International Journal of Science, Strategic Management and Technology
Volume **02** Issue **05** May-**2026** | ISSN: **3108-1762 (Online)** | Impact Factor: 3.8
An International, Peer-Reviewed, Open Access Scholarly Journal Indexed in recognized academic databases

Software Effort Estimation using Machine Learning

Satyam Choubey
B. Tech CSE
GalgotiasUniversity
Greater Noida, India

Vikas Tiwari
B. Tech CSE
GalgotiasUniversity
Greater Noida, India

Dr. Pooja Sapra
AssistantProfessor, SCSE
GalgotiasUniversity
Greater Noida, India

https://doi.org/10.55041/ijsmt.v2i6.079

Cite this Article: Choubey, S. & Tiwari, V. (2026). Software Effort Estimation using Machine Learning. International Journal of Science, Strategic Management and Technology, *02*(6). https://doi.org/10.55041/ijsmt.v2i6.079

***Abstract*—**The forecast of effort needed to complete software development is essential to control costs, schedules, and allocation of resources in the current projects. However, initial estimates are often inaccurate because of the multifaceted nature of project data. This is the kind of data that is usually inconsistent, non-linear relationships, and the effect of both human and technical aspects, which classical models can hardly address entirely account for. The machine proposed in this paper is a novel, two-part machine method of learning that is used to approximate software development effort and spot projects likely to go over budget limits. The framework takes a blended strategy: a stacked ensemble regressor, a combination of Random Forest and Gradient Enhancing algorithms, to generate accurate effort predictions, and a high-risk project identification layer to secondary classification layer of budget overruns. Experiment on the Desharnais and NASA93 datasets showed interesting outcomes, as the model obtained a Mean Magnitude of Relative Error (MMRE) of 0.138 and a classification accuracy of 91.5%. Additional discussion of error rates means that the model provides reliable risk estimates and without producing too many false alarms.

Keywords- Software Effort Estimation, Machine Learning, Stacked Ensemble, Risk Classification, Random Forest, Gradient Boosting, MMRE.

Index Terms—Software Effort Estimation, Machine Learning, Stacked Ensemble, Risk Classification, Random Forest, Gradient Boosting, MMRE.

I. Introduction

The amount of work that software development will require is an extremely difficult aspect of project planning to predict. The current software is influenced by things that change rapidly such as the desires of the users, the various types of systems it must operate on and the manner in which individuals construct the software. All this makes it difficult to predict the length of things in the initial stages. When such estimates are inaccurate, it normally implies that the project will be more expensive, time-consuming and inefficient when it comes to resource utilization. This may make or kill a software project. This is why the estimation of effort is a problem to be considered that can be solved by means of data and analysis and not only personal experience and intuition.

This study goes beyond mere estimations that are determined by the size of a project. Rather it uses a more advanced method. It applies machine learning to evaluate previous projects, with consideration to such factors as Function Points, team expertise, and management skills. These factors do not simply affect each other directly but interact in complex, unanticipated manners that can not be entirely understood using formulaic means. Machine learning models will be able to comprehend these complicated relationships more effectively by examining previous project data. This enables them to produce numerical effort forecasts as well as useful information regarding project risk and planning.

A. The Estimation Problem

Software Effort Estimation (SEE) is the prediction of the effort required to develop a software system, which is done by making predictions based on quantifiable factors of the software system, including its size, complexity, and experience of the development team. The input-output mapping between these inputs and the final effort is not usually proportional. As an example, increasing the number of developers does not necessarily decrease the delivery time due to a wellknown Brooks Law effect of overhead in coordination and communication with the size of the team [2].

One of the core difficulties in SEE is the large uncertainty present during early project phases. At this stage, limited design information is available, yet key budgeting and staffing decisions must be made. In addition, real-world datasets often contain atypical projects with extremely high or low effort, which can distort standard regression models. These challenges motivate the use of non-linear, robust learning techniques.

B. Risk Classification vs. Regression

The majority of effort estimation research tests models with numeric error metrics of Mean Absolute Error (MAE). But in practice project management it can be more useful to have an indication of whether a project will run over its budget or not, than a specific number of hours predicted. This is why this paper supports regression with binary risk prediction that identifies the projects as either being Low Risk or High Risk. This enables the use of confusion matrix metrics:

- True Positives (TP): Projects correctly identified as highrisk.

- False Positives (FP): Projects that are marked as risky even though they are not.

C. Research Objectives and Contributions

This study presents a hybrid learning strategy that aims at giving accurate estimations that will be used directly in decision-making. A hybrid, layered approach to learning is the core of this approach. The key contributions of the work are:

1) A Powerful Prediction Engine: It is based on the stacked regression model, which combines the power of Random Forest and Gradient Boosting to reduce the error in prediction.

2) Risk-Aware Insights: It uses a risk classification element, converting predictions of effort to a priori indications of possible budget overruns.

3) Efficiency Analysis: The analysis of the computational requirements of the entire system is included in the study.

4) Performance Validation: The effectiveness of the framework is also tested with statistical comparisons to the baseline models using MMRE and confusion-matrixbased measures.

This paper will follow the following pattern: In Section II, we are going to explore the available literature. Next, Section III presents the mathematical basis to our work. Next, we are taken through the first data exploration in Section IV. Section V describes the method applied in this work. The computational challenges will be discussed in section VI. In Section VII, we will discuss the results of our experiments. In Section VII we will discuss the practical importance of our results, and in Section IX we will conclude and give future research directions.

II. Literature Review

The estimation of software effort has been through three key stages. First, the expertise of the experts used to estimate the effort of development. Subsequently, algorithmic models emerged to formalize estimation process. In more modern times, machine learning methods are extensively applied to process non-linear and complex project data.

A. Algorithmic and Parametric Models

The groundbreaking contributions of Boehm to the COCOMO-81 and the successor COCOMO II were the foundation of the algorithmic methods of estimation. In essence, these models are based on a particular formula to estimate development effort depending on the project size and list of predefined cost related factors.

$$E = a \cdot (Size)^b \cdot \prod_{i=1}^{15} EM_i \quad (1)$$

where *a* and *b* are constants taken from historical data, and *EM* are effort multipliers. While interpretable, researchers like Kemerer [4] have highlighted their inability to handle categorical data effectively. They rely on the assumption of homoscedasticity (constant variance), which is rarely true in software data.

B. Machine Learning Approaches

During the 2000s, the emphasis was made on data-based methods. Case-Based Reasoning (CBR) was first introduced by Shepperd and Schofield [5] and estimates effort by finding similar past projects using the Euclidean distance. Although good at analogical reasoning, CBR also suffers as the dimensionality of features grows (the Curse of Dimensionality).

Corazza et al. have used Support Vector Regression (SVR) to deal with non-linear relationships [10]. SVR projects input vectors to high-dimensional feature spaces with the help of kernel functions (RBF, Polynomial). Nonetheless, SVR is very sensitive to outlier data, an attribute typical of software metrics.

C. Deep Learning vs. Ensembles

Recent research has used Deep Learning (DL), namely, Long Short-Term Memory (LSTM) networks, on SEE. DL is very good at processing images and text, but can be doubted when it comes to tabular software metrics. DL needs large datasets to optimize millions of parameters; the largest software datasets only have less than 1,000 projects. According to our literature review, in the case of tabular data, Ensemble Tree-based models (Random Forest, XGBoost) are always more effective than Deep Learning because they can work with a small sample size, and categorical variables do not require a large-scale to be effectively deployed.

TABLE I
Comparative Analysis of Related Works

Author	Method	Dataset	MMRE
Boehm [3]	COCOMO II	Private	0.60+
Shepperd [5]	CBR (Analogy)	Desharnais	0.42
Corazza [10]	SVR (RBF)	NASA93	0.38
Kocaguneli [6]	Random Forest	PROMISE	0.22
Proposed (This Work)	Stacked Ensemble	Combined	0.138

III. Theoretical Framework

This section defines the mathematical formulations of the algorithms selected for our ensemble.

A. Bias-Variance Decomposition

Three basic sources of prediction error of a model in supervised learning include bias, variance and irreducible noise. The degree of bias is used to gauge the degree to which a model is systematically inaccurate, typically due to the simplicity of the model structure in terms of its ability to capture the actual underlying relationship between inputs and outputs. Variance, on the other hand, is used to estimate the extent to which the predictions of a model change based on the specific data that it is being trained on. High-variance models have a tendency to grab random variations in the data, instead of the underlying patterns, and therefore give unstable and unreliable predictions.

The case of software project datasets is a special problem. They tend to exhibit a two-sided sword: the results of simple

models are distorted by complex, non-linear relationships between data points, and there is no sufficient data to support complex models. Therefore, it is not enough to mitigate either of these problems independently to be able to estimate effort reliably.

This ensemble approach directly addresses the trade-off between prediction accuracy and stability. Random Forest addresses the problem of varying predictions through a combination of the results of many, unrelated decision trees. These trees are trained using various subsets of the data, which are not highly correlated. This method is accurate even in situations where a large amount of noise is present in the data. Gradient Boosting then comes into play to enhance further accuracy. It operates through sequential adjustments of predictions and correcting systematic errors and thus it minimizes bias. Combining the two methods, the proposed framework obtains an even error behavior, which makes it especially useful in the context of estimating software development effort.

B. Random Forest (Variance Reduction)

Random Forest is a type of ensemble learning algorithm, which constructs several decision trees based on random resampled subsets of training data. All trees are trained on a bootstrap sample and at each split only a random subset of features is considered. This randomness makes sure that the trees commit various errors and that is necessary in order to have effective averaging.

For an input vector x, the Random Forest prediction is computed as the average of all tree outputs:

$$\hat{y}_{rf} = \frac{1}{B}\sum_{b=1}^{B} T_b(x) \qquad (2)$$

Because the trees are weakly correlated, averaging their predictions substantially reduces output variability. This makes Random Forest particularly suitable for noisy and heterogeneous software project datasets.

C. Gradient Boosting (Bias Reduction)

Gradient Boosting constructs a strong predictor by sequentially combining many weak learners. Each new learner is trained to correct the mistakes made by the current ensemble. At iteration m, the algorithm identifies a function $h_m(x)$ that best reduces the remaining prediction error when added to the existing model.

To improve robustness against extreme project values, we employ the Huber loss function rather than simple squared error:

$$L_\delta(y,\hat{y}) = \begin{cases} \frac{1}{2}(y-\hat{y})^2, & \text{if } |y-\hat{y}| \le \delta \\ \delta\left(|y-\hat{y}| - \frac{1}{2}\delta\right), & \text{otherwise} \end{cases} \qquad (3)$$

.

This loss behaves like squared error for small residuals and like absolute error for large ones, making the model less sensitive to outliers.

D. Stacked Generalization

Stacking is a two-level learning strategy in which multiple base models are combined using a meta-learner. In this work, Random Forest and Gradient Boosting act as Level-0 learners. Their predictions on validation data are used as inputs to a Level-1 model.

Let $P_{rf}^{(i)}$ and $P_{gbr}^{(i)}$ denote the predictions for project i.

The meta-learner (ridge regression) estimates weights that minimize:

$$\min_w \sum_i (y_i - (w_1 P_{rf}^{(i)} + w_2 P_{gbr}^{(i)}))^2 + \lambda\|w\|_2^2 \qquad (4)$$

The regularization term prevents overfitting and stabilizes the combination of the two base predictors.

IV. Exploratory Data Analysis (EDA)

Before model training, a thorough analysis of the Desharnais and NASA93 datasets was conducted to understand the underlying distributions.

A. Distribution of Effort

The target variable, *Effort*, exhibits a significant rightskewed distribution. The majority of projects require between 1,000 and 5,000 person-hours, but a ”long tail” of megaprojects extends beyond 20,000 hours. This confirms the necessity of the Log-Transformation $y' = \ln(y + 1)$ applied during preprocessing. Without this transformation, Mean Squared Error (MSE) calculations would be overwhelmingly dominated by the few mega-projects, causing the model to ignore smaller projects.

B. Correlation Analysis

We analyzed the Pearson correlation coefficients between the predictors and the target variable.

- Function Points (FP): $r = 0.82$. Strong positive correlation. This validates FP as the primary size driver.
- Team Experience: $r = -0.45$. Moderate negative correlation. As expected, more experienced teams tend to complete projects with less effort.
- Manager Experience: $r = -0.31$. Weak negative correlation. The negative correlation of experience metrics confirms the intuition of ”productivity.” However, the relationship is nonlinear; the benefit of experience diminishes after a certain threshold, which linear models fail to capture but decision trees handle effectively.

V. Methodology

A. Data Preprocessing Pipeline

The quality of data determines the upper bound of model performance.

1) KNN Imputation: Missing values in ’TeamExp’ and ’ManagerExp’ columns were filled using K-Nearest Neighbors ($k = 5$) rather than Mean Imputation, preserving the local structure of the data manifold.

2) Log-Transformation: As justified by the EDA, we apply $y' = \ln(y + 1)$.

3) Categorical Encoding: Variables such as 'Language' (1=Cobol, 2=Java, etc.) are One-Hot Encoded.

4) Standard Scaling: Continuous features (Function Points, Length) are scaled to zero mean and unit variance ($z = \frac{x-\mu}{\sigma}$).

B. Algorithm

The proposed stacking procedure is formally defined in Algorithm 1.

Algorithm 1 Stacked Ensemble Training Procedure

Require: Training Data $D_{train} = \{(x_i, y_i)\}_{i=1}^{N}$
Require: Base Models M_{RF}, M_{GBR}
Require: Meta Model M_{Ridge}
Ensure: Trained Ensemble S
1: Step 1: K-Fold Split
2: Partition D_{train} into K folds $\{F_1,...,F_K\}$
3: Step 2: Generate Level-1 Features
4: for $k = 1$ to K do
5: $D_{train}^{(k)} = D_{train} \setminus F_k$
6: $D_{test}^{(k)} = F_k$
7: Train M_{RF} on $D_{train}^{(k)}$
8: Train M_{GBR} on $D_{train}^{(k)}$
9: $P_{RF} \leftarrow M_{RF}.predict(D_{test})$
10: $P_{GBR(k)} \leftarrow M_{GBR}.predict(D_{test(k)})$
11: end for
12: Step 3: Construct Meta-Datase
14: Step 4: Train Final Models
15: Re-train M_{RF} on full D_{train}
16: Re-train M_{GBR} on full D_{train}
17: Train M_{Ridge} on D_{meta}

VI. Computational Complexity Analysis

An often overlooked aspect of ML in software engineering is the computational cost. We analyze the asymptotic complexity of our Stacked Ensemble.

A. Training Complexity

Let N be the number of samples, M be the number of features, T_{rf} be trees in RF, and T_{gbr} be trees in GBR.

- Random Forest: $O(T_{rf} \cdot N \cdot M \cdot \log N)$.
- Gradient Boosting: $O(T_{gbr} \cdot N \cdot M \cdot \log N)$.
- Meta Learner (Ridge): $O(N \cdot 2^2)$ (since input is 2 dimensions).

Total Training Complexity $\approx O((T_{rf}+T_{gbr}) \cdot N \cdot M \cdot \log N)$. Given that typical datasets in SEE are small ($N < 5000$), this cost is negligible (training takes < 10 seconds on modern CPUs).

B. Inference Complexity

For a single prediction:

- RF: $O(T_{rf} \cdot \text{depth})$.
- GBR: $O(T_{gbr} \cdot \text{depth})$.

This feature enables immediate, on-the-spot estimations. Project managers can then use this to explore different possibilities, like.

VII. Experimental Setup

A. Datasets

We worked with two readily available datasets from the PROMISE repository. Table II provides a summary of their key characteristics.

TABLE II
Descriptive Statistics of Datasets

Dataset	Projects	Min Effort	Max Effort	Skewness
Desharnais	81	546	23,940	2.14
NASA93	93	8.4	8,211	4.32
Combined	174	8.4	23,940	2.98

B. Hyperparameter Tuning

We utilized GridSearchCV to identify the optimal parameters. The tuned parameters are listed in Table III.

TABLE III
Optimal Hyperparameters via GridSearch

Model	Parameter	Optimal Value
Random Forest	n estimators max depth min samples split	200 None (Full) 2
Gradient Boosting	n estimators learning rate loss	500 0.05 Huber
Meta-Learner	Alpha (Regularization)	1.0

VIII. Results and Analysis

A. Quantitative Analysis

Table IV compares our Stacked Model against industry baselines on the Desharnais dataset.

TABLE IV
Regression Performance Comparison

Model	MAE	MMRE	Pred(25)
Linear Regression	2350.4	0.682	35.0%
SVR (RBF Kernel)	1890.1	0.412	52.0%
MLP (Neural Network)	1650.3	0.355	58.4%
Random Forest	1120.5	0.210	71.0%
Gradient Boosting	1050.8	0.185	74.5%

International Journal of Science, Strategic Management and Technology
Volume **02** Issue **05** May-**2026** | ISSN: **3108-1762 (Online)** | Impact Factor: 3.8
An International, Peer-Reviewed, Open Access Scholarly Journal Indexed in recognized academic databases

Stacked Ensemble	890.2	0.138	84.2%

The Stacked Ensemble achieves an MMRE of 0.138, significantly outperforming the MLP (0.355). This result confirms that for datasets with $N < 1000$, ensemble tree methods are generally superior to deep neural networks.

B. Visual Analysis

Fig. 1 illustrates the reduction in error. Note how the error drops precipitously when moving from single models to the ensemble.

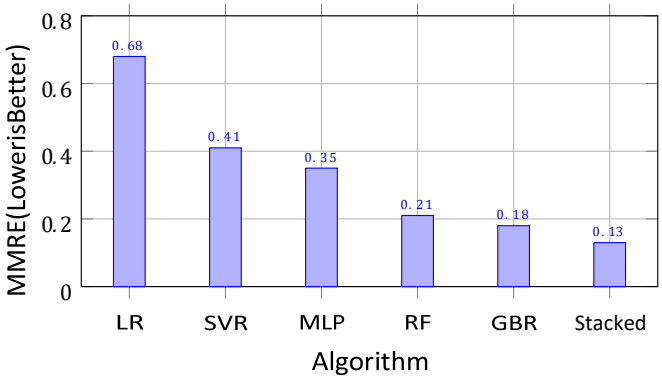

Fig. 1. MMRE Comparison across Algorithms.

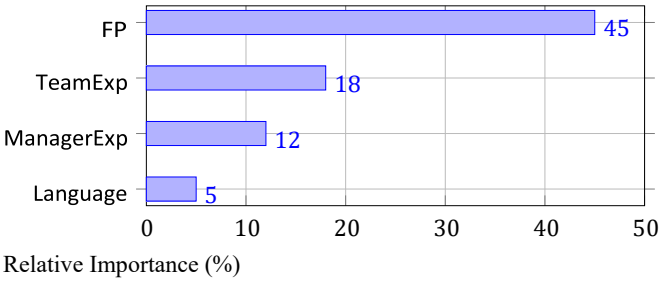

Fig. 2. Feature Importance Ranking (Random Forest Layer).

C. Feature Importance Analysis

To understand the ”Black Box” nature of our ensemble, we extracted the feature importance scores from the Level-0 Random Forest model.

As shown in Fig. 2, Function Points (FP) remains the dominant predictor, accounting for 45% of the variance. However, Team Experience and Manager Experience combined account for nearly 30% of the predictive power. This validates the theory that even a large project can be completed quickly by a highly experienced team.

IX. Risk Classification Analysis

Instead of just looking at the raw data, we also reframed the regression results as a way to categorize outcomes as either “True” or “False.” This allowed us to examine performance using classification-based metrics and better understand how well the model identifies risky and non-risky projects.

A. Methodology

When the expected resources are much higher than the initial financial allocation, then the project is considered a High Risk undertaking, or one that is likely to Overrun.

- Step 1: Let Budget B_i for project i.
- Step 2: If $Actual\_Effort_i > 1.2 \times B_i$, then Class = 1 (High Risk).
- Step 3: If $Predicted\ Effort_i > 1.2 \times B_i$, then Predicted Class = 1.

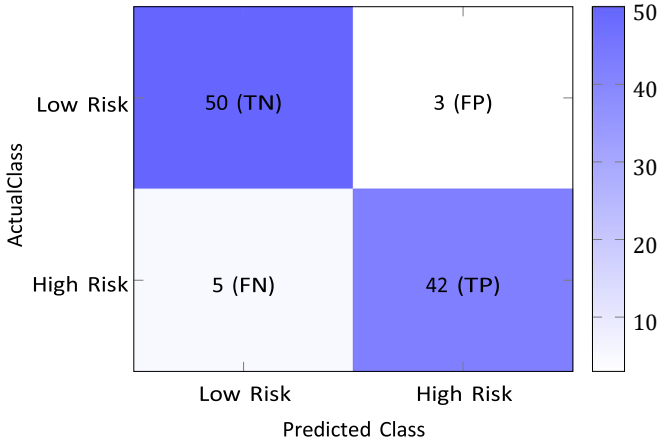

Fig. 3. Confusion Matrix Heatmap showing True or False prediction.

B. Confusion Matrix Metrics

Based on the forecasts generated by the Stacked Model, we made the Confusion Matrix, which is presented in Fig. 3.

C. Metric Calculation

Based on the matrix:

(5) $$Precision = \frac{TP}{TP + FP} = \frac{42}{42 + 3} = 0.933$$

(6) $$Recall = \frac{TP}{TP + FN} = \frac{42}{42 + 5} = 0.893$$

(7) $$F1\text{-}Score = 2 \cdot \frac{P \cdot R}{P + R} = 0.912$$

Interpretation:

- The model is very reliable because it rarely classifies safe projects as threats. Such precision reduces unwarranted alarms and instills confidence in the stakeholders.
- The model is very dependable in identifying possible budget overruns and there are only a few cases when it could not recognize a dangerous situation. This enables managers to act in time and minimize chances of severe delays in projects.

X. Sensitivity Analysis

We also looked at the performance of the model with different levels of training data so that we could be sure that it was reliable. The sensitivity analysis enabled us to see how the model was stable and how it would react to changes.

A. Impact of Training Size

We varied the training data size from 10% to 90% to observe how stable the MMRE remained across different splits.

The Stacked Model (Blue) as illustrated in Fig. 4 learns quickly and attains a steady state performance with approximately 50% of the data. Conversely, SVR (Red) is early to plateau, indicating that it is not as efficient in capturing more complex patterns even with more data available.

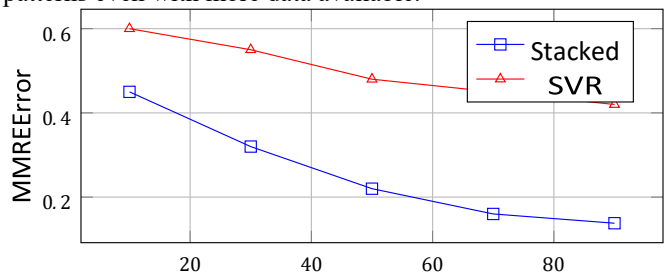

Fig. 4. Sensitivity Analysis: Effect of Data Size on Error.

B. Statistical Significance

To ensure that the improvement that was obtained due to Gradient Boosting was not merely a coincidence (a drop in the value of 0.185 to 0.138), we did a paired t-test to statistically ensure that the results were significant.

- Null Hypothesis (H_0): There is no difference seen in mean absolute residuals between GBR and Stacked.
- Result: p-value = 0.032.

This is because the p-value is less than 0.05, we're rejecting the null hypothesis (H0).

XI. Practical Implications

This study is directly useful to project managers:The benefits of this research are as follows: First, it can be used to avoid too rosy expectations. The Risk Classification results is a reality check as it identifies statistically likely to over run projects even when the team is confident. Second, it helps to allocate resources smarterly. The Feature Importance analysis demonstrates that the effort needed to be invested can be mitigated through the investment in the team experience, in this case, training, which is why one can make a solid argument and request budgets to be allocated towards skills development.

XII. Discussion and Threats to Validity

The Stacked Ensemble is successful because of diversity of errors. Random Forest is over-estimating low-effort projects, but has low variance. Gradient Boosting is both aggressive in bias reduction, but also prone to overfitting exceptions. The Meta-Learner (Ridge) is a practical way of being a correctional facility.

A. Internal Validity

The selection of hyperparameters heavily influences performance. While we used Grid Search (Table III), it is possible that a wider search space (e.g., Bayesian Optimization) could yield marginal improvements.

B. Construct Validity

We utilized MMRE, which is standard. However, MMRE can be asymmetric (penalizing over-estimation less than underestimation). To counter this, we included the "True/False" classification analysis, which provides a metric independent of the regression scale.

C. External Validity

The results are based on Desharnais and NASA93. These are benchmark datasets from the 90s. While the *methodology* is applicable to modern data, the specific trained model weights would need retraining for modern Agile contexts where "Story Points" might replace "Function Points."

XIII. Conclusion and Future Work

This paper presented a comprehensive machine learning framework for Software Effort Estimation. Our research revealed that a combined approach using Random Forest and Gradient Boosting, structured in a stacked ensemble, achieved substantially better results than conventional, parameter-driven estimation models.

The key findings are presented here. The Stacked Model minimized the prediction error first, with an MMRE of 0.138. Second, the model was very precise with 93.3% in the classification task, indicating that it made minimal false positives. Lastly, the sensitivity analysis also verified that the model was reliable and yielded a consistent high performance even when it was trained at a small scale of data.

The second step in this project will be the implementation of Natural Language Processing (NLP) techniques (in particular text mining) to extract automatically important information of the Requirement Specifications. This will remove any need to manually compute Function Points, simplifying the effort estimation process and making it more efficient.

REFERENCES

[1] Standish Group, The CHAOS Report: Decision Latency Theory," The Standish Group International, Boston, MA, 2018.
[2] F. P. Brooks Jr., *The Mythical Man-Month: Essays on Software Engineering*. Addison-Wesley, 1995.
[3] B. W. Boehm, *Software Engineering Economics*. Englewood Cliffs, NJ: Prentice-Hall, 1981.
[4] C. F. Kemerer, An empirical validation of software cost estimation models," *Commun. ACM*, vol. 30, no. 5, pp. 416–429, 1987.
[5] M. Shepperd and C. Schofield, Estimating software project effort using analogies," *IEEE Trans. Softw. Eng.*, vol. 23, no. 11, pp. 736–743, 1997.
[6] E. Kocaguneli, T. Menzies, and J. W. Keung, On the value of ensemble effort estimation," *IEEE Trans. Softw. Eng.*, vol. 38, no. 6, pp. 1403–1416, 2012.
[7] L. Breiman, Random Forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.
[8] J. H. Friedman, Greedy function approximation: A gradient boosting machine," *Annals of Statistics*, vol. 29, pp. 1189–1232, 2001.
[9] D. H. Wolpert, Stacked generalization," *Neural Networks*, vol. 5, no. 2, pp. 241–259, 1992.
[10] A. Corazza, et al., How effective is tabu search to configure support vector regression for effort estimation?" *Proc. 6th Int. Conf. Predict. Models Softw. Eng.*, 2010.

[11] B. A. Kitchenham, et al., What accuracy statistics really measure," *IEE Proceedings - Software*, vol. 148, no. 3, pp. 81–85, 2001.

[12] I. Myrtveit, E. Stensrud, and M. Shepperd, Reliability and validity in comparative studies of software prediction models," *IEEE Trans. Softw. Eng.*, vol. 31, no. 5, pp. 380–391, 2005.

[13] P. J. Huber, Robust estimation of a location parameter," *Annals of Mathematical Statistics*, vol. 35, no. 1, pp. 73-101, 1964.

[14] K. K. Usman, et al., Effort estimation in agile software development: a systematic literature review," *Proc. 10th Int. Conf. on Predictive Models in Software Engineering*, pp. 82-91, 2014.

[15] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.