IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 13 November 2023, accepted 26 November 2023, date of publication 30 November 2023,
date of current version 6 December 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3337809

RESEARCH ARTICLE

Gradient Boosting Optimized Through Differential Evolution for Predicting the Testing Effort of Software Projects

ANGEL J. SÁNCHEZ-GARCÍA[1], CUAUHTÉMOC LÓPEZ-MARTÍN[2], AND ALAIN ABRAN[3], (Life Senior Member, IEEE)

[1]Facultad de Estadística e Informática, Universidad Veracruzana, Xalapa, Veracruz 91000, Mexico
[2]Department of Information Systems, Universidad de Guadalajara, Zapopan, Jalisco 45100, Mexico
[3]Department of Software Engineering and Information Technology, École de Technologie Supérieure, Université du Québec, Montreal, QC H3C 1K3, Canada

Corresponding author: Angel J. Sánchez-García (angesanchez@uv.mx)

ABSTRACT Software testing (ST) is one of the most important software development life cycle (SDLC) phases and ST effort is often expressed as a percentage of SDLC effort. Unfortunately, in the literature ST effort percentage ranges from 10% to 60%. In the literature most of the machine learning algorithms and metaheuristics for optimizing them have looked at predicting overall SDLC effort without focusing on any specific SDLC phase, including testing. Therefore, this study investigates the application of the Software Testing Effort Prediction (STEP) of Gradient Boosting (GB) machine learning regression algorithm optimized through Differential Evolution (DE). Its prediction accuracy is compared with those obtained when the GB is also optimized through Particle Swarm Optimization (PSO) and Genetic Algorithms (GA). The performance of GB-DE, GB-PSO, and GB-GA was also compared to that of statistical regression (SR). Seven data sets of actual projects were selected from an international public repository for software projects. The results showed that GB-DE was statistically better than SR in all seven data sets at 95% confidence, whereas GB-PSO and GB-GA were better than SR in four and three data sets, respectively. Thus, we can conclude that GB-DE can be used for STEP of either new projects or enhancement projects developed in either the third or fourth programming language generation.

INDEX TERMS Testing effort prediction, gradient boosting, differential evolution, particle swarm optimization, genetic algorithms, ISBSG.

I. INTRODUCTION

Software Engineering is a discipline in which principles, methods, and techniques are applied for the management of software development life cycle (SDLC) processes, including for definition of business needs (requirements engineering), development (design, construction, and testing), and maintenance phases of software projects. Each SDLC phase has its specific objectives and activities to performed to achieve a high quality software product.

Software quality is directly associated with software testing (ST) [1], which influences the cost, and success of a software project [2], [3], [4], [5]. To avoid losing customers, software companies are investing considerable resources in ST to release defect-free products [6], [7]. Therefore, software project managers must adequately predict the number of resources required for project completion within the established schedule, costs, and quality targets [8], [9].

It is advisable to have a team dedicated exclusively to ST activities as it allows unbiased testing [10]. When a software project is planned, software testing effort prediction (STEP) is performed to assign suitable resources, such that testing teams are able to execute efficient tests for identifying and fixing them [11]. This team requires an STEP to schedule its activities, request resources, and negotiate the update of deadlines [7], [8], [9], [10], [12].

The main problem which motivates the present study is that the percentage of effort dedicated to ST with respect to the

The associate editor coordinating the review of this manuscript and approving it for publication was Xiong Luo.

total SDLC varies among the published studies, ranging from 10% to 25% [13], 25% to 40% [14], 27% [15], 40% to 50% [16], [17], [18], [19], [20], 50% to 60% [21], [22], [23], and more than 60% for critical software [24].

Therefore, these percentages between 10% and 60% represents a large uncertainty range for software managers. An STEP is needed since either under or over-prediction can negatively impact budgets. Accordingly, there exists the need for researching on STEP for managing, planning, and assign ST resources.

Regarding software prediction field:

- Several variables have been used in STEP models to represent the effort such as number of CPU hours [21], number of executed test cases [22], number of lines of code changed or added by class throughout the life cycle of the defect [25], and person-hours needed to define, implement, and execute the tests [1], [24].
- The most widely used variable regarding SDLC effort is person-hours [26], [27], [28], [29], [30], [31], [32], [33].
- Software project size is often used in SDLC effort prediction as an explanatory variable [30].

Therefore, we selected person-hours to represent the effort, and size of the software project as the explanatory for STEP.

In the software effort prediction field, the models commonly used have been based on statistical regression equations, and machine learning models [30].

The No Free Lunch theorem (NFL) [34] states that there is no optimization or machine learning algorithm that works best on all problems. In other words, this theorem highlights the importance of choosing and adapting optimization and machine learning algorithms to specific problems instead of looking for one that solves all problems. Thus, we chose to explore a special type of machine learning algorithms called ensemble regression methods.

Ensemble regression methods are machine learning algorithms that combine multiple single regression models to build a more powerful and accurate model [35]. These methods are based on the idea that a combination of multiple weak models can overcome the individual limitations of each model and provide more accurate and robust predictions.

Ensemble methods have been used in different domains [36], and their performance has outperformed that of individual regressors [37], [38]. Rather than relying on a single regression model, ensemble methods take advantage of model diversity and collective wisdom to improve overall model performance. Each weak model, often called a ''base estimator'', is independently trained on part of the data set or a modified version of the data set. Next, the predictions from each model are combined using an aggregation strategy to obtain the final prediction.

Gradient Boosting (GB) is an ensemble machine learning algorithm for regression and classification tasks that produces a final prediction model in the form of an ensemble of weak tree models. The main advantage is that the construction of each tree adapts to the residual errors of the previous model. In other words, in each iteration it highlights records that have been incorrectly predicted in previous iterations.

Machine learning techniques commonly use metaheuristics to optimize their parameters to obtain the best performance. In accordance with our findings, we identified 31 types of metaheuristic algorithms used in software effort prediction models. We selected to Differential Evolution (DE) to the GB optimization, owing to DE is a stochastic evolutionary metaheuristic considered as a promising algorithm for solving optimization problems in the real world [39]. This type of metaheuristic is used to find approximate solutions to problems with search spaces that may be difficult to address using conventional exact or algorithmic methods.

There are two main reasons for selecting DE, firstly, in accordance with our findings (''II.Related work'' section) we did not find any study using DE for STEP. Secondly, DE has certain advantages that adjust to the type of optimization problem we have, such as those mentioned below:

- Its mutation operator enables rapid exploration of the search space and effective exploitation of promising regions.
- DE has few parameters to adjust, compared to some others such as the Genetic algorithm.
- DE adjusts to problems of different dimensions and complexities, as is the case with the GB hyperparameters optimization.
- DE (like other evolutionary computing metaheuristics) tends to perform a broader exploration of the search space than collective intelligence metaheuristics (such as Particle Swarm Optimization), which is useful with a complex search space.

Thus, the contribution of the present study is the investigation of the application of the GB regression algorithm optimized through the DE metaheuristic for STEP. We based the utility and originality of our contributions on the following reasons:

- SDLC effort prediction is a common practice, whereas STEP studies are still scarce and a major challenge [18].
- Machine learning (ML) models are more accurate than non-ML models when applied to SDLC predictions [27].
- Ensemble methods have shown better performance than individual regressors [37], [38].
- GB has shown better prediction performance than other ML algorithms such as Random Forest, SupportVector Machine and probabilistic approach-based Bayesian regression when used in several applications [40]. Moreover, it has been reported to yield good results when applied to SDLC effort prediction [38]. Because GB requires tuning several parameters, we use DE to optimize its hyperparameters since DE works with the representation of real numbers.
- In accordance with the ''II. Related work'' section of the present study, we did not identify any studies

proposing the application of GB for STEP. Accordingly, no metaheuristic was applied for the GB optimization.

In the present study, we compared the results of our proposal with simple linear regression (SLR) for the following reasons: any new proposed model must outperform that of a statistical regression model [41] and a statistical regression model is mostly used when comparing its results to those obtained from SDLC ML models [27], [30].

Because the proposed STEP model is based on the GB regression algorithm whose parameters are optimized by the DE metaheuristic, the null (H_0) and alternative (H_1) hypotheses to be tested in our study are formulated as follows:

- H_{DE0}: The prediction accuracy of GB optimized through DE (GB-DE) is statistically equal to that of SLR when these models are applied to predict the testing effort of software projects using the size of project as the explanatory variable.
- H_{DE1}: The prediction accuracy of the GB optimized through (GB-DE) is not statistically equal to that of the SLR when these models are applied to predict the testing effort of software projects using the size of the project as the explanatory variable.

In addition to DE, the two algorithms mostly applied for optimization are Particle Swarm Optimization (PSO) and Genetic Algorithm (GA) in accordance with our results obtained after identifying studies using metaheuristics for optimizing software effort prediction models (Section ''II.Related work'' of the present study). Thus, the GB will be also optimized through PSO and GA, such that their performance is also compared to that of SLR. Therefore, two additional hypotheses are also formulated:

- H_{PSO0}: The prediction accuracy of the GB optimized through PSO (GB-PSO) is statistically equal to that of the SLR when these models are applied to predict the testing effort of software projects using the size of project as the explanatory variable.
- H_{PSO1}: The prediction accuracy of the GB optimized through PSO (GB-PSO) is statistically not equal to that of the SLR when these models are applied to predict the testing effort of software projects using the size of project as the explanatory variable.
- H_{GA0}: The prediction accuracy of the GB optimized through GA (GB-GA) is statistically equal to that of the SLR when these models are applied to predict the testing effort of software projects using the size of project as the explanatory variable.
- H_{GA1}: The prediction accuracy of the GB optimized through (GB-GA) is statistically not equal to that of the SLR when these models are applied to predict the testing effort of software projects using the size of project as the explanatory variable.

Owning to research on software prediction should involve real data from industry [29], data of recent software projects, the use of the same validation method to train and test the models, as well as the use of the same data sets to compare prediction models [30], the data sets of projects used in our study come from the 2022 release of the ISBSG international public repository of software projects (i.e., ISBSG release 2022). The ISBSG includes 11,281 software projects developed between 1989 and 2021. These projects were performed in more than 30 different countries, having as major contributors to Spain (25.5%), Netherlands (20.1%), United States (9.7%), Finland (7.2%), France (5.7%), India (5.3%), Australia (4.9%), China (4.9%), Japan (3.4%), and Canada (3.1%). Regarding industry sector, the major sectors are Communications (29.5%), Insurance (14.8%), Manufacturing (10.4%), Government (10.2%), Banking (9.1%), Medical and health care (5.3%), Financial (4.5%), Wholesale/Retail (2.2%), Electronics/computers (2.0%) and Service industry (1.9%). As for application types, the major ones are Financial Transaction Process/Accounting (27.1%), Customer management (12.9%), Management Information System (8.4%), Transaction/Production System (8.2%), Billing (6.4%), Business (4.3%), Sales and Marketing (4.3%), Data Warehouse (3.9%), Complex applications (2.9%), Workflow (2.9%), Embedded systems (2.3%), Integration (2.2%), Document management (1.9%), Stock control (1.8%), Catalogue (1.7%), Logistics (1.6%), Electronic Data Interchange (1.4%), Analysis and Reporting (1.4%), Network management, Communications (1.3%), Web based (1.2%), Application software (1.0%), and Office information system (0.9%) [42]. The ISBSG has also been used in many studies on software effort prediction [31].

This paper is structured as follows: Section II analyzes the related work on the generation of models for STEP when they are optimized through metaheuristics. Section III describes the metaheuristics implemented to optimize the GB in this study (i.e., DE, PSO and GA). Section IV presents the datasets and projects used to validate the proposed method. Section V details the empirical setup and the results. Section VI presents a comparison with previous studies, limitations, and validity threats. Finally, Section VII concludes the study and mentions the future work.

II. RELATED WORK

This related work section is divided in three subsections: studies related to the application of metaheuristics for software effort prediction models; studies specific to software testing effort prediction models, and studies related to Differential Evolution in the software effort prediction models.

A. STUDIES RELATED TO THE APPLICATION OF METAHEURISTICS FOR SOFTWARE EFFORT PREDICTION MODELS

Table 1 includes forty-eight studies published between 2000 and 2023.

- 29 of them (i.e., 60%) were published in the last five years,
- 39 of them (i.e., 81%) between 2016 and 2023, and
- only three of them before 2010.

TABLE 1. Studies classified by publication year on metaheuristic algorithms applied for feature weighting in software effort prediction models.

Year	Study
2000	[43]
2006	[44]
2008	[45]
2012	[19]
2013	[46] [47]
2014	[48] [49]
2015	[50]
2016	[51] [52] [53] [54]
2017	[55] [56] [57]
2018	[58] [59] [60]
2019	[61] [62]
2020	[63] [64] [65] [66] [67]
2021	[68] [69] [70] [71] [72] [73] [74] [75] [76]
2022	[77] [78] [79] [80] [81] [82] [83] [84] [85]
2023	[86] [87] [88] [89]

TABLE 2. Metaheuristic algorithms applied for feature weighting in software effort prediction models.

Algorithm	Study
Ant colony	[72]
Ant lion	[79]
Bacterial foraging	[56]
Bee colony	[57] [58]
Bat	[49] [72] [79] [80]
Biogeography	[63]
Black hole	[81]
Crow	[71]
Cuckoo	[19] [72] [73]
Differential evolution (DE)	[48] [50] [51] [52] [53] [55] [59] [68] [69] [70] [76] [77] [79] [86]
Dolphin	[64]
Dragonfly	[85]
Equilibrium optimizer	[71]
Firefly	[51] [57] [61] [80] [81]
Fireworks	[79]
Flower pollination	[73]
Genetic algorithm	[43] [44] [45] [53] [57] [59] [64] [65] [70] [72] [76] [78] [79] [80] [81] [87] [88]
Genetic elephant herding	[83]
Gray wolf	[71] [72] [74] [89]
Harmony	[54] [74]
Moth-flame	[71]
Particle swarm optimization	[46] [53] [57] [58] [59] [65] [66] [68] [70] [72] [73] [75] [76] [78] [79] [82] [86]
Salp swarm	[71]
Satin bowerbird	[57] [62] [70]
Seagull Optimization	[71]
Simulated annealing	[75]
Sine cosine	[71]
Strawberry plant	[60] [72] [74]
Tabu	[47]
Whale	[67] [68] [71] [84] [86]
Woodpecker mating	[71]

In summary, the application of metaheuristics suggests a recent research issue in the field of software-effort prediction.

The 31 types of metaheuristic algorithms used in the studies listed in Table 1 are alphabetically ordered in Table 2. The three types of algorithms mostly applied were DE (in 14 studies), GA and PSO (each in 17 studies).

The metaheuristic algorithms listed in Table 2 were used to optimize the parameters of the following three types of models:

1) Machine learning:
   - Adaptive Neuro-Fuzzy Inference System [57], [79], [83].
   - Analogy-based estimations [44], [48], [51], [52], [53], [55], [59], [69], [75].
   - Case-based reasoning model [46], [87].
   - Classification and regression Tree [81].
   - Deep belief network [84].
   - Deep Neural Network [72].
   - Bi-directional Associative Memory [88]
   - Fully connected neural network [89].
   - Functional link neural network [67].
   - Fuzzy logic [78].
   - Multi-layer perceptron neural network [43], [71], [85].
   - Neuro-fuzzy inference system [70].
   - Radial basis function neural network [67].
   - Symbolic regression method [50].
   - Support vector regression [47], [62].
   - Wavelet neural network [80].
2) Mathematical methods:
   - Fuzzy logic system [61], [65], [81].
   - Grey relational analysis [45].
3) Equations:
   - Statistical Equations [54], [56], [60], [63], [64], [66], [68], [73], [74], [76], [77], [82], and [86].
   - Non-statistical equations [19], [49], and [56].

Table 3 includes 14 studies related to the application of DE for optimizing the parameters of the software effort prediction models. It shows the source and size (number of software projects) of the data sets, the criteria to evaluate the prediction accuracy, the validation method used for training and testing the models, and whether the conclusions of the studies were based on a statistically significant difference. In Table 3 it can be observed that the data sets used are very old and most have very small data. In addition, when the studies do not report a model validation method, the results may not be repeatable because the projects used for training and testing the models are not indicated.

B. STUDIES RELATED TO THE SOFTWARE TESTING EFFORT PREDICTION (STEP)

Thirty-one studies on STEP were classified into four categories: practices, methods, metrics, and models for predicting probability, category, and person-hours [90]. Twelve of them had the objective of predicting person-hours (i.e., effort) and only two of them based their conclusions on statistical significance. Each of these two studies used a data set of software projects and one type of prediction model: a neural network [14] and a statistical regression model [92]. In addition to these two models, four machine learning models have been applied to STEP, and their results are reported based on statistical significance: case-based reasoning, decision trees, support vector regression, and genetic programming [90].

We also identified a study that used different types of FSM in statistical regression models for SDEP [91].

As for the three STEP studies recently published, the first proposed a stacking ensemble model involving neural networks, support vector regression, and decision tree for predicting the effort of a specific type of testing: regression test [93], the second analyzed the impact of data quality on STEP [94], whereas the third applied statistical regressions to predict the testing effort for mobile applications [95].

C. STUDIES RELATED TO DIFFERENTIAL EVOLUTION (DE) IN THE SOFTWARE EFFORT PREDICTION MODELS

When a metaheuristic is applied to an optimization problem, it is important to report three important aspects: the version of the metaheuristic used, the parameters used, and performance results of the algorithm.

Regarding the version of metaheuristics, an algorithm can have different versions, which vary in the operators or the way to represent solutions. These metaheuristics also require parameter tuning, such as the probability of applying an operator or the way in which a tradeoff is made between the exploration of new solutions and the exploitation of a promising solution in the search space. This is important because the selection of these parameters can lead to a global optimum or premature convergence.

Bioinspired metaheuristics (evolutionary algorithms and collective intelligence) are stochastic. Therefore, executing them several times can generate good, although different, results for each execution. To determine the performance of an algorithm, it is suggested to execute it several times and analyze its variability and results over time using graphical methods, such as convergence plots. This provides an idea of the expected results each time a potential solution is searched and helps researchers make decisions about selecting a particular metaheuristic for a specific problem.

Table 4 summarizes the studies described in Table 3 with respect to the experimental characteristics and results of applying DE to the software effort prediction models. It can be noted that only two of the 14 studies reported on the search space and that at most 50% of the studies reported on the DE version, DE performance, and number of executions.

III. OPTIMIZATION OF GRADIENT BOOSTING REGRESSOR

This section describes the operation of the GB and the parameters to be optimized. In addition, the implementation of the metaheuristics to be compared and the fitness function to be minimized are described.

A. GRADIENT BOOSTING

GB was developed by Breiman [96] based on the idea of gradually improving the performance of a model through a combination of simpler models (usually trees). Later, Friedman proposed minimizing a loss function by fitting weak models at each stage [97], [98].

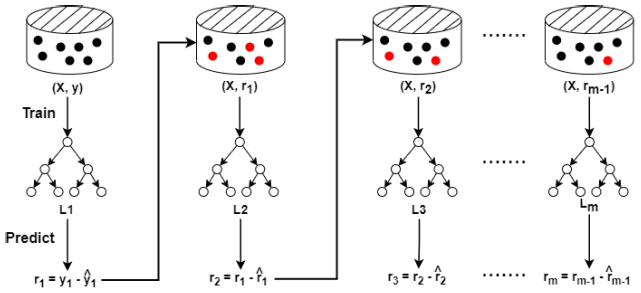


FIGURE 1. Representation of the GB process.

At each iteration of the algorithm, a new weak model is fitted to correct the errors made by previous models. These weak models are added sequentially and weighted to control for their influence on the final model.

In regression problems, the objective is to find a function $F(X)$ that best approximates the values of the dependent variable y as shown in (1), where L is a Loss function to be minimized given the predicted values y_i and the actual value y. This is the first tree to be built with a single leaf, after which, trees with greater depth are built.

$$F(X) = \sum_{i=1}^{n} L(y_i, y) \tag{1}$$

Different loss functions can be used, such as the MSE, Mean Absolute Error (MAE), or Huber loss, which is a combination of MSE and MAE. In the present study, the Huber Loss (HL) function is used since it is less sensitive to unusual values compared to MSE and MAE [99]; therefore, it is a robust alternative because it balances the quadratic penalty of the MSE with the linear penalty of the MAE. The Huber function loss is given by (2).

$$HL = \begin{cases} \frac{1}{2}(y - \hat{y})^2 & for|(y - \hat{y})| \leq \alpha \\ \alpha\left(|(y - \hat{y})| - \frac{1}{2}\alpha\right) & otherwise \end{cases} \tag{2}$$

In the GB algorithm, the first weak learner L_1 is fitted using data X, from which the response variable y is predicted, and the residuals $r_1 = y_1 - \hat{y}_1$ are calculated. Next, a new model L_2 is fitted, which attempts to predict the residuals of the previous model; in other words, it attempts to correct the errors that model L_1 has made as $L_2(X, r_1)$ to later obtain its residuals $r_2 = r_1 - \hat{r}_1$. This process is repeated $m - 1$ times, where m is the number of weak learners, as shown in Fig. 1.

GB is characterized by having a considerable number of hyperparameters "which the performance can differ greatly depending on its setting" [100]. The number of weak learners, is the number of estimators used to obtain the predicted value.

Because the objective of GB is to minimize the residual in each iteration, it is susceptible to overfitting. One way to avoid this problem is using a regularization value also known as "learning rate", which limits the influence of each weak learner on the ensemble (i.e., the rate from which the model learns). The correct choice of this parameter may

TABLE 3. Data from studies related to DE applied for feature weighting in software effort prediction models.

Study	Data set source	Size	Prediction accuracy criterion	Validation method	Statistical significance
[48]	Albrecht	NR	MRE, Pred(l)	NR	No
[50]	Poznan University of Technology dataset	24	MRE, Pred(l), MSE	k-fold cross validation ($k = 3$)	Yes
[51]	NASA 60	60	MRE, Pred(l)	NR	No
[52]	Desharnais	77	MRE, Pred(l)	NR	No
[53]	Artificial data set	300	MRE, Pred(l)	k-fold cross validation ($k = 3$)	No
	Desharnais	77			
	ISBSG Release 11	66			
[55]	Artificial data set	300	MRE, Pred(l)	NR	No
	Kemerer	15			
	COCOMO 81	63			
[59]	Desharnais	77	MRE, Pred(l), Absolute residuals, standardized accuracy (SA), and effect size (Δ)	NR	No
	COCOMO 81	63			
	NASA	18			
	Maxwell	62			
	Albrecht	24			
	China	499			
[68]	NASA 93	93	MRE, Pred(l), RMSE	NR	No
[69]	Kemerer	15	MRE, Absolute residuals, Standardized accuracy (SA)	k-fold cross validation ($k = 3$ and 10)	No
	ISBSG Release 10	37			
	Miyasaki	48			
	Desharnais	77			
	China	499			
[70]	Kemerer	15	MER, MRE, Pred(l), RMSE	k-fold cross validation ($k = 3$)	No
[76]	COCOMO 81	33	MRE, Pred(l), RMSE	NR	No
[77]	NASA 93	93	MRE, Pred(l), RMSE	NR	No
[78]	COCOMO 81	63	MER, MRE, Pred(l), RMSE	NR	No
[86]	NASA 93	93	MRE, Pred(l), RMSE	NR	No

MER: magnitude error relative; MRE: Magnitude of relative error; MSE: Mean square error; RMSE: Root mean square error; NR: Not reported.

TABLE 4. Experimental characteristics and results applying DE in software effort prediction models.

Study	Search space	DE version	Parameter values	DE performance	Executions
[48]	No	Simple DE Population based DE	No	NR	NR
[50]	No	DE rand/1/bin	SP = 40, Gen = 60, F = 0.7, CR = 0.4	NR	10
[51]	No	NR	NR	Execution time and memory used	NR
[52]	No	NR	NR	NR	NR
[53]	No	DE rand/1/bin	SP = 30, Gen = 100, F= 0.1, CR = 0.1	Descriptive statistics and Convergence plots	NR
[55]	No	NR	Gen = 1000	NR	NR
[59]	No	DE rand/1/bin DE best/1/bin DE rand/2/bin DE best/2/bin DE rand-to-best/1/bin	F = 0.8, CR = 0.5	Descriptive statistics	10
[68]	No	NR	SP = 100, Gen = 200	Descriptive statistics and Convergence plots	20
[69]	No	DE2 and DE8	SP = 100, Gen = 20, F = 0.8, CR = 0.7	NR	20
[70]	No	NR	SP = 80, F = 0.8, CR = 0.5	NR	NR
[76]	No	DE rand/1/bin	SP = 100 * D, Gen = 100, CR = 0.24, F = 0.66	Convergence plots	10
[77]	No	NR	NR	Pareto front plots	30
[78]	Yes	NR	SP = 100, F = [0.1-0.9], CR = [0.1-0.9]	NR	50
[86]	Yes	NR	SP = 100, Gen = 100, 200, and 500, F = [0.1 - 0.9], CR = [0.1 - 0.8]	Boxplots and Pareto front plots	30

SP: Size of population, Gen: Maximum number of generations, F: Scaling Factor, CR: Crossover rate parameter.

vary depending on the problem. If the learning rate is low, more trees are required to obtain good results, but the risk of overfitting is lower. Table 5 describes the GB parameters to be optimized and their search space (value bounds).

B. HYPERPARAMETER OPTIMIZATION

Based on the problems described in the previous section, we identified an optimization problem. Therefore, we used DE to optimize GB hyperparameters. In addition,

TABLE 5. Description of hyperparameters of the GB algorithm.

Hyperparameter	Description	Search space
Number of estimators	Represents the number of weak learners. It also represents the number of boosting stages.	[2, 100]
Sumbsample	Fraction of samples used for individual weak learners adjustment.	(0.0,1.0]
Alpha	The alpha-quantile of the Huber loss function shown in (2).	(0.0,1.0)
Validation fraction	Proportion of training data to reserve as a validation set at an early interruption.	(0.0,1.0)
Learning rate	Parameter that reduces the contribution of each tree at each iteration. Although the maximum value can be infinite, we limit the search space to a maximum value of 100.	[0.0,100]

we compared the results of the PSO and a GA implementations for the following reasons.

First, we selected another metaheuristic that also belonged to evolutionary algorithms. We selected GA because it is one of the two most reported in the literature review presented in section II and we consider it to be a good point of comparison. Furthermore, the type of representation of the genetic algorithm as a vector of real numbers fits the optimization problem at hand. Also, by having several parameters to adjust, it could provide greater flexibility in finding solutions.

The other metaheuristic to compare is PSO because we want to take another metaheuristic that do not belong to evolutionary computation, but rather with a collective intelligence approach, since cooperation and social adaptation of this metaheuristic could be potentially good for this problem. Specifically from collective intelligence, PSO was selected to be compared with DE, because together with GA, they are the most reported in the software effort estimation literature. Additionally, like DE, PSO has a limited number of parameters to adjust, making it simple to configure. Also, the representation of solutions allows us to address problems with different dimensions. Finally, PSO seeks a balance between the exploration of the search space and the exploitation of current solutions, which we consider can be effective for the hyperparameter optimization problem.

It is worth mentioning that we implemented the canonical versions of the three metaheuristics for the following reasons:

- Comparisons between different algorithms usually provide general evidence, that is, this comparison provides us with a baseline to evaluate its performance in its most basic version. This can be useful for understanding the basic behavior of a metaheuristic before considering improvements or modifications.
- The fairest and most usual way of comparing metaheuristics (even as presented in the reported related work) is to implement the canonical forms, which is important to avoid bias in the evaluation.
- The comparison between variants would be longer and would move away from the main problem for software engineers, where the aim is to provide simpler or more concrete evidence of the precision for estimating software effort in the testing stage. For simplicity, canonical versions are simpler and easier to understand compared to their variants, making it easier to communicate findings to software engineering practitioners and other researchers.

DE, PSO, and GA need an objective function called "fitness function", which is described in the following subsection.

1) FITNESS FUNCTION

The term fitness function is linked to evolutionary algorithms derived from the theory of evolution as a way of finding the fittest individuals in the process of natural selection. The term"fitness" was introduced to measure the reproductive success of individuals using genetic algorithms [101]. However, this term is currently used in many evolutionary algorithms.

The fitness function guides the evolution of a population of individuals. It defines how good or bad a potential solution to a problem is (i.e., how close or far it is from the objective).

Optimizing the hyperparameters of a regression model problem requires a function to measure the prediction accuracy of each built model (i.e., the prediction error). Since prediction accuracy measures commonly used such as mean square error, mean magnitude of relative error, and mean magnitude of error relative to the estimate are based on ratios and then correspond to biased accuracy measures [102], we rather the absolute residual (AR) and its mean MAR, which are defined in (3) and (4).

$$MAR = \frac{1}{N}\sum_{i=1}^{N} AR_i \tag{3}$$

$$AR_i = |Actual\ testing\ effort_i - Predicted\ testing\ effort_i| \tag{4}$$

Because we search for the solution that has the smallest MAR value, this becomes a minimization problem. To obtain the MAR shown in (3), we performed a Leave-One-Out Cross Validation (LOOCV) process, where each project was used as test data and the other n-1 projects from each dataset as training data.

Next, we detail the implemented versions of each metaheuristic that were tested and compared to optimize the GB hyperparameters using the described fitness function.

2) DIFFERENTIAL EVOLUTION (DE)

DE [103] works from vectors as shown in (5), where $\vec{x}_{i,G}$ represents vector i in generation G, and NP is the size of the population.

$$\vec{x}_{i,G} = 1, \ldots, NP \tag{5}$$

During the mutation process, each vector $\vec{x}_{i,G}$ (target vector) in the population generates a descendant $\vec{u}_{i,G}$ (trial

vector) through a mutant vector $\overrightarrow{v}_{i,G}$. This mutant vector is obtained as shown in (6), where $\overrightarrow{x}_{r0,G}$, $\overrightarrow{x}_{r1,G}$ and $\overrightarrow{x}_{r2,G}$ are randomly selected vectors from the current population, taking care that $r0 \neq r1 \neq r2$. $\overrightarrow{x}_{r0,G}$ is known as the base vector and $\overrightarrow{x}_{r1,G}$ and $\overrightarrow{x}_{r2,G}$ are the difference vectors. F is a user-defined scale factor that controls how much an individual is different from other individuals in the previous generation.

$$\overrightarrow{v}_{i,G} = \overrightarrow{x}_{r0,g} + F(\overrightarrow{x}_{r1,g} + \overrightarrow{x}_{r2,g}) \tag{6}$$

In the crossover operator, once the mutant vector $\overrightarrow{v}_{i,G}$ is generated, it is combined with the target vector $\overrightarrow{x}_{i,G}$ to generate the trial vector $\overrightarrow{u}_{i,G}$ as shown in (7), where $CR \in [0, 1]$ is the probability that a value in the vector is combined by the crossover operator with another value, in other words, it defines the similarity between the trial vector and the mutant vector. The value $rand_j$ generates a real random number with a uniform distribution between 0 and 1, $j \in 1, \ldots, n$ is the jth variable of the vector, $J_{rand} \in [1, n]$ is an integer that prevents the trial from being a copy of the target vector. For our problem, $n = 5$ (five GB hyperparameters listed in Table 5).

$$\overrightarrow{u}_{i,j,G} = \begin{cases} \overrightarrow{v}_{i,j,G} \ (rand_j \leq CR) or (j = J_{rand}) \\ \\ \overrightarrow{x}_{i,j,G} \qquad otherwise \end{cases} \tag{7}$$

Finally, the best vector between the target vector and the trial vector, according to the fitness function, is selected to remain in the population as shown in (8) (assuming a minimization problem).

$$\overrightarrow{x}_{i,G+1} = \begin{cases} \overrightarrow{u}_{i,G} \ f(\overrightarrow{u}_{i,G}) \leq f(\overrightarrow{x}_{i,G}) \\ \\ \overrightarrow{x}_{i,G} \qquad otherwise \end{cases} \tag{8}$$

This variant of DE is known as $DE\ rand/1/bin$, where "*rand*" means the criterion used for the selection of the base vector $\overrightarrow{x}_{i,G}$, "1" indicates the number of difference vectors used and "*bin*" is the type of crossover operator used. In this case it is binomial, as shown in Equation (7).

3) PARTICLE SWARM OPTIMIZATION (PSO)

PSO is a metaheuristic that simulates the behavior of natural particles [104]. In PSO, unlike GA and DE, information is obtained from the entire population. PSO operates with a population (swarm) of candidate solutions (called particles). These particles move along the search space and their movement depends on the best position obtained, as well as the best global position found by the swarm. To do this, each particle has a position, $\overrightarrow{p}$ (which for our problem is represented by a vector of the form $[x_1, x_2, x_3, x_4, x_5]$, a value for each GB hyperparameter) in search space and a velocity, $\overrightarrow{v}$ (of the form $v_{x1}, v_{x2}, v_{x3}, v_{x4}, v_{x5}$) which determines its movement through space.

Furthermore, each particle emulating the real world exhibits both inertia and acceleration. The inertia continues to the particle in the direction in which it moves into the population, whereas the acceleration depends on two characteristics:

- Each particle is attracted to the best location it has found (p_i^{Best}).
- Each particle is attracted to the best location found by the set of particles (p_g^{Best}).

Based on these characteristics, $\overrightarrow{v}_i(t)$ which denotes the velocity of particle i at time t is calculated as in (9), where c_1 and c_2 are the constants of attraction to the best location of the particle and the global best location of the swarm respectively. Also, r_1 and r_2 are two random numbers $\in$ [0,1]. Once the velocities of all particles have been updated, their positions are updated in each iteration as in (10).

$$\begin{aligned} v_i(t+1) = v_i(t) &+ c_1 \cdot r_1 \cdot (p_i^{Best} - p_i(t)) \\ &+ c_2 \cdot r_2 \cdot (p_g^{Best} p_i(t)) \end{aligned} \tag{9}$$

$$p_i(t+1) = p_i(t) + v_i(t) \tag{10}$$

4) GENETIC ALGORITHM (GA)

The GA was proposed by Holland in 1975 [101]. This is perhaps one of the three most used metaheuristics in software effort prediction models as shown in Table 2. It is characterized by having potential solutions in the form of a binary string called genotype (although there is also a representation for real numbers), where each element of the string is called a chromosome. These individuals evolve over time. In GA, it is important to define the crossover and mutation operators, as well as the individual selection process and individual replacement mechanism, all of which are described next.

When an individual is represented by a string of chromosomes, it allows for more versatile solution exploration. Owing to the nature of the hyperparameters, we selected a real representation, in which each chromosome is a real number. In this study, we implemented a deterministic tournament for the selection process of parents, where several individuals from the population were randomly selected and their fitness values were compared [105]. The two individuals with the best fitness in each tournament were selected as parents to generate the offspring. This choice was based on the idea that selecting the best individuals in each tournament helps maintain and improve the quality of the solutions throughout the generations. Additionally, as a random tournament, there is an opportunity for all individuals to participate in tournaments.

Crossover and mutation operators are important for population variations. The crossover operator that we selected to simulate reproduction and generate offspring was "Intermediate recombination" [106], in which, from the two parents: $P_1 = V_1, \ldots, V_m$ and $P_2 = W_1, \ldots, W_m$, the value of each son's allele was generated by mixing the parental alleles, as in (11) and (12) for the two sons S_1 and S_2. In this method, P_1 and P_2 intersect at the position k. The parameter $a \in$ [0,1] is given by the user and biases the proportion of information that each son will have from each of its parents starting from

position k. This method helps maintain genetic diversity in the population by introducing new combinations of genes.

$$S_1 = V_1, \ldots V_k, W_{k+1} * a + V_{k+1} * (1-a), \ldots, W_m * a + V_m * (1-a) \quad (11)$$

$$S_2 = W_1, \ldots W_k, V_{k+1} * a + W_{k+1} * (1-a), \ldots, V_m * a + W_m * (1-a) \quad (12)$$

For the mutation operator, we use the ''uniform mutation'' method, which replaces the value of a randomly selected chromosome with a random value between the allowed limits. This selection is justified because a uniform mutation is used for genes with integer or float values. Finally, the type of replacement was elitist, where the best individuals survived in each generation to maintain the best fit.

After describing the versions and implementations of the three metaheuristics, we summarize the following: On the one hand, PSO is an algorithm based on the collective intelligence approach, inspired by the behavior of groups of organisms that share information with each other. DE and GA are based on biological evolution characteristics, such as natural selection and reproduction.

On the other hand, the way of representing individuals is different, while in PSO the potential solutions are represented with particles that have a position and velocity in the search space, the solutions in GA are represented as a vector of chromosomes where each gen is a value to optimize. In DE and GA, crossover and mutation operators allow to generate new solutions from those in the population (descendants). On the other hand, PSO does not have crossover or mutation operators, since the particles update their values based on their own knowledge and the knowledge of the entire swarm (there are no descendants).

Finally, for the problem of optimizing GB hyperparameters, PSO and GA have the same objective, but the search for an optimal solution is different due to their representation of each individual and the operators that allow reaching the best solution.

IV. DATASETS OF SOFTWARE PROJECTS

In the ISBSG, the testing effort by project is reported in person-hours spent on the following activities: testing plan, system testing, performance testing, creation and execution of automated tests, and acceptance testing [107]. However, the reporting of testing effort is not a mandatory field in the ISBSG repository, and a number of projects have null values in this field, similar to the other criteria used to extract projects, such as functional size methods or language type.

Table 6 shows the number of projects extracted from the ISBSG repository excluding non-null values for testing effort, functional size method (FSM), language type, resource level, and those with A or B ratings on data quality, and functional sizing methods rating [107].

The ISBSG also classifies software projects according to the type of development. Table 7 shows the final 2,009 projects in Table 6, classified by new, enhancement, and re-development types.

TABLE 6. Criteria for selecting software projects from ISBSG data set (11,281 projects).

Attribute	Selected value(s)	Projects
Test effort	Not null	3,618
Functional sizing methods	Not null	3,616
Language type	Not null	2,935
Resource level	Not null	2,838
Data quality rating	A, B	2,529
Unadjusted Function Point Rating	A, B	2,127
Functional sizing methods	Fisma, Nesma, IFPUG V4+	2,009

TABLE 7. Software projects classified by type of development.

Type of development	Projects
New	481
Enhancement	1,517
Re-development	11

As the re-development projects in Table 7 consist of 11 projects, only the new and enhancement types were considered in our study. In Appendix A, each data set is classified by type of programming language, FSM, and resource level.

Twelve scatter plots (testing effort vs. FSM) were generated. They corresponded to the 12 data sets of Appendix A having a number of projects higher than or equal to 30. All of them had resource level type 1, which refers to people whose time is included in the reported effort data. ISBSG records four types of resource levels, and type 1 corresponds to the development team effort (the sum of efforts spent by the project team, project management, and project administration) [107].

The 12 scatter plots showed skewness, heteroscedasticity, and unusual values. Then, the χ^2, Shapiro-Wilk, skewness, and kurtosis statistical distribution tests were performed to test the effort and FSM variables using the data set, whose results are shown in Appendix B. Thus, data of all data sets were normalized by applying the natural logarithm (ln) [41], and the unusual values with studentized residuals greater than three in absolute value were excluded. Fig. 2 shows the scatter plot corresponding to the raw data of the 91 new projects sized in FISMA and coded in third generation programming languages, whereas Fig. 3 shows the normalized data set from Fig.2 including the trend line, and the four excluded unusual values (enclosed in red circles).

Table 8 lists the calculated values used for selecting seven final data sets used in our study. The types of FSM corresponds to IFPUGV4+ [108], FISMA [109], and NESMA [110]. Each data set was selected based on the following criteria.

1) A percentage of unusual values not higher than five [111].
2) Functional form data showing a positive correlation, that is, showing that the higher the value of size (i.e., FSM), the higher is the testing effort [112].
3) A coefficient of determination (r^2) higher than 0.5 should be considered acceptable [113].

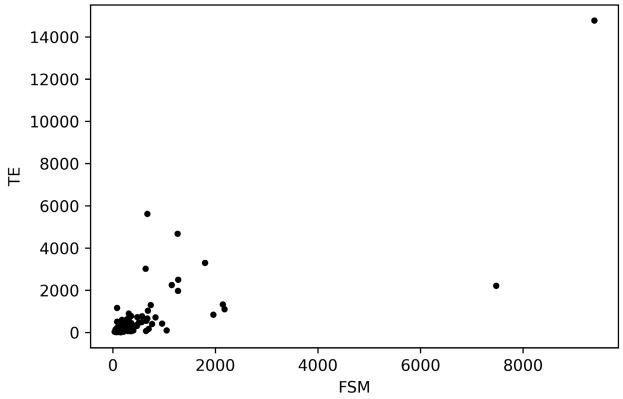


FIGURE 2. Raw data set of 91 new projects.

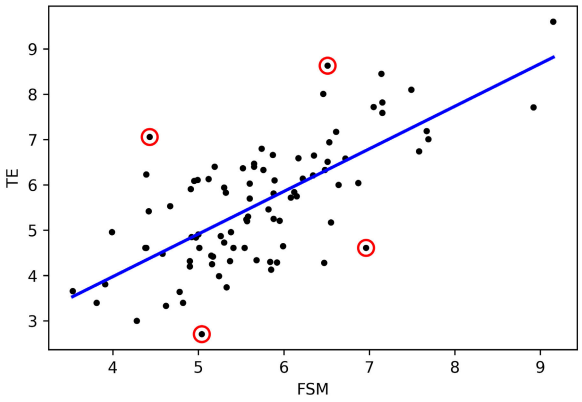


FIGURE 3. Normalized data set from Fig. 2.

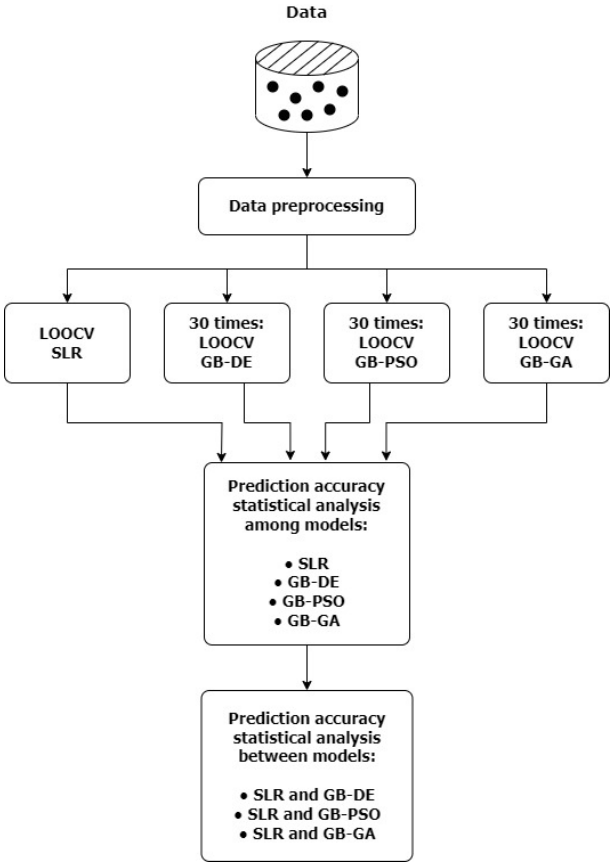


FIGURE 4. Diagram describing the procedure followed in the present study.

V. EMPIRICAL STUDY AND RESULTS

In the following subsections, we describe the setup of the experiments, performance of the metaheuristics, and results obtained.

A. EMPIRICAL STUDY SETUP

The DE, PSO, and GA metaheuristics were implemented to optimize the hyperparameters of the GB regressor to minimize the MAR objective function. Fig. 4 shows a diagram with the steps followed for the comparison of the described algorithms.

In this study, each metaheuristic was independently executed 30 times with a unique random number seed (calculating their statistical values in the process) as recommended in [114]. The more executions that are performed, the more robust the algorithm's performance.The DE, PSO, and GA metaheuristics were implemented to optimize the hyperparameters of the GB regressor to minimize the MAR objective function.

The empirical study was conducted on a platform using Python 3.9.13, a macOS High Sierra 10.13.6 operating system with an Intel Core i5 3.1 Ghz and 8GB RAM.

The parameters used for each metaheuristic are presented in Table 9, where it can be observed that for DE, we encourage the diversity of vectors in the population with a value of F = 0.8 (suggested in [69]) and a value of CR =0.5 (suggested in [59], [70]) promoting a balance of crossover probability.

B. METAHEURISTICS PERFORMANCE

Table 10 shows the descriptive statistics of the performance of each metaheuristic during 30 executions, optimizing the hyperparameters of the GB. In Table 10, the GB optimized through DE has the best results in all datasets with a lower MAR values were obtained from DE rather than with PSO or GA, and often with a greater standard deviation. In addition, despite performing 30 executions, the standard deviation of each metaheuristic was minimal. Thus, we can assume that, regardless of the randomness of these algorithms, the results are similar.

In Fig. 5 to 11, three of the 30 executions of each metaheuristic by data set are shown (the execution with the best MAR, the worst one and the median of the 30 executions). In general, PSO tends to converge prematurely, whereas for DE, it is possible to obtain better results and not get stuck in local optima. Moreover, Fig. 5 to 11 show that the best performance between PSO and GA depends on the data set because there is no metaheuristic that outperforms the other.

C. RESULTS

Table 11 shows the best Mean Absolute Residual (MAR) with its respective Median Absolute Residual (MdAR) using the model. This shows that GB-DE has better values than

TABLE 8. Coefficients of correlation (r) and determination (r^2) by data set.

Development type	Language type	Functional size method	Number of software projects	Unusual values	Studentized residual	Percentage of unusual values	Final number of software projects	r	r^2
Enhancement	3GL	FISMA	66	3	-2.19	4.5	63	0.7635	0.5829
	3GL	NESMA	375	14	2.56	4.0	361	0.8320	0.6923
	4GL	NESMA	625	22	-2.74	3.5	603	0.8262	0.6825
New	3GL	FISMA	91	4	-2.44	4.3	87	0.7635	0.5830
	3GL	NESMA	47	2	2.32	4.25	45	0.8253	0.6812
	4GL	IFPUG V4+	46	2	2.12	4.34	44	0.7511	0.5642
	4GL	NESMA	93	4	2.14	4.30	89	0.8463	0.7162

TABLE 9. Parameters setup for each metaherustic.

Differential Evolution	Particle Swarm Optimization	Genetic Algorithm
• F: 0.8 (suggested in [69]). • CR: 0.5 (suggested in [59] [70]). • Version: DE/rand/1/bin (best results obtained empirically in [115]).	• $C_1 = C_2 = 2$ (According to the result of prior studies [116] and replicated in [53]).	• Individual representation: real. • Selection method: deterministic tournament proposed in [105] from five randomly selected individuals. • Mutation: Uniform mutation. • Crossover: Intermediate recombination [117], with $k = 2$ and $a = 0.25$ (suggested in [106]). • Replacement: Elitist.

• Size Population: 30 (according to [103], the size of the population must be in the range $[5 * D, 10 * D]$ where D is the number of variables or dimensions and there must be at least four individuals).
• Number of generations: 20

GB-PSO, GB-GA, and SLR, except for the two MdARs for GB-PSO.

As mentioned in the Introduction, one MAR is not sufficient to establish a statistically significant improvement in the STEP. Therefore, a suitable statistical test was applied to compare the accuracies among the models [118]. Table 12 presents the χ^2, Shapiro-Wilk, skewness and kurtosis normality tests for each set of Absolute Residuals (AR) of the results in Table 11, and none of the sets of ARs meet the assumption of normality with 95% confidence. Accordingly, the Friedman non-parametric test was applied to determine whether at least one algorithm generated significantly different results from the others. All Friedman p-values included in Table 12 were less than 0.05. Thus, we can conclude that each data set of software projects has at least one statistically different prediction model.

The differences between the ARs of the projects were taken (SLR and GB-DE, SLR and GB-PSO, and SLR and GB-GA) with the objective of identifying the specific model with statistical differences. Next, the four normality statistical tests were performed for each difference of ARs: if any of their four p-values were lower than 0.05, then data were non-normally distributed at 95% confidence, and a Wilcoxon test was applied (the medians of the models in Table 11 should be compared); otherwise, a t-paired test was performed (the means of models of Table 11 were compared) [119] as shown in Table 13.

VI. DISCUSSION

The present study was motivated by the following issues:

- The ST aims to ensure that the software meets the established requirements and works correctly through extensive tests to identify and correct defects before the software is delivered to the final users.
- The STEP is relevant for software managers and software practitioners.
- Many studies have proposed models to predict SDLC effort. However, few studies have specifically focused on STEP.

Regarding the results, we can identify that GB-DE outperforms the results of SLR, although the transformed data have a linear relationship between the explanatory variable and the dependent variable, GB does not assume this type of relationship, making it more robust and flexible when combining multiple decision trees. Furthermore, GB is an ensemble algorithm that by combining several weak trees to build a stronger one, decreases the sensitivity to outlier data that a linear regression has and leads to a significant reduction in the prediction error. Also, by having a GB hyperparameter optimization process, the best values of all possible combinations were used to improve the results.

A. COMPARISON WITH PREVIOUS STUDIES

The related work section has highlighted the following seven issues:

1) There are only two studies similar to ours in the sense that they use a metaheuristic for optimizing the parameters of STEP models [19], [49]; however, none of them have proposed the application of DE, PSO

TABLE 10. Descriptive statistics of fitness function results through the 30 executions by each metaheuristic optimizing the hyperparameters of the GB by data set.

Development type	Number of software projects	Language type	Functional size method	GB-DE	GB-PSO	GB-GA
Enhancement	63	3GL	FISMA	Max = 0.66 Min = 0.55 Mean = 0.92 Median = 0.64 SD = 0.06	Max = 0.79 Min = 0.72 Mean = 0.75 Median = 0.76 SD = 0.03	Max = 0.75 Min = 0.69 Mean = 0.71 Median = 0.71 SD = 0.02
Enhancement	361	3GL	NESMA	Max = 0.46 Min = 0.43 Mean = 0.45 Median = 0.46 SD = 0.01	Max = 0.48 Min = 0.48 Mean = 0.48 Median = 0.48 SD = 0.00	Max = 0.48 Min = 0.47 Mean = 0.47 Median = 0.48 SD = 0.00
Enhancement	603	4GL	NESMA	Max = 0.52 Min = 0.52 Mean = 0.52 Median = 0.52 SD = 0.00	Max = 0.53 Min = 0.53 Mean = 0.53 Median = 0.53 SD = 0.00	Max = 0.53 Min = 0.53 Mean = 0.53 Median = 0.53 SD = 0.00
New	87	3GL	FISMA	Max = 0.68 Min = 0.67 Mean = 0.67 Median = 0.68 SD = 0.01	Max = 0.76 Min = 0.73 Mean = 0.74 Median = 0.75 SD = 0.01	Max = 0.71 Min = 0.70 Mean = 0.70 Median = 0.71 SD = 0.00
New	45	3GL	NESMA	Max = 0.63 Min = 0.61 Mean = 0.62 Median = 0.61 SD = 0.01	Max = 0.69 Min = 0.66 Mean = 0.67 Median = 0.67 SD = 0.01	Max = 0.67 Min = 0.64 Mean = 0.66 Median = 0.66 SD = 0.01
New	44	4GL	IFPUG V4+	Max = 0.67 Min = 0.60 Mean = 0.65 Median = 0.67 SD = 0.04	Max = 0.75 Min = 0.72 Mean = 0.74 Median = 0.74 SD = 0.01	Max = 0.80 Min = 0.74 Mean = 0.77 Median = 0.79 SD = 0.03
New	89	4GL	NESMA	Max = 0.43 Min = 0.43 Mean = 0.43 Median = 0.43 SD = 0.00	Max = 0.48 Min = 0.45 Mean = 0.46 Median = 0.46 SD = 0.02	Max = 0.45 Min = 0.44 Mean = 0.45 Median = 0.45 SD = 0.00

TABLE 11. Prediction accuracy by model (the values of GB-DE, GB-PSO and GB-GA correspond to the best MAR with its respective MdAR from Table 12).

TD	LT	FSM	NSP	SLR		GB-DE		GB-PSO		GB-GA	
				MAR	MdAR	MAR	MdAR	MAR	MdAR	MAR	MdAR
Enhancement	3GL	FISMA	63	0.67	0.60	0.55	0.37	0.72	0.58	0.69	0.47
	3GL	NESMA	361	0.47	0.40	0.43	0.28	0.48	0.38	0.47	0.40
	4GL	NESMA	603	0.56	0.45	0.52	0.40	0.53	0.39	0.53	0.42
New	3GL	FISMA	87	0.72	0.68	0.67	0.56	0.73	0.57	0.70	0.61
	3GL	NESMA	45	0.83	0.80	0.61	0.39	0.66	0.33	0.64	0.48
	4GL	IFPUG V4+	44	0.77	0.67	0.60	0.54	0.72	0.67	0.74	0.31
	4GL	NESMA	89	0.55	0.43	0.43	0.29	0.45	0.34	0.44	0.31

and GA metaheuristics for optimizing the parameters of Gradient Boosting, and only Cuckoo [19] and Bat [49] metaheuristics for optimizing the parameters of equations.

2) As for data sets (Table 3):

   a) None of the studies were specifically used for the testing phase (i.e., STEP) but predicted the development effort of the projects.
   b) None of them classified the projects based on their type of development, functional size method, or programming language.
   c) One of them [53] was artificially generated.
   d) They were obtained from projects not recently developed: COCOMO 81, Albrecht, Kemerer, and Desharnais, were published in 1981, 1983, 1987, and 1989, respectively. The NASA-93 and NASA-60 data sets were integrated for projects developed between 1971-1987, and between 1980 and 1990. The most recent had 24 projects published in 2011.
   e) Only two studies used the absolute residuals as prediction accuracy criteria.
   f) Only five of them reported the type of validation method used for training and testing the models, and none used a leave-one-out cross validation method.
   g) Only two studies based their conclusions on statistical differences.

TABLE 12. Normal statistical test by best result obtained from Table 12 for each metaheuristic.

Development type	Language type	Functional size method	Number of software projects	Data set by model	χ^2	S-W	Skewness	Kurtosis	Friedman p-value
Enhancement	3GL	FISMA	63	SLR	0.0428	0.0001	0.0171	0.4333	0.0182
				GB-DE	0.0349	0.0000	0.0113	0.5876	
				GB-PSO	0.0220	0.0018	0.0931	0.0283	
				GB-GA	0.0006	0.0000	0.0003	0.1648	
	3GL	NESMA	361	SLR	0.0000	0.0000	0.0000	0.9957	0.0005
				GB-DE	0.0000	0.0000	0.0000	0.4326	
				GB-PSO	0.0000	0.0000	0.0000	0.6989	
				GB-GA	0.0000	0.0000	0.0000	0.4469	
	4GL	NESMA	603	SLR	0.0000	0.0000	0.0000	0.3587	0.0104
				GB-DE	0.0000	0.0000	0.0000	0.1167	
				GB-PSO	0.0000	0.0000	0.0000	0.0000	
				GB-GA	0.0000	0.0000	0.0000	0.0000	
New	3GL	FISMA	87	SLR	0.1491	0.0088	0.0739	0.4340	0.0384
				GB-DE	0.0157	0.0000	0.0252	0.0691	
				GB-PSO	0.0169	0.0000	0.0049	0.6178	
				GB-GA	0.0075	0.0000	0.0021	0.5633	
	3GL	NESMA	45	SLR	0.5076	0.1810	0.2512	0.8424	0.0014
				GB-DE	0.0111	0.0001	0.0035	0.4822	
				GB-PSO	0.0001	0.0000	0.0002	0.0249	
				GB-GA	0.0012	0.0000	0.0008	0.1265	
	4GL	IFPUG V4+	44	SLR	0.1320	0.0262	0.1779	0.1349	0.0495
				GB-DE	0.0896	0.0081	0.0293	0.7847	
				GB-PSO	0.1617	0.0099	0.0628	0.6690	
				GB-GA	0.1194	0.0022	0.0560	0.4383	
	4GL	NESMA	89	SLR	0.0128	0.0003	0.0392	0.0345	0.0473
				GB-DE	0.0000	0.0000	0.0000	0.0184	
				GB-PSO	0.0000	0.0000	0.0000	0.0143	
				GB-GA	0.0000	0.0000	0.0000	0.0046	

TABLE 13. Statistical tests for data distribution between models by dataset.

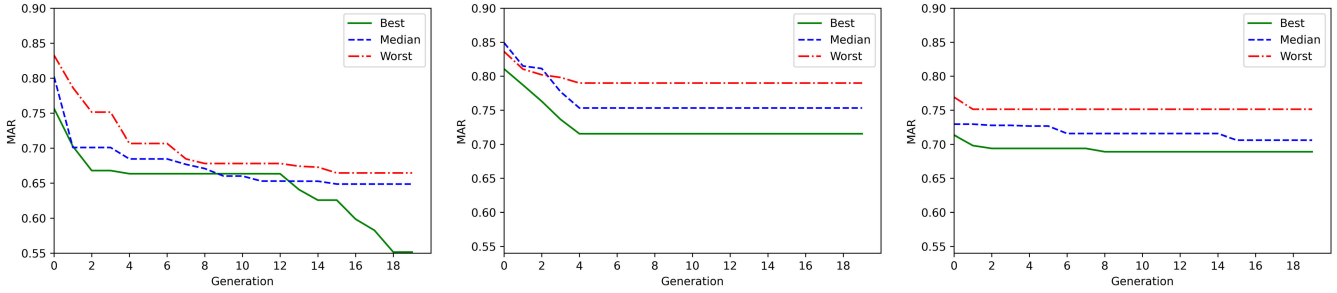
Development type	Language type	Functional size method	Number of software projects	Model compared with SLR	χ^2	S-W	Skewness	Kurtosis	Wilcoxon or t-paired p-values
Enhancement	3GL	FISMA	63	GB-DE	0.0739	0.0193	0.3561	0.0368	0.0440
				GB-PSO	0.0800	0.0086	0.8892	0.0249	0.3750
				GB-GA	0.0133	0.0039	0.3790	0.0050	0.6005
	3GL	NESMA	361	GB-DE	0.0000	0.0000	0.0011	0.0000	0.0483
				GB-PSO	0.0000	0.0000	0.7155	0.0000	0.0003
				GB-GA	0.0000	0.0000	0.0090	0.0000	0.6617
	4GL	NESMA	603	GB-DE	0.2654	0.1408	0.1981	0.3181	0.0456
				GB-PSO	0.0076	0.0108	0.0228	0.0326	0.0497
				GB-GA	0.2241	0.1677	0.2016	0.2435	0.0498
New	3GL	FISMA	87	GB-DE	0.0003	0.0036	0.0031	0.0071	0.0482
				GB-PSO	0.3271	0.4361	0.5055	0.1807	0.5716
				GB-GA	0.9525	0.8033	0.9701	0.7569	0.3917
	3GL	NESMA	45	GB-DE	0.0305	0.0199	0.0200	0.0211	0.0063
				GB-PSO	0.0281	0.0393	0.0189	0.2004	0.0482
				GB-GA	0.0637	0.0318	0.0397	0.2826	0.0239
	4GL	IFPUG V4+	44	GB-DE	0.7847	0.9469	0.7636	0.5300	0.0487
				GB-PSO	0.9624	0.8386	0.8390	0.8510	0.3511
				GB-GA	0.7954	0.5545	0.4995	0.9659	0.3936
	4GL	NESMA	89	GB-DE	0.2848	0.5294	0.1626	0.4535	0.0491
				GB-PSO	0.3932	0.4700	0.2080	0.5957	0.0492
				GB-GA	0.2779	0.5286	0.1758	0.3935	0.0473

3) Regarding DE (Table 4) - Experimental setup:
   a) Ten studies reported parameters used in the implementation of the DE algorithm.
   b) Six of them reported the version of DE used.
4) Regarding DE (Table 4) - Performance:
   a) Only seven studies reported the performance of the algorithm, either with convergence graphs, descriptive statistics, or Pareto frontiers.
   b) Only six studies have reported the execution of the DE algorithm several times to determine its performance.

B. LIMITATIONS

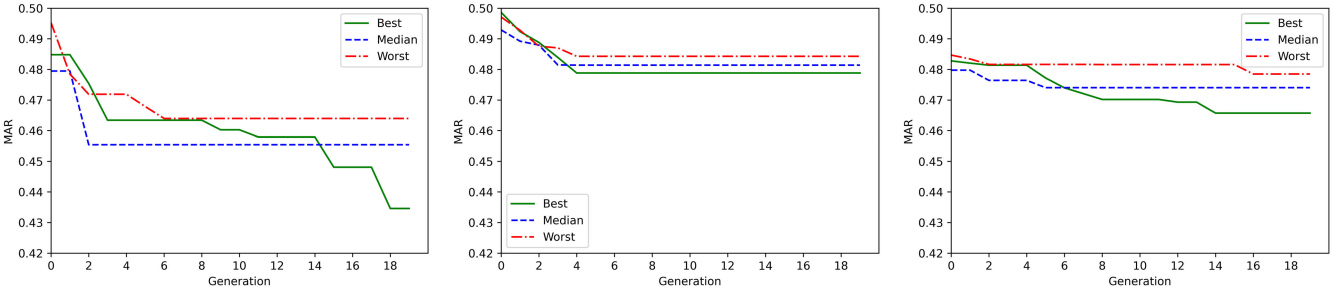
The limitations in this research work are the following:

- ISBSG Release 2022 includes 11,281 software projects; however, we could only select 1,292 projects in seven



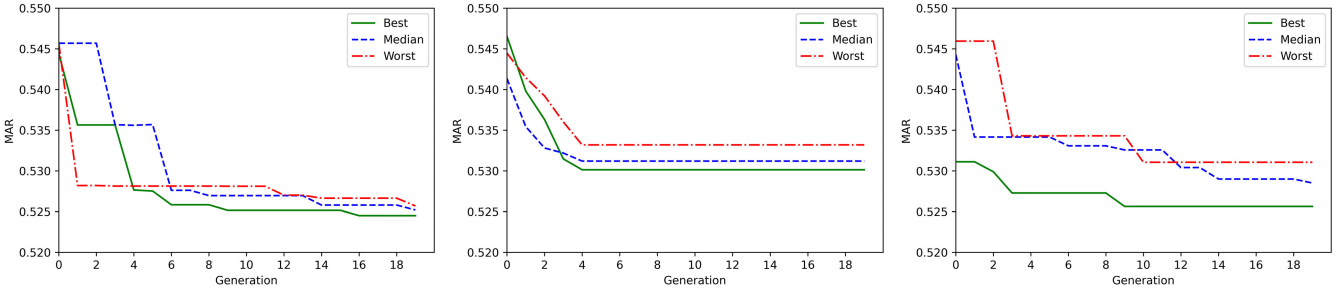
(a) DE Performance (b) PSO Performance (c) GA Performance

FIGURE 5. Convergence plots of executions for the data set: Enhancement, 3GL, FISMA (N = 63 projects).



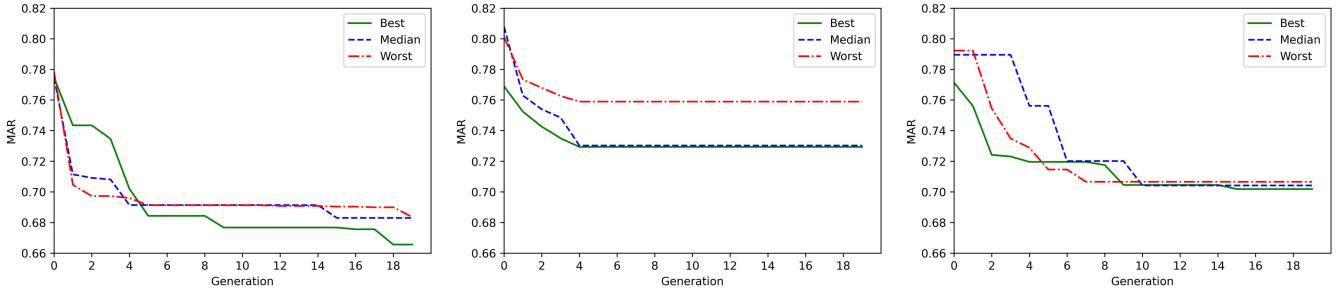
(a) DE Performance (b) PSO Performance (c) GA Performance

FIGURE 6. Convergence plots of executions for the data set: Enhancement, 3GL, NESMA (N = 631 projects).



(a) DE Performance (b) PSO Performance (c) GA Performance

FIGURE 7. Convergence plots of executions for the data set: Enhancement, 4GL, NESMA (N = 603 projects).

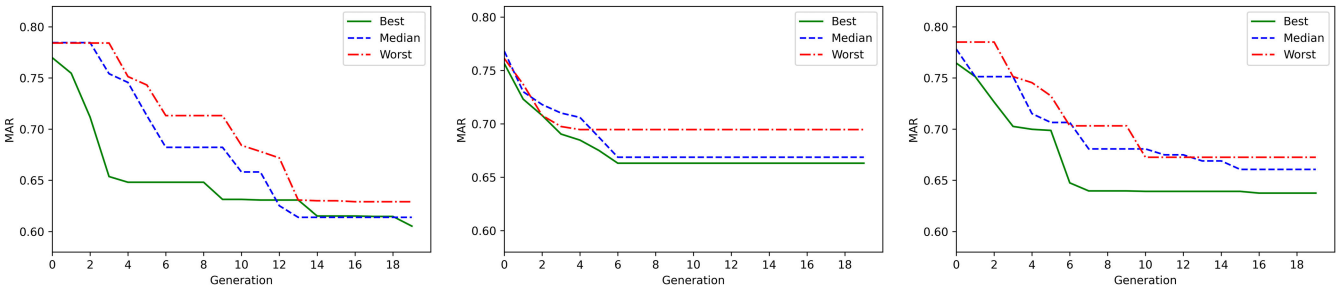


(a) DE Performance (b) PSO Performance (c) GA Performance

FIGURE 8. Convergence plots of executions for the data set: New, 3GL, FISMA (N = 87 projects).

data sets by observing the criteria suggested for the ISBSG guidelines (described in Section IV).

- We only used three types of FSM as the explanatory variable (i.e., FISMA, NESMA, IFPUG V4+).
- We implemented only in a single version that the three metaheuristics used for hyperparameter optimization; that is, the variants of these metaheuristics were not explored, nor were they exhaustively compared to other metaheuristics.
- We did not optimize the hyperparameters of the three metaheuristics used; but they were used in the values suggested in Table 9.



(a) DE Performance (b) PSO Performance (c) GA Performance

FIGURE 9. Convergence plots of executions for the data set: New, 3GL, NESMA (N = 45 projects).

TABLE 14. Validity threats addressed in our work.

Category	Description	How was addressed in our study
C	Is there any statistical verification of the results?	All our conclusions were based on the results of statistical tests.
C	Does the study verify all the assumptions of the chosen statistical test?	We selected the statistical tests based on the number of data sets to be compared, data dependence, and data distribution.
C	Does the study account for randomness of search-based approaches (SBAs)?	To deal with randomness, the experiments were executed 30 times independently as described in "V-A. Experiments setup".
C	Do the results of the study take into account validation bias if any?	To reduce any bias in the validation, a leave-one-out cross validation method was used to avoid the randomness for selecting the data sets to be used for training and testing the models.
I	Does the study perform appropriate steps for tuning the parameters of search-based approaches (SBAs)?	The parameters for the search-based approaches (SBAs) were selected in accordance with the criteria suggested in empirical studies included in the "II Related work" section.
I	Does the study take steps to effectively use inconsistent or noisy data?	The process for handling unusual values is described in "IV. Datasets and projects".
I	Does the study use attribute selection methods to eliminate noisy and redundant attributes?	The process for selecting attributes is described in "IV. Datasets and projects".
I	Does the study ensure that there is no confounding effect on the relationship between the explanatory and the dependent variables due to extraneous attributes?	According to [30], the size is the variable that best represents the behavior of the effort; therefore, it is the most used variable to predict the effort of software projects.
CT	Does the study use effective representatives of the concepts which represent explanatory variables?	We used FSM as explanatory variable because FSM can be measured from the requirements phase, rather than that source lines of code, which depends on the programming language and a coding standard [122].
CT	Does the study use performance measures which are effective representatives of the capability of the developed models?	We used the Mean Absolute Residual (MAR) whose advantages were described in "III-B1. fitness function" section.
CT	Does the study take necessary steps in order to overcome human errors while data collection?	We selected those projects best ranked (i.e., "A" and "B" in Table 6) in accordance with the ISBSG guidelines.
ET	Are the datasets used by the study real industrial datasets?	The data was obtained from the ISBSG 2022 data set, which contains data from real software projects in the industry.
ET	Are the datasets used in the study appropriate in number?	The number of datasets is appropriate since they have the minimum required for statistical tests as mentioned in "IV. Datasets and projects".
ET	Does the study use appropriately sized or varied sized datasets?	We used seven data sets, containing from 44 to 603 projects.
ET	Are the datasets used in the study developed using different programming languages?	The programming languages used to code the projects were 3GL and 4GL.
ET	Do the datasets used in the study belong to different companies/domains or have different characteristics?	The projects collected by ISBSG belong to different companies as well as different industrial sectors (such as banking, communication, construction, education, financial, government, among others) and application groups (such as business applications, real-time applications, mathematically intensive application or infrastructure software).
ET	Does the study provide proper details for replicability?	The process used for data selection, preprocessing and processing, as well as implementation of metaheuristics and results are detailed for replicability.

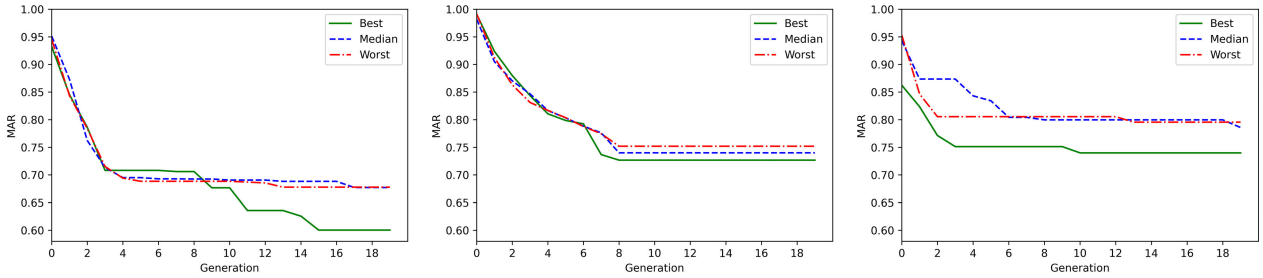
C: Conclusion, I: Internal, CT: Construct, ET: External

C. VALIDITY THREATS

There are four categories of valid threats in search-based predictive modeling for Software Engineering: conclusion, internal, external and construct [120]. The descriptions of the 17 validity threats, as well as the way in which they were addressed in our work are presented in Table 14.

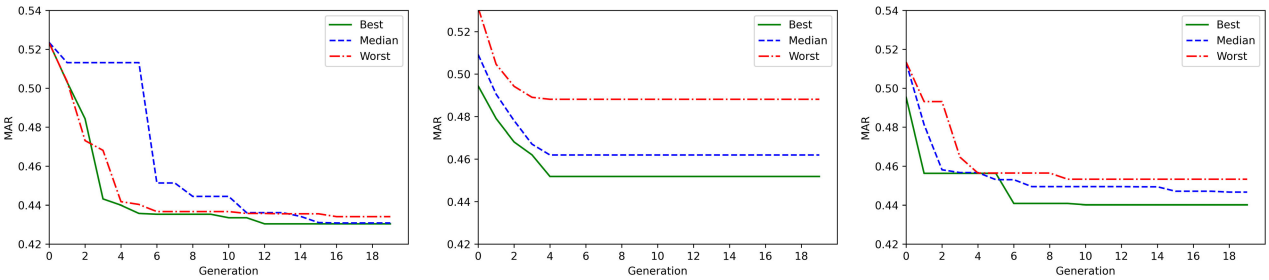
In addition to the validity threats listed in Table 14, we identified the following:

- Internal: The ISBSG suggests using projects with a rating code of "A" or "B" applied to the functional size to build prediction models. However, the ISBSG does not report the method used to to suggest such categories. Moreover, we excluded unusual values based only on mathematical criteria. We are aware that to exclude an extreme value, it is also necessary to know the factors that strong influence



(a) DE Performance (b) PSO Performance (c) GA Performance

FIGURE 10. Convergence plots of executions for the data set: New, 4GL, IFPUG V4+ (N = 44 projects).



(a) DE Performance (b) PSO Performance (c) GA Performance

FIGURE 11. Convergence plots of executions for the data set: New, 4GL, NESMA (N = 89 projects).

on the value [121]; however, we were not able to know it.

- External: The accuracy of the prediction given projects not used to train the models depend on the specification of the software requirements. This is because, in the requirements stage, the size of the projects is estimated, which is the explanatory variable in the generated models to predict the software testing effort.
- Construct: The values of the dependent and explanatory variables used to train and test the models are transformed.

VII. CONCLUSION

In accordance with the results in Table 13, we can reject the null hypothesis H_{DE0}, and accept the alternative H_{DE1} for all of the datasets evaluated for GB-DE with a confidence of 95%.

As for GB – PSO, we reject the null hypothesis H_{PSO0} for the four data sets, and accept the alternative H_{PSO1} at 95% confidence.

Regarding GB-GA, we reject the null hypothesis H_{GA0} for the three data sets, and accept alternative H_{GA1} at 95% confidence.

Based on the results presented in Tables 11 and 13, the GB-DE can be used to STEP using FSM as explanatory variable of the following projects:

1) New software projects:
   - Coded in 3GL and measured with FISMA or NESMA.
   - Coded in 4GL and measured with IFPUG V4+ or NESMA.

TABLE 15. Number of projects for testing effort analysis.

Development type	Languaje type	Functional size method	Resource Level	Number of projects
Enhancement	2GL	IFPUG V4+	1	3
	3GL	FISMA	1	66
	3GL	IFPUG V4+	1	211
	3GL	NESMA	1	375
	3GL	IFPUG V4+	2	2
	3GL	NESMA	2	3
	3GL	FISMA	4	3
	3GL	IFPUG V4+	4	9
	3GL	NESMA	4	3
	4GL	FISMA	1	9
	4GL	IFPUG V4+	1	93
	4GL	NESMA	1	625
	4GL	IFPUG V4+	4	4
	5GL	IFPUG V4+	1	1
	5GL	NESMA	1	42
	ApG	FISMA	1	14
	ApG	IFPUG V4+	1	53
	ApG	IFPUG V4+	2	1
New	2GL	IFPUG V4+	1	2
	3GL	FISMA	1	91
	3GL	IFPUG V4+	1	121
	3GL	NESMA	1	47
	3GL	IFPUG V4+	2	8
	3GL	IFPUG V4+	3	1
	3GL	FISMA	4	5
	3GL	IFPUG V4+	4	25
	4GL	FISMA	1	18
	4GL	IFPUG V4+	1	46
	4GL	NESMA	1	93
	4GL	IFPUG V4+	2	4
	4GL	NESMA	3	1
	4GL	IFPUG V4+	2	5
	4GL	NESMA	4	3
	ApG	FISMA	1	6
	ApG	IFPUG V4+	1	2
	ApG	IFPUG V4+	2	2
	ApG	IFPUG V4+	4	1

2) Software enhancement projects:
   - Coded in 3GL and measured with FISMA or NESMA.

TABLE 16. Normal statistical tests applied to 14 data sets with projects greater than 30.

Development type	Language type	Functional size method	Number of software projects	Variable	χ_2	S-W	Skewness	Kurtosis
Enhancement	3GL	FISMA	66	TE	0.0000	0.0000	0.0000	0.0000
				FSM	0.0000	0.0000	0.0000	0.0000
	3GL	IFPUG V4+	211	TE	0.0000	0.0000	0.0000	0.0000
				FSM	0.0000	0.0000	0.0000	0.0000
	3GL	NESMA	375	TE	0.0000	0.0000	0.0000	0.0000
				FSM	0.0000	0.0000	0.0000	0.0098
	4GL	IFPUG V4+	93	TE	0.0000	0.0000	0.0000	0.0000
				FSM	0.0000	0.0000	0.0000	0.0000
	4GL	NESMA	625	TE	0.0000	0.0000	0.0000	0.0000
				FSM	0.0000	0.0000	0.0000	0.0081
	5GL	NESMA	42	TE	0.0411	0.0005	0.0248	0.0426
				FSM	0.4013	0.0015	0.2553	0.9400
	ApG	IFPUG V4+	53	TE	0.0000	0.0000	0.0000	0.0000
				FSM	0.0000	0.0000	0.0000	0.0000
New	3GL	FISMA	91	TE	0.0000	0.0000	0.0000	0.0000
				FSM	0.0000	0.0000	0.0000	0.0000
	3GL	IFPUG V4+	121	TE	0.0000	0.0000	0.0000	0.0000
				FSM	0.0000	0.0000	0.0000	0.0000
	3GL	NESMA	47	TE	0.0000	0.0000	0.0001	0.0000
				FSM	0.0000	0.0000	0.0008	0.0008
	4GL	IFPUG V4+	46	TE	0.0000	0.0000	0.0000	0.0000
				FSM	0.0000	0.0000	0.0001	0.0000
	4GL	NESMA	93	TE	0.0000	0.0000	0.0000	0.0000
				FSM	0.0000	0.0000	0.0000	0.0000

- Coded in 4GL and measured with NESMA.

We can conclude that the GB-DE can be used by software managers and software practitioners for STEP of either new projects or enhancement projects, developed in either third or fourth programming language generation. Thus, the practical applications of GB-DE for software project managers could allow obtaining a more accurate STEP to allocate the suitable teams resources and budget for testing activities.

As future work the following elements are proposed:

- The exploration of other Boosting algorithms for regression, such as Histogram Based Gradient Boosting, reduce the errors made by a previous single model.
- Consideration of other Bagging algorithms such as Random Forest, which allows the reduction of variance, where simple algorithms are used in parallel with the objective of taking advantage of the independence between the single algorithms, reducing the error when averaging the predictions.
- The study of other metaheuristics inspired by physical phenomena, such as simulated annealing or hill climbing, to optimize the parameters of a regression algorithm.
- Implementation of hyperparameter optimization for metaheuristics implemented to optimize the parameters of any regressor.
- Comparisons between different versions of metaheuristics to conclude the version that adapts to the characteristics of the regression algorithm used.
- Application of Boosting, Bagging, and metaheuristics to predict the effort of other software process phases, such as construction or maintenance.
- The models used in this study applied without non-transformed data.

APPENDIX A
NUMBER OF PROJECTS FOR TESTING EFFORT ANALYSIS

Table 15 classifies each data set of the 481 and 1,517 projects in Table 7 by type of programming language, FSM, and resource level. Because of the minimum data set size required for the statistical tests used in the present study, we only selected those data sets from Table 15 whose size was higher than or equal to 30 software projects, that is, five and seven data sets for new and enhancement types, respectively.

APPENDIX B
NORMAL STATISTICAL TESTS APPLIED TO 14 DATA SETS

Table 16 shows that the smallest p-value among the tests performed on the data set was less than 0.01. Therefore, it can be rejected that the testing effort and FSM come from a normal distribution with 99% confidence for the 24 cases.

ACKNOWLEDGMENT

The authors would like to thank Facultad de Estadística e Informática, Universidad Veracruzana, México; Universidad de Guadalajara, México; Consejo Nacional de Humanidades, Ciencias y Tecnologías (Conahcyt) México; and École de technologie supérieure, Université du Québec, Canada.

REFERENCES

[1] P. Bourque and R. Fairley, *Guide to the Software Engineering Body of Knowledge*, document SWEBOK Version 3.0, IEEE Computer Society, Washington, DC, USA, 2014.

[2] H. Ji, S. Huang, Y. Wu, Z. Hui, and C. Zheng, "A new weighted naive Bayes method based on information diffusion for software defect prediction," *Softw. Quality J.*, vol. 27, no. 3, pp. 923–968, Sep. 2019.

[3] J. J. Li, A. Ulrich, X. Bai, and A. Bertolino, "Advances in test automation for software with special focus on artificial intelligence and machine learning," *Softw. Quality J.*, vol. 28, no. 1, pp. 245–248, Mar. 2020.
[4] H. Yenigun, N. Yevtushenko, and A. R. Cavalli, "Guest editorial: Special issue on testing software and systems," *Softw. Quality J.*, vol. 27, no. 2, pp. 497–499, Jun. 2019.
[5] A. Sharma and D. S. Kushwaha, "An empirical approach for early estimation of software testing effort using SRS document," *CSI Trans. ICT*, vol. 1, no. 1, pp. 51–66, Mar. 2013.
[6] É. R. C. de Almeida, B. T. de Abreu, and R. Moraes, "An alternative approach to test effort estimation based on use cases," in *Proc. Int. Conf. Softw. Test. Verification Validation*, Apr. 2009, pp. 279–288.
[7] E. Aranha and P. Borba, "An estimation model for test execution effort," in *Proc. 1st Int. Symp. Empirical Softw. Eng. Meas. (ESEM)*, Sep. 2007, pp. 1–9.
[8] Z. Li, X.-Y. Jing, and X. Zhu, "Progress on approaches to software defect prediction," *IET Softw.*, vol. 12, no. 3, pp. 161–175, Jun. 2018.
[9] A. Kaur and K. Kaur, "Investigation on test effort estimation of mobile applications: Systematic literature review and survey," *Inf. Softw. Technol.*, vol. 110, pp. 56–77, Jun. 2019.
[10] E. Aranha and P. Borba, "Estimating manual test execution effort and capacity based on execution points," *Int. J. Comput. Appl.*, vol. 31, no. 3, pp. 167–172, Jan. 2009.
[11] K. R. Jayakumar and A. Abran, "A survey of software test estimation techniques," *J. Softw. Eng. Appl.*, vol. 6, no. 10, pp. 47–52, 2013.
[12] D. G. E. Silva, M. Jino, and B. T. D. Abreu, "Machine learning methods and asymmetric cost function to estimate execution effort of software testing," in *Proc. 3rd Int. Conf. Softw. Test., Verification Validation*, Apr. 2010, pp. 275–284.
[13] V. Nguyen, V. Pham, and V. Lam, "qEstimation: A process for estimating size and effort of software testing," in *Proc. Int. Conf. Softw. Syst. Process*, May 2013, pp. 20–28.
[14] S. Mensah, J. Keung, K. E. Bennin, and M. F. Bosu, "Multi-objective optimization for software testing effort estimation," in *Proc. Int. Conf. Softw. Eng. Knowl. Eng.*, Jul. 2016, pp. 527–530.
[15] K. Punitha and S. Chitra, "Software defect prediction using software metrics—A survey," in *Proc. Int. Conf. Inf. Commun. Embedded Syst. (ICICES)*, Chennai, India, Feb. 2013, pp. 555–558.
[16] C. Abhishek, V. P. Kumar, H. Vitta, and P. R. Srivastava, "Test effort estimation using neural network," *J. Softw. Eng. Appl.*, vol. 3, no. 4, pp. 331–340, 2010.
[17] S. Aloka, P. Singh, G. Rakshit, and P. R. Srivastava, "Test effort estimation-particle swarm optimization based approach," in *Proc. 4th Int. Conf. Contemp. Comput.*, 2011, pp. 463–474.
[18] P. R. Srivastava, S. Kumar, A. Singh, and G. Raghurama, "Software testing effort: An assessment through fuzzy criteria approach," *J. Uncertain Syst.*, vol. 5, no. 3, pp. 183–201, Aug. 2011.
[19] P. R. Srivastava, A. Varshney, P. Nama, and X. S. Yang, "Software test effort estimation: A model based on cuckoo search," *Int. J. Bio-Inspired Comput.*, vol. 4, no. 5, pp. 278–285, 2012.
[20] P. Bhattacharya, P. R. Srivastava, and B. Prasad, "Software test effort estimation using particle swarm optimization," in *Proc. Int. Conf. Inf. Syst. Design Intell. Appl.* (Advances in Intelligent and Soft Computing), 2012, pp. 827–835.
[21] F. Calzolari, P. Tonella, and G. Antoniol, "Maintenance and testing effort modeled by linear and nonlinear dynamic systems," *Inf. Softw. Technol.*, vol. 43, no. 8, pp. 477–486, Jul. 2001.
[22] C.-T. Lin and C.-Y. Huang, "Enhancing and measuring the predictive capabilities of testing-effort dependent software reliability models," *J. Syst. Softw.*, vol. 81, no. 6, pp. 1025–1038, Jun. 2008.
[23] R. Tiwari and N. Goel, "Reuse: Reducing test effort," *ACM SIGSOFT Softw. Eng. Notes*, vol. 38, no. 2, pp. 1–11, Mar. 2013.
[24] D. S. Kushwaha and A. K. Misra, "Software test effort estimation," *ACM SIGSOFT Softw. Eng. Notes*, vol. 33, no. 3, pp. 1–5, May 2008.
[25] Y. Singh, A. Kaur, and R. Malhotra, "Predicting testing effort using artificial neural network," in *Proc. World Congr. Eng. Comput. Sci. (WCECS)*, 2008, pp. 1–6.
[26] M. Jorgensen and M. Shepperd, "A systematic review of software development cost estimation studies," *IEEE Trans. Softw. Eng.*, vol. 33, no. 1, pp. 33–53, Jan. 2007.
[27] J. Wen, S. Li, Z. Lin, Y. Hu, and C. Huang, "Systematic literature review of machine learning based software development effort estimation models," *Inf. Softw. Technol.*, vol. 54, no. 1, pp. 41–59, Jan. 2012.
[28] V. K. Bardsiri, D. N. A. Jawawi, and E. Khatibi, "Towards improvement of analogy-based software development effort estimation: A review," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 24, no. 7, pp. 1065–1089, Sep. 2014.
[29] A. Idri, F. A. Amazal, and A. Abran, "Analogy-based software development effort estimation: A systematic mapping and review," *Inf. Softw. Technol.*, vol. 58, pp. 206–230, Feb. 2015.
[30] S. S. Gautam and V. Singh, "The state-of-the-art in software development effort estimation," *J. Softw., Evol. Process*, vol. 30, no. 12, Dec. 2018, Art. no. e1983.
[31] A. Ali and C. Gravino, "A systematic literature review of software effort prediction using machine learning methods," *J. Softw., Evol. Process*, vol. 31, no. 10, Oct. 2019, Art. no. e2245.
[32] Y. Mahmood, N. Kama, and A. Azmi, "A systematic review of studies on use case points and expert-based estimation of software development effort," *J. Softw., Evol. Process*, vol. 32, no. 7, Jul. 2020, Art. no. e2245.
[33] C. E. Carbonera, K. Farias, and V. Bischoff, "Software development effort estimation: A systematic mapping study," *IET Softw.*, vol. 14, no. 4, pp. 328–344, Aug. 2020.
[34] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Trans. Evol. Comput.*, vol. 1, no. 1, pp. 67–82, Apr. 1997.
[35] X. Lu, W. Zhou, X. Ding, X. Shi, B. Luan, and M. Li, "Ensemble learning regression for estimating unconfined compressive strength of cemented paste backfill," *IEEE Access*, vol. 7, pp. 72125–72133, 2019.
[36] I. K. Nti, A. F. Adekoya, and B. A. Weyori, "A comprehensive evaluation of ensemble learning for stock-market prediction," *J. Big Data*, vol. 7, no. 1, pp. 1–40, Dec. 2020.
[37] H. T. Hoc, R. Silhavy, Z. Prokopova, and P. Silhavy, "Comparing stacking ensemble and deep learning for software project effort estimation," *IEEE Access*, vol. 11, pp. 60590–60604, 2023.
[38] S. S. Ali, J. Ren, K. Zhang, J. Wu, and C. Liu, "Heterogeneous ensemble model to optimize software effort estimation accuracy," *IEEE Access*, vol. 11, pp. 27759–27792, 2023.
[39] V. H. Cantú, C. Azzaro-Pantel, and A. Ponsich, "Constraint-handling techniques within differential evolution for solving process engineering problems," *Appl. Soft Comput.*, vol. 108, Sep. 2021, Art. no. 107442.
[40] D. Alekseeva, N. Stepanov, A. Veprev, A. Sharapova, E. S. Lohan, and A. Ometov, "Comparison of machine learning techniques applied to traffic prediction of real wireless network," *IEEE Access*, vol. 9, pp. 159495–159514, 2021.
[41] B. Kitchenham and E. Mendes, "Why comparative effort prediction studies may be invalid," in *Proc. 5th Int. Conf. Predictor Models Softw. Eng.*, May 2009, pp. 1–5.
[42] *ISBSG Demographics*, Int. Softw. Benchmarking Standards Group, Melbourne, VIC, Australia, 2022.
[43] K. K. Shukla, "Neuro-genetic prediction of software development effort," *Inf. Softw. Technol.*, vol. 42, no. 10, pp. 701–713, Jul. 2000.
[44] S.-J. Huang and N.-H. Chiu, "Optimization of analogy weights by genetic algorithm for software effort estimation," *Inf. Softw. Technol.*, vol. 48, no. 11, pp. 1034–1045, Nov. 2006.
[45] S.-J. Huang, N.-H. Chiu, and L.-W. Chen, "Integration of the grey relational analysis with genetic algorithm for software effort estimation," *Eur. J. Oper. Res.*, vol. 188, no. 3, pp. 898–909, Aug. 2008.
[46] D. Wu, J. Li, and Y. Liang, "Linear combination of multiple case-based reasoning with optimized weight for software effort estimation," *J. Supercomput.*, vol. 64, no. 3, pp. 898–918, Jun. 2013.
[47] A. Corazza, S. Di Martino, F. Ferrucci, C. Gravino, F. Sarro, and E. Mendes, "Using Tabu search to configure support vector regression for effort estimation," *Empirical Softw. Eng.*, vol. 18, no. 3, pp. 506–546, Jun. 2013.
[48] I. Thamarai and S. Murugavalli, "Advantage of using evolutionary computation algorithm in software effort estimation," *Int. J. Appl. Eng. Res.*, vol. 9, no. 4, pp. 30167–30178, Jan. 2014.
[49] P. R. Srivastava, A. Bidwai, A. Khan, K. Rathore, R. Sharma, and X. S. Yang, "An empirical study of test effort estimation based on bat algorithm," *Int. J. Bio-Inspired Comput.*, vol. 6, no. 1, pp. 57–70, 2014.
[50] T. Urbanek, Z. Prokopova, R. Silhavy, and V. Vesela, "Prediction accuracy measurements as a fitness function for software effort estimation," *SpringerPlus*, vol. 4, no. 1, pp. 1–17, Dec. 2015.
[51] I. Thamarai and S. Murugavalli, "Analogy based software effort estimation based on differential evolution and hybrid fuzzy logic and firefly algorithm," *Asian J. Inf. Technol.*, vol. 15, no. 9, pp. 1484–1493, 2016.

[52] I. Thamarai and S. Murugavalli, "An evolutionary computation approach for project selection in analogy based software effort estimation," *Indian J. Sci. Technol.*, vol. 9, no. 21, pp. 1–6, Jun. 2016.

[53] A. K. Bardsiri and S. M. Hashemi, "A differential evolution-based model to estimate the software services development effort," *J. Softw., Evol. Process*, vol. 28, no. 1, pp. 57–77, Jan. 2016.

[54] S. M. S. Jafari and F. Ziaaddini, "Optimization of software cost estimation using harmony search algorithm," in *Proc. 1st Conf. Swarm Intell. Evol. Comput. (CSIEC)*, Bam, Iran, Mar. 2016, pp. 131–135.

[55] I. Thamarai and S. Murugavalli, "A study to improve the software estimation using differential evolution algorithm with analogy," *J. Theor. Appl. Inf. Technol.*, vol. 95, no. 20, pp. 5587–5597, Oct. 2017.

[56] A. Wadhwa, S. Jain, and C. Gupta, "An effective precision enhancement approach to estimate software development cost: Nature inspired way," *J. Telecommun., Electron. Comput. Eng.*, vol. 9, no. 3, pp. 85–91, 2017.

[57] S. H. S. Moosavi and V. K. Bardsiri, "Satin bowerbird optimizer: A new optimization algorithm to optimize ANFIS for software development effort estimation," *Eng. Appl. Artif. Intell.*, vol. 60, pp. 1–15, Apr. 2017.

[58] T. T. Khuat and M. H. Le, "A novel hybrid ABC-PSO algorithm for effort estimation of software projects using agile methodologies," *J. Intell. Syst.*, vol. 27, no. 3, pp. 489–506, Jul. 2018.

[59] T. R. Benala and R. Mall, "DABE: Differential evolution in analogy-based software development effort estimation," *Swarm Evol. Comput.*, vol. 38, pp. 158–172, Feb. 2018.

[60] M. S. Khan, C. A. ul Hassan, M. A. Shah, and A. Shamim, "Software cost and effort estimation using a new optimization algorithm inspired by strawberry plant," in *Proc. 24th Int. Conf. Autom. Comput. (ICAC)*, Newcastle Upon Tyne, U.K., Sep. 2018, pp. 1–6.

[61] V. Resmi, S. Vijayalakshmi, and R. S. Chandrabose, "An effective software project effort estimation system using optimal firefly algorithm," *Cluster Comput.*, vol. 22, no. 5, pp. 11329–11338, Sep. 2019.

[62] R. Marco, "Optimizing software effort estimation models based on metaheuristic methods: A proposed framework," *Int. J. Adv. Trends Comput. Sci. Eng.*, vol. 8, no. 1.5, pp. 294–304, Nov. 2019.

[63] A. Ullah, B. Wang, J. Sheng, J. Long, M. Asim, and Z. Sun, "Optimization of software cost estimation model based on biogeography-based optimization algorithm," *Intell. Decis. Technol.*, vol. 14, no. 4, pp. 441–448, Jan. 2021.

[64] A. A. Fadhil, R. G. H. Alsarraj, and A. M. Altaie, "Software cost estimation based on dolphin algorithm," *IEEE Access*, vol. 8, pp. 75279–75287, 2020.

[65] S. Chhabra and H. Singh, "Optimizing design of fuzzy model for software cost estimation using particle swarm optimization algorithm," *Int. J. Comput. Intell. Appl.*, vol. 19, no. 1, Mar. 2020, Art. no. 2050001.

[66] M. D. Alanis-Tamez, C. López-Martín, and Y. Villuendas-Rey, "Particle swarm optimization for predicting the development effort of software projects," *Mathematics*, vol. 8, no. 10, p. 1819, Oct. 2020.

[67] A. Kaushik, D. K. Tayal, and K. Yadav, "The role of neural networks and metaheuristics in agile software development effort estimation," *Int. J. Inf. Technol. Project Manage.*, vol. 11, no. 2, pp. 50–71, Apr. 2020.

[68] S. P. Singh, G. Dhiman, P. Tiwari, and R. H. Jhaveri, "A soft computing based multi-objective optimization approach for automatic prediction of software cost models," *Appl. Soft Comput.*, vol. 113, Dec. 2021, Art. no. 107981.

[69] Z. H. Wani, J. I. Bhat, and K. J. Giri, "A generic analogy-centered software cost estimation based on differential evolution exploration process," *Comput. J.*, vol. 64, no. 1, pp. 462–472, Nov. 2019.

[70] A. Karimi and T. J. Gandomani, "Software development effort estimation modeling using a combination of fuzzy-neural network and differential evolution algorithm," *Int. J. Electr. Comput. Eng. (IJECE)*, vol. 11, no. 1, pp. 707–715, Feb. 2021.

[71] M. K. Parizi, F. Keynia, and A. K. Bardsiri, "HSCWMA: A new hybrid SCA-WMA algorithm for solving optimization problems," *Int. J. Inf. Technol. Decis. Making*, vol. 20, no. 2, pp. 775–808, Mar. 2021.

[72] M. S. Khan, F. Jabeen, S. Ghouzali, Z. Rehman, S. Naz, and W. Abdul, "Metaheuristic algorithms in optimizing deep neural network model for software effort estimation," *IEEE Access*, vol. 9, pp. 60309–60327, 2021.

[73] A. Puspaningrum, F. P. B. Muhammad, and E. Mulyani, "Flower pollination algorithm for software effort coefficients optimization to improve effort estimation accuracy," *Jurnal Informatika*, vol. 9, no. 2, p. 139, Nov. 2021.

[74] C. A. Ul Hassan and M. S. Khan, "An effective nature inspired approach for the estimation of software development cost," in *Proc. 16th Int. Conf. Emerg. Technol. (ICET)*, Islamabad, Pakistan, Dec. 2021, pp. 1–6.

[75] Z. Shahpar, V. K. Bardsiri, and A. K. Bardsiri, "Polynomial analogy-based software development effort estimation using combined particle swarm optimization and simulated annealing," *Concurrency Comput., Pract. Exper.*, vol. 33, no. 20, Oct. 2021, Art. no. e6358.

[76] S. P. Singh, V. P. Singh, and A. K. Mehta, "Differential evolution using homeostasis adaption based mutation operator and its application for software cost estimation," *J. King Saud Univ. Comput. Inf. Sci.*, vol. 33, no. 6, pp. 740–752, Jul. 2021.

[77] S. K. Gouda and A. K. Mehta, "A new evolutionism based self-adaptive multi-objective optimization method to predict software cost estimation," *Softw., Pract. Exper.*, vol. 52, no. 8, pp. 1826–1848, Aug. 2022.

[78] S. K. Gouda and A. K. Mehta, "Software cost estimation model based on fuzzy C-means and improved self adaptive differential evolution algorithm," *Int. J. Inf. Technol.*, vol. 14, no. 4, pp. 2171–2182, Jun. 2022.

[79] M. Arora, S. Verma, Kavita, M. Wozniak, J. Shafi, and M. F. Ijaz, "An efficient ANFIS-EEBAT approach to estimate effort of scrum projects," *Sci. Rep.*, vol. 12, no. 1, p. 7974, May 2022.

[80] A. Kaushik and N. Singal, "A hybrid model of wavelet neural network and metaheuristic algorithm for software development effort estimation," *Int. J. Inf. Technol.*, vol. 14, no. 3, pp. 1689–1698, May 2022.

[81] W. Rhmann, B. Pandey, and G. A. Ansari, "Software effort estimation using ensemble of hybrid search-based algorithms based on metaheuristic algorithms," *Innov. Syst. Softw. Eng.*, vol. 18, no. 2, pp. 309–319, Jun. 2022.

[82] A. Ardiansyah, R. Ferdiana, and A. E. Permanasari, "MUCPSO: A modified chaotic particle swarm optimization with uniform initialization for optimizing software effort estimation," *Appl. Sci.*, vol. 12, no. 3, p. 1081, Jan. 2022.

[83] S. Sharma and S. Vijayvargiya, "An optimized neuro-fuzzy network for software project effort estimation," *IETE J. Res.*, pp. 1–12, Jan. 2022.

[84] A. Kaushik, N. Singal, and M. Prasad, "Incorporating whale optimization algorithm with deep belief network for software development effort estimation," *Int. J. Syst. Assurance Eng. Manage.*, vol. 13, no. 4, pp. 1637–1651, Aug. 2022.

[85] A. Jain and A. Bansa, "Effort estimation using neural network and metaheuristic optimizer," in *Proc. 10th Int. Conf. Rel., INFOCOM Technol. Optim., Trends Future Directions (ICRITO)*, Noida, India, Oct. 2022, pp. 1–5.

[86] S. K. Gouda and A. K. Mehta, "A self-adaptive differential evolution using a new adaption based operator for software cost estimation," *J. Inst. Eng., India, B*, vol. 104, no. 1, pp. 23–42, Feb. 2023.

[87] S. Hameed, Y. Elsheikh, and M. Azzeh, "An optimized case-based software project effort estimation using genetic algorithm," *Inf. Softw. Technol.*, vol. 153, Jan. 2023, Art. no. 107088.

[88] C. S. Yadav, R. Singh, S. Satpathy, S. B. Priya, B. T. Geetha, and V. Goyal, "Energy efficient and optimized genetic algorithm for software effort estimator using double hidden layer bi-directional associative memory," *Sustain. Energy Technol. Assessments*, vol. 56, Mar. 2023, Art. no. 102986.

[89] S. Kassaymeh, M. Alweshah, M. A. Al-Betar, A. I. Hammouri, and M. A. Al-Ma'aitah, "Software effort estimation modeling and fully connected artificial neural network optimization using soft computing techniques," *Cluster Comput.*, vol. 56, pp. 1–24, Feb. 2023.

[90] C. López-Martín, "Machine learning techniques for software testing effort prediction," *Softw. Quality J.*, vol. 30, no. 1, pp. 65–100, Mar. 2022.

[91] K. R. Jayakumar and A. Abran, "Estimation models for software functional test effort," *J. Softw. Eng. Appl.*, vol. 10, no. 4, pp. 338–353, 2017.

[92] F. Ferrucci, C. Gravino, and F. Sarro, "Exploiting prior-phase effort data to estimate the effort for the subsequent phases: A further assessment," in *Proc. 10th Int. Conf. Predictive Models Softw. Eng.*, Sep. 2014, pp. 42–51.

[93] T. Labidi and Z. Sakhrawi, "On the value of parameter tuning in stacking ensemble model for software regression test effort estimation," *J. Supercomput.*, vol. 79, no. 15, pp. 17123–17145, Oct. 2023.

[94] Ż. Radliński, "The impact of data quality on software testing effort prediction," *Electronics*, vol. 12, no. 7, p. 1656, Mar. 2023.

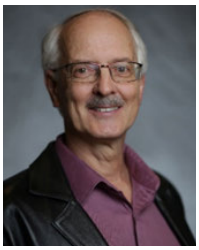
[95] A. Kaur and K. Kaur, "A COSMIC function points based test effort estimation model for mobile applications," *J. King Saud Univ. Comput. Inf. Sci.*, vol. 34, no. 3, pp. 946–963, Mar. 2022.

[96] L. A. Breiman, "The edge," Statist. Dept., Univ. California, Berkeley, CA, USA, Tech. Rep. 486, 1997.

[97] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Ann. Statist.*, vol. 29, no. 5, pp. 1189–1232, Oct. 2001.
[98] J. H. Friedman, "Stochastic gradient boosting," *Comput. Statist. Data Anal.*, vol. 38, no. 4, pp. 367–378, Feb. 2002.
[99] T. Hastie, R. Tibshirani, and J. H. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. New York, NY, USA: Springer, 2009, pp. 1–758.
[100] K. Kiatkarun and P. Phunchongharn, "Automatic hyper-parameter tuning for gradient boosting machine," in *Proc. 1st Int. Conf. Big Data Analytics Practices (IBDAP)*, Sep. 2020, pp. 1–6.
[101] J. H. Holland, *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI, USA: Univ. Michigan Press, 1976.
[102] M. Shepperd and S. MacDonell, "Evaluating prediction systems in software project estimation," *Inf. Softw. Technol.*, vol. 54, no. 8, pp. 820–827, Aug. 2012.
[103] R. Storn and K. Price, "Differential evolution—A simple and efficient heuristic for global optimization over continuous spaces," *J. Global Optim.*, vol. 11, pp. 341–359, Dec. 1997.
[104] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. IEEE Int. Conf. neural Netw.*, vol. 4. Nov./Dec. 1995, pp. 1942–1948.
[105] A. Wetzel, "Evaluation of effectiveness of genetic algorithms in combinatorial optimization," M.S. thesis, Univ. Pittsburgh, Pittsburgh, PA, USA, 1983.
[106] H. Mühlenbein and D. Schlierkamp-Voosen, "Predictive models for the breeder genetic algorithm I. Continuous parameter optimization," *Evol. Comput.*, vol. 1, no. 1, pp. 25–49, Mar. 1993.
[107] *Guidelines for Use of the ISBSG Data*, Int. Softw. Benchmarking Standards Group, Melbourne, VIC, Australia, 2022.
[108] The International Function Point Users Group. (2020). *IFPUG*. [Online]. Available: https://ifpug.org/
[109] *Information Technology, Systems and Software Engineering, FiSMA 1.1 Functional Size Measurement Method*, Standard ISO/IEC 29881:2010, 2010.
[110] *Software Engineering, NESMA Functional Size Measurement Method, Definitions and Counting Guidelines for the Application of Function Point Analysis*, Standard ISO/IEC 24570:2018, 2018.
[111] J. P. Fox, *Bayesian Item Response Modeling: Theory and Applications* (Statistics for Social and Behavioral Sciences). Berlin, Germany: Springer, 2010.
[112] I. Myrtveit and E. Stensrud, "Validity and reliability of evaluation procedures in comparative studies of effort prediction models," *Empirical Softw. Eng.*, vol. 17, nos. 1–2, pp. 23–33, Feb. 2012.
[113] W. S. Humphrey, *A Discipline for Software Engineering*, 1st ed. Reading, MA, USA: Addison-Wesley, 1995.
[114] S. Luke, *Essentials of Metaheuristics*, 2nd ed. Springer, 2015.
[115] E. Mezura-Montes, M. E. Miranda-Varela, and R. D. C. Gómez-Ramón, "Differential evolution in constrained numerical optimization: An empirical study," *Inf. Sci.*, vol. 180, no. 22, pp. 4223–4262, Nov. 2010.
[116] V. K. Bardsiri, D. N. A. Jawawi, S. Z. M. Hashim, and E. Khatibi, "A PSO-based model to increase the accuracy of software development effort estimation," *Softw. Quality J.*, vol. 21, no. 3, pp. 501–526, Sep. 2013.
[117] A. E. Eiben and J. E. Smith, "Representation, mutation, and recombination," in *Introduction to Evolutionary Computing*. Berlin, Germany: Springer, 2015.
[118] T. Dybå, V. B. Kampenes, and D. I. K. Sjøberg, "A systematic review of statistical power in software engineering experiments," *Inf. Softw. Technol.*, vol. 48, no. 8, pp. 745–755, Aug. 2006.
[119] D. S. Moore, G. P. McCabe, and B. A. Craig, *Introduction to the Practice of Statistics*, 6th ed. New York, NY, USA: W. H. Freeman and Company, 2009.
[120] R. Malhotra and M. Khanna, "Threats to validity in search-based predictive modelling for software engineering," *IET Softw.*, vol. 12, no. 4, pp. 293–305, Aug. 2018.
[121] A. Abran, *Software Project Estimation: The Fundamentals for Providing High Quality Information to Decision Makers*. Hoboken, NJ, USA: Wiley, 2015.
[122] S. D. Sheetz, D. Henderson, and L. Wallace, "Understanding developer and manager perceptions of function points and source lines of code," *J. Syst. Softw.*, vol. 82, no. 9, pp. 1540–1549, Sep. 2009.

ANGEL J. SÁNCHEZ-GARCÍA received the M.S. and Ph.D. degrees in artificial intelligence from the Artificial Intelligence Research Center, Universidad Veracruzana, Veracruz, Mexico, in 2013 and 2018, respectively. Since 2017, he has been a full-time Professor with Facultad de Estadística e Informática, Universidad Veracruzana. His research interests include machine learning in software engineering, evolutionary computation, software measurement, computer vision, and robotics. For more details visit https://uv.mx/personal/angesanchez.

CUAUHTÉMOC LÓPEZ-MARTÍN received the Ph.D. degree in computer science from the Center for Computing Research, National Polytechnic Institute of Mexico, in 2007. He is a Researcher with the Information Systems Department, Universidad de Guadalajara, Jalisco, Mexico. He has more than 20 years of industry and higher education experience in information system development and software engineering. His research relates to software prediction techniques, software processes, and statistics applied to software engineering. For more details visit https://dti.cucea.udg.mx/es/directorio/cuauhtemoc-lopez-martin.



ALAIN ABRAN (Life Senior Member, IEEE) received the master's degree in management sciences and electrical engineering from the University of Ottawa, Canada, and the Ph.D. degree in electrical and computer engineering from École Polytechnique, Canada. He is an Emeritus Professor with École de Technologie Supérieure (ETS), Canada. He was with Canadian banking industry, for 20 years. He has more than 20 years of teaching and research experience with Université du Québec á Montréal (UQAM) and ETS. His industry-oriented research has influenced a number of international standards in software engineering, such as ISO 15939, ISO 19759, ISO 19761, and ISO 14143-3. His research interests include software estimation, software quality measurement, software functional size measurement, software project, and software maintenance management.