (Buenos Aires, 2014).

[2] Moløkken-Østvold, K., Jørgensen, M., Tanilkan, S. S., Gallis, H., Lien, A. C., & Hove, S. W. (2004, September). A survey on software estimation in the Norwegian industry. In Software Metrics, 2004. Proceedings. 10th International Symposium on (pp. 208-219). IEEE.

[3] Omaña, M. (2010). Manufactura Esbelta: una contribución para el desarrollo de software con calidad. Enl@ce: revista Venezolana de Información, Tecnología y Conocimiento, 7(3), 11-26.

[4] Pressman, R. (2010). Ingeniería de Software. McGraw-Hill Interamericana de España.

[5] Boehm, B. W., Madachy, R., & Steece, B. (2000). Software cost estimation with Cocomo II with Cdrom. Prentice Hall PTR.

[6] Boehm, B. W., & Valerdi, R. (2008). Achievements and challenges in cocomo-based software resource estimation. Software, IEEE, 25(5), 74-83.

[7] Axelrad, V., Granik, Y., Boksha, V. V., & Rollins, J. G. (1994, September). Cost and yield estimation in a virtual IC factory. In Microelectronic Manufacturing (pp. 102-108). International Society for Optics and Photonics.

[8] Madachy, R. (2007). Distributed global development parametric cost modeling. In Software Process Dynamics and Agility (pp. 159-168). Springer Berlin Heidelberg.

[9] Jones, C. (1996). How software estimation tools work. American Programmer, 9, 18-27.

[10] Madachy, R., & Boehm, B. (2008). Comparative Analysis of COCOMO II, SEER-SEM and True-S Software Cost Models. USC-CSSE-2008-816, University of Southern California Center for Systems and Software Engineering.

[11] Southwell, S. V. (1996). Calibration of the Softcost-R Software Cost Model to the Space and Missile Systems Center (SMC) Software Database (SWDB) (No. AFIT/GSM/LAS/96S-6). AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF LOGISTICS AND ACQUISITION MANAGEMENT.

[12] Rodríguez, P., Martínez, H. (2010). Modelos Paramétricos de Estimación de Esfuerzo dentro del Proceso de Planificación de Proyectos Software. Revista de Procesos y Métricas, Vol. 7. Asociación Española para la Gobernanza, la Gestión y la Medición de las TI.

[13] Ferens, D. V. (1998, July). The conundrum of software estimation models. InAerospace and Electronics Conference, 1998. NAECON 1998. Proceedings of the IEEE 1998 National (pp. 320-328). IEEE.

[14] Boehm, B., Abts, C., & Chulani, S. (2000). Software development cost estimation approaches - A survey. Annals of Software Engineering, 10(1-4), 177-205.

[15] Webber, B. G. (1995). A Calibration of the Revic Software Cost Estimating Model (No. AFIT/GCA/LAS/95S-13). AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF LOGISTICS AND ACQUISITION MANAGEMENT.

[16] Cordero Carrasco, R. J. (2013). Una herramienta de apoyo a la estimación del esfuerzo de desarrollo de software en proyectos pequeños (Doctoral dissertation, Universidad de Chile).

[17] Mizell, C., & Malone, L. (2007). A project management approach to using simulation for cost estimation on large, complex software development projects.

[18] International Organization for Standardization & International Electrotechnical Commission. (2005). ISO/IEC 25000, Software product Quality Requirements and Evaluation (SQuaRE).

[19] Boehm, B. (2000). Safe and simple software cost analysis. IEEE software, (5), 14-17.

[20] Baik, J., Boehm, B., & Steece, B. M. (2002). Disaggregating and calibrating the CASE tool variable in COCOMO II. Software Engineering, IEEE Transactions on, 28(11), 1009-1022.

[21] López, J. E., & DOLADO COSÍN, J. J. (2008). Estimación del Esfuerzo Software: Factores vinculados a la aplicación a desarrollar. Actas de los Talleres de las Jornadas de Ingeniería del Software y Bases de Datos, 2(1).

[22] Robiolo, M. G., & Informáticas, C. (2009). Transacciones, Objetos de Entidad y Caminos: métricas de software basadas en casos de uso, que mejoran la estimación temprana de esfuerzo (Doctoral dissertation, Tesis Doctoral UNLP. http://postgrado. info.unlp.edu.ar/Carrera/Doctorado/Tesis%20Doctorales.html).

[23] Rodríguez, D., Pytel, P., Tomasello, M., Pollo Cattaneo, M. F., Britos, P. V., & García Martínez, R. (2011). Estudio del Modelo Paramétrico DMCoMo de Estimación de Proyectos de Explotación de Información. In XVII Congreso Argentino de Ciencias de la Computación.

[24] Kemerer, C. F. (1987). An empirical validation of software cost estimation models. Communications of the ACM, 30(5), 416-429.

[25] Kitchenham, B. A., & Taylor, N. R. (1984). Software cost models. ICL technical journal, 4(1), 73-102.

[26] Low, G. C., & Jeffery, D. R. (1990). Function points in the estimation and evaluation of the software process. Software Engineering, IEEE Transactions on, 16(1), 64-71.

[27] Hochstetter, J., Diaz, C., & Cares, C. (2012, June). Software call for tenders: Metrics based on speech acts. In Information Systems and Technologies (CISTI), 2012 7th Iberian Conference on (pp. 1-6). IEEE.

[28] Rubio, M. G. (2006). Aplicación de técnicas de clustering para la estimación del esfuerzo en la construcción de proyectos software (Doctoral dissertation, Universidad de Alcalá).

[29] Chiu, N. H., & Huang, S. J. (2007). The adjusted analogy-based software effort estimation based on similarity distances. Journal of Systems and Software, 80(4), 628-640.

[30] Huang, S. J., Chiu, N. H., & Chen, L. W. (2008). Integration of the grey relational analysis with genetic algorithm for software effort estimation. European Journal of Operational Research, 188(3), 898-909.

[31] Kumar, K. V., Ravi, V., Carr, M., & Kiran, N. R. (2008). Software development cost estimation using wavelet neural networks. Journal of Systems and Software, 81(11), 1853-1867.

[32] Jodpimai, P., Sophatsathit, P., & Lursinsap, C. (2010, March). Estimating software effort with minimum features using neural functional approximation. InComputational Science and Its Applications (ICCSA), 2010 International Conference on (pp. 266-273). IEEE.

[33] Garcia, A., Gonzalez, I., Colomo-Palacios, R., Lopez, J. L., & Ruiz, B. (2011). Methodology for software development estimation optimization based on neural networks. Latin America Transactions, IEEE (Revista IEEE America Latina), 9(3), 384-398.

[34] Peralta, M. (2004). Estimación del esfuerzo basada en casos de uso. Reportes Técnicos en Ingeniería de Software. Buenos Aires-Argentina, 6(1), 1-16.

[35] Páez Anaya, I. D. (2012). Estudio empírico del estado actual de la estimación de software en Pymes de Colombia.

[36] Robiolo, G., Castillo, O., Rossi, B., & Santos, S. (2013). Es posible superar la precisión basada en el juicio de expertos de la estimación de esfuerzo de productos de software? X Workshop Latinoamericano de Ingeniería de Software Experimental, ESELAW.

[37] Boehm, B. W. (1981). Software engineering economics (Vol. 197). Englewood Cliffs (NJ): Prentice-hall.

[38] Putnam, L. H. (1978). A general empirical solution to the macro software sizing and estimating problem. IEEE transactions on Software Engineering, 4(4), 345-361.

[39] Galorath, D. D., & Evans, M. W. (2006). Software sizing, estimation, and risk management: when performance is measured performance improves. CRC Press.

[40] Vizcaíno, A., García, F., & Piattini, M. (2014). Visión General del Desarrollo Global de Software. International Journal of Systems and Software Engineering for Big Companies.

[41] Lopez-Martin, C. (2011). A fuzzy logic model for predicting the development effort of short scale programs based upon two independent variables. Applied Soft Computing, 11(1), 724-732.

[42] De Barcelos Tronto, I. F., da Silva, J. D. S., & Sant'Anna, N. (2008). An investigation of artificial neural networks based prediction systems in software project management. Journal of Systems and Software, 81(3), 356-367.

[43] Huang, X., Ho, D., Ren, J., & Capretz, L. F. (2007). Improving the COCOMO model using a neuro-fuzzy approach. Applied Soft Computing, 7(1), 29-40.
[44] Afzal, W., & Torkar, R. (2011). On the application of genetic programming for software engineering predictive modeling: A systematic review. Expert Systems with Applications, 38(9), 11984-11997.
[45] Ruíz Constanten, Y., & Cordero Morales, D. (2013). Estimación en proyectos de software integrando los métodos de Boehm y Humphrey. Revista Cubana de Ciencias Informáticas, 7(3), 23-36.
[46] Páez, J. A. (2003). Avances en la toma de decisiones en proyectos de desarrollo de software (Doctoral dissertation, Universidad de Sevilla).
[47] Suri, P. K., & Ranjan, P. (2012). Comparative Analysis of Software Effort Estimation Techniques. International Journal of Computer Applications, 48(21).
[48] Isasi Viñuela, P., & Galván León, I. M. (2004). Redes de Neuronas Artificiales.Un Enfoque Práctico, Editorial Pearson Educación SA Madrid España.
[49] Boehm, B., Clark, B., Horowitz, E., Westland, C., Madachy, R., & Selby, R. (1995). Cost models for future software life cycle processes: COCOMO 2.0. Annals of software engineering, 1(1), 57-94.
[50] Albrecht, A. J., & Gaffney Jr, J. E. (1983). Software function, source lines of code, and development effort prediction: a software science validation. Software Engineering, IEEE Transactions on, (6), 639-648.
[51] Attarzadeh, I., Mehranzadeh, A., & Barati, A. (2012, July). Proposing an enhanced artificial neural network prediction model to improve the accuracy in software effort estimation. In Computational Intelligence, Communication Systems and Networks (CICSyN), 2012 Fourth International Conference on (pp. 167-172). IEEE.
[52] Pino Díez, R., Fuente García, D. D. L., Parreño Fernández, J., & Priore Moreno, P. (2002). Aplicación de redes neuronales artificiales a la previsión de series temporales no estacionarias o no invertibles.
[53] Russell, S. J., & Norvig, P. (2009). Artificial Intelligence: A Modern Approach. Third Edition. Prentice Hall.
[54] Warwick, K. (2013). Artificial intelligence: the basics. Routledge.
[55] Pesado, P., Bertone, R. A., Esponda, S., Pasini, A. C., Boracchia, M., Martorelli, S., & Rodriguez Eguren, S. (2012). Planes de Mejora, Mejora de Procesos de Gestión y Calidad en el desarrollo de Sistemas de Software. In XIV Workshop de Investigadores en Ciencias de la Computación.

Mario G. Almache C. Ingeniero en Informática, graduado en la Universidad Central del Ecuador, 1996. Es docente a tiempo completo en la Universidad de las Fuerzas Armadas ESPE, Ecuador. Sus áreas principales de interés para investigación son: machine learning, modelado matemático y simulación de sistemas dinámicos.

Jenny A. Ruiz R. Ingeniera en Sistemas e Informática, graduada en la Universidad Técnica de Ambato, Ecuador 1998. Con una maestría en Informática de la misma Universidad. Es docente a tiempo completo en la Universidad de las Fuerzas Armadas ESPE, Ecuador. Entre sus temas de investigación están: ingeniería de software, ingeniería de requisitos, estimación de proyectos de software.

Geovanny Raura. Es Ingeniero de Sistemas e Informática con un grado de maestría en Ingeniería de Software. Es profesor investigador a tiempo completo en la Universidad de las Fuerzas Armadas ESPE de Ecuador. Entre sus temas de investigación de interés están la ingeniería de software empírica, la ingeniería de requisitos, y técnicas de desarrollo de software basadas en metodologías ágiles

Efraín R. Fonseca. Se recibió como Doctor en julio de 2014. Tiene diez años de experiencia como consultor en la industria de las TI. Es profesor investigador en la Universidad de las Fuerzas Armadas ESPE de Ecuador. Entre sus temas de investigación de interés están: el proceso experimental en ingeniería de software, métodos de investigación en ingeniería de software empírica, análisis y diseño orientado a objetos y representaciones ontológicas en ingeniería de software.