IJCI v10-01(2023) 18-35
International Journal of Computers and Information (IJCI)
Available online at *https://ijci.journals.ekb.eg/*

Assessment Reliability for Open-Source Software using Probabilistic Models and Marine Predators Algorithm

Islam S.Ramadan[a], Hany M. Harb[a], Hamdy M. Mousa[b], Mohamed G. Malhat[b]

[a] *Computer Science Department, Faculty of Information Technology, MUST University, 6th of October city 77, Egypt*
[b] *Computer Science Department, Faculty of Computers and Information, Menoufia University, Shebin Elkom 32511, Egypt*
Islam.saied@must.edu.eg, hany.harb@must.edu.eg, hamdimmm@hotmail.com, m.gmalhat@yahoo.com

Abstract

With the advancement of technology today, computer software has a high effect in different areas of our life. In particular open-source software is used by many software companies because it helps them to create their new software without implementing it from scratch. Therefore, the quality of open-source software is a significant issue and one of the most popular research topics in the literature. Checking software reliability during the development cycle is an indicator of releasing the software or not. In this paper, we propose to use Marine Predators Algorithm with six probabilistic models for assessing reliability more accurately. We select three versions of a standard dataset for GNU's Not Unix Network Object Model Environment projects. We compare the implemented models by three evaluation criteria: mean square error, sum of square error, and reliability. The results for all versions of the dataset show that SRGM-5 is based on the methodology of imperfect debugging estimates the most accurate reliability results in terms of mean square error and sum of square error.

Keywords: Software reliability; Probabilistic models; Marine Predators Algorithm; Open-source software;

1. Introduction

Software is a group of instructions that ask the computer to execute specific tasks [1].Today computer software has been used in all areas of our life like education, marketing, medicine, and military [2]. For example, the medical diagnosis system in the medical field discovers the disease according to laboratory data and the patients' symptoms [3]. Computer software is divided into two categories:

First, open-source software (OSS) is a computer program its implementation code is accessible with no charge. The copyright of OSS allows the users to distribute, use, and modify the source code easily [4]. For example:- WordPress, Drupal, Ubuntu, Open Office, and Firefox [5].

Second, closed source software (CSS) is a computer program that will be available only when you buy it from the publisher. You can use it only and are not allowed to make redistribution or modifications [6].

There are a lot of advantages for OSS Over CSS, such that:

First, freedom to use: It's available for free, and this gives software companies a chance to modify OSS source code to meet their needs [7].

Second, self-Reviewing: The availability of OSS code allows anyone to review the code and not be limited to paid security analysts [8].

Third, number of Reviewers: If the largest software companies develop CSS, it will be reviewed by a limited number of reviewers. On the other side, OSS has a lot of contributions that can access and review the code easily [8].

Fourth, software Cost: OSS doesn't need any licensing fees [9].

According to the advantages of OSS listed above, many applications used today in different fields depend on OSS. Actually, between 80% and 90% of current software use OSS such that software developers make modifications for OSS code to develop new software without building it from scratch [10]. The performance and quality of OSS is a significant area in the literatures [11-14] . Different approaches and methods have been proposed to evaluate and assess OSS [15]. We can do that by assessing the reliability of the software [15]. Assessment software reliability is a critical metric because of a lot of reasons, such as, as mentioned in [16]: First, it determines the approval or the failure of the software. Second, reliability assessment helps us to deliver the perfect software to the customer. So there exist different models to evaluate the reliability of OSS, such that these models divided into two types [17]:

First, deterministic models: examine the code of the software without adding any random values or events, such as McCabe's Cyclomatic Complexity and Halstead Model [17].

Second, probabilistic models: define the cases of failures and the error correction as random events, such as Input Domain, Error Seeding, Failure Rate, and Curve Fitting [17].

The majority of the previous works use probabilistic models in assessing reliability because these models depend on error correction, cases of failures, and evaluate the number of cumulative faults noticed in a specific time [18]. The prediction ability of probabilistic models mainly depends on the values of their parameters so if the models' parameters are estimated accurately, the reliability would also be accurate [18]. The traditional estimation techniques like least square estimation (LSE) [19] and maximum likelihood estimation (MLE) [19] are not the ideal solutions [18], and both of them are used by the literatures as in [20- 24].

In this paper, we need to enhance the assessment of the reliability of open-source software. To achieve this goal, we select six software reliability growth models (SRGMs) proposed in the previous related works [20,24] because SRGMs are used to determine software reliability [18]. With the selected SRGMs, we use the Marine Predators Algorithm (MPA), which is a new nature-inspired metaheuristic algorithm [25], as an estimation technique for the SRGMs' parameters to estimate these parameters accurately because of a lot of reasons as follows: First, the software failures are stochastic naturally, and we need to use an estimation technique that maps this stochastic behaviour [18]. Second, as mentioned above, both LSE and MLE are traditional estimation techniques, and they are not the ideal solution Third, for software reliability prediction, parameter estimation has the highest priority [26]. Fourth, any SRGM's ability for prediction depends on the values of its parameters, so SRGMs need the best technique for parameter estimation [27]. Finally, after MPA uses each SRGM's mean value function as an objective function to estimate its parameters, we make substitutions with these parameters in equation (29). We perform the previously explained steps for three different versions of OSS, which is GNU's Not Unix Network Object Model Environment (GNOME) projects that is available at [28]. We compare the selected models by various evaluation criteria to assess the performance of the chosen models and decide the best model for assessment reliability to answer the essential issue of which model is optimal for assessment reliability. The empirical results indicated that SRGM-5 is the most accurate model for assessment reliability for all versions of GNOME projects. It is based on the methodology of imperfect debugging [20]. Furthermore, SRGM-6 is the least efficient model for assessment reliability for all versions of GNOME projects. It is based on the methodology of Gompertz distribution [24].

The remaining parts of the paper are organized into sections, such that Section II covers the related work in reliability assessment for OSS. Section III represents the proposed work. Section IV illustrates an empirical study for the selected models with various criteria over three versions of datasets. Finally, section V represents the conclusion of the paper.

2. Related Work

Recently, many research papers focused on estimating the reliability of OSS, and this is due to the importance of OSS. Software companies implement their new software depending on OSS significantly. One research paper written by Gandhi et al. in [20] proposed five OSS reliability models based on two methodologies: First, perfect debugging methodology and second, imperfect debugging methodology. Use dataset for GNOME that is available at in [28] to test the proposed models. Estimate the values of the model's parameters using least square estimation. The experimental results show that the models based on imperfect debugging methodology have more fitting results than other models.

Zhu et al. in [21] proposed a model for assessing OSS reliability without neglecting the relationship between faults in the previous and recent releases. Use Juddi dataset that is available at [29] and Apache dataset that is available at [30] to estimate reliability for both of them. The proposed model estimates reliability more accurately than other models utilized in the comparison.

Diwakar and Aggarwal [22] suggest the faults of the current software version are divided into two parts: First, the previous version's faults will still exist in the current version. Second, the new faults caused by any updates of the code. Use Apache dataset that is available at [30] , and estimate reliability by the proposed model. The proposed model estimates reliability more accurately than other models utilized in the comparison.

Wang in [23] proposes a model with fault introduction established on the distribution of generalized Pareto. Use dataset for three different Apache projects: Avro, Beam, and GORA, that are available at [30] to test the proposed model. Estimate the values of the model's parameters using LSE. The results clarify the high efficiency of the proposed model, and it's fitting results are better than other models utilized in the comparison.

Yaghoobi in [24] proposes a model to estimate reliability for a tandem software dataset that is published in [30] and compares this model with other models to select the optimal one for assessment reliability. The results show that the proposed model is the best one.

All related works assess the reliability by a list of SRGMs and estimate the parameters of these SRGMs using LSE. In our proposed work, we assess the reliability of OSS by using the proposed SRGMs that already exist in the related works and MPA to estimate the parameters of the implemented SRGMs. After that, we substitute with the estimated parameters in equation (29). Consequently, the estimation technique for SRGMs' parameters is the difference between the related works and our proposed work.

3. The proposed Work

We assess the reliability of OSS using six probabilistic models and Marine Predators Algorithm by using SRGM's mean value function as an objective function for the MPA estimation technique. MPA estimates SRGM's parameters using a specific version of the OSS dataset. After that, we make a substitution with these parameters in the reliability equation (29). The reliability assessment of OSS consists of two stages. In the first stage, we run the MPA code 30 times and take the average of all estimated parameters and the estimated evaluation metrics to be confident of the result's accuracy because MPA estimates different values every time we run its code. We move on to the second stage after computing the SRGM's parameters, which will be an input to make a substitution directly in the reliability equation (29) to know the reliability of the used version of the OSS dataset based on the selected SRGM. We can describe the framework for our proposed work as illustrated in Fig.1.

As mentioned above we assess the reliability of OSS using six probabilistic models and Marine Predators Algorithm both of them will be explained in detail as follows:

3.1. Probabilistic Models

We select six models from the probabilistic category [20,24]. These models are based on three methodologies: perfect debugging methodology, imperfect debugging methodology, and Gompertz distribution methodology [20,24]. Both perfect and imperfect debugging have different cases, such that each case has specific fault content function [20].

Fault content function represents the number of users working on the operational phase of OSS and is described by *a(N)*. this function shows the relation between number of users deal with OSS and the modification rate in its code, such that the more number of OSS' users, the more modification rate in its code , and hence the possibility of increasing the fault content also [20].

The methodologies of the selected proposed models from the previous related works are discussed as follows:

First, perfect debugging methodology: This methodology assumes that the process of detecting faults does not produce any additional faults and it includes one case only [20].

Case 1: The fault content's function is described by *α(N)* equals to constant *a*. This case assumes that the process of debugging does not produce any additional faults. SRGM-1 uses this case, and the function of this case as follows [20]:

$$a(N) = a \tag{1}$$

Such that *a* represents The initial number of faults in the software, and *N* refers to the cumulative number of the software's users in the time interval (0,1] [20].

Second, imperfect debugging methodology: This methodology assumes that there is an opportunity of getting more new faults while fixing the existing ones. It has four cases [20].

Case 2: we assumed the faults' number to be a linear function of users' number. SRGM- 2 uses this case, and the function of this case as follows [20]:

$$a(N) = a(1 + \alpha N) \tag{2}$$

Such that both *a* and *N* are referred to as mentioned in case 1 and α refers to the rate of error generation [20].

Case 3: we assumed that an exponential function for fault content, and this means that the introduction of the faults is exponential for users. SRGM-3 uses this case, and the function of this case as follows [20]:

$$a(N) = ae^{\alpha N} \tag{3}$$

Such that both *a, N*, and α are referred to as mentioned in case 2 [20].

Case 4: we assumed that the new fault's introduction rate as a function of the faults' number, which already released from the software. SRGM-4 uses this case, and the function of this case as follows [20]:

$$a(N) = a + \alpha m(N) \tag{4}$$

Such that both *a, N*, and α are referred to as mentioned in case 2, and *m* refers to the number of expected faults removed in time interval (0,t] [20].

Case 5: we assumed that the introduction of new faults is exponentially per noticed faults. SRGM-5 uses this case, and the function of this case as follows [20]:

$$a(N) = c + a(1 - e^{-\alpha N}) \tag{5}$$

Such that both *a, N*, and α are referred to as mentioned in case 2, and *c* is supposed as constant.

Third, gompertz distribution methodology: This methodology is a kind of mathematical model that acts as a growth model. SRGM-6 uses this case, and It has two assumptions as follows [24]:

First, initial faults' number is relied on Poisson distribution with a parameter a. Second, the failures happen at independent random times by the Gompertz distribution with the cumulative function:

$$F(t) = 1 - e^{(1-e^{bt})} \quad (6)$$

Such that b refers to the rate of removing fault.

Every SRGM from the selected models has a mean value function depends on the case of the model as listed above. The mean value functions for each SRGM are as follows [20,24]:

The SRGM-1 has a mean value function estimated by

$$m1(t) = a(1 - e^{-bN}) \quad (7)$$

Such that m1(t) refers to the number of predictable faults that will be removed in time interval (0,1], a refers to the initial faults in a software, and b refers to the rate of removing faults [20].

The SRGM-2 has a mean value function estimated by

$$m2(t) = a(\alpha + (1 - \frac{\alpha}{b})(1 - e^{-bN})) \quad (8)$$

Such that a, b are referred to as mentioned in equation (7), *m2(t)* refers to the number of predictable faults that will be removed in time interval (0,1], α refers to the rate of error generation, N refers to the cumulative number of the software's users in the time interval (0,1] [20].

The SRGM-3 a mean value function estimated by

$$m3(t) = \frac{ab}{\alpha+b}(e^{\alpha N} - e^{-bN}) \quad (9)$$

Such that a, b, N, α are referred to as mentioned in equation (8), *m3(t)* refers to the number of predictable faults that will be removed in time interval (0,1] [20].

The SRGM-4 has a mean value function estimated by

$$m4(t) = \frac{a}{1-\alpha}(1 - e^{-b(1-\alpha)N}) \quad (10)$$

Such that a, b, N, α are referred to as mentioned in equation (8), *m4(t)* refers to the number of predictable faults that will be removed in time interval (0,1] [20].

The SRGM-5 has a mean value function estimated by

$$m5(t) = (a + c)(1 + e^{-bN}) - \frac{ab}{b-\alpha}(e^{-\alpha N} - e^{-bN}) \quad (11)$$

Such that a, b, N, α are referred to as mentioned in equation (8), *m5(t)* refers to the number of predictable faults that will be removed in time interval (0,1], c supposed as constant [20].

The SRGM-6 has a mean value function estimated by

$$m6(t) = a(1 - e^{(1-e^{bt})}) \quad (12)$$

Such that a, b are referred to as mentioned in equation (8), t refers to calendar time [24].

It's known that OSS has a lot of advantages, such as OSS supports propagation innovation because its source code is available between people [20]. OSS are used by two kinds of people, which are First, Innovators: people who use OSS because they are already interested in software. Second, Imitators: people who use OSS after Innovators candidate it to them [20]. The availability of a new product or idea between people over time is known as diffusion [20]. As a result for that, Bass model defines the mathematical representation of diffusion process using the following equation [20]:

$$\frac{dN(t)}{dt} = (p + q\frac{N(t)}{N'})(N' - N(t)) \quad (13)$$

Such that *N(t)* represents cumulative of adopters' number at time *t*, *p(N' – N(t))* refers to numbers of Innovators, therefore *p* refers to the coefficient of innovation, and $q\frac{N(t)}{N'}(N'-N(t))$ refers to numbers of Imitators, therefore *q* refers to the coefficient of imitation. Under the initial condition of *N(0)=0* the solution of equation (13), and the result in which is evaluated as follows:

$$N(t) = N' \frac{1-e^{-(p+q)t}}{1+\frac{q}{p}e^{-(p+q)t}} \tag{14}$$

For each *N* parameter which refers to the cumulative number of the software's users in the time interval (0,1] in equation (7) to equation (11) is equal to *N(t)*, which is represented by equation (14), so every *N = N(t)* that represents the number of users of OSS in specific time *t* [20].

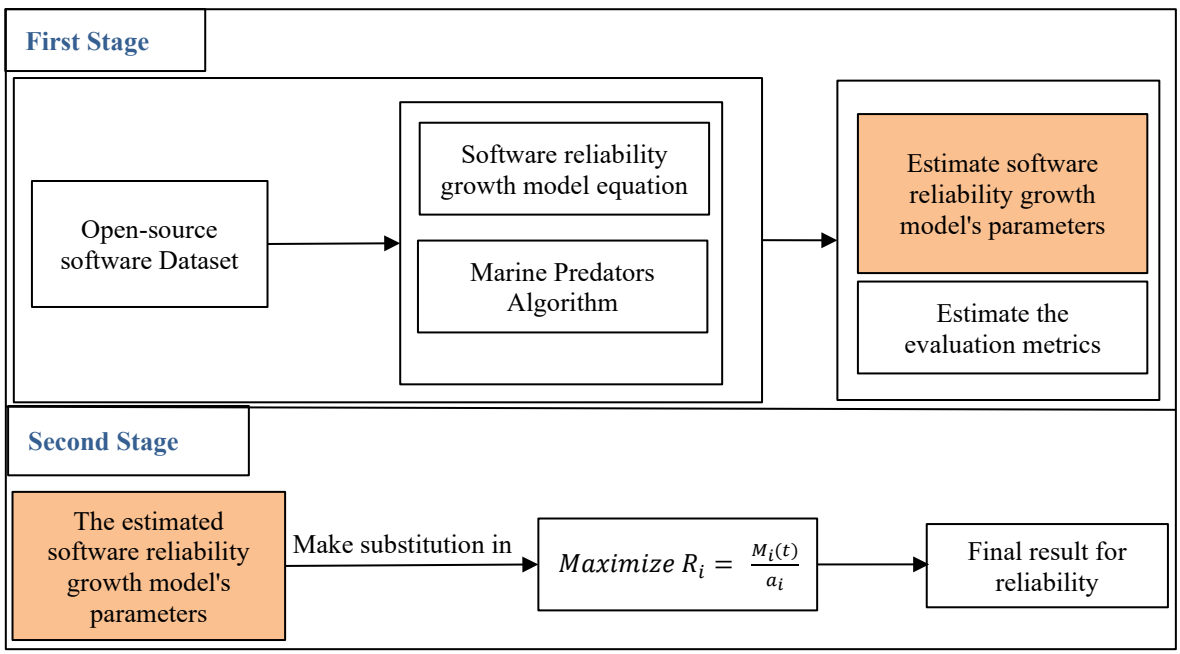

Fig. 1. The framework for our proposed work.

3.2. Marine Predators Algorithm

Marine Predators Algorithm is a new nature-inspired metaheuristic influenced basically by food-finding techniques of ocean predators, such as Brownian and Lévy motions. Because the prey is a predator while it hunts for food, MPA considers both predator and prey as search agents [25].

According to equation (15), a_i is the initial population in each search agent, and it is initialized randomly. The equation's variables have lower and upper bounds represented by a_{min} and a_{max} respectively, r represents a random value from 0 to 1 [25].

$$a_i = a_{min} + r(a_{max} - a_{min}) \tag{15}$$

MPA defines the best predators in a matrix called *E* for Elite and described mathematically in equation (16), and there exists another matrix for preys called *P* described mathematically in equation (17) [25].

$$E = \begin{bmatrix} a^I_{1,1} & a^I_{1,2} & \dots & a^I_{1,d} \\ a^I_{2,1} & a^I_{2,2} & \dots & a^I_{2,d} \\ \dots & \dots & \dots & \dots \\ a^I_{n,1} & a^I_{n,2} & \dots & a^I_{n,d} \end{bmatrix} \tag{16}$$

$$P = \begin{bmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,d} \\ a_{2,1} & a_{2,2} & \dots & a_{2,d} \\ \dots & \dots & \dots & \dots \\ a_{n,1} & a_{n,2} & \dots & a_{n,d} \end{bmatrix} \quad (17)$$

MPA combines two kinds of searching strategies, which are Brownian and Lévy, by dividing the process of optimization into three phases, so it makes a balance between exploitation and exploration [25].

In the first phase, the search agents (Preys) adopt the Brownian for exploration in case exploration is more essential, as illustrated in equation (18) and equation (19). The execution of this phase is done in the first third of iterations[25] .

$$s_i = r_B \odot (E_i - r_B \odot P_i)\ i = 1, \dots . n \quad (18)$$

$$P_i = P_i + d \cdot R \odot s_i \quad (19)$$

Such that s_i is the size of step for the $prey_i$, R is a vector contains uniformly random numbers from 0 to 1 while d = 0.5, r_B is Brownian movement represented as random vector, $\odot$ represents element-wise multiplication [25].

In the second phase, when both of exploitation and exploration have the equal degree of the importance half of search agents (Prey) makes exploitation by using Lévy strategy as illustrated in equation (20), and equation (21) [25].

$$s_i = r_L \odot (E_i - r_L \odot P_i)\ i = 1, \dots . \frac{n}{2} \quad (20)$$

$$P_i = P_i + d \cdot R \odot s_i \quad (21)$$

Such that r_L represents Lévy strategy as random vector. And on the other side the other half (Predators) makes exploration by using Brownian strategy as illustrated in equation (22), and equation (23) [25].

$$s_i = r_B \odot (r_B \odot E_i - P_i)\ i = \frac{n}{2} \dots . . n \quad (22)$$

$$P_i = E_i + d \cdot AP \odot s_i \quad (23)$$

Such that AP is adaptive parameter which equal to $(1 - \frac{iter}{\text{maxIter}})^{(2 * \frac{iter}{maxIter})}$

In the third phase, the search agents (predators) make exploitation by using Lévy when exploitation is more important, as illustrated in equation (24) and equation (25). In that way, the movements from the exploration phase to the exploitation phase. Are done smoothly [25] .

$$s_i = r_L \odot (r_L \odot E_i - P_i)\ i = 1, \dots \dots n \quad (24)$$

$$P_i = E_i + d \cdot AP \odot s_i \quad (25)$$

MPA simulates the impact of Fish Aggregating Devices to prevent becoming stuck in a local optimum (FADs). This effect causes search agents to make large jumps in other dimensions with a specific probability, as illustrated in equation (26).

$$P_i = \begin{cases} P_i + AP[a_{min} + R \odot (a_{max} - a_{min})] \odot B \ \ if \ r \leq FADs \\ P_i + [FADs\ (1 - r) + r\ ](P_{r1} - P_{r2}) \ \ if \ r > FDAs \end{cases} \quad (26)$$

The FADs effect's probability is represented by the FADs parameter that equals .02. B is a binary array that is built from another array R of the same length. R consists of real numbers between 0 to 1. If the equivalent value in R is smaller than 0.2, the element in B equals 1, otherwise it equals 0. The random indexes of the P matrix are represented by $r1$ and $r2$. Only newly created search agents with higher fitness values than their corresponding agents in the current population are added to the new population.

The Elite matrix is updated if the solutions at the end of every iteration are better than the Elite solutions.

According to our empirical, MPA requires four inputs the maximum number of iterations, the mean value function of SRGM as an objective function with dataset of GNOME OSS, the number of parameters in the objective function of SRGM, and lower and upper bounds for each parameter. After executing MPA, it calculates the estimated parameters, and the evaluation metrics. The pseudocode of MPA is presented as shown in Table 1.

Table 1. MPA pseudocode [31].

MPA pseudocode
1. Make initialization for first population using Equation 13. 2. While minimal than the max number of iterations 3. Estimate each search agent. 4. If $iterNum < Max(iter) / 3$ Update search agents using Brownian by Equation 16 and Equation 17. Else If $2 \times Max(iter)/3 > iterNum > Max(iter) / 3$ Half of the search agents are updated using Lévy by Equation 18 and Equation 19 And for the other half is updated using Brownian by Equation 20 and Equation 21. Else If $iterNum > 2 \times Max(iter)/3$ Update the search agents using Lévy by Equation 22 and Equation 23. End If. 5. Search agents use a certain probability to take jumps by Equation 24. 6. In the new population the new agents are added if and only if it is better than its past counterparts. 7. Update Elite matrix. End While.

Although MPA is a recent metaheuristic algorithm, it is used in different research topics in the literatures as in [31-34] because it is an effective heuristic algorithm that has a lot of advantages, such as [35]: First, containing a restricted number of defined variables. Second, uncomplicated procedures. Third, gradient-free nature. Fourth, flexibility.

4. Experimental Results and Discussion

This section contains the dataset used in the experiments, the used software and hardware, assessment criteria, and practical results.

4.1. Datasets

We tested the performance of the chosen models using three versions of GNOME OSS projects as shown in Table 2. Each version of dataset consists of two columns, one represents number of weeks passed from releasing each version of GNOME projects and the second represents number of faults detected in each week [36]. A GUI desktop for Unix systems is the primary objective of GNOME, which has near to two million lines of code [37].

Table 2. The detected faults for official public releases of GNOME projects [36].

GNOME 2.0		GNOME 2.2		GNOME 2.4	
Weeks from release	Detected faults	Weeks from release	Detected faults	Weeks from release	Detected faults
1	6	1	5	1	4
2	5	2	4	2	5
3	3	3	5	3	2
4	2	4	5	4	7
5	5	5	9	5	3
6	5	6	5	6	1
7	8	7	2	7	3
8	4	8	1	8	4
9	8	9	2	9	3
10	3	10	3	10	5
11	2	11	2	11	1
12	1	13	1	12	3
13	6	15	4	15	2
14	8	16	1	18	1
15	6	17	1	19	1
16	2	18	1	20	5
17	2	22	1	21	2
18	1	24	2	23	1
19	1			46	1
20	1				
21	1				
22	2				
24	3				

4.2. Evaluation Criteria

We use three evaluation criteria for the chosen models which are Mean Square Error (MSE), Sum of Square Error (SSE), and Reliability [38,39]. These criteria determine the efficiency of the selected models' fitting results[23].

4.2.1. SSE is evaluated using the equation [38]:

$$SSE = \sum_{j=1}^{k} \left(M_j - M(t_j) \right)^2 \tag{27}$$

Such that M_j denotes the total number of faults that detected at time t_j according to the actual data, and $M(t_j)$ denotes the total number of faults that estimated at time t_j.

4.2.2. MSE is evaluated using the equation [38]:

$$MSE = \frac{\sum_{j=1}^{k} \left(M_j - M(t_j) \right)^2}{k-p} \tag{28}$$

Such that both of M_j , and $M(t_j)$ as mentioned in the equation of SSE, p represents number of the model's parameters and k represents sample size of faults.

4.2.3. Reliability is evaluated for OSS using the equation [39]:

$$Maximize\ R_i = \frac{M_i(t)}{a_i} \tag{29}$$

Such that R_i denotes the reliability for i release, M_j denotes the model's mean value function, and a_i denotes the initial faults in the software.

MPA estimates the mean value functions' parameters for each model. According to this estimation, we can evaluate the predictive efficiency of the models such that, if we compare two models, the perfect model has a small value for SEE and MSE than the other model [23].

4.3. Results and Analysis

4.3.1. The evaluation of the mean value functions' parameters

As mentioned before, the predictive ability of SRGMs mainly depends on their parameters, and according to Equation (29), the parameters' values are needed to estimate the reliability. MPA evaluates these parameters for each model for all datasets as shown in Table 3, Table 4, and Table 5. The shared parameters between all models are the initial faults in the software and removal rate for faults, so these parameters have the most effect on estimation reliability [40].

Table 3. The estimated values of parameters for all SRGMs based on GNOME dataset version 2.0.

Model	a	b	N'	p	q	α	c
SRGM-1	90.3375	0.08	50	0.0105	0.1313	-	-
SRGM-2	83.8828	0.5	38	0.0024	0.1100	0.0015	-
SRGM-3	84.1911	0.9	15	0.00347	0.11300	0.003	-
SRGM-4	83.7277	0.1898	74.999	0.0033	0.11104	0.01614	-
SRGM-5	88.26089	0.75309	61.62822	0.001133	0.10885	0.00535	81.9236
SRGM-6	92.4120	0.0550	-	-	-	-	-

Table 4. The estimated values of parameters for all SRGMs based on GNOME dataset version 2.2.

Model	a	b	N'	p	q	α	c
SRGM-1	92.4276	0.009	95	0.0701	0.1314	-	-
SRGM-2	70.5326	0.07822	18.6016	0.05508	0.1351	0.00051	-
SRGM-3	50	1.7176	63.782	0.00816	0.0754	0.01935	-
SRGM-4	52.0865	0.03078	69.7414	0.0500	0.13513	0.2018	-
SRGM-5	75.85257	14.29899	2.31148	0.00285	0.37015	0.11504	36.0690
SRGM-6	50.7519	0.1019	-	-	-	-	-

Table 5. The estimated values of parameters for all SRGMs based on GNOME dataset version 2.4.

Model	a	b	N'	p	q	α	c
SRGM-1	93.6802	0.009	95	0.0513	0.1	-	-
SRGM-2	69.9365	0.1155	12.4396	0.04348	0.09	0.0039	-
SRGM-3	62.9181	0.27051	6.4774	0.04123	0.09435	0.01886	-
SRGM-4	73.4746	0.0280	46.6601	0.0457	0.08841	0.0539	-
SRGM-5	75	1.51812	7.6531	0.009	0.180416	0.04563	35.3531
SRGM-6	51.2273	00.0762	-	-	-	-	-

4.3.2. The fitting results for three versions of GNOME dataset

There are three evaluation metrics to estimate the predictive efficiency of the implemented models as shown in Table 6, Table 7, and Table 8. Each table has the fitting results for the distinct GNOME dataset version.

According to Table 6, and Fig. 2, which represent the evaluation metrics for SRGMs based on GNOME 2.0, SRGM-5 has the smallest value for SSE, and this means that this model has the most predictive execution than the remaining models because the predicted values by this model are the most in line with the actual values. According to that, SRGM-5 is the best model to assess reliability accurately. This model is based on the methodology of imperfect debugging. This methodology supposes that, in the debugging operation, there is a possibility for at least introducing only one new fault. On the other hand, SRGM-6 has the highest value for SSE, so this model has less predictive execution than the remaining models because the predicted values by this model are different from the actual values. According to that, SRGM-6 is the worst model to assess reliability. This model is based

on the methodology of Gompertz distribution. This methodology allows either increasing or decreasing the rates of failures based on the shape parameters.

Table 6. The fitting results for all SRGMs based on GNOME dataset version 2.0.

Model	SSE	MSE	Reliability
SRGM-1	99.5532	5.5307	0.9348
SRGM-2	95.8630	5.6390	0.9852
SRGM-3	96.2211	5.6600	0.99660
SRGM-4	96.1235	5.65451	1.0002
SRGM-5	95.34044	5.95666	0.9557
SRGM-6	155.7741	7.4178	0.9356

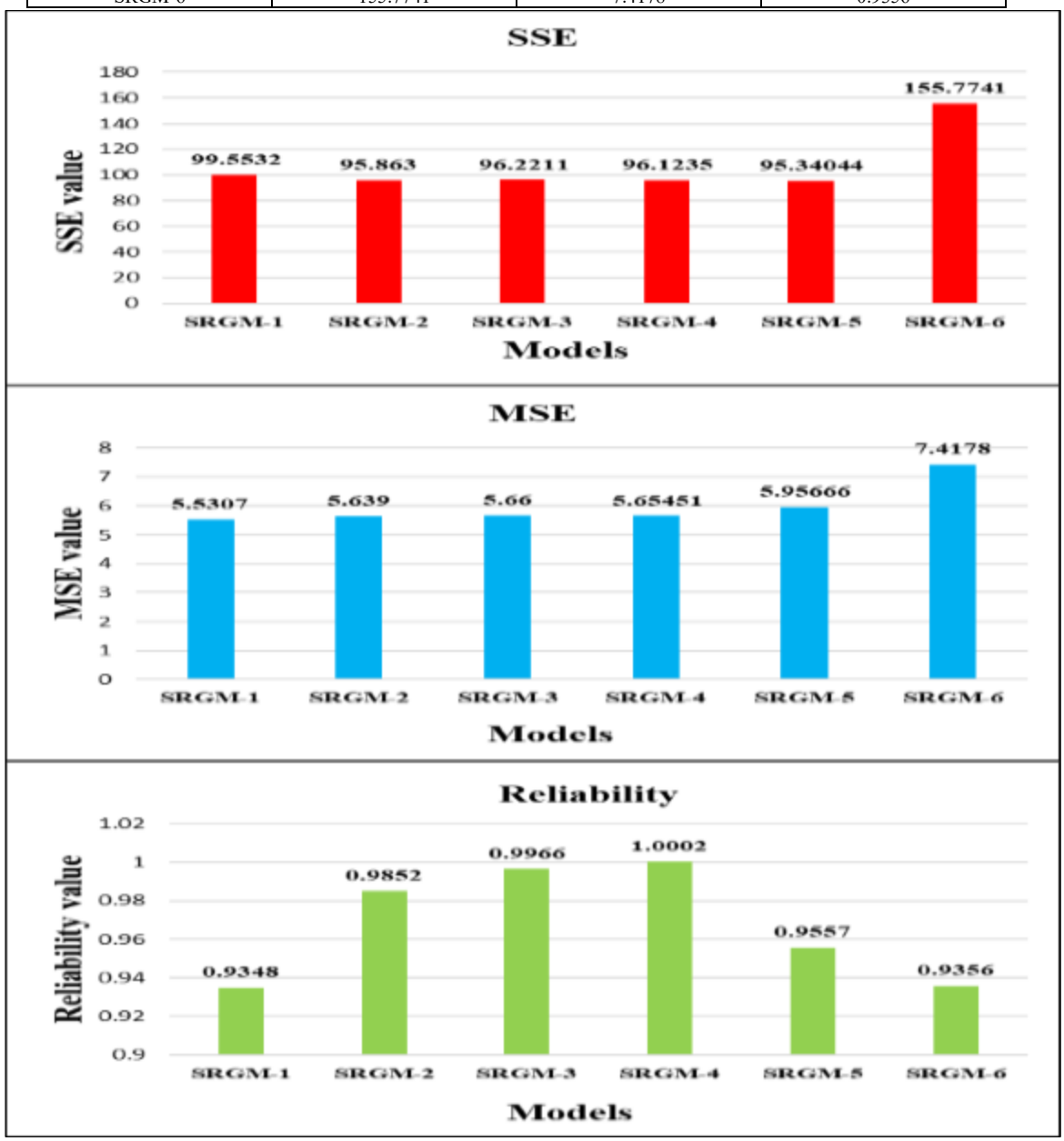

Fig. 2. The fitting bar chart's results for all SRGMs based on GNOME dataset version 2.0.

According to Table 7, and Fig. 3, which represent the evaluation metrics for SRGMs based on GNOME 2.2, SRGM-5 has the smallest value for SSE, and this means that this model has the most predictive execution than the remaining models because the predicted values by this model are the most in line with the actual values. According to that, SRGM-5 is the best model to assess reliability accurately. This model is based on the methodology of imperfect debugging. This methodology supposes that, in the debugging operation, there is a possibility for at least introducing only one new fault. And on the other hand, SRGM-6 has the highest value for SSE, so this model has less predictive execution than the remaining models because the predicted values by this model are different from the actual values. According to that, SRGM-6 is the worst model to assess reliability. This model is based on the methodology of Gompertz distribution. This methodology allows either increasing or decreasing the rates of failures based on the shape parameters.

Table 7. The fitting results for all SRGMs based on GNOME dataset version 2.2.

Model	SSE	MSE	Reliability
SRGM-1	45.0693	3.4668	0.5665
SRGM-2	44.66213	3.7222	0.7526
SRGM-3	47.5781	3.9999	1.59206
SRGM-4	44.60919	3.7174	1.0100
SRGM-5	15.5150	1.41046	0.6994
SRGM-6	70.1503	4.3843	0.999973

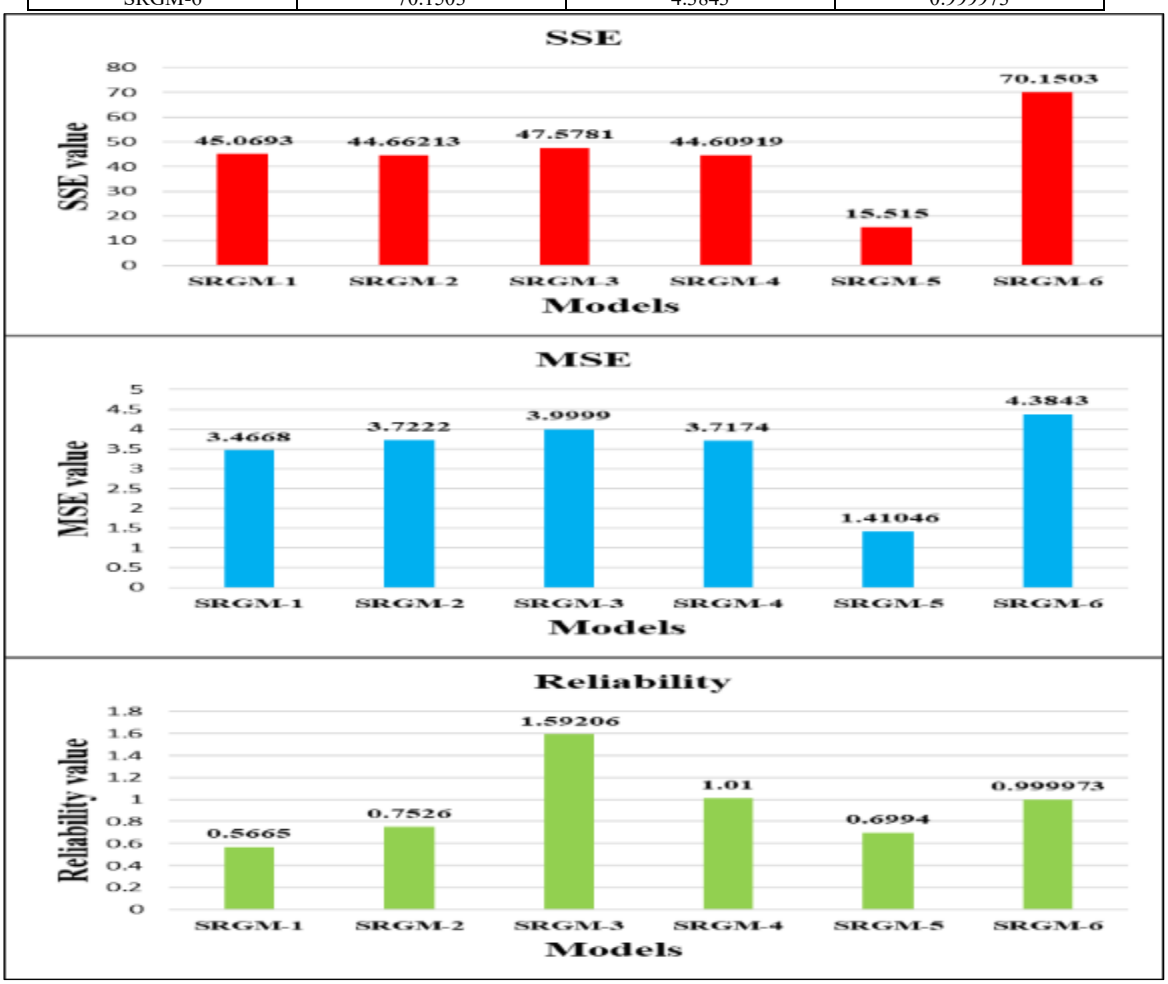

Fig. 3. The fitting bar chart's results for all SRGMs based on GNOME dataset version 2.2.

According to Table 8, and Fig. 4, which represent the evaluation metrics for SRGMs based on GNOME 2.4, SRGM-5 has the smallest value for SSE, and this means that this model has the most predictive execution than the remaining models because the predicted values by this model are the most in line with the actual values. According to that, SRGM-5 is the best model to assess reliability accurately. This model is based on the methodology of imperfect debugging. This methodology supposes that, in the debugging operation, there is a possibility for at least introducing only one new fault. And on the other hand, SRGM-6 has the highest value for SSE, so this model has less predictive execution than the remaining models because the predicted values by this model are different from the actual values. According to that, SRGM-6 is the worst model to assess reliability. This model is based on the methodology of Gompertz distribution. This methodology allows either increasing or decreasing the rates of failures based on the shape parameters.

Table 8. The fitting results for all SRGMs based on GNOME dataset version 2.4.

Model	SSE	MSE	Reliability
SRGM-1	55.8126	3.9866	0.5737
SRGM-2	54.7943	4.2149	0.7516
SRGM-3	54.7460	4.2114	0.8915
SRGM-4	54.8054	4.21581	0.7475
SRGM-5	53.1495	4.42913	0.744271
SRGM-6	79.2473	4.6616	0.9999

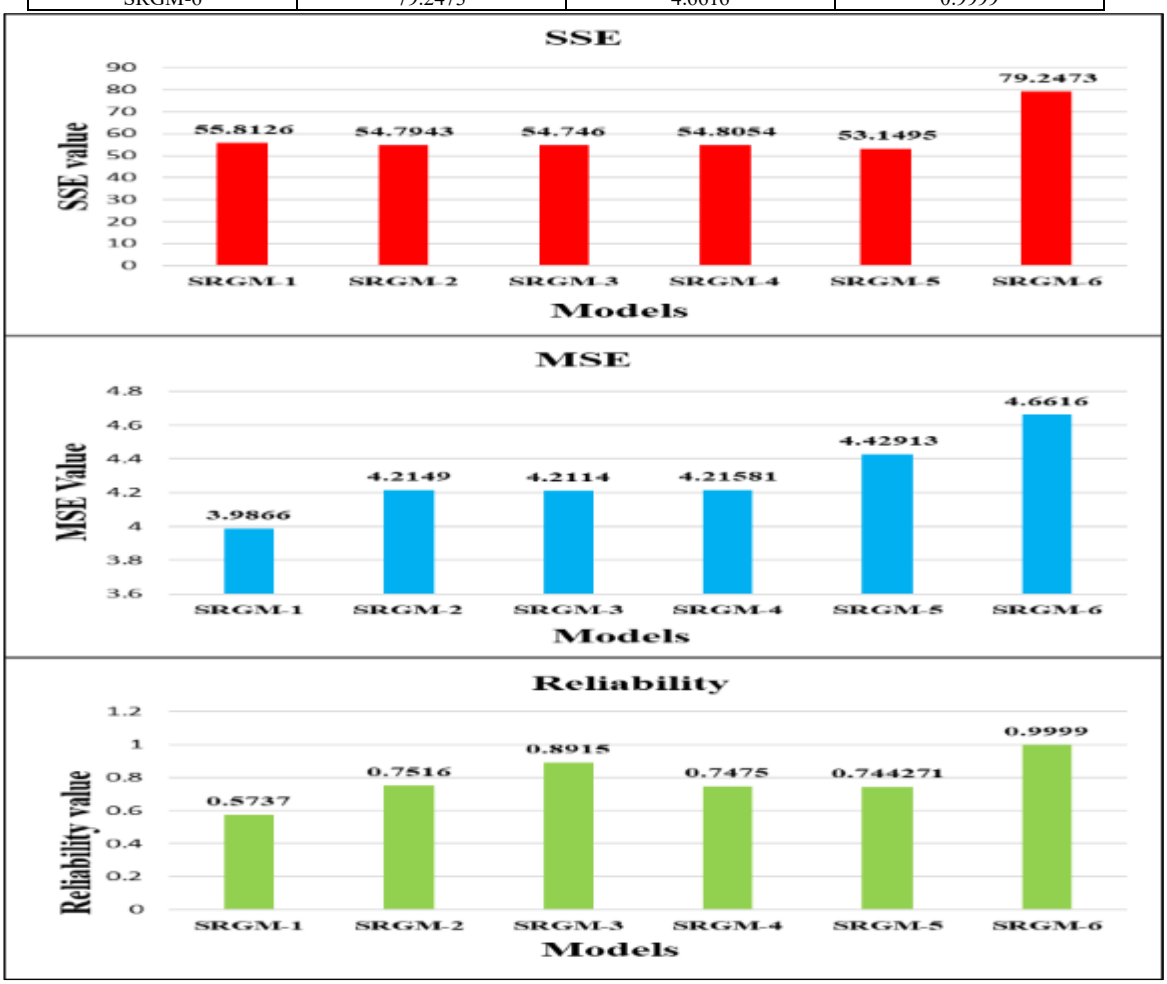

Fig. 4. The fitting bar chart's results for all SRGMs based on GNOME dataset version 2.4.

Fig. 5, Fig. 6, and Fig. 7 show a relation between the time in weeks as the x-axis and the faults as the y-axis. Fig. 5 determines the fitting results for the first version of the dataset GNOME 2.0, Fig. 6 determines the fitting results for the second version of the dataset GNOME 2.2, and Fig. 7 determines the fitting results for the third version of the dataset GNOME 2.4.

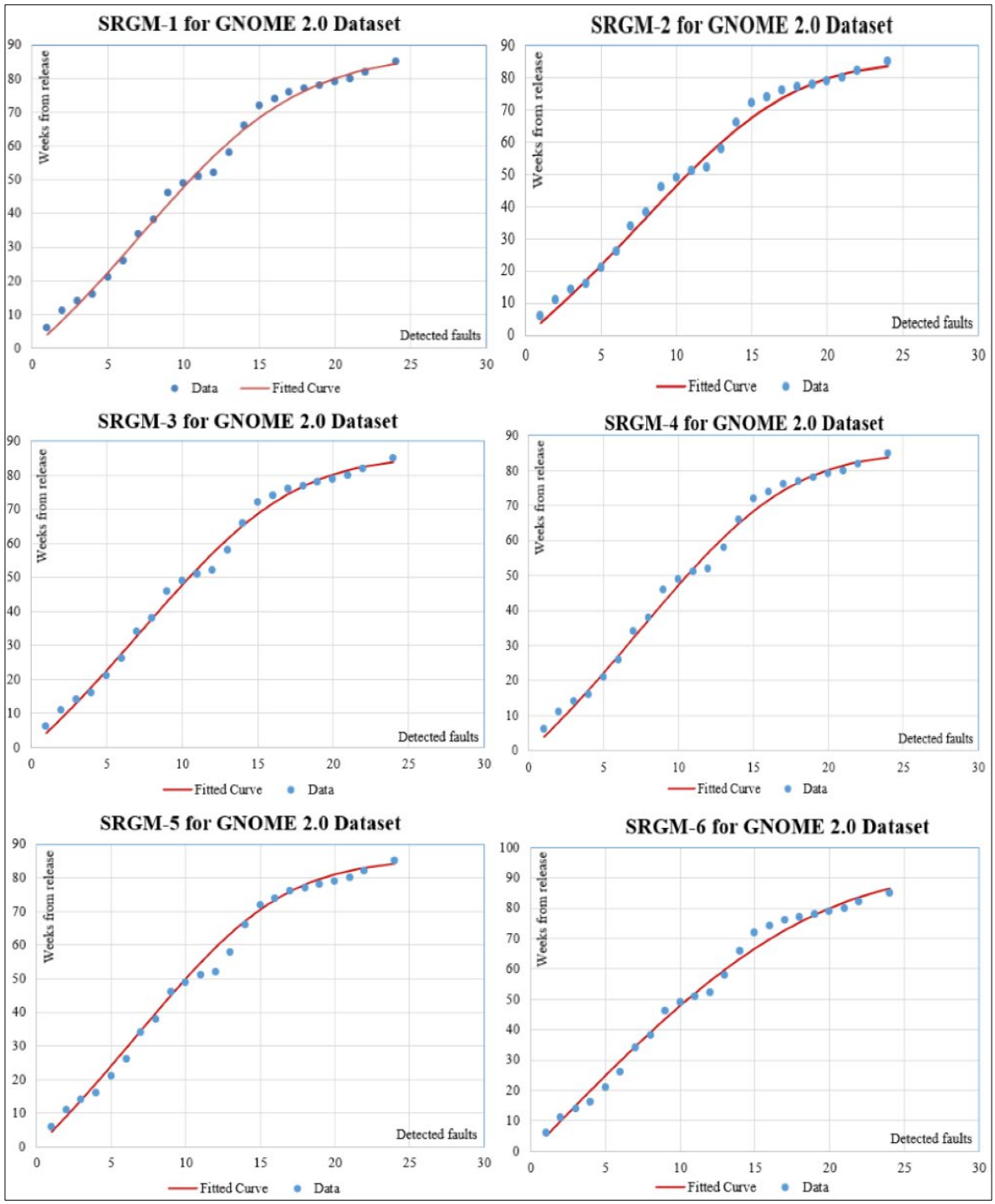

Fig. 5. The fitting curves for SRGMs based on GNOME dataset version 2.0.

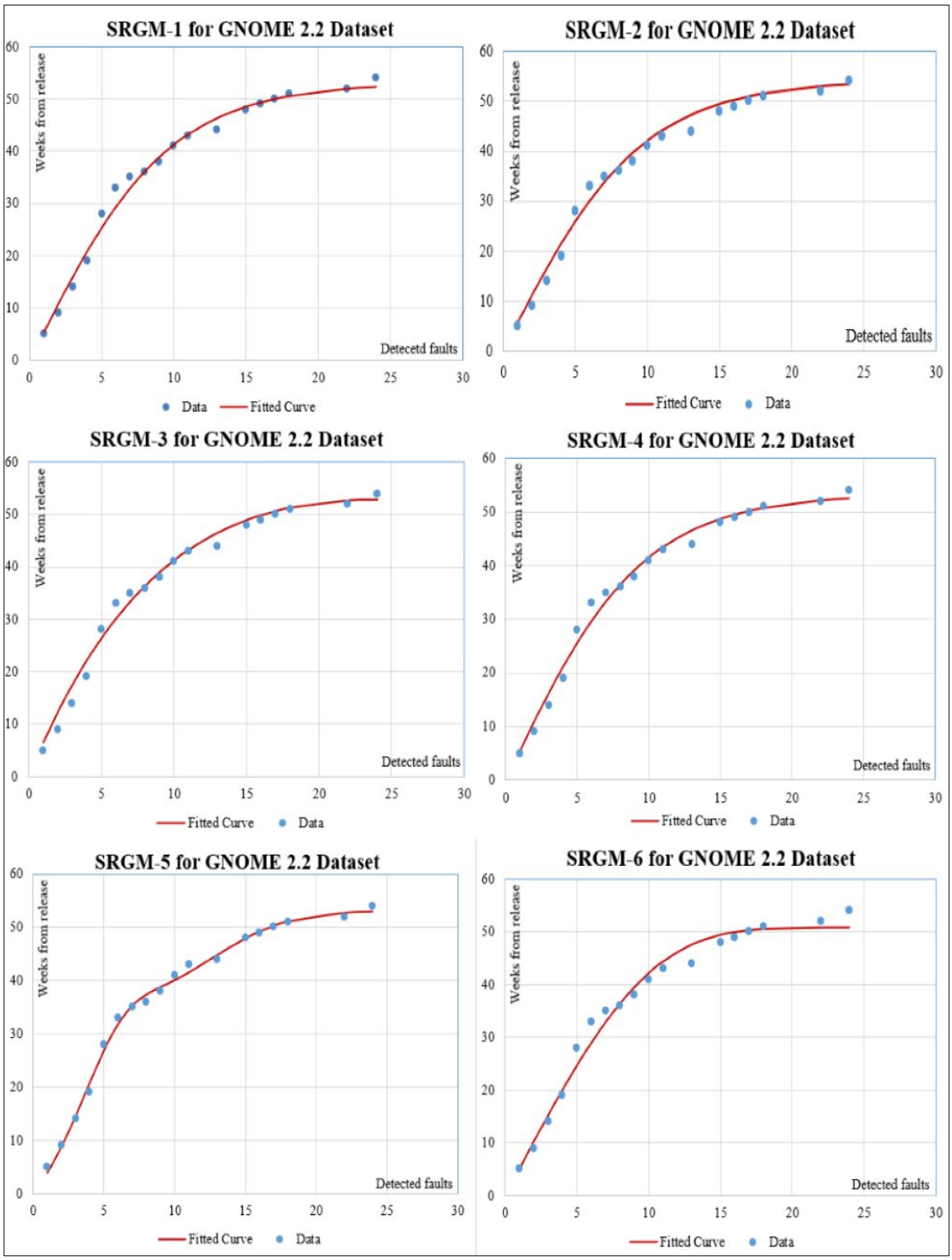

Fig. 6. The fitting curves for SRGMs based on GNOME dataset version 2.2.

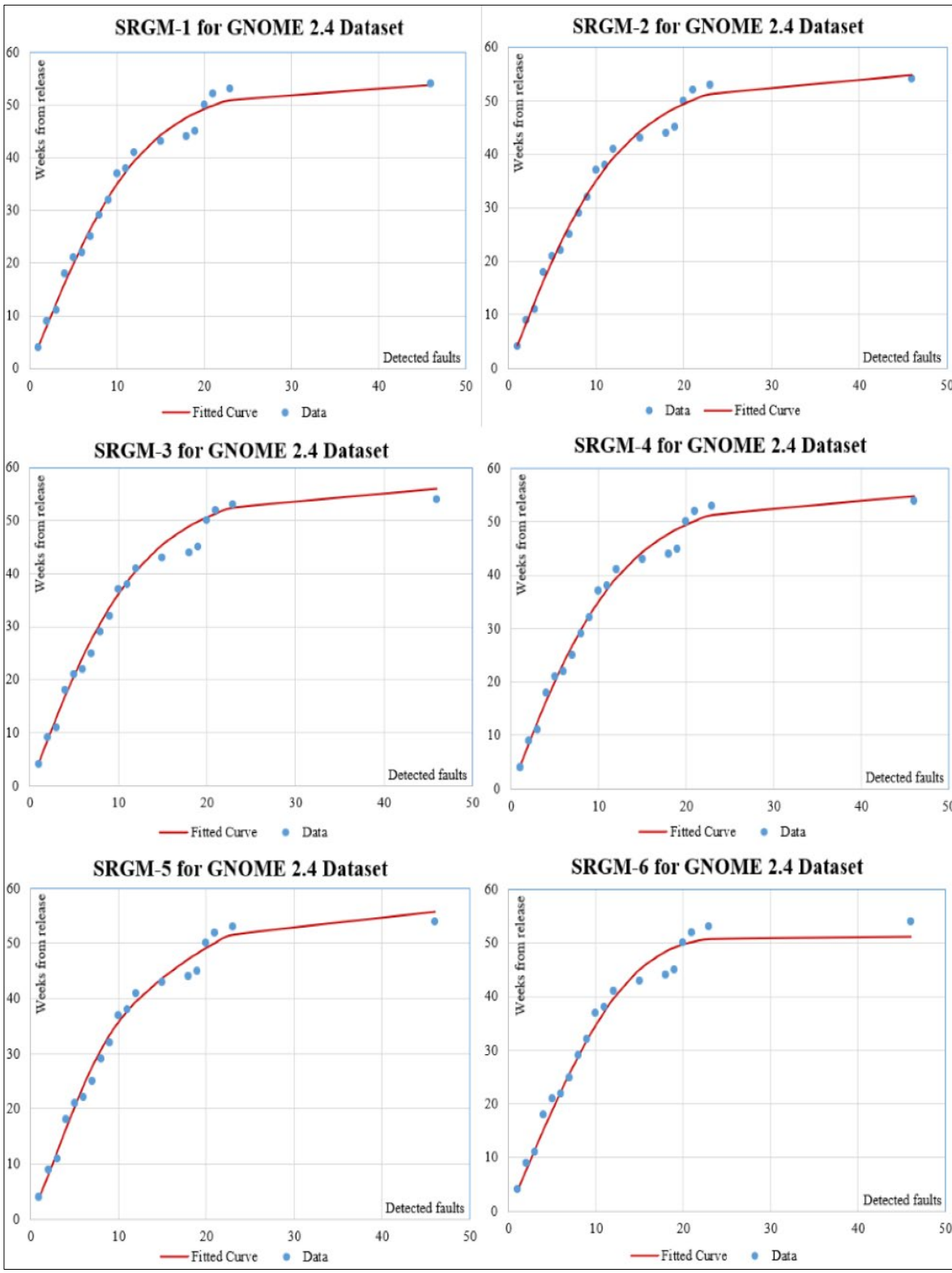

Fig. 7. The fitting curves for SRGMs based on GNOME dataset version 2.4.

It's known that we can detect more faults over time, so the figures show the positive relationship between the time and the number of faults. The previous fitting figures show how much MPA estimates the SRGMs' parameters to be the estimated values more in line with the actual values of GNOME dataset. There are three steps for estimating parameters as follows: First, MPA calculates the SRGM's parameters according to the upper and the lower bounds for each parameter. Second, SRGM's mean value function calculates the estimated values by making substitution with the estimated parameters. Third, SRGM's mean value function calculates the evaluation metrics based on the actual and estimated values. The previous steps are repeated according to the specified numbers of iterations for MPA until estimates the maximum optimal value for SSE and MSE.

5. Conclusion

In our research, we assess the reliability of open-source software (GNOME) by six models from the probabilistic category based on three different methodologies. We apply MPA to estimate the parameters of the mean value function for the selected models to assess the reliability accurately using these estimated parameters. We used SSE, MSE, and reliability as evaluation metrics to test the performance of the selected models. SRGM-5 estimates the models' parameters with the minimum degree of errors, so SRGM-5 calculates the most accurate reliability value for all GNOME dataset versions. And on the other hand, SRGM-6 estimates the worst reliability value for all GNOME dataset versions. We can conclude that the best model for assessment reliability is SRGM-5 when using GNOME dataset. In the future, we will assess CSS reliability using probabilistic models and metaheuristic algorithms.

References

[1] T. Saravanan, S. Jha, G. Sabharwal and S. Narayan, "Comparative Analysis of Software Life Cycle Models," *2020 2nd International Conference on Advances in Computing, Communication Control and Networking (ICACCCN)*, pp. 906-909,2020.

[2] S. Shukla, R.K.Behera, S. Misra, S.K.Rath," Software reliability assessment using deep learning technique," in *Towards Extensible and Adaptable Methods in Computing*, pp.57-68, 2018.

[3] D. Raval, D. Bhatt, M. K. Kumhar, V. Parikh, and D. Vyas, "Medical Diagnosis System Using Machine Learning," *International Journal of Computer Science & Communication*, vol. 7, pp. 177–182, 2015.

[4] R. M. A. Wadi and L. S. Khalf, "Knowledge Management in Higher Education Institutions: Facts and Challenges," in *European, Asian,Middle Eastern, North African Conference on Management \& Information Systems*, pp.241-248, 2021.

[5] S.Randhawa, "Open source software and libraries," *Twenty First Century Publications*, 2008.

[6] N. Medrous and K. Nemmiche, "Towards a new form of free merchandising with Open Source Software," *Journal of Business and Trade Economics*, 2021.

[7] K. Ilievski, A. Behlić, and J. Achkoski, "E-learning platforms: The future of education," *2-nd International Scientific Conference MILCON'19,Skopje.*, pp. 62-67, 2019.

[8] O. Y. Aydin and Y. Aydın, "Security of Open Source Software," Accessed: May. 30,2022 [Online], April 2018. Available: https://tinyurl.com/y5epm7yc.

[9] A. K. Ray and D. B. Ramesh, "Open Source Software (OSS) for Management of Library and Information Services: An Overview," *International Journal of Library and Information Studies*, vol. 7, no. 2, pp. 20–31, 2017.

[10]RiskSense, "The Dark Reality of Open Source,"Accessed May. 30,2022, [Online], May 2020. Available: https://tinyurl.com/bdhnf9y8

[11] F. Kluitenberg, "Evaluating Quality of Open Source Components," MSc dissertation, Twente Univ., 2018.

[12] A. Alami, Y. Dittrich, and A. Wsowski, "Influencers of quality assurance in an open source community," *Proc. - Int. Conf. Softw. Eng.*, pp. 61–68, 2018.

[13] A. Al Hussein, "An Object-Oriented Software Metric Tool to Evaluate The Quality of Open Source Software," *IJCSNS International Journal of Computer Science and Network Security*, vol. 17, no. 4, pp. 345–351, 2017.

[14] W. Agustiono, "An Open Source Software Quality Model and Its Applicability for Assessing E-commerce Content Management Systems," in *International Conference on Science and Technology (ICST 2018)*, vol. 1, pp. 699–704, 2018.

[15] L. V. Utkin and F. P. A. Coolen, "A robust weighted SVR-based software reliability growth model," *Reliability Engineering \& System Safety*, vol. 176, no. April, pp. 93–101, 2018.

[16] K. Sahu and R. K. Srivastava, "Needs and importance of reliability prediction: An industrial perspective," *Information Sciences Letters*, vol. 9, no. 1, pp. 33–37, 2020.

[17] P. Kumar, L. K. Singh, and C. Kumar, "Suitability analysis of software reliability models for its applicability on NPP systems," *Quality and Reliability Engineering International*, vol. 34, no. 8, pp. 1491–1509, 2018.

[18] A. A. Musa, S. H. Imam, A. Choudhary, and A. P. Agrawal, "Parameter estimation of software reliability growth models: A comparison between grey Wolf optimizer and improved grey Wolf optimizer," in *2021 11th International Conference on Cloud Computing, Data Science \& Engineering (Confluence)*, pp. 611–617, 2021.

[19] A. Choudhary, A. S. Baghel, and O. P. Sangwan, "Efficient parameter estimation of software reliability growth models using harmony search," *IET Software*, vol. 11, no. 6, pp. 286–291, 2017.

[20] N. Gandhi, N. Gondwal, and A. Tandon, "Reliability Modeling of OSS Systems based on Innovation-Diffusion Theory and Imperfect Debugging," in *ICITKM*, vol. 14, pp. 53–58, 2018.

[21] M. Zhu and H. Pham, "A multi-release software reliability modeling for open source software incorporating dependent fault detection process," *Annals of Operations Research*, vol. 269, no. 1, pp. 773–790, 2018.

[22] Diwakar and A. G. Aggarwal, "Multi Release Reliability Growth Modeling for Open Source Software Under Imperfect Debugging," in *System Performance and Management Analytics*, pp. 77–86, 2019.

[23] J. Wang, "Model of Open Source Software Reliability with Fault Introduction Obeying the Generalized Pareto Distribution," *Arabian Journal for Science and Engineering*, vol. 46, no. 4, pp. 3981–4000, 2021.

[24] T. Yaghoobi, "Selection of optimal software reliability growth model using a diversity index," *Soft Computing*, vol. 25, no. 7, pp. 5339–5353, 2021.

[25] A. Faramarzi, M. Heidarinejad, S. Mirjalili, and A. H. Gandomi, "Marine Predators Algorithm: A nature-inspired metaheuristic," *Expert systems with applications*, vol. 152, p. 113377, 2020.

[26] C.S.Chowdary, R.S.Prasad, K.Sobhana," Burr type III software reliability growth model," *IOSR Journal of Computer Engineering*, vol.1, no.17, pp.49-54,2015.

[27] A. Chaudhary, A. P. Agarwal, A. Rana, and V. Kumar, "Crow Search Optimization Based Approach for Parameter Estimation of SRGMs," in *2019 Amity International Conference on Artificial Intelligence (AICAI)*, pp. 583–587, 2019.

[28] GNOME OSS dataset. Accessed: May. 30,2022 [Online]. Available: https://gitlab.gnome.org/GNOME

[29] JUDDI OSS dataset. Accessed: May. 30,2022 [Online]. Available: https://juddi.apache.org/

[30] Apache OSS dataset. Accessed: May. 30,2022 [Online]. Available: https://issues.apache.org/

[31] M. Ghoneimy, H. A. Hassan, and E. Nabil, "A New Hybrid Clustering Method of Binary Differential Evolution and Marine Predators Algorithm for Multi-omics Datasets," *International Journal of Intelligent Engineering and Systems*, vol. 14, no. 2, pp. 421–431, 2021.

[32] A. Eid, S. Kamel, and L. Abualigah, "Marine predators algorithm for optimal allocation of active and reactive power resources in distribution networks," Neural Computing and Applications, vol. 33, no. 21, pp. 14327–14355, 2021.

[33] J. Yang, M. Zheng, and S. Chen, "Illumination correction with optimized kernel extreme learning machine based on marine predators algorithm," *Color Research \& Application*, vol. 47, no. 3, pp. 630–643, 2022.

[34] X. Lu, Y. A. Nanehkaran, and M. Karimi Fard, "A Method for Optimal Detection of Lung Cancer Based on Deep Learning Optimized by Marine Predators Algorithm," *Computational Intelligence and Neuroscience*, vol. 2021, 2021.

[35] A. H. Yakout, W. Sabry, A. Y. Abdelaziz, H. M. Hasanien, K. M. AboRas, and H. Kotb, "Enhancement of frequency stability of power systems integrated with wind energy using marine predator algorithm based PIDA controlled STATCOM," *Alexandria Engineering Journal*, vol. 61, no. 8, pp. 5851–5867, 2022.

[36] X. Li, Y. F. Li, M. Xie, and S. H. Ng, "Reliability analysis and optimal version-updating for open source software," *Information and Software Technology*, vol. 53, no. 9, pp. 929–936, 2011.

[37] D. M. German, "The GNOME Project: A case study of open source, global software development," *Software Process: Improvement and Practice*, vol. 8, no. 4, pp. 201–215, 2003.

[38] K. Sharma, R. Garg, C. K. Nagpal, and R. K. Garg, "Selection of Optimal Software Reliability Growth Models Using a Distance Based Approach," *IEEE Transactions on Reliability*, vol. 59, no. 2, pp. 266–276, 2010.

[39] B. Pachauri, A. Kumar, and J. Dhar, "Reliability analysis of open source software systems considering the effect of previously released version," *International Journal of Computers and Applications*,2019.

[40] C. Y. Huang, J. H. Lo, and S. Y. Kuo, "Pragmatic study of parametric decomposition models for estimating software reliability growth,"in *Proceedings Ninth International Symposium on Software Reliability Engineering (Cat. No. 98TB100257)*, pp. 111–123, 1998.