Thanh Tung Khuat* and My Hanh Le

A Novel Hybrid ABC-PSO Algorithm for Effort Estimation of Software Projects Using Agile Methodologies

DOI 10.1515/jisys-2016-0294
Received November 11, 2016; previously published online March 23, 2017.

Abstract: In modern software development processes, software effort estimation plays a crucial role. The success or failure of projects depends greatly on the accuracy of effort estimation and schedule results. Many studies focused on proposing novel models to enhance the accuracy of predicted results; however, the question of accurate estimation of effort has been a challenging issue with regards to researchers and practitioners, especially when it comes to projects using agile methodologies. This study aims at introducing a novel formula based on team velocity and story point factors. The parameters of this formula are then optimized by employing swarm optimization algorithms. We also propose an improved algorithm combining the advantages of the artificial bee colony and particle swarm optimization algorithms. The experimental results indicated that our approaches outperformed methods in other studies in terms of the accuracy of predicted results.

Keywords: Software effort estimation, agile software development, user story, particle swarm optimization, artificial bee colony, swarm optimization algorithm.

2010 Mathematics Subject Classification: 68T20, 68T35, 68N01, 68W25.

1 Introduction

Software effort estimation has an important role to play in a software development process [35] because the success or failure of a project relies greatly on the accuracy of effort and schedule estimates. According to the International Society of Parametric Analysis [10] and the Standish Group International [23], two-thirds of software projects fail to be delivered on time and within budget. There are two key reasons, including (i) improper estimation in terms of project size, cost, and human resource needed for projects and (ii) uncertainty of software requirements. Hence, it is desired to seek an approach to predict efforts for software projects accurately. Underestimating the costs might lead to the approval of proposed systems that then exceed their budgets, causing poor quality and failure to complete on time. In contrast, overestimating may result in too many resources committed to a project that brings about not winning the contract, which can lead to the loss of jobs. A wide range of studies introduced many methods for software effort estimation, ranging from expert judgment [15], algorithmic models such as COCOMO [5], SLIM [29] to machine learning-based techniques like fuzzy logic [22], and neural networks [24].

To overcome the uncertainty of software requirements, agile software development methods have been proposed, because of their inherent advantages such as iterative development, rapid delivery, reduced risk, and enabling organizations to respond to requirement volatility. A concerning issue, therefore, is how to efficiently estimate the effort necessary to develop projects adopting agile methodologies. Size is a primary factor for many models used to estimate effort. In agile projects, story points are relative measures widely

***Corresponding author: Thanh Tung Khuat,** The University of Danang, University of Science and Technology, Danang, Vietnam, e-mail: thanhtung09t2@gmail.com
My Hanh Le: The University of Danang, University of Science and Technology, Danang, Vietnam

used to determine the size of a project, as well as how difficult tasks are. In addition to story points, the velocity of a project team, which is the total number of story points that the team can convey in a sprint, is also a factor that affects efforts of the project under development. Based on story points and team velocity, we will predict efforts that need to be provided to the software project using the agile methodology effectively. This paper focuses on constructing a formula for agile software effort estimation, and then swarm optimization algorithms are used to find out suitable parameters of the proposed model based on historical software data.

Our contributions in this paper include the following:

- We propose a novel formula for agile software effort estimation based on velocity and story points.
- We apply swarm optimization algorithms including particle swarm optimization (PSO) and artificial bee colony (ABC) to find parameters of the estimation model.
- We devise a hybrid version of PSO and ABC to enhance the accuracy of the proposed estimation model.
- We evaluate the efficiency of the proposed approaches compared with other methods.

The rest of this paper is organized as follows. Section 2 briefly describes work related to our study, while Section 3 gives information about factors used for agile software effort estimation. Section 4 shows swarm optimization algorithms and a proposed hybrid algorithm. Section 5 presents our proposed estimation model and how to apply swarm optimization algorithms for solving the problem. Experimental results are shown in Section 6. Section 7 gives information about threats to the validity of the proposed methods, and Section 8 concludes with the obtained results of the study.

2 Related Work

There are many papers and studies in the effort estimation for software projects using agile methodologies. Keaveney and Conboy [19] researched on the applicability of traditional estimation approaches to agile projects by focusing on four case studies using agile methodologies of different organizations. The authors utilized the main estimation techniques, being expert knowledge and analogy to past projects. The experimental results indicated that the estimation inaccuracy using the proposed method was a less frequent occurrence compared to the use of traditional approaches. Hussain et al. [14] provided an efficient approach that helps in removing problems like formalized user requirements, and thus function points might be applied for agile software effort estimation. Andreas et al. [2] introduced a method to predict the effort in agile software development with an investigation about estimation possibilities, especially for Extreme Programming (XP). The authors focused on the characteristics of agile methodologies, and provided guidelines for measurement aspects within XP projects. The proposed approach was assessed using a survey including 17 questions. Each question considers the influence of agile methodologies to effort estimation. The survey showed that the benefit of agile methods is hard to evaluate and need further studies, as well as the costs of maintenance projects must be taken into consideration. Coelho and Basu [7] gave an overview of different effort estimation techniques based on story points for the agile software development process. The authors introduced steps followed in the story point-based approach, and highlighted the area that needs to be studied further. Popli and Chauhan [28] proposed a model for effort and cost estimation in agile software development by applying regression analysis. This technique is appropriate for project planning, execution, and monitoring effectively.

The factors having an impact on the effort that is necessary to develop a software project using agile methodologies are subjects of studies as well. In Ref. [1], Abrahamsson and Koskela described the way to collect metrics to measure the productivity, quality and schedule estimation, and cost and effort estimation for an agile software development project using XP. The authors provided evidence that agile methods are efficient and suitable for a variety of situations and environments. Hamouda [12] proposed a process and methodology assuring relativity in software sizing while using agile story points on the level of the Capability Maturity Model Integration (CMMI) organizations. Zia et al. [41] introduced an effort estimation model for agile software development to combine most characteristics of agile methodologies, especially adaption and iteration, where it is concentrated on user stories as a basis for estimation. The model was evaluated using

the empirical data collected from 21 software projects. The experimental results indicated that the model provided acceptable estimation accuracy in terms of the mean magnitude of relative error (MMRE). Nevertheless, the regression approach used in their work to generate the predicted effort from team velocity and story point of projects can still be improved to fortify the accuracy of estimates. Our study focuses on proposing an estimation formula for agile software projects, and then the parameters of this model will be optimized adopting swarm optimization algorithms.

Machine learning techniques were also applied to software effort estimation problems. Oliveira [26] provided a comparative research on support vector regression (SVR), radial basis function neural networks (RBFNs), and the linear regression for the estimation of software development effort. Their experiment was conducted on NASA project data sets, and the experimental results showed that SVR performed better than RBFN and linear regression analysis. Satapathy et al. [31] predicted the effort of agile software projects using the story point approach, in which the total number of story points and project velocity were used to estimate the effort involved in developing an agile software product. The obtained results were optimized by applying four different SVR kernel methods. The authors concluded that the radial basis function kernel-based SVR technique outperformed three other kernel methods. In Ref. [27], Panda et al. attempted to ameliorate the prediction accuracy of the agile software effort estimation process proposed by Zia et al. To solve this problem, different kinds of neural networks consisting of general regression neural network (GRNN), probabilistic neural network (PNN), group method of data handling (GMDH) polynomial neural network, and cascade-correlation neural network were used and compared. In our previous work [20], we improved the accuracy of the estimation model of Zia et al. using an artificial neural network (ANN) optimized by the combination of the fireworks and Levenberg-Marquardt algorithms. The experimental results overcame the results of Panda et al. [27]. Table 1 summarizes studies related to the effort estimation for agile software projects.

In this paper, a hybrid algorithm of ABC and PSO is proposed to optimize parameters of the estimation formula. ABC and PSO were combined in some literature. In Ref. [34], Shi et al. introduced a way to integrate ABC and PSO by executing two their subsystems in parallel. Information exchange from bee colony to particle swarm occurs in the scout bee phase or when the particle velocity is updated at a certain probability

Table 1: Related Work on the Effort Estimation for Agile Software Projects.

Author	Year	Metric	Method and algorithm
Abrahamsson and Koskela [1]	2004	Lines of code, user stories	A survey of the empirical data obtained from a controlled case study on extreme programming in practical settings
Keaveney and Conboy [19]	2006	Past project data	Expert knowledge and analogy to past projects
Oliveira [26]	2006	Developed lines and methodology	Linear regression, RBFN, and SVR
Andreas et al. [2]	2008	A survey with 17 questions	Interviewing and conducting an analysis of the effort estimation possibilities within agile software development methodologies using a survey
Coelho and Basu [7]	2012	Story points	Introducing steps followed in the story point-based approach, and highlighting the area that needs to be studied further
Zia et al. [41]	2012	Story points and project velocity	Linear regression
Hussain et al. [14]	2013	Function points	Using COSMIC functional size measurement method and a supervised text mining approach from user requirements
Popli and Chauhan [28]	2014	Story points and velocity	Regression analysis
Hamouda [12]	2014	Software size and story points	Proposing a process and methodology assuring relativity in software sizing and story points for CMMI organizations
Satapathy et al. [31]	2014	Story points and project velocity	Four different SVR kernel methods
Panda et al. [27]	2015	Story points and project velocity	GRNN, PNN, GMDH polynomial neural network, and cascade-correlation neural network
Khuat and Le [20]	2016	Story points and team velocity	Multilayer neural network optimized by fireworks and Levenberg-Marquardt algorithms

level. El-Abd [11] introduced another way to combine ABC with PSO. In El-Abd's algorithm, PSO is run first to update position, velocity, and local best position for each individual. After that, the best location of each particle is changed using ABC update rules. In Ref. [21], Kiran and Gunduz introduced a crossover operation-based hybridization of PSO and ABC called the HPA. In their algorithm, the best solutions of the populations obtained at each iteration of the PSO and ABC are recombined to generate an individual called *TheBest*, and this solution is taken as *Gbest* for PSO and neighbors of onlooker bees for ABC to increase the exploration and exploitation abilities of HPA. Hence, the social structures of ABC and PSO have been reinforced and provide HPA with a better global search process. Wang et al. [38] used these combination ways together with a feed-forward neural network to construct classification methods for abnormal brain detection, and obtained interesting outcomes. In this work, we associate PSO with employed bee and onlooker bee phases of the ABC algorithm using a new formula to seek neighboring food sources.

3 An Effort Estimation Model for Agile Projects

In Ref. [41], Zia et al. proposed a model to estimate the effort of agile software projects. This model uses story points and team velocity to predict the effort for a project.

3.1 Computing the Story Point of an Agile Project

Story points are a number of user stories associated with their complexities completed in a unit time. To calculate the story points of an agile project, we first determine the story sizes and the complexity of each story size. The story size is an estimate of the relative scale of tasks with regard to the actual development effort. Each story size is assigned a value from 1 to 5 based on its scale. Value 1 shows that the story is very small, which needs a tiny effort level with only a few working hours. Value 2 indicates that it is expected to finish a user story in 1 or 2 days of working; meanwhile, value 3 presents that we need from 2 to 5 working days to complete a user story. Value 4 is given to the story having a very large size and requiring more than a week of working to accomplish, and we need to consider breaking it down into a set of smaller stories. Value 5 represents an extremely large story, and it is really difficult to estimate time accurately. After specifying the scale of the story, we have to consider its complexity. The complexity is also measured by five values assigned to the user story according to its nature. Value 1 states that the story requires basic programming skills to complete, and their technical and business requirements are very clear with no ambiguity. Value 5 shows that the story is extremely complex with many dependencies on other stories, systems, or subsystems, and it needs a set of skills or experience that is important but absent in the team along with the extensive research and significant refactoring. The details of the user story complexity are clearly described in Ref. [41].

The total story points (S_P) for N user stories of a project are computed using Eq. (1).

$$S_P = \sum_{i=1}^{N} C_i \cdot S_i, \tag{1}$$

where C_i and S_i are the complexity and size of the i^{th} project, respectively.

3.2 Determining Agile Velocity

The initial agile velocity of a team is simply how many units of effort that this team might complete in a typical sprint. It is also defined as how many story points that a team can handle in one sprint, and it is determined as follows:

$$V_i = \text{units of effort completed} / \text{sprint time}.$$

In practice, the velocity of a project is not only simply measured by units of effort and sprint time, but it is also influenced by two other factors including friction and variable or dynamic forces.

The friction forces are constants that drag on productivity and reduce the project velocity. They consist of team composition, process, environmental factors, and team dynamics. Their influences are long term; however, they are easy to deal with. Table 2 gives information about four friction factors with a range of values, and these values have been tuned following their risk severity [41].

The value of friction (*FR*) is computed as the product of all four friction factors (*FF*) using Eq. (2):

$$FR = \prod_{i=1}^{4} FF_i. \tag{2}$$

The variable or dynamic forces decelerate the project or the performance of team members, and bring about the project velocity to be irregular. These forces are usually unpredictable and unexpected. They include team changes, new tools requiring learning, vendor defects, responsibilities outside of the project of team members, personal issues, stakeholders, unclear requirements, changing requirements, and relocation. Table 3 describes variable or dynamic force factors and the values associated with them on the basis of same analogy as for size.

Dynamic force (*DF*) is then computed as the product of all nine variable factors (*VF*) using Eq. (3):

$$DF = \prod_{i=1}^{9} VF_i. \tag{3}$$

Deceleration of an agile software project is the product of friction and dynamic forces impacting the velocity as Eq. (4):

$$D = FR \cdot DF. \tag{4}$$

The final velocity of a project under the influence of friction and dynamic forces is computed using Eq. (5):

$$V = (V_i)^D. \tag{5}$$

Table 2: Friction Factors.

Friction factor	Stable	Volatile	Highly volatile	Very highly volatile
Team composition	1	0.98	0.95	0.91
Process	1	0.98	0.94	0.89
Environmental factors	1	0.99	0.98	0.96
Team dynamics	1	0.98	0.91	0.85

Table 3: Dynamic Force Factors.

Variable factor	Normal	High	Very high	Extra high
Expected team changes	1	0.98	0.95	0.91
Introduction of new tools	1	0.99	0.97	0.96
Vendor's defects	1	0.98	0.94	0.90
Team member's responsibility outside the project	1	0.99	0.98	0.98
Personal issues	1	0.99	0.99	0.98
Expected delay in stakeholder response	1	0.99	0.98	0.96
Expected ambiguity in details	1	0.98	0.97	0.95
Expected changes in environment	1	0.99	0.98	0.97
Expected relocation	1	0.99	0.99	0.98

From the team velocity and story points of a project, Zia et al. [41] used the regression method to predict the duration needed to complete the project. In this paper, we devise a novel formula for agile software effort estimation based on two factors: final velocity and story point. The parameters of this model are then optimized using swarm optimization algorithms and a set of data shown in Ref. [41].

4 Swarm Optimization Algorithms

In this paper, we consider two most common swarm optimization algorithms including ABC and PSO. A hybrid version of these two algorithms is then presented based on the advantages of each algorithm to improve the accuracy of predicted results for software effort.

4.1 ABC Algorithm

The ABC algorithm was first proposed by Karaboga and Basturk [16] based on simulating intelligent behaviors of real honey bee colonies. There are two various honey bee categories sharing knowledge to successfully locate such sources. The employed bees first exploit food sources and then give their information about the quality of the food sources to unemployed bees. Unemployed bees consist of scout bees, which search for a new food source randomly when current food source is exhausted, and onlookers, which wait at the nest and establish communication with the employed bees.

In ABC, a swarm includes three kinds of bees, which are employed bees, scouts, and onlookers. The number of food sources is equal to the number of employed bees, and the number of employed bees is also equal to the number of onlooker bees. At the beginning phase, all employed bees in the population are scout bees, and their food source positions are randomly initialized using Eq. (6):

$$x_{ij} = lb_j + rnd \cdot (ub_j - lb_j), \tag{6}$$

where $i = 1, 2, \ldots, N_E$, and $j = 1, 2, \ldots, D$. x_{ij} is the j^{th} dimension of the i^{th} food source, which will be assigned to the i^{th} employed bee. lb_j and ub_j are the lower and upper bounds of the j^{th} dimension respectively, rnd is a random number in the range of [0, 1], N_E is the number of employed bees, and D is the dimensionality of the problem.

After the initialization phase, all scout bees become employed bees, and the quality of food sources of employed bees is assessed using Eq. (7):

$$\text{fit}_i = \begin{cases} \dfrac{1}{1+f_i}, & \text{if } f_i \geq 0 \\ 1 + abs(f_i), & \text{if } f_i < 0 \end{cases}, \tag{7}$$

where fit_i is the fitness of the i^{th} food source and f_i is the objective function value specific for the problem.

The employed bees start to seek around the self-food sources for new food sources. A new food source position around current food source of each employed bee is obtained using Eq. (8):

$$v_{ij} = x_{ij} + \phi_{ij} \cdot (x_{ij} - x_{kj}), \tag{8}$$

where $i \in \{1, 2, \ldots, N_E\}$, j is a random value in the range of $[1, D]$, k is the index of a randomly chosen individual ($k \neq i$), and ϕ_{ij} is a random number uniformly distributed in the range of $[-1, 1]$. After that, the fitness values of v_i and X_i are compared with each other. If v_i is better than X_i, the position of the new food source will replace the old food source X_i, and the trial counter of the food source is reset; otherwise, the trial counter of the food source X_i is increased by 1. This is a greedy selection mechanism.

After the employed bees return to the hive, the employed bees share self-food source positions with the onlooker bees. An onlooker bee opts an employed bee and memorizes its food source position to eliminate its food source by using roulette-wheel selection mechanism, given as follows:

$$p_i = \frac{\text{fit}_i}{\sum_{j=1}^{N_E} \text{fit}_j}, \tag{9}$$

where p_i is the probability that the i^{th} employed bee is selected by an onlooker bee. Intuitively, employed bees with higher values of fitness will have more chances to be chosen. After choosing an employed bee, the onlooker bee searches around the food source position of that employed bee using Eq. (8). If the fitness of the food source found by the onlooker bee is better than the old one, the employed bee memorizes the new food source position of the onlooker bee, and the trial counter of this food source is reset; otherwise, the trial counter of the food source is increased by 1.

The occurrence of the scout bee phase in ABC depends on the user-defined limit value and trial counters of the food sources. After the onlooker bee phase, if the maximal value of the trial counter of any food source is higher than the limit, a scout bee will seek a new food source position using Eq. (6). It is noticed that only one scout bee can occur at the each ABC iteration. The cycle of ABC terminates when the maximum iteration number is met or an error tolerance happens. The details of ABC are shown in Algorithm 1.

ABC has been used in many applications in several different fields. One of the most interesting application areas is training neural networks [18, 39]. ABC was also adopted by some researchers to solve the optimization problems encountered in electrical engineering. In Ref. [30], Rao et al. presented a new technique

Algorithm 1: The Pseudo Code of the ABC Algorithm.

Input:
- The maximum cycle number: *MCN*
- The number of employed bees: N_E
- The number of trials for abandoning food source: *limit*
- The dimensionality: *D*

Output: The best individual in the population: $\vec{X}_{best} = \{x_1, x_2, \ldots, x_D\}$.

Initialize a population of solutions $X_i = \{x_{ij}\}$, $i = 1, \ldots, N_E$, $j = 1, \ldots, D$
Compute fitness value for each X_i using Eq. (7)
cycle = 1
while *cycle* ≤ *MCN* **do**
  for $i = 1$ to N_E **do**
    – Generate a new solution v_i for the employed bee X_i using Eq. (8)
    – Compute fitness value for each v_i using Eq. (7)
    – Apply the greedy selection process
  end for
  For $i = 1$ to N_E **do**
    – Compute the probability value p_i for the solution X_i using Eq. (9)
  end for
  Formulate the set of potential solutions *Sol* by using the roulette-wheel selection mechanism to select N_E solutions in the population based on the probability value p_i
  for each solution X_i in *Sol* **do**
    – Generate a new solution v_i for the employed bee X_i using Eq. (8)
    – Compute fitness value for each v_i using Eq. (7)
    – Apply the greedy selection process
  end for
  for $i = 1$ to N_E **do**
    if value *limit* of solution X_i is reached **then**
      Produce a random solution and replace X_i with this solution
      break;
    end if
  end for
  Memorize the best solution achieved so far
  cycle = *cycle* + 1
end while

applying an ABC algorithm for determining the sectionalizing switch to be operated to deal with the distribution system loss minimization problem. Bijami et al. [4] used ABC for simultaneous coordinated tuning of two power system stabilizers to damp the power system inter-area oscillations. An interesting application area of ABC is data mining, such as clustering [17], feature selection [36], and knowledge discovery [13]. A comprehensive survey presenting ABC and its applications in detail can be found in Ref. [18]. In this study, we apply ABC to another field, which is software effort estimation.

4.2 Particle Swarm Optimization

PSO, which was proposed by Eberhart and Kennedy [9], is a population-based optimization algorithm, where the system is initialized with a population of random particles, and the algorithm searches for optima by updating generations. This algorithm works as follows: each particle in PSO represents a bird corresponding to a solution, and it has a fitness value computed by a fitness function. Particles have velocity information leading them in the search area. The algorithm is started with a certain number of random generated particles. Suppose that the search space is D-dimensional. The position of the i^{th} particle might be represented by a D-dimensional vector, $X_i = \{x_{i1}, \ldots, x_{iD}\}$, and the velocity of this particle is $v_i = \{v_{i1}, \ldots, v_{iD}\}$. Each particle seeks the most suitable solution in the search space by updating its velocity and position information using Eqs. (10) and (11), respectively.

$$v_{ij} = w \cdot v_{ij} + c_1 \cdot r_1 \cdot (Pbest_{ij} - x_{ij}) + c_2 \cdot r_2 \cdot (Gbest_j - x_{ij}), \tag{10}$$

$$x_{ij} = x_{ij} + v_{ij}, \tag{11}$$

where v_{ij} and x_{ij} are the velocity and the position of the j^{th} dimension of the i^{th} particle, respectively. The constant w, which is called inertia weight, plays the role to balance between the global search ability and local search ability [33]. c_1 and c_2 are the acceleration coefficients influencing the maximum size of the step that a particle can take in each iteration. r_1 and r_2, which are the random numbers in the range of [0, 1], affect the stochastic nature of the algorithm [3]. $Pbest_i$ is the best position that the i^{th} particle has visited, while $Gbest$ is the best position in the whole population. One has the following [40]:

$$Pbest(i, t) = \underset{k=1,..,t}{arg\ min}[f(X_i(k))], \quad i \in \{1, .., N_P\}, \tag{12}$$

$$Gbest(t) = \underset{\substack{k=1,..,t \\ i=1,...,N_P}}{arg\ min}[f(X_i(k))], \tag{13}$$

where i is the index of particle, N_P denotes the total number of particles in the swarm, t is the current iteration number, f denotes fitness function, and X_i is the position of the i^{th} particle.

After changing the position, each particle updates its personal best position using Eq. (14):

$$Pbest_i^{t+1} = \begin{cases} Pbest_i^t, & \text{if } f(Pbest_i^t) \le f(X_i^{t+1}) \\ X_i^{t+1}, & \text{if } f(Pbest_i^t) > f(X_i^{t+1}) \end{cases}. \tag{14}$$

Finally, the global best of the swarm is updated using Eq. (15).

$$Gbest^{t+1} = arg \quad min\, f(Pbest^{t+1}). \tag{15}$$

The PSO is shown in detail in Algorithm 2.

Similar to ABC, PSO was applied to several different fields of research. The first practical application of PSO was in the field of ANN training [9]. Since then, there are thousands of publications reporting application of PSO to other fields such as electrical, electronic engineering and automatic control, communication, operations, fuel and energy, medical engineering, and chemical and biological engineering. A recent survey

Algorithm 2: The Pseudo Code of the PSO Algorithm.

Input: *Max_Iterations*, w, c_1, c_2, N_p
Output: The best individual in the population: $\vec{X}_{best} = \{x_1, x_2, \ldots, x_D\}$.

– Initialize a swarm including N_p particles with positions $X_i = \{x_{ij}\}$, $i = 1, \ldots, N_p$, $j = 1, \ldots, D$ with a uniformly distribution using Eq. (6)
– Initialize *Pbest* of each particle to its initial position: $Pbest(i, 0) = X_i(0)$
– Initialize *Gbest* to the minimal value of the swarm: $Gbest(0) = arg\ min[f(X_i(0))]$
– Initialize velocity of each particle: $v_{ij} \sim U(-|ub_j - lb_j|, |ub_j - lb_j|)$, where lb_j and ub_j are the lower and upper bounds of the j^{th} dimension, respectively
$t = 1$
while $t \leq Max\_Iterations$ **do**
  for each particle i **do**
    Update v_i using Eq. (10)
    Update X_i using Eq. (11)
    Compute the fitness value for each particle i
    Update $Pbest_i$ using Eq. (14)
  end for
  Update *Gbest* using Eq. (15)
  $t = t + 1$
end while
return *Gbest*

on PSO [40] described a comprehensive investigation of PSO, modifications, hybridization, extensions, and its applications.

4.3 The Hybridization of ABC and PSO

From the algorithms mentioned above, it is clear that the global best solution of the population does not need to be directly used in ABC algorithm to find new food source positions [21]. Meanwhile, it can be concluded that when the particles in the PSO get stuck in the local minima, it may not escape from the local minima by using random search as scout bees in ABC. We can also find that the reason for ABC being so good is that its update equation only updates a single variable in an individual with D variables [11] instead of all variables as PSO. To overcome the disadvantages and take the advantages of two algorithms, we propose a hybrid global optimization approach by combing the ABC algorithm with the PSO searching mechanism, called ABC-PSO.

In this algorithm, three phases of ABC are used, and we use velocity and the way of finding new food source positions of PSO for the employed bee phase. After updating the position of new food source, the current best position that the individual has visited is updated using Eq. (14). If the current best position is changed, the trial counter of the food source is reset; otherwise, its value is increased by 1. In the onlooker bee phase, for each selected employed bee, the onlooker bee will memorize its position and find a new food source position based on information of the best food source position that the employed bee has visited. This is done by using Eq. (16). The new food source position replaces the current best food source one, and the trial counter of the food source is reset if it has better value; otherwise, the trial counter value is increased by 1. The scout bee phase is the same as ABC. The details the of ABC-PSO algorithm are shown in Algorithm 3 and Figure 1.

$$x_{ij} = \begin{cases} Pbest_{ij}, & \text{if } j \neq m \\ Pbest_{im} + \phi_{im} \cdot (Pbest_{im} - Pbest_{km}), & \text{if } j = m \end{cases} \tag{16}$$

where x_{ij} is the j^{th} dimension of the i^{th} employed bee selected, m is a random index in the range of $[1, D]$, k is the index of a randomly chosen individual ($k \neq i$), and ϕ_{im} is a random number in the range of $[-1, 1]$.

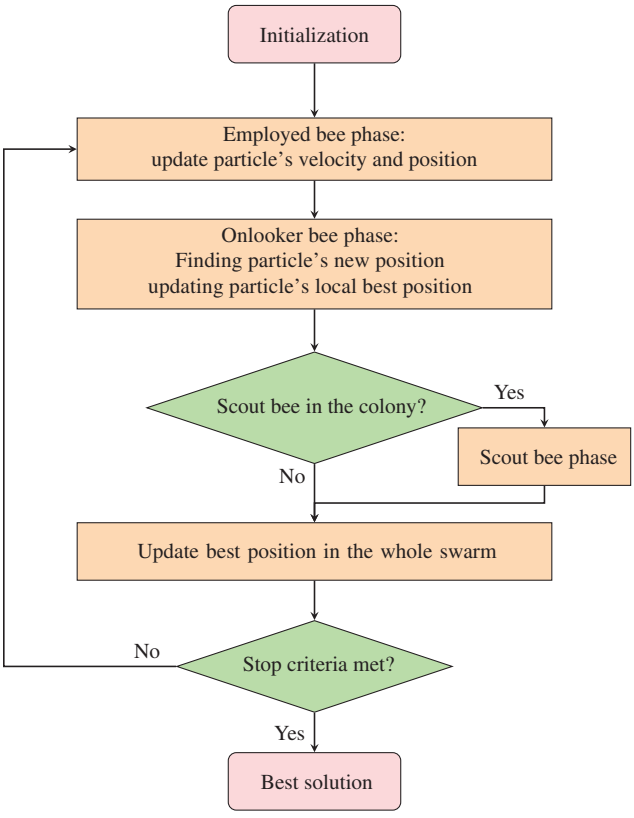

Figure 1: Flowchart of the Proposed Algorithm.

5 Applying Swarm Optimization Algorithms for the Software Effort Estimation Problem

5.1 The Proposed Agile Software Effort Estimation Problem

Based on the final velocity and story point mentioned in Section 2, we propose a formula to predict the effort of agile software projects as follows:

$$Eff = \frac{A \cdot S_P}{B \cdot V} + C \cdot ln(S_P) + D \cdot ln(V) + E, \tag{17}$$

where Eff is the predicted effort measured in day unit, S_P is the value of story point for the project computed as in Section 2, and V is the final velocity of the team.

Our purpose is to determine the values of A, B, C, D, and E so that the predicted values for the project are most accurate compared to actual values. In this study, we use swarm optimization algorithms to seek the values of parameters using historical data of previous projects. When we have a predictive model with appropriate parameters, it can be used to predict the effort of a new project. Figure 2 describes the flowchart of the proposed process.

5.2 Measuring Estimation Quality

The approaches that are widely used to evaluate the quality of software effort estimation models consist of

– The MMRE [25];

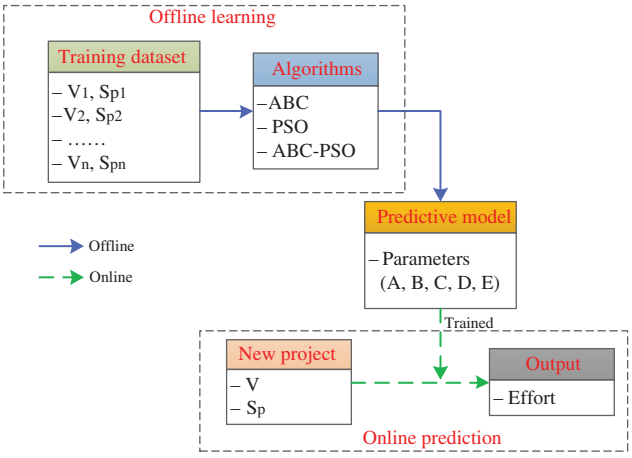

Figure 2: Flowchart of the Proposed Process.

Algorithm 3: The Pseudo Code of the ABC-PSO Algorithm.

Input:
- The maximum cycle number: *MCN*
- The number of employed bees: N_E
- The number of trials for abandoning food source: *limit*
- The dimensionality: *D*
- Coefficients *w*, c_1, c_2

Output: Ensure the best individual in the population: $\vec{X}_{best} = \{x_1, x_2, \ldots, x_D\}$.

Initialize a population of solutions $X_i = \{x_{ij}\}$, $i = 1, \ldots, N_E$, $j = 1, \ldots, D$ using Eq. (6) and their velocities
Compute fitness value for each X_i using Eq. (7)
cycle = 1
while *cycle* ≤ *MCN* **do**
  for $i = 1$ to N_E **do**
    – Update the velocity of individual *i* and its positions using Eqs. (10) and (11), respectively
    – Compute fitness value for each X_i using Eq. (7)
    – Determine the current best position of individual *i* using Eq. (14), and update the trial counter
  end for
  for $i = 1$ to N_E **do**
    – Compute the probability value p_i for the solution X_i using Eq. (9)
  end for
  Formulate the set of potential solutions *Sol* using the roulette-wheel selection mechanism to select N_E solutions in the population based on the probability value p_i
  for each solution X_i in *Sol* **do**
    – Finding a new food source position using Eq. (16)
    – Compute fitness value for each X_i using Eq. (7)
    – Determine the current best position of individual *i* using Eq. (14), and update trial counter
  end for
  for $i = 1$ to N_E **do**
    If value *limit* of solution X_i is reached, **then**
      Produce a random solution and replace X_i with this solution
      break;
    end If
  end for
  Update the *Gbest* of the whole population using Eq. (15)
  cycle = *cycle* + 1
end while

- The median magnitude of relative error (MdMRE) [37];
- The prediction at level N (PRED(N)) [8].

The MMRE is probably the most widely employed evaluation criterion for appraising the performance of software prediction models [6]. The MMRE is defined in Eq. (18).

$$\text{MMRE} = \frac{1}{T}\sum_{i=1}^{T}\text{MRE}_i, \tag{18}$$

where T is the number of observations, i expresses each observation for which effort is predicted, and MRE is the magnitude of relative error, which is computed as

$$\text{MRE}_i = \frac{|A_i - P_i|}{A_i}, \tag{19}$$

where A_i and P_i are actual and predicted effort values of the i^{th} test data, respectively.

Conte et al. [8] indicated that MMRE ≤0.25 is acceptable for effort estimation models. Given two data sets A and B, suppose that data set A includes small projects whereas B contains large projects. Given everything else is equal, and MMRE(B) is smaller than MMRE(A), as a result, a prediction model assessed on data set B will be considered as better than a competing model evaluated on data set A.

Unlike the mean value, the median always shows the middle value m, given a distribution of values, and assures that there is the same number of values above m as below m. Therefore, the median of MRE values for the number of observations called the MdMRE is an alternative to evaluate the performance of software prediction models. Similar to MMRE, the value of MdMRE ≤0.25 is acceptable for effort estimation models.

Another method that is commonly used is the prediction at level N known as PRED(N). It is the percentage of projects for which the predicted values fall within N% of their actual values. For instance, if PRED(25) = 85, this indicates that 85% of the projects fall within 25% error ranges. Conte et al. [8] claimed that N should be set at 25%, and a good estimation system should offer this accuracy level to 75% of the effort. Equation (20) illustrates the way to compute the value of PRED(N):

$$\text{PRED}(N) = \frac{100}{T} * \sum_{i=1}^{T} \begin{cases} 1, & \text{if MRE}_i \le \dfrac{N}{100}. \\ 0, & \text{otherwise} \end{cases} \tag{20}$$

The squared correlation coefficient (R^2), also known as the coefficient of determination, is computed in Eq. (21). The higher the values of R^2, the better the values of estimated results are.

$$R^2 = 1 - \frac{\sum_{i=1}^{T}(A_i - P_i)^2}{\sum_{i=1}^{T}(A_i - A_{\text{mean}})^2}, \tag{21}$$

where A_{mean} is the mean of actual effort values.

Although MMRE and MRE were frequently used for assessing the effort estimation accuracy, Shepperd and MacDonell [32] criticized that the use of these criteria is biased. For instance, we have two projects where the first project is an overestimate and the second project is an underestimate. The actual and estimated values of the effort of project 1 are 20 and 100, respectively. Project 2 has the actual effort value being 100, and the estimated value is 20. Both estimates have identical absolute residual with 80; however, the MMRE values differ by an order of magnitude. Consequently, MMRE will be biased toward prediction systems that underestimate [32]. Therefore, Shepperd and MacDonell proposed a novel measure called mean absolute residual (MAR), and it is shown in Eq. (22):

$$\text{MAR} = \frac{\sum_{i=1}^{T} |A_i - P_i|}{T}. \tag{22}$$

This paper uses all measures above to assess the accuracy of the various software effort estimation models presented.

5.3 Representation of Individuals and Fitness Function

In this study, each individual used in the swarm optimization algorithms is represented as follows:

$$X_i = \{A_i, B_i, C_i, D_i, E_i\}.$$

After algorithms finish, they will return the best individual that contains parameters such that the predicted effort using Eq. (17) is closest to the actual effort.

To assess the fitness of each individual in swarm optimization algorithms, we utilize the fitness function in Eq. (23):

$$f(X_i) = \text{MMRE}(X_i) + \text{MAR}(X_i), \tag{23}$$

where MMRE(X_i) and MAR(X_i) are the MMRE and MAR of individual X_i on T projects in the training dataset, respectively. In this study, we use 21 agile software projects in Zia et al.'s work [41] to assess the effectiveness of our proposed approach, and we also compare it with the regression method as introduced by Zia et al.

6 Experimentation

6.1 Experimental Setup

To evaluate the efficiency among ABC, PSO, and ABC-PSO, we establish the same population size (note that the population size of the ABC and ABC-PSO algorithms is double of the number of employed bees) and the number of iterations.

The setting parameters of ABC are as follows:

- The number of employed bees: $N_E = 50$
- The number of cycles: $MCN = 1000$
- The number of trials for abandoning food source: $limit = 50$

The parameters of the ABC-PSO are set as follows:

- The number of employed bees: $N_E = 50$
- The number of cycles: $MCN = 1000$
- The number of trials for abandoning food source: $limit = 50$
- $w = 0.25$, $c_1 = c_2 = 2$

The settings of PSO are presented below:

- The number of iterations: $Max\_Iterations = 1000$
- Population size: $N_P = 100$
- $w = 0.25$, $c_1 = c_2 = 2$

The range of parameters of the proposed estimation model is shown in Table 4.

Table 4: Range of Parameters of the Proposed Model.

Parameter	Minimum value	Maximum value
A	−20	20
B	−20	20
C	−20	20
D	−20	20
E	−20	20

6.2 Experimental Results

The results reported in this paper are the best results of each algorithm in 25 independent runs. The parameters of the proposed model using each algorithm are presented as below:

- Using ABC-PSO: $A = 16.274$; $B = 16.432$; $C = 3.866$; $D = -0.128$; $E = -15.691$.
- Using ABC: $A = -13.026$; $B = -12.185$; $C = -0.160$; $D = 4.087$; $E = -4.377$.
- Using PSO: $A = 11.086$; $B = 10.185$; $C = -0.051$; $D = 13.359$; $E = -16.284$.

The story points, velocities, actual effort, and predicted effort using swam optimization algorithms and Zia et al.'s method on 21 agile projects are shown in Table 5.

We assess the accuracy of the predicted effort using swarm optimization algorithms based on criteria MMRE, MdMRE, PRED(8), R^2, and MAR. Table 6 shows the obtained results of the approaches.

It is seen that three algorithms gave the better results than Zia et al.'s regression method in terms of almost all evaluation criteria. This indicates the efficiency of swarm optimization algorithms compared with the simple regression. Among three swarm optimization algorithms, the improved version ABC-PSO showed the best results, and it outperformed two original algorithms (ABC and PSO) with regard to all criteria, while PSO showed the worst results. The hybrid version significantly enhanced the effectiveness of PSO with optimization parameters for the software effort estimation model.

Table 5: Experimental Results.

Story point	Velocity	Actual effort	Predicted effort			
			Using ABC-PSO	Using ABC	Using PSO	Zia et al.'s work
156	2.7	63	60.9	60.6	59.6	58
202	2.5	92	84.7	84.9	83.6	81
173	3.3	56	56	55.7	56.5	52
331	3.8	86	92.8	93.3	96.1	87
124	4.2	32	32	32.3	34.8	29
339	3.6	91	99.9	100.6	103	95
97	3.4	35	30.1	30.4	30.9	29
257	3	93	90.5	90.8	91.4	84
84	2.4	36	36	35.9	33.3	35
211	3.2	62	70.2	70	70.8	66
131	3.2	45	43.6	43.4	43.6	41
112	2.9	37	40.7	40.5	39.7	39
101	2.9	32	36.5	36.5	35.6	35
74	2.9	30	26.1	26.6	25.5	26
62	2.9	21	21.3	22.2	21	22
289	2.8	112	108.3	109.3	109.5	103
113	2.8	39	42.4	42.2	41.2	40
141	2.8	52	53.2	52.9	52	50
213	2.8	80	80.2	80.3	80	76
137	2.7	56	53.5	53.1	52	51
91	2.7	35	35	35	33.4	34

Table 6: Results for Algorithms Based on the Criteria MMRE, MdMRE, PRED(8), R^2, and MAR.

Algorithm	MMRE (%)	MdMRE (%)	PRED (8) (%)	R^2	MAR
ABC-PSO	**5.69**	**3.33**	**66.67**	**0.9734**	**3.12**
ABC	5.84	5.18	61.9	0.9732	3.15
PSO	6.69	7.14	61.9	0.9626	3.66
Zia et al.'s regression	7.19	7.14	57.14	0.9638	4

Bold text indicates that the results are statistically significant.

To enrich the study of the ABC-PSO algorithm, we carried out statistical tests to see whether the predicted results using ABC-PSO are statistically different from those using other algorithms in datasets. We used a normality test on the results obtained and identified that data were not normally distributed. For this reason, we employed the Wilcoxon test, which is a non-parametric test; this type of test should be used when the distribution is not normal. Table 7 gives the results of the Wilcoxon test based on the 95% confidence interval (CI). Bold text indicates that the results are statistically different at 95% CI. If the p-value is ≤0.05, then we conclude that the results using ABC-PSO are statistically different from the others at 95% CI. Otherwise, results using ABC-PSO are not statistically different at 95% CI. Based on Table 7, we can see that the ABC-PSO algorithm gave statistically significant results different from Zia et al.'s regression method; however, it fails to be statistically different from the two remaining algorithms.

In Ref. [27], Panda et al. used different types of neural networks such as GRNN, PNN, GMDH polynomial neural network, and cascade-correlation neural network to predict the effort based on 21 software projects in Zia et al.'s work. The performance of the ABC-PSO algorithm is compared with these studies. Table 8 presents the comparison of obtained results using different types of ANN and the PSO-ABC algorithm.

It is clear that our ABC-PSO algorithm outperformed all different kinds of ANNs in terms of criteria R^2 and MMRE. The experimental results indicated that our proposed method significantly ameliorated the accuracy of results obtained in comparison with other methods.

7 Threats to Validity

This paper introduces a method to improve the accuracy of predictions of effort for the agile software projects using the estimation model of Zia et al. [41].

Table 7: Wilcoxon Test for ABC-PSO Algorithm.

ABC-PSO vs.	p-Value at 95% CI
ABC	0.28462
PSO	0.5892
Zia et al.'s regression	**8E–05**

Bold text indicates that the results are statistically different at 95% CI.

Table 8: Comparison Results Using ABC-PSO with Different Kinds of Artificial Neural Networks.

Method	R^2	MMRE (%)
ABC-PSO	**0.9734**	**5.69**
GRNN	0.7125	35.81
PNN	0.6614	157.76
GMDH polynomial neural network	0.6259	15.63
Cascade-correlation neural network	0.9303	14.86

Bold text indicates that the results are statistically significant.

Threats to construct validity are related to the way that the effort estimation models are defined. In this paper, the proposed model assumes that the value of initial project velocity is given by taking from the past projects developed by the same team in similar working conditions. However, when a team is new, the company will not have any past team velocity record. In that case, no obvious assignment to initial project velocity can be allocated. To deal with this problem, we are able to take advantage of the average velocity values of all the teams working in similar conditions with the same size of the project and assign them to the initial project velocity, and then these facts are used to train the classifier by applying swarm intelligence algorithms. Another limitation of the proposed approach is that the complexity of each story is assessed in range from 1 to 5, and it might not cover all diversified characteristics of agile projects in practice. Therefore, further studies need to clarify methods in order to evaluate the complexity of each user story. The focus of our research is to apply swarm intelligence algorithms to create a new formula for software effort estimation based on Zia et al.'s model; thus, we only use two metrics for each agile software project, which are team velocity and story points. In general, there are a wide number of other metrics for an actual project using agile methodologies such as the amount of new logical lines of code the team produced in a release, code integrations, and functional or non-functional requirements [1]. Hence, we intend to generate another estimation model for agile software projects by combining various features into a model.

In our study, threats to external validity insist on the generalization of other types of dataset. In this work, records of 21 projects developed by six software houses from the work of Zia et al. [41] are used without information with regard to the kind of projects taken for research. To increase the persuasiveness, data covering all categories of software developed by agile methods should be collected and experimented for studies in the future. The proposed formula and algorithms in this paper should be compared and evaluated through a real case study as well. This problem is not solved in this work due to the difficulty in collecting the data from industrial software projects. This issue will be resolved in our future research.

8 Conclusion and Future Work

This paper proposed a novel formula based on velocity and story points to estimate the effort for agile software projects. The parameters of the estimation model were then optimized by using swarm optimization algorithms. We also introduced a hybrid version of the ABC and PSO algorithms. The experimental results proved that the ABC-PSO algorithm outperformed ABC and PSO on all evaluation criteria. This new algorithm also gave better results compared with different kinds of ANNs in other studies.

In future work, we intend to apply the proposed algorithm for industrial projects and assess it comprehensively. We are going to apply machine-learning techniques such as support vector machine, random forest, and stochastic gradient boosting for the effort estimation problem of agile software projects.

Bibliography

[1] P. Abrahamsson and J. Koskela, Extreme programming: a survey of empirical data from a controlled case study, in: *Proceedings of International Symposium on Empirical Software Engineering*, pp. 73–82, 2004.

[2] S. Andreas, K. Martin and D. Reiner, Effort estimation for Agile software development projects, in: *Proceedings of 5th Software Measurement European Forum*, pp. 113–126, 2008.

[3] F. V. D. Bergh and A. P. Engelbrecht, A cooperative approach to particle swarm optimization, *IEEE Trans. Evolut. Comput.* **8** (2004), 225–239.

[4] E. Bijami, M. Shahriari-kahkeshi and H. Zamzam, Simultaneous coordinated tuning of power system stabilizers using artificial bee colony algorithm, in: *Proceedings of 26th International Power System Conference*, pp. 1–8, 2011.

[5] B. Boehm, *Software Engineering Economics*, Prentice Hall, Englewood Cliffs, 1981.

[6] L. Briand and I. Wieczorek, *Resource Modeling in Software Engineering*, *Encyclopedia of Software Engineering*, Wiley, Chichester, 2002.

[7] E. Coelho and A. Basu, Effort estimation in agile software development using story points, *Int. J. Appl. Inform. Syst.* **3** (2012), 7–10.

[8] S. D. Conte, H. E. Dunsmore and V. Y. Shen, *Software Engineering Metrics and Models*, Benjamin-Cummings Publishing Co., Inc., Menlo Park, CA, 1986.
[9] R. C. Eberhart and J. Kennedy, A new optimizer using particle swarm theory, in: *Proceedings of the Sixth International Symposium on Micromachine and Human Science*, Nagoya, Japan, pp. 39–43, 1995.
[10] D. Eck, B. Brundick, T. Fettig, J. Dechoretz and J. Ugljesa, *Parametric Estimating Handbook*, 4th edition, The International Society of Parametric Analysis (ISPA), Vienna, VA, 2009.
[11] M. El-Abd, A hybrid ABC-SPSO algorithm for continuous function optimization, in: *Proceedings of IEEE Symposium on Swarm Intelligence (SIS)*, 2011.
[12] A. E. D. Hamouda, Using agile story points as an estimation technique in CMMI organizations, in: *Agile Conference (AGILE)*, pp. 16–23, 2014.
[13] T. J. Hsieh and W. C. Yeh, Knowledge discovery employing grid scheme least squares support vector machines based on orthogonal design bee colony algorithm, *IEEE Trans. Syst. Manage. Cybern. Part B Cybern.* **41** (2011), 1198–1212.
[14] I. Hussain, L. Kosseim and O. Ormandjieva, Approximation of cosmic functional size to support early effort estimation in agile, *Data Knowl. Eng.* **85** (2013), 2–14.
[15] M. Jorgensen, Forecasting of software development work effort: evidence on expert judgment and formal models, *Int. J. Forecast* **23** (2007), 449–462.
[16] D. Karaboga and B. Basturk, A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm, *J. Global Optim.* **39** (2007), 459–471.
[17] D. Karaboga and C. Ozturk, A novel clustering approach: artificial bee colony algorithm, *Appl. Soft Comput.* **11** (2011), 652–657.
[18] D. Karaboga, B. Gorkemli, C. Ozturk and N. Karaboga, A comprehensive survey: artificial bee colony (ABC) algorithm and applications, *Artif. Intell. Rev.* **42** (2014), 21–57.
[19] S. Keaveney and K. Conboy, Cost estimation in agile development projects, in: *Proceedings of the Fourteenth European Conference on Information Systems*, pp. 183–197, 2006.
[20] T. T. Khuat and M. H. Le, An effort estimation approach for agile software development using fireworks algorithm optimized neural network, *Int. J. Comput. Sci. Inform. Secur.* **14** (2016), 122–130.
[21] M. S. Kiran and M. Gunduz, A recombination-based hybridization of particle swarm optimization and artificial bee colony algorithm for continuous optimization problems, *Appl. Soft Comput.* **13** (2013), 2188–2203.
[22] C. Lopez-Martin, A fuzzy logic model for predicting the development effort of short scale programs based upon two independent variables, *Appl. Soft Comput.* **11** (2011), 724–732.
[23] J. Lynch, *Chaos Manifesto*, The Standish Group, Boston [Online] http://www.standishgroup.com/newsroom/chaos_2009.php. Accessed 5 November, 2016.
[24] 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.* **86** (2013), 144–160.
[25] S. Olatunji and A. Selamat, Type-2 fuzzy logic based prediction model of object oriented software maintainability, *Intell. Softw. Methodol. Tools Techn.* **513** (2015), 329–342.
[26] A. L. Oliveira, Estimation of software project effort with support vector regression, *Neurocomputing* **69** (2006), 1749–1753.
[27] A. Panda, S. M. Satapathy and S. K. Rath, Empirical validation of neural network models for agile software effort estimation based on story points, *Proc. Comput. Sci.* **25** (2015), 772–781.
[28] R. Popli and N. Chauhan, Cost and effort estimation in agile software development, in: *IEEE International Conference on Optimization, Reliability, and Information Technology (ICROIT)*, pp. 57–61, 2014.
[29] L. H. Putnam, A general empirical solution to the macro software sizing and estimating problem, *IEEE Trans. Softw. Eng.* **4** (1978), 345–361.
[30] R. S. Rao, S. V. L Narasimham and M. Ramalingaraju, Optimization of distribution network configuration for loss reduction using artificial bee colony algorithm, *Int. J. Elect. Power Energy Syst. Eng.* **1** (2008), 116–122.
[31] S. M. Satapathy, A. Panda and S. K. Rath, Story point approach based agile software effort estimation using various SVR kernel methods, in: *International Conference on Software Engineering and Knowledge Engineering*, pp. 304–307, 2014.
[32] M. Shepperd and S. MacDonell, Evaluating prediction systems in software project estimation, *Inform. Softw. Technol.* **54** (2012), 820–827.
[33] Y. Shi and R. Eberhart, A modified particle swarm optimizer, in: *Proceedings of International Conference on IEEE World Congress on Computational Intelligence*, pp. 69–73, 1998.
[34] X. Shi, Y. Li, H. Li, R. Guan, L. Wang and Y. Liang, An integrated algorithm based on artificial bee colony and particle swarm optimization, in: *Proceedings of the Sixth International Conference on Natural Computation*, 2010.
[35] R. Silhavy, P. Silhavy and Z. Prokopova, Applied least square regression in use case estimation precision tuning, in: *Proceedings of the 4th Computer Science On-line Conference*, 2015, pp. 11–17.
[36] N. Suguna and K. G. Thanushkodi, An independent rough set approach hybrid with artificial bee colony algorithm for dimensionality reduction, *Am. J. Appl. Sci.* **8** (2011), 261–266.
[37] F. Valdes and A. Abran, Comparing the estimation performance of the EPCU model with the expert judgment estimation approach using data from industry, *Softw. Eng. Res. Manage. Appl.* **296** (2010), 227–240.
[38] S. Wang, Y. Zhang, Z. Dong, S. Du, G. Ji, J. Yan, J. Yang, Q. Wang, C. Feng and P. Phillips, Feed-forward neural network optimized by hybridization of PSO and ABC for abnormal brain detection, *Int. J. Imaging Syst. Technol.* **25** (2015), 153–164.

[39] S. Wang, Y. Zhang, G. Ji, J. Yang, J. Wu and L. Wei, Fruit classification by wavelet-entropy and feedforward neural network trained by fitness-scaled chaotic ABC and biogeography-based optimization, *Entropy* **17** (2015), 5711–5728.
[40] Y. Zhang, S. Wang and G. Ji, A comprehensive survey on particle swarm optimization algorithm and its applications, *Math. Probl. Eng.* **2015** (2015), Article ID 931256.
[41] Z. K. Zia, S. K. Tipu and S. K. Zia, An effort estimation model for agile software development, *Adv. Comput. Sci. Appl.* **2** (2012), 314–324.