A Comparative Study of the UYSAL Software Effort Model Versus NASA-Based and Classical Effort Estimation Models

Mitat Uysal

Dogus University, Istanbul, Turkey

Abstract: Accurate software effort estimation remains a central problem in software engineering because it directly affects budgeting, staffing, scheduling, and project risk. This paper presents a structured comparison of the UYSAL software effort modeling approach with NASA-based estimation practices and several widely cited classical parametric models (e.g., COCOMO, Walston–Felix, Bailey–Basili, Doty), along with brief positioning against later-generation models (e.g., COCOMO II, SLIM/Putnam, Function Point based approaches). The paper first explains the UYSAL approach in detail—covering (i) multivariate interpolation (double Lagrange) for building an effort surface and (ii) COCOMO-inspired parametric equations whose parameters are calibrated via heuristic optimization on NASA project data. Then, it contrasts model assumptions, inputs, calibration behavior, interpretability, and practical use in NASA-like environments.

Keywords: Software effort estimation, NASA datasets, COCOMO, heuristic optimization, genetic algorithms, parametric cost models

1. INTRODUCTION

Software effort estimation remains one of the central problems in software engineering because it directly influences project planning, budgeting, staffing decisions, scheduling, and risk management. Accurate predictions of development effort are essential for organizations that manage large and complex software systems. However, despite decades of research, effort estimation remains challenging due to differences in development environments, project characteristics, and measurement quality.

NASA and NASA-affiliated research environments have played a major role in the empirical study of software engineering processes. Through initiatives such as the Software Engineering Laboratory (SEL), NASA collected detailed project measurements in order to understand how development practices, methodologies, and project size influence the total development effort. These datasets later became important benchmarks for evaluating software cost estimation models and have been widely used in comparative studies of effort estimation techniques.

Traditional parametric estimation models typically express effort as a function of project size, often measured in thousands of lines of code (KLOC). Well-known examples include classical empirical models such as Walston–Felix, Bailey–Basili, and Doty, as well as the widely used

COCOMO model family. While these models provide interpretable relationships between project size and development effort, many of them rely primarily on size as the main predictor and therefore may not fully capture variations caused by development methodology, tool support, or process maturity [1-4].

Within this context, the UYSAL software effort modeling approach introduces a framework that extends traditional size-based estimation models in two important ways. First, it constructs a multivariate effort surface using double Lagrange interpolation, allowing effort to be represented as a nonlinear function of both project size and development methodology. Second, it introduces COCOMO-inspired parametric equations whose parameters are calibrated through heuristic optimization techniques, such as simulated annealing, using empirical project datasets derived from NASA-related repositories [5].

By incorporating both size and methodology variables and by allowing model parameters to be calibrated using optimization techniques, the UYSAL approach provides a more flexible modeling framework than classical single-variable estimation models. This enables the model to adapt to different datasets and project environments while maintaining a clear mathematical structure that supports interpretation and analysis.

The objective of this paper is to present a structured comparison between the UYSAL software effort modeling approach and several well-known effort estimation models used in NASA-related studies. In particular, the paper analyzes differences in model structure, input variables, calibration strategies, and interpretability. The comparison also highlights how the inclusion of methodology variables and heuristic calibration techniques can influence estimation accuracy when applied to NASA software project datasets.

2. NASA CONTEXT AND EFFORT MODEL FAMİLIES

2.1 NASA measurement culture and datasets

NASA has historically played an important role in the empirical study of software engineering processes. Through the Software Engineering Laboratory (SEL) established at NASA’s Goddard Space Flight Center, systematic measurements of software development projects were collected in order to analyze the relationships between development processes, tools, methodologies, and project outcomes. The primary objective of these studies was to improve software quality, reduce development risks, and better understand how development effort scales with project characteristics [6].

Over time, several NASA-related datasets became widely used benchmarks in the field of software effort estimation. Among the most commonly referenced datasets is the NASA93 dataset, which contains information on software projects including size, development characteristics, and recorded development effort. These datasets have been redistributed through repositories such as the PROMISE and tera-PROMISE repositories, allowing researchers to evaluate and compare software estimation models using common benchmark data [7–9].

Because these datasets contain projects with varying development environments, methodologies, and sizes, they provide a useful experimental platform for evaluating both classical parametric models and modern data-driven estimation techniques.

2.2 Classical parametric models (size-only)

Early software effort estimation models were primarily based on empirical observations relating software size to development effort. Many of these models use a power-law relationship of the general form:

$$\text{Effort} = A \times (Size)^{B}$$

where *Size* is typically measured in thousands of lines of code (KLOC), and A and B are empirically derived constants.

Several classical estimation models follow this structure and are often used as baseline comparisons in software engineering studies.

One of the well-known models is the **Walston–Felix model**, which expresses development effort as:

$$\text{Effort} = 5.2 \times (\text{KLOC})^{0.91}$$

Another widely cited model is the **Bailey–Basili model**, which includes a constant term in addition to the power-law component:

$$\text{Effort} = 5.5 + 0.73 \times (\text{KLOC})^{1.16}$$

Similarly, the **Doty model**, which is often applied to larger software systems, takes the form:

$$\text{Effort} = 5.288 \times (\text{KLOC})^{1.047}$$

These models are attractive because they are simple, interpretable, and easy to apply during early stages of software project planning. However, their main limitation is that they rely almost entirely on software size as the predictive variable. As a result, variations caused by development methodology, tool support, team experience, or process maturity are not explicitly represented within the model structure.

2.3 COCOMO and NASA usage

The Constructive Cost Model (COCOMO) introduced by Boehm is one of the most influential models in software cost estimation. The original formulation of COCOMO was derived through regression analysis of empirical project data and follows a similar power-law relationship between software size and development effort [10].

The basic form of the COCOMO model can be written as

$$\text{Effort} = A \times (\text{KLOC})^{B}$$

where the coefficients A and B depend on the type of software project and the development environment.

Later developments of the model introduced additional parameters such as cost drivers and scale factors, allowing the model to incorporate factors related to personnel capability, development tools, product complexity, and development environment. These extensions resulted in the COCOMO II model, which is widely used in both industry and academic research.

Because NASA datasets contain detailed information on software projects, they have frequently been used to evaluate and recalibrate COCOMO-style models. In many studies, the model parameters are tuned using optimization techniques in order to better match the characteristics of specific datasets or development environments [5].

2.4 Later-generation models (brief positioning)

Beyond classical parametric models and COCOMO, several other estimation frameworks have been proposed in software engineering research.

One such approach is the Putnam SLIM model, which describes the relationship between development effort, project duration, and software size through a mathematical "software equation" derived from empirical observations of software development processes [11].

Another widely used approach is Function Point Analysis (FPA), which estimates software size based on functional characteristics rather than lines of code. This method was introduced by Albrecht and is particularly useful in environments where code size is difficult to estimate during early stages of project planning [12].

More recent studies have also explored machine learning and nonlinear regression techniques, such as neural networks, fuzzy systems, and radial basis function models, for estimating software development effort using historical project datasets [13],[14].

These developments highlight the ongoing effort within the software engineering community to improve estimation accuracy by incorporating additional project characteristics and by applying more advanced modeling techniques.

3. THE UYSAL SOFTWARE EFFORT MODEL IN DETAIL

The UYSAL software effort modeling framework can be viewed as a two–stage modeling approach that combines a data-driven interpolation surface with parametric estimation equations calibrated through heuristic optimization [15],[16]. The goal of this approach is to represent software development effort as a function of both project size and development methodology.

The modeling framework therefore consists of two complementary components:

1. *A multivariate interpolation model* that constructs a continuous effort surface from empirical project data.
2. *COCOMO-inspired parametric equations* that explicitly incorporate both project size and methodology variables, whose parameters are calibrated through heuristic search techniques on NASA software project datasets.

This hybrid structure allows the model to capture nonlinear relationships between project characteristics and development effort while maintaining interpretability and mathematical structure.

3.1 UYSAL multivariate interpolation model (effort surface)

In the first component of the approach, software effort is represented as a nonlinear function of two variables:

- DL: Developed Lines of code (software size)
- ME: Methodology indicator (representing development process maturity)

Thus, the effort function can be expressed as

$$E = f(DL, ME)$$

where

- E represents the development effort
- DL represents the software size
- ME represents the methodology variable.

Instead of assuming a fixed functional form, the model constructs the effort function directly from observed data using multivariate interpolation.

3.1.1 Two-dimensional (double) Lagrange interpolation

To construct the effort surface, the UYSAL model uses *two-dimensional Lagrange interpolation* on a rectangular grid of empirical data points.

Suppose effort values are known at grid points

$$(DL_i, ME_j)$$

with corresponding effort values E_{ij}.
Then the interpolated effort function can be written as

$$E(DL, ME) = \sum_{i=0}^{n} \sum_{j=0}^{m} E_{ij}\, L_i(DL) M_j(ME)$$

where

- $L_i(DL)$are the Lagrange basis polynomials in the size dimension,
- $M_j(ME)$are the Lagrange basis polynomials in the methodology dimension.

The one-dimensional Lagrange basis polynomial is defined as

$$L_i(DL) = \prod_{k \neq i} \frac{DL - DL_k}{DL_i - DL_k}$$

and similarly

$$M_j(ME) = \prod_{k \neq j} \frac{ME - ME_k}{ME_j - ME_k}$$

A key property of the Lagrange basis functions is that

$$L_i(DL_k) = \begin{cases} 1 & k = i \\ 0 & k \neq i \end{cases}$$

which guarantees that the interpolated surface passes exactly through the observed data points.

3.1.2 Practical meaning

The resulting interpolation function defines a *continuous effort surface* in the two-dimensional space formed by project size and methodology level.

From an empirical perspective, typical observations include:

- Effort generally increases with project size (DL).
- Improved development methodologies (higher ME values) may reduce the required development effort, reflecting better productivity.

Because the interpolation model is derived directly from empirical data, it provides a flexible representation capable of capturing nonlinear relationships between software size, methodology, and development effort.
However, interpolation models can also be sensitive to:

- sparse or unevenly distributed datasets,
- measurement noise in the input variables,
- extrapolation outside the observed data range.

For this reason, the interpolation model is complemented by a parametric modeling approach, which is described in the following section.

3.2 UYSAL COCOMO-inspired parametric models with methodology factor

While the interpolation model provides a flexible representation of the effort surface, a parametric formulation is also desirable for practical estimation and interpretation. For this purpose, the UYSAL approach introduces COCOMO-inspired parametric equations that incorporate both project size and methodology variables.

Let

- DL: developed lines of code (software size)
- ME: methodology indicator
- E: development effort

The effort can then be modeled as a function

$$E = f(DL, ME)$$

where the relationship is approximated by a parametric equation whose coefficients are determined from empirical data.

Two candidate parametric models are proposed.

3.2.1 Model 1 (five-parameter model)

The first model extends the classical COCOMO structure by including a term representing the development methodology. The effort equation is written as

$$\mathrm{E} = \mathrm{a}(\mathrm{DL})^{\mathrm{b}} + \mathrm{c}(\mathrm{ME})^{\mathrm{d}} + \mathrm{e}$$

where

- aand bcontrol the contribution of software size,
- cand drepresent the methodology effect,
- eis a constant offset.

This structure preserves the familiar power-law relationship used in traditional effort estimation models while allowing development methodology to influence the estimated effort.

Compared with classical size-only models, this formulation allows the model to represent situations where improved development processes or tools reduce development effort.

3.2.2 Model 2 (seven-parameter model)

To capture more complex nonlinear relationships, a second parametric model with additional logarithmic terms is introduced. The model can be expressed as

$$E = a(DL)^b + c(ME)^d + f\ln(DL) + g\ln(ME) + h$$

where

- a, bdescribe the power-law effect of software size,
- c, ddescribe the power-law effect of methodology,
- f, gcapture additional logarithmic nonlinearities,
- $h$is a constant offset term.

The inclusion of logarithmic terms allows the model to represent situations where the marginal effect of size or methodology decreases as the project grows larger. Such nonlinear effects are frequently observed in empirical software engineering datasets.
Because this model contains more parameters than Model 1, it provides greater flexibility but also requires careful calibration to avoid overfitting when datasets are small.

3.3 Parameter calibration as an optimization problem

In the UYSAL framework, the parameters of the parametric models are not fixed in advance. Instead, they are determined by fitting the model to empirical project data.

Let

- E_i^{obs}denote the observed effort for project i
- E_i^{pred}denote the model-predicted effort

The calibration problem is formulated as minimizing the sum of squared errors (SSE):

$$SSE = \sum_{i=1}^{N} (E_i^{obs} - E_i^{pred})^2$$

where $N$is the number of projects in the dataset.

In the UYSAL approach, heuristic optimization techniques such as simulated annealing are used to search for parameter values that minimize this objective function. This strategy allows the model parameters to adapt to the statistical characteristics of the dataset, particularly when applied to NASA software project datasets.

Unlike traditional estimation models that use fixed coefficients derived from earlier datasets, this optimization-based calibration makes the model dataset-adaptive. As a result, the estimation accuracy can improve when the model is applied to environments that differ from the datasets originally used to derive classical estimation equations.

4. NASA-BASED AND OTHER MODELS FOR COMPARISON

To evaluate the effectiveness of the UYSAL software effort estimation framework, it is useful to compare its predictions with several classical and NASA-related estimation models that have been widely used in software engineering research. These models provide baseline references and help illustrate how different modeling assumptions influence estimation accuracy.

Most classical software effort estimation models are based on empirical relationships between software size and development effort. In many cases, these relationships are expressed using power-law equations in which effort grows as a nonlinear function of the number of lines of code.

4.1 Classical parametric baseline models

The classical parametric software effort estimation models discussed earlier in Section 2 provide important baseline references for evaluating new estimation approaches. These models, including the Walston–Felix, Bailey–Basili, and Doty models, express development effort primarily as a power-law function of software size measured in KLOC.

Because these models rely mainly on project size as the predictor variable, they provide a useful benchmark for comparing estimation methods that incorporate additional variables such as development methodology. In the present study, these classical models are therefore used as baseline comparators for evaluating the predictive behavior of the proposed UYSAL models.

4.2 COCOMO model

The Constructive Cost Model (COCOMO) introduced by Boehm remains one of the most influential parametric models in software cost estimation. The basic form of the model can be written as

$$E = A(KLOC)^B$$

where the coefficients $A$and Bdepend on the development mode and project characteristics.

Later extensions of the model, particularly COCOMO II, introduced additional cost drivers and scale factors in order to account for various environmental and project-related factors such as product complexity, personnel capability, and development tools.

Because NASA software datasets contain detailed project information, they have frequently been used for evaluating and recalibrating COCOMO-style models using optimization techniques such as genetic algorithms or other heuristic search methods [13].

4.3 Data-driven models and nonlinear estimation approaches

In addition to classical parametric models, several data-driven estimation techniques have been proposed in the literature. These include machine learning and nonlinear regression approaches such as neural networks, fuzzy logic models, and Radial Basis Function (RBF) models [14].

For example, Shin and Goel proposed the use of RBF models for empirical software effort prediction, demonstrating that nonlinear data-driven models can capture complex relationships between project characteristics and development effort.

These approaches highlight the growing interest in combining empirical datasets with advanced modeling techniques to improve estimation accuracy [17-21].

5. COMPARISON: UYSAL VS NASA AND OTHER MODELS

To evaluate the performance and conceptual advantages of the proposed UYSAL software effort estimation framework, it is useful to compare it with classical estimation models and NASA-based approaches. These models differ in their underlying assumptions, input variables, and calibration strategies.

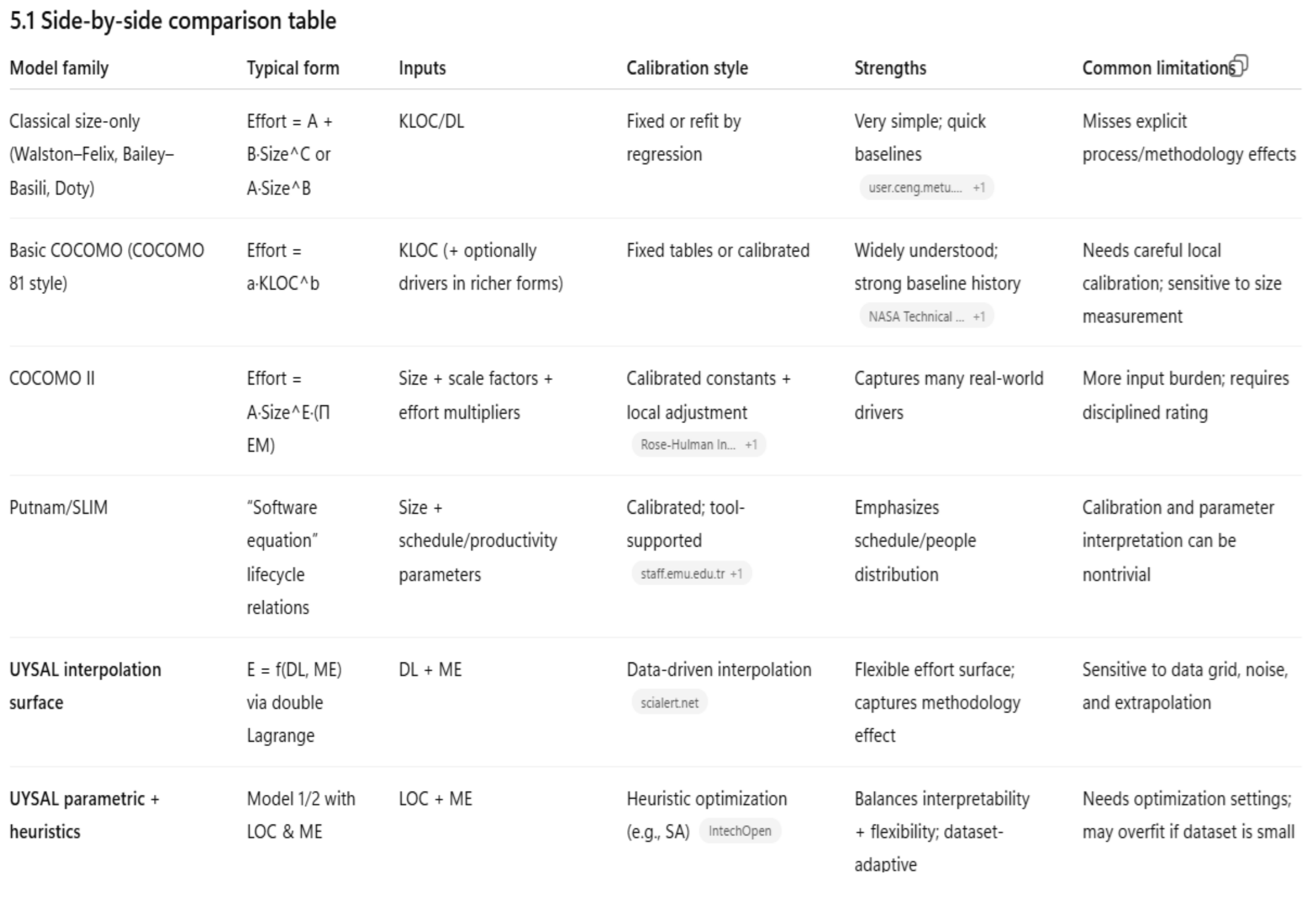
Classical estimation models such as Walston–Felix, Bailey–Basili, and Doty rely primarily on software size (KLOC) as the main predictor of development effort. While these models provide simple and interpretable relationships, they do not explicitly incorporate variables related to development methodology or process maturity. As a result, their predictive performance may vary when applied to datasets that contain projects developed under different methodological conditions.

In contrast, the UYSAL modeling approach introduces an additional explanatory variable representing development methodology (ME). By combining this variable with software size (DL), the model is able to represent a broader range of productivity variations that may occur across different development environments.

5.1 Comparative model characteristics

To clarify the conceptual differences between the proposed UYSAL framework and widely used software effort estimation models, a side-by-side comparison is presented in Table 1. The table summarizes the typical mathematical form, input variables, calibration strategies, strengths, and common limitations of the major model families used in software effort estimation studies.

Table 1 – Comparative characteristics of software effort estimation models



5.1 Side-by-side comparison table

Model family	Typical form	Inputs	Calibration style	Strengths	Common limitations
Classical size-only (Walston–Felix, Bailey–Basili, Doty)	Effort = A + B·Size^C or A·Size^B	KLOC/DL	Fixed or refit by regression	Very simple; quick baselines user.ceng.metu.... +1	Misses explicit process/methodology effects
Basic COCOMO (COCOMO 81 style)	Effort = a·KLOC^b	KLOC (+ optionally drivers in richer forms)	Fixed tables or calibrated	Widely understood; strong baseline history NASA Technical ... +1	Needs careful local calibration; sensitive to size measurement
COCOMO II	Effort = A·Size^E·(Π EM)	Size + scale factors + effort multipliers	Calibrated constants + local adjustment Rose-Hulman In... +1	Captures many real-world drivers	More input burden; requires disciplined rating
Putnam/SLIM	"Software equation" lifecycle relations	Size + schedule/productivity parameters	Calibrated; tool-supported staff.emu.edu.tr +1	Emphasizes schedule/people distribution	Calibration and parameter interpretation can be nontrivial
UYSAL interpolation surface	E = f(DL, ME) via double Lagrange	DL + ME	Data-driven interpolation scialert.net	Flexible effort surface; captures methodology effect	Sensitive to data grid, noise, and extrapolation
UYSAL parametric + heuristics	Model 1/2 with LOC & ME	LOC + ME	Heuristic optimization (e.g., SA) IntechOpen	Balances interpretability + flexibility; dataset-adaptive	Needs optimization settings; may overfit if dataset is small

The comparison highlights an important structural difference between classical estimation models and the proposed UYSAL framework. Traditional models typically rely on software size as the primary predictor of development effort. In contrast, the UYSAL approach incorporates both software size (DL) and methodology (ME), allowing the model to represent productivity differences arising from development processes and development environments.

Furthermore, the use of heuristic optimization techniques for parameter calibration enables the UYSAL models to adapt to the statistical characteristics of the target dataset. This flexibility may improve prediction accuracy when the model is applied to datasets such as the NASA software project collections.

5.2 Visual comparison and discussion using NASA dataset

To further illustrate the differences between the considered effort estimation models, graphical comparisons are performed using the NASA software project dataset. These visual analyses provide an intuitive understanding of how different models behave when predicting software development effort under varying project conditions.

Graphical evaluation is particularly useful because it allows researchers to observe estimation trends, deviations, and prediction consistency across projects. In this study, two visual comparisons are presented. The first focuses on estimation accuracy using the Mean Absolute Relative Error (MAR), while the second evaluates prediction reliability using the Pred(0.25) metric.

Figure 1 presents a comparison of the MAR values obtained by different effort estimation models applied to the NASA dataset. Lower MAR values indicate better prediction accuracy. The figure illustrates how the proposed UYSAL models perform relative to classical size-based estimation models.

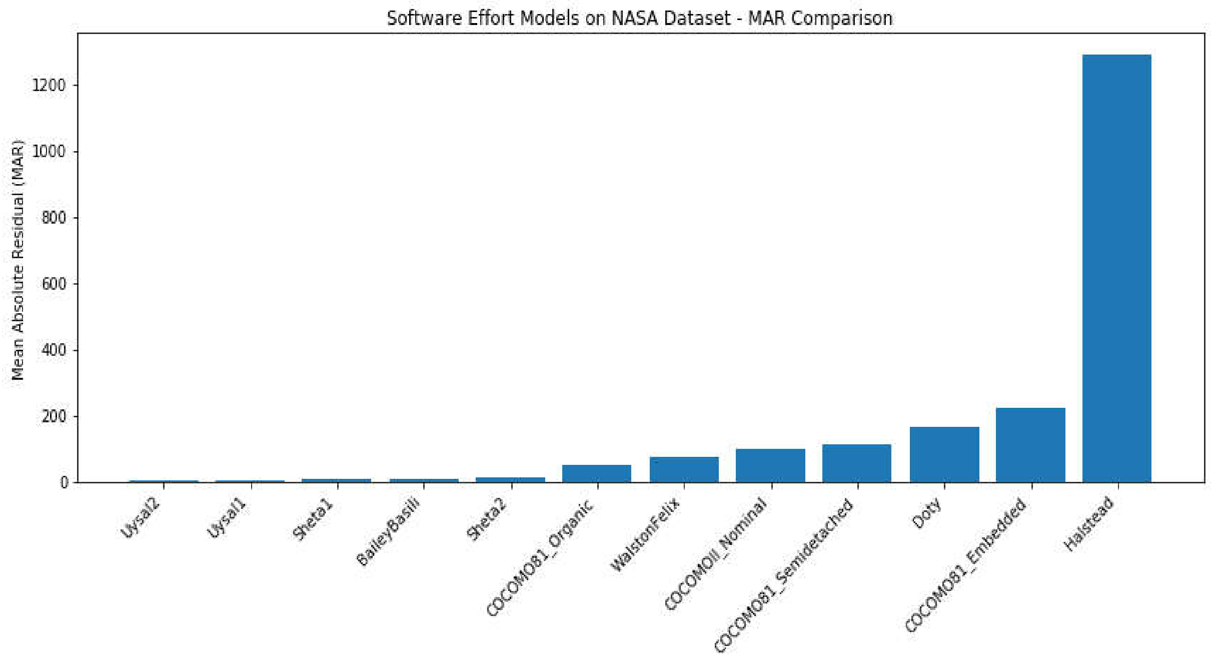


Figure-1-Software Effort Models on NASA Dataset-MAR Comparison

Figure 2 shows the comparison based on the Pred(0.25) metric, which measures the proportion of predictions whose relative error is within 25% of the actual observed effort. Higher Pred(0.25) values indicate better estimation reliability.

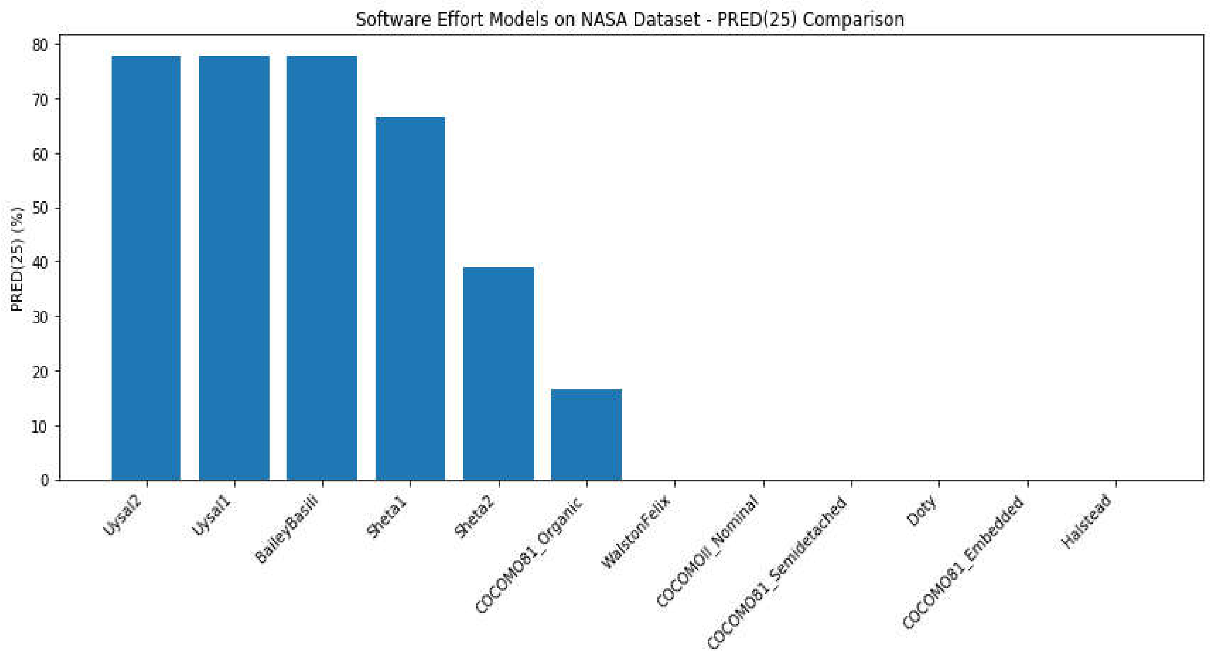


Figure-2-Software Effort Models on NASA Dataset-Pred(25) Comparison

The results suggest that the UYSAL models provide competitive performance compared with classical estimation models. By incorporating both software size (DL) and methodology (ME), the UYSAL framework is able to capture productivity differences that are not explicitly represented in size-only models. Furthermore, the use of heuristic optimization for parameter calibration enables the model to adapt to the statistical characteristics of the NASA dataset.

6. EVALUATION PRACTICE ON NASA DATASETS

To evaluate the effectiveness of the proposed models, experiments were conducted using the NASA software project dataset. The dataset contains software projects with recorded values of development size and actual development effort, allowing the comparison of predicted and observed effort values.

Several commonly used evaluation metrics are employed to assess the performance of the estimation models, including Mean Absolute Error (MAE), Root Mean Square Error (RMSE), and Pred(0.25). These metrics provide insight into both prediction accuracy and reliability.

Figure 3 illustrates the relationship between the actual effort values and the effort values predicted by the estimation models. A good estimation model should produce predicted values that closely follow the diagonal trend, indicating strong agreement between predicted and observed effort.

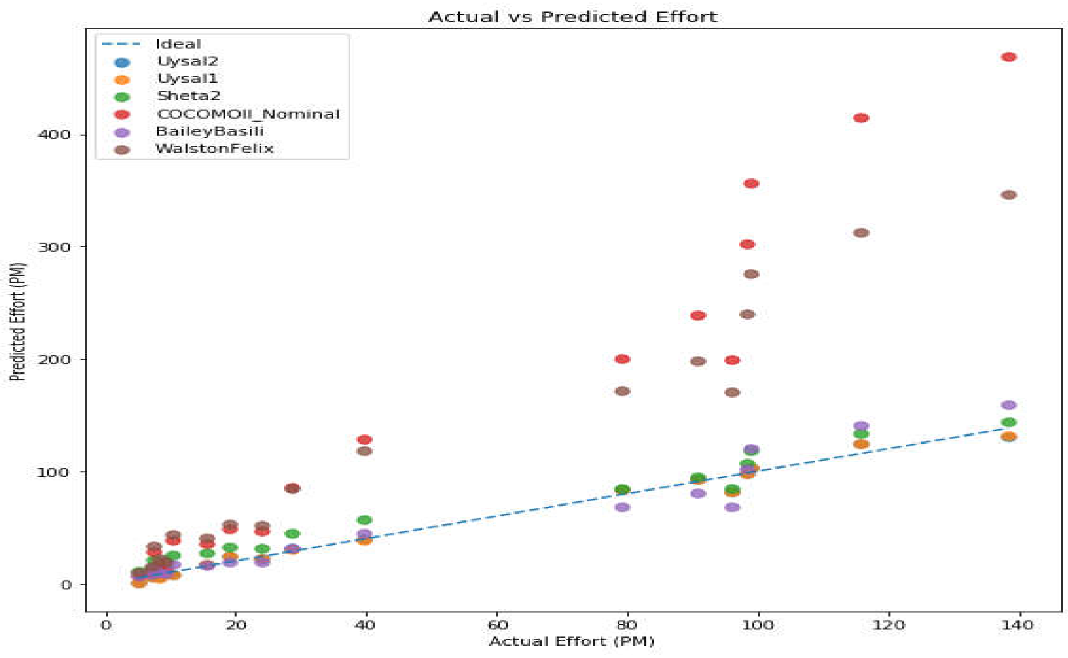


Fig.3 Actual vs Predicted Effort

Figure 4 presents the absolute error distribution across projects for the different models. This figure helps visualize how prediction errors vary from one project to another and highlights the relative stability of the estimation approaches.

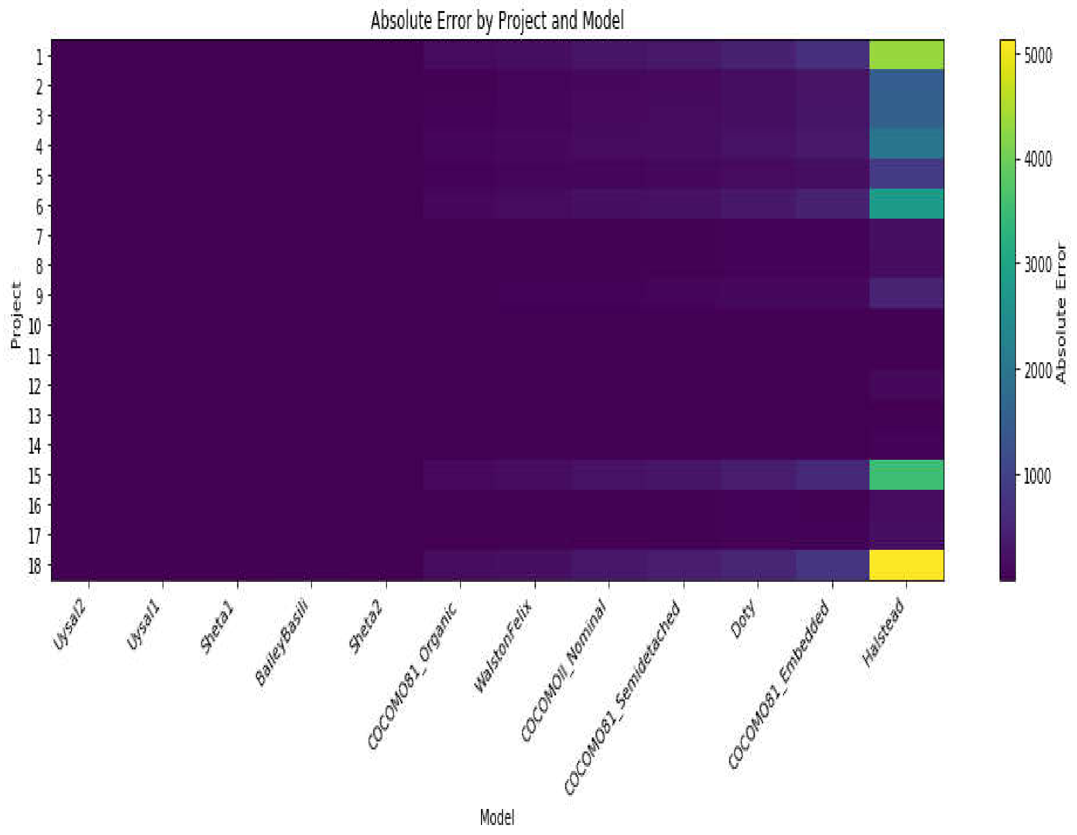
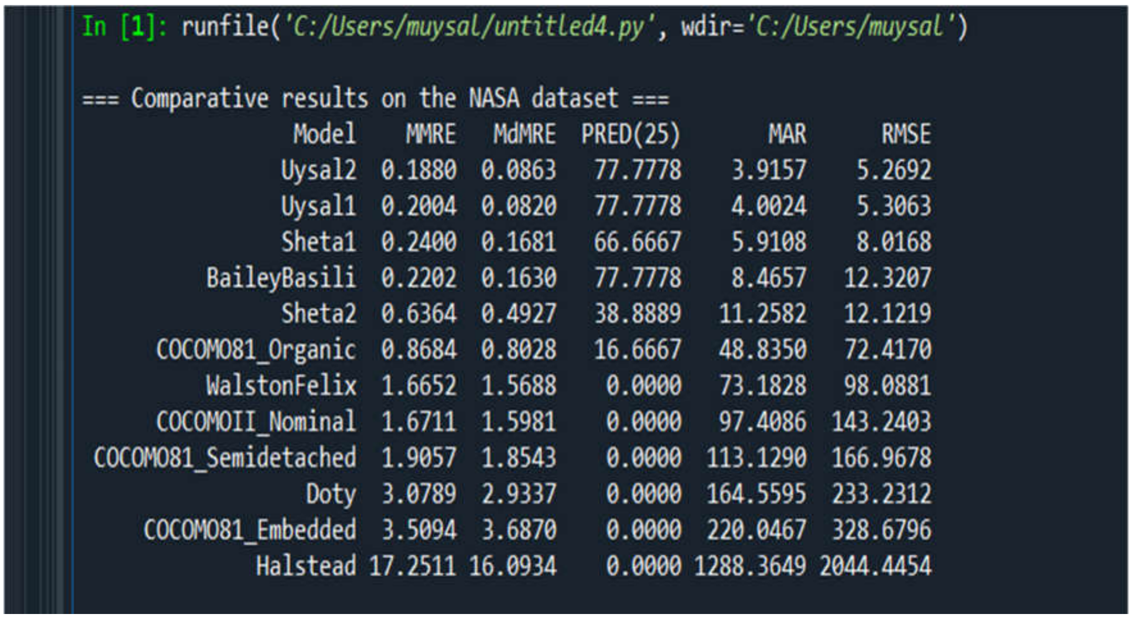


Fig.4 Absolute Error by Project and Model

Finally, Table 2 summarizes the quantitative results obtained from the evaluation metrics. The table provides a numerical comparison of the estimation accuracy of the different models when applied to the NASA dataset.

Overall, the results demonstrate that the proposed UYSAL models provide competitive estimation accuracy and are capable of capturing important variations in software development effort when both project size and development methodology are considered.

TABLE-2 Comparative Results on the NASA dataset

○ 

```
In [1]: runfile('C:/Users/muysal/untitled4.py', wdir='C:/Users/muysal')

=== Comparative results on the NASA dataset ===
                 Model    MMRE   MdMRE  PRED(25)       MAR      RMSE
                Uysal2  0.1880  0.0863   77.7778    3.9157    5.2692
                Uysal1  0.2004  0.0820   77.7778    4.0024    5.3063
                Sheta1  0.2400  0.1681   66.6667    5.9108    8.0168
           BaileyBasili  0.2202  0.1630   77.7778    8.4657   12.3207
                Sheta2  0.6364  0.4927   38.8889   11.2582   12.1219
       COCOMO81_Organic  0.8684  0.8028   16.6667   48.8350   72.4170
           WalstonFelix  1.6652  1.5688    0.0000   73.1828   98.0881
       COCOMOII_Nominal  1.6711  1.5981    0.0000   97.4086  143.2403
  COCOMO81_Semidetached  1.9057  1.8543    0.0000  113.1290  166.9678
                   Doty  3.0789  2.9337    0.0000  164.5595  233.2312
      COCOMO81_Embedded  3.5094  3.6870    0.0000  220.0467  328.6796
               Halstead 17.2511 16.0934    0.0000 1288.3649 2044.4454
```

7. DISCUSSION

The comparison presented in this study highlights the differences between classical software effort estimation models and the proposed UYSAL modeling framework. Traditional models generally rely on software size as the main predictor of development effort. While such models are simple and widely used, they may not fully capture productivity variations that arise from differences in development methodology or project environments.

The UYSAL approach extends classical estimation models by incorporating both software size and methodology variables. This multivariate representation allows the model to describe variations in development effort more flexibly. In addition, the use of heuristic optimization for parameter calibration enables the model to adapt to the characteristics of the dataset being analyzed.

However, the effectiveness of the model also depends on the quality and consistency of the input variables. In particular, the methodology indicator should be defined carefully in order to avoid introducing noise into the estimation process. When reliable measurements are available, the proposed framework can provide useful insights into the relationship between project size, development methodology, and software effort.

8. CONCLUSION

This paper presented a comparative analysis of the UYSAL software effort modeling approach with classical effort estimation models and NASA-based estimation practices. The study examined both conceptual differences and empirical behavior when these models are applied to software project datasets.

The UYSAL framework extends traditional effort estimation approaches in two important ways. First, it introduces a multivariate representation of development effort using interpolation techniques, allowing effort to be modeled as a function of both software size and development methodology. Second, it proposes COCOMO-inspired parametric models whose parameters can be calibrated using heuristic optimization methods.

Experimental comparisons conducted using the NASA93 software project dataset demonstrate that the proposed models provide competitive estimation performance. By incorporating both size and methodology factors, the UYSAL framework offers a flexible alternative to classical size-only estimation models.

Future research may investigate the application of the proposed framework to additional datasets and explore the integration of further project characteristics that influence software development effort.

REFERENCES

[1] Software Engineering Laboratory (SEL), NASA/GSFC—SEL organizational description and goals, *NASA Technical Reports Server*, 1981.
[2] B. Basili, “Models and Metrics for Software Management and Engineering,” 1980.
[3] R. Pressman, *Software Engineering: A Practitioner’s Approach*, 1992.
[4] M.Uysal,”Using Heuristic Search Algorithms for Predicting the Effort of Software Projects, *Applied and Computational Mathematics* ,Vol. 8, No. 2, pp. 251-262,2009.
[5] A. Sheta, “Estimation of the COCOMO Model Parameters Using Genetic Algorithms for NASA Software Projects,” *Journal of Computer Science*, vol. 2, no. 2, pp. 118–123, 2006.
[6] B. Boehm, “Cost Models for Future Software Life Cycle Processes (COCOMO 2.0),” 1995.
[7] tera-PROMISE / OpenScience, “nasa93 dataset (COCOMO NASA 2 / software cost estimation),” dataset description page.
[8] U. Jeklin et al., “Evaluation of COCOMO Model Accuracy in Software Effort Estimation,” 2025 (uses MAE/MMRE/Pred(0.25) on COCOMO/NASA dataset).
[9] PROMISE Repository dataset file: “cocomonasa_2.arff,” dataset record indicating 93 NASA projects and attributes.
[10] NASA report noting COCOMO development from 63-project database
[11] B. W. Boehm, *Software Engineering Economics* (COCOMO/SLIM discussion excerpt PDF).
[12] M. F. Hillman, “40 Years Journey of Function Point Analysis…,” 2019.
[13] M. Shin and A. L. Goel, “Empirical Data Modeling in Software Engineering Using Radial Basis Functions,” *IEEE Transactions on Software Engineering*, 2000.

[14] A. F. Sheta, "Software Effort Estimation for NASA Projects Using Genetic Programming," 2010.
[15] M. Uysal, "Multivariate Interpolation Model to Estimate the Effort Component of Software Projects," *Information Technology Journal*, vol. 5, pp. 1143–1145, 2006. DOI: 10.3923/itj.2006.1143.1145.
[16] M. Uysal, "Estimation of the Effort Component of the Software Projects Using Heuristic Algorithms," in *New Trends in Technologies*, IntechOpen, 2010. DOI: 10.5772/7583.
[17] "COCOMO II Model Definition Manual," including calibrated constants and drivers.
[18] A. Mittal et al., "Software cost estimation using fuzzy logic," *ACM* .
[19] Lecture-style summary of empirical estimation equations (Walston–Felix, Bailey–Basili, Doty, etc.) .
[20] QSM history page referencing SLIM methodology origins and (risk/Monte Carlo) tool evolution.
[21] S. Nakamura, *Numerical Analysis and Graphic Visualization*.