A Novel Approach to Design Neuro-Fuzzy Expert System for Software Estimation

B V A N S S Prabhakar Rao[1] & P Seetha Ramaih[2]
[1]*Research Scholar, Department of Computer Science and Engineering, JNTU Kakinada, Kakinada, Andhra Pradesh, India*
[2]*Professor Emeritus, Dept. of CS & SE, College of Engineering (A), Andhra University Visakhapatnam, Andhra Pradesh, India*

Abstract

The software industry estimators does not estimate required projects effort, cost and schedule well. Developing an effective estimation model is one of the most challenging and important activities in software development. Existing project estimating models are good but not satisfying the needs of the customer organizations. Software developers and clients are extremely confident about the project and team's capabilities. There is modest that software cost estimation models can reimburse for when software projects lack the necessary information and knowledge at the time when cost and schedule are estimated. Furthermore, most estimation models need careful understanding of the model parameters as well as a certain level of domain expertise in order to use them efficiently. Without the proper understanding, teams may potentially end up exaggerating the capability of the team's personnel, or understating the complexities of the project. These misrepresentations lead to inaccurate and non-realistic estimations. That result in poor project planning with schedule overruns. This paper suggests a mechanism to design a model to cater the needs of the project managers to make accurate estimations. The proposed system will help to manage resources, control and plan a project, and deliver a project on time, on schedule and on budget.

1. Introduction

A Software Project is dynamic. The code changes, the design changes, and the requirements change. What's more, changes in the requirements lead to more changes in the design, and changes in the design lead to even more changes in the code and test cases. But the estimation process is static. If the software cost is underestimated, inefficiencies will be brought to the project and the actual cost will be surely increased. Overestimated software cost will lose the bid, waste the time, money, staff and other resources, and lead to the financial loss, even economic failure of the organization. Therefore, the ability to accurately estimate the software cost needed to complete the project on time is crucial for software organizations [1-5].

How many were delivered on time and on budget? How many estimates were accurate? IT projects are notorious for over-running, and here are several reasons why it occurs… [Craig Buckler, 2010].

- *The project is poorly scoped*
- *Development time is estimated by non-programmers*
- *Developer estimates are too optimistic*
- *The project is not adequately dissected*
- *Estimated time is used*
- *Sometimes more developers may involve for quick estimation*
- *The project scope changes*
- *Estimates are fixed*
- *Testing time is forgotten*
- *Estimates are taken too literally*

and many more ……

"Measure twice, cut once" is highly relevant to the construction part of software development. The worst software projects end up doing construction two or three times or more. Doing the most expensive part of the project twice is as bad an idea in software as it is in any other line of work. Software projects can be measured in numerous ways. For any project attribute, it's possible to measure that attribute in a way that's superior to not measuring it at all. If data is to be used in a scientific experiment, it must be quantified. Be aware of measurement side effects. Measurement has a motivational effect. People pay attention to whatever is measured, assuming that it's used to evaluate them. Choose what you measure carefully. People tend to focus on work that's measured and to ignore work that isn't. To argue against measurement is to argue that it's better not to know what's really happening on your project. When you measure an aspect of a project, you

know something about it that you didn't know before. You can see whether the aspect gets bigger or smaller or stays the same. The measurement gives you a window into at least that aspect of your project. The window might be small and cloudy until you refine your measurements, but it will be better than no window at all. To argue against all measurements because some are inconclusive is to argue against windows because some happen to be cloudy [8].

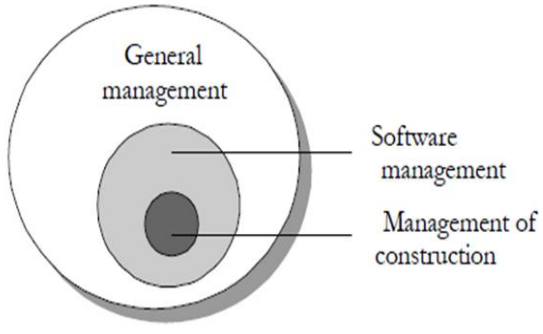

Figure 1. Software management

2. Estimation and Measurement

Until a system is deployed at the customer site and developed product in use, no one can ever be quite sure whether it meets the customer's needs or not. To solve the above problem understand the software application domain that is being developed and communicate effectively with clients and users. Without proper understanding of the customer requirements and domain knowledge estimation is not achievable. Software development is heavily labour-intensive; however, skills of team members can vary dramatically and probably are the biggest single factor affecting success of a project [12, 19, 20].

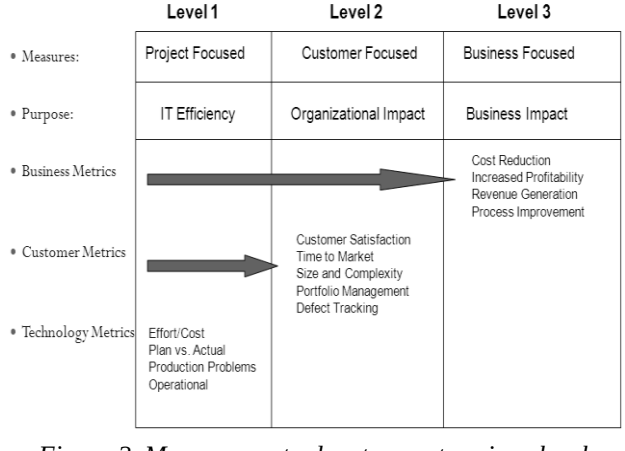

Figure 2. Measurement advantages at various levels

Estimation and measurement are two faces of the same attribute of a software application: size. This explanation can be extended to other similar attributes of software projects that include effort, schedule, and quality parameters. Estimation is certainly not done at the end of the project. During the contract process there is a need to estimate the size, effort, and cost of a software product that is yet to be developed [10,11,13].

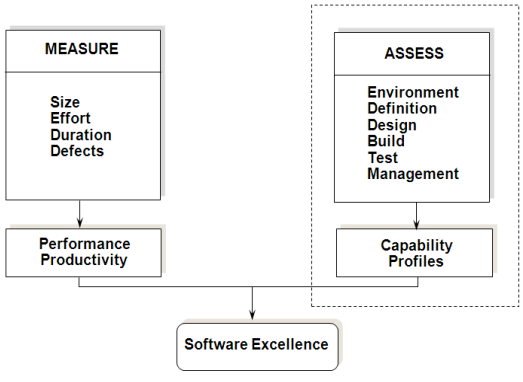

Figure 3. Qualitative Assessment of the product

With the exception of the contract phase and the time preceding the first milestone, measurement activity takes place in all other situations. At the completion of every milestone, both measurement of the completed activities and estimation of the balance milestones based on the analysis of the data collated, are done. This helps in tracking as well as making corrections to a project schedule that might be going off track and helps the project deliver on time, and within budget [9].

3. Hybrid Intelligent System

A hybrid intelligent system is one that combines at least two intelligent technologies. Suppose, combining a neural network with a fuzzy system results in a hybrid neuro-fuzzy system. The combination of probabilistic reasoning, fuzzy logic, neural networks and evolutionary computation forms the core of soft computing, an emerging approach to building hybrid intelligent systems capable of reasoning and learning in an uncertain and imprecise environment [7, 16-18].

3.1 Initial weights for training

Assign initial weights for size, defect tracking, productivity, overall quality and maintainability by obtaining the values from similar previous projects. Of course most of the software projects are unique. At that time it is better to depend on the expert suggestion.

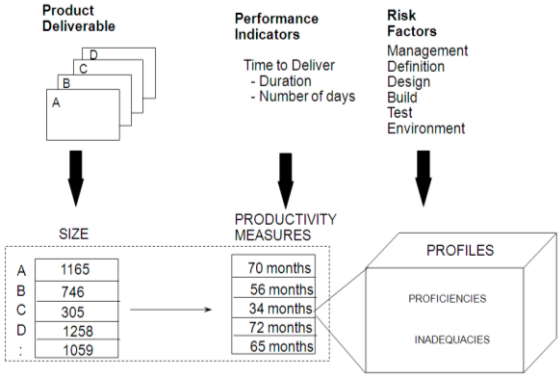

Figure 4. Developing a baseline of data

Size

S11: Total lines of code written
S12: Total comment lines
S13: Total number of classes or routines
S14: Total data declarations
S15: Total blank lines

Defect Tracking

D11: Severity of each defect
D12: Location of each defect (class or routine)
D13: Origin of each defect (requirements, design, construction, test)
D14: Way in which each defect is corrected
D15: Person responsible for each defect
D16: Number of lines affected by each defect correction
D17: Work hours spent correcting each defect
D18: Average time required to find a defect
D19: Average time required to fix a defect
D20: Number of attempts made to correct each defect
D21: Number of new errors resulting from defect correction

Productivity

P11: Work-hours spent on the project
P12: Work-hours spent on each class or routine
P13: Number of times each class or routine changed
P14: Dollars spent on project
P15: Dollars spent per line of code
P16: Dollars spent per defect

Overall Quality

Q11: Total number of defects
Q12: Number of defects in each class or routine
Q13: Average defects per thousand lines of code
Q14: Mean time between failures
Q15: Compiler-detected errors

Maintainability

M11: Number of public routines on each class
M12: Number of parameters passed to each routine
M13: Number of private routines and/or variables on each class
M14: Number of local variables used by each routine
M15: Number of routines called by each class or routine
M16: Number of decision points in each routine
M17: Control-flow complexity in each routine
M18: Lines of code in each class or routine
M19: Lines of comments in each class or routine
M20: Number of data declarations in each class or routine
M21: Number of blank lines in each class or routine
M22: Number of gotos in each class or routine
M23: Number of input or output statements in each class or routine

Religious Issues

R11: Programming language
R12: Indentation style
R13: Placing of braces
R14: Choice of IDE
R15: Commenting style
R16: Efficiency vs. readability trade-offs
R17: Choice of methodology—for example, scrum vs. extreme programming vs. evolutionary delivery
R18: Programming utilities
R19: Naming conventions
R20: Use of gotos
R21: Use of global variables
R22: Measurements, especially productivity measures such as lines of code per day

4. Neuro-Fuzzy Expert System

Fuzzy logic and neural networks are natural complementary tools in building intelligent systems. While neural networks are low-level computational structures that perform well when dealing with raw data, fuzzy logic deals with reasoning on a higher level, using linguistic information acquired from domain experts. However, fuzzy systems lack the ability to learn and cannot adjust themselves to a new environment. On the other hand, although neural networks can learn, they are opaque to the user. Integrated neuro-fuzzy expert systems can combine the parallel computation and learning abilities of neural networks with the human-like knowledge representation and explanation abilities of fuzzy systems. As a result, neural networks become more transparent, while fuzzy systems become capable of learning.

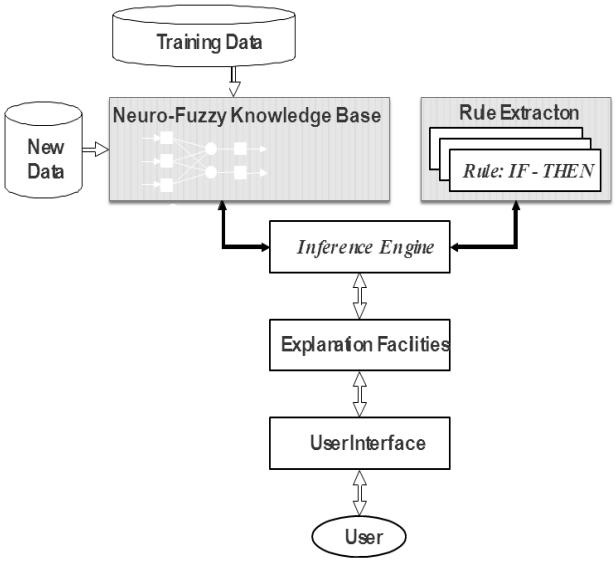

Figure 5. Neuro-Fuzzy Expert System

Expert systems rely on logical inferences and decision trees and focus on modelling human reasoning. Neural networks rely on parallel data processing and focus on modelling a human brain. Expert systems treat the brain as a black-box. Neural networks look at its structure and functions, particularly at its ability to learn. Knowledge in a rule-based expert system is represented by IF-THEN production rules. Knowledge in neural networks is stored as synaptic weights between neurons. In expert systems, knowledge can be divided into individual rules and the user can see and understand the piece of knowledge applied by the system. In neural networks, one cannot select a single synaptic weight as a discrete piece of knowledge. Here knowledge is embedded in the entire network; it cannot be broken into individual pieces, and any change of a synaptic weight may lead to unpredictable results.
A hybrid system that combines a neural network, fuzzy system and a rule-based expert system is called a neuro-fuzzy expert system. The heart of a neural expert system is the inference engine. It controls the information flow in the system and initiates inference over the neural knowledge base [6, 7, 14, 15].

5.1 Rule Extraction

Neurons in the network are connected by links, each of which has a numerical weight attached to it. The weights in a trained neural network determine the strength or importance of the associated neuron inputs. If we set each input of the input layer to either +1 (true), -1 (false), or 0 (unknown), we can give a semantic interpretation for the activation of any output neuron.

An inference can be made if the known net weighted input to a neuron is greater than the sum of the absolute values of the weights of the unknown inputs.

$$\sum_{i=1}^{n} x_i \, w_i \; > \sum_{j=1}^{n} \left| w_j \right|$$

where $i \in$ known, $j \notin$ known and n is the number of neuron inputs.

5.2 Knowledge Base

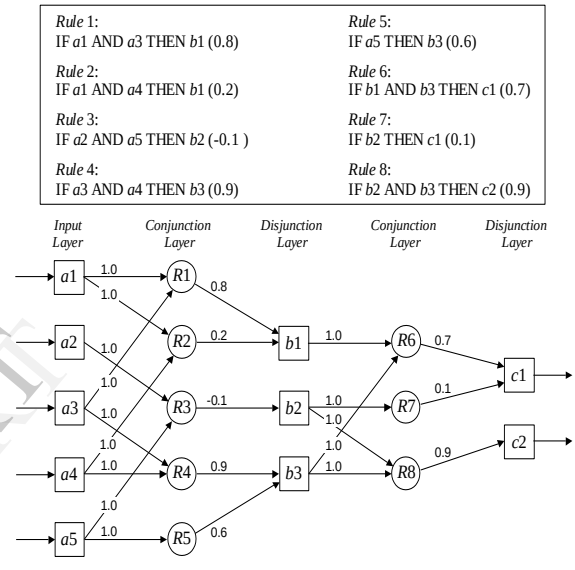

Figure 6. Multilayer Knowledge Base

6. Learning Process

A neuro-fuzzy system is essentially a multi-layer neural network, and thus it can apply standard learning algorithms developed for neural networks, including the back-propagation algorithm. When a training input-output example is presented to the system, the back-propagation algorithm computes the system output and compares it with the desired output of the training example. The error is propagated backwards through the network from the output layer to the input layer. The neuron activation functions are modified as the error is propagated. To determine the necessary modifications, the back-propagation algorithm differentiates the activation functions of the neurons.

Suppose that fuzzy IF-THEN rules incorporated into the system structure are supplied by a domain expert. *Prior* or existing knowledge can dramatically expedite the system training [14-18].

Besides, if the quality of training data is poor, the expert knowledge may be the only way to come to a solution at all. However, experts do occasionally make mistakes, and thus some rules used in a neuro-fuzzy system may be false or redundant. Therefore, a neuro-fuzzy system should also be capable of identifying bad rules.

Given input and output linguistic values, a neuro-fuzzy system can automatically generate a complete set of fuzzy IF-THEN rules.

After training we can eliminate all rules whose certainty factors are less than some sufficiently small number, say 0.1. As a result, we obtain the same set of four fuzzy IF-THEN rules that represents the XOR operation.

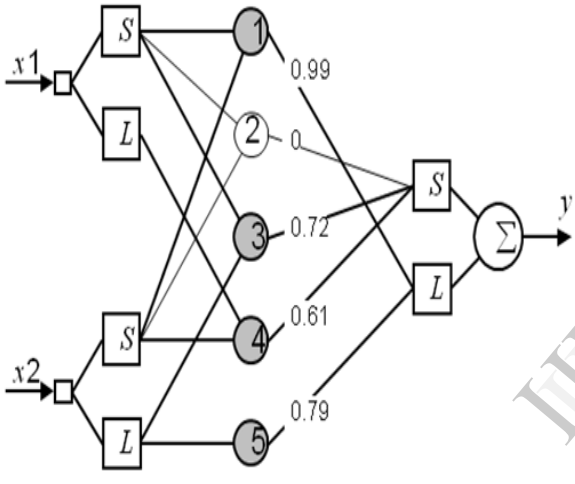

Figure 7. Five Rule System

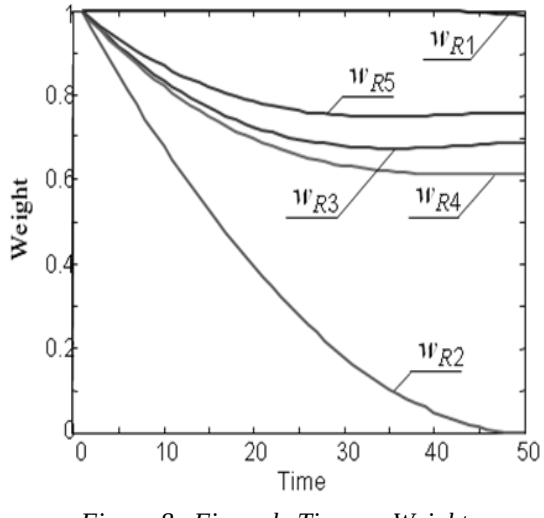

Figure 8. Five rule Time vs. Weight

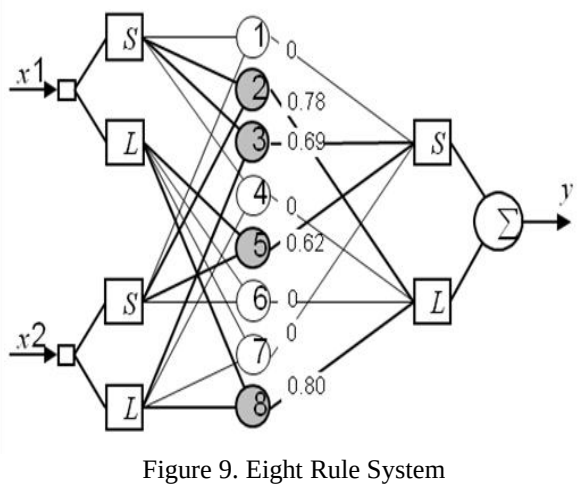

Figure 9. Eight Rule System

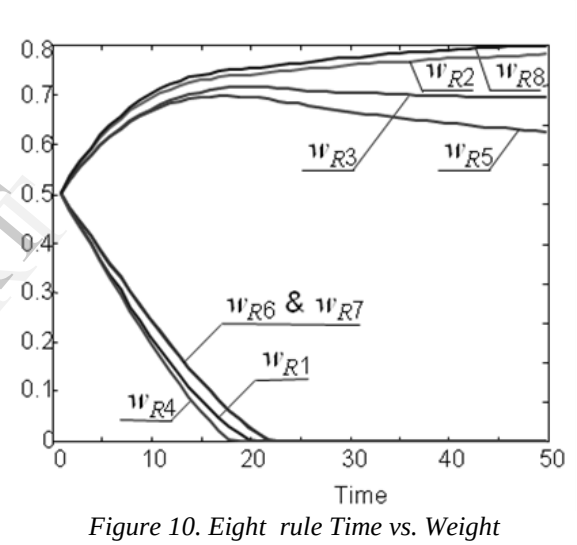

Figure 10. Eight rule Time vs. Weight

7. Conclusion

The combination of fuzzy logic and neural networks with expert system concept constitutes a powerful means for designing intelligent systems. The required domain knowledge for software application can be put into a neuro-fuzzy system by human experts in the form of linguistic variables and fuzzy rules. When a representative set of examples is available, a neuro-fuzzy system can automatically transform it into a robust set of fuzzy IF-THEN rules, and thereby reduce our dependence on expert knowledge when building intelligent systems. This estimation brings together today's most valuable tips, techniques, and best practices for accurately estimating software project efforts, costs, and schedules. Further work can be extended with all types of software products.

8. References

[1] Moataz A. Ahmed, Irfan Ahmad, and Jarallah S. AlGhamdi, "Probabilistic size proxy for software effort prediction: A framework", Elsevier, SciVerse ScienceDirect, Information and Software Technology, 55, 2013; Pp. 241-251.

[2] V. Khatibi Bardsiri, D.N.A. Jawawi, S.Z.M. Hashim, and E. Khatibi, "Increasing the accuracy of software development effort estimation using projects clustering", The Institution of Engineering and Technology 2012, Vol. 6, Iss. 6, Pp.461-473.

[3] Magne Jorgensen, Simula Research Laboratory, Practical Guidelines for Expert-Judgment-Based Software Effort Estimation, May/June 2005, IEEE SOFTWARE, Pp.57-63.

[4] M. Jorgensen, "A Review of Studies on Expert Estimation of Software Development Effort," J. Systems and Software, vol. 70, nos. 1–2, 2004, pp. 37–60.

[5] M. Jorgensen and D.I.K. Sjoberg, "An Effort Prediction Interval Approach Based on the Empirical Distribution of Previous Estimation Accuracy," J. Information and Software Technology, vol. 45, no. 3, 2003, pp. 123–136.

[6] Ali Bou Nassif, Danny Ho, and Luiz Fernando Capretz, "Towards an early software estimation using log-linear regression and multilayer perceptron model", Elsevier, SciVerse ScienceDirect, The Journal of Systems and Software, 86, 2013; Pp. 144-160.

[7] Krishnamoorthy Srinivasan and Douglas Fisher, "Machine Learning Approaches to Estimating Software Development Effort", IEEE Transactions on Software Engineering, Vol.21, No.2, February 1995, Pp. 126-137.

[8] S. McConnell, Code Complete: A Practical Handbook of Software Construction, Microsoft Press, second ed., 2004.

[9] M.A. Parthasarathy, Practical Software Estimation – Function Point Methods for Insourced and Outsourced Projects, Infosys Press, Pearson, First Impression, 2007.

[10] Gopalaswamy Ramesh and Ramesh Bhattiprolu, Software Maintenance - Effective Practices for Geographically Distributed Environments, TMH, 2009.

[11] Roger S Pressman, Seventh Edition, Software Engineering, A Practitioner.s Approach; McGraw Hill International Edition.

[12] Thimothy C. Lethbridge and Robert Laganiere, Object-Oriented Software Engineering Practical software development using UML and Java, Tata McGraw-Hill Publish Company Limited, 2008.

[13] Rajib Mall, Fundamentals of Software Engineering, Third Edition, PHI Learning Private Limited, 2011.

[14] S. Rajasekaran, G.A. Vijayalakshmi Pai, Neural Networks, Fuzzy Logic, and Genetic Algorithms, Synthesis and Applications, PHI Learning Private Limited, 2009.

[15] Yegnanarayana, Artificial Neural Netwoks, PHI Learning Private Limited, 2010.

[16] B V A N S S Prabhakar Rao *& P Sita Ramaiah;* Adaptive System for Estimating Development Effort, Journal of Communication, Navigation and Signal Processing (Journal of CNS); ISSN 2277-1735, Vol 1, Issue 1, Jan, 2012, pp 52-56.

[17] G. R. Finnie and G.E. Wittig and J-M. Desharnais, "A Comparison of Software Effort Estimation Techniques: Using Function Points with Neural Networks, Case-Based Reasoning and Regression Models", J. Systems Software, Elsevier Science Inc, 1997; 39: 281-289.

[18] B V A N S S Prabhakar Rao and P Seetha Ramaiah, Software Size Estimation Using Fuzzy Backpropagation Network Method; International Journal of Computer Science Issues(IJCSI), Vol. 9, Issue 1, No 1, January 2012, ISSN (Online): 1694-0814, pp. 339-348.

[19] M. Morisio and M. Ezran, and C. Tully, Success and failure factors in software reuse, IEEE Transactions on Software Engineering, volume 28,4, 2002, pages 340--357

[20] More Success and Failure Factors in Software Reuse, "T. Menzies and J.S. Di Stefano", IEEE Transactions on Software Engineering, 2003.