Optimization of use case point through the use of metaheuristic algorithm in estimating software effort

Ardiansyah [a,1,*], Mulki Indana Zulfa [b,2], Ali Tarmuji [a,3], Farisna Hamid Jabbar [a,4]

[a] Department of Informatics, Universitas Ahmad Dahlan, Yogyakarta, Indonesia
[b] Department of Electrical Engineering, Jenderal Soedirman University, Yogyakarta, Indonesia
[1] ardiansyah@tif.uad.ac.id; [2] mulki_indanazulfa@unsoed.ac.id; [3] ali.tarmuji@tif.uad.ac.id; [4] farisna2000018170@webmail.uad.ac.id
* corresponding author

ARTICLE INFO

Article history
Received September 5, 2023
Revised December 10, 2023
Accepted February 19, 2023
Available online February 29, 2024

Keywords
Software effort estimation
Optimization techniques
Metaheuristics algorithms
Use case points
Firefly algorithm

ABSTRACT

Use Case Points estimation framework relies on the complexity weight parameters to estimate software development projects. However, due to the discontinue parameters, it lead to abrupt weight classification and results in inaccurate estimation. Several research studies have addressed these weaknesses by employing various approaches, including fuzzy logic, regression analysis, and optimization techniques. Nevertheless, the utilization of optimization techniques to determine use case weight parameter values has yet to be extensively explored, with the potential to enhance accuracy further. Motivated by this, the current research delves into various metaheuristic search-based algorithms, such as genetic algorithms, Firefly algorithms, Reptile search algorithms, Particle swarm optimization, and Grey Wolf optimizers. The experimental investigation was carried out using a Silhavy UCP estimation dataset, which contains 71 project data from three software houses and is publicly available. Furthermore, we compared the performance between models based on metaheuristic algorithms. The findings indicate that the performance of the Firefly algorithm outperforms the others based on five accuracy metrics: mean absolute error, mean balance relative error, mean inverted relative error, standardized accuracy, and effect size. This research could be useful for software project managers to leverage the practical implications of this study by utilizing the UCP estimation method, which is optimized using the Firefly algorithm.

1. Introduction

Estimating software development projects belongs to the planning phase in the software development life cycle (SDLC) [1]. Software Effort Estimation (SEE) assesses the effort and associated costs needed to create a new software system. Effort estimation is paramount for software companies, as they must guarantee that the software is delivered within predefined time and budget limitations. Nevertheless, most software projects exceed their designated time and budget limit, with delays and cost overruns being persistent challenges in software development for numerous years [2].

There are three groups of methods for estimating software effort: expert judgment, algorithmic, and machine learning. Expert-based methods entail consulting one or more experts, leveraging their domain expertise and comprehension of the organizational context to estimate the cost of software projects. Analytical hierarchy process (AHP), Delphi, and Planning Poker methods belong to the expert judgment group. The algorithmic group consists of methods used to estimate effort based on software requirements specifications. Methods that fall into this category are COCOMO [3], SLIM, Function Points [4], Use Case Points (UCP), COSMIC, and so on. Meanwhile, the machine learning-based group

is a set of methods that applies learning algorithms to estimate software project efforts such as regression, artificial neural network (ANN), case-based reasoning (CBR), and others. Among algorithmic approach, COCOMO and function points suffered from its parameter vulnerabilities: function points and line of codes (LoC) [5]. Moreover, they are not compliance with object oriented development as widely used today by most of software organizations [6]. One popular software sizing and estimation framework is Use Case Points (UCP) [7] that adopt object oriented development paradigm. UCP determines the software size by multiplying use case diagrams and the productivity factor (PF). In the second phase of the UCP process, as indicated by Table 1, the complexity weight of the use case (UC) is categorized into three levels: simple, average, and complex. The weight assignment to each complexity level is determined by the number of transactions of the Use Cases (UC). The original UCP framework consists of three levels of complexity weights, namely 5, 10, and 15, for Simple, Average, and Complex, respectively. The original level of complexity weighting has faced criticism from researchers due to its discontinuous nature in the complexity hierarchy, leading to occasional inaccuracies in measurements and abrupt classifications of use cases. For instance, the use case containing eight transactions possesses twice the weight of the use case involving seven transactions. Furthermore, they exclude the consideration of large use case transactions. As an illustration, a use case with 25 transactions holds equal weight as a use case involving nine transactions.

Numerous efforts have been made to tackle the sudden classification problem associated with the complexity weighting level in the use case. For example, the currently available fuzzy methods are employed and suggested by Xie *at al.,* Wang *et al.*, and Nassif *et al.* [8]–[10]. The fuzzy approach is consistently employed for discretizing the prevailing level of complexity weighting. It aims to alleviate the sudden classification problem by offering a continuous and incremental classification process. Most research demonstrates that using fuzzy on UCP enhances estimation performance compared with the original UCP.

Employing a continuous level of use case weighting enables us to enhance UCP accuracy through an optimization approach. Optimization stands out as a renowned approach for addressing continuous problems. The implementation of metaheuristic optimization in software effort estimation is called search-based software estimation (SBSE).

The research focused on search-based software effort estimation has been employed in numerous studies [11]. For example, Cased-Based Reasoning (CBR) effort estimation has been optimized by using Genetic Programming (GP) [12], [13], Particle Swarm Optimization (PSO) [14], evolutionary-based algorithm [15]–[17], and hybrid PSO-SA [18]. Firefly [19], [20], PSO [21]–[24], GA [25], and DE [26] are employed to enhance the optimization of COCOMO effort estimation. Although the previous works succeeded in improving the utilization of the use case complexity weight by applying PSO [27] and MUCPSO [28], these two investigations were still hampered by prematur convergence and local optimum traps. In addition, these studies only use the PSO algorithm and do not investigate various alternative metaheuristic algorithms.

Thus, this article extensively investigates various metaheuristic search-based models aimed at optimizing parameter values for the use case complexity weight. The methods under scrutiny encompass Genetic Algorithm (GA), Firefly Algorithm (FA), Reptile Search Algorithm (RSA), Particle Swarm Optimization (PSO), and Grey Wolf Optimizer (GWO). These metaheuristic algoritms are employed because they have more diverse characteristics as well as strengths and weaknesses to solve the particular optimization problem in search-based software effort estimation. For example, RSA and GWO are two algorithms based on encircling and hunting mechanisms. PSO and Firefly are algorithms based on large flocks of animals looking for food. Meanwhile, GA is an algorithm that adopts evolutionary theory. The experimental analysis was carried out using a publicly accessible UCP estimation dataset. The effectiveness of diverse metaheuristic search-based models has been assessed through various performance evaluation metrics. The findings and insights derived from these experiments can be instrumental in constructing ensemble models for UCP estimation. Moreover, the contribution of this presented work: our study involved a thorough exploration of five metaheuristic search-based models using unimodal and multimodal test functions, a novel approach that had not been previously undertaken in search-based software effort estimation field.

2. Related Works

The examination of effort estimation in UCP research reveals three main trends: the refinement of the UCP sizing approach, the scrutiny and simplification of UCP, and the fusion of UCP with data mining and machine learning (MLDM) techniques.

Numerous research works have suggested the overhaul of the UCP sizing method. Braz and Vergilio [29] adapted the use case complexity weight by applying fuzzy theory, and [28] effectively optimized this adjusted weight. In Robiolo and Orosco [30], two new variables, namely size-transactions and entity objects, were introduced and calculated based on the information in the use case description. The research presented by Mohagheghi *et al.* [31] has revised the assessment of actor complexity and addressed the consideration of non-functional requirements. This study made a significant contribution to enhancing the adaptability of UCP for incremental development approaches. Anda *et al.* [32],[33] examined and simplified the UCP better to comprehend the effects of technical and environmental complexity factors. The authors recommended fine-tuning the environmental factors according to the organization's type to enhance estimation accuracy. In contrast, Ochodek *et al.* [34], [35] omitted various segments of the UCP in order to simplify the UCP calculation process. The researcher contended that these sections are of negligible importance in effort estimation. In a recent development, Nhung *et al.* [36] enhanced the accuracy of the modified UCP by refinement of the environment complexity factors (ECF) and technical complexity factors (TCF) and employing a model of multiple regression for estimation.

Recent years have witnessed the exploration of MDLM techniques to enhance the UCP's performance. The introduction of a log-linear regression model [37] establishes relations between UCP, effort, and productivity factors. A hybrid model was introduced in a subsequent study referenced as in Ochodek *et al.* [34]. This model concurrently predicts the PF and effort estimation by leveraging past projects. Similarly, Nassif *et al.* [38] calculated the effort required by utilizing team productivity and UCP with Treebost.

The overarching objective of SEE research is to reduce the disparity between the real and predicted effort value. The rigidity weight level of the use case complexity has a notable influence on the accuracy of the estimation, as indicated by Nassif *et al.* [10]. Furthermore, it is worth noting that the initial complexity and assigned weight value may not accurately represent real-world scenarios, as suggested in Qi *et al.* [39]. Fortunately, this was previously validated by Karner [7], indicating that the suggested complexity weight is derived from the subjective assessment of individuals at Objective Systems. Karner [7] also emphasizes the need for additional data to refine parameters, models, and weights. The initial weight should not be considered the definitive and optimal weighting parameter. In simpler terms, it is crucial to have flexibility in granularity to establish the most appropriate weighting system and achieve the highest estimation performance. Three primary methodologies revolve around proposing improvements to the complexity weight of use cases: introducing additional weight levels, discretizing the existing scale, and calibrating in this way as outlined.

Two critical parameters for SEE were introduced in Silhavy and Silhavy [40]: the specifications of actors and the use case. At the same time, Nhung *et al.* [41] assessed TCF and ECF. To compute the number of transactions and automatically generate the class diagrams, Qi and Boehm [42] proposed USIM to determine the project size. Numerous studies have suggested incorporating additional complexity weight levels to account for factors that influence the level of complexity weight. Examples such as the nature of the application and the particular manner in which use cases are employed constitute two determinants affecting the degree of complexity weighting. Thus, it is advisable to re-evaluate the complexity weight by specific circumstances. As part of their proposal, Galorath and Evans [43] introduced weight values of 10 for simple, 15 for average, and 20 for complex. In contrast, Periyasamy and Ghode [44] proposed "most complex" as an additional degree of complexity weight to enhance the complexity weight of the use case. Periyasamy and Ghode [44] suggested including an extra "critical" level in their proposal. An additional "very high" level for use cases with more than 14 transactions was proposed [45], with corresponding weight assignments of 5, 10, 15, and 20. In contrast, on the other hand, three more complexity weights were introduced by Nassif [46]: 20, 25, and 30.

In order to reduce the existing level of complexity weights, a fuzzy approach is commonly used. This approach aims at reducing categorization abruption by gradually and continuously classifying them. Early attempts to discretize the established complexity weight levels were put forward by Xie *at al.,* Wang *et al.*, and Nassif *et al.* [8]–[10]. Wang *et al.* [9] expanded the complexity weight levels from three to five. They introduced the Extended Use Case Points (EUCP) by combining the theory of fuzzy logic and the Bayesian Belief Network (BBN). The outcomes indicated that the proposed EUCP outperformed UCP in the case of the two projects. Xie *at al.* [8] introduced the discretization degree of complexity weight that extended use-case complexity from 3 to 4. The outcomes of their study indicated that the suggested weight level demonstrated an increment of 5.5 person-hours, accompanied by a 15.45% margin of error, as determined from an analysis of four authentic project datasets. Nassif *et al.* [10] proposed ten levels of complexity weight based on the count of transactions associated with each use case. Their results showed a 22% improvement in particular projects concerning the proposed approach, assuming that ten transactions were covered by the most extensive use case and had a complexity factor 15.

The investigation into UCP adjustment was initiated by Nassif *et al.* [47] and continued by Qi *et al.* [39]. In the work of Nassif *et al.* [47], a six-tier use case complexity weighting system was introduced as an alternative to the original three-tier weighting scheme outlined by Karner [7]. They used a neural network to calibrate these six proposed complexity weight levels. In order to cope with any sudden change in complexity levels and weights, Fuzzy Logic has been applied following successful calibration of the weights. Unfortunately, there was no evidence of any experimental results or model validation in this study.

On the other hand, Qi *et al.* [39] delved into Bayesian analysis for calibrating use case weighting. As inputs, their study collected the weight of the use case and the empirical project data. The weighting of a priori-based use case is used to calculate the mean and variance. Simultaneously, empirical project data was utilized to calibrate the weights of the use case via a multiple linear regression process. In the last phase of this calibration procedure, the outcomes of the mean and variance were harnessed to compute the Bayesian weighted average, ultimately producing Bayesian weight estimations as the resulting output. This approach underwent assessment through data analysis from 105 projects, which was subsequently juxtaposed against a priori estimations, regression-based, and Karner's UCP method. As a result, the Bayesian approach demonstrated superior accuracy in effort estimation.

These methodologies have a broader option to enhance the complexity factor by expanding Karner's UCP method, which has shown positive results by using continuous and incremented classification values. However, despite the endeavors of researchers referenced in Nhung *et al.* [36], Hoc *et al.* [48], and Hariyanto and Wahono [49] to enhance the performance of UCP, they did not delve into the possibilities associated with continuous complexity weight levels when optimizing functions. Metaheuristics optimization, renowned for its ability to handle continuous function optimization, can effectively mitigate abrupt complexity-level changes. While prior research has indeed succeeded in enhancing the utilization of use case complexity weight through the application of PSO [27] and MUCPSO [28], it is essential to note that these two studies have focused solely on the utilization of the PSO algorithm. They have not explored the potential benefits of various alternative metaheuristic algorithms.

Therefore, this study comprehensively examines various metaheuristic search-based models to optimize the weights of use case complexity. The methods under investigation include genetic algorithms, firefly algorithms, reptile search algorithms, particle swarm optimization, and grey wolf optimizers.

2.1. UCP

The UCP method suggested by Karner [7] comprises seven steps, as summarized in Table 1. Firstly, calculate UAW (unadjusted actor weighting) and UUCW (unadjusted use case weighting) by classifying actors and use cases into three levels of complexity: Simple (1), Average (2), and Complex (3). The sum of UAW and UUCW yields unadjusted use case points (UUCP). Secondly, it calculates technical and environmental complexity factors. Finally, project size and estimated effort can be determined using these factors.

Table 1. Use Case Points estimation parameters and formula

Parameter	Formula
UAW	$UAW = \sum_{i=1}^{3} W_i \times Actor_i$, where W_i represents the weight factor, categorized as 1 for simple, 2 for average, and 3 for complex actors.
UUCW	$UUCW = \sum_{i=1}^{3} W_i \times UC_i$, where W_i represents a weight factor, classified as 5 for simple, 10 for average, and 15 for complex use cases, respectively.
UUCP	$UUCP = UAW + UUCW$
TCF	$TCF = 0.6 + \left(0.01 \times \sum_{i=1}^{13} W_i \times G_i\right)$
ECF	$ECF = 1.4 + \left(-0.03 \times \sum_{i=1}^{8} W_i \times G_i\right)$
Project Size	$UCP = UUCP \times TCF \times ECF$
Estimated Effort	$Effort = UCP \times PF$, PF represents the productivity factor, and assigning a value of 20 person-hours per UCP is possible.

In addition to the original weight levels, Table 2 presents the complexity weight levels proposed by [6] and [46].

Table 2. The original and modified use case complexity weight level

Number of Use Case Transactions	Original weight level	Modified weight level
1-2	5	5.00
3	5	6.45
4	10	7.50
5	10	8.55
6	10	10.00
7	10	11.40
8	15	12.50
9	15	13.60
>10	15	15.00

2.2. Metaheuristic Algorithms

2.2.1. Grey Wolf Optimizer

The Grey Wolf Optimizer algorithm, introduced by Mirjalili *et al.* [50], is inspired by grey wolves' social structure and hunting behavior. Grey wolves in the wild operate within a hierarchical structure consisting of alpha, beta, delta, and omega wolves, each with distinct roles in the pack. These roles are translated into the GWO algorithm's search mechanisms to explore and exploit the solution space effectively. The GWO algorithm begins with an initial population of grey wolves, where each wolf represents a potential solution to the optimization problem. The algorithm iteratively updates the positions of the wolves to converge towards an optimal solution as shown in Algorithm 1 (Fig. 1).

```
Generate initial grey wolf population x_i(i = 1, 2, ..., n)
Initialize a, A, and C
Calculate the fitness value of each wolf
X_α = the best wolf
X_β = the runner up best wolf
X_δ = the third best wolf
While (t < max number of iterations)
  For each wolf
    Update the position of the current wolf
  End for
  Update a, A, and C
  Compute the fitness value of all wolf
  Update X_α, X_β, dan X_δ
  t = t + 1
end while
return X_α
```

Fig. 1. Grey wolf optimizer pseudocode

2.2.2. Particle Swarm Optimizer

PSO drew inspiration from the behavior of birds flocking and fish schooling in search of locations with an ample food supply [25]. PSO begins by randomly generating a population based on swarm size parameters. This population comprises N particles, with each particle i representing a potential solution to the problem. Each individual particle is represented by the vector x_i in the decision space and possesses both position (x) and velocity (v) attributes, as indicated in Eq. (1) and Eq. (2).

$$v_i = \omega v_i + C_1 R_1 \times (Pbest_i - x_i) + C_2 R_2 \times (Gbest_i - x_i) \quad (1)$$

$$x_{i+1} = x_i + v_i \quad (2)$$

where v_i represents the current or initialized velocity of the particle, which is endowed with a random number value falling within the interval [0, 1] at the onset of the population generation process. Constants C_1 and C_2, cognitive and social learning factors, remain unchanging throughout the computation. R_1 and R_2 are stochastic variables defined within the [0, 1] range. $Pbest_i$ represents the most optimal position attained by particle i, while $Gbest_i$ represents the overall best position achieved by the entire ensemble particles. Lastly, x_i is used to denote the present position of the particle. Additionally, ω is an inertia weight, a constant value set to 0.9. Algorithm 2 outlined the PSO in pseudo-code format to provide a schematic representation of the algorithm (Fig. 2).

```
1   Generate initial population x_i(i = 1, 2, ..., n)
2   Gbest = maximum fitness value of particle in population
3   Pbests = initial population
4   While t < max iterations
5     For each particle in the population
6       Update velocity
7       Update position
8       Update particle
9       Update pbest
10    End for
11    Updated population
12    Update gbest
13  End while
    Return gbest
```

Fig. 2. Particle swarm optmization pseudocode

2.2.3. Firefly

The Firefly Algorithm (FA) [51], [52] draws inspiration from the idealized behavior of fireflies' flashing characteristics. These flashing characteristics can be simplified into the following three rules for ease of understanding:

- In the FA, all fireflies are considered unisex, meaning that one firefly is attracted to other fireflies irrespective of their gender.
- In the FA, the attractiveness of a firefly is directly proportional to its brightness. Therefore, comparing two flashing fireflies, the less bright one will move toward the brighter one. This attractiveness is influenced by brightness, and both attractiveness and brightness decrease as the distance between fireflies increases. If no firefly is found to be brighter than a particular firefly, it will move randomly.
- The brightness or light intensity of a firefly is influenced or determined by the characteristics of the objective function landscape that is being optimized.

The light intensity $I(r)$ changes monotonically and exponentially with distance r. This relationship can be expressed as:

$$I = I_0 e^{-\gamma r} \quad (3)$$

where the original light intensity is I_0 and the light absorption coefficient is γ. The attractiveness β of a firefly can be defined based on the light intensity seen by neighboring fireflies. This relationship is expressed as follows:

$$\beta = \beta_0 e^{-\gamma r^2} \tag{4}$$

where the attractiveness at $r = 0$ is β_0 plays a crucial role in this definition. It is important to note that the exponent γr can be replaced with other functions, such as γr^m when $m > 0$. In summary, Algorithm 3 outlined the FA in pseudo-code format to provide a schematic representation of the algorithm (Fig. 3).

```
Initialize a population of fireflies x_i(i = 1, 2, ..., n)
γ = Coefficient of light absorption
while (t < MaxGeneration)
  for i = 1:n all n fireflies
    for j = 1:i all n fireflies
      light intensity I_i at x_i is calculated using f(x_i)
      if (I_i > I_j)
        Move firefly i towards j in all d dimensions
      end if
      attractiveness varies with distance r via exp[−γr]
      evaluate new solutions and update light intensity
    end for j
  end for i
  Rank the fireflies and find the current best
end while
return the best firefly
```

Fig. 3. Firefly algorithm pseudocode

2.2.4. Genetic Algorithm

The Genetic algorithms first proposed by Holland [53] encompass the following fundamental processes: 1) encoding the objective or optimization functions, 2) establishing a fitness function or selection criteria, 3) generating a population of individuals, 4) cycling through evolution iterations, which involve evaluating the fitness of all individuals in the population, creating a new population through actions such as crossover, mutation, and fitness-proportionate reproduction, replacing the old population, and iterating once more using the new population; 5) decode the outcomes acquired from the solution to the problem. These steps can be represented schematically in the pseudo-code of genetic algorithms, as illustrated in Algorithm 4 (Fig. 4).

```
Define an objective function f(x), x = (x_1, ..., x_n)^T
Translate the solution into chromosome representations.
Initialize GA parameters: number of chromosomes, number of generations, mutation rate (mr), crossover
rate (cr), etc
Generate the initial population
While (t < number of generations)
      Generate fresh solution through a combination of crossover and mutation
      If cr > rand, Crossover; end if
      If mr > rand, Mutate; end if
      If fitness value increases, accept the new solution
      Select the current best solution for the next generation
End while
Return the best solution
```

Fig. 4. Genetic algorithm pseudocode

2.2.5. Reptile Search Optimizer

The optimization process starts with the creation of a randomly selected set of candidate solutions that will be used to form an early population at RSA. Throughout the repetition trajectory, the search mechanisms of RSA systematically explore potential positions in search of near-optimal solutions. In this pursuit, each solution adjusts its positions based on the processes outlined in the RSA algorithm, potentially replacing its positions with those from the best-obtained solution found thus far.

The search procedures are classified as two primary methods of exploration and exploitation to ensure balance in both exploration and exploitation. Exploration is based on a strategy of high or belly walking and exploitation based on hunting coordination or cooperation. The potential candidates use these strategies to widen the search area when $t \leq$ and seek to converge with near-optimal solutions if $t > \frac{T}{2}$. In the initial exploration phase, the high walking movement strategy is employed when $t \leq \frac{T}{4}$. Subsequently, as t progresses and reaches between $t \leq 2\frac{T}{4}$ and $t > \frac{T}{4}$, the belly walking movement strategy is adopted. During the exploitation phase, the hunting coordination strategy is put into action when it falls within the range of $t \leq 3\frac{T}{4}$ and $t > 2\frac{T}{4}$. In contrast, the hunting cooperation strategy is deployed when $t < T$ and $t > 3\frac{T}{4}$. The RSA shall cease to operate as soon as it complies with the applicable termination criteria. In Algorithm 5 (Fig. 5), a pseudocode is provided for the proposed RSA algorithm.

```
Initialize RSA parameters α, β, etc
Initialize reptile population x_i(i = 1, 2, ..., N)
While (t < T)
  Calculate fitness value for all reptile
  Find the best fitness value of all reptiles so far
  Update Evolutionary Sense (ES)
  For (i = 1 to N) do
    For (j = 1 to N) do
      Update the η, R, P, and values, respectively
      If (t ≤ T/4) then
        Update reptile using a high walking procedure
      Else if (t ≤ 2T/4 and t > T/4) then
        Update reptile using belly walking procedure
      Else if (t ≤ 3T/4 and t > 2T/4) then
        Update reptile using hunting coordination procedure
      Else
        Update reptile using hunting cooperation procedure
      End if
    End for j
  End for i
End while
Return best reptile (best(X))
```

Fig. 5. Reptile Search Optimize pseudocode

3. Method

The objective of effort estimation is to reduce the disparity in accuracy between the actual effort and the estimated effort, as denoted in Eq. 5. Hence, Fig. 6 illustrates the proposed method of this study. As depicted in the figure, the complexity weighting comprises two primary elements: actors and use cases. Optimization in the use case component is performed individually using metaheuristic algorithms, including GWO, PSO, GA, RSA, and FA. Each algorithm will search the use case complexity weight that provide optimal result according to the allowed weight range. The results of these optimizations for actors and use cases are combined to generate the Unadjusted Use Case Points (UUCP).

Additionally, in conjunction with complexity factors (TCF and ECF), the actors and use cases contribute to the calculation of software size, specifically in UCP metric units. Subsequently, the obtained software size is multiplied by the PF (Productivity Factor) parameter to derive the estimated effort value, expressed in person-hour units. A detailed description of methods including dataset and evaluation techniques used is discussed specifically in section 3.1, 3.2, and 3.3.

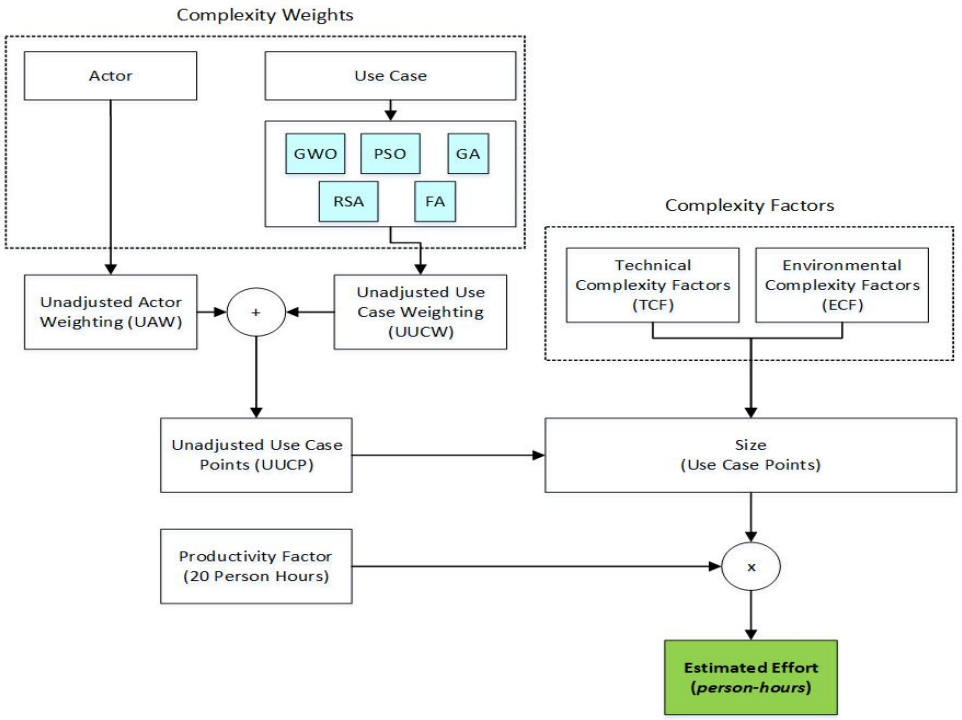

Fig. 6. The proposed model

3.1. Experimental Design

Fig. 7 shows the complete experimental design flow. The experimental design is provided to outline the details of the experiment at each stage. The dataset is divided into two, namely training data and test data. Test data is only taken for one instance. This means that each project will definitely become test data. This is in accordance with the validation method used, namely Leave one out cross validation.

Each test data consists of seven attributes or effort drivers as shown in Table 3. The UUWC value is generated by the summation function of multiplying the use case weights with the Simple, Average, and Complex attributes. The weight of the three use case will be determined using the five metaheuristic algorithms. The weight that gives optimum results is then used to calculate UUCP and estimated effort. The estimated effort is then compared with the actual results in order to calculate the accuracy performance.

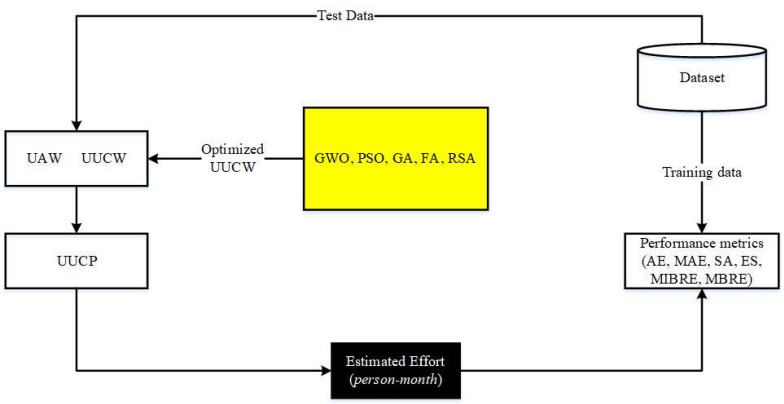

Fig. 7. Experimental design flow

3.2. Dataset

This research utilized historical project data derived from three actual software companies. Seventy-one projects collected by Silhavy [54] constitute the dataset for this project. The dataset encompasses various problem domains, including insurance, government, banking, etc. The dataset comprises a total of thirty-eight (38) effort drivers. For this study, we utilized seven (7) effort drivers and excluded the remainder. The comprehensive set of effort drivers encompasses simple, average, and complex use case (UC), technical and environmental complexity factors, UAW, and actual effort. These effort drivers were selected based on their significant influence within the UCP estimation methodology, as outlined in Table 1. Most projects were executed utilizing programming languages such as Java and C#. Summary statistics for the project dataset are provided in Table 3.

Table 3. Descriptive statistics for a dataset of 71 projects

Effort Driver	Avg	StDev	Skewness	Kurtosis	Max	Min
Simple	2.7	2.9	3.29	17.658	20	0.00
Average	15.84	5.37	0.296	0.140	30	3.00
Complex	14.29	4.45	0.191	-0.290	27	5.00
UAW	10.49	5.01	0.803	-1.264	19	6.00
TCF	0.92	0.114	-0.269	-1.019	1.12	0.71
ECF	0.86	0.117	-0.556	0.861	1.09	0.51
Actual Effort	6558.72	664.24	0.574	-0.922	7970	5775

Referring to Table 3 reveals distinct distributions among the variables under consideration. UC-Average, UC-Complex, and TCF exhibit distribution patterns that closely resemble a normal distribution, as evidenced by their skewness values approaching zero. Conversely, UC-Simple deviates from a normal distribution, with its skewness significantly deviating from zero. Notably, the UC-Complex variable demonstrates a relatively wider distribution, indicated by a kurtosis value of -0.290. At the same time, UC-Simple displays a leptokurtosis curve, indicated by its substantially high kurtosis value of 17.658. It is important to note that a kurtosis value below three suggests a lower susceptibility to outliers. Only UC-Simple surpasses this threshold among the mentioned variables, indicating a higher susceptibility to outliers. This infers that Average, Complex, UAW, TCF, ECF, and Actual Effort variables are comparatively less prone to outlier effects. Notably, the majority of the project use cases are of average complexity. This highlights that the Average variable has a mean and maximum value that is broader in range compared to the Simple and Complex effort drivers.

3.3. Model Validation and Evaluation

In this study, model validation was carried out utilizing the Leave-One-Out Cross-Validation (LOOCV) technique. LOOCV involves partitioning each dataset into n-1 folds for training data and one-fold for testing data. LOOCV was chosen due to its lower conclusion instability resulting from random selection in training and testing data, in contrast to the potential instability experienced in k-fold, 3-way, and 10-way techniques [55]. The comprehensive list of model performance evaluation formulas is presented in Table 4.

Table 4. Performance measurement list

Measure	Formula
Absolute error (AE)	$\mathrm{AE} = \lvert y_i - \hat{y}_i \rvert$
Mean absolute error (MAE)	$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n} \mathrm{AE}_i$
Mean balance relative error	$\mathrm{MBRE} = \frac{1}{n}\sum_{i=1}^{n} \frac{\mathrm{AE}_i}{\min(y_i,\hat{y}_i)}$
Mean inverted balance relative error	$\mathrm{MIBRE} = \frac{1}{n}\sum_{i=1}^{n} \frac{\mathrm{AE}_i}{\max(y_i,\hat{y}_i)}$
Standardized accuracy [56], [57]	$\mathrm{SA}_{P_j} = 1 - \left(\frac{\mathrm{MAE}_{P_j}}{\overline{MAE}_{P_0}}\right) \times 100$
Effect size [56], [57]	$\Delta = \frac{\mathrm{MAE}_{P_j} - \overline{\mathrm{MAE}}_{P_0}}{S_{P_0}}$

The performance of the estimation model was assessed using a variety of measurements. In this study, six measurement metrics were employed: absolute error (AE), mean absolute error (MAE), mean balance relative error (MBRE), mean inverted balance relative error (MIBRE), standardized accuracy, and effect size. These performance metrics were chosen due to their non-biased nature, in contrast to metrics such as mean squared error (MSE), mean absolute percentage error (MAPE), and prediction accuracy (PRED) that have been known to exhibit bias in their results [56], [57].

Based on Table 4, y_i and $\hat{y}_i$ represent the actual and estimated effort for the i-th project, respectively. MAE_{P_j} denotes the Mean Absolute Error (MAE) value generated by the j-th estimation model, such as FA+UCP, RSA+UCP, etc. Meanwhile, $\overline{MAE_{P_0}}$ is obtained from the random guessing technique, and S_{P_0} represents the standard deviation produced by the P_0 model.

3.4. Problem Formulation

This section presents the mathematical formulation of the optimization problem in UCP estimation. The objective function in UCP estimation is the minimum absolute error (MAE) between the estimated effort and the actual effort as follows:

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|\hat{y}_i - y_i| \quad (5)$$

Subject to:

$$\hat{y}_i = 20 \times UCP \quad (6)$$

$$UCP = UUCP \times TCF \times ECF \quad (7)$$

$$UUCP = UAW + UUCW \quad (8)$$

$$UUCW = \sum_{i=1}^{3} w_i \times UC_i \quad (9)$$

$$5 \leq UC_1 \leq 7.49 \quad (10)$$

$$7.5 \leq UC_2 \leq 12.49 \quad (11)$$

$$12.5 \leq UC_3 \leq 15 \quad (12)$$

where the value of y_i, TCF, ECF, and UAW taken from the data set, and w_i with values of 5, 10, and 15 respectively.

4. Results and Discussion

The experimental results are described in two parts. First are the evaluation results of 12 benchmark functions, and second are the experimental results of metaheuristic optimization on UCP. Furthermore, the parameter settings should be determined based on the best-performing or commonly used configurations in Table 5 to ensure a fair comparison between algorithms.

Table 5. Parameter settings of five metaheuristic algorithms

Algorithm	Parameters
Grey Wolf Optimizer (GWO) [58]	PopSize = 100, T_{max}= 20
Firefly Algorithm (FA) [59]	PopSize = 20, T_{max}= 20, α = 0.5, $\beta_{min} = 0.2$, β = 1, γ = 1
Particle Swarm Optimization (PSO)	PopSize = 70, $T_{max} = 20$, $\omega_{max} = 0.9$, $\omega_{min} = 0.4$, $C_1 = 2$, $C_2 = 2$
Genetic Algorithms (GA)	PopSize = 20, T_{max}= 20, Cr=0.25, Mr=0.1
Reptile Search Optimizer (RSA) [60]	PopSize = 30, T_{max}= 20, α = 0.1, β = 0.1

4.1. Evaluation of Classic Benchmark Functions

In order to assess the exploitation and exploration potential of optimization algorithms, 12 benchmark functions, consisting of six unimodal and six multimodal functions, are used in this study. An overview of the benchmark functions, including their functions names, types, dimensions, ranges, and f_{min} value is given in Table 6. Additionally, their two-dimensional representations are depicted in Fig. 8.

Table 6. The benchmark functions: F1-F6 for unimodal, and F7-F12 for multimodal

Functions	Type	Dim	Range	f_{min}
F1 Sphere	Unimodal	30	[-100, 100]	0
F2 Schwefel 2.22	Unimodal	30	[-10, 10]	0
F3 Schwefel 1.2	Unimodal	30	[-100, 100]	0
F4 Schwefel 2.21	Unimodal	30	[-100, 100]	0
F5 Rosenbrock	Unimodal	30	[-30, 30]	0
F6 Step	Unimodal	30	[-100, 100]	0
F7 Quartic Noise	Multimodal	30	[-1.25, 1.28]	0
F8 Schwefel 2.26	Multimodal	30	[-500, 500]	$-418.9829 \times \text{Dim}$
F9 Rastrigin	Multimodal	30	[-5.12, 5.12]	0
F10 Ackley	Multimodal	30	[-32, 32]	0
F11 Griewank	Multimodal	30	[-600, 600]	0
F12 Penalized	Multimodal	30	[-50, 50]	0

There are six benchmark functions, labeled with ID and function names F1-F6, which are unimodal and are employed to assess the exploitation ability. Subsequently, six benchmark functions, denoted by ID and function names F7-F12, are considered multimodal, featuring numerous local optima that increase as the dimensionality grows. These functions are utilized to evaluate the exploration capability.

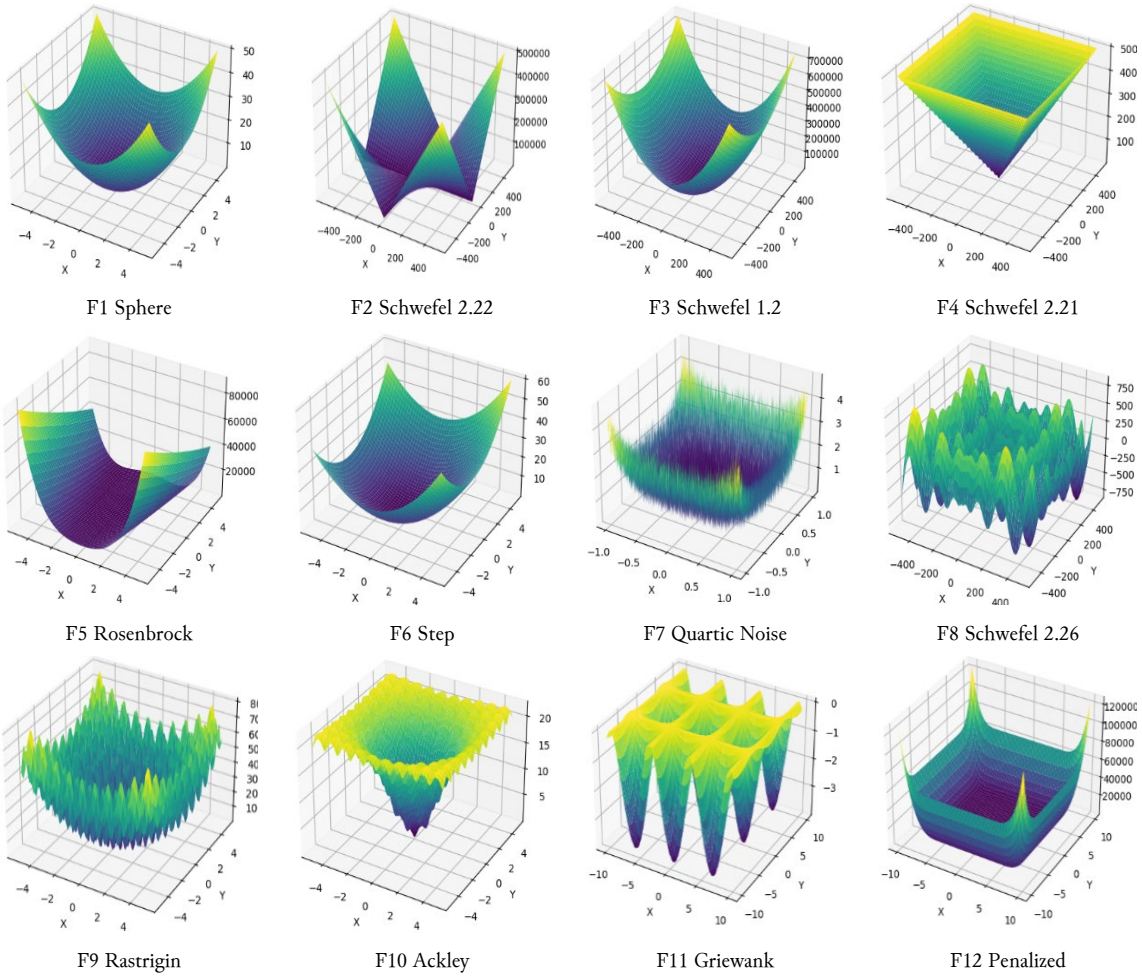

Fig. 8. Twelve classic benchmark functions F1 to F12

Table 7 provides an overview of the assessment results pertaining to four metrics: the best optimal solution, the least favorable solution, the average solution, and the standard deviation (STD).

Table 7. Comparison of FA, GA, GWO, PSO, and RSA for 12 benchmark functions

Functions	Metric	FA	GA	GWO	PSO	RSA
F1 Sphere	Best	5.60E+04	5.41E+04	5.83E-24	2.80E+05	0.00
	Worst	8.52E+04	9.42E+04	6.28E-21	3.00E+05	8.42E-26
	Mean	6.77E+04	7.67E+04	1.31E-21	2.87E+05	**4.63E-27**
	STD	7576.46	10078.89	1.84E-21	4572.47	**1.71E-26**
F2 Schwefel 2.22	Best	1.84E+02	3.13E+02	1.32E-21	5.63E+02	3.63E-57
	Worst	2.27E+02	2.13E+02	6.83E-14	5.83E+02	1.00E-08
	Mean	2.10E+02	2.53E+02	**5.11E-15**	5.76E+02	1.67E-09
	STD	10.99	26.66	**1.36E-14**	6.59	3.73E-09
F3 Schwefel 1.2	Best	7.40E+05	5.63E+05	4.13E-24	4.15E+06	0.00E+00
	Worst	1.23E+06	1.39E+06	5.80E-21	4.52E+06	7.38E-22
	Mean	1.04E+06	1.09E+06	1.13E-21	4.34E+06	**2.58E-23**
	STD	120362.33	224568.22	1.51E-21	103173.77	**1.32E-22**
F4 Schwefel 2.21	Best	8.21E+01	8.36E+01	1.28E-20	1.00E+02	0.00E+00
	Worst	8.88E+01	9.79E+01	6.48E-15	1.00E+02	2.45E-21
	Mean	8.61E+01	9.12E+01	4.65E-16	1.00E+02	**1.89E-22**
	STD	1.99	3.97	1.32E-15	0	**5.83E-22**
F5 Rosenbrock	Best	1.45E+08	2.07E+08	2.87E+01	2.12E+09	2.88E+01
	Worst	2.63E+08	4.10E+08	2.90E+01	2.36E+09	2.90E+01
	Mean	2.17E+08	2.93E+08	2.89E+01	2.23E+09	**2.90E+01**
	STD	26225079.4	48752581	0.078	59548404	**0.04**
F6 Step	Best	5.92E+04	5.43E+04	7.13E-01	2.82E+05	7.24E+00
	Worst	8.64E+04	9.09E+04	7.50E+00	3.00E+05	7.50E+00
	Mean	7.19E+04	7.34E+04	**5.38E+00**	2.89E+05	7.37E+00
	STD	7046.94657	9133.95	**2.33**	4186.15	0.12
F7 Quartic Noise	Best	3.08E+09	2.31E+09	1.07E+01	4.12E+10	9.32E+00
	Worst	6.02E+09	7.94E+09	1.36E+01	4.65E+10	1.12E+01
	Mean	4.69E+09	5.36E+09	1.25E+01	4.38E+10	**1.03E+01**
	STD	807671301	147885086	0.75	1.21E+09	**0.49**
F8 Schwefel 2.26	Best	2.52E-02	2.01E+00	-8.88E-05	-2.02E+03	-3.74E-05
	Worst	4.15E+00	1.52E+03	1.35E-16	1.57E+03	0.00E+00
	Mean	1.44E+00	1.39E+02	-1.20E-05	3.27E+01	**-2.54E-06**
	STD	1.27	307.87	2.46E-05	828.31	**7.23E-06**
F9 Rastrigin	Best	3.50E+02	3.72E+02	0.00E+00	8.14E+02	0.00E+00
	Worst	4.24E+02	5.35E+02	0.00E+00	8.68E+02	0.00E+00
	Mean	3.86E+02	4.62E+02	**0.00E+00**	8.40E+02	**0.00E+00**
	STD	19.46	44.01	**0**	11.26	**0**
F10 Ackley	Best	5.80E+02	5.88E+02	2.91E+01	6.10E+02	2.91E+01
	Worst	6.26E+02	6.52E+02	2.91E+01	6.29E+02	2.91E+01
	Mean	6.06E+02	6.29E+02	**2.91E+01**	6.16E+02	**2.91E+01**
	STD	11.65	14.17	**0**	6.21	**0**
F11 Griewank	Best	6.96E+02	6.66E+03	1.64E+00	3.67E+04	3.73E+01
	Worst	1.08E+04	1.30E+04	5.80E+03	4.16E+04	1.27E+04
	Mean	7.41E+03	9.57E+03	**2.80E+02**	3.91E+04	4.07E+03
	STD	2709.47	1488.27	**1031.02**	977.02	3213.44
F12 Penalized	Best	4.85E+04	2.57E+06	2.71E+02	-2.18E+11	3.60E+00
	Worst	3.99E+07	1.69E+10	2.71E+02	6.40E+10	2.71E+02
	Mean	1.23E+07	2.28E+09	2.27E+02	-2.86E+10	**1.94E+02**
	STD	11486247.2	433100633	76.47	6.63E+10	**98.40**

Based on Table 7, GWO performs better than any of the metaheuristics algorithms in applying to a subset of functions with ID values F2 and F6, given that it is based on two quantitative metrics, namely mean solution and STD, from 6 unimodal functions designed as F1 to F6. Meanwhile, RSA outperforms all other algorithms in the F1, F3, F4, and F5 test functions.

Subsequently, the analysis of six multimodal functions, identified as F7 through F12, reveals that the RSA method consistently surpasses its competing counterparts. Specifically, it attains significantly reduced mean solutions for functions F7, F8, F9, F10, and F12. RSA exhibits suboptimal performance solely when confronted with the F11 function, wherein GWO demonstrates superior capabilities for this specific function.

Similar benchmarking has been carried out by Yang [51], which shows that FA is superior to PSO and GA in the Schwefel, Rosenbrock, Ackley, Rastrigin, and Griewank test functions. These results are the same as the results obtained in this study. However, when faced with relatively new algorithms, namely RSA and GWO, it turns out that FA's performance is inferior to both. This is due to RSA's exploration ability, which uses two techniques: high walk and belly walk. Likewise, two other techniques are used in the exploitation phase: hunting coordination and cooperation. These four techniques produce great diversity. In addition, the stochastic coefficients of GWO produce dense solutions to exploit the optimal solution area.

4.2. Evaluation of Empirical Results

This section presents the empirical findings derived from our experimental configuration, encompassing model validation and assessment. A higher SA value signifies an estimation model's robustness and statistical significance. A more excellent effect size value indicates a reduced likelihood that the predictive model was derived by random chance. To address these considerations, we pose two research questions (RQ):

RQ1: To what extent is P_i superior to P_0?

RQ2: To what extent do FA+UCP, GA+UCP, GWO+UCP, PSO+UCP, and RSA+UCP outperform the Karner+UCP model?

4.2.1. RQ1: The performance of P_i versus P_0

The five models underwent validation through the assessment of (SA) and ES (Δ), with the baseline model being random guessing (P_0). As provided in Table 8, all models achieved SA values superior to random guessing, with FA+UCP achieving the highest SA value. This demonstrates that these models were engaged in prediction rather than random guessing, as they consistently outperformed random guessing. As a result, within the framework of this investigation, these models produced meaningful and reliable predictions.

On the contrary, all models displayed notably superior ES measurements compared to P_0. FA+UCP, GA+UCP, GWO+UCP, PSO+UCP, and RSA+UCP demonstrated substantial effect size improvements. Thus, we can confidently assert that the emergence of these five models was not a result of random chance. The significance test rendered inconclusive results for all six null hypotheses, as p-values were below the 0.05 threshold.

Table 8. The outcomes of SA, Δ, and Sig. are assessed relative to a baseline model of P_0

Algorithm	SA	ES	Sig.
FA+UCP	99.7337849077717	1.7306198385471399	0.00 ($p < 0.05$)
GA+UCP	99.70517937608656	1.729459431930611	0.00 ($p < 0.05$)
GWO+UCP	99.71166481163766	1.7315092038540898	0.00 ($p < 0.05$)
PSO+UCP	99.71243041125878	1.7279026606546029	0.00 ($p < 0.05$)
RSA+UCP	99.72075459687476	1.7289635371943752	**4.2** $p < 0.05$)

4.2.2. RQ2: The performance of FA+UCP, GA+UCP, GWO+UCP, PSO+UCP, and RSA+UCP versus Karner+UCP model

The validation process of the five models was conducted using the Karner+UCP model as the reference point. The selection of the Karner model is based on its significance as a fundamental reference in the realm of effort estimation studies utilizing the UCP approach. Numerous prior studies, such as

those conducted by Azzeh and Nassif [61], Silhavy *et al.* [54], [62], and Nassif *et al.* [37], have employed the Karner model for comparative purposes in their investigations.

Based on Table 9, concerning the effect size, it is worth highlighting that all the models exhibited substantially more significant effect size enhancements than the Karner+UCP model. These enhancements exceeded the thresholds indicative of medium to significant effects, which can be practically meaningful. Consequently, we can confidently assert that these five models did not arise by chance since the significance test yielded rejection for all four null hypotheses.

Table 9. SA, Δ, and Sig. results are evaluated with the baseline model being Karner+UCP

Algorithm	Standardized Accuracy	Δ	Sig.
FA+UCP	44.17589661527976	0.6380015048902027	0.00 ($p < 0.05$)
GA+UCP	38.891466757195836	0.5616821891487283	0.00 ($p < 0.05$)
GWO+UCP	38.736842824674646	0.5594490640918746	0.00 ($p < 0.05$)
PSO+UCP	39.97863138376142	0.5773833456833154	0.00 ($p < 0.05$)
RSA+UCP	41.60756357990875	0.6009088714141645	0.00 ($p < 0.05$)

Next, we assess the performance of all models to determine the optimum solution obtained for the estimated UCP effort. Table 10 provides an overview of the examination results, assessed using six key metrics: the best optimal solution, the least favorable solution, the average solution, the standard deviation (STDev), MBRE, and MIBRE. The following sections further elaborate on these metrics to comprehensively understand the results. In the context of UCP estimation, FA+UCP delivers the best solution among all the algorithms. However, when considering the worst and mean solution metrics, GA+UCP performs the worst and highest mean solution. This discovery is consistent with previous research [28], which also demonstrated that the GA+UCP algorithm performed inadequately in UCP effort estimation.

Table 10. Comparison of FA+UCP, GA+UCP, GWO+UCP, PSO+UCP, and RSA+UCP for UCP estimation method

Method	Best	Worst	Mean	Median	StDev	MBRE	MIBRE
FA+UCP	**1008.4625**	**1008.5421**	**1008.4916**	**753.59**	**0.019**	**0.2689**	**0.1541**
GA+UCP	1084.5897	1128.3754	1108.6966	826.87	8.3154	0.3123	0.1701
GWO+UCP	1082.8284	1123.9388	1100.0671	821.24	10.8890	0.3057	0.1680
PSO+UCP	1069.7307	1107.0869	1090.2106	771.72	8.7341	0.2821	0.1663
RSA+UCP	1054.8891	1055.4724	1054.9256	**753.59**	0.1368	0.2764	0.1613

Table 10 shows that the FA+UCP has performed better than the existing algorithms regarding different key metrics, but these results need validation. In this section, a statistical analysis was carried out to examine the characteristics of various algorithms. The Wilcoxon rank-sum test was utilized to assess the comparative effectiveness of the algorithms employed in this investigation. The selection of the Wilcoxon rank-sum test was based on the fact that the experimental data used for this analysis does not need to adhere to any specific distribution and has less effect caused by outliers [63]. The Wilcoxon test results for all algorithms are shown in Table 11. The table shows that the *p-values* for most algorithms except GWO+UCP versus GA+UCP are less than 0.05. Hence, we can conclude that the FA+UCP algorithm has significantly improved over other existing algorithms.

Table 11. The p-value results from the Wilcoxon-rank sum statistical test for each metaheuristic optimization algorithm, along with Friedman mean rank (FMR)

	FA+UCP	GA+UCP	GWO+UCP	PSO+UCP	RSA+UCP
FA+UCP		3.007E-13	2.4298E-13	0.000060	0.000196
GA+UCP	3.007E-13		**0.237863**	0.006960	0.002346
GWO+UCP	2.4298E-13	**0.237863**		0.010603	0.005171
PSO+UCP	0.000060	0.006960	0.010603		0.005062
RSA+UCP	0.000196	0.002346	0.005171	0.005062	
FMR	**1.73**	**3.94**	**4.10**	**2.76**	**2.66**
Rank	**1**	**4**	**5**	**3**	**2**

Fig. 9 presents a plot showing the actual and estimated effort values between FA+UCP and Karner+UCP models. The x-axis represents the project instance, while the y-axis represents the value of effort. The continuous black line represents the value of actual effort, while the dotted blue and orange represent the estimated effort value produced by FA+UCP and Karner+UCP models. In all cases, the models aim to produce estimated values that closely align with the actual effort values. When the dotted blue or orange approaches and aligns with the solid black line, it indicates an accurate estimation by the model. As evident from the results, it can be observed that FA+UCP exhibits the highest proximity to the actual regression line, implying that the model was estimated with the highest degree of accuracy.

FA+UCP closely approximates the regression line, indicating that the model has been estimated with the highest degree of precision.

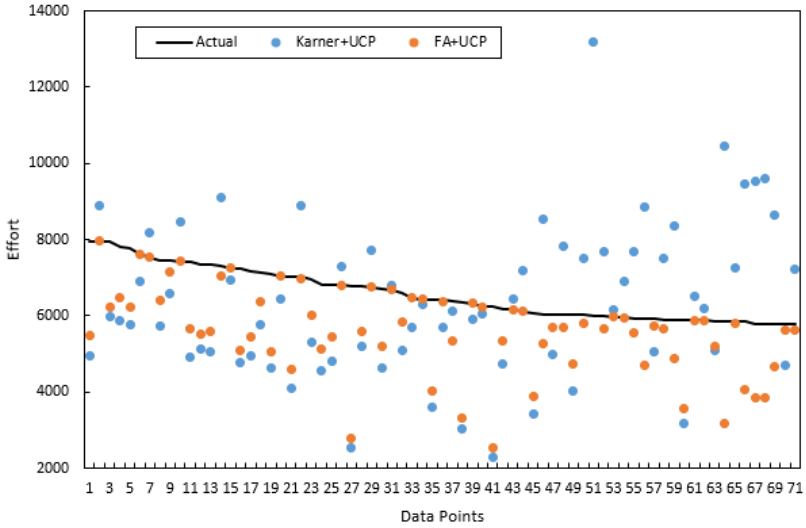

Fig. 9. The plot showing both the actual and estimated effort values between FA+UCP and Karner+UCP models

In order to assess whether the optimization method is effective in finding optimum solutions for UCP functions, further detailed analysis has been carried out. The convergence behavior comparison of each optimization method from test data number 3 is shown in Fig. 10. The comparison reveals that FA+UCP and PSO+UCP exhibit a faster convergence rate when compared with GWO+UCP, GA+UCP, and RSA+UCP since the second iteration.

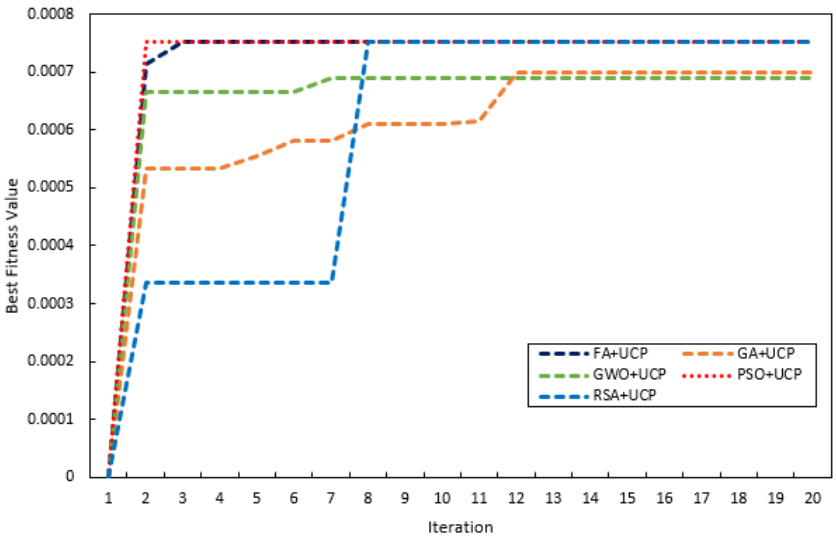

Fig. 10. Convergence behavior comparison of all optimization methods

Beyond convergence analysis, we also investigated diversity analysis for optimal solutions on the benchmark algorithms with UCP estimating methods, as shown in Fig. 11. We see that the median values of FA+UCP, PSO+UCP, and RSA+UCP are lower than those of GWO+UCP and GA+UCP. From the interquartile perspective, all optimization-based algorithms show a similar shape size, indicating that these models have the same spread.

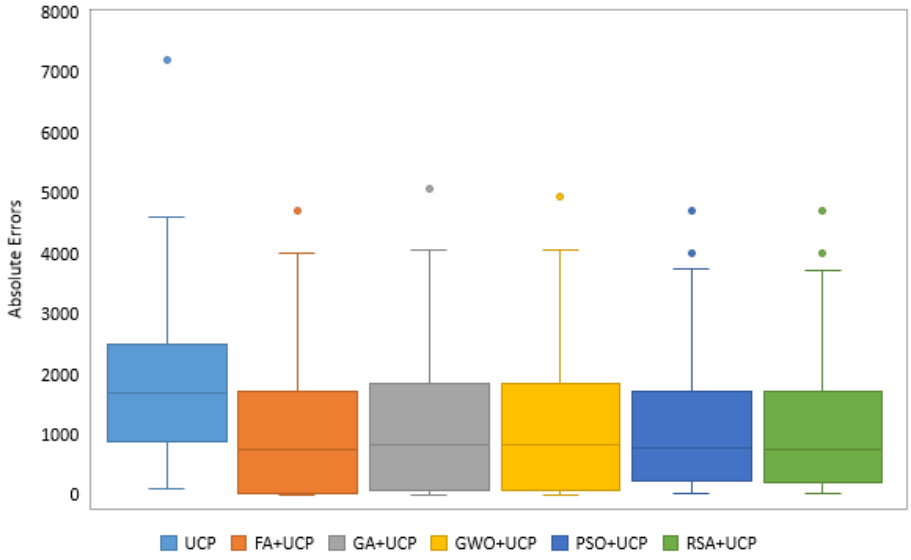

Fig. 11. Best solution diversity analysis of all models

As detailed in section 5.1, RSA and GWO consistently outperformed FA across all benchmark test functions. However, in the evaluation of UCP estimation, FA+UCP exhibited superior performance compared to RSA+UCP and GWO+UCP. This finding aligns with a prior study conducted by Ghatasheh *et al.* [19], which employed COCOMO as the estimation method and indicated that FA outperforms GA and PSO. The rationale behind this lies in the ability of FA+UCP to maximize the efficiency of UCP convergence by handling smaller dimensional sizes effectively. This is critical because larger dimensional sizes tend to result in suboptimal performance for optimization algorithms, as previously documented by Boussaïd *et al.* [64], and Halim *et al.* [65]. Furthermore, GWO+UCP underperformed compared to FA+UCP due to its optimal performance conditions being defined with population sizes of 50 and a maximum of 600 iterations [66]. In contrast, in our study, the parameter settings only three dimensions and a maximum of 20 iterations.

4.3. Threats to Validity

Simple, average, and complex actors in the dataset were determined by previous researchers and are publicly available. However, we are not aware as to how they were calculated. Thus, the accuracy of the actors cannot be confirmed. This is a possible threat to construct validity. This study utilizes only one dataset, which raises concerns about the generalizability of the results. Neverthless, the dataset employed is an industrial dataset created by proficient developers. Consequently, the findings may be applicable to software industrial practices. Furthermore, the dataset predominantly employ the waterfall development methodology, indicating that the conclusions drawn may not be transferable to the agile methodology.

One possible threat to external validity in this study is the set of metaheuristic algorithms explored (GA, PSO, GWO, RSA, and FA). Metaheuristic is vast and dynamic field, and any individual study can only utilize a limited subset of the numerous known metaheuristic algorithms. For instance, this study does not investigate Salp algorithms, which were emphasized by Tawhid and Ibrahim [67]. In practical terms, it is not feasible to examine all conceivables algorithms. The most we can do is establish our experimental procedure and anticipate that other researchers will implement it with a different set of metaheuristic algorithms.

5. Conclusion

This study suggests that the FA+UCP model exhibited outstanding results compared with GA+UCP, PSO+UCP, GWO+UCP, and RSA+UCP. As a result, the FA+UCP model introduced in this study can be valuable for improving the Use Case Point (UCP) estimation performance. Its high accuracy and ability to search appropriate use case complexity weight make it a promising tool for UCP to provide more accurate software effort estimation. Hence, the findings of this study hold practical implications for software project managers. They can utilize the UCP estimation method which is optimized using the Firefly algorithm. It must be borne in mind that the parameter configuration of FA+UCP in this study, which is $\alpha = 0.5, \gamma = 1$, was found to be weak in 12 classical test functions but excelled in the UCP effort estimation evaluation. Therefore, further research is needed to identify the optimal configuration for the Firefly algorithm to excel in both classical benchmark test functions and UCP effort estimation evaluations.

Acknowledgment

The authors would like to express their sincere gratitude to Direktorat Riset, Teknologi, dan Pengabdian kepada Masyarakat (DRTPM) for Grant Contract Number 002/PFR/LPPM UAD/VI/2023 for the financial support. This research would not have been possible without this grant's generous funding and resources. We also thank our colleagues and research collaborators for their valuable insights and contributions during this study.

Declarations

Author contribution. Conceptualization, Ardiansyah.; methodology, Ardiansyah.; formal analysis, M.I.Zulfa; software, Ardiansyah.; validation, M.I. Zulfa., A. Tarmuji., F.H.Jabbar.; investigation, Ardiansyah, M.I. Zulfa, A. Tarmuji.; resources, Ardiansyah.; data curation, Ardiansyah, F.H.Jabbar.; writing—original draft preparation, Ardiansyah; writing—review and editing, Ardiansyah, M.I.Zulfa, A.Tarmuji., F.H.Jabbar.; visualization, supervision, M.I.Zulfa, A. Tarmuji., F.H.Jabbar.; project administration, Ardiansyah. All authors have read and agreed to the published version of the manuscript.
Funding statement. This research was funded by Direktorat Riset, Teknologi, dan Pengabdian kepada Masyarakat (DRTPM) for Grant Contract Number 002/PFR/LPPM UAD/VI/2023 Year 2023.
Conflict of interest. The authors declare no conflict of interest.
Additional information. No additional information is available for this paper.

References

[1] A. Dennis, B. H. Wixom, and R. M. Roth, *Systems Analysis and Design*, 7th ed., no. Mi. Wiley, 2019. [Online]. Available at: https://hr.nih.gov/working-nih/competencies/competencies-dictionary/systems-analysis-and-design#:-:text=Analyzes the business needs and,(what the system does).

[2] M. Choetkiertikul, H. K. Dam, T. Tran, A. Ghose, and J. Grundy, "Predicting Delivery Capability in Iterative Software Development," *IEEE Trans. Softw. Eng.*, vol. 44, no. 6, pp. 551–573, Jun. 2018, doi: 10.1109/TSE.2017.2693989.

[3] B. Boehm, B. Clark, E. Horowitz, C. Westland, R. Madachy, and R. Selby, "Cost models for future software life cycle processes: COCOMO 2.0," *Ann. Softw. Eng.*, vol. 1, no. 1, pp. 57–94, 1995, doi: 10.1007/BF02249046.

[4] A. J. Albrecht, "Measuring Application Development Productivity," in *Proceedings of the Joint SHARE, GUIDE, and IBM Application Development Symposium*, 1979, pp. 83–92. [Online]. Available at: https://a.xueshu.baidu.com/usercenter/paper/show?paperid=cfe8018844c56557d39b005cb8daca3a.

[5] S. Shukla and S. Kumar, "Know-UCP: locally weighted linear regression based approach for UCP estimation," *Appl. Intell.*, vol. 53, no. 11, pp. 13488–13505, 2023, doi: 10.1007/s10489-022-04160-5.

[6] S. Shukla and S. Kumar, "Towards non-linear regression-based prediction of use case point (UCP) metric," *Appl. Intell.*, vol. 53, no. 9, pp. 10326–10339, May 2023, doi: 10.1007/s10489-022-04002-4.

[7] G. Karner, "Resource Estimation for Objectory Projects." p. 9, 1993. [Online]. Available at: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=17b5f04743cd13f6077fbdec227719e5d83dba10.

[8] Y. Xie, J. Guo, and A. Shen, "Use Case Points Method of Software Size Measurement Based on Fuzzy Inference," in *Proceedings of the 4th International Conference on Computer Engineering and Networks*, vol. 355, W. E. Wong, Ed. Cham: Springer International Publishing, 2015, pp. 11–18, doi: 10.1007/978-3-319-11104-9_2.

[9] F. Wang, X. Yang, X. Zhu, and L. Chen, "Extended Use Case Points Method for Software Cost Estimation," in *2009 International Conference on Computational Intelligence and Software Engineering*, pp. 1–5, Dec. 2009, doi: 10.1109/CISE.2009.5364706.

[10] A. B. Nassif, L. F. Capretz, and D. Ho, "Enhancing Use Case Points Estimation Method Using Soft Computing Techniques," *J. Glob. Res. Comput. Sci.*, vol. 1, no. 4, pp. 12–21, Dec. 2010. [Online]. Available at: https://arxiv.org/ftp/arxiv/papers/1612/1612.01078.pdf.

[11] L. Brezočnik, I. Fister, and V. Podgorelec, "Solving Agile Software Development Problems with Swarm Intelligence Algorithms," in *Lecture Notes in Networks and Systems*, vol. 76, E. Karabegovi, Ed. Sarajevo, Bosnia and Herzegovina: Springer, Cham, pp. 298–309, 2020, doi: 10.1007/978-3-030-18072-0_35.

[12] F. Ferrucci, C. Gravino, R. Oliveto, and F. Sarro, "Genetic Programming for Effort Estimation: An Analysis of the Impact of Different Fitness Functions," *2nd Int. Symp. Search Based Softw. Eng.*, no. 25, pp. 89–98, 2010, doi: 10.1109/SSBSE.2010.20.

[13] J. Murillo-Morera, C. Quesada-López, C. Castro-Herrera, and M. Jenkins, "A genetic algorithm based framework for software effort prediction," *J. Softw. Eng. Res. Dev.*, vol. 5, no. 1, p. 4, Dec. 2017, doi: 10.1186/s40411-017-0037-x.

[14] D. Wu, J. Li, and C. Bao, "Case-based reasoning with optimized weight derived by particle swarm optimization for software effort estimation," *Soft Comput.*, vol. 22, no. 16, pp. 5299–5310, Aug. 2018, doi: 10.1007/s00500-017-2985-9.

[15] T. R. Benala and R. Mall, "DABE: Differential evolution in analogy-based software development effort estimation," *Swarm Evol. Comput.*, vol. 38, no. May 2016, pp. 158–172, 2018, doi: 10.1016/j.swevo.2017.07.009.

[16] M. Dashti, T. J. Gandomani, D. H. Adeh, H. Zulzalil, and A. B. M. Sultan, "LEMABE: a novel framework to improve analogy-based software cost estimation using learnable evolution model," *PeerJ Comput. Sci.*, vol. 7, no. January, p. 24, 2021, doi: 10.7717/PEERJ-CS.800.

[17] Z. Shahpar, V. K. Bardsiri, and A. K. Bardsiri, "An evolutionary ensemble analogy-based software effort estimation," *Softw. Pract. Exp.*, vol. 52, no. 4, pp. 929–946, Apr. 2022, doi: 10.1002/spe.3040.

[18] Z. Shahpar, V. Khatibi, and A. Khatibi Bardsiri, "Hybrid PSO-SA Approach for Feature Weighting in Analogy-Based Software Project Effort Estimation," *J. AI Data Min.*, vol. 9, no. 3, pp. 329–340, 2021, doi: 10.22044/jadm.2021.10119.2152.

[19] N. Ghatasheh, H. Faris, I. Aljarah, and R. M. H. Al-Sayyed, "Optimizing Software Effort Estimation Models Using Firefly Algorithm," *J. Softw. Eng. Appl.*, vol. 08, no. 03, pp. 133–142, 2015, doi: 10.4236/jsea.2015.83014.

[20] V. Resmi, S. Vijayalakshmi, and R. S. Chandrabose, "An effective software project effort estimation system using optimal firefly algorithm," *Cluster Comput.*, vol. 22, no. S5, pp. 11329–11338, Sep. 2019, doi: 10.1007/s10586-017-1388-0.

[21] K. Langsari and R. Sarno, "Optimizing effort and time parameters of cocomo ii estimation using fuzzy multi-objective PSO," *Int. Conf. Electr. Eng. Comput. Sci. Informatics*, vol. 4, no. September, pp. 466–471, 2017, doi: 10.11591/eecsi.4.1047.

[22] N. A. Zakaria, A. R. Ismail, N. Z. Abidin, N. H. M. Khalid, and A. Y. Ali, "Optimized COCOMO parameters using hybrid particle swarm optimization," *Int. J. Adv. Intell. Informatics*, vol. 7, no. 2, p. 177, Apr. 2021, doi: 10.26555/ijain.v7i2.583.

[23] 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, doi: 10.3390/math8101819.

[24] 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. 01, p. 2050005, Mar. 2020, doi: 10.1142/S1469026820500054.

[25] M. Khazaiepoor, A. Khatibi Bardsiri, and F. Keynia, "A Hybrid Approach for Software Development Effort Estimation using Neural networks, Genetic Algorithm, Multiple Linear Regression and Imperialist Competitive Algorithm," *Int. J. Nonlinear Anal. Appl*, vol. 11, no. 1, pp. 2008–6822, 2020, doi: 10.22075/ijnaa.2020.4259.

[26] K. E. Rao and G. A. Rao, "Ensemble learning with recursive feature elimination integrated software effort estimation: a novel approach," *Evol. Intell.*, vol. 14, no. 1, pp. 151–162, Mar. 2021, doi: 10.1007/s12065-020-00360-5.

[27] A. Ardiansyah, R. Ferdiana, and A. E. Permanasari, "Optimizing complexity weight parameter of use case points estimation using particle swarm optimization," *Int. J. Adv. Intell. Informatics*, vol. 8, no. 2, p. 165, Jul. 2022, doi: 10.26555/ijain.v8i2.811.

[28] 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, doi: 10.3390/app12031081.

[29] M. R. Braz and S. R. Vergilio, "Using fuzzy theory for effort estimation of object-oriented software," in *16th IEEE International Conference on Tools with Artificial Intelligence*, no. Ictai, pp. 196–201, 2004, doi: 10.1109/ICTAI.2004.119.

[30] G. Robiolo and R. Orosco, "Employing use cases to early estimate effort with simpler metrics," *Innov. Syst. Softw. Eng.*, vol. 4, no. 1, pp. 31–43, Apr. 2008, doi: 10.1007/s11334-007-0043-y.

[31] P. Mohagheghi, B. Anda, and R. Conradi, "Effort estimation of use cases for incremental large-scale software development," in *Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005.*, 2005, pp. 303–311, doi: 10.1109/ICSE.2005.1553573.

[32] B. Anda, H. Dreiem, D. I. K. Sjoberg, and M. Jorgensen, "Estimating Software Development Effort based on Use Cases – Experiences from Industry," in *The Unified Modeling Language. Modeling Languages, Concepts, and Tools*, C. G. Kobryn, Ed. Springer Berlin Heidelberg, 2001, pp. 487–502, 2001, doi: 10.1007/3-540-45441-1_35.

[33] B. Anda, E. Angelvik, and K. Ribu, "Improving Estimation Practices by Applying Use Case Models," *Prod. Focus. Softw. Process Improv.*, vol. 2559, no. 1325, pp. 383–397, 2002, doi: 10.1007/3-540-36209-6_32.

[34] M. Ochodek, J. Nawrocki, and K. Kwarciak, "Simplifying effort estimation based on Use Case Points," *Inf. Softw. Technol.*, vol. 53, no. 3, pp. 200–213, Mar. 2011, doi: 10.1016/j.infsof.2010.10.005.

[35] M. Ochodek, B. Alchimowicz, J. Jurkiewicz, and J. Nawrocki, "Improving the reliability of transaction identification in use cases," *Inf. Softw. Technol.*, vol. 53, no. 8, pp. 885–897, 2011, doi: 10.1016/j.infsof.2011.02.004.

[36] H. L. T. K. Nhung, V. Van Hai, R. Silhavy, Z. Prokopova, and P. Silhavy, "Parametric Software Effort Estimation Based on Optimizing Correction Factors and Multiple Linear Regression," *IEEE Access*, vol. 10, pp. 2963–2986, 2022, doi: 10.1109/ACCESS.2021.3139183.

[37] A. B. Nassif, D. Ho, and L. F. Capretz, "Towards an early software estimation using log-linear regression and a multilayer perceptron model," *J. Syst. Softw.*, vol. 86, no. 1, pp. 144–160, Jan. 2013, doi: 10.1016/j.jss.2012.07.050.

[38] A. B. Nassif, L. F. Capretz, D. Ho, and M. Azzeh, "A treeboost model for software effort estimation based on use case points," *Proc. - 2012 11th Int. Conf. Mach. Learn. Appl. ICMLA 2012*, vol. 2, no. June, pp. 314–319, 2012, doi: 10.1109/ICMLA.2012.155.

[39] K. Qi, A. Hira, E. Venson, and B. W. Boehm, "Calibrating use case points using bayesian analysis," in *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement - ESEM '18*, 2018, pp. 1–10, doi: 10.1145/3239235.3239236.

[40] R. Silhavy and P. Silhavy, "Using Actors and Use Cases for Software Size Estimation," *Electronics*, vol. 10, no. 5, pp. 1–20, 2021, doi: 10.3390/electronics10050592.

[41] H. Le Thi Kim Nhung, H. T. Hoc, and V. Van Hai, "An Evaluation of Technical and Environmental Complexity Factors for Improving Use Case Points Estimation," in *Advances in Intelligent Systems and Computing*, vol. 1294, 2020, pp. 757–768, doi: 10.1007/978-3-030-63322-6_64.

[42] K. Qi and B. W. Boehm, "Detailed use case points (DUCPs)," in *Proceedings of the 10th International Workshop on Modelling in Software Engineering - MiSE '18*, 2018, pp. 17–24, doi: 10.1145/3193954.3193955.

[43] D. D. Galorath and M. W. Evans, *Software Sizing, Estimation, and Risk Management*. Boca Raton: Auerbach Publications, p. 576, 2006, doi: 10.1201/9781420013122.

[44] K. Periyasamy and A. Ghode, "Cost Estimation Using Extended Use Case Point (e-UCP) Model," in *2009 International Conference on Computational Intelligence and Software Engineering*, Dec. 2009, pp. 1–5, doi: 10.1109/CISE.2009.5364515.

[45] A. Minkiewicz, "Use Case Sizing." PRICE Research, 2004. [Online]. Available at: https://studylib.net/doc/14990688/use-case-sizing#google_vignette.

[46] A. B. Nassif, "Software Size and Effort Estimation from Use Case Diagrams Using Regression and Soft Computing Models," The University of Western Ontario, p. 233, 2012. [Online]. Available at: https://ir.lib.uwo.ca/cgi/viewcontent.cgi?referer=&httpsredir=1&article=1732&context=etd.

[47] A. B. Nassif, L. F. Capretz, and D. Ho, "Calibrating use case points," in *Companion Proceedings of the 36th International Conference on Software Engineering - ICSE Companion 2014*, 2014, pp. 612–613, doi: 10.1145/2591062.2591141.

[48] H. T. Hoc, V. Van Hai, and H. Le Thi Kim Nhung, "AdamOptimizer for the Optimisation of Use Case Points Estimation," in *Advances in Intelligent Systems and Computing*, vol. 1294, 2020, pp. 747–756, doi: 10.1007/978-3-030-63322-6_63.

[49] M. Hariyanto and R. S. Wahono, "Software Development Project Estimation Using Fuzzy Use Case Points," *J. Softw. Eng.*, vol. 1, no. 1, pp. 54–63, 2015. [Online]. Available at: https://journal.ilmukomputer.org/index.php?journal=jse&page=article&op=view&path%5B%5D=35.

[50] S. Mirjalili, S. M. Mirjalili, and A. Lewis, "Grey Wolf Optimizer," *Adv. Eng. Softw.*, vol. 69, pp. 46–61, Mar. 2014, doi: 10.1016/j.advengsoft.2013.12.007.

[51] X. S. Yang, "Firefly algorithms for multimodal optimization," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 5792 LNCS, pp. 169–178, 2009, doi: 10.1007/978-3-642-04944-6_14.

[52] Xin-She Yang, *Nature Inspired Metaheuristic Algorithms*, 2nd ed. Luniver Press, p. 75, 2010. [Online]. Available at: https://www.researchgate.net/publication/235979455_Nature-Inspired_Metaheuristic_Algorithms.

[53] J. H. Holland, *Adaptation in Natural and Artificial Systems*. London: MIT Press, p. 211, 1992, doi: 10.7551/mitpress/1090.001.0001.

[54] R. Silhavy, P. Silhavy, and Z. Prokopova, "Analysis and selection of a regression model for the Use Case Points method using a stepwise approach," *J. Syst. Softw.*, vol. 125, pp. 1–14, Mar. 2017, doi: 10.1016/j.jss.2016.11.029.

[55] E. Kocaguneli and T. Menzies, "Software effort models should be assessed via leave-one-out validation," *J. Syst. Softw.*, vol. 86, no. 7, pp. 1879–1890, 2013, doi: 10.1016/j.jss.2013.02.053.

[56] W. B. Langdon, J. Dolado, F. Sarro, and M. Harman, "Exact Mean Absolute Error of Baseline Predictor, MARP0," *Inf. Softw. Technol.*, vol. 73, pp. 16–18, May 2016, doi: 10.1016/j.infsof.2016.01.003.

[57] M. Shepperd and S. MacDonell, "Evaluating prediction systems in software project estimation," *Inf. Softw. Technol.*, vol. 54, no. 8, pp. 820–827, Aug. 2012, doi: 10.1016/j.infsof.2011.12.008.

[58] M. H. Nadimi-Shahraki, S. Taghian, and S. Mirjalili, "An improved grey wolf optimizer for solving engineering problems," *Expert Syst. Appl.*, vol. 166, no. August 2020, p. 113917, 2021, doi: 10.1016/j.eswa.2020.113917.

[59] H. Peng, W. Zhu, C. Deng, and Z. Wu, "Enhancing firefly algorithm with courtship learning," *Inf. Sci. (Ny).*, vol. 543, pp. 18–42, 2020, doi: 10.1016/j.ins.2020.05.111.

[60] L. Abualigah, M. A. Elaziz, P. Sumari, Z. W. Geem, and A. H. Gandomi, "Reptile Search Algorithm (RSA): A nature-inspired meta-heuristic optimizer," *Expert Syst. Appl.*, vol. 191, no. November, p. 116158, Apr. 2022, doi: 10.1016/j.eswa.2021.116158.

[61] M. Azzeh and A. B. Nassif, "A hybrid model for estimating software project effort from Use Case Points," *Appl. Soft Comput. J.*, vol. 49, pp. 981–989, 2016, doi: 10.1016/j.asoc.2016.05.008.

[62] R. Silhavy, P. Silhavy, and Z. Prokopova, "Evaluating subset selection methods for use case points estimation," *Inf. Softw. Technol.*, vol. 97, no. June 2017, pp. 1–9, 2018, doi: 10.1016/j.infsof.2017.12.009.

[63] J. Demsar, "Statistical Comparisons of Classifiers over Multiple Data Sets," *J. Mach. Learn. Res.*, vol. 7, pp. 1–30, 2006. [Online]. Available at: https://www.jmlr.org/papers/volume7/demsar06a/demsar06a.pdf.

[64] I. Boussaïd, J. Lepagnot, and P. Siarry, "A survey on optimization metaheuristics," *Inf. Sci. (Ny).*, vol. 237, pp. 82–117, Jul. 2013, doi: 10.1016/j.ins.2013.02.041.

[65] A. H. Halim, I. Ismail, and S. Das, *Performance assessment of the metaheuristic optimization algorithms: an exhaustive review*, Springer Netherlands, vol. 54, no. 3, pp. 2323–2409, 2021, doi: 10.1007/s10462-020-09906-6.

[66] J. O. Agushaka, A. E. Ezugwu, L. Abualigah, S. K. Alharbi, and H. A. E. W. Khalifa, "Efficient Initialization Methods for Population-Based Metaheuristic Algorithms: A Comparative Study," *Archives of Computational Methods in Engineering*, vol. 30, no. 3. pp. 1727–1787, 2023, doi: 10.1007/s11831-022-09850-4.

[67] M. A. Tawhid and A. M. Ibrahim, "Improved salp swarm algorithm combined with chaos," *Math. Comput. Simul.*, vol. 202, pp. 113–148, 2022, doi: 10.1016/j.matcom.2022.05.029.