ESPE

UNIVERSIDAD DE LAS FUERZAS ARMADAS

INNOVACIÓN PARA LA EXCELENCIA

VICERRECTORADO DE INVESTIGACIÓN
INNOVACIÓN Y TRANSFERENCIA DE TECNOLOGÍA

MAESTRÍA EN INGENIERÍA DE SOFTWARE

II PROMOCIÓN

TESIS DE GRADO MAESTRÍA EN INGENIERÍA DE SOFTWARE

TEMA: MODELO NEURONAL PARA LA ESTIMACIÓN DEL ESFUERZO EN PROYECTOS DE SOFTWARE

AUTORES: ING. MARIO GIOVANNY ALMACHE CUEVA

ING. JENNY ALEXANDRA RUIZ ROBALINO

DIRECTOR: ING. GEOVANNY RAURA RUIZ MSc

LATACUNGA

2015

UNIVERSIDAD DE LAS FUERZAS ARMADAS – ESPE
MAESTRÍA EN INGENIERÍA DE SOFTWARE

CERTIFICADO

ING. JORGE GEOVANNY RAURA RUIZ MSc.

CERTIFICA

Que el trabajo titulado MODELO NEURONAL PARA LA ESTIMACIÓN DEL ESFUERZO EN PROYECTOS DE SOFTWARE, realizado por el Ing. Mario Giovanny Almache Cueva e Ing. Jenny Alexandra Ruiz Robalino, ha sido guiado y revisado periódicamente y cumple normas estatuarias establecidas por la ESPE, en el Reglamento de Estudiantes de la Universidad de las Fuerzas Armadas-ESPE.

El presente trabajo trata de una investigación original. Propone un modelo para la estimación del esfuerzo en proyectos de software, basado en el uso de las características y atributos de la Norma ISO 25000 para la calidad del software, así como en la utilización de una red neuronal artificial en backpropagation. Los resultados del modelo neuronal muestran una mejora significativa en las estimaciones realizadas para tiempo y costo, respecto a los modelos Cocomo 81 y Cocomo II. Por lo expuesto se recomienda su publicación.

El mencionado trabajo consta de un documento empastado y un disco compacto el cual contiene los archivos en formato portátil de Acrobat (pdf).

Autoriza a los Ing. Mario Giovanny Almache Cueva e Ing. Jenny Alexandra Ruiz Robalino que lo entregue al Ing. Lucas Rogerio Garcés Guayta, en su calidad de Director del programa de postgrado.

En la ciudad de Latacunga, a los 14 días del mes de agosto del 2015

Ing. Jorge Geovanny Raura Ruiz MSc.

UNIVERSIDAD DE LAS FUERZAS ARMADAS – ESPE
MAESTRÍA EN INGENIERÍA DE SOFTWARE

DECLARACIÓN DE RESPONSABILIDAD

MARIO GIOVANNY ALMACHE CUEVA
JENNY ALEXANDRA RUIZ ROBALINO

DECLARAMOS QUE:

El proyecto de grado denominado MODELO NEURONAL PARA LA ESTIMACIÓN DEL ESFUERZO EN PROYECTOS DE SOFTWARE, ha sido desarrollado en base a una investigación exhaustiva, respetando derechos intelectuales de terceros, conforme las citas que constan el pie de las páginas correspondiente, cuyas fuentes se incorporan en la bibliografía. Consecuentemente este trabajo es de nuestra autoría.

En virtud de esta declaración, nos responsabilizamos del contenido, veracidad y alcance científico del proyecto de grado en mención.

En la ciudad de Latacunga, a los 14 días del mes de agosto del 2015

Ing. Mario Giovanny Almache Cueva

Ing. Jenny Alexandra Ruiz Robalino

UNIVERSIDAD DE LAS FUERZAS ARMADAS – ESPE

MAESTRÍA EN INGENIERÍA DE SOFTWARE

AUTORIZACIÓN

MARIO GIOVANNY ALMACHE CUEVA

JENNY ALEXANDRA RUIZ ROBALINO

Autorizamos a la Universidad de las Fuerzas Armadas la publicación, en la biblioteca virtual de la Institución del trabajo **"MODELO NEURONAL PARA LA ESTIMACIÓN DEL ESFUERZO EN PROYECTOS DE SOFTWARE**", cuyo contenido, ideas y criterios son de nuestra exclusiva responsabilidad y autoría.

En la ciudad de Latacunga, a los 14 días del mes de agosto del 2015

Ing. Mario Giovanny Almache Cueva

C.C 1708718950

Ing. Jenny Alexandra Ruiz Robalino

C.C 1802102101

AGRADECIMIENTO

Aprovechamos estas líneas para expresar nuestro más profundo y sincero agradecimiento a todas aquellas personas, que con su ayuda, han colaborado en la realización del presente trabajo, en especial al Ing. Geovanny Raura MSc., director de esta investigación, por la acertada supervisión y apoyo a la misma.

Un merecido reconocimiento al distinguido cuerpo docente de la Maestría en Ingeniería de Software, de la promoción II en Espe-Latacunga, quienes supieron compartir toda su valiosa experiencia y conocimiento con los maestrantes de dicha promoción.

Un sincero agradecimiento 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, de quienes se pudo recopilar la información necesaria para viabilizar este trabajo.

Finalmente, un agradecimiento muy especial merece la comprensión, paciencia y el ánimo recibidos de nuestras familias y amigos.

A todos ellos, muchas gracias.

DEDICATORIA

A mis hijos Alexander y Doménica, quienes son la causa que me motiva a soñar y luchar por esos sueños. A mi esposa Rosa, quien a cada momento está apoyándome y alentándome para seguir adelante. A mis padres, pues ellos sembraron en mí la semilla de la perseverancia para cumplir los más nobles objetivos

Mario G. Almache C.

DEDICATORIA

A mis dos amores: Mi esposo Xavier y mi hija Andrea, por ser el eje y motor de mi vida quienes se constituyeron en verdadera fuente de mi inspiración, en las largas jornadas durante y después del desarrollo de este trabajo. De forma especial a mis Padres quienes siempre estuvieron motivándome a continuar en el camino de aprender a ser un buen ser humano y mejor profesional a todos ellos mi agradecimiento especial.

Jenny A. Ruiz R.

ÍNDICE DE CONTENIDO

CAPÍTULO II

MÉTODOS CONVENCIONALES PARA ESTIMACIÓN DEL ESFUERZO DE DESARROLLO EN PROYECTOS DE SOFTWARE

CAPÍTULO III

REDES NEURONALES ARTIFICIALES

CAPÍTULO IV
DISEÑO DE MONEPS

CAPÍTULO V
IMPLEMENTACIÓN DE MONEPS

CAPÍTULO VI

CONCLUSIONES Y RECOMENDACIONES

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

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 algunos inconvenientes como: la poca precisión en las predicciones realizadas y generalmente se hace una mínima consideración de los aspectos no funcionales del software. Proponemos un nuevo 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-NORMA ISO/IEC 25000**
- **REDES NEURONALES ARTIFICIALES**
- **INTELIGENCIA ARTIFICIAL**

ABSTRACT

Early estimation of effort to build a software product is crucial to predict the cost and time required for its development. Models and techniques for estimating software effort have some disadvantages such as: inaccuracy in the predictions and minimal consideration of non-functional aspects of software. We propose a new model to estimate the effort in software development called MONEPS, which aims to improve the accuracy in estimating the effort using an Artificial Neural Network (ANN) in Backpropagation, whose input layer is based on a set of features and attributes taken from the standard ISO / IEC 25000 for software quality. The ANN was trained with data collected from applications developed in academic environment, of which their development time and associated costs were known. The forecasting of time and cost referred to two testing cases shows more accuracy in the neuronal model compared with Cocomo-81 and Cocomo-II models. MONEPS has achieved convergence of functional and non-functional aspects to improve accuracy of effort predictions in software projects.

Key Words:

- **SOFTWARE- STANDARD ISO/IEC 25000**
- **ARTIFICIAL NEURAL NETWORKS**
- **ARTIFICIAL INTELLIGENCE**

CAPÍTULO I

FUNDAMENTOS

1.1 Introducción

La estimación temprana del esfuerzo para la construcción de un producto software, es crucial en la previsión del costo y del tiempo necesarios para el desarrollo de software. 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. (Trendowicz, Adam, Jeffery, Ross, 2014)

Un proceso adecuado para la educción de requerimientos más un modelo de estimación apropiado, posibilitarán mejorar las predicciones para el desarrollo de software. A continuación se aborda un marco de referencia para la estimación de proyectos de software y el proceso de educción de éstos, que posibilitarán iniciar un proceso de estimación útil.

1.2 El problema de la estimación

La especificación de requerimientos es uno de los procesos más complejos e importantes que se realizan al inicio de todo proyecto de desarrollo de software, tal es así que, la observación de un mismo problema por varias personas puede provocar varias interpretaciones de un mismo término. Algunos problemas que se producen por la educción de los requisitos son: los usuarios no pueden o no saben describir muchas de sus tareas, mucha información importante no llega a verbalizarse; a veces se deben inventar los requisitos. La educción no debería ser un proceso pasivo, sino cooperativo que permita tener al menos la base de un levantamiento de información, de la cual se puede tener referentes muy importantes como insumos de un proceso de estimación [1].

Por otra parte, las empresas desarrolladoras de software no poseen un estándar del modelo de especificación de requerimientos que permita

asegurar la comprensión e identificación apropiada de los requerimientos, siendo éste el punto de partida para el inicio de la Ingeniería de Software en el desarrollo de un producto software. A su vez, los requerimientos se dividen en funcionales y no funcionales. Los requerimientos no funcionales de alguna manera se han convertido en los más críticos desde el punto de vista de los analistas de sistemas y los usuarios. Habría que considerar también que, aparte de los requerimientos iniciales, se procede a asociar(o adaptar) algún modelo conocido para la estimación del esfuerzo requerido en el desarrollo de un proyecto de software lo que permitirá tener algún indicio sobre el tiempo necesario para la culminación del mismo y, rara vez, indicios acerca de su costo (Trendowicz, Adam, Jeffery, Ross, 2014)

Se presenta una propuesta denominada Modelo Neuronal de Estimación para el Esfuerzo de Desarrollo en Proyectos de Software (MONEPS), basada en el uso de la tecnología de las Redes Neuronales, con la finalidad de lograr un mayor nivel de confiabilidaden en las predicciones de tiempo y costo inherentes al desarrollo de proyectos de software, dando también la importancia necesaria a los requerimientos no funcionales, que existen al momento de realizar el relevamiento de información para el inicio de un proyecto de desarrollo de software [2].

1.3 Objetivo general del proyecto

Construir un modelo de estimación para el esfuerzo de desarrollo en proyectos de software, que sea escalable y de fácil uso, utilizando las redes neuronales artificiales, para mejorar la precisión en las estimaciones de tiempo y costo necesarios para el desarrollo de software.

1.4 Objetivos especificos del proyecto

- Identificar las variables de entrada más relevantes para el modelo neuronal de estimación.
- Establecer una arquitectura neuronal óptima para el funcionamiento del MONEPS.

- Verificar la consistencia del modelo propuesto alimentándolo con muestras de proyectos académicos.
- Contrastar los resultados del modelo neuronal con lo datos obtenidos por otros modelos de estimación para el esfuerzo de desarrollo de software.
- Difundir los resultados obtenidos en el modelo neuronal a través de la elaboración de artículos técnicos.

1.5 ¿Qué es la estimación del esfuerzo en proyectos de software?

La definición de estimar, tomada de la Real Academia Española de la Lengua: "(Del lat. a estimare) Tr. Apreciar, poner precio, evaluar algo" [65]

En el ámbito de la Ingeniería de Software estimar no solamente debería limitarse a obtener un dato numérico o evaluar algo (según la definición de la Real Academia Española) sino que, debería ser un verdadero proceso que incluya los requerimientos funcionales y no funcionales. Por otro lado, los indicadores de acuerdo a la metodología de estimación utilizada, deben permitir la emisión de una cuantificación pero también una cualificación, lo más exacta y precisa posible, que permita una mejor toma de decisiones en este importante proceso de la Ingeniería de Software (Trendowicz, Adam, Jeffery, Ross, 2014) [3].

1.6 Requerimientos de software

Uno de los primeros procesos que se realizan en un proyecto de construcción de software es la especificación de requisitos de software. Los objetivos de este proceso son: identificar, validar y documentar los requisitos de software; es decir determinar las características que deberá tener el sistema o las restricciones que deberá cumplir para que sea aceptado por el cliente y los futuros usuarios del sistema de software.

El producto final de este proceso es el documento de especificación de requisitos de software y en éste se señala, con el detalle adecuado, lo que el usuario necesita del sistema de software. Es por ello que, el documento de requisitos de software, se considera como un contrato entre el cliente y el equipo de desarrollo del sistema.

En este punto valdría la pena citar los análisis de algunos autores respecto a los requisitos: Boehm, 1975: 45% de los errores tienen su origen en los requisitos y en el diseño preliminar. De Marco, 1984: 56% de los errores que tienen lugar en un proyecto software se deben a una mala especificación de requisitos.

Los factores principales que conducen al fracaso en los proyectos de software son: Falta de comunicación con los usuarios, requisitos incompletos, cambios a los requisitos. Aquí es donde juega un rol protagónico la Ingeniería de requisitos, trata de los principios, métodos, técnicas y herramientas que permiten descubrir, documentar y mantener los requisitos para sistemas basados en computadora, de forma sistemática y repetible.

Los requerimientos, para su mejor comprensión presentan la siguiente clasificación [4]:

1.6.1 Requerimientos Funcionales

Describen la funcionalidad o servicios que el sistema debe proveer, indican cómo el sistema debería reaccionar a un ingreso en particular y cómo el sistema debería comportarse en situaciones particulares.

1.6.2 Requerimientos No funcionales

Son requerimientos que no están directamente relacionados con funciones específicas que el sistema proveerá. Muchos de los requerimientos no funcionales se relacionan al sistema como un todo, muchas veces son más críticos que los requerimientos funcionales.

1.7 Educción de requisitos

Es el proceso a través del cual se obtienen los requerimientos; éste puede involucrar problemas como los siguientes:

- Los usuarios no pueden /saben describir muchas de sus tareas
- Mucha información importante no llega a verbalizarse.

- A veces hay que inventar los requisitos (sistemas orientados a miles de usuarios).
- La educción no debería ser un proceso pasivo, sino cooperativo

En este proceso de educción de requisitos se deben usar las siguientes técnicas preliminares [5]:

- Utilizar preguntas libres de contexto
- Brainstorming (Lluvia de ideas)
- Entrevistas: es el método tradicional
- Observación y análisis de tareas
- Escenarios: los requisitos se sitúan en el contexto de uso.
- Prototipado: útiles cuando la incertidumbre es total acerca del futuro sistema.

En la figura 1.1 se muestra, de forma esquematizada, el proceso de requisitos.

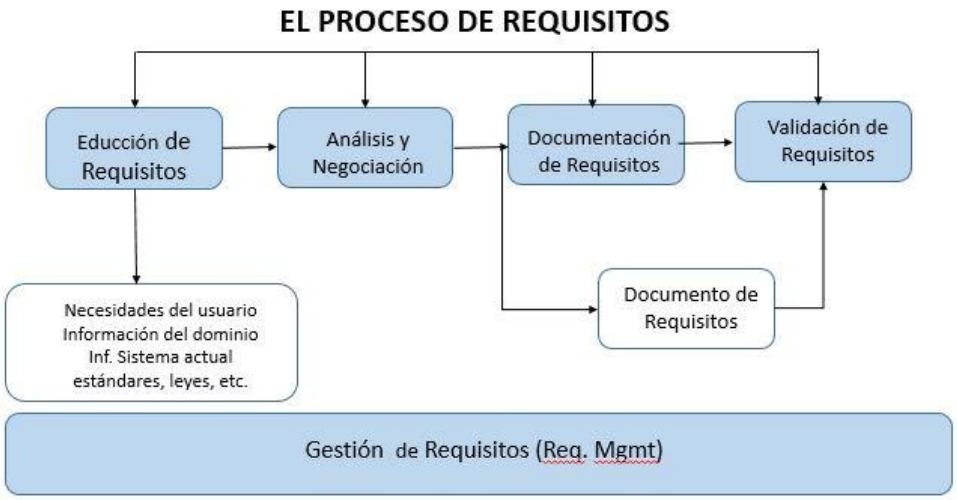


Figura 1.1 El proceso de requisitos

Fuente: [66]

Otros aspectos importantes a ser tomados en cuenta en el proceso de requisitos son:

- El proceso de educir requisitos puede descomponerse en las actividades: hallazgo de hechos, reunión e integración de la

información. La educción no debería ser un proceso pasivo, sino cooperativo.

- El análisis de requisitos consiste en detectar y resolver conflictos entre requisitos; se precisan los límites del sistema y la interacción con su entorno, se trasladan los requisitos de usuario a requisitos del software (implementables), se realizan tres tareas fundamentales: clasificación, modelización y negociación.

- Validación de requisitos: descubrir problemas en el documento de requisitos antes de comprometer recursos a su implementación; el documento debe revisarse para descubrir omisiones, conflictos, ambigüedades, comprobar la calidad del documento y su grado de adhesión a estándares.

- Documentación de requisitos: existen estándares previamente definidos proporcionados para este propósito, tal es el caso de la IEEE 830 que es un documento que sirve para el relevamiento de la información de un sistema de información [6].

- Gestión de Requisitos: consiste básicamente en gestionar los cambios en los requisitos; asegura la consistencia entre los requisitos y el sistema construido (o en construcción), consume grandes cantidades de tiempo y esfuerzo, abarca todo el ciclo de vida del producto.

1.8 Métrica

“Es una medida que proporciona una indicación cuantitativa de extensión, cantidad, dimensión, capacidad y tamaño de algunos atributos según un proceso o producto a evaluarse” [67]

1.8.1 Métricas de software

Una métrica es una medida efectuada sobre algún aspecto del sistema en desarrollo o del proceso empleado que permite, previa comparación con unos valores (medidas) de referencia, obtener conclusiones sobre el aspecto medido con el fin de adoptar las decisiones necesarias para obtener la calidad del sistema o proceso [7] [68].

1.8.2 Clasificación de las Métricas de Software

Las métricas de software se basan fundamentalmente en mediciones directas e indirectas, y se clasifican en:

- Métricas Técnicas. Se centran en la medición de las características de software, por ejemplo, la complejidad lógica, el grado de modularidad, la estructura del sistema, etc.
- Métricas de Calidad. Proporcionan una indicación de cómo se ajusta el software a los requisitos implícitos y explícitos del cliente, es decir, cómo el sistema está cumpliendo los requisitos del cliente.
- Métricas de productividad. Se centran en el rendimiento del proceso de la Ingeniería de Software, es decir, que tan productivo será el software a diseñarse.
- Métricas orientadas a la persona. Proporcionan medidas e información sobre la forma que la gente desarrolla el software, y sobre todo, el punto de vista humano de la efectividad de las herramientas y métodos. Son las medidas realizadas sobre el personal que desarrolla el sistema.
- Métricas orientadas al tamaño. Permiten conocer el tiempo en el cual se culminaría un software y el personal a requerirse. Son medidas directas sobre el software y el proceso a desarrollarse.
- Métricas orientadas a la función. Son medidas indirectas del software y del proceso por el cual se desarrolla. En lugar de

calcular las líneas de código (LDC), las métricas orientadas a la función se centran en la funcionalidad o utilidad del programa [8].

CAPÍTULO II

MÉTODOS CONVENCIONALES PARA ESTIMACIÓN DEL ESFUERZO DE DESARROLLO EN PROYECTOS DE SOFTWARE

2.1 Introducción

Las ***técnicas de estimación***, pueden ser clasificadas en tres principales categorías [9]:

- Juicio del experto: un estimador de proyectos de software usa su experticia basada en información histórica y en proyectos similares para estimar. El principal inconveniente de esta técnica es la dificultad de estandarización en los criterios de los expertos.
- Modelos algorítmicos: es la categoría más popular en la literatura; esos modelos incluyen COCOMO, SLIM y SEER-SEM. El factor principal de costo de estos modelos es el tamaño del software, que usualmente está dado en 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.
- Aprendizaje de máquina: actualmente, estas técnicas están siendo utilizadas en conjunción o como alternativas a los modelos algorítmicos. Estas técnicas pueden incluir lógica difusA, redes neuronales, minería de datos, sistemas neuro-difusos, algoritmos genéticos.

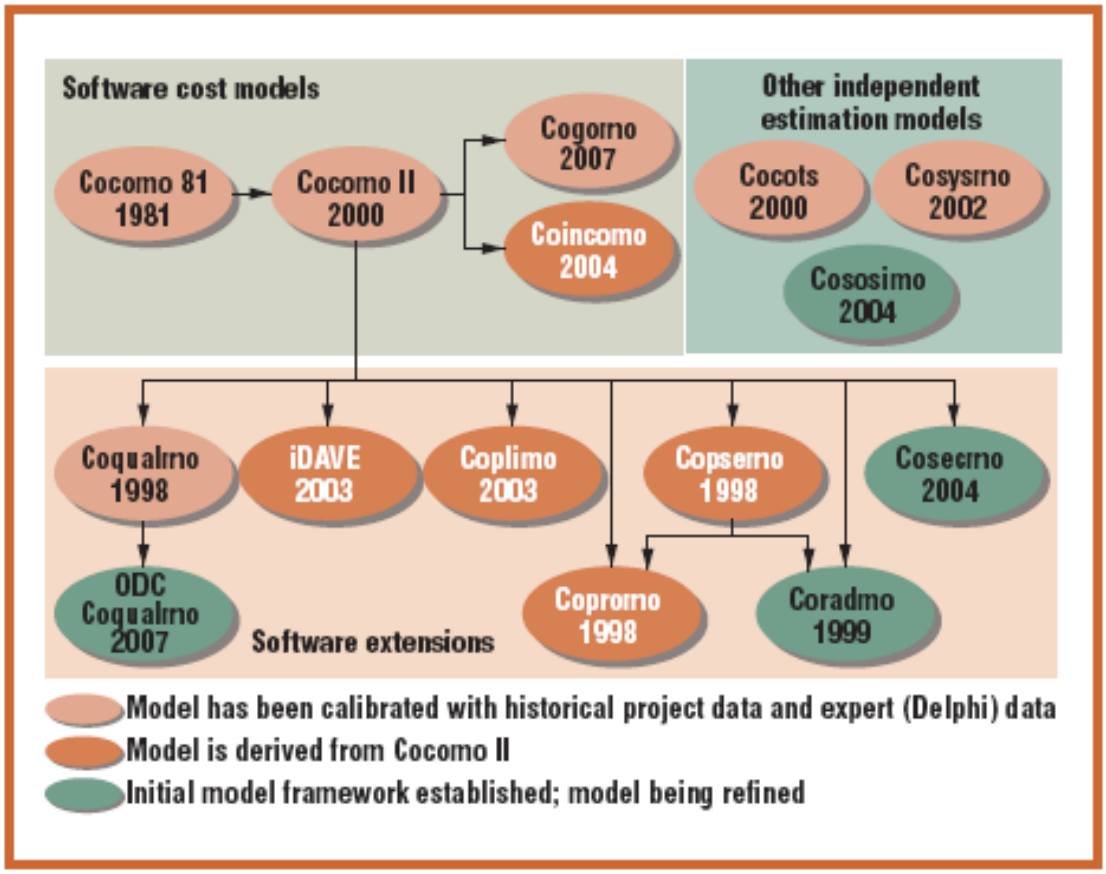
El modelo COCOMO 81 (1981) y su actualización COCOMO II (1997) han ido evolucionando, siendo la base de las herramientas de estimación existentes en la actualidad, razón por la que, a continuación se abordarán los fundamentos de los modelos COCOMO [10].

2.2 Metodología COCOMO

La metodología COCOMO (Modelo Constructivo de Costos ó Constructive Cost Model) fue desarrollada por Barry M. Boehm en 1981; el

modelo engloba un grupo de modelos algorítmicos que tratan de establecer una relación matemática para estimar el esfuerzo y tiempo requerido para desarrollar un producto software [11].

En la Figura 2.1 se muestra la evolución de COCOMO II respecto a las antiguas versiones [12].



The Cocomo suite of models. Dates indicate the time that the first paper was published for the model.

Figura 2.1 El modelo COCOMO y la evolución de COCOMO II
Fuente: [14]

2.2.1 La estimación en el modelo COCOMO II

La estimación cubre las etapas de Elaboración y Construcción propuestas por RUP/MBASE, tal como se muestra en la Figura 2.2 RUP/MBASE. Incluye todos los costos directos del proyecto, pero no los indirectos. Los datos empíricos que soportan el modelo se obtienen de una

muestra limitada de proyectos (83 proyectos en la versión inicial COCOMO II) [13,14].

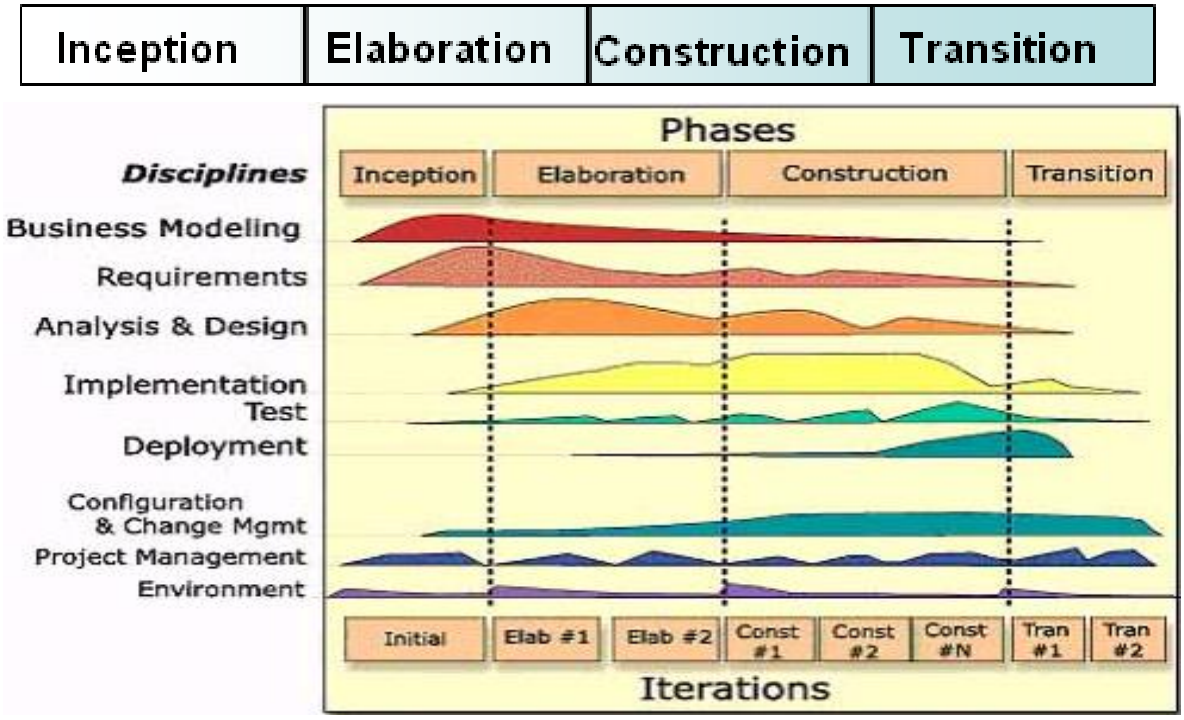


Figura 2.2 RUP/MBASE

Fuente: [69]

La familia de COCOMO II, a su vez tiene entre sus modelos, los siguientes [15]:

a) El Modelo de Composición de Aplicaciones: indicado para proyectos construidos con herramientas modernas de construcción de interfaces.

   Medida: puntos objeto

b) El Modelo de Diseño Anticipado: este modelo puede utilizarse para obtener estimaciones aproximadas del costo de un proyecto antes de que esté determinada su arquitectura. Utiliza un pequeño conjunto de drivers de costo.

   Medida: Puntos de función sin ajustar ó KSLOC (Miles de líneas de código fuente, depende de del lenguaje).

c) Modelo Post- Arquitectura: es el modelo COCOMO II más detallado; se utiliza una vez que se ha desarrollado por completo la arquitectura del proyecto. Modela en base a 17 drivers de costo.
Medida: Puntos de función sin ajustar ó KLSOC (Miles de líneas de código fuente).

En la figura 2.3 se puede apreciar el proceso de estimación utilizado en el modelo COCOMO II.

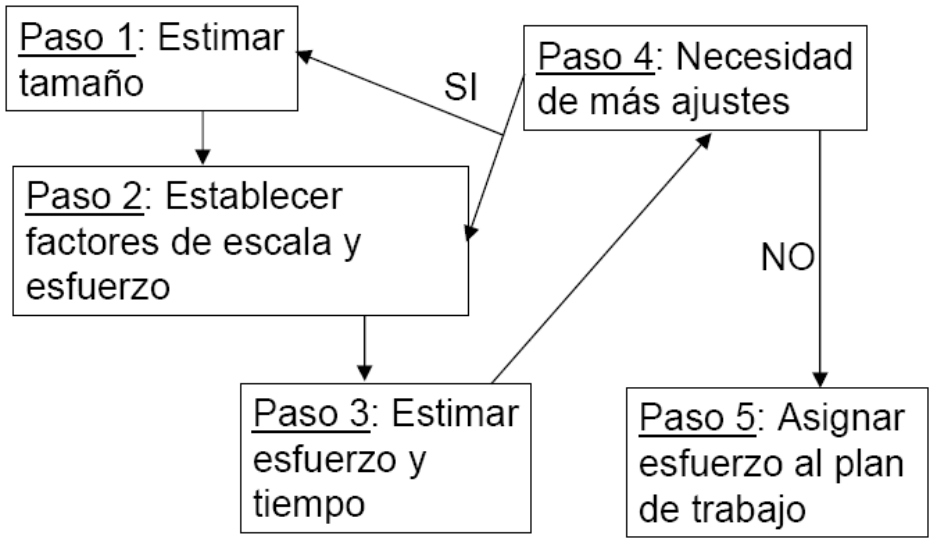


Figura 2.3 Proceso de estimación de COCOMO II

Fuente: [70]

2.2.2 Estructura de la metodología para COCOMO 81

La metodología COCOMO 81 se divide en tres modelos [16]:

- COCOMO básico: calcula el esfuerzo y el costo del desarrollo en función del tamaño del programa estimado en LOC (Líneas de código).
- COCOMO intermedio: calcula el esfuerzo del desarrollo en función del tamaño del programa y un conjunto de drivers de costo que incluyen la evaluación subjetiva del producto, del hardware, del personal y de los atributos del proyecto.

- COCOMO detallado: incorpora las características de la versión intermedia y lleva a cabo una evaluación del impacto de los drivers de costo en cada fase (análisis, desarrollo, etc.) del proceso.

Los modelos de COCOMO 81 están definidos para tres tipos de proyectos de software: [8]

- Orgánicos: para proyectos pequeños y sencillos; equipos pequeños con experiencia en el desarrollo de aplicaciones; requisitos poco rígidos.
- Semiacoplados: para proyectos de tamaño y complejidad intermedia; equipos con variados niveles de experiencia; requisitos poco o medio rígidos.
- Empotrados: para proyectos que deben ser desarrollados con un conjunto de requisitos hardware y software muy restringidos.

A continuación, en la Figura 2.4 se muestra una clasificación que permite aplicar estos modos y modelos.

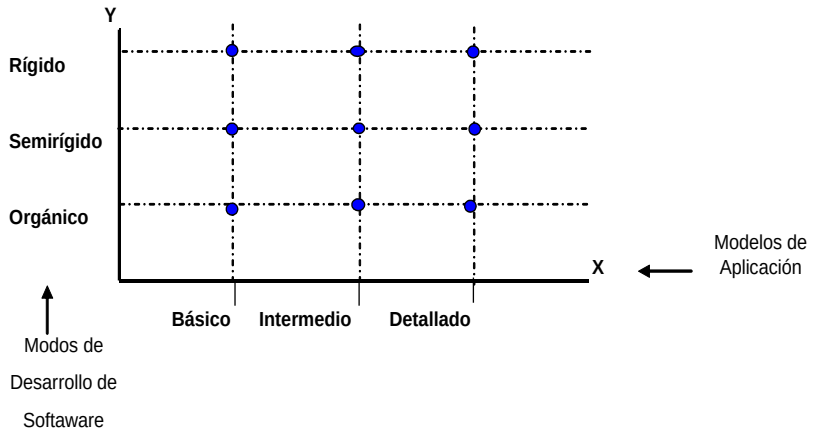


Figura 2.4 Cálculo de Modo y Modelo de COCOMO

Fuente: [71]

Por otro lado, los modos de COCOMO 81 son:

- COCOMO básico: el modelo básico se usa para obtener una aproximación rápida del esfuerzo. Usa las variables a, b, c y d, que varían en función de los modos. Conforme se aumenta la complejidad del modo, aumentan los valores de las variables (esfuerzo) [17].

Las fórmulas utilizadas en este modo son:

Esfuerzo

E = a * KLOC*b (personas/mes)

Tiempo de duración del desarrollo

T= c*Esfuerzo*d (d en meses)

Persona necesaria para un proyecto

P= E/T (p personas)

Los coeficientes a y c y los exponentes b y d se pueden obtener de la Tabla 2.5.

Tabla 2.5

Coeficientes de COCOMO Básico

Tipo de Proyecto	a	b	c	d
Orgánico	2.4	1.05	2.5	0.38
Semiacoplado	3.0	1.12	2.5	0.35
Empotrado	3.6	1.20	2.5	0.32

- COCOMO intermedio: añade al modelo básico 15 factores de ajuste o guías de costo (ver Tabla 2.6). Logramos mayor precisión en la estimación gracias a los nuevos factores. La fórmulas son similares a las del modelo básico pero con el añadido de los factores mencionados anteriormente [18].

Tabla 2.6

Factores de ajuste en COCOMO Intermedio

Manejadores de Costo	Very Low	Low	Nominal	High	Very High	Extra High
ACAP Analyst Capability	1.46	1.19	1.00	0.86	0.71	-
AEXP Applications Experience	1.29	1.13	1.00	0.91	0.82	-
CPLX Product Complexity	0.70	0.85	1.00	1.15	1.30	1.65
DATA Database Size	-	0.94	1.00	1.08	1.16	-
LEXP Language Experience	1.14	1.07	1.00	0.95	-	-
MODP Modern Programming Practices	1.24	1.10	1.00	0.91	0.82	-
PCAP Programmer Capability	1.42	1.17	1.00	0.86	0.70	-
RELY Required Software Reliability	0.75	0.88	1.00	1.15	1.40	-
SCED Required Development Schedule	1.23	1.08	1.00	1.04	1.10	-
STOR Main Storage Constraint	-	-	1.00	1.06	1.21	1.56
TIME Execution Time Constraint	-	-	1.00	1.11	1.30	1.66
TOOL Use of Software Tools	1.24	1.10	1.00	0.91	0.83	-
TURN Computer Turnaround Time	-	0.87	1.00	1.07	1.15	-
VEXP Virtual Machine Experience	1.21	1.10	1.00	0.90	-	-
VIRT Virtual Machine Volatility	-	0.87	1.00	1.15	1.30	-

Además, el valor de cada variable/atributo, dependiendo de su calificación, se muestra en la Tabla 2.7.

Las fórmulas utilizadas en este modo son:

Esfuerzo

E= a* KLDC*b *FAE (personas/mes)

Tiempo de duración del desarrollo

T= c *Esfuerzo* d (meses)

Personas necesarias para un proyecto

P= E/T (personas)

Tabla 2.7

Coeficientes de COCOMO Intermedio

Tipo de Proyecto	a	b	c	d
Orgánico	3.20	1.05	2.50	0.38
Semiacoplado	3.00	1.12	2.50	0.35
Empotrado	2.80	1.20	2.50	0.32

- **COCOMO detallado:** Los factores correspondientes a los atributos son sensibles a la fase sobre la que se realizan las estimaciones, puesto que aspectos tales como la experiencia en la aplicación, utilización de herramientas de software, etc., tienen mayor influencia en unas fases que en otras, y además porque van variando de una etapa a otra [19].

  Establece una jerarquía de tres niveles de productos, de forma que los aspectos que representan gran variación a bajo nivel, se consideran a nivel módulo; los que representan pocas variaciones, a nivel de subsistema; y los restantes son considerados a nivel sistema.

2.3 Métricas de puntos de función (PF) de Albretch

Los PF son una métrica aceptada como estándar en el mercado de software; se consideran como importantes las siguientes referencias de PF [20]:

- IFPUG (International Function Point Users Group).
- CPM 4.0 de 1994 (Counting Practice Manual)
- Inicialmente Albrecht en IBM (1979)

¿Cómo medir el Software? Es una de las tareas más complejas y difíciles pero se comprende mucho mejor con las siguientes definiciones

PF: Es una métrica que se puede aplicar en las primeras fases de desarrollo. Se basa en características fundamentalmente “Externas” de la aplicación a desarrollar. Los PF miden dos tipos de características:

- Los elementos de función (entradas, salidas, ficheros, etc.)
- Los factores de complejidad.

2.3.1 Elementos de PF

Son elementos fácilmente identificables en los diagramas de especificación del Sistema (Diagramas de Flujo de Datos, Entidad-Relación, Diccionario de Datos); los usuarios los entienden perfectamente.

El análisis de los Puntos de Función se desarrolla considerando cinco parámetros básicos:

1. Entrada (EI, ExternalInput).
2. Salida (EO, External Output).
3. Consultas (EQ, ExternalQuery).
4. Grupos de datos lógicos internos (ILF, InternalLogic File).
5. Grupos de datos lógicos externos (EIF, External Interface File).

Estos elementos enumerados anteriormente se pueden entender de mejor forma observando la Figura 2.8.

Figura 2.8 Elementos de los Puntos de Función

Fuente: [72]

Entradas: Un EI es un proceso elemental a través del cual se permite la entrada de datos al sistema. Estos datos provienen de una aplicación ajena al sistema, o del usuario, el cual los introduce a través de una pantalla de entrada de datos [8].

Ejemplos: Formularios, captura de imágenes, voz, controles o mensajes a través de los cuales un usuario final u otra aplicación agregada, borra o modifica datos del programa.

Salidas: Es un proceso cuyo propósito principal es presentar información al usuario mediante un proceso lógico diferente al de recuperar datos.

Ejemplos: Pantallas, reportes, figuras o mensajes que el programa genera para el uso de un usuario final u otra aplicación.

Consultas: Es un proceso elemental con componentes de entrada y salida que consiste en la selección y recuperación de datos de uno o más Ficheros Lógicos Internos o Ficheros Lógicos Externos, y su posterior devolución al usuario o aplicación solicitante.

Ejemplo: Consulta de un registro en una base de datos según un campo específico.

Archivos lógicos internos: Conjunto de datos relacionados que el usuario identifica, cuyo propósito principal es almacenar datos mantenidos a través de alguna transacción que se está considerando en el conteo [21].

Ejemplo: Archivos controlados por otro programa con los cuales una aplicación interactúa.

A continuación, en la Figura 2.9 se presenta la evolución de los métodos de estimación basados en puntos de función.

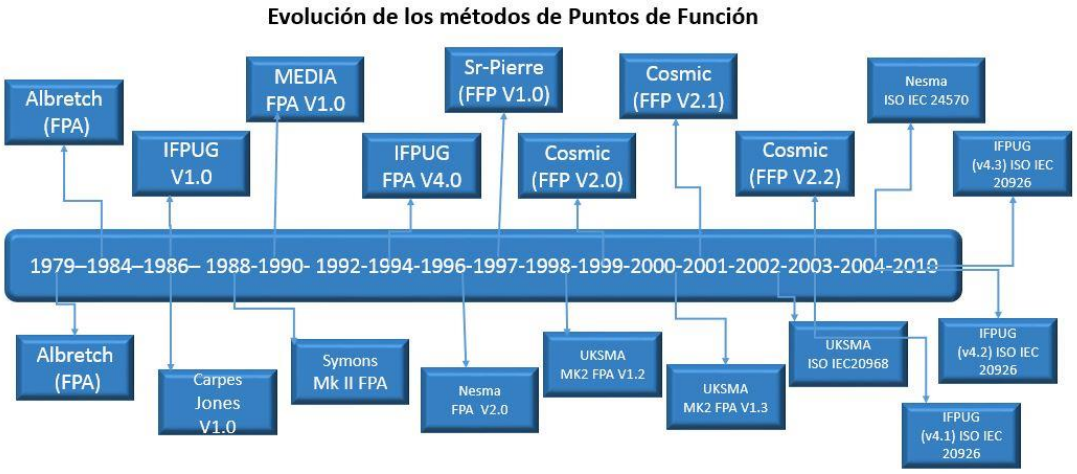


Figura 2.9 Evolución de los métodos de los Puntos de Función

Fuente: [25]

2.3.2 Factores de complejidad de los PF

Estos factores se muestran en la Tabla 2.10

Tabla 2.10

Factores de complejidad de PF

Fuente: [25]

CARACTERÍSTICA	DESCRIPCIÓN
Comunicación de datos	Cuántas facilidades de comunicación hay disponibles para ayudar en el intercambio de información con la aplicación o el sistema?
Procesamiento distribuido de datos	Cómo se manejan los datos y las funciones de procesamiento distribuido
Rendimiento	Existen requerimientos de velocidad o tiempo de respuesta?
Configuraciones fuertemente utilizadas	Cómo de intensivas se utilizan las plataformas hardware donde se ejecuta el sistema
Frecuencia de transacciones	Con qué frecuencia se ejecutan las transacciones? Diariamente, semanalmente,…
Entrada de datos on- line	Qué porcentaje de la información se ingresa on-line'
Eficiencia del usuario final	Aplicación diseñada para maximizar la eficiencia del usuario final
Actualizaciones Online	Cuántos Archivos Lógicos Internos se actualizan por una transacción on-line?
Procesamiento complejo	Hay procesamientos lógicos o matemáticos intensivos en la aplicación'
Reusabilidad	La aplicación se desarrolla para suplir una o muchas de las necesidades de los usuarios?
Facilidad de instalación	Qué tan difícil es la instalación y la conversión al nuevo sistema?
Facilidad de operación	Cómo de efectivos y/o automatizados deben ser los procedimientos de arranque, parada, backup y restore
Instalación en distintos lugares	La aplicación fue concebida para su instalación en múltiples sitios y organizaciones?
Facilidad de cambio	La aplicación fue concebida para facilitar los cambios sobre la misma?

2.4 Métodos de conteo de puntos de función.

- Método de Albretch.
- Método de Análisis de Puntos de Función MKII-FPA
- Método de Análisis de Puntos de Función FPA
- Método de Medición de Puntos de Función IFPUG-FPA
- Método COSMIC-FFP [22]

Las organizaciones que aportan a este proyecto son referenciadas a continuación:

Organizaciones de Apoyo Latinoamericano

- LASMA (Latin American Software Metrics Association A.C.): es un nuevo capítulo del IFPUG para la comunidad de habla hispana. El propósito es acercar a la gente de toda Latinoamérica interesada en compartir conocimientos acerca de la metodología de FunctionPoints, artículos, noticias, etc.

- BFPUG (Brazilian Function Point Users Group): es un grupo constituido con el objetivo de estimular y divulgar la utilización de métricas en el desenvolvimiento de sistemas, particularmente los análisis de punto de función. Cuenta con el apoyo de SUCESU-RJ (Sociedad de Usuarios de las Computadoras y Equipamientos de Telecomunicaciones de Río de Janeiro), además del apoyo del IFPUG en los Estados Unidos (International Function Point Users Group) y sus subgrupos en otros países.

Organizaciones de Apoyo Internacional

- UKSMA: Es la asociación de usuarios de métricas del software del Reino Unido. Su desarrollador fue Charles Symons en el Reino Unido, su objetivo fue mejorar el índice del tamaño del método de Albretch, y diseñó el método para ser compatible con las ideas del análisis estructurado [23].
- El NESMA: Fue fundado en 1989 como el NEFPUG (Grupo de Usuarios de Puntos de Función de los Países Bajos) siendo esta organización Holandesa, uno de los primeros grupos de usuarios de FP en el mundo.

- IFPUG (International FunctionPointsUserGroup): Es una organización que se preocupa de todos los aspectos de la medición de software: productividad, calidad, complejidad, e incluso implicancias psicológicas.
- COSMICON (Common Software Measurement International Consortium): Se creó para contar con métodos más precisos de estimación y medición de las características del software que sean igualmente fiables.
- SFPUG La Sociedad Española de Usuarios de Puntos de Función (Spanish Funtion Points Users Group) se constituyó en el año 2006, y está registrada en la Asociación del Ministerio del Interior de España como una comunidad científica sin ánimo de lucro. En el mismo año de su constitución fue reconocida por el International Function Points Users Group (IFPUG) como su capítulo oficial en España y por el Common Software Measurement International Consortium (COSMIC) como el grupo oficial de interés sobre COSMIC en España [24].
- ISO/IEC (Organización Internacional para la Estandarización/ Comisión Electrotécnica Internacional): La organización ISO/IEC es un organismo especializado en la elaboración, ejecución y control de todos los estándares orientados a la Medida del Tamaño Funcional, actualmente la 'ISO/IEC 14143-1:1998' InformationTechnology – Software Measurement – Functional Size Measurement. Es el estándar más implementado a nivel mundial, este define los conceptos de una métrica de tamaño, basada en la funcionalidad y las características que debe cumplir un método para poder ser considerado una medida del tamaño de la funcionalidad

2.5 Metodología de los PF

Es una métrica para establecer el tamaño y complejidad de los sistemas informáticos basada en la cantidad de funcionalidad requerida y entregada a los usuarios.

Los métodos de PF son ideas para la experimentación planificada o forma de comunicar los resultados experimentales y teóricos.

Puntos de Función

Los Puntos de Función son una técnica que permite medir la funcionalidad de una aplicación desde la perspectiva del usuario e independiente de todas las consideraciones de lenguaje [25].

Objetivos de los puntos de función

- Medir lo que el usuario pide y lo que el usuario recibe.
- Medir independientemente de la tecnología utilizada en la implantación del sistema.
- Proporcionar una métrica de tamaño que dé soporte al análisis de la calidad y la productividad.
- Proporcionar un medio para la estimación del software.
- Proporcionar un factor de normalización para la comparación de distintos softwares.

Además, el proceso de contabilizar los Puntos de Función debería ser:

- Simple para minimizar la carga de trabajo de los procesos de medida.
- Conciso en sus resultados.

El procedimiento para cálculo de los puntos de función se muestra en la Figura 2.11.

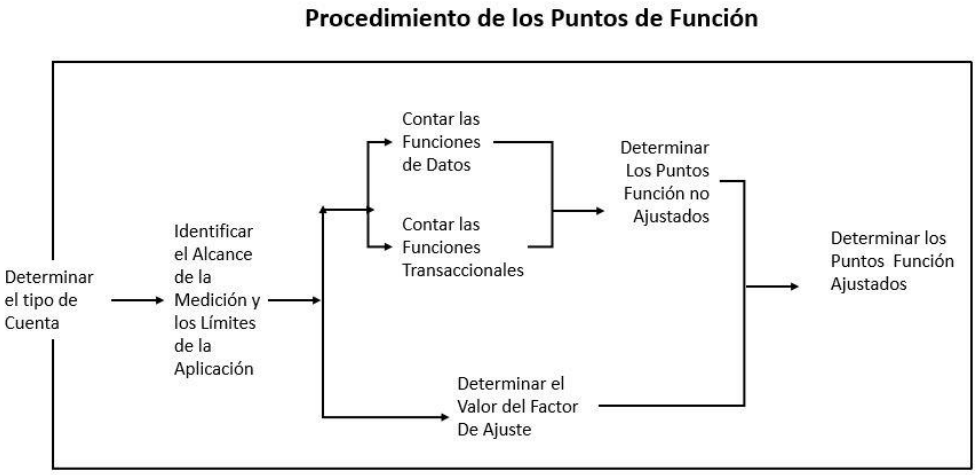


Figura 2.11 Procedimiento de los puntos de función

Fuente: [25]

2.5.1 Pasos para determinar el tipo de conteo

Paso 1. Identificar el tipo de conteo

Existen tres tipos de conteos:

- Conteo de los proyectos por primera vez: mide la funcionalidad inicial del proyecto proporcionada al usuario.
- Conteo por mantenimiento del proyecto: mide las modificaciones por cambios de una aplicación. El número de puntos de función debe actualizarse para que refleje los cambios en la funcionalidad de la aplicación.
- Conteo actualizado del proyecto: mide una aplicación ya contada. Este conteo proporciona el tamaño total de la aplicación proporcionada al usuario y cambia cada vez que se hacen modificaciones en la aplicación [26].

Paso 2. Identificar los alcances de la medición y las fronteras de la aplicación

El propósito de una medición consiste en dar una respuesta a un problema de negocio. La definición de la frontera en una aplicación no depende de consideraciones técnicas y/o de la implementación, esto se refiere más al límite existente entre el software a medirse y el usuario. En la Figura 2.12 se muestra un ejemplo de aplicación y su frontera:

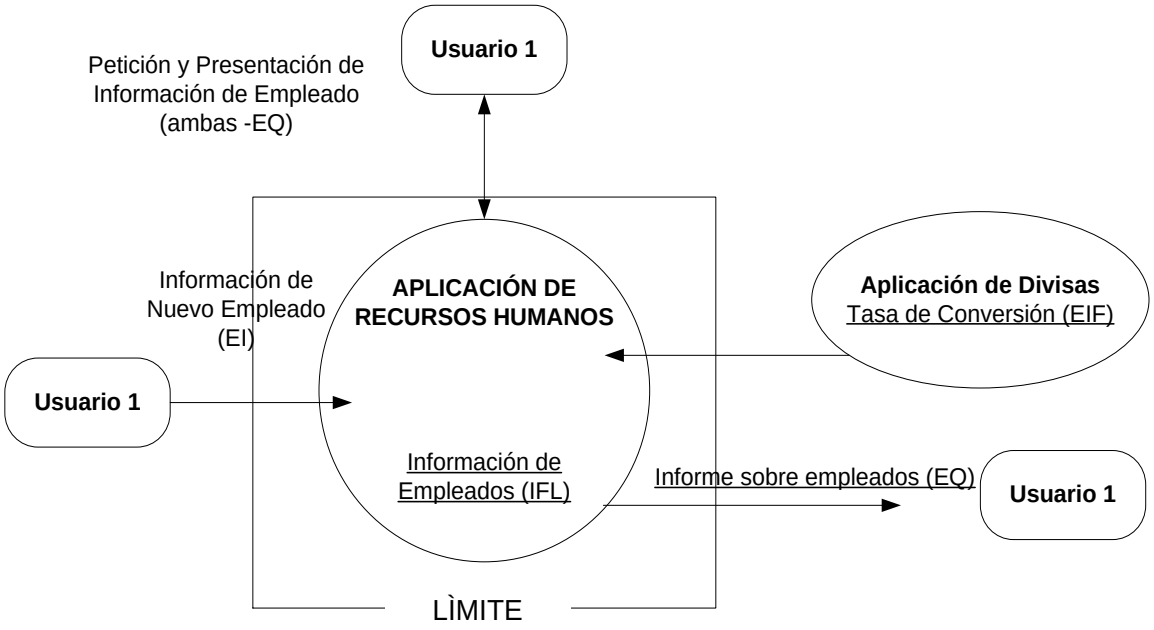


Figura 2.12 Frontera o limitación de la aplicación

Fuente: [73]

Paso 3. Contar las funciones de datos

Identificar y contar la capacidad de almacenamiento de los datos. Aquí se distinguen dos tipos de funciones de datos:

- Archivo Lógico Interno
- Archivo de Interfaz Externo

Paso 4. Contar las funciones transaccionales

Se distinguen tres tipos de funciones transaccionales [27]:

- Entrada Externa (EI).

- Salida Externa, (EQ) ó Consulta Externa (EQ).
- Archivos Lógicos Internos (IFL).

En la Figura 2.13 se muestran los tipos de funciones transaccionales.

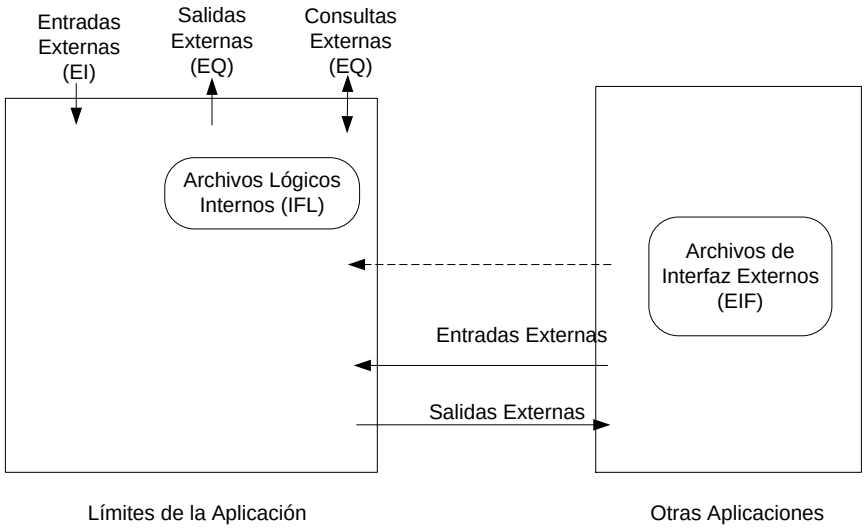


Figura 2.13 Funciones transaccionales

Fuente: [74]

Paso 5 Determinar los puntos de función no ajustados

Sumar el número de componentes de cada tipo, conforme a la complejidad asignada. Se recurre a los datos de la Figura 2.12 para obtener el total de las funciones transaccionales y de datos [28]:

Paso 6. Determinar el valor del factor de ajuste (FA)

El factor de ajuste por lo general se obtiene sumando el valor de 0.65 a la sumatoria de los grados de influencia correspondiente a las características generales del sistema, multiplicado por 0.01. Por lo tanto, el factor de ajuste

se aplica según el método (Albretch ó Mark II) como se muestra a continuación:

FA = (TDI x 0.01) + 0.65

Donde TDI es el grado total de influencia de las características (Fig. 2.12).

Paso 7. Determinar los puntos función ajustados (PF)

El valor del punto de función ajustado se obtiene del resultado de los PF no ajustados por el factor de ajuste, como se muestra en la siguiente relación [28]:

PF = UFP x FA

Donde UFP son los puntos de función sin ajuste.

CAPÍTULO III

REDES NEURONALES ARTIFICIALES

3.1 Introducción

Aunque todavía se ignora mucho sobre la forma en que el cerebro aprende a procesar la información, se han desarrollado modelos que tratan de mimetizar tales habilidades, denominados redes neuronales artificiales (RNA) o modelos de computación conexionista (otras denominaciones son computación neuronal y procesamiento distribuido paralelo o P.D.P.) [29]. A continuación se muestra un marco de referencia que permitirá conocer los principales modelos de RNA, sus mecanismos de funcionamiento, así como su utilidad práctica.

3.2 Síntesis histórica de las RNA

En 1943, Warren McCulloch y Walter Pitts propusieron el clásico modelo de neurona artificial en el que se basan las redes neuronales actuales. Seis años después, en 1949, en su libro TheOrganization of Behavior, Donald Hebb presentaba su conocida “regla de Hebb” **[75]**, que es el fundamento de todos los algoritmos de aprtendizaje para las redes neuronales artificiales.

En 1956, se organizó en Dartmouth la primera conferencia sobre Inteligencia Artificial (IA). Aquí, se discutió el uso potencial de las computadoras para simular “todos los aspectos del aprendizaje o cualquier otra característica de la inteligencia” y se presentó la primera simulación de una red neuronal, aunque todavía no se sabían interpretar los datos resultantes [30].

En 1957, Frank Rosenblatt presentó el perceptrón, una red neuronal con aprendizaje supervisado cuya regla de aprendizaje era una modificación de la propuesta por Hebb. El perceptrón trabaja con patrones de entrada binarios, y su funcionamiento, por tratarse de una red supervisada, se realiza en dos fases: una primera, de entrenamiento, en la que se presentan las

entradas y la salidas deseadas; en la fase siguiente, de operación, la red «es capaz» de responder adecuadamente cuando se le vuelven a presentar los patrones de entrada. En 1969 Minsky y Papert demostraron las grandes limitaciones de esta red.

En los años 60, se propusieron otros dos modelos, también supervisados, basados en el perceptrón de Rosenblatt denominados Adaline y Madaline. En éstos, el entrenamiento de la red se realiza teniendo en cuenta el error, calculado como la diferencia entre la salida deseada y la dada por la red, al igual que en el perceptrón. Sin embargo, la regla de aprendizaje empleada es distinta. Se define una función error para cada neurona que da cuenta del error cometido para cada valor posible de los pesos cuando se presenta una entrada a la neurona. Así, la regla de aprendizaje hace que la variación de los pesos se produzca en la dirección y sentido contrario del vector gradiente del error. A esta regla de aprendizaje, se la denomina Delta. **[76]**

En la década de los 80’s, Rumelhart, McClelland & Hinton crean el grupo PDP (Parallel Distributed Processing). Como resultado de los trabajos de este grupo salieron los manuales con más influencia desde la crítica de Minsky y Papert. Destaca el capítulo dedicado al algoritmo de retropropagación, que soluciona los problemas planteados por Minsky y Papert y extiende enormemente el campo de aplicación de los modelos de computación conexionistas.

3.3 Las redes neuronales biológicas

El cerebro es un procesador de información con unas características muy notables: es capaz de procesar a gran velocidad grandes cantidades de información procedentes de los sentidos, combinarla o compararla con la información almacenada y dar respuestas adecuadas incluso en situaciones nuevas. Logra discernir un susurro en una sala ruidosa, distinguir una cara en una calle mal iluminada o leer entre líneas en una declaración política;

pero lo más impresionante de todo es su capacidad de aprender a representar la información necesaria para desarrollar tales habilidades sin instrucciones explícitas para ello.

Para la ciencia sigue siendo un gran misterio la manera exacta de cómo el cerebro genera el pensamiento; no obstante se puede constatar que, severos golpes en la cabeza pueden producir disminuciones drásticas en las capacidades mentales (como p.e. pérdida de la memoria o inconciencia).

A grandes rasgos, recordemos que el cerebro humano se compone de decenas de billones de neuronas interconectadas entre sí, formando circuitos o redes que desarrollan funciones específicas. Una neurona típica (Figura 3.1) recoge señales procedentes de otras neuronas a través de una pléyade de delicadas estructuras llamadas dendritas. La neurona emite impulsos de actividad eléctrica a lo largo de una fibra larga y delgada denominada axón, que se escinde en millares de ramificaciones [31].

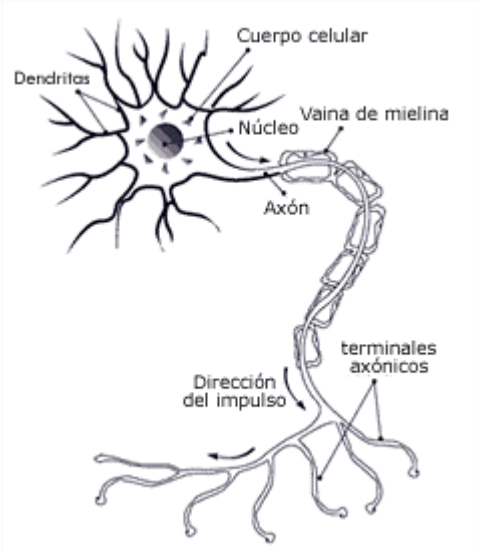


Figura. 3.1 Estructura básica de una neurona biológica

Fuente: [77]

Las extremidades de estas ramificaciones llegan hasta las dendritas de otras neuronas y establecen unas conexiones llamadas sinapsis, en las cuales se produce la transformación del impulso eléctrico en un mensaje

neuroquímico mediante la liberación de unas sustancias llamadas neuro-transmisores.**[78]**

El efecto de los neurotransmisores sobre la neurona receptora puede ser excitatorio o inhibitorio **[79],** y es variable (la intensidad del efecto depende de numerosos factores), de manera que podemos hablar de la fuerza o efectividad de una sinapsis. Las señales excitatorias e inhibitorias recibidas por una neurona se combinan, y en función de la estimulación total recibida, la neurona toma un cierto nivel de activación, que se traduce en la generación de breves impulsos nerviosos con una determinada frecuencia o tasa de disparo, y su propagación a lo largo del axón hacia las neuronas con las cuales se conecta.

3.4 Cerebros biológicos y computadoras digitales

Los cerebros y las computadoras realizan tareas totalmente distintas y sus características también lo son. Podría anticiparse que esta situación no se prolongará por mucho tiempo, ya que en tanto el cerebro humano evoluciona muy lentamente, la memoria de las computadoras aumenta con gran velocidad. De cualquier forma, la diferencia en capacidad de almacenaje es menor comparada con la diferencia en la velocidad de conmutación y paralelismo. El elevado paralelismo del procesamiento neuronal biológico contrasta con computadoras comunes que sólo cuentan con una o algunas CPU´s. En la Tabla 3.2 se muestra una burda comparación entre una computadora y un cerebro humano.

El cerebro es capaz de realizar una tarea compleja (por ejemplo reconocer un rostro) en menos de un segundo, tiempo apenas suficiente para completar unos cuantos cientos de ciclos. Una computadora, conectada en serie, requiere decenas de miles de millones de ciclos para realizar lo mismo, y no tan bien. Además, el cerebro es más tolerante a fallas que los computadores; un error de hardware que un solo bit pase por alto, podría provocar la ruina de todo un cómputo, en tanto que hay neuronas que

mueren al mismo tiempo sin ningún efecto adverso sobre el funcionamiento global del cerebro [32].

Tabla 3.2

Comparación básica entre una computadora y un cerebro humano

Fuente: [80]

Característica	Computadora	Cerebro humano
Unidades de cómputo	1 CPU, 10^5 compuertas	10^{11} neuronas
Unidades de almacenamiento	10^9 bits RAM, 1010 bits en disco	10^{11}neuronas, 10^{14}sinapsis
Ciclo de tiempo	10^{-8}seg	10^{-3}seg
Ancho de banda	10^9 bit/seg	10^{14} bit/seg
Actualización/seg de neuronas	10^5	10^{14}

3.5 Redes Neuronales Artificiales (RNA)

A pesar de las diferencias entre las computadoras y los cerebros humanos, éstos últimos han sido modelados a través de un conjunto de unidades básicas de procesamiento interconectadas, llamadas Neuronas Artificiales (NA). En la Figura 3.3 se muestra una representación neuronal artificial.

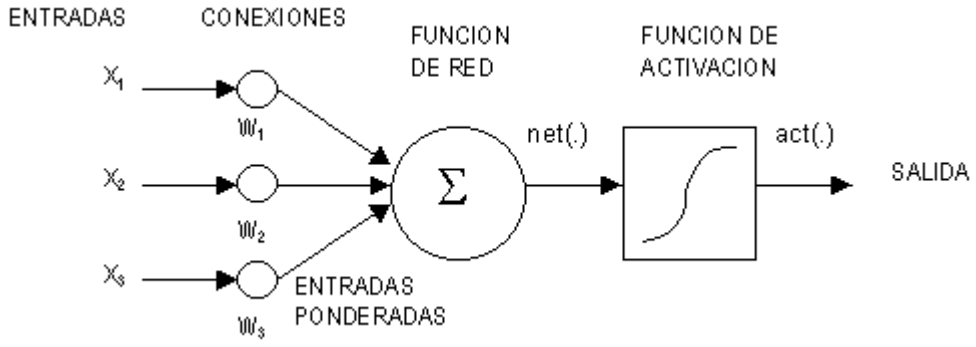


Figura 3.3 Representación de una neurona artificial

Fuente: [81]

A cada conexión se le asigna un peso numérico; los pesos constituyen el principal recurso de memoria a largo plazo en las redes neuronales, y el

aprendizaje se realiza con la actualización de tales pesos. Algunas de las unidades están conectadas al ambiente externo, y son designadas ya sea como unidades de entrada o unidades de salida. Los pesos serán modificados de manera tal que la conducta entrada/salida de la red esté más acorde con la del ambiente que produce las entradas.

Las unidades constan de un conjunto de conexiones de entrada provenientes de otras unidades, un conjunto de vínculos de salida que van hacia otras unidades, un nivel de activación del momento y recursos para calcular cuál será el nivel de activación del siguiente paso, con base en sus entradas y pesos respectivos [33].

3.6 Topologías de RNA

Para diseñar una red debemos establecer cómo estarán conectadas unas unidades con otras y determinar adecuadamente los pesos de las conexiones. Lo más usual es disponer las unidades en forma de capas, pudiéndose hablar de redes de una, de dos o de más de dos capas, las llamadas redes multicapa. Algunos autores cuentan sólo aquellas capas que poseen conexiones de entrada modificables; según este criterio la capa de entrada no contaría como tal.

Aunque inicialmente se desarrollaron redes de una sola capa, lo más usual es disponer tres o más capas: la primera capa actúa como buffer de entrada, almacenando la información bruta suministrada a la red o realizando un sencillo pre-proceso de la misma, la llamamos capa de entrada; otra capa actúa como interfaz o buffer de salida, almacenando la respuesta de la red para que pueda ser leída, la llamamos capa de salida; y las capas intermedias, principales encargadas de extraer, procesar y memorizar la información, las denominamos capas ocultas.

Además del número de capas de una red, en función de cómo se interconectan unas capas con otras, podemos hablar de redes recurrentes (feed-back) y redes no recurrentes o redes en cascada (feed-forward). En las redes en cascada la información fluye unidireccionalmente de una capa a otra (desde la capa de entrada a las capas ocultas y de éstas a la capa de salida), y además, no se admiten conexiones intracapa. En las redes recurrentes la información puede volver a lugares por los que ya había pasado, formando bucles, y se admiten las conexiones intracapa (laterales), incluso de una unidad consigo misma [34]. En el siguiente gráfico (Figura 3.4) se muestra una clasificación de las RN, tomando en consideración el tipo de conectividad entre sus elementos (neuronas individuales).

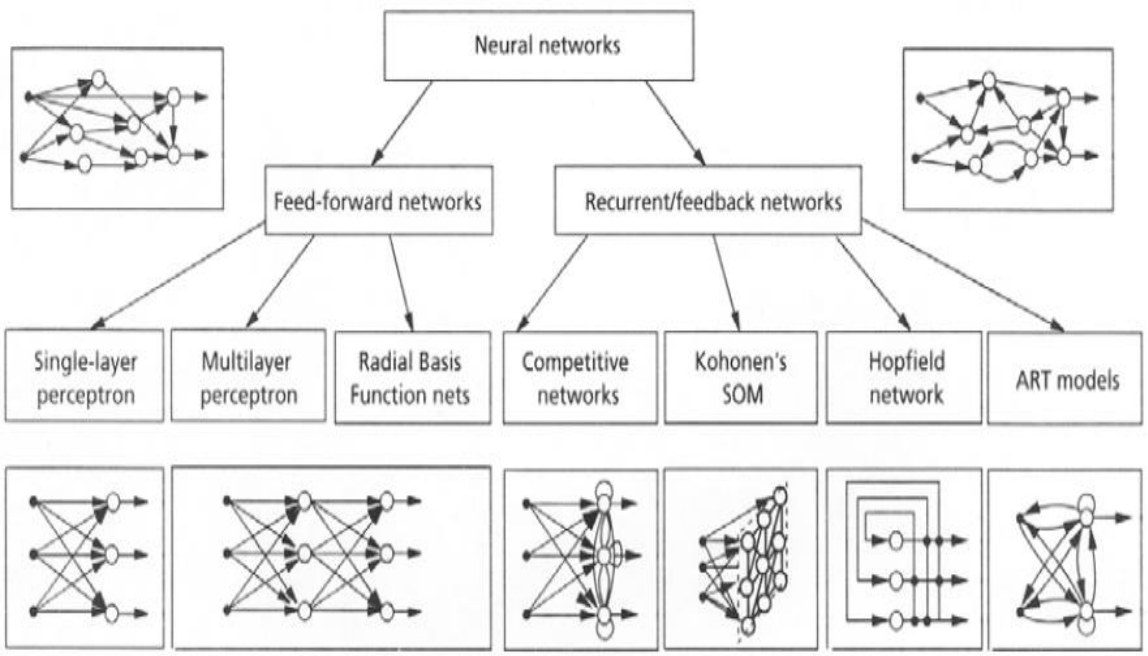


Figura 3.4 Categorías para RN

Fuente: [82]

3.7 Funciones en una RNA

En una RNA encontramos tres tipos de funciones que definen su comportamiento [35]:

- Función de Red: cuantifica la incidencia total en cualquier neurona, como combinación de todas las entradas (vector X) y los correspondientes pesos (vector W); generalmente se utiliza la función Lineal de Base:

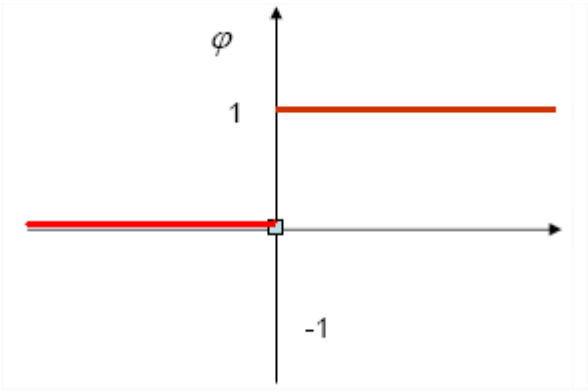
$$u_i(w,x) = \sum_{j=1}^{n} w_{ij}x_j$$

- Función de Activación: define el nivel de excitación de cada neurona, pudiendo ser de tipo lineal, umbral, y no lineal; una de las funciones más utilizada en las implementaciones neuronales es la Sigmoidea (no lineal en rango 0-1, con α>0), cuya fórmula es:

$$f(u_i) = \frac{1}{1+\exp(-\alpha u_i)}$$

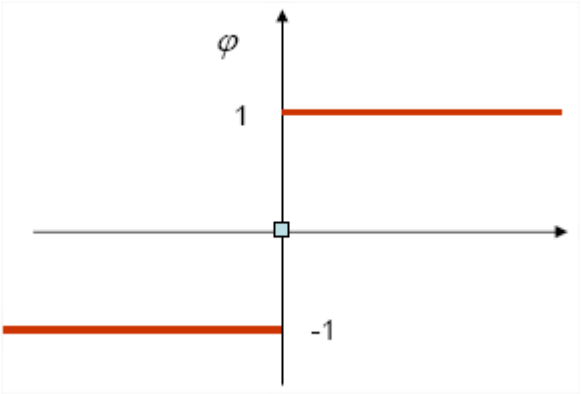
- Función de Salida: cuantifica el nivel respuesta de las neuronas; generalmente se suele utilizar la función Identidad, aplicada a la función de activación.

A continuación se muestra una lista de las funciones de activación (también llamadas *de transferencia*) más utilizadas en las RNA´s:



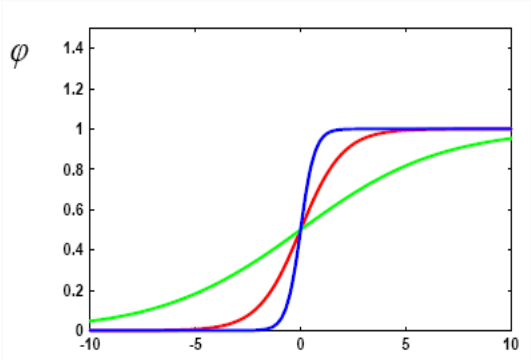
$$\varphi(x) = \begin{cases} 1 & si \quad x \geq 0 \\ 0 & si \quad x < 0 \end{cases}$$

a) Función Umbral o Threshold



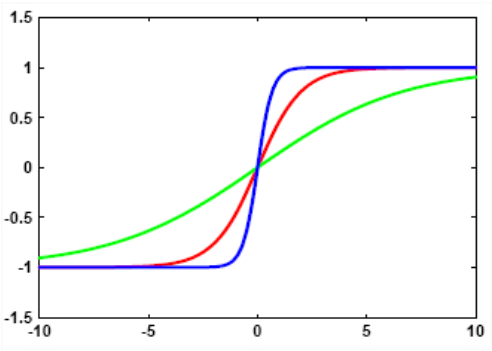
$$\varphi(x) = \begin{cases} -1 & si \quad x < 0 \\ 0 & si \quad x = 0 \\ 1 & si \quad x > 0 \end{cases}$$

b) Función Signo



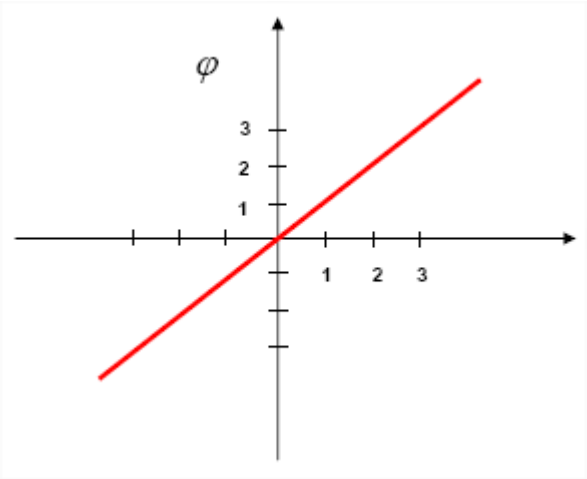
$$\varphi(x) = \frac{1}{1 + \exp(-\alpha x)} \qquad x \in R,\ \alpha > 0$$

c) *Función Logística*



$$\varphi(x) = \tanh(x/2) = \frac{1 - \exp(-\alpha x)}{1 + \exp(-\alpha x)} \qquad x \in R,\ \alpha > 0$$

d) Función Tangente hiperbólica



$$\varphi(x) = x \qquad x \in R$$

e) Función Lineal

Figura 3.5 Principales funciones de transferencia para RNA

Fuente: [83]

3.8 Perceptrones

Las redes feed-forward (hacia adelante) con niveles o capas fueron estudiadas por primera vez a finales de la década de los 50´s bajo el nombre de perceptrones. Si bien se sometieron a estudio redes de todos los tamaños y topologías, el único elemento de aprendizaje efectivo en esa época fueron las redes de un solo nivel. Actualmente el término perceptrón es sinónimo de una red de prealimentación de un sólo nivel [36].

El perceptrón simple fue el primer modelo de red neuronal artificial desarrollado en 1958 por Rosenblatt. Despertó un enorme interés en los años 60 debido a su capacidad para aprender a reconocer patrones sencillos. Está formado por varias neuronas para recibir las entradas a la red y una neurona de salida que es capaz de decidir cuándo una entrada a la red pertenece a una de las dos clases que es capaz de reconocer. A continuación se muestra una representación gráfica del perceptrón simple (Figura 3.6):

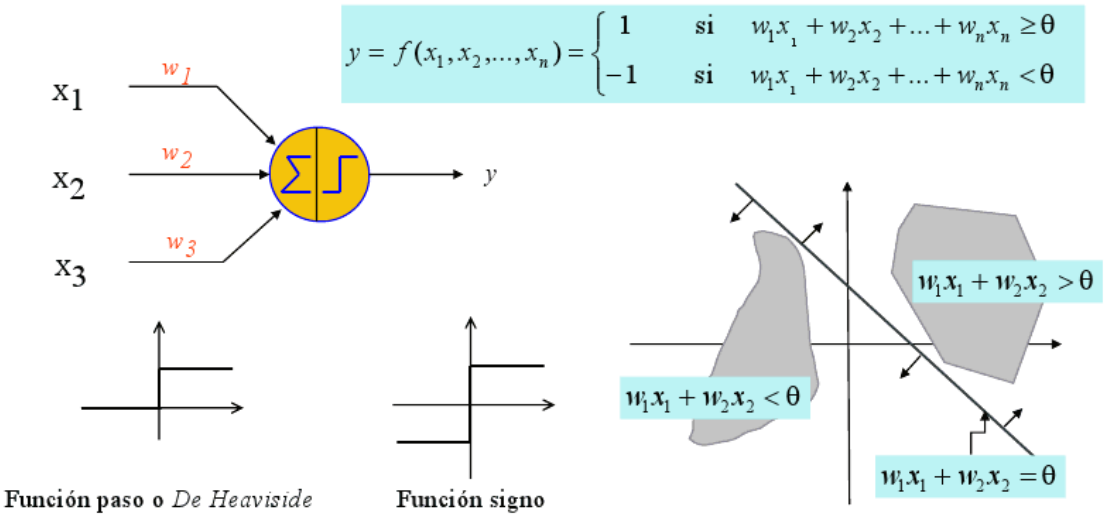


Figura 3.6 Esquema de un perceptrón típico, con sus posibles funciones de transferencia

Fuente: [84]

La neurona de salida del perceptrón realiza la suma ponderada de las entradas, resta el umbral y pasa el resultado a una función de transferencia de tipo escalón. La regla de decisión es: responder +1 si el patrón presentado pertenece a una clase A, o -1 si el patrón pertenece a una clase B.

Al constar sólo de una capa de entrada y otra de salida con una única neurona, tiene una capacidad de representación bastante limitada. Sólo es capaz de discriminar patrones muy sencillos y linealmente separables en el plano [37]. Es incapaz, por ejemplo, de representar la función OR-EXCLUSIVA (ver Figura 3.7a y 3.7b)

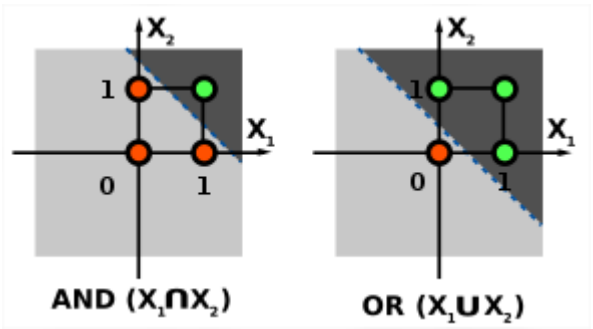


Figura 3.7 (a) Perceptrón que separa linealmente las respuestas de las compuertas lógicas AND y OR.

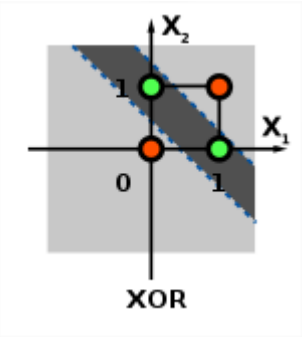


Figura 3.7 (b) Perceptrón que le es imposible separar linealmente las respuestas de la compuerta lógica XOR.

Fuente: [85]

La separabilidad lineal limita a las redes con sólo dos capas a la resolución de problemas en los cuáles el conjunto de puntos (valores de entrada) sean separables geométricamente. En el caso de dos entradas, la separación se lleva a cabo mediante una línea recta. Para tres entradas, la separación se realiza mediante un plano en el espacio tridimensional y así sucesivamente hasta el caso de N entradas, en el que el espacio N-dimensional es dividido en un hiperplano.

El hecho de que el perceptrón sólo es capaz de representar funciones linealmente separables se deduce de la expresión:

Salida = W*X

donde:

- W es el vector de los pesos de las conexiones con la neurona de salida.
- X es el vector de las entradas a la neurona de salida.

El perceptrón produce un 1 sólo si W*X > 0; o sea que, todo el espacio de entradas se divide en dos partes a lo largo de un límite definido por W*X = 0, es decir, un plano en el espacio de entrada con coeficientes definidos por los pesos.

Regla de aprendizaje del Perceptrón

El algoritmo de aprendizaje del perceptrón es de tipo supervisado, lo que requiere que sus resultados sean evaluados y se realicen las oportunas modificaciones de los pesos si fuera necesario.

Para entender el mecanismo de aprendizaje del perceptrón nos basaremos en la función lógica OR. Lo que se pretende, al modificar los pesos, es encontrar una recta que divida el plano en dos espacios de las dos clases de valores de entrada: las que producen un 1 a la salida y las entradas que producen un 0 a la salida. Concretamente deberá separar las entradas 01, 10, 11 de la entrada 00. (Función OR: produce 0 cuando las dos entradas son 0. En cualquier otro caso produce 1)

La ecuación de salida (y) vendrá dada por:

$y = f(w_1x_1 + w_2x_2 + \ldots + w_nx_n)$

Donde cada w_i representa el peso (valor) de la conexión, x_i es cada valor de entrada y f es la función de transferencia o de salida. En el caso de la función de salida, si la suma anterior es mayor que 0, la salida será 1 y en caso contrario, -1 (función escalón).

Se supone también que se añade una neurona con una entrada fija a 1 y peso w_0 igual al opuesto del valor del umbral y que deberá ser ajustado durante la etapa de aprendizaje.

A continuación se expone el algoritmo y posteriormente un ejemplo que aclara el funcionamiento.

Algoritmo

1. Inicialización de los pesos y del umbral : inicialmente se asigna valores aleatorios a cada uno de los pesos w_1, w_2,…w_n, y w_0 = umbral.
2. Presentación de un nuevo par (Entrada, Salida esperada):
3. Cálculo de la salida actual y(t)

$$y(t) = f\ [\Sigma(wi(t)\ xi(t)) - umbral]$$

Donde wi(t) denota el peso i-ésimo, xi(t) representa la entrada i-ésima, t es la iteración actual, Neti = Σ(wi(t) xi(t)).

4. Adaptación de los pesos

$$wi(t+1) = wi(t) + ß\ [d(t) - y(t)]\ xi(t)$$

Donde d(t) representa la salida deseada y ß es un factor de ganancia (velocidad de aprendizaje), en el rango 0.0 a 1.0. En la función OR ß=1.

5. Volver al paso 2

Ejemplo

a) Sean inicialmente los valores aleatorios :

$$w_0 = 1.5 \quad w_1 = 0.5 \quad w_2 = 1.5$$

b) Se van tomando uno a uno los cuatro patrones de entrada

b.1) Patrón 00

Entradas : $x_o = 1$; $x_1 = 0$; $x_2 = 0$

Pesos : $w_o(t) = 1.5$; $w_1(t) = 0.5$; $w_2(t) = 1.5$

Net_i : 1 (1.5)+ 0 (0.5) + 0 (1.5) = 1.5

Salida que produce f : 1, puesto que $Net_i > 0$

Salida deseada : 0, puesto que OR(00) = 0

Pesos modificados : $w_o(t+1) = 0.5$; $w_1(t+1) = 0.5$; $w_2(t+1) = 1.5$

b.2) Patrón 01

Entradas : $x_o = 1$; $x_1 = 0$; $x_2 = 1$

Pesos : $w_o(t) = 0.5$; $w_1(t) = 0.5$; $w_2(t) = 1.5$

Net_i : 1 (0.5)+ 0 (0.5) + 1 (1.5) = 2

Salida que produce f : 1, puesto que $Net_i > 0$

Salida deseada : 1, puesto que OR(01) = 1

Error (deseada - obtenida) = 0

Pesos no modificados : $w_i(t+1) = w_i(t)$

b.3) Patrones 10 y 11 : la salida obtenida es igual que la deseada por lo que no varían los pesos.

c) Se toman de nuevo los cuatro patrones de entrada

c.1) Patrón 00

Entradas : $x_o = 1$; $x_1 = 0$; $x_2 = 0$

Pesos : $w_o(t) = 0.5$; $w_1(t) = 0.5$; $w_2(t) = 1.5$

Net_i : 1 (0.5)+ 0 (0.5) + 0 (1.5) = 0.5

Salida que produce f : 1, puesto que Neti> 0

Salida deseada : 0, puesto que OR(00) = 0

Pesos modificados : $w_o(t+1) = -0.5$; $w_1(t+1) = 0.5$; $w_2(t+1) = 1.5$

c.2) Patrón 01

Entradas : $x_o = 1$; $x_1 = 0$; $x_2 = 1$

Pesos : $w_o(t) = -0.5$; $w_1(t) = 0.5$; $w_2(t) = 1.5$

Net_i : 1 (-0.5)+ 0 (0.5) + 1 (1.5) = 2

Salida que produce f : 1, puesto que $Net_i > 0$

Salida deseada : 1, puesto que OR(01) = 1

Error (deseada - obtenida) = 0

Pesos no modificados : $w_i(t+1) = w_i(t)$

c.3) Patrones 10 y 11 : la salida obtenida es igual que la deseada por lo que no varían los pesos.

d) Se toman de nuevo los cuatro patrones de entrada

d.1) Patrón 00

Entradas : $x_o = 1$; $x_1 = 0$; $x_2 = 0$

Pesos : $w_o(t) = -0.5$; $w_1(t) = 0.5$; $w_2(t) = 1.5$

Net_i : 1 (-0.5)+ 0 (0.5) + 0 (1.5) = -0.5

Salida que produce f : 0, puesto que $Net_i < 0$

Salida deseada : 0, puesto que OR(00) = 01

Error (deseada - obtenida) = 0

Pesos no modificados : $w_i(t+1) = w_i(t)$

d.2) Patrones 01, 10 y 11 : la salida obtenida es igual que la deseada por lo que no varían los pesos.

Con estos nuevos pesos, al calcular la salida que se obtiene para cualquiera de los cuatro patrones de entrada ya no se comete ningún error, por lo que la etapa de aprendizaje concluye.

3.9 Red en Backpropagation

3.9.1 Estructura

En una red en Backpropagation existe una capa de entrada con n neuronas y una capa de salida con m neuronas y al menos una capa oculta de neuronas internas. Cada neurona de una capa (excepto las de entrada) recibe entradas de todas las neuronas de la capa anterior y envía su salida a todas las neuronas de la capa posterior (excepto las de salida). No hay conexiones hacia atrás ni laterales entre neuronas de la misma capa [38]. En la Figura 3.8 se muestra una red en Backpropagation.

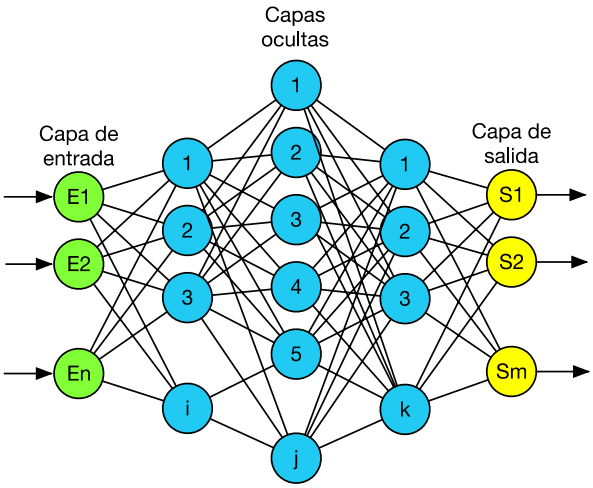


Figura 3.8 Red en Backpropagation de cinco capas
Fuente: [86]

El algoritmo de aprendizaje (Backpropagation o propagación del error hacia atrás) es el que ha dado dicha denominación a estas redes; la regla de aprendizaje se puede aplicar en modelos de redes con más de dos capas de neuronas. Una característica importante de este algoritmo es la representación interna del conocimiento que es capaz de organizar en la capa intermedia de las células para conseguir cualquier correspondencia entre la entrada y la salida de la red [39].

La importancia de esta red consiste en su capacidad de auto-adaptar los pesos de las neuronas de las capas intermedias para aprender la relación que existe entre un conjunto de patrones dados como ejemplo y sus salidas correspondientes, para poder aplicar esa misma relación, después del entrenamiento, a nuevos vectores de entrada con ruido o incompletas, dando una salida activa si la nueva entrada es parecida a las presentadas durante el aprendizaje. Esta característica, exigida en los sistemas de aprendizaje, es la capacidad de generalización, entendida como la facilidad de dar salidas satisfactorias a entradas que el sistema no ha visto nunca en su fase de entrenamiento.

3.9.2 Funcionamiento

El funcionamiento de la red consiste en el aprendizaje de un conjunto predefinido de pares de entradas-salidas dados como ejemplo, empleando un ciclo propagación-adaptación de dos fases [40]:

1. Se aplica un patrón de entrada como estímulo para la primera capa de las neuronas de la red, se va “propagando” a través de todas las capas superiores hasta generar una salida, se compara el resultado obtenido en las neuronas de salida con la salida que se desea obtener y se calcula un valor del error para cada neurona de salida.
2. Estos errores se transmiten “hacia atrás”, partiendo de la capa de salida, hacia todas las neuronas de la capa intermedia que contribuyan directamente a la salida, recibiendo el porcentaje de error aproximado a la participación de la neurona intermedia en la salida original. Este proceso se repite, capa por capa, hasta que todas las neuronas de la red hayan recibido un error que describa su aportación relativa al error total. Basándose en el valor del error recibido, se reajustan los pesos de conexión de cada neurona, de manera que en la siguiente vez que se presente el mismo patrón, la salida esté más cercana a la deseada (disminuya el error).

A diferencia de la regla delta en el caso del Perceptrón, esta técnica requiere el uso de neuronas cuya función de activación sea continua y por tanto diferenciable. Generalmente la función será de tipo sigmoidal (ver Figura 3.9):

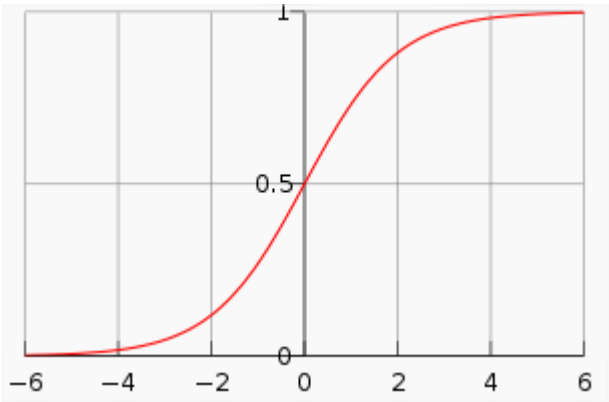


Figura 3.9 Función sigmoidal [87] $f(x) = 1/(1+e^{-x})$

3.9.3 Algoritmo de entrenamiento

Paso 1
Inicializar los pesos con valores aleatorios pequeños
Paso 2
Presentar el patrón de entrada, Xp : x_{p1}, ..., x_{pn}, y especificar la salida deseada : d_1,..., d_m.
Paso 3
Calcular la salida actual de la red: y_1, ..., y_m
Para ello :

- Se calculan las entradas netas para las neuronas ocultas procedente de las neuronas de entrada
- Se calculan las salidas de las neuronas ocultas
- Se realizan los mismos cálculos para obtener las salidas de las neuronas de salida

Paso 4
Calcular los términos de error para todas las neuronas
Paso 5
Actualización de los pesos

Paso 6
Repetición del proceso hasta que el término de error resulte aceptablemente pequeño para cada uno de los patrones aprendidos.

3.9.4 Aplicaciones de las redes en backpropagation

Este tipo de redes se están aplicando a distintas clases de problemas [41]. Esta versatilidad se debe a la naturaleza general de su proceso de aprendizaje ya que solamente se necesitan dos ecuaciones para propagar las señales de error hacia atrás. La utilización de una u otra ecuación sólo depende de si la unidad de proceso es o no de salida. Algunos de los campos de aplicación más representativos son:

- Codificación de información: la idea consiste en que la información de entrada se recupere en la salida a través de un código interno.
- Traducción de texto en lenguaje hablado
- Reconocimiento de lenguaje hablado
- Reconocimiento óptico de caracteres (OCR)
- Aplicaciones en cardiología

  - Clasificación de señales electrocardiográficas (ECG)
  - Detección de taquicardias ventriculares y supraventriculares
  - Detección de complejos QRS anómalos
  - Reconocimiento de formas anormales en señales ECG
  - Emulación hardware de una red neuronal para el procesado de ECG
  - Cancelación de ruido en señales ECG

- Comprensión/descomprensión de datos

3.4 Modelos de redes recurrentes

Se caracterizan porque pueden tener ciclos o bucles en las conexiones (conexiones recurrentes). En la figura 3.10 se muestra un ejemplo de RNA recurrente.

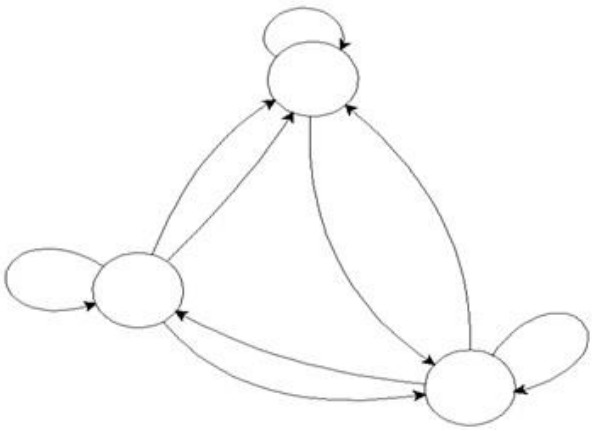


Figura 3.10 Conexiones recurrentes: una neurona con ella misma o entre neuronas

Fuente: [88]

Al permitir conexiones recurrentes aumenta el número de pesos o de parámetros ajustables de la red; aumenta la capacidad de representación. Por otro lado, se complica el aprendizaje, pues, las activaciones no dependen sólo de las activaciones de la capa anterior sino también de la activación de cualquier otra neurona conectada a ella, e incluso de su propia activación [42].

En estas redes es necesario incluir la variable tiempo para la determinación del estado de activación ai:

$$a_i(t+1) = f_i\Big(\sum_j w_{ji} a_j(t)\Big)$$

Donde fi representa la función de activación utilizada, wji denotan los pesos que inciden en la neurona j-ésima.

La variable tiempo hace que las redes tengan un comportamiento dinámico o temporal. Existen dos formas de entender el modo de actuación y aprendizaje:

- Evolución de las activaciones de la red hasta alcanzar un punto estable.
- Evolución de las activaciones de la red en modo continuo.

Red de Hopfield

Es capaz de recuperar patrones almacenados a partir de información incompleta e incluso a partir de patrones con ruido (en la Figura 3.11 se puede observar una red de Hopfield con cuatro neuronas). Entre las características más importantes de estas redes tenemos:

- Actúan como una “memoria asociativa” procesando patrones estáticos (sin variable tiempo).
- Cada neurona está conectada con todas las demás

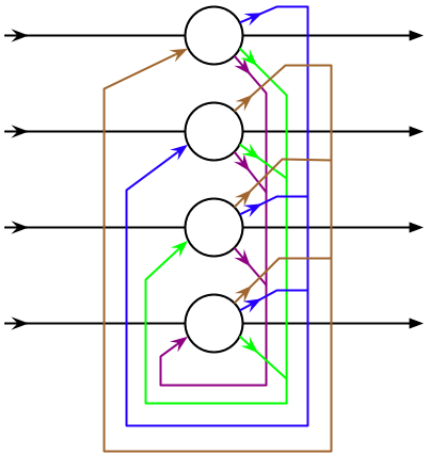


Figura 3.11 Red de Hopfield con cuatro neuronas

Fuente: [89]

Otras características de la red de Hopfield son [43]:

- Matriz de pesos $W=(w_{ij})$, orden n x n.
- wij; peso de la conexión de neurona i a neurona j
- Matriz simétrica: $w_{ij}=w_{ji}$
- Los elementos de la diagonal son nulos (no existen conexiones reflexivas)
- Las neuronas poseen dos estados -1 y 1
- Estado de la neurona i en k+1:

$$s_i(k+1) = \begin{cases} 1 & \text{si} \quad \sum_{j=1}^{N} w_{ij} s_j(k) > \theta_i \\ s_i(k) & \text{si} \quad \sum_{j=1}^{N} w_{ij} s_j(k) = \theta_i \\ -1 & \text{si} \quad \sum_{j=1}^{N} w_{ij} s_j(k) < \theta_i \end{cases}$$

Donde $s_i(k)$ es el estado de la neurona i en el instante anterior k, y θ_i es un umbral fijo aplicado a la neurona i.

La red de Hopfield tiene dos fases de operación:

- Fase de almacenamiento: se determinan los valores que tendrán los pesos para almacenar un conjunto de patrones.
- Fase de recuperación: mecanismo para recuperar la información almacenada a partir de información incompleta.

Una mejora de la red de Hopfield lo constituyen las llamadas máquinas de Bolztmann, mismas que implementan un mecanismo funcionamiento de tipo probabilista, basado en la técnica de *enfriamiento simulado* [44].

Máquinas de Bolztmann

La máquina de Boltzmann es útil para el reconocimiento de patrones intentando recuperar información no disponible de un estado (es decir completando las partes que no conocemos). Estas redes de consisten en neuronas conectadas entre sí que pueden estar conectadas bidireccionalmente y que tienen salidas binarias. Las neuronas se distinguen en dos grupos: las visibles y las no visibles. Las primeras constituyen la interfaz de la red y las segundas son sólo para un mejor desempeño de la red.

Hay dos arquitecturas principales para las máquinas de Boltzmann: Completación de Boltzmann y la red de Boltzmann de entrada-salida [45].

La diferencia está en la capa visible, pues en las de completación sólo hay un tipo y están conectadas entre ellas de forma bidireccional (todas las neuronas con todas, inclusive las no visibles); en la red de entrada-salida las visibles se dividen en las neuronas de entrada y en las de salida, siendo las de entrada únicamente conectadas unidireccionalmente con la capa no visible y las neuronas de salida. Las de salida se conectan con todas las que no son de entrada de forma bidireccional.

La característica principal de las redes de Boltzmann es que la función de salida es estocástica con probabilidad:

$$P_k = \frac{1}{1+\exp(-\frac{net_k}{T})}$$

Donde net_k es la diferencia de energía en el sistema cuando $x_k=0$ y $x_k=1$(donde x_k es la salida de la k-ésima unidad) y está dada por la siguiente expresión:

$$net_k = \sum_{\substack{j=1 \\ J\neq k}}^{n} w_{kj} x_j$$

El parámetro T representa la temperatura del sistema. Para simular el proceso del annealing usamos el siguiente algoritmo [46]:
Sea x' el vector de entrada con componentes desconocidos.

1. Asignar los valores conocidos del vector de entrada x' a las neuronas visibles.
2. Imputar todas los valores desconocidos y de las neuronas no visibles con valores aleatorios en el conjunto {0,1}.
3. Seleccionar una unidad x_k aleatoriamente y calcular su valor de entrada a la red (net_k)
4. Sin importar el valor actual de la unidad seleccionada, asignar el valor $x_k=1$ con probabilidad P_k (definida anteriormente). Esta elección estocástica se puede implementar con una comparación entre P_k y un valor z seleccionado al azar de la distribución uniforme. Con z entre 0 y 1 y menor o igual que P_k.

5. Repetir los pasos 3 y 4 hasta que todas la unidades tengan una probabilidad de ser seleccionadas para una actualización. Este número de actualización de unidades se llama ciclo de procesamiento. El realizar un ciclo completo no garantiza que todas la unidades hayan sido actualizadas.
6. Repetir el paso 5 hasta llegar al *equilibrio térmico*. **[90]**
7. Bajar la temperatura T y repetir pasos 3 a 7.

La convergencia al estado de mínima energía se asegura por el teorema de German y German que asegura que si las temperaturas del k-ésimo paso del algoritmo está acotada inferiormente por $\frac{T_0}{\log(1+k)}$ donde T_0 es una constante suficientemente grande se alcanzará un estado de energía con diferencia mínima (ε) al estado de mínima energía [47].

El algoritmo se detiene cuando T se ha reducido hasta un valor pequeño. Cuando esto sucede la red se ha estabilizado y el resultado final será las salidas de las neuronas visibles. Esperamos que el resultado final sea un vector x que tenga todos sus componentes conocidos.

CAPÍTULO IV

DISEÑO DE MONEPS

4.1 Introducción

"Un modelo de Calidad puede considerarse como el conjunto de factores de calidad y de relaciones entre ellos que brindan una base para la elicitación de requisitos y para su evaluación a nivel de los componentes de software". **[91]**

Los modelos de calidad se estructuran, generalmente, como una jerarquía (ya sea un árbol, ya sea un grafo dirigido) donde factores de calidad más genéricos, como eficiencia o usabilidad, se descomponen en otros más particulares, como tiempo de respuesta o facilidad de aprendizaje, probablemente en diversos niveles de descomposición [48].

A continuación se presentan los tipos de modelos de calidad, ISO 25000, 9126, considerados como estándares y que servirán de marco de referencia para MONEPS.

4.2 Tipos de modelos de calidad

En los modelos de calidad fijos existe un catálogo de factores de calidad de partida que se usa como base para la evaluación de la calidad. Este enfoque supone que el modelo de calidad contiene todos los factores de calidad posibles, y que se usará un subconjunto de dichos factores para cada proyecto concreto [49].

En la siguiente Figura 4.1 se muestra los modelos de calidad y su división en Fijos, Mixtos y a Medida; en éstos el orden no es un tema importante sino más bien de contexto general (Luna Tellez Linda,2012).

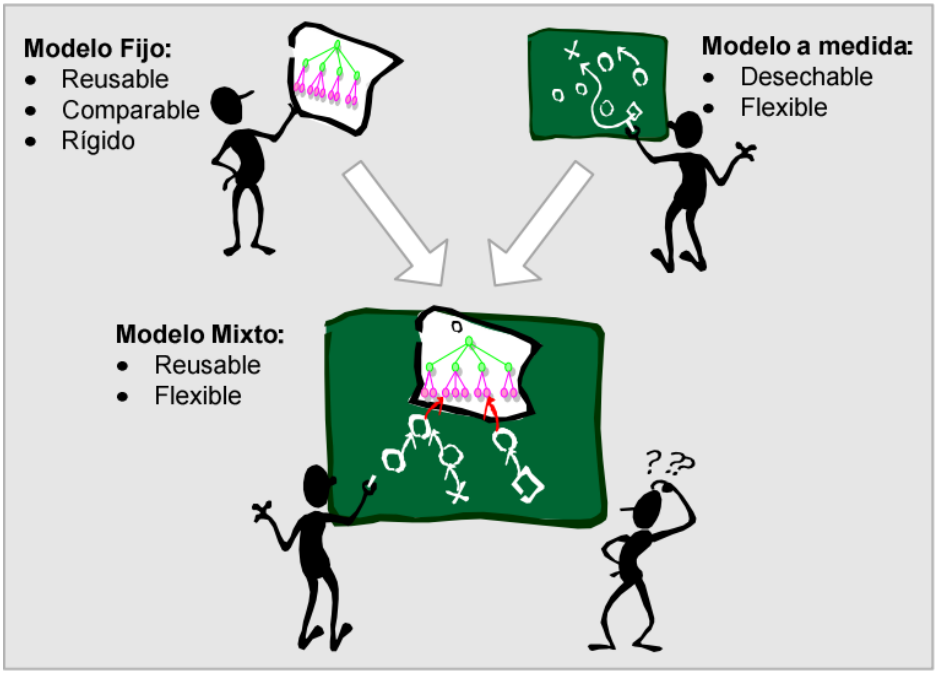


Figura 4.1 Modelos de calidad

Fuente: [92]

Para los modelos de calidad fijos (p.e ver la Figura 4.2) existe un catálogo de factores de calidad de partida que se usa como base para la evaluación de la calidad; en este enfoque se sobre entiende que el modelo de calidad contiene todos los factores de calidad posibles, además se utilizará un subconjunto de dichos factores para cada proyecto en concreto. La propuesta típica consiste en una estructuración de los factores en una jerarquía multinivel con un conjunto de factores de más alto nivel unos criterios que descomponen dichos factores y, eventualmente, métricas para la medida de cada criterio [48].

En los modelos de calidad a medida (p.e ver Figuras 4.3a y 4.3b) no existe ningún catálogo de factores de partida, y dichos factores deben ser identificados para cada proyecto. La idea que guía la construcción de estos modelos debe partir de la identificación de los objetivos a alcanzar. Estos objetivos serían los factores más abstractos que deben descomponerse en factores más concretos hasta llegar a hacer operativos los objetivos, de forma que pueda ser medida su consecución. La ventaja de estos modelos

es su total adaptabilidad. Por otro lado, tienen como inconveniente que el costo de su construcción es muy alto comparado con el de los modelos fijos, y la reutilización de modelos de un proyecto a otro es difícil, dado que los factores identificados para un proyecto no tienen por qué ser adecuados para otro proyecto [48, 50].

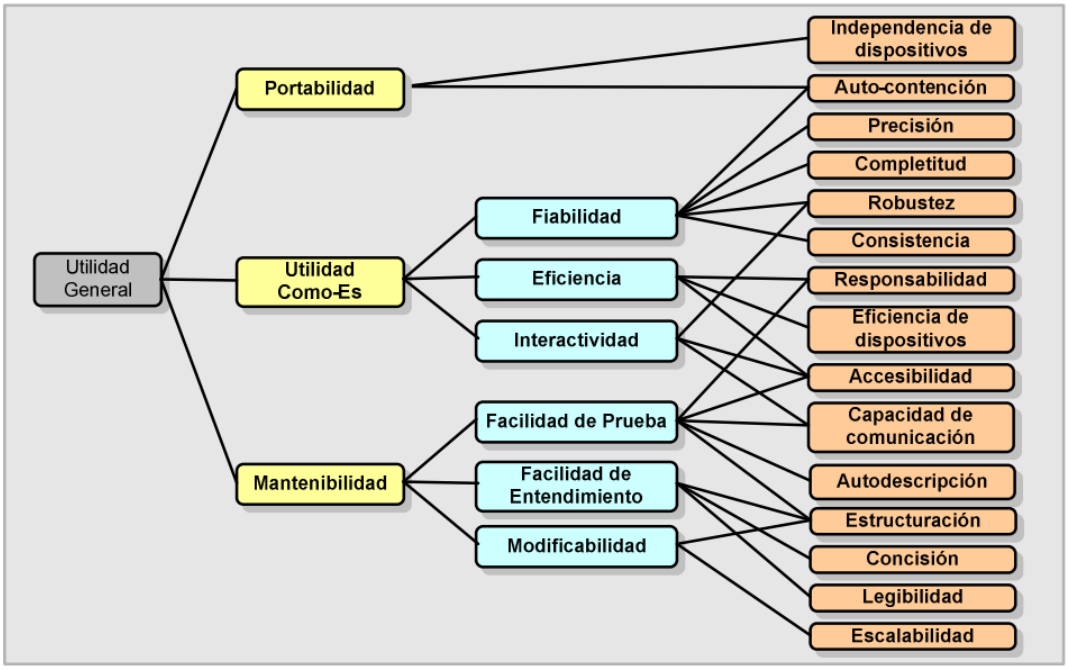


Figura 4.2 Ejemplo de modelo de calidad fijo basado en Boehm et.al.

Fuente: [92]

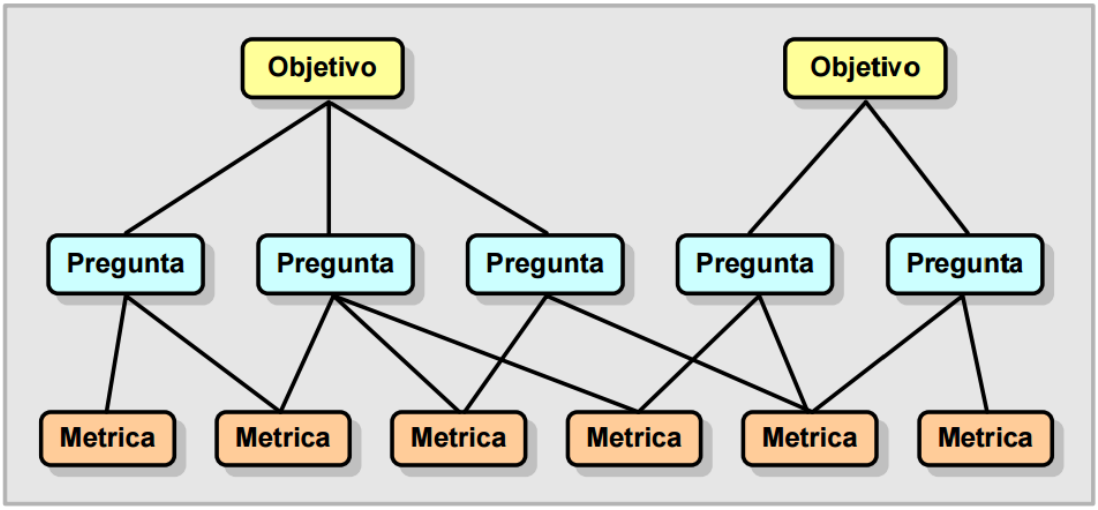


Figura 4.3 a Descomposición de objetivos en preguntas y métricas (método GQM: Goal, Question, Metric)

Objetivo	*Propósito*	Mejorar
	Objeto	Las líneas de tiempo
	Objeto (proceso)	Cambiar tiempo de proceso
	Punto de vista	Administradores
Pregunta 1	¿Cuál es la velocidad de proceso requerida actualmente?	
Métrica 1	• Tiempo promedio del ciclo • Desviación estándar • % de veces fuera del límite	
Pregunta 2	¿Está mejorando el rendimiento?	
Métrica 2	• (Tiempo promedio del ciclo actual × 100) / Tiempo promedio inicial • Calificación subjetiva de los administradores	

Figura 4.3 b Ejemplo de modelo de calidad a medida (método GQM)

Fuente: [92]

Los modelos de calidad mixtos intentan combinar las ventajas de los dos tipos anteriores de modelos. La idea es que exista un conjunto de factores de calidad más abstractos que sean reutilizados en todos los proyectos posibles, y que puedan ser refinados y operacionalizados para un proyecto particular. En este caso podemos destacar como propuestas de este tipo de modelos el ADEQUATE (Horgan et al., 1999), el modelo de Gilb (1988) y el modelo propuesto en el estándar ISO/IEC 9126-1 (2001) que se presenta a continuación [48, 50].

4.3 Estándares de modelos de calidad

Se destacan dos estándares de modelos de calidad ya citados, el estándar IEEE 1061 y el estándar ISO/IEC 9126.

El estándar IEEE 1061 (1998) tiene como objetivo la definición de métricas de software y su uso en la evaluación de componentes software. Fue aprobado en 1992 y revisado y modificado en 1998. Propone la construcción de modelos de calidad a medida adaptados a cada proyecto. No fija ningún factor de calidad, pero sí una clasificación de los factores de los que debe constar un modelo en un nivel más alto y abstracto de factores,

que deben descomponerse en subfactores, que a su vez se descomponen en métricas como se indica en la Figura 4.4.

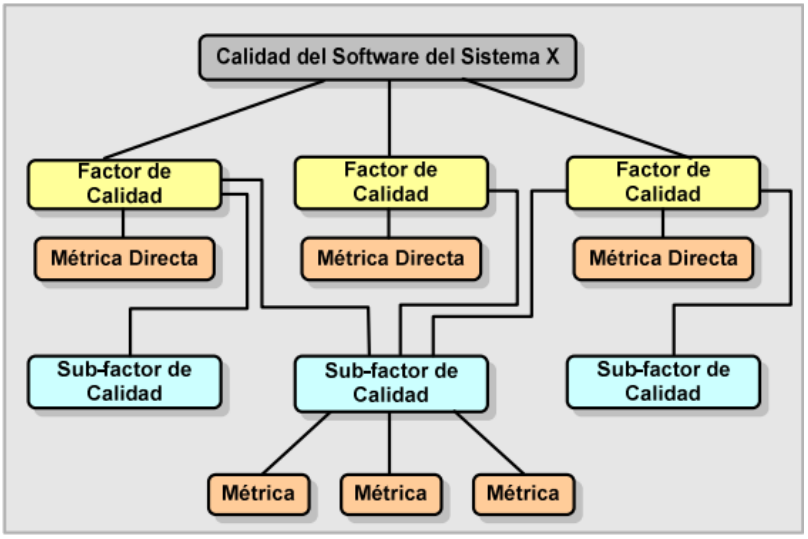


Figura 4.4: Estructura de los modelos de calidad según el estándar IEEE 1061

Fuente: [92]

El estándar ISO/IEC 9126 tiene como objetivo la definición de un modelo de calidad y su uso como marco para la evaluación de software. Como ya se ha mencionado, los modelos de calidad concordantes con este estándar pertenecen a la categoría de modelos mixtos, ya que el estándar propone una jerarquía de factores de calidad clasificados como características, subcaracterísticas y atributos según su grado de abstracción, entre los que se propone un conjunto de factores de partida compuestos de 6 características y 27 subcaracterísticas.

El estándar ISO/IEC 9126 distingue entre calidad interna y calidad externa, e introduce también el concepto de calidad en uso. La calidad interna tiene como objetivo medir la calidad del software mediante factores medibles durante su desarrollo. La calidad externa pretende medir la calidad del software teniendo en cuenta el comportamiento de este software en un sistema del cual forme parte [50].

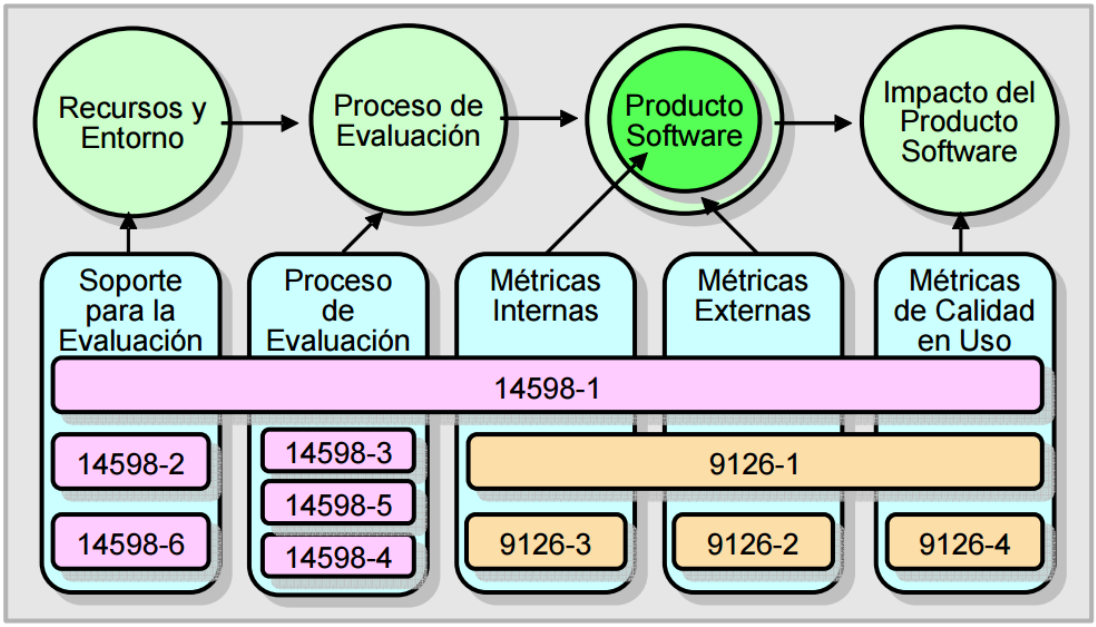


Figura 4.5: Relación entre los estándares 9126 y 14598 de ISO/IEC

Fuente: [92]

4.4 El estándar de calidad ISO/IEC 9126-1

El punto de interés es la parte del catálogo de factores de calidad del estándar ISO/IEC 9126 (ver Figura 4.5) que goza de un reconocimiento más amplio por la comunidad. Ésta es la parte 1 del estándar (ISO, 2001), y concretamente dentro de la misma, nos centramos en los modelos de calidad para la evaluación de la calidad externa del software. Esta decisión se fundamenta en el contexto de utilización de los modelos que presentamos en este capítulo: la evaluación de la calidad de componentes software existentes en el mercado. Para dichos componentes no es factible la evaluación de su calidad interna, debido al desconocimiento sobre cómo han sido desarrollados, así como tampoco una evaluación de su calidad en uso, ya que ésta depende del uso de los componentes una vez seleccionados.

4.5 ISO 25000

CONCEPTO DE ISO 25000

En lo que se refiere a calidad del producto la norma ISO/IEC 25000 proporciona una guía para el uso de las nuevas series de estándares internacionales, llamados Requisitos y Evaluación de Calidad de Productos

de Software (SQuaRE). Constituyen una serie de normas basadas en la ISO 9126 y en la ISO 14598 (Evaluación del Software), y su objetivo principal es guiar el desarrollo de los productos de software con la especificación y evaluación de requisitos de calidad. Establece criterios para la especificación de requisitos de calidad de productos software, sus métricas y su evaluación [51].

El objetivo para la creación de la ISO 25000 fue cubrir los procesos de 9 requerimientos de calidad de software y evaluación de calidad de software, con el apoyo de procesos de medición de calidad.

BENEFICIOS DE LA ISO 25000

- La coordinación de la guía en la medición y evaluación de la calidad de los productos de software.
- Orientación para la especificación de requisitos de calidad de los productos de software.
- La familia ISO 25000 está orientada al producto software, permitiendo definir el modelo de calidad y el proceso a seguir para evaluar dicho producto.

MÉTODO IQMC PARA LA CONSTRUCCIÓN DE MODELOS DE CALIDAD

Según Coral Calero y otros autores, en el libro “Calidad del producto y proceso software”, se explica sobre el método IQMC **[93]**, el cual consiste de siete pasos, que pueden ser simultaneados y/o iterados si se considera oportuno. En el primer paso, el ámbito de calidad es explorado en profundidad y los seis pasos restantes conducen la construcción del modelo de calidad partiendo de las características de calidad, y su descomposición en subcaracterísticas del catálogo ISO/IEC 9126-1 extendido, actualmente ISO 25000 [52].

Los pasos a realizarse son:

Paso 0. Estudio del ámbito del software: Este paso consiste en realizar un estudio del ámbito al cual pertenecen los componentes software para los que se quiere evaluar la calidad.

Paso 1. Determinación de subcaracterísticas de calidad: Teniendo en cuenta que partimos del catálogo ISO/IEC 9126-1 extendido, el añadido de subcaracterísticas no será muy habitual y lo que puede pasar es que alguna de las existentes deba reformularse ligeramente para adaptarla al dominio de interés, o eliminarse en el caso de subcaracterísticas no técnicas.

Paso 2. Refinamiento de la jerarquía de subcaracterísticas: Se descomponen las subcaracterísticas del más bajo nivel de abstracción formando jerarquías de subcaracterísticas. En lo que se refiere a las subcaracterísticas técnicas, al igual que en el paso anterior, el añadido de subcaracterísticas no será muy habitual, excepto en el caso de la descomposición de la subcaracterística Adecuación perteneciente a la características Funcionalidad, pues como se ha comentado anteriormente, esta subcaracterística depende del dominio concreto para el cual se construye el modelo.

Paso 3. Refinamiento de subcaracterísticas en atributos: Este refinamiento tiene como objetivo llegar a tener descompuestas las subcaracterísticas en atributos medibles ya sea de forma directa o indirecta a partir del valor de otros atributos básicos.

Paso 4. Refinamiento de atributos derivados en básicos: Se descomponen los atributos complejos (derivados) hasta obtener atributos básicos, los cuales pueden ser medidos de forma directa.

Paso 5. Establecimiento de relaciones entre factores de calidad: Se establecen las relaciones entre factores de calidad que permiten conocer las dependencias entre los distintos factores de calidad del modelo.

Paso 6. Determinación de métricas para los atributos: Se determinan las métricas para los atributos identificados [48].

En la figura 4. 6 podemos observar de mejor manera los pasos explicados.

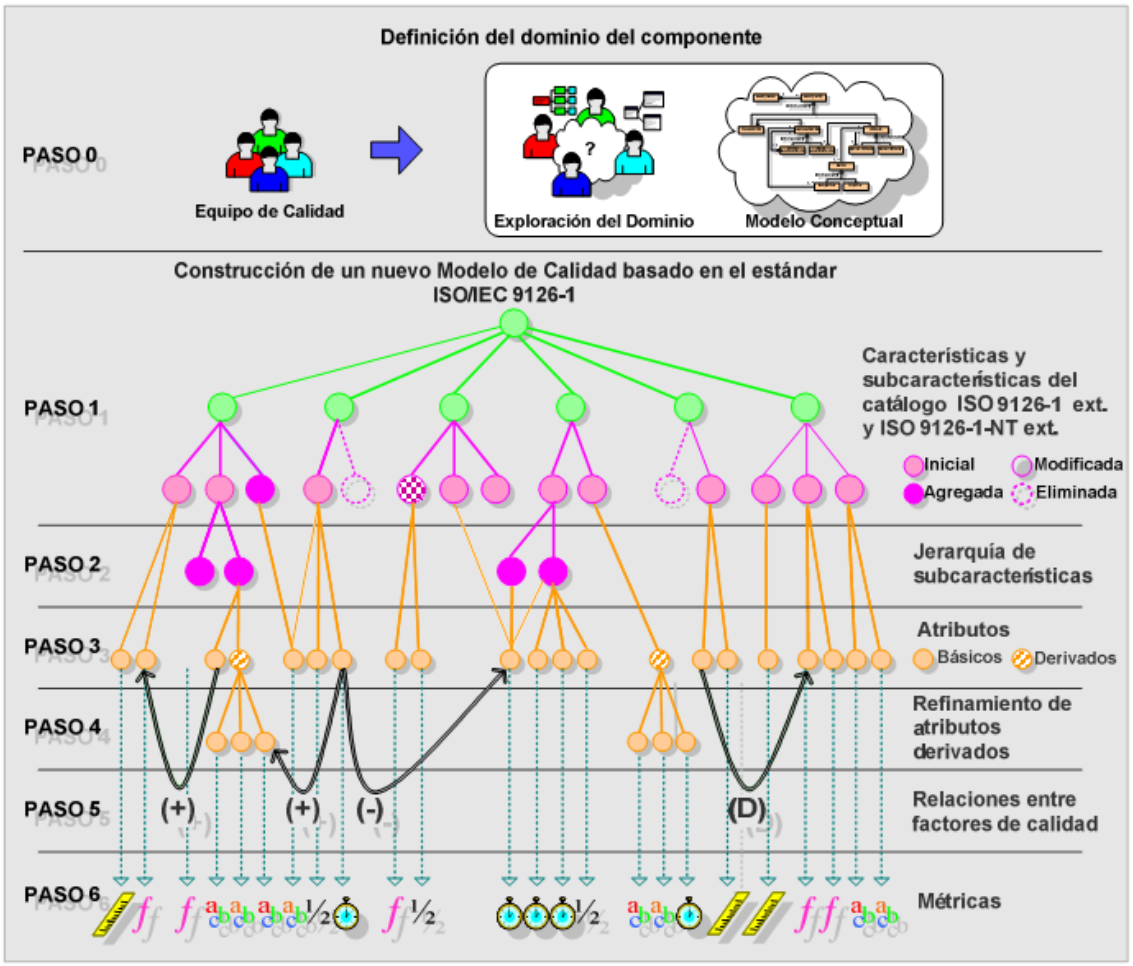


Figura 4.6 Pasos del Modelo IQMC

Fuente: [94]

A continuación se presenta la matriz (ver Tabla 4.7) con los atributos y su justificación para el modelo MONEPS. Adicionalmente, en las figuras 4.8 y 4.9 se muestran las relaciones y estructura de la norma ISO 25000.

4.6 Matriz de atributos de MONEPS y su justificación

Tabla 4.7
Matriz de atributos y justificaciones.

Nº	CODIGO	NOMBRE ATRIBUTO	JUSTIFICACION DEL ATRIBUTO	TIPO
I	A	SEGURIDAD	La seguridad es una característica que involucra la definición de niveles de usuario, seguridad e integridad de la información, lo que presupone que el sistema tendrá un mayor esfuerzo en el tiempo y en el costo.	
1	A.1	DISPONIBILIDAD DE DATOS		Cualitiativo
2	A.2	NIVELES DE SEGURIDAD		Cualitiativo
3	A.3	CONFIDENCIALIDAD DE LA INFORMACION		Cualitiativo
4	A.4	ADMINISTRACION PERFILES USR		Cualitiativo
II	B	OPERABILIDAD PRODUCTO SOFTWARE	La operabilidad es relevante porque permite definir número de usuarios concurrentes al sistema, flexibilidad para la presentación de vistas para el usuario, lo que incide en el tiempo y el costo de desarrollo de un proyecto.	
5	B.1	PARAMETRIZACION		Cualitiativo
6	B.2	ADM GLOBAL DEL SISTEMA		Cualitiativo
7	B.3	CONCURRENCIA		Cualitiativo
III	C	USABILIDAD	La usabilidad es un factor de impacto ya que, la generación de una buena presentación del producto hacia el usuario, involucra que no se escatime tiempo ni dinero, pues este atributo involucra la definición de estilos de interfaces acorde a cada proyecto, lo que incide a su vez en una fácil navegación.	

CONTINÚA →

9	C.2	TIEMPO DE RESPUESTA		Cuantitativo
10	C.3	ESTILOS DE INTERFACES		Cualitiativo
IV	D	COMPORTAMIENTO EN EL TIEMPO	La taza de actualización de la información en un sistema es un factor clave que puede afectar en su rendimiento, y por ende, habrá que tomar la mayor cantidad de consideraciones y excpeciones en aras de mantener un buen rendimiento.	
11	D.1	TAZA DE ACTUALIZACIÓN DE LA INFORMACIÓN		Cuantitativo
V	E	UTILIZACIÓN DE RECURSOS HW	El uso de recursos de hardware incide en el esfuerzo destinado a la construcción del proyecto, ya que no es lo mismo planificar o desarrollar un sistema destinado al uso de un solo equipo, que orientado al uso de servidores, estaciones de trabajo y toda la infraestructura física requerida.	
12	E.1	REQUERIMIENTOS DE HW		Cualitiativo
13	E.2	ARQUITECTURA DE HW		Cualitativo
14	E.3	SERVIDORES		Cuantitativo
15	E.4	ESTACIONES DE TRABAJO		Cuantitativo
VI	F	UTILIZACIÓN DE RECURSOS SW	El uso de frameworks de desarrollo y /o repositorios es fundamental, pues está claro que dicha selección dependerá del tipo de proyecto a desarrollar. Por lo tanto,la incidencia de este atributo es relevante en la estimación del esfuerzo que se genera al desarrollar un proyecto software.	

CONTINÚA →

16	F.1	TIPO DE LENGUAJE		Cualitiativo
17	F.2	PLATAFORMA DESARROLLO		Cualitiativo
VII	G	ADMINISTRACION BDD	La Base de Datos o repositorio es importante, pues depende de la refactorización a nivel de cambios en los requisitos de software. Además, consideraciones como el uso de licencias y el tamaño de la Base de Datos podrían determinar alteraciones críticas en los tiempos y costos proyectados.	
18	G.1	TIPO DE LICENCIA		Cualitativo
19	G.2	TAMAÑO DE BDD		Cuantitativo
20	G.3	NORMALIZACION DE BDD		Cualitiativo
21	G.4	REFACTORIZACION DE BDD		Cualitativo
22	G.5	TIPO DE BDD		Cualitiativo
VIII	H	ARQUITECTURA DE SW	La cantidad de componentes identificados en la arquitectura de software y sus interrelaciones son factores determinantes para la complejidad de los sistemas, y por ende afectará al esfuerzo de los proyectos de software.	
23	H.1	TIPO DE ARQUITECTURA		Cualitiativo
24	H.2	CAPAS DE ARQUITECTURA		Cuantitativo
IX	I	FUNCIONALIDAD	La complejidad y la cantidad de requisitos de usuario son factores críticos en la determinación del esfuerzo y el costo asociado a los proyectos de software.	
25	I.1	NUMERO DE REQUISITOS FUNCIONALES		Cuantitativo

CONTINÚA →

26	I.2	NUMERO DE CASOS DE USO ANÁLISIS		Cuantitativo
27	I.3	NUMERO DE CASOS DE USO DE DISEÑO		Cuantitativo
28	I.4	COMPLEJIDAD DEL SISTEMA		Cualitativo
X	J	PARADIGMA DE PROGR. USADO	Un adecuado proceso de abstracción del sistema incidirá en la terminación temprana o tardía de un sistema. Es posible (p.e.) que a veces funcione mejor un paradigma estructurado o en otros casos un paradigma orientado a objetos	
29	J.1	PARADIGMA USADO		Cualitativo
30	J.2	NUMERO DE ELEMENTOS IDENTIFICADOS		Cuantitativo
XI	K	MANTENIBILIDAD	Lograr que un sistema tenga un nivel aceptable de estabilidad involucra la generación de un plan de pruebas más minucioso, así como una rigurosa depuración de los errores detectados.	
31	K.1	ESCALABILIDAD		Cualitiativo
		ESTABILIDAD		
32	K.2	Frecuencia de actualización por corrección de errores		Cualitiativo
33	K.3	Frecuencia de nuevas versiones		Cualitiativo
XII	L	PORTABILIDAD	Consideraciones como la interacción con otros sistemas y la adaptabilidad de un Sw, podrían involucrar un mayor esfuerzo del equipo de desarrollo para llevar adelante la construcción de los nuevos sistemas.	
34	L.1	ADAPTABILIDAD		Cualitiativo

CONTINÚA →

35	L.2	FACILIDAD DE INSTALACION		Cualitiativo
36	L.3	INTERACCION CON OTROS SISTEMAS		Cualitiativo
XIII	M	PERSONAL	Este es un aspecto relevante que determinará la duración de un proyecto de Sw, pues, es bien sabido que cada nuevo proyecto plantea nuevos desafíos para el equipo de desarrollo, y por lo tanto, aspectos como la experiencia y la definición acertada de roles en el personal, puede incidir en el éxito de los proyectos.	
37	M.1	EXPERIENCIA DEL EQUIPO DE DESARROLLO EN PROYECTOS TIPO		Cualitiativo
		ROLES Y CANTIDAD		
38	M.2	ROL: ARQUITECTO		Cuantitativo
39	M.3	ROL: PROGRAMADOR		Cuantitativo
40	M.4	ROL: ANALISTA		Cuantitativo
41	M.5	ROL: PLANIFICADOR		Cuantitativo
42	M.6	ROL: TESTER		Cuantitativo

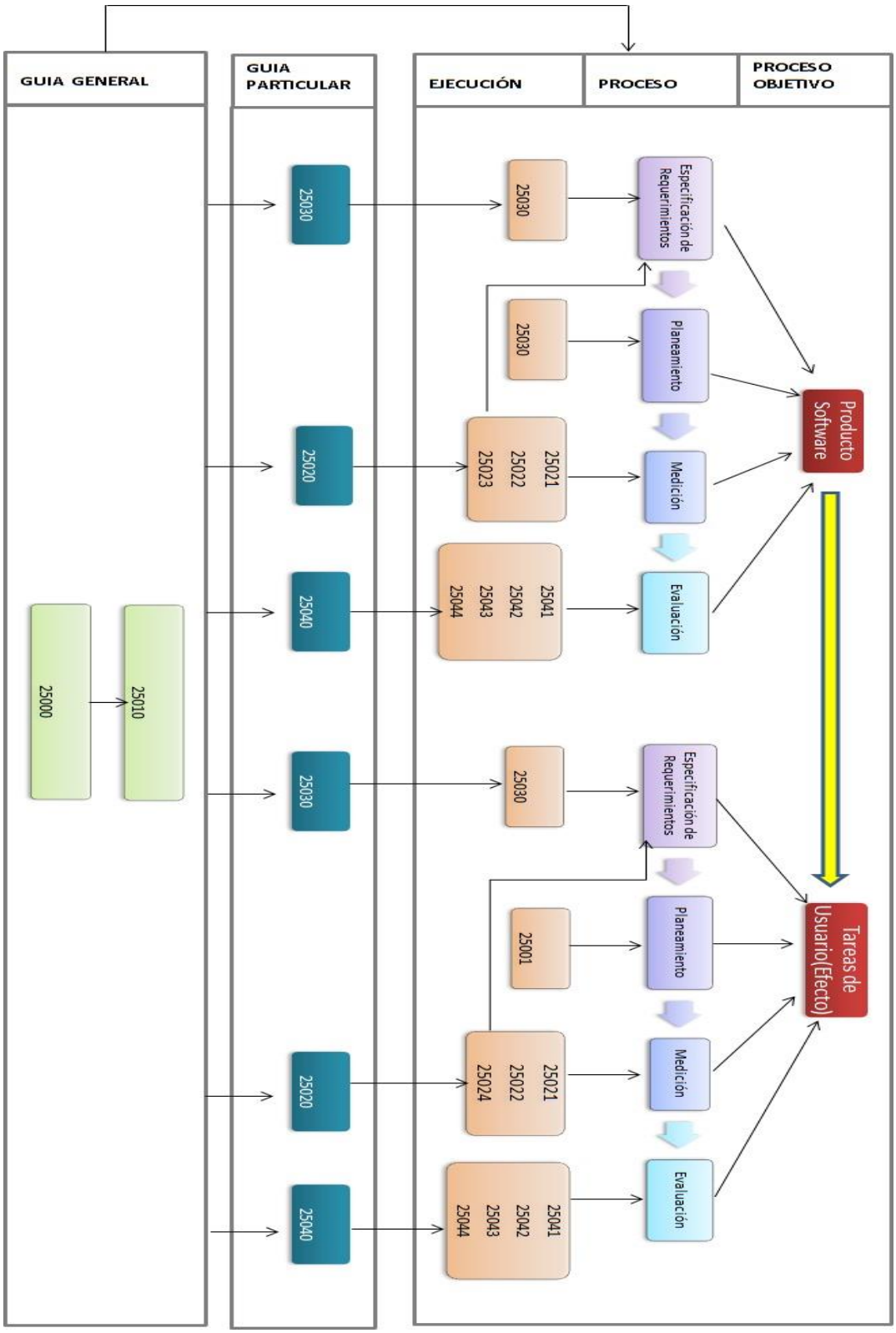


Figura 4.8 Matriz de estudio y análisis de la ISO 25000. Fuente: Square(Software Product Quality Requeriments and Evaluation) Requisitos y Evaluación de Calidad de Productos Software

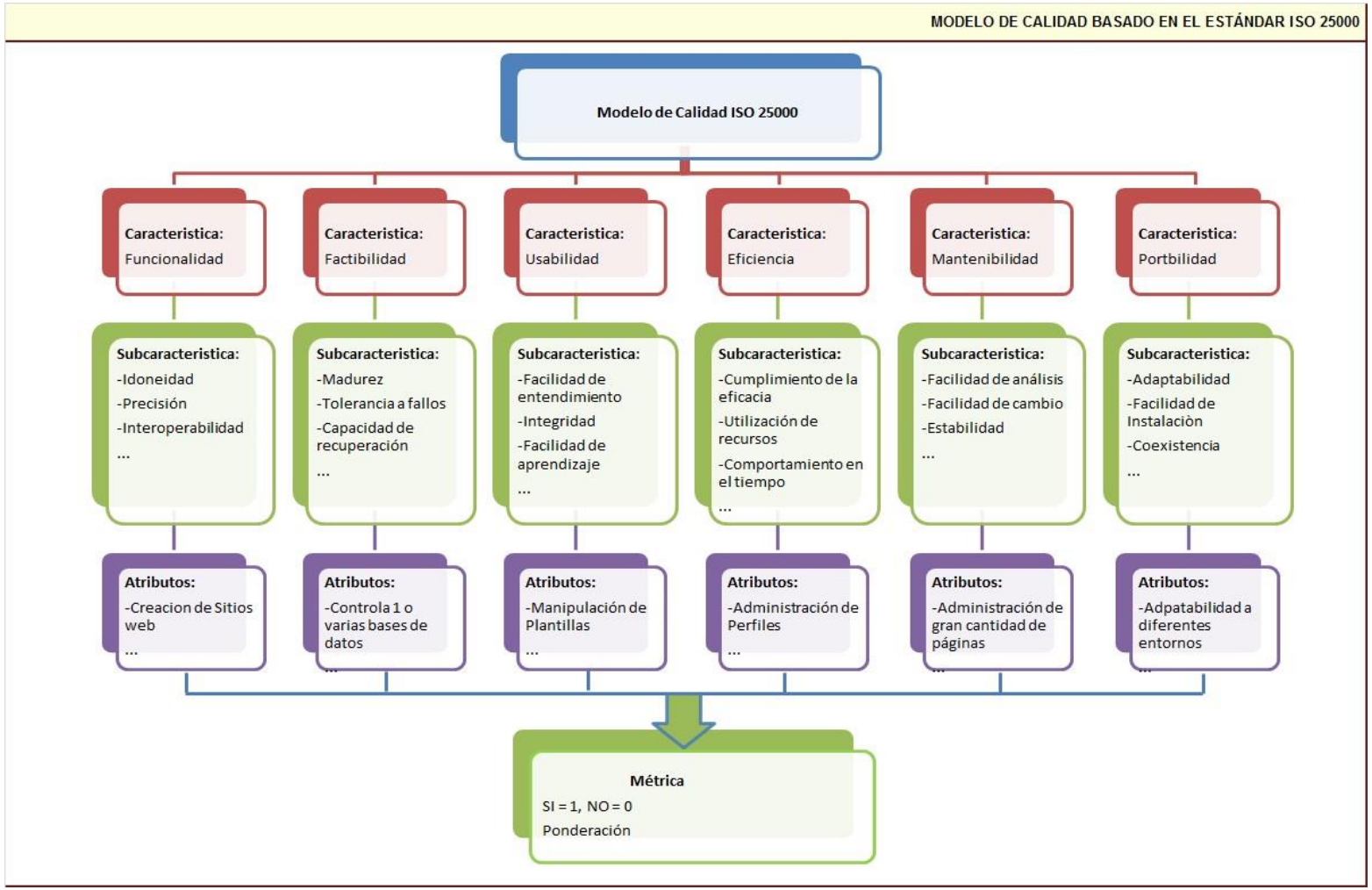


Figura 4.9 Modelo de calidad basado en la ISO 25000
Fuente: http:// iso25000.com/index.php/normas-iso-25000/iso-2501

4.7 Descripción general de las características de MONEPS

A continuación se detallan las definiciones de la norma ISO 25000 [48], referidas a las características que sirvieron de base para MONEPS.

Funcionalidad:

Es el conjunto de atributos que se refieren a la existencia de un conjunto de funciones y sus propiedades específicas. Las funciones cumplen unos requerimientos o satisfacen unas necesidades implícitas. Las sub-características de la Funcionalidad son:

- Idoneidad.
- Precisión.
- Interoperabilidad.
- Seguridad.
- Cumplimiento de la funcionalidad.

Factibilidad:

Es el conjunto de atributos que se refieren a la capacidad del software de mantener su nivel de rendimiento bajo unas condiciones especificadas durante un período definido. Las sub-características de la factibilidad son:

- Madurez.
- Tolerancia a fallos.
- Capacidad de recuperación.
- Cumplimiento de la fiabilidad.

Usabilidad:

Es el conjunto de atributos que se refieren al esfuerzo necesario para usarlo, y sobre la valoración individual de tal uso, por un conjunto de usuarios definidos e implícitos. Las sub-características de la Facilidad de Uso son:

- Integridad.
- Facilidad de aprendizaje.

- Operatividad.
- Cumplimiento de la usabilidad.

Eficiencia:

Es el conjunto de atributos que se refieren a las relaciones entre el nivel de rendimiento del software y la cantidad de recursos utilizados bajo unas condiciones predefinidas. Las sub-características de la Eficiencia son:

- Cumplimiento de la eficacia.
- Utilización de recursos.
- Comportamiento en el tiempo.

Mantenimiento:

Es el conjunto de atributos que se refieren al esfuerzo necesario para hacer modificaciones especificadas. Las sub-características de la Facilidad de Mantenimiento son:

- Facilidad de análisis.
- Facilidad de cambio.
- Estabilidad.
- Facilidad de pruebas.
- Cumplimiento del mantenimiento.

Portabilidad:

Es el conjunto de atributos que se refieren a la habilidad del software para ser transferido desde un entorno a otro. Las sub-características de la portabilidad son [48]:

- Adaptabilidad.
- Facilidad de Instalación.
- Coexistencia.
- Intercambiabilidad.
- Cumplimiento de portabilidad.

CAPÍTULO V

IMPLEMENTACIÓN DE MONEPS

5.1 Introducción

El carácter predictivo de las Redes Neuronales Artificiales (RNA) las convierte en una herramienta muy promisoria para las tareas de estimación de tiempo y costo en proyectos de software. Sin embargo, antes de poder utilizarlas, es necesario identificar la topología adecuada que permita llevar a efecto tales actividades. Además, será de vital importancia detectar los insumos necesarios para que la RNA sea capaz de aprender a realizar estimaciones aceptables de esfuerzo.

En el presente capítulo, se plantea un Modelo Neuronal para la Estimación del Esfuerzo en Proyectos de Software (MONEPS), basado en las características relevantes de la calidad de software (referidas en el capítulo anterior), así como en consideraciones estructurales y funcionales de las RNA's.

5.2 Características de las RNA's

Las RNA's están inspiradas en las redes neuronales biológicas del cerebro humano [53]. Estas redes están constituidas por elementos que se comportan de forma similar a la neurona biológica (Figura 5.1a) en sus funciones más comunes. Estos elementos están organizados de una forma parecida a la que presenta el cerebro humano (Figura 5.1b).

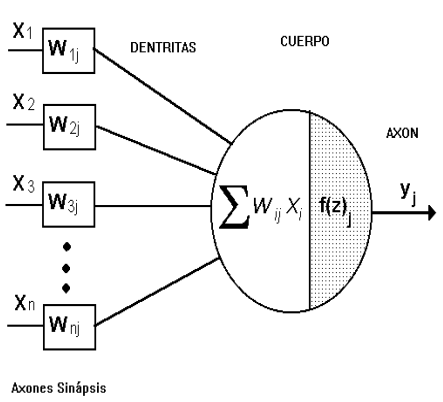


Figura 5.1a Neurona artificial

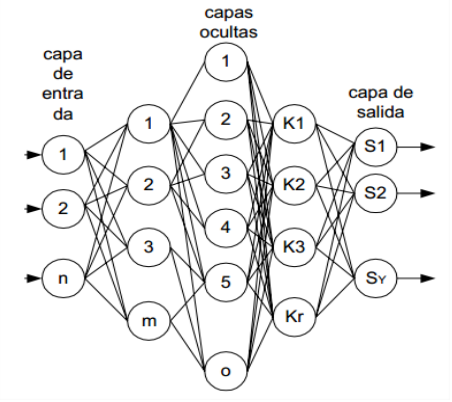


Figura 5.1b Ejemplo de una RNA

Fuente:[95,96]

Las RNA’s presentan una serie de características propias del cerebro, entre las cuales podemos citar [54]:

- **Aprendizaje:** adquirir el conocimiento de una cosa por medio del estudio, ejercicio o experiencia. Estas redes pueden cambiar su comportamiento en función del entorno; si se les muestra un conjunto de entradas, ellas mismas se ajustan para producir unas salidas consistentes.
- **Generalización:** capacidad de extender una cosa o concepto; las RNA’s pueden generalizar automáticamente los conceptos u objetos debido a su propia estructura y naturaleza. Estas redes ofrecen, dentro de un margen, respuestas correctas a entradas que presentan pequeñas variaciones debido a los efectos de ruido o distorsión.
- **Abstracción:** aislar mentalmente o considerar por separado las cualidades de un objeto. Algunas RNA’s son capaces de abstraer la esencia de un conjunto de entradas que aparentemente no presentan aspectos comunes o relativos.

Como ya se referenció en el capítulo 2, en las RNA’s, las unidades de proceso (neuronas) se caracterizan por tener una función de activación que convierte la entrada total recibida de otras unidades en un valor de salida, el cual hace la función de tasa de disparo de la neurona.

Las conexiones sinápticas se simulan mediante conexiones ponderadas; la fuerza o peso de la conexión cumple el papel de la efectividad de la sinapsis. Las conexiones determinan si es posible que una unidad influya sobre otra; los pesos definen la intensidad de la influencia.

Una unidad de proceso recibe varias entradas procedentes de las salidas de otras unidades de proceso. La entrada total de una unidad de proceso se suele calcular como la suma de todas las entradas ponderadas, es decir, multiplicadas por el peso de la conexión. El efecto inhibitorio o excitatorio de las sinapsis se logra usando pesos negativos o positivos respectivamente.

5.3 Aplicaciones de las RNA's

Se muestra a continuación algunas aplicaciones de las RNA's:

- **Conversión Texto a Voz**: uno de los principales promotores de la computación neuronal en esta área es Terrence Sejnowski. La conversión texto-voz consiste en cambiar los símbolos gráficos de un texto en lenguaje hablado. El sistema de computación neuronal presentado por Sejnowski y Rosemberg, llamado NetTalk, convierte texto en fonemas, y con la ayuda de un sintetizador de voz (Dectalk), genera voz a partir de un texto escrito [55]:

  La ventaja que ofrece la computación neuronal frente a las tecnologías tradicionales en la conversión texto-voz es la propiedad de eliminar la necesidad de programar un complejo conjunto de reglas de pronunciación en el ordenador. A pesar de que el sistema NetTalk ofrece un buen comportamiento, la computación neuronal para este tipo de aplicación abre posibilidades de investigación y expectativas de desarrollo comercial.

- **Procesamiento del Lenguaje Natural**: incluye el estudio de cómo se construyen las reglas del lenguaje. Los científicos del conocimiento Rumelhart y McClelland han integrado una red neuronal de proceso natural del lenguaje. El sistema realizado ha aprendido el tiempo verbal *pass tense* de los verbos en Inglés. Las características propias de la computación neuronal, como la capacidad de generalizar a partir de datos incompletos y la capacidad de abstraer, permiten al sistema generar buenos pronósticos para verbos nuevos o verbos desconocidos [56]:

- **Compresión de Imágenes**: la compresión de imágenes es la transformación de los datos de una imagen a una representación diferente que requiera menos memoria o que se pueda reconstruir

una imagen imperceptible. Cottrel, Munro y Zisper de la Universidad de San Diego y Pisttburgh han diseñado un sistema de compresión de imágenes utilizando una red neuronal con un factor de compresión de 8:1 [57].

- **Reconocimiento de Caracteres:** es el proceso de interpretación visual y de clasificación de símbolos. Los investigadores de Nestor, Inc. han desarrollado un sistema de computación neuronal que, tras el entrenamiento con un conjunto de tipos de caracteres de letras, es capaz de interpretar un tipo de carácter o letra que no haya visto con anterioridad. **[97]**

- **Reconocimiento de Patrones en Imágenes:** una aplicación típica es la clasificación de objetivos detectados por un sonar. Existen varias RNA's basadas en el popular algoritmo de aprendizaje Backpropagation, cuyo comportamiento es comparable con el de los operadores humanos. Otra aplicación normal es la inspección industrial [58].

- **Problemas de Combinatoria:** en este tipo de problemas la solución mediante cálculo tradicional requiere un tiempo de proceso que es exponencial con el número de entradas. Un ejemplo es el problema del agente viajero; el objetivo es elegir el camino más corto posible que debe realizar el agente para cubrir un número limitado de ciudades en un área geográfica específica. Este tipo de problema ha sido abordado con éxito por Hopfield y el resultado de su trabajo ha sido el desarrollo por una RNA que ofrece buenos resultados para este problema de combinatoria [59].

5.4 RNA para MONEPS

5.4.1 Topología neuronal para MONEPS

La estructura neuronal seleccionada para MONEPS es de tipo Backpropagation (ver figura 5.1b), pues, aparte de cumplir con las características descritas en el numeral anterior, estas redes también presentan otras bondades [60, 61]:

- 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).
- Uno de los pocos inconvenientes de estas redes es que, no existe una regla clara que permita determinar la cantidad apropiada de capas ocultas (y cantidad de neuronas en éstas) para facilitar la labor de aprendizaje. También, pueden quedar "sobreentrenadas" **[98]** debido al uso excesivo de pesos sinápticos.

5.4.2 Diseño neuronal de MONEPS

Como ya quedó evidenciado en el capítulo 4, cada componente de la matriz de atributos identificada, se corresponderá con una neurona de la capa de entrada de la red. 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 de desarrollo). En la figura 5.2 se puede apreciar la RNA simplificada para MONEPS.

Como se puede apreciar, la red tiene conexiones unidireccionales que van desde la capa de entrada hasta la capa de salida. Además, las

neuronas están conectadas solamente con otras ubicadas en la siguiente capa adyacente.

En la misma figura 5.2 se pueden apreciar sólo algunos atributos que se han constituido en las entradas que posee la red neuronal. Para un mayor detalle es necesario consultar la matriz de atributos del numeral 4.6

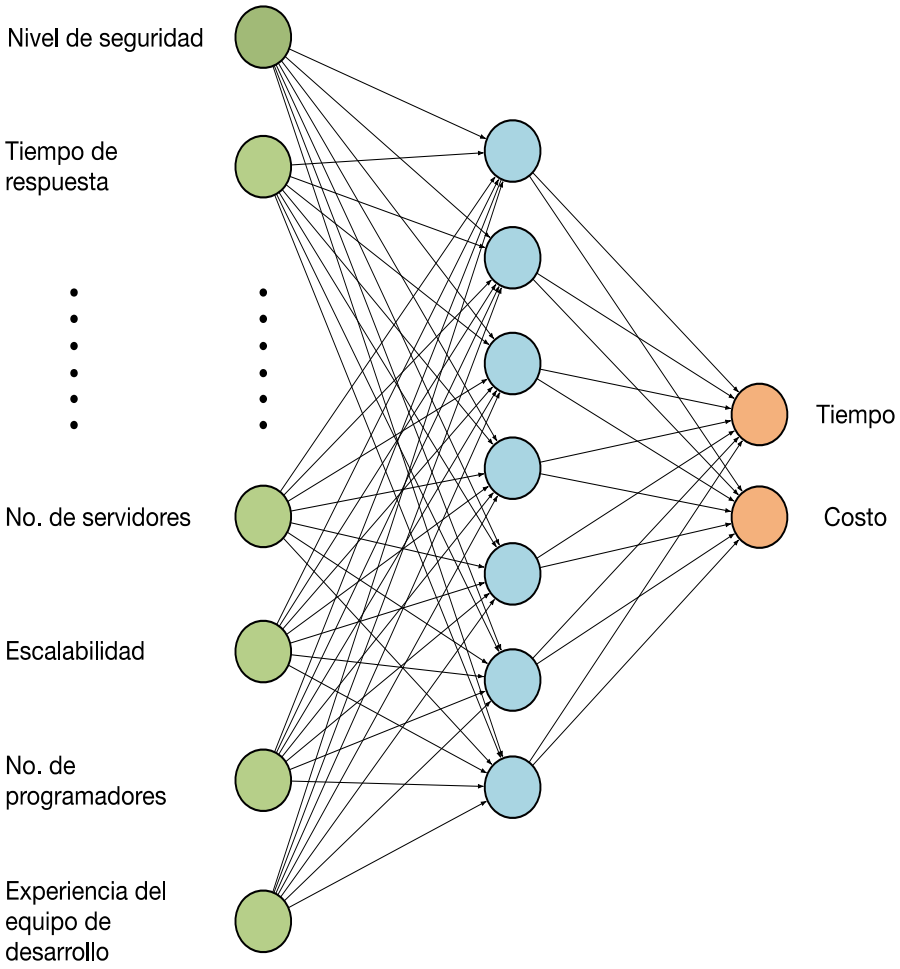


Figura 5.2 RNA utilizada en MONEPS
Fuente: [99]

La implementación de la RNA fue factible mediante el uso de la herramienta automatizada denominada JustNN **[100]**, cuyas principales características serán detalladas en el siguiente numeral.

5.5 La herramienta JustNN

Para la construcción de redes neuronales artificiales existe un abanico interesante de opciones: se las puede construir utilizando un lenguaje de programación convencional (p.e. Java, C++, C#), se puede utilizar una herramienta de propósito específico (p.e. Tiberius, EasyNN, Phytia, Neural

Networks), o se pueden utilizar herramientas que comparten características de las dos primeras categorías (p.e. Matlab).

Se ha optado por el uso de la herramienta JustNN ya que, aparte de tener una interfaz de usuario muy transparente, proporciona todo un conjunto de opciones que posibilitan el diseño y construcción de RNA's funcionales. Es la parte gratuita de un conjunto de productos que comprende EasyNN-plus y SwingNN (producidos por Neural Planner Software Ltd.), referidos a la construcción, uso y optimización de redes neuronales en cascada.

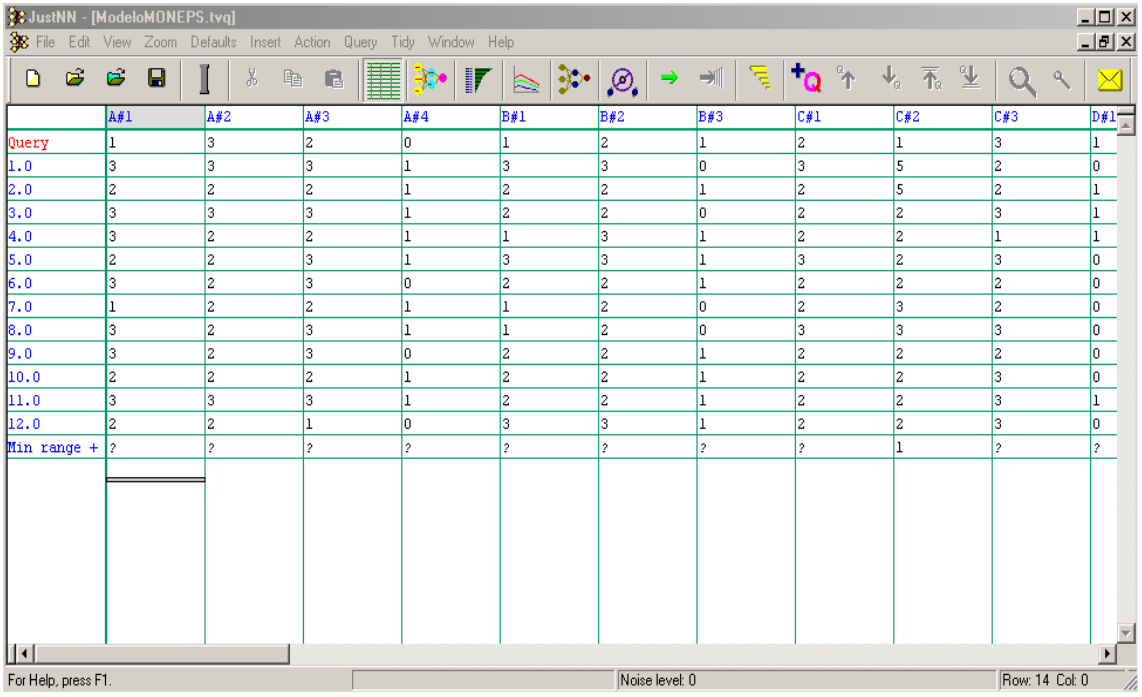
Con JustNN la construcción de redes neuronales, a partir de los datos disponibles, no puede ser más sencilla. Importar los datos desde formatos txt, csv, xls, bmp o archivos binarios se lo puede hacer con sólo unos pocos clicks. Se puede crear una red neuronal multicapa para que aprenda de los datos que el usuario le provea; se puede validar el aprendizaje mientras la red está aprendiendo; el usuario puede probar o consultar la red con nuevos datos para producir resultados y ver qué entradas son realmente importantes.

La rejilla de JustNN no tiene límite para el número de filas (ejemplos de entrenamiento) y puede tener un máximo de 1.000 columnas (atributos de cada ejemplo). Tampoco hay límites en los nodos y conexiones; además esta herramienta es muy rápida y fácil de usar. En la figura 5.3 se muestra la ventana principal de trabajo de JustNN.

JustNN es un software para la construcción de redes neuronales para Microsoft Windows y, por ende, compatible con aplicaciones de esta plataforma. Los usuarios que utilizan este software pueden crear redes neuronales en cascada de manera fácil. Además, entre otras prestaciones, esta herramienta permite:

- Importar información desde documentos de texto, hojas de cálculo, imágenes o archivos binarios
- Utilizar varias funciones de edición y pre-formato.
- Construir redes neuronales directamente en el grid (rejilla).
- Entrenar, validar y consultar las redes neuronales.

Para un mejor entendimiento de la herramienta se puede consultar el manual de JustNN en el Anexo 1 del presente documento.



	A#1	A#2	A#3	A#4	B#1	B#2	B#3	C#1	C#2	C#3	D#1
Query	1	3	2	0	1	2	1	2	1	3	1
1.0	3	3	3	1	3	3	0	3	5	2	0
2.0	2	2	2	1	2	2	1	2	5	2	1
3.0	3	3	3	1	2	2	0	2	2	3	1
4.0	3	2	2	1	1	3	1	2	2	1	1
5.0	2	2	3	1	3	3	1	3	2	3	0
6.0	3	2	3	0	2	2	1	2	2	2	0
7.0	1	2	2	1	1	2	0	2	3	2	0
8.0	3	2	3	1	1	2	0	3	3	3	0
9.0	3	2	3	0	2	2	1	2	2	2	0
10.0	2	2	2	1	2	2	1	2	2	3	0
11.0	3	3	3	1	2	2	1	2	2	3	1
12.0	2	2	1	0	3	3	1	2	2	3	0
Min range +	?	?	?	?	?	?	?	?	1	?	?

Figura 5.3 Ventana principal de JustNN

5.6 Comportamiento de la red Backpropagation

La Backpropagation es una red en cascada que utiliza un conjunto de patrones (o ejemplos) de entrenamiento que le permitirán aprender a realizar alguna actividad requerida por el diseñador de la red. Una vez que se ha aplicado un patrón a la entrada de la red (a manera de estímulo), éste se propaga desde la primera capa a través de las capas superiores de la red, hasta generar una salida. La señal de salida obtenida se compara con la salida deseada y se calcula una señal de error en cada una de las neuronas de salida [62].

Las errores de las salidas se propagan hacia atrás, partiendo de la capa de salida, hacia todas las neuronas de la capa oculta que contribuyen directamente a la salida. Sin embargo las neuronas de la capa oculta solamente reciben una fracción de la señal total del error, basándose aproximadamente en la contribución relativa que haya aportado cada neurona a la salida original. Este proceso se repite, capa por capa, hasta que todas las neuronas de la red hayan recibido una señal de error que

describa su contribución relativa al error total. Basándose en la señal de error percibida, se actualizan los pesos de conexión de cada neurona, para hacer que la red converja hacia un estado que permita clasificar correctamente todos los patrones de entrenamiento [33].

La importancia de este proceso consiste en que, a medida que se entrena la red, las neuronas de las capas intermedias se organizan a sí mismas de tal modo que las distintas neuronas *aprenden* a reconocer diversas características del espacio total de entrada. Después del entrenamiento, cuando se les presente un patrón arbitrario de entrada que contenga ruido o que esté incompleto, las neuronas de la capa oculta de la red responderán con una salida activa si la nueva entrada contiene un patrón que se asemeje a aquella característica que las neuronas individuales hayan aprendido a reconocer durante su entrenamiento. Y a la inversa, las unidades de las capas ocultas tienen una tendencia a inhibir su salida si el patrón de entrada no contiene la característica para reconocer, para la cual han sido entrenadas [63].

Varias investigaciones han demostrado que, durante el proceso de entrenamiento, la red Backpropagation tiende a desarrollar relaciones internas entre neuronas con el fin de organizar los datos de entrenamiento en clases. Esta tendencia se puede extrapolar, para llegar a la hipótesis consistente en que todas las unidades de la capa oculta de una Backpropagation son asociadas de alguna manera a características específicas del patrón de entrada como consecuencia del entrenamiento, es decir, la red ha encontrado una representación interna que le permite generar las salidas deseadas cuando se le dan las entradas, en el proceso de entrenamiento. Esta misma representación interna se puede aplicar a entradas que la red no haya visto antes, y la red clasificará estas entradas según las características que compartan con los ejemplos de entrenamiento [64].

5.7 Entrenamiento de la red

Antes que la RNA pueda realizar estimaciones de tiempo y costo para proyectos de software, es necesario entrenarla. Esta actividad es posible debido a que, como se describió en el numeral anterior, un conjunto de ejemplos (patrones de entrenamiento) irá modificando cada uno de los pesos sinápticos de la red hasta que ésta tenga un desempeño satisfactorio. Los patrones de entrenamiento (proyectos académicos de software) y sus respectivas codificaciones pueden consultarse en el Anexo 3 del presente documento.

El desempeño de la RNA está dado por el *error cuadrático medio* **[101]** en la respuesta de la red (nodos de salida) cuando todo el lote de ejemplos (denominado también *época*) ha sido procesado en el entrenamiento. En el caso de MONEPS, el error medio cuantificará la diferencia que hay entre la respuesta esperada y la respuesta obtenida en la red, para los diversos valores de costo (en dólares) y tiempo (en meses) de los ejemplos que alimentan la red.

Como se referenció en el numeral 5.4.2, cada ejemplo de entrenamiento estará caracterizado por los atributos que se describen a continuación (ver Tabla 5.4):

Tabla 5.4
Lista de atributos (en total 42) que posee cada ejemplo de entrenamiento de MONEPS.

N°	CODIGO	NOMBRE ATRIBUTO	TIPO	VALORES
I	A	SEGURIDAD		
1	A.1	DISPONIBILIDAD DE DATOS	Cualitiativo	1:baja, 2:media, 3:alta
2	A.2	NIVELES DE SEGURIDAD	Cualitiativo	1:baja, 2:media, 3:alta
3	A.3	CONFIDENCIALIDAD DE LA INFORMACION	Cualitiativo	1:baja, 2:media, 3:alta
4	A.4	ADMINISTRACION PERFILES USR	Cualitiativo	0:no, 1:si
II	B	OPERABILIDAD PRODUCTO SOFTWARE		
5	B.1	PARAMETRIZACION	Cualitiativo	1:baja, 2:media, 3:alta
6	B.2	ADM GLOBAL DEL SISTEMA	Cualitiativo	1:baja, 2:media, 3:alta
7	B.3	CONCURRENCIA	Cualitiativo	0:no permite concurrencia de Usr 1:permite concurrencia de Usr
III	C	USABILIDAD		
8	C.1	FACILIDAD DE NAVEGACION	Cualitiativo	1:baja, 2:media, 3:alta
9	C.2	TIEMPO DE RESPUESTA	Cuantitativo	1,2,3… segundos
10	C.3	ESTILOS DE INTERFACES	Cualitiativo	1:baja, 2:media, 3:alta
IV	D	COMPORTAMIENTO EN EL TIEMPO		
11	D.1	TAZA DE ACTUALIZACIÓN DE LA INFORMACIÓN	Cuantitativo	0,1,2,3… veces al día
V	E	UTILIZACIÓN DE RECURSOS HW		
12	E.1	REQUERIMIENTOS DE HW	Cualitiativo	0:bajo, 1:medio, 2:alto (Inversamente proporcional al número de recursos)
13	E.2	ARQUITECTURA DE HW	Cualitativo	0: no tiene arquitectura, 1: tiene arquitectura
14	E.3	SERVIDORES	Cuantitativo	1,2,3… servidores
15	E.4	ESTACIONES DE TRABAJO	Cuantitativo	1,2,3 …estaciones de trabajo
VI	F	UTILIZACIÓN DE RECURSOS SW		
16	F.1	TIPO DE LENGUAJE	Cualitiativo	1:imperativo, 2:declarativo, 3:orientado a objetos, 4: orientado al problema
17	F.2	PLATAFORMA DESARROLLO	Cualitiativo	1:WEB, 2:Desktop

CONTINÚA →

VII	G	ADMINISTRACION BDD		
18	G.1	TIPO DE LICENCIA	Cualitativo	0:libre, 1:propietario
19	G.2	TAMAÑO DE BDD	Cuantitativo	1,2,3… tablas
20	G.3	NORMALIZACION DE BDD	Cualitiativo	1: forma especial Boyce Codd, 2:cuarta FN, 3: tercera FN, 4:2da FN, 5: no normalizada
21	G.4	REFACTORIZACION DE BDD	Cualitativo	0:no,1: si
22	G.5	TIPO DE BDD	Cualitiativo	0: no relacional, 1: relacional
VIII	H	ARQUITECTURA DE SW		
23	H.1	TIPO DE ARQUITECTURA	Cualitiativo	1:cliente-servidor,2: Modelo Vista Controlador, 3:Factory, 4:Facade, 5:otros
24	H.2	CAPAS DE ARQUITECTURA	Cuantitativo	1:capa a capa , 2: varias capas.
IX	I	FUNCIONALIDAD		
25	I.1	NUMERO DE REQUISITOS FUNCIONALES	Cuantitativo	1,2,3 … requisitos funcionales
26	I.2	NUMERO DE CASOS DE USO ANÁLISIS	Cuantitativo	1,2,3 … casos de uso de análisis
27	I.3	NUMERO DE CASOS DE USO DE DISEÑO	Cuantitativo	1,2,3 … casos de uso de diseño
28	I.4	COMPLEJIDAD DEL SISTEMA	Cualitativo	1:bajo, 2:medio, 3:alto
X	J	PARADIGMA DE PROGR. USADO		
29	J.1	PARADIGMA USADO	Cualitativo	1: Progr. orientada a objetos, 2: Progr. estructurada, 3: Progr. basada en agentes, 4: Progr. basada en aspectos, 5: Otros
30	J.2	NUMERO DE ELEMENTOS IDENTIFICADOS	Cuantitativo	1, 2, 3 …clases, funciones, agentes, aspectos, u otros
XI	K	MANTENIBILIDAD		
31	K.1	ESCALABILIDAD	Cualitiativo	1:baja, 2:media, 3:alta
		ESTABILIDAD		
32	K.2	Frecuencia de actualización por corrección de errores	Cualitiativo	1:baja, 2:media, 3:alta
33	K.3	Frecuencia de nuevas versiones	Cualitiativo	0:ninguna, 1:baja, 2: media, 3:alta

CONTINÚA →

XII	L	PORTABILIDAD		
34	L.1	ADAPTABILIDAD	Cualitiativo	1:baja, 2:media, 3:alta
35	L.2	FACILIDAD DE INSTALACION	Cualitiativo	1:baja, 2:media, 3:alta
36	L.3	INTERACCION CON OTROS SISTEMAS	Cualitiativo	0:no, 1:si
XIII	M	PERSONAL		
37	M.1	EXPERIENCIA DEL EQUIPO DE DESARROLLO EN PROYECTOS TIPO	Cualitiativo	1:baja, 2:media, 3:alta
		ROLES Y CANTIDAD		
38	M.2	ROL: ARQUITECTO	Cuantitativo	0,1,2,3…personas
39	M.3	ROL: PROGRAMADOR	Cuantitativo	0,1,2,3…personas
40	M.4	ROL: ANALISTA	Cuantitativo	0,1,2,3…personas
41	M.5	ROL: PLANIFICADOR	Cuantitativo	0,1,2,3…personas
42	M.6	ROL: TESTER	Cuantitativo	0,1,2,3…personas

Como ejemplos de entrenamiento para MONEPS, se consideraron 9 proyectos de desarrollo de software, realizados por estudiantes de los últimos niveles de la carrera de Ingeniería en Sistemas e Informática de la Universidad de las Fuerzas Armadas-ESPE.

En la figura 5.5 se tiene una vista parcial de la red neuronal implementada en la herramienta JustNN. Como se mencionó anteriormente, MONEPS posee 42 neuronas en la capa de entrada y 2 neuronas en la capa de salida; la capa oculta tiene 14 neuronas.

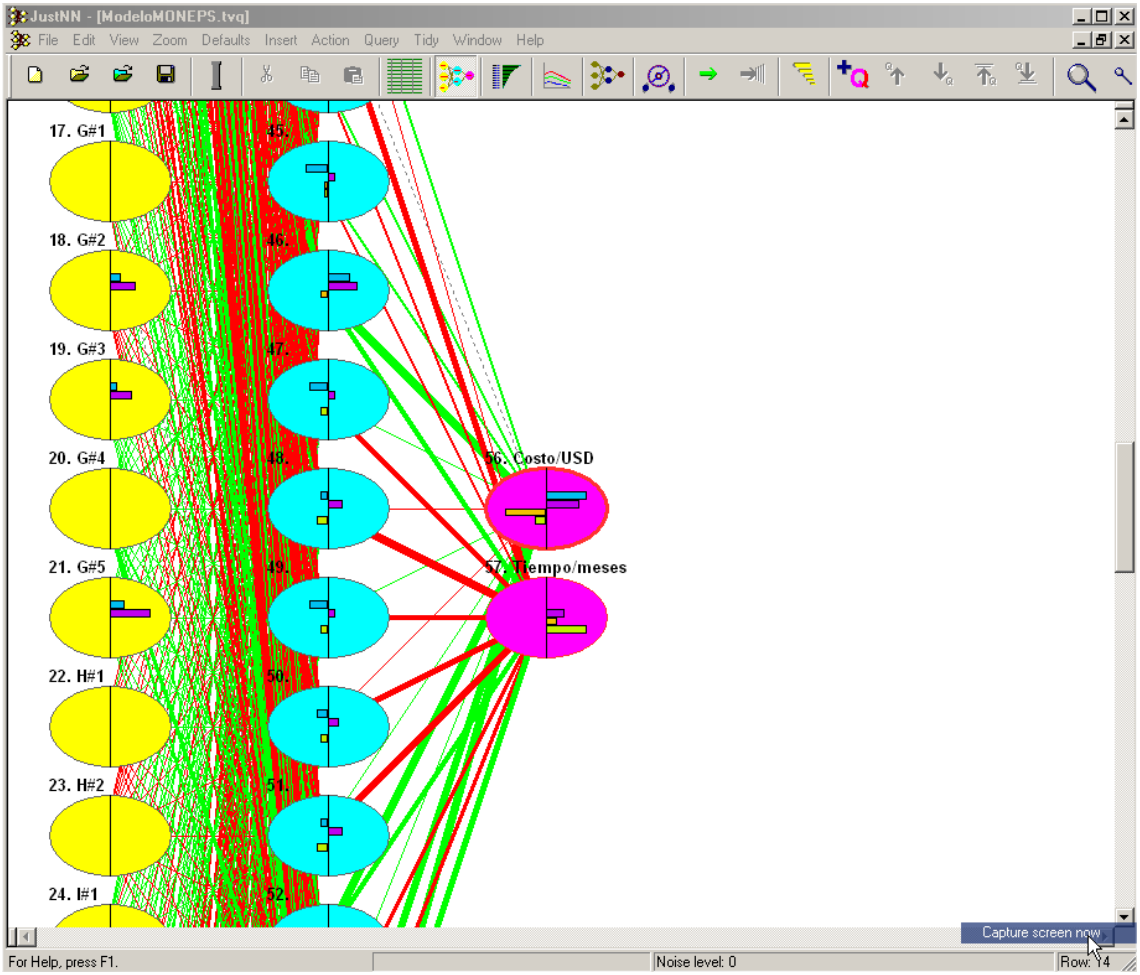


Figura 5.5 Vista parcial de las 3 capas que forman la RNA utilizada por MONEPS.

5.8 Resultados y evaluación del modelo

Después de realizado el entrenamiento de la RNA se pudo constatar que ésta tuvo un desempeño satisfactorio después de 45 ciclos de aprendizaje o épocas (ver figura 5.6). El error de la red, durante la fase de entrenamiento estuvo en el orden del 0.4 %, lo que denota un rendimiento muy bueno en relación al mínimo error requerido (target error=1%).

Por consiguiente, podemos observar que la red aprendió rápidamente a configurar patrones de comportamiento para tiempos y costos referidos a proyectos académicos de software, lo que se puede ver en la figura 5.7.

Figura 5.6 Resultados obtenidos durante la fase de entrenamiento de la RNA utilizada por MONEPS.

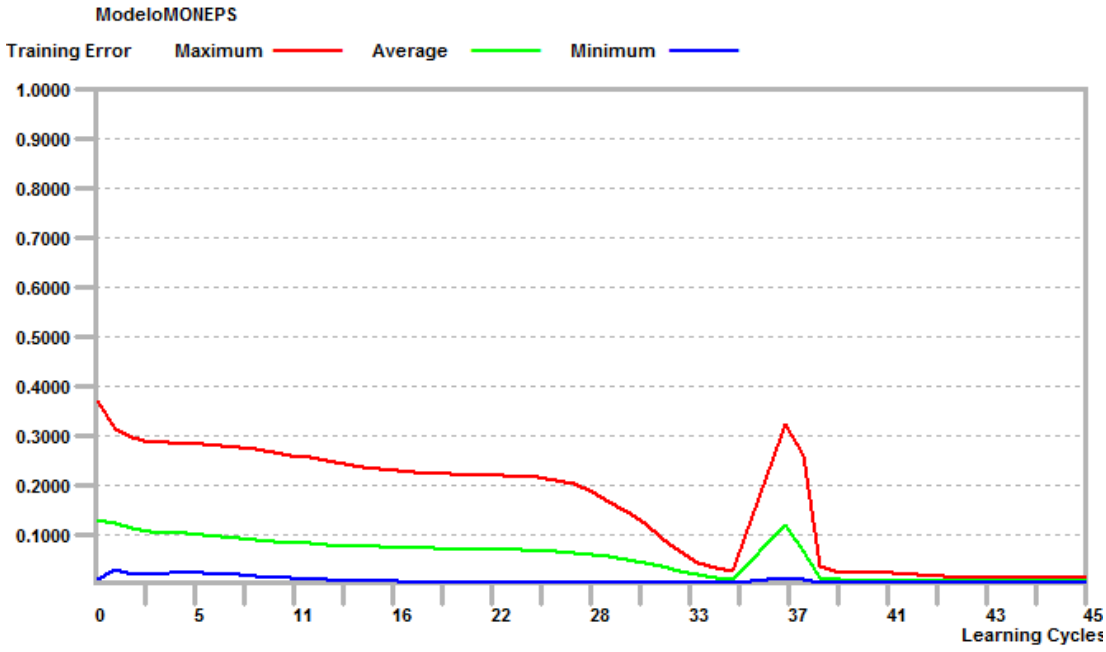


Figura 5.7 Variación del error máximo, mínimo y medio durante el entrenamiento de la red.

La herramienta JustNN permite determinar la importancia relativa de cada atributo de la RNA (ver figura 5.8 y tabla 5.5). Podemos observar que, por ejemplo, los atributos de la categoría *usabilidad* (categoría C) tienen un alto impacto en la determinación del esfuerzo para el desarrollo del software. En contraste, la disponibilidad de un *tester de software* (atributo M.6) tiene poca incidencia en dicho esfuerzo.

ModeloMONEPS 45 cycles. Target error 0.0100 Average training error 0.004358
The first 42 of 42 Inputs in descending order.

Column	Input Name	Importance	Relative Importance
5	B#2	6.9493	
7	C#1	6.6091	
30	K#1	6.0020	
9	C#3	5.8804	
35	L#3	5.1906	
3	A#4	5.0505	
8	C#2	4.6256	
0	A#1	4.6250	
1	A#2	4.4109	
6	B#3	4.3445	
10	D#1	4.2762	
33	L#1	4.1759	
20	G#4	4.1425	
32	K#3	4.1022	
13	E#3	3.9416	
36	M#1	3.5745	
29	J#2	3.5735	
25	I#2	3.5210	
24	I#1	3.4751	
26	I#3	3.3491	
23	H#2	3.3283	
15	F#1	3.1472	
11	E#1	3.0052	
12	E#2	2.9542	
2	A#3	2.8422	
18	G#2	2.8328	
14	E#4	2.7719	
34	L#2	2.5783	
16	F#2	2.3370	
39	M#4	2.0556	
4	B#1	1.9773	
21	G#5	1.9527	
27	I#4	1.9266	
19	G#3	1.9255	
40	M#5	1.8260	
31	K#2	1.7010	
22	H#1	1.5873	
17	G#1	1.2359	
37	M#2	1.2028	
28	J#1	1.1970	
38	M#3	1.1421	
41	M#6	1.1194	

Figura 5.8 Importancia relativa de cada atributo de la red.

Tabla 5.5
Nombre e importancia relativa de cada atributo de la red.

Código	Atributo	Importancia Relativa	Código	Atributo	Importancia Relativa
A#1	DISPONIBILIDAD DE DATOS	4.62	G#5	TIPO DE BDD	1.95
A#2	NIVELES DE SEGURIDAD	4.41	H#1	TIPO DE ARQUITECTURA	1.58
A#3	CONFIDENCIALIDAD DE LA INFORMACION	2.84	H#2	CAPAS DE ARQUITECTURA	3.32
A#4	ADMINISTRACION PERFILES USR	5.05	I#1	NUMERO DE REQUISITOS FUNCIONALES	3.47
B#1	PARAMETRIZACION	1.97	I#2	NUMERO DE CASOS DE USO ANÁLISIS	3.52
B#2	ADM GLOBAL DEL SISTEMA	6.94	I#3	NUMERO DE CASOS DE USO DE DISEÑO	3.34
B#3	CONCURRENCIA	4.34	I#4	COMPLEJIDAD DEL SISTEMA	1.92
C#1	FACILIDAD DE NAVEGACION	6.60	J#1	PARADIGMA USADO	1.19
C#2	TIEMPO DE RESPUESTA	4.62	J#2	NUMERO DE ELEMENTOS IDENTIFICADOS	3.57
C#3	ESTILOS DE INTERFACES	5.88	K#1	ESCALABILIDAD	6.00
D#1	TAZA DE ACTUALIZACIÓN DE INFORMACIÓN	4.27	K#2	FRECUENCIA DE ACTUALIZACIÓN POR CORRECCIÓN DE ERRORES	1.70
E#1	REQUERIMIENTOS DE HW	3.00	K#3	FRECUENCIA DE NUEVAS VERSIONES	4.10
E#2	ARQUITECTURA DE HW	2.95	L#1	ADAPTABILIDAD	4.17
E#3	SERVIDORES	3.94	L#2	FACILIDAD DE INSTALACION	2.57
E#4	ESTACIONES DE TRABAJO	2.77	L#3	INTERACCION CON OTROS SISTEMAS	5.19
F#1	TIPO DE LENGUAJE	3.14	M#1	EXPERIENCIA DEL EQUIPO DE DESARROLLO EN PROYECTOS TIPO	3.57
F#2	PLATAFORMA DESARROLLO	2.33	M#2	ROL: ARQUITECTO	1.20
G#1	TIPO DE LICENCIA	1.23	M#3	ROL: PROGRAMADOR	1.14
G#2	TAMAÑO DE BDD	2.83	M#4	ROL: ANALISTA	2.05
G#3	NORMALIZACION DE BDD	1.92	M#5	ROL: PLANIFICADOR	1.82
G#4	REFACTORIZACION DE BDD	4.14	M#6	ROL: TESTER	1.11

Sin embargo, el aspecto más importante en la evaluación de la RNA que conforma MONEPS está dado 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 los cuales

formaron el *conjunto de prueba*. **[102]** En la fase de evaluación, la RNA fue consultada sobre el tiempo y costo de desarrollo para estos tres proyectos académicos; previamente se ingresaron (por cada proyecto) 42 datos que definían la naturaleza de cada proyecto académico. Los resultados de esta fase se muestran en la Tabla 5.9.

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

Tabla 5.9 Tiempo y costo estimado para 2 casos de prueba 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

5.9 Contrastación de resultados COCOMO 81-COCOMO II vs. MONEPS

5.9.1 COCOMO 81

Para poder realizar la comparación se consideraron dos proyectos académicos, a los que se aplicó el cálculo del Modelo de Construcción de Costos (COCOMO 81), que considera los siguientes pasos:

1. Determinar el KLOC (numero de líneas de código) del proyectos.
2. Distinguir el modelo y el modo al que pertenece elproyecto, considerando el tamaño del mismo. Hay tres modelos (Básico,

Intermedio y Detallado); y tenemos tres modos (Orgánico, Semilibre y Rígido). Para más detalle ver el Anexo No. 4 (Aplicación de COCOMO 81 y COCOMO II en proyectos académicos).

3. Aplicar las fórmulas de esfuerzo, tiempo de desarrollo y productividad para el proyecto.

Una vez aplicados los pasos anteriores, se obtuvieron los resultados que se resumen en la Tabla 5.10.

Tabla 5.10
Resultados obtenidos para dos proyectos académicos en base a COCOMO 81

Caso	Nombre del proyecto	Esfuerzo (pers-mes)	Tiempo de desarrollo (meses)	Productividad (personas)
1	CODESOFT	14.20	6.85	2.07
2	FACTURACIÓN	19.64	7.75	2.53

La Tabla 5.11 resume los cálculos realizados por COCOMO 81 y MONEPS en la variable tiempo de duración para dos proyectos.

Tabla 5.11

Datos obtenidos con COCOMO 81 y MONEPS

Caso	Nombre del proyecto	Tiempo real de duración (meses)	Tiemp estimado por COCOMO 81 (meses)	Tiempo estimado por MONEPS (meses)
1	CODESOFT	3.0	6.85	3.65
2	FACTURACIÓN	5.0	7.75	3.97

Como se puede apreciar, al menos para proyectos de esta naturaleza (académicos), Moneps muestra una mejor aproximación en la predicción del tiempo. Recuérdese que MONEPS también puede estimar costos, cálculo que no es contemplado en el modelo COCOMO 81.

5.9.2 COCOMO II

En lo que respecta a COCOMO II, este modelo permite realizar estimaciones en función del tamaño del software y de un conjunto de factores de costo y de escala. Los factores de costo describen aspectos relacionados con la naturaleza del producto, hardware utilizado, personal involucrado, y características propias del proyecto. El conjunto de factores de escala explica las economías y deseconomías de escala producidas a medida que un proyecto de software incrementa su tamaño. (Trendowicz, Adam, Jeffery, Ross, 2014)

COCOMO II posee tres modelos denominados Composición de Aplicación, Diseño Temprano y Post-Arquitectura. Cada uno de ellos orientados a sectores específicos del mercado de desarrollo de software y a las distintas etapas del desarrollo de software.

Utilizando la herramienta informática USC-COCOMO II se procedió a realizar los cálculos correspondientes a los proyectos académicos antes referidos. El resumen de resultados se muestra en la Tabla 5.12:

Tabla 5.12

Resultados obtenidos para dos proyectos académicos en base a COCOMO II

Caso	Nombre del proyecto	Esfuerzo (pers-mes)	Tiempo de desarrollo (meses)	No. personas
1	CODESOFT	19.10	12.20	1.6
2	FACTURACIÓN	23.50	10.20	1.9

Para más detalle ver el Anexo No. 4 (Aplicación de COCOMO 81 y COCOMO II en proyectos académicos).

La Tabla 5.13 resume los cálculos realizados por COCOMO II y MONEPS en la variable tiempo de duración, para dos proyectos.

Tabla 5.13

Datos obtenidos con COCOMO II y MONEPS

Caso	Nombre del proyecto	Tiempo real de duración (meses)	Tiemp estimado por COCOMO II (meses)	Tiempo estimado por MONEPS

				(meses)
1	CODESOFT	3.0	12.20	3.65
2	FACTURACIÓN	5.0	10.20	3.97

También se puede apreciar que, al menos para proyectos de esta naturaleza (académicos), MONEPS sigue manteniendo una mejor aproximación en la predicción de tiempo. En la Tabla 5.14 se muestra un resumen de los resultados obtenidos con COCOMO II y MONEPS, para las predicciones de tiempo y costo, mientras que la Tabla 5.15 presenta el cálculo del error relativo correspondiente.

Tabla 5.14

Tiempos y costos estimados con COCOMO-II y MONEPS

Caso	Nombre del proyecto	Tiempo (meses)			Costo (USD)		
		Real	Estimado por Cocomo-II	Estimado por Moneps	Referen.	Estimado por Cocomo-II	Estimado por Moneps
1	CODESOFT	3.0	12.20	3.65	7558.80	10371.01	8139.45
2	FACTURACIÓN	5.0	10.20	3.97	8810.00	12376.34	7273.05

Tabla 5.15

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%

Cabe mencionar que, se procedió a realizar una demostración correspondiente a la utilidad del modelo neuronal, donde participaron

algunos profesionales expertos en el área académica y en desarrollo de software, a fin de obtener sus criterios y recomendaciones para poder retroalimentar el modelo propuesto. Las opiniones y recomendaciones se muestran en las encuestas contenidas en el Anexo 2 del presente documento.

CAPÍTULO VI

CONCLUSIONES Y RECOMENDACIONES

6.1 CONCLUSIONES

- Se ha logrado construir una red neuronal en backpropagation cuyas entradas se fundamentan en los atributos del estándar de la norma ISO 25000, para estimar el costo y tiempo de desarrollo en productos software.

- La arquitectura de la red neuronal fue identificada en base a las características contemporáneas del software y, considerando el desempeño de las topologías neuronales disponibles.

- Se realizaron algunas pruebas con el modelo neuronal, utilizando proyectos desarrolados por estudiantes de los últimos niveles de la carrera de Ingeniería en Sistemas e Informática, en la Universidad de las Fuerzas Armadas-ESPE; esto ha permitido comprobar la consistencia y utilidad del modelo propuesto, pues, los tiempos y costos estimados por la red neuronal estuvieron muy cercanos a los valores reales de tales proyectos académicos.

- El modelo neuronal fue mostrado a algunos profesionales expertos en el área académica y en desarrollo de software, a fin de obtener sus criterios y recomendaciones, mismos que han permitido retroalimentar el modelo propuesto.

- Obtener información de proyectos de desarrollo de software de la empresa pública o privada no es una tarea sencilla, dada la reserva de información o indisponibilidad de ésta.

- No existe en el estado del arte, referido a los modelos de estimación de esfuerzo en software, uno que abarque todos los aspectos funcionales y menos aún los no funcionales. MONEPS ha logrado la convergencia de aspectos funcionales y no funcionales en la estimación de tiempo y costo para productos software.

- La identificación de métricas más especializadas para los aspectos no funcionales de un sistema, es un verdadero desafío en la Ingeniería del Software.

- Al menos, para proyectos académicos, y una vez que se han realizado estimaciones con los modelos COCOMO 81 y COCOMO II, MONEPS ha mostrado estimaciones de tiempo y costo más cercanos a los reales.

- MONEPS es de fácil uso y escalable; permitiendo 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 red neuronal con datos de nuevas aplicaciones.

6.2 RECOMENDACIONES

- Profundizar en el análisis de la familia de estándares ISO-2500 u otros estándares orientados a verificar la calidad del software para un mejor entendimiento de los aspectos funcionales y no funcionales más críticos en el desarrollo de software, lo que a su vez permitirá mejorar el desempeño de MONEPS.

- Como no existe un modelo universal de estimación para el esfuerzo en proyectos de software, es importante investigar las diversas opciones de modelos y técnicas, para adecuarlos a nuestra propia realidad.

- Para una mejor adaptabilidad de MONEPS, cada empresa/usuario puede configurar los atributos de entrada; de esta manera, el modelo neuronal podrá ajustarse a las restricciones de configuración impuestas.

- Otra mejora posterior de MONEPS se basa en la adición o eliminación de atributos, así como en la depuración de métricas para los aspectos no funcionales.

- Para tener estimaciones de costo y tiempo, acorde a los requerimientos locales de software, es recomendable recopilar más casos de productos desarrollados en la empresa pública y/o privada, que permitan alimentar al modelo neuronal.

- Las empresas desarrolladoras de software deberían iniciar los proyectos realizando algún tipo de estimación temprana del esfuerzo requerido, lo que permitirá reducir riesgos de incumplimiento o pérdidas importantes de capitales.

- Se debe incentivar el estudio de métodos y técnicas alternativas para la estimación del esfuerzo en proyectos de software; pues, como se ha visto, no hay un modelo universal de estimación, y los disponibles tienen falencias contextuales que no han sido corregidas.

- Se debería realizar un estudio formal acerca de la verdadera utilidad de los modelos tradicionales de estimación acorde a las nuevas tendencias tecnológicas y requerimientos de software.

BIBLIOGRAFÍA

1. Diaz Villegas, J. E., & Robiolo, G. 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 (2014).
2. Pytel, P., Uhalde, C., Ramón, H. D., Castello, H., Tomasello, M., Pollo Cattaneo, M. F., ... & García Martínez, R. Ingeniería de requisitos basada en técnicas de ingeniería del conocimiento. In XIII Workshop de Investigadores en Ciencias de la Computación. (2011).
3. IREB Ingeniería de requermientos Profesional. Formación para el "Profesional Certificado en Ingeniería de Requerimientos –Nivel Básico IREB @, de acuerdo al programa estudios vesión 2.1 (O 1.03.20.11) IREB@ Versión ESP1.0 Capítulo IV Documentación de requsitos disponible en: URL:www.ireb.org. (2014).
4. Portillo, J. P. S. La Especificació n de Requisitos con Casos de Uso: Buenas y Malas Prácticas. Pontificia Universidad Católica del Perú, 1.
5. Ferré, X. Incrementos de Usabilidad al Proceso de Desarrollo Software. In JISBD (pp. 293-302). (2003).
6. Agut, R. M. Especificación de Requisitos Software según el estándar de IEEE 830. (2012).
7. Boehm, B. W., Madachy, R., & Steece, B. Software cost estimation with Cocomo II with Cdrom. Prentice Hall PTR. (2000).
8. Trendowicz, Adam, Jeffery, Ross, Software Project Effort Estimation, Foundations and Best Practice Guidelines for Success,Constructive Cost Model –COCOMO pags: 277-293. (2014).
9. Peralta, M. Estimación del esfuerzo basada en casos de uso. Reportes Técnicos en Ingeniería de Software. Buenos Aires-Argentina, 6(1), 1-16. (2004).
10. Boehm, B. W. Software engineering economics (Vol. 197). Englewood Cliffs (NJ): Prentice-hall. (1981).
11. Rodríguez, D., Pytel, P., Tomasello, M., Pollo Cattaneo, M. F., Britos, P. V., & García Martínez, R. Estudio del Modelo Paramétrico DMCoMo de Estimación de Proyectos de Explotación de Información. In XVII Congreso Argentino de Ciencias de la Computación. (2011).
12. Boehm, B. W., & Valerdi, R. Achievements and challenges in cocomo-based software resource estimation. Software, IEEE, 25(5), 74-83. (2008).
13. Muñoz, C. C., Velthuis, M. G. P., & de la Rubia, M. Á. M. Calidad del producto y proceso software. Editorial Ra-Ma. (2010).
14. Madachy, R., & Boehm, B. 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. (2008).
15. Gómez, A., López, M. D. C., Migani, S., & Otazú, A. UN MODELO DE ESTIMACION DE PROYECTOS DE SOFTWARE.
16. Kemerer, C. F. An empirical validation of software cost estimation models. Communications of the ACM, 30(5), 416-429. (1987).

17. Gómez, A., López, M. D. C., Migani, S., & Otazú, A. UN MODELO DE ESTIMACION DE PROYECTOS DE SOFTWARE. (2015).
18. Visconti, M. Ingeniería de software avanzada. Universidad Tecnica Federico Santa Maria. Departamento de Informatica. Valparaíso Chile. Documento digital. (2010).
19. Ruíz Constanten, Y., & Cordero Morales, D. Estimación en proyectos de software integrando los métodos de Boehm y Humphrey. Revista Cubana de Ciencias Informáticas, 7(3), 23-36. (2013).
20. Rodríguez, D., Pytel, P., Tomasello, M., Pollo Cattaneo, M. F., Britos, P. V., & García Martínez, R. Estudio del Modelo Paramétrico DMCoMo de Estimación de Proyectos de Explotación de Información. In XVII Congreso Argentino de Ciencias de la Computación. (2011).
21. Rubio, S. E. D. Puntos por Función. Una métrica estándar para establecer el tamaño del software. Boletín de Política Informática. (2003).
22. Buritica Gonzáles, S. M., Hernández Molina, O. J., & Hoyos Aristizábal, J. A. Puntos de función ajustados para sistemas de información fundamentados en ingeniería de software y telecomunicaciones. (2013).
23. Mk, I. I. MK II FUNCTION POINT ANALYSIS. (2002).
24. Gallego, J. J. C., & Buglione, L. El proceso de Bolonia y la profesión informática. Novática: Revista de la Asociación de Técnicos de Informática, (182), 5-6. (2006).
25. Pressman, R. Ingeniería de Software. McGraw-Hill Interamericana de España. (2010).
26. Guerra, J., Luza, C., & Coral, M. Una aplicación práctica del método de análisis de puntos de función. Revista de investigación de Sistemas e Informática, 2(3), 14-20. (2005).
27. Bertolami, M., & Oliveros, A. SFP: Un Procedimento de Estimación de Puntos de Función de Escenarios. In WER (pp. 101-108). (2006).
28. Gramajo, E., García-Martínez, R., Rossi, B., Claverie, E., & Britos, P. Combinación de Alternativas para la Estimación de Proyectos Software.CAPIS–Centro de Actualización Permanente en Ingeniería del Software. (1998).
29. Izaurieta, F., & Saavedra, C. Redes neuronales artificiales. Departamento de Física, Universidad de Concepción Chile. (2000).
30. Galipienso, M. I. A., Quevedo, M. A. C., Pardo, O. C., Ruiz, F. E., & Ortega, M. A. L. Inteligencia artificial: modelos, técnicas y áreas de aplicación. Editorial Paraninfo. (2003).
31. Allende, H., Moraga, C., & Salas, R. Artificial neural networks in time series forecasting: A comparative analysis. Kybernetika, 38(6), 685-707. (2002).
32. Fausett, L. Fundamentals of neural networks: architectures, algorithms, and applications. Prentice-Hall, Inc. (1994).
33. Russell, S. J., & Norvig, P. Artificial Intelligence: A Modern Approach. Third Edition. Prentice Hall. (2009).
34. Aguilera, S., & Constenla, G. Aplicación de Redes Neurales en el reconocimiento óptico de caracteres. In XII Workshop de Investigadores en Ciencias de la Computación. (2010).
35. Jain, A. K., Mao, J., & Mohiuddin, K. M. Artificial neural networks: A tutorial. Computer, (3), 31-44. (1996).

36. Bishop, C. M. Pattern recognition and machine learning. Springer. (2006).
37. Abraham, A. Artificial neural networks. Handbook of measuring system design. (2005).
38. Klerfors, D. Artificial neural networks. St. Louis University, St. Louis, Mo. (1998).
39. Hilera González, J. R., & Martínez Hernando, V. J. Redes neuronales artificiales: fundamentos modelos y aplicaciones. Madrid: Editorial Alfaomega Ra-Ma. (2000).
40. Díez, R. P., Gómez, A. G., & de Abajo Martínez, N. Introducción a la inteligencia artificial: sistemas expertos, redes neuronales artificiales y computación evolutiva. Universidad de Oviedo. (2001).
41. Sierra, A. P. Redes de neuronas artificiales y algoritmos genéticos. Servicio de Publicaciones. (1996).
42. Quesada, F. J. G., Graciani, M. A. F., Bonal, M. T. L., & Díaz-Mata, M. A. Aprendizaje con redes neuronales artificiales. Ensayos: Revista de la Facultad de Educación de Albacete, (9), 169-180. (1994).
43. Ortiz, J. A. P. Modelos Predcitivos Basados en Redes Neuronales Recurrentes de tiempo discreto. Universidad de Alicante. Departamento de lenguaje y sistemas informáticos. (2002).
44. Romay, M. M. G., D'Anjou, A. D. A., Irigoyen, F. X. A., Alonso, J. A. L., Múgica, P. L., Mendizábal, Y. Y., ... & González, A. I. Experimentos de aprendizaje con Máquinas de Boltzmann de alto orden. Informática y automática: revista de la Asociación Española de Informática y Automática, 29(4), 42-57. (1996).
45. Junyent, M. V., & Gras, J. A. De la máquina de Turing a la máquina de Boltzmann: dinámica interactiva y fenómenos globales en redes conexionistas. Anuario de psicología/The UB Journal of psychology, (56), 27-48. (1993).
46. Herrera, F. Introducción a los algoritmos metaheurísticos. Ciencias de la Computación e IA. (2006).
47. Ingber, L. Simulated annealing: Practice versus theory. Mathematical and computer modelling, 18(11), 29-57. (1993).
48. Carvallo, J. P., Franch, X., & Quer, C. Calidad de componentes software. Cap. libro en Calidad del Producto y Proceso Software, 287-316.
49. Espinel Mena, G. P., & Montaluisa Yugla, F. J. Propuesta de implementación de un modelo de calidad para la selección de software base para servidores (Doctoral dissertation, LATACUNGA/ESPE). (2012).
50. Luna Tellez Linda, UAGRM. Estándares de Calidad McCall; ISO/IEC 25000; ISO 9126. (2012).
51. International Organization for Standardization & International Electrotechnical Commission. ISO/IEC 25000, Software product Quality Requirements and Evaluation (SQuaRE). (2005).
52. J. P. Carvallo. Systematic Construction of Quality Models for COTS-Based Systems. PhD Thesis, LSI-UPC, (2005).
53. De Mendoza, I. P. H. Redes Neuronales Artificiales. Inteligencia Artificial. (2009).
54. Warwick, K. Artificial intelligence: the basics. Routledge. (2013).

55. López, R. F., & Fernandez, J. M. F. Las redes neuronales artificiales. Netbiblo. (2008).
56. Rumelhart, D. E., & McClelland, J. L. On learning the past tenses of English verbs (No. ICS-8507). CALIFORNIA UNIV SAN DIEGO LA JOLLA INST FOR COGNITIVE SCIENCE. (1985).
57. MUNRO, P., & ZIPSER, D. Image compression by back propagation: an example of extensional programming'. Models of cognition: A review of cognitive science. (1989).
58. Perdomo, J. G. Un nuevo enfoque para la resolución de problemas: redes neuronales. Revista EAN, (24), 35-40. (2015).
59. Vélez, M. C., & Montoya, J. A. Metaheurísticos: una alternativa para la solución de problemas combinatorios en administración de operaciones. Revista EIA, (8), 99-115. (2007).
60. Isasi Viñuela, P., & Galván León, I. M. Redes de Neuronas Artificiales.Un Enfoque Práctico, Editorial Pearson Educación SA Madrid España. (2004).
61. Pino Díez, R., Fuente García, D. D. L., Parreño Fernández, J., & Priore Moreno, P. Aplicación de redes neuronales artificiales a la previsión de series temporales no estacionarias o no invertibles. (2002).
62. Bazan, N. NUEVAS HERRAMIENTAS DE INVESTIGACION: LAS REDES NEURONALES. Revista electrónica de Ciencias Aplicadas al Deporte, 5(17). (2012).
63. Serrano Acevedo, M. E., Rico Arias, J. Á., & Rico Arias, J. Á. Algunas aplicaciones de las redes neuronales. Opciones, 5(10). (2013).
64. Salazar Castillo, J. E. Diseño e implementación de un sistema de entrenamiento en redes neuronales utilizando el Software Neurosystems SIEMENS. (2009).
65. http://lema.rae.es/drae/srv/search?id=X6szvNxl2DXX2yXDrLQn
66. López, Ernest Teniente; Costa, Dolors Costal; Samsó, Ma Ribera Sancho. *Especificación de sistemas software en UML*. Universitat Politecnica de Catalunya. Iniciativa Digital Politecnica. (2004.)
67. http://catarina.udlpa.mx/u_dl_A/tales/documentos/lisgonzales_d_H/capítulo2-pdf
68. http://tecnomaestros.awardspace.com/métricas_software.php
69. Stutzke, R. D. Estimating software-intensive systems: projects, products, and processes. Pearson Education. (2005).
70. Boehm, B. W., Madachy, R., & Steece, B. (2002)Software cost estimation with Cocomo II.
71. Software Engineering Lab. COCOMO -81- SEL-UC3M, Pags(9,10,11,12,13). (2007).
72. Software Engineering Lab. Albretch: Tema 2, Pags(9,10). (2007).
73. Software Engineering Lab. Albretch: Tema 1, Pags(3,4,5). (2007).
74. Software Engineering Lab. Albretch: Tema 2, Pags(3,4,5,6,7,8). (2007).
75. http://www.facmed.unam.mx/Libro-NeuroFisio/Personas/Hebb/Hebb.html
76. http://www.e-torredebabel.com/Psicologia/Conexionismo/Conexionismo-MecanismosdeProcesamiento.htm
77. http://cmapspublic2.ihmc.us/rid%3D1LVQ1DG19-ZD95ZV-5X3/Dendritas%25201.png

78. http://bibliotecadigital.ilce.edu.mx/sites/ciencia/volumen3/ciencia3/130/html/sec_10.html
79. https://sites.google.com/site/informacisinapsisquimica/sinapsis-inhibitoria-y-excitatoria
80. http://nosololinux.com/2006/02/13/el-hombre-contra-la-maquina/
81. http://www.academia.edu/7245695/Redes_Neuronales_Artificiales
82. http://www.rmib.somib.org.mx/Htmls/Vol35No3/3/3.html
83. http://thales.cica.es/rd/Recursos/rd98/TecInfo/07/capitulo2.html
84. https://advancedtech.wordpress.com/category/ann/
85. http://marco-redes-neuronales.blogspot.com/2012/08/perceptron-simple.html
86. Figura 3.8 Red en Backpropagation de cinco capas
87. http://controlinteligente.netai.net/index_archivos/page0007.htm
88. http://www.lcc.uma.es/~munozp/documentos/modelos_computacionales/temas/Tema8MC-05.pdf
89. https://iaevolutiva.wordpress.com/category/redes-neuronales/
90. http://www.profesorenlinea.cl/fisica/Calor_Equilibrio_termico.html
91. Calidad del producto y proceso software. Muñoz, C. C., Velthuis, M. G. P., & de la Rubia, M. Á. M.
92. CARVALLO Juan Pablo, SYSTEMATIC CONSTRUCCTION OF QUALITY MODELS FOR COST-BASED SYSTEMS,Capítulo 10, 10.2.1 Tipos de Modelos de Calidad.
93. Individual Quality Model Construction: conjunto de guías y técnicas para identificar factores de calidad que deben ser incluidos en un modelo de calidad.
94. CARVALLO Juan Pablo, SYSTEMATIC CONSTRUCCTION OF QUALITY MODELS FOR COST-BASED SYSTEMS,Capítulo 10, 10.2.3 Propiedades de los Modelos de Calidad.
95. http://redesneuronales-mini.blogspot.com/2010/02/concepto-de-red-neuronal.html
96. http://technolifeandmore.blogspot.com/2012/11/reporte-final-redes-neuronales.html
97. http://cadasistemas.e.telefonica.net/nr.htm
98. El éxito al responder las muestras de entrenamiento sigue incrementándose mientras que su actuación con muestras nuevas va empeorando.
99. Figura 5.2 RNA utilizada en MONEPS
100. Las caracteristicas completas de JustNN están disponibles en: http://www.justnn.com/
101. http://proton.ucting.udg.mx/posgrado/cursos/idc/neuronales2/ReglaB.htm
102. Proyectos académicos de software que no fueron considerados en la fase de entrenamiento de la RNA.
103.

ANEXOS

ANEXO Nº1
MANUAL DE JUSTNN

ANEXO Nº 2
ENCUESTAS DE MONEPS

ANEXO Nº 3

CODIFICACIÓN PARA PROYECTOS ACADÉMICOS DE SOFTWARE

ANEXO Nº 4

APLICACIÓN DE COCOMO 81 Y COCOMO II EN PROYECTOS ACADÉMICOS