Applying Machine Learners to GUI Specifications in Formulating Early Life Cycle Project Estimations

Gary D. Boetticher
Department of Software Engineering
University of Houston - Clear Lake
2700 Bay Area Boulevard
Houston, TX 77058 USA
+1 281 283 3805
boetticher@cl.uh.edu

ABSTRACT

Producing accurate and reliable early life cycle project estimates remains an open issue in the software engineering discipline. One reason for the difficulty is the perceived lack of detailed information early in the software life cycle. Most early life cycle estimation models (e.g. COCOMO II, Function Point Analysis) use either the requirements document or a size estimate as the foundation in formulating polynomial equation models. This paper explores an alternative approach using machine learners, in particular neural networks, for creating a predictive effort estimation model. GUI-specifications captured early in the software life cycle serve as the basis for constructing these machine learners. This paper conducts a set of machine learning experiments with software cost estimation empirical data gathered from a "real world" eCommerce organization. The alternative approach is assessed at the program unit level, project subsystem level, and project level. Project level models produce 83 percent average accuracy, pred(25), for the client-side subsystems.

Keywords
Machine learning, machine learners, requirements engineering, software engineering, neural networks, backpropagation, software metrics, effort estimation, SLOC, project estimation, programming effort

1. INTRODUCTION

One of the most important issues in software engineering is the ability to accurately estimate software projects early in the life cycle. Low estimates result in cost overruns. High estimates equate to missed financial opportunities.

From a financial context, more than $300 billion is spent each year on approximately 250,000 software projects [30]. This equates to an average budget of $1.2 million. Coupling these facts with Boehm's observation [2] that project estimates range from 25 to 400 percent early in the life cycle indicates a financial variance of $300 thousand to $4.8 million.

Why does such a high variance exist? One primary reason is the severe lack of data early in the life cycle. In the embryonic stage of a software project the only available artifact is a requirements document. This high-level document provides relatively few metrics (e.g., nouns, verbs, adjectives, or adverbs) for estimating a project's effort. Due to the complexity and ambiguity of the English language, formulating an accurate and reliable prediction, based upon a requirements document, is a nearly impossible task.

Despite this high variance, the ability to generate accurate and reliable estimates early in software life cycle is extremely desirable. Many IT managers are under pressure to offer relatively narrow ranges of estimates regarding anticipated completion rates.

The software engineering discipline recognizes the importance of building early life cycle estimation models. The traditional approach involves formulating a polynomial equation based upon empirical data. Well-known equations include the COnstructive COst MOdel II (COCOMO II) and Function Point Analysis. Each has produced reasonable results since their inception [13, 28]. However, there are several drawbacks in using these well-known equations.

Using these equations is a time-consuming process. The complexity of each requires extensive human intervention and is subject to multiple interpretations [21, 31]. What is needed is an alternative approach for generating accurate estimates early in the software life cycle. An approach which produces accurate estimates, is automated to avoid subjective interpretation; and is relatively simple to implement.

This paper describes a process of applying machine learners, in particular neural networks, in formulating estimation models early in the software life cycle. A series of empirical experiments are based on input and output measures extracted from four different 'real world' project subsystems. The input measures for each experiment are derived by utilizing the GUI interface specification document. The GUI interface document offers the advantage of being an early life cycle artifact rich in objective measures for building effort estimation models.

The set of experiments use 109 different data samples (or program units). Each program unit corresponds to a form consisting of up to twelve different types of widgets (e.g., edit boxes, buttons). Extracted widget counts serve as the input measures. The output measure is the actual, not estimated, effort expended in developing that particular program unit.

Section 2 provides background information and motivation for using machine learners in project estimation. Section 3 discusses related research in the area of machine learning applied to effort estimation. Section 4 describes a set of machine learning experiments. Section 5 offers a discussion of the experiments. And section 6 draws several conclusions and describes future directions.

2. BACKGROUND

Different techniques for cost estimation have been discussed in the literature [2, 16, and 18]. Popular approaches include: algorithmic and parametric models, expert judgment, formal and informal reasoning by analogy, price-to-win, top-down, bottom-up, rules of thumb, and available capacity.

Two well-known parametric approaches for early life cycle estimation are COCOMO II and Function Point Analysis (FPA).

The COCOMO II equation embeds many project parameters within the equation. It is defined as follows [10]:

$$Effort = A * (Size)^B * EM \qquad \text{Equation 1}$$

where

Effort refers to the Person Months needed to complete a project

A	represents the type of project. There are three possible values for this parameter.
Size	is defined by using a SLOC estimate or Function Point Count.
B	is a derived metric which includes the sum of five cost driver metrics.
EM	is an abbreviation for Effort Multiplier. The COCOMO II equation defines seven effort multipliers for early life cycle estimating.

A difficulty in applying the COCOMO II equation is managing the very large solution space. In the early life cycle version, there are 3 options for project type, 5^5 options for the cost drivers, and 5^7 options for the effort multipliers. Multiplying all the options together reveals a search space of 732,421,875 different settings. This excludes the effort value supplied for the *Size* parameter.

A more fundamental problem with COCOMO II is that it requires an estimate for the *size* of the project represented by SLOC of Function Points. If the size of a project were actually known early in the life cycle, then it would be easy to formulate a reasonable effort estimate.

Another parametric approach is Function Point Analysis (FPA). This process starts with the requirements document where a user identifies all processes. Each process is categorized into one of five function types; different Record Element Type, Data Element Types; and File Types Referenced. Based upon the settings chosen, the equation produces an *Unadjusted Function Point* (UFP) for each process. There are seven possible values for each UFP ranging from 3 through 15.

The next step involves defining the *Global System Characteristics* (GSC). Collectively, there are 14 different GSC parameters with 5 possible settings for a total search space of 6,103,515,625 options. The total GSC may range from 0 through 70.

After applying several mathematical operations to the UFP, it is multiplied by the total GSC to produce the final Adjusted Function Point. A software project with only one function point may range from 1.95 to 20.25 Adjusted Function Points. Assuming a mean of 11.1, this produces a variance of 83.7. This Adjusted Function Point variance does not compare well with Boehm's early life cycle variance of 4 [2].

Assuming a perfect Adjusted Function Point value is determined, the next step in the FPA requires the model builder (presumably a project domain expert) to define a constant by which to multiply the final Adjusted Function Point total. This last step, which is totally subjective, is the most critical step in the process and very sensitive to distortion.

The complex nature of COCOMO II and FPA suggests the need for an alternative approach to early life cycle project estimation.

One approach would be to formulate an early life cycle model using a machine learning algorithm. There are different types of machine learners including predictors, classifiers, and controllers. Since this is a predictor type problem, a neural network approach is chosen for conducting a series of experiments.

The goal of these experiments is to define a tool that deterministically constructs an accurate early life cycle estimation model. Deterministically means that there are no subjective measures introduced into the modeling process.

To establish a context for the application of machine learners to software project estimation; the following section describes previous research in this area.

3. RELATED RESEARCH

Related research consists of the utilization of various types of machine learners for predicting project effort. Also, there had been some previous research in using GUI metrics for estimating project effort. This section describes both of these contexts in terms of machine learning algorithms deployed, if applicable, and results achieved.

In [1, 15, 20, 24, and 25], a Case-Based Reasoning (CBR) approach is adopted in constructing a cost model for the latter stages of the development life cycle. Delany [12] also uses a CBR approach applied early in development life cycle.

Chulani [9] uses a Bayesian approach to cost modeling and generates impressive results. He collects information on 161 projects from commercial, aerospace, government, and non-profit organizations [9]. The COCOMO data sets contain attributes that, for the most part, can be collected early in the software life cycle (exception: COCOMO requires source lines of code which must be estimated). Regression analysis was applied to the COCOMO data set to generate estimators for software project effort. However, some of the results of that analysis were counter-intuitive. In particular, the results of the regression analysis disagreed with certain domain experts regarding the effect of software reuse on overall cost.

To fix this problem, a Bayesian learner was applied to the COCOMO data set. In Bayesian learning, a directed graph (the belief network) contains the probabilities that some factor will lead to another factor. The probabilities on the edges can be seeded from (e.g.) domain expertise. The learner then tunes these probabilities according to the available data. Combining expert knowledge and data from the 161 projects yielded an estimator that was within 30% of the actual values, 69% of the time [9]. It is believed that the above COCOMO result of pred(30) = 69% is a high-watermark in early life cycle software cost estimation.

Cordero [11] applies a Genetic Algorithm (GA) approach in the tuning of COCOMO II.

Briand [8] introduces optimized set reduction (OSR) in the construction of software cost estimation model.

Srinivasan [29] builds a variety of models including neural networks, regression trees, COCOMO, and SLIM. The training set consists of COCOMO data (63 projects from different applications). The training models are tested against the Kemerer COCOMO data (15 projects, mainly business applications). The regression trees outperformed the COCOMO and the SLIM model. The neural networks and function point-based prediction models outperformed regression trees.

Samson [26] applies neural network models to predict effort from software sizing using COCOMO-81 data. The neural network models produced better results than the COCOMO-81.

Wittig *et al.* [32] estimated development effort using a neural network model. They achieved impressive results of 75 percent accuracy pred(25).

Boetticher [6] conducted more than 33,000 different neural network experiments on empirical data collected from separate corporate domains. The experiments assessed the contribution of different metrics to programming effort. This research produced a cross-validation rate of 73.26%, using pred(30).

Hodgkinson [19] adopted a top-down approach using a neurofuzzy cost estimator in predicting project effort. Results were comparable to other techniques including least-squares multiple linear regression, estimation via analogy, and neural networks.

Lo *et al.* [23] constructed a GUI effort estimation multivariate regression model using 33 samples. Independent variables consisted of GUI metrics classified into 5 groups: static widgets (labels), data widgets not involving lists (edit boxes, check boxes, radio buttons), data widgets involving lists (list boxes, memo boxes, file lists, grids, combo boxes), action widgets involving the database (buttons), and action widgets not involving the database (buttons). Instead of using the actual effort values for the dependent variable, estimates from 4 experts with a least 1 year of experience were averaged. The average of these estimates ranged from 3 to 48 hours. Variance of the expert's estimates is not presented in the paper. The initial internal results were pred(25) = 75.7% and MARE 20.1%. External results (against another system) yielded were pred(25) = 33.3% and MARE = 192%.

4. MACHINE LEARNING EXPERIMENTS

4.1 General Description

This section describes a series of machine learning experiments based on data gathered from four 'real-world' project subsystems.

Prior to conducting the experiments, it was necessary to decide which ML approach to adopt. A neural network paradigm for creating models seemed like a natural choice. This decision was based upon the author's previous successes using neural networks to model software metrics [3, 4, 5, 6, and 7].

Advantages of using neural networks include [17]: the ability to deal with domain complexity, ability to generalize, along with adaptability, flexibility, and parallelism.

There is also support in the literature for applying neural networks in estimation tasks [22, 26, and 29]. However, some researchers consider the relative merits of neural nets over other machine learning techniques (e.g. decision tree learning) an open issue [27].

4.2 Neural Network Overview

A supervised neural network can be viewed as a directed graph composed of nodes and connections (weights) between nodes. A set of vectors, referred to as a training set, is presented to the neural network one vector at a time. Each vector consists of input values and output values. In Figure 1, the inputs are $\boldsymbol{x}_0$ through $\boldsymbol{x}_{N-1}$ and the output is $\boldsymbol{y}$. The goal of a neural network is to characterize a relationship between the inputs and outputs for the whole set of vectors. During the training of a neural network, inputs from a training vector propagate throughout the network. As inputs traverse the network, they are multiplied by appropriate weights and the products are summed up. In Figure 1, this is $\boldsymbol{w}_i \cdot \boldsymbol{x}_i$. If the summation exceeds some specified threshold for a node, then output from that node serves as input to another node. This process repeats until the neural network generates an output value for the corresponding input portion of a vector. This calculated output value is compared to the desired output and an error value is determined. Depending on the neural network algorithm, either the weights are recalibrated after every vector, or after one pass (called an epoch) through all the training vectors. In either case the goal is to minimize the error total. Processing continues until a zero error value is achieved, or training ceases to converge. After training is properly completed, the neural network model which characterizes the relationship between inputs and outputs for all the vectors is embedded *within the architecture (the nodes and connections) of the neural network*. After successful completion of training, a neural network architecture is frozen and tested against an independent set of vectors called the test set. If properly trained, the neural network produces reasonable results against the test suite.

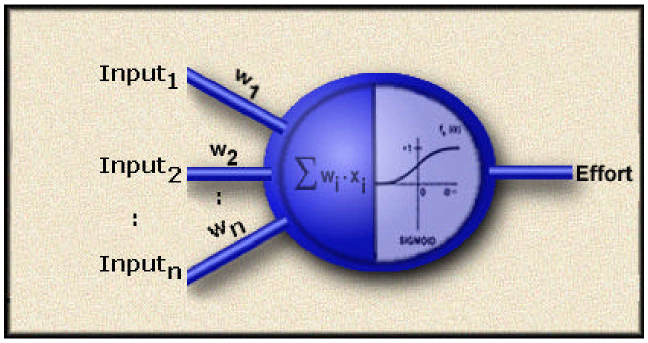

Figure 1: Sample Neural Network

All experiments utilize a variant of the backpropagation neural network, called the quickprop. The quickprop algorithm converges much faster than a typical backpropagation approach [14]. It uses the higher-order derivatives in order to take advantage of the curvature [14]. The quickprop algorithm uses second order derivatives in a fashion similar to Newton's method. Using quickprop in all the experiments also ensures stability and continuity.

4.3 Description of Experiments

Four datasets were used in the experiments. The GUI metrics were extracted from one of four major subsystems of an electronic commerce (procurement) product used in the process industry. Table 1 shows each major system along with the number of program units.

Table 1. Description of the major subsystems.

Major Subsystem	**Number of program units**
Buyer Administrator	7
Buyer	60
Distribution Server	10
Supplier	32
TOTAL	**109**

Each program unit consists of a GUI form along with corresponding code written in Delphi.

In the context of neural networks, a program unit is referred to as a vector. Each vector consists of a set of inputs along with a set of outputs. Each vector contained twelve input parameters based upon GUI categories described below. These include: Buttons; Charts; Check boxes/radio buttons; Combo boxes; Grid (string grid, database grid); Grid Tabs; Edit boxes; Labels; Memo/List boxes; Menu bars; Navigation bars; and Trees.

The grid tabs refers to how many tabs were available for each grid. For example, the default is three (worksheets) in Microsoft© EXCEL.

The output consists of the actual, not estimated, effort required for developing each program unit. Effort values ranged from 1 to 160 hours. All program units were developed by a single developer. This reduces the impact of the human element in terms of various skill and knowledge levels in the model formulation process.

All experiments use a fully-connected neural network architecture of 12-5-11-5-1 (see Figure 2), meaning twelve inputs, one layer of five hidden nodes, followed by another hidden layer of eleven hidden nodes, followed by another hidden layer of five nodes, then an output layer of one node.

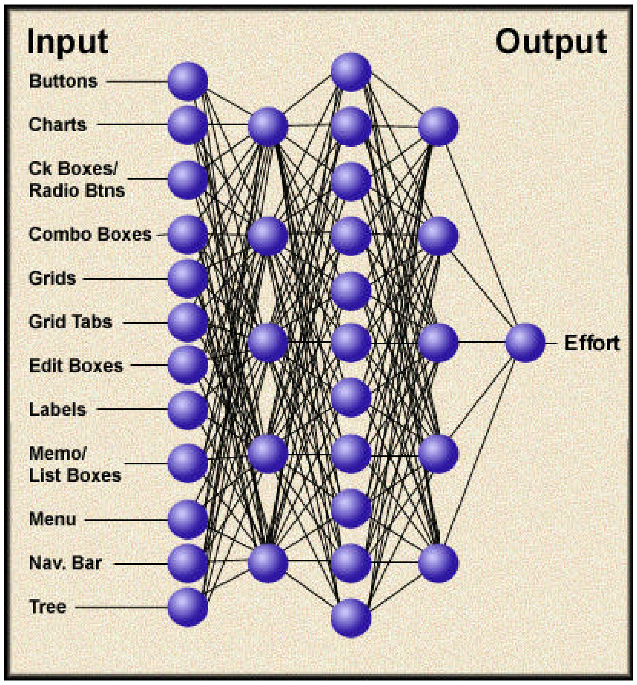

Figure 2: 12-5-11-5-1 Neural Network Architecture

In order to minimize experimental variance among experiments, we standardized the experimental process. Different components of each neural network model remain constant. Alpha, which represents how quickly a neural network learns, may range from zero to one. Alpha for these experiments is always one. Momentum, a variable which helps neural networks break out of local minima, may also range from zero to one. Momentum is also always set to 1. The threshold function is a function associated with each node after the input layer. Function selection determines when a node fires. Firing a node essentially propagates a value further through the network. All experiments use an asymmetrical sigmoid function as a threshold function.

One scan through the training data is considered an epoch. Each experiment iterates through 10,000 epochs. Initial trials indicated that 5,000 to 7,000 epochs were sufficient for determining the highest correlation along with the highest accuracy (with respect to the test data). The “most accurate test results” is defined as number of correct matches within 25 percent, or pred(25), of the actual effort values for the test vectors.

4.4 Experiments and Results

Four different neural network experiments are performed for each subsystem. For each set of experiments, data from one of the subsystems is treated as a test suite and the data from the other three subsystems is combined into a training set. Each experiment is performed ten times in order to discount any outliers. After each experiment all the weights in the neural network are reset.

Table 2 presents the Pred(25) and MARE for each of the four types of experiments. The Pred(25) of 64.3 percent for the ‘Buyer Administrator’ test set is reasonable, however the other Pred(25) values are low relative to [9, 23]. The relatively low values for the Pred(25) may be attributed to range of effort values, 1 to 160 hours. Nineteen percent of all the vectors had an effort of one. As a consequence, the neural network experienced difficulty in adequately approaching these low effort values. Secondly, the test sets were organized by subsystem. This led to extrapolation issues. A total of 15 vectors from three different test sets contained maximum values for one or more inputs/output. The only way to avoid any extrapolation problems would be to continuously retain the 15 vectors in the training set. This is not a realistic solution. Finally, for some of the input metrics, there were less than three instances of values. Essentially, the corresponding metric contributed little to the training process.

The relatively large range of effort values, along with the large percentage of effort values less than or equal to five (65 percent) inflated the MARE values.

Table 2. Results from initial experiments

	Buyer Admin		Buyer Client		Distribution Server		Supplier Client	
RUN	**Pred (25)**	**MARE**	**Pred (25)**	**MARE**	**Pred (25)**	**MARE**	**Pred (25)**	**MARE**
1	71%	54%	32%	231%	40%	212%	34%	176%
2	57%	54%	33%	355%	50%	160%	38%	126%
3	57%	98%	32%	258%	50%	319%	44%	179%
4	57%	72%	23%	385%	60%	84%	41%	200%
5	57%	89%	27%	248%	50%	66%	38%	288%
6	71%	38%	33%	210%	50%	176%	41%	171%
7	71%	82%	25%	253%	50%	74%	38%	189%
8	71%	44%	18%	312%	50%	55%	38%	198%
9	71%	53%	23%	254%	50%	492%	38%	178%
10	57%	103%	25%	652%	60%	68%	44%	131%
AVE	**64.3%**	**68.5%**	**27.2%**	**316%**	**51%**	**172%**	**39.1%**	**184%**

One question is whether the results are any different when perceived from the subsystem (project) level. Viewing the results from a subsystem perspective, as opposed to a program unit perspective, dramatically improves upon the results. Table 3 shows the Pred(25) and MARE for each of the four major subsystems. The results are determined by summing the calculated effort values for each program unit for each individual experiment. Hence three of the four models produced estimates 80 percent or higher with MARE values less than 18 percent. The best case generates a Pred(25) of 90 percent and a MARE of 12.2 percent.

Table 3. Aggregate analysis of the subsystems

Subsystem	Pred(25)	MARE
Buyer Administrator	80%	17.6%
Buyer Client	80%	14.6%
Distribution Server	20%	96.7%
Supplier Client	90%	12.2%

A natural extension of the subsystem results would be to aggregate them into corresponding project results. Table 4 presents two worst-case scenarios and one average-case scenario. For the worst-case scenarios the minimum average efforts and maximum average efforts are totaled. The average-case scenario is the average estimate (for the ten experiments) for each subsystem.

Table 4. Aggregate analysis of the project

Subsystem	Worst case Min.	Worst Case Max.	Ave. Cases	Actual Effort
Buyer Admin.	158	289	220	215
Buyer Client	958	1660	1313	1202
Dist. Server	114	246	170	307
Supplier Client	505	790	644	576
TOTAL	**1735**	**2985**	**2347**	**2300**

Thus, in the worst-case scenarios, the collective estimates range from 24.6% below the actual project effort to 29.8% above the actual project effort. The average-case estimate is within 2% of the actual project effort.

5. DISCUSSION

The vectors were grouped according to the project subsystems, rather than applying a statistical process for organizing the vectors. This followed the natural contours of the project at the expense of generating artificially better results.

One question that persists is, "Why are the results so low for the distribution server presented in table 3?" In general, server-side applications are not intended to be interactive. As a consequence, GUI-based metrics might not be appropriate for server-side software. However, the server-side metrics did not taint the results presented in table 4.

This work extends the previous research of *Lo et al* [23] by assessing more data in greater detail using better effort values. See Table 5 below.

Table 5. Comparison of current with previous research

Category	Lo [23]	Current work
Data Samples	33	109
Number of GUI metrics utilized	5	12
Does formulating GUI metrics require human interpretation?	Yes	No
Model type	Multivariate regression	Neural networks
Nature of effort values	Defined through expert estimates	Based on actual effort values
Best results	Pred(25) = 75.7%, MARE = 20.1%	Pred(25) = 90%, MARE = 18%

Extrapolation issues frequently arose during the conducting of experiments. Adding more data might reduce the frequency of extrapolation, but it will never eliminate the problem. The extrapolation issues did not seem to affect the results produced at the subsystem and overall project levels.

It is worth noting the software environment from which this data emerged. This organization did not have a formal process. Most likely it would be characterized as a one in the context of the Capability Maturity Model. Thus, the experiments show that it is possible to construct very reasonable early life cycle project estimates in light of a poorly defined process.

6. CONCLUSIONS AND FUTURE DIRECTIONS

This research describes a process of formulating early life cycle project estimates based on GUI specifications. Twelve different types of widgets are counted. Most, if not all, of the widgets are frequently used and well understood within the software industry.

Using well-defined widgets simplifies the measurement gathering process. Thus, the learning curve is rather shallow (as compared to COCOMO II or FPA) for understanding the model formulation process.

The model formulation process is repeatable since there is no subjectivity involved in counting the widgets.

The results at the program unit level seemed low. However, the results improved dramatically when viewed from the project subsystem level. Three of the four subsystems produced a Pred(25) of 80% or higher and a MARE of 18% or lower.

One strategy to improve upon the results at the program unit level would be to reduce the number of classes for the effort values. Thus a set of actual effort values ranging from 10 to 15 hours may be collapsed into 12.5 hours.

Since the literature describes various applications of Machine Learner in effort estimation, it would be plausible to conduct additional experiments using other machine learning algorithms.

The process does not fare very well in situations where development is computationally complex with little or no GUI specifications. This is evident in the server-side results. One future activity would be to integrate the neural network-based GUI effort estimation approach with an algorithmic approach, such as COCOMO II or FPA. This could reduce the complexity related to COCOMO II and FPA and extend the neural network GUI approach to accommodate non-GUI software development.

ACKNOWLDEGMENTS

The author wishes to express his deepest gratitude for the insightful comments and constructive criticism from the peer reviewers.

REFERENCES

1. Bisio, R., F. Malabocchia, "Cost Estimation of Software Projects through Case-Base Reasoning." *Case-Based Reasoning Research and Development. First International Conference*, ICCBR-95 Proceedings, 1995, Pp.11-22.

2. Boehm, B., *Software Engineering Economics*, Englewood Cliffs, NJ, Prentice-Hall, 1981.

3. Boetticher, G., K. Srinivas and D. Eichmann, "A Neural Net-Based Approach to Software Metrics," *Proceedings of the 5th International Conference on Software Engineering and Knowledge Engineering*, June 1993, Pp. 271-274. Available from http://nas.cl.uh.edu/boetticher/publications.html

4. Boetticher, G. and D. Eichmann, "A Neural Net Paradigm for Characterizing Reusable Software," *Proceedings of the First Australian Conf. on Software Metrics*, November 1993, Pp. 41-49. Available from http://nas.cl.uh.edu/boetticher/publications.html

5. Boetticher, G., "Characterizing Object-Oriented Software for Reusability in a Commercial Environment," *Reuse '95 Making Reuse Happen – Factors for Success*, Morgantown, WV, August 1995. This paper is available at: http://nas.cl.uh.edu/boetticher/publications.html

6. Boetticher, G., "An Assessment of Metric Contribution in the Construction of a Neural Network-Based Effort Estimator," Second Int. Workshop on Soft Computing Applied to Soft. Engineering, 2001. Available at http://nas.cl.uh.edu/boetticher/publications.html

7. Boetticher, G., "Using Machine Learning to Predict Project Effort: Empirical Case Studies in Data-Starved Domains," Workshop on Model-Based Requirements Engineering, 2001. Available at http://nas.cl.uh.edu/boetticher/publications.html

8. Briand, Lionel C., Victor R. Basili, and William Thomas. Pattern Recognition Approach for Software Engineering Data Analysis. *IEEE Trans. on Soft. Eng.*, November 1992, Pp. 93-942.

9. Chulani, S., and Boehm, B., and B. Steece, "Bayesian Analysis of Empirical Software Engineering Cost Models", *IEEE Transaction on Software Engineering*, 25 4, July/August, 1999.

10. COCOMO II Model Definition Manual, 1999. Available from the following link: http://sunset.usc.edu/research/COCOMOII/Docs/modelman.pdf

11. Cordero, R., M. Costramagna, and E. Paschetta. "A Genetic Algorithm Approach for the Calibration of COCOMO-like Models," *12th COCOMO Forum*, 1997.

12. Delany, S.J., P. Cunningham, “The Application of Case-Based Reasoning to Early Project Cost Estimation and Risk Assessment,” *Department of Computer Science, Trinity College Dublin, TDS-CS-2000-10*, 2000.

13. Devnani-Chulani, Sunita, Clark, B., Barry Boehm, “Calibration Approach and Results of the COCOMO II Post-Architecture Model,” ISPA, June 1998.

14. Fahlman, S.E., *An Empirical Study of Learning Speed in Back-Propagation Networks*, Tech Report CMU-CS-88-162, Carnegie Mellon University, September 1988.

15. Finnie, G.,R., Wittig, G.,E., J.M. Desharnais, “Estimating software development effort with case-based reasoning,” *Proceedings of International Conference on Case-Based Reasoning*, D. Leake, E. Plaza, (Eds), 1997, Pp.13-22.

16. Heemstra, F. “Software Cost Estimation,” *Information and Software Technology*, October 1992, Pp. 627-639.

17. Hertz, J., Krogh A., R.G. Palmer., *Introduction to the Theory of Neural Computation,* Addison Wesley, New York, 1991.

18. Hihn, J., H. Habib-Agahi, “Cost Estimation of Software Intensive Projects: A Survey of Current Practices,” *Proceedings of the International Conference on Software Engineering*, 1991, Pages 276-287.

19. Hodgkinson, A.C., Garratt, P.W., “A Neurofuzzy Cost Estimator,” *Proc. 3rd International Conf. Software Engineering and Applications (SAE)*, 1999, pp. 401-406.

20. Kadoda, G., Cartwright, M., Chen, L. and Shepperd, M., “Experiences Using Case-Based Reasoning to Predict Software Project Effort,” *Empirical Software Engineering Research Group Technical Report*, Bournemouth University, January 27 2000.

21. Kemerer, Chris, “Reliability of Function Points Measurement: A Field Experiment,” *Communications of the ACM 36*, 2 (February 1993), Pp. 85-97.

22. Kumar, S., Krishna, B. A., Satsangi, P.J., "Fuzzy Systems and Neural Networks in Software Engineering Project Management," *Journal of Applied Intelligence*, 4, 1994, Pp. 31 - 52.

23. Lo, R., Webby, R., R. Jeffrey, “Sizing and Estimating the Coding and Unit Testing Effort for GUI Systems,” *Proceedings of the 3rd International Software Metrics,* Los Alamitos: IEEE Computer Society Press, 166-173, 1996.

24. Mukhopadhyay, Tridas, and Sunder Kekre, “Software effort models for early estimation of process control applications,” *IEEE Transactions on Software Engineering*, 18 (10 October), 1992, Pp. 915-924.

25. Prietula, M., S. Vicinanza, T. Mukhopadhyay, “Software effort estimation with a case-based reasoner,” *Journal of Experimental and Theoretical Artificial Intelligence*, 8(3-4), 1996, Pp. 341-363.

26. Samson, B., Ellison, D., Dugard, P., “Software Cost Estimation Using an Albus Perceptron,” Information and Software Technology, 1997, pp. 55-60.

27. Shavlik, J.W., Mooney, R.L., and G.G. Towell, Symbolic and Neural Learning Algorithms: An Experimental Comparison, *Machine Learning*", 1991, Pp. 111-143.

28. Siddiqee, M. Waheed. "Function Point Delivery Rates Under Various Environments: Some Actual Results," 259-264. *Proceedings of the Computer Management Group's International Conference*. San Diego, CA, December 5-10, 1993.

29. Srinivasan, K., and D. Fisher, “Machine Learning Approaches to Estimating Software Development Effort,” *IEEE Trans. Software Engineering*, February, 1995, Pp. 126-137.

30. The Standish Group, Chaos Chronicles III, The Standish Group, January, 2003.

31. Wittig, G. E., G.R. Finnie, “Software Design for the Automation of Unadjusted Function Point Counting,” *Business Process Re-Engineering Information Systems Opportunities and Challenges*, *IFIP TC8 Open Conference*. Gold Coast, Queensland, Australia, May 8-11, 1994, Pp. 613-623.

32. Wittig, G., G. Finnie, “Estimating software development effort with connectionist models,” Information and Software Technology, 1997, pp. 469-476.