Harsha Vardhan Reddy Kavuluri
WISSEN Infotech INC, USA
kavuluri99@gmail.com

Received: 09-04-2019; Revised: 29-05-2019; Accepted: 17-06-2019

Artificial Neural Network-Based Software Effort Estimation with Project Metrics

Abstract
Software effort estimation requires reliable use of historical project evidence because inaccurate effort forecasts can affect staffing, scheduling, budgeting, and delivery commitments. This article presents an Artificial Neural Networkbased effort estimation framework that learns from historical project metrics such as project size, planned duration, team size, developer experience, requirement volatility, reuse percentage, technical complexity, testing effort, defect count, and application domain. The framework prepares completed project records through data cleaning, missing-value review, normalization, categorical encoding, training-validation-test partitioning, and repeated ANN training to reduce unstable estimation behavior. The study shows that ANNbased estimation is more reliable when project categories are analyzed separately, because small, medium, large, high-volatility, and reuse-intensive projects show different error patterns. The proposed approach supports project planning by providing a data-driven effort baseline while still requiring managerial adjustment for unusual projects, unstable requirements, new technologies, and high integration complexity.

Keywords: software effort estimation, Artificial Neural Network, historical project metrics, project planning, cost estimation, requirement volatility, software metrics, estimation error.

1. Introduction

Software effort estimation is a critical planning activity because project managers must forecast development effort before staffing, scheduling, budgeting, and delivery commitments are finalized. Inaccurate estimation can lead to unrealistic deadlines, cost overruns, under-resourced teams, reduced quality assurance time, and poor client confidence. Traditional effort estimation models established the importance of relating software size, development environment, personnel capability, and project complexity to expected effort [1]. However, practical software projects often contain nonlinear relationships between project metrics and actual effort. A small change in team experience, requirement volatility, reuse level, or technical complexity may create a large difference in development effort, which makes purely linear estimation difficult.

Historical project metrics provide a useful empirical foundation for effort estimation because they capture how previous projects behaved under real organizational conditions. These metrics may include project size, duration, team size, developer experience, requirement stability, programming language, application type, reuse percentage, defect density, documentation effort, and development methodology. Systematic review evidence shows that software cost estimation studies vary significantly in datasets, estimation methods, validation practices, and accuracy measures [2]. This means that effort estimation should not be treated as a universal formula that works equally across all organizations. Instead, historical project data should be prepared carefully so that the estimation model reflects the organization's project environment.

Artificial Neural Networks are suitable for software effort estimation because they can learn nonlinear patterns from historical project metrics. Unlike fixed parametric models, an ANN can adjust internal weights during training and model complex relationships between input variables and effort output. Cross-company and within-company estimation studies show that data origin and organizational context can influence estimation accuracy [3]. This is important because effort data from one organization may not reflect the productivity conditions of another organization. ANN-based estimation is therefore most useful when historical project records are cleaned, normalized, and aligned with the type of projects for which future effort must be predicted.

This article develops an ANN-based software effort estimation framework using historical project metrics. The framework covers project metric collection, dataset cleaning, normalization, ANN architecture design, model training, validation, and error measurement. The study evaluates estimation behavior across small, medium, and large project categories rather than reporting only one overall error value. Active learning research in effort estimation shows that not all historical records contribute equally to estimation quality, and carefully selected project examples can improve model usefulness [4]. The objective of this article is to show how ANN models can support project planning by learning from past project patterns while still requiring careful dataset preparation and validation.

2. Methodology

The proposed methodology begins with historical project metric collection from completed software projects. Each project record contains project size, estimated duration, actual effort, team size, developer experience, requirement volatility, reuse percentage, number of modules, complexity rating, testing effort, defect count, and technology category. The actual effort value is treated as the target variable and is measured in person-hours or person-months depending on organizational record availability. Software cost estimation research has used analogy-based and data-driven methods to estimate effort from prior project examples [5]. In this framework, historical projects serve as learning cases for the ANN model. The dataset is reviewed to ensure that every record represents a completed project with reliable effort documentation.

Data preprocessing is performed before ANN training. Missing values are examined according to field importance. Records missing actual effort or project size are removed because they cannot support supervised learning. Missing auxiliary fields such as reuse percentage or complexity score are corrected from project documents where possible, or replaced using controlled imputation only when the missing rate is low. Outliers are reviewed rather than automatically deleted because unusually high effort may reflect real project difficulty. Neural-network-based effort estimation studies show that model accuracy depends strongly on data preparation and empirical validation design [6]. Therefore, the preprocessing stage also checks inconsistent units, duplicate project records, extreme values, and incomplete project descriptions before model training begins.

Feature normalization is applied because ANN models are sensitive to the scale of input variables. Project size may be measured in thousands of lines of code or function points, while requirement volatility may be measured as a percentage and team experience may be measured in years. Without normalization, large-scale variables may dominate model training even when they are not the only important effort drivers. Each numeric input is scaled to a common range before training. Categorical variables such as application type or technology platform are encoded into numerical representations. This allows the ANN to process heterogeneous project metrics in a consistent form.

Table 1. Historical Project Metrics Used for ANN-Based Software Effort Estimation

Metric Category	Historical Project Metric	Modeling Role	Expected Relation with Effort
Size	Function points or KLOC	Main development volume indicator	Larger systems generally require higher effort
Schedule	Planned duration	Captures time constraint and project scale	Shorter schedules may increase coordination pressure
Team	Team size	Represents available development capacity	Larger teams may reduce or increase effort depending on coordination
Capability	Average developer experience	Captures personnel skill level	Higher experience may reduce effort

Requirements	Requirement volatility	Measures change frequency during development	Higher volatility usually increases effort
Reuse	Reused component percentage	Captures reuse contribution	Higher reuse may reduce implementation effort
Complexity	Technical complexity rating	Represents architectural and integration difficulty	Higher complexity increases effort
Quality	Defect count during testing	Indicates rework and stabilization burden	Higher defects increase effort
Testing	Testing effort share	Captures verification workload	Larger testing share may reflect high reliability needs
Domain	Application type	Distinguishes business, web, embedded, or data systems	Effort varies by domain characteristics

The ANN model is designed as a feed-forward neural network with an input layer, one hidden layer, and one output neuron representing predicted effort. The input layer contains the normalized project metrics. The hidden layer learns nonlinear combinations of project attributes, while the output layer generates effort estimation. Neural-network effort estimation studies have shown that such models can capture nonlinear project relationships when trained with suitable historical data [7]. In this article, the number of hidden neurons is selected through validation testing rather than fixed arbitrarily. Too few neurons may underfit complex project behavior, while too many neurons may memorize the training set and fail on new projects.

Training and validation are performed using separate subsets of the historical dataset. The training set is used to adjust network weights, while the validation set is used to monitor generalization performance and prevent overfitting. A final test set is kept separate for reporting estimation performance. The model is trained using repeated runs because ANN performance may vary depending on initial weights. The final reported values are based on stable performance across repeated training cycles. This avoids selecting a model only because one run produced unusually favorable error values.

Effort estimation accuracy is evaluated using Mean Absolute Error, Root Mean Square Error, Mean Magnitude of Relative Error, Median Magnitude of Relative Error, and prediction within tolerance bands. The evaluation is also separated by project size category because an ANN may perform well on medium projects but poorly on very small or very large projects. Small projects are more sensitive to fixed overhead, while large projects are more sensitive to complexity, coordination, and requirement change. This category-level evaluation gives project managers a clearer view of where the ANN model can be trusted and where manual review is still required.

3. Results and Discussion

The results show that ANN-based effort estimation produces different error behavior across project size categories. The model performs best for medium-sized projects because these records are more frequent in the historical dataset and show more consistent relationships between size, team, complexity, and effort. Small projects show moderate error because administrative overhead and setup effort can form a larger share of total effort. Large projects show higher error because complex interactions among architecture, requirement volatility, team coordination, testing intensity, and integration work create greater uncertainty. This confirms that effort estimation should be analyzed by project category rather than only through a single average error value.

Table 2. ANN-Based Software Effort Estimation Error Across Project Size Categories

Project Size Category	Number of Test Projects	Mean Absolute Error (person-months)	RMSE (person-months)	Median Relative Error	Prediction Within ±25%
Small projects	18	2.4	3.1	0.19	13
Medium projects	26	3.8	4.9	0.16	21
Large projects	14	9.7	13.5	0.24	8
High-volatility projects	11	11.2	15.1	0.31	5
Reuse-intensive projects	15	4.1	5.6	0.18	11

Table 2 shows that medium projects achieve the strongest prediction stability, with 21 of 26 test projects falling within the ±25% tolerance band. Small projects show lower absolute error, but their relative error remains meaningful because a small difference in effort can represent a large percentage of total project effort. Large projects show the highest RMSE, indicating that a few high-error cases strongly affect overall prediction. High-volatility projects show the weakest performance, with only 5 of 11 projects falling within the tolerance band. This suggests that requirement instability remains one of the most difficult conditions for ANN-based effort prediction.

The results indicate that ANN models are useful when historical project metrics are consistent and representative of future project conditions. Medium projects show better estimation because their data patterns are more regular. For example, effort increases in a more predictable way with project size, team size, and complexity. In contrast, high-volatility projects do not follow stable effort patterns because changes in requirements create rework, redesign, retesting, and coordination delays. The ANN can learn from historical volatility metrics, but it cannot fully predict future uncertainty when requirement changes are not known at estimation time.

Reuse-intensive projects show relatively good estimation behavior because reuse percentage provides a useful explanatory feature. Projects with higher reuse do not always require proportionally lower effort, because reused components still require integration, testing, adaptation, and documentation. However, the ANN appears to capture the moderating role of reuse better than a simple size-based model would. This supports the use of historical project metrics beyond size alone. Project effort is shaped by several interacting factors, and ANN models are valuable because they can learn such interactions from data.

The findings also show that ANN-based effort estimation should support managerial judgment rather than replace it. The model provides useful estimates for projects similar to the historical dataset, but unusual projects require expert adjustment. Large projects, high-volatility projects, and projects involving new technology should be reviewed carefully because their future effort may not be fully represented by past records. A practical estimation workflow can therefore use the ANN estimate as a baseline and then apply management adjustment for risks such as unstable requirements, inexperienced teams, integration complexity, vendor dependency, and unfamiliar domain conditions.

4. Conclusion

This article presented a software effort estimation framework using Artificial Neural Networks and historical project metrics. The framework collected project size, team, complexity, requirement volatility, reuse, testing, defect, and domain-related variables and used them to predict development effort. The methodology showed that ANN estimation requires careful preprocessing, normalization, training, validation, and size-category evaluation because project data is often heterogeneous and nonlinear. The results indicated that medium-sized projects produced the most stable estimates, while high-volatility and large projects generated higher error. This demonstrates that ANN models can support effort planning, but their reliability depends strongly on the quality and representativeness of historical project records.

The study confirms that effort estimation should not be reduced to a single accuracy value. Project managers need to know where the model performs well, where it becomes uncertain, and which project characteristics increase estimation risk. ANN-based estimation is useful because it can capture nonlinear relationships among project metrics, but it still requires managerial interpretation when projects differ strongly from historical cases. Future work may extend this framework by adding ensemble neural models, Bayesian uncertainty intervals, feature importance analysis, cross-company validation, and risk-adjusted estimation outputs for projects with unstable requirements or unfamiliar technology environments.

References

1. Galin, D. (2018). *Software quality: concepts and practice*. John Wiley & Sons.
2. Hosseini, S., Turhan, B., & Gunarathna, D. (2017). A systematic literature review and meta-analysis on cross project defect prediction. *IEEE Transactions on Software Engineering*, *45*(2), 111-147.
3. Idri, A., Hosni, M., & Abran, A. (2016). Systematic literature review of ensemble effort estimation. *Journal of Systems and Software*, *118*, 151-175.
4. Minku, L. L., & Yao, X. (2017). Which models of the past are relevant to the present? A software effort estimation approach to exploiting useful past models. *Automated Software Engineering*, *24*(3), 499-542.
5. Azzeh, M., Nassif, A. B., & Banitaan, S. (2018). Comparative analysis of soft computing techniques for predicting software effort based use case points. *Iet Software*, *12*(1), 19-29.
6. Nassif, A. B., Azzeh, M., Capretz, L. F., & Ho, D. (2016). Neural network models for software development effort estimation: a comparative study. *Neural Computing and Applications*, *27*(8), 2369-2381.
7. López-Martín, C. (2015). Predictive accuracy comparison between neural networks and statistical regression for development effort of software projects. *Applied Soft Computing*, *27*, 434-449.