I.J. Information Technology and Computer Science, **2022, 3, 1-15**
Published Online June 2022 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijitcs.2022.03.01

Reliability Assessment for Open-Source Software Using Deterministic and Probabilistic Models

Islam S. Ramadan and Hany M. Harb
Department of Computer Science, College of Information Technology, MUST University, Cairo, Egypt
E-mail: islam.saied@must.edu.eg, hany.harb@must.edu.eg

Hamdy M. Mousa and Mohammed G. Malhat
Department of Computer Science, College of Computers and Information, Menoufia University, Menoufia, Egypt
E-mail: hamdimmm@hotmail.com, m.gmalhat@yahoo.com

Received: 12 January 2022; Revised: 02 March 2022; Accepted: 01 April 2022; Published: 08 June 2022

Abstract: Nowadays, computer software plays a significant role in all fields of our life. Essentially open-source software provides economic benefits for software companies such that it allows building new software without the need to create it from scratch. Therefore, it is extremely used, and accordingly, open-source software's quality is a critical issue and one of the top research directions in the literature. In the development cycles of the software, checking the software reliability is an important indicator to release software or not. The deterministic and probabilistic models are the two main categories of models used to assess software reliability. In this paper, we perform a comparative study between eight different software reliability models: two deterministic models, and six probabilistic models based on three different methodologies: perfect debugging, imperfect debugging, and Gompertz distribution. We evaluate the employed models using three versions of a standard open-source dataset which is GNU's Not Unix Network Object Model Environment projects. We evaluate the employed models using four evaluation criteria: sum of square error, mean square error, R-square, and reliability. The experimental results showed that for the first version of the open-source dataset SRGM-4 based on imperfect debugging methodology achieved the best reliability result, and for the last two versions of the open-source dataset SRGM-6 based on Gompertz distribution methodology achieved the best reliability result in terms of sum of square error, mean square error, and R-square.

Index Terms: Software reliability models, Deterministic models, Probabilistic models, Open-source software, least squares estimation.

1. Introduction

Recently, the software's usage has rapidly grown in all fields of life such as industry, medical, business, marketing, military, and education [1]. For example, OpenEMR is an open-source electronic medical record system used to save and transmit important health information for patients [2]. Also, in the education field, LabVIEW is a graphical environment for programming used by many scientists and engineers to develop control systems and sophisticated measurement using wires and graphical icons like a flowchart [3]. The software is classified into two types [4], which are:

- Open-Source Software (OSS): OSS allows anyone to use source code for free. The owner of OSS gives the rights to learn, edit, and share the software to anyone and for any purpose. The source code of OSS is always written in a programming language to run on a computer in a binary format [5].
- Closed Source Software (CSS): CSS is implemented by specific developers or companies, so its source code is changed only by the company or the developer who developed the software and cannot be publicly available to the intended users [6].

In the literature, the researchers and software experts have determined various advantages for OSS [5], such as:

- Freedom to use: Any person can download, adjust the source code based on his/her needs, and release the software without any limitations.
- Software cost: OSS usually does not require any licensing fees or any maintenance fees. The expenditure can only need for documentation and support.
- License management: OSS can be download and installed several times in several locations, so it never tracks, counts, or supervises for license compliance.

- Free from vendor lock-in: The users of OSS are independent to adapt software according to the requirement.

As a result, for the advantages, OSS projects allow the developers to combine new capabilities to their applications quickly without the need to make it from scratch. Therefore, it is noticed that between 80% and 90% of the code in the most recent applications used OSS components [7]. Additionally, OSS allows companies to decrease their software costs, so it provides economic benefits [8]. Therefore, we don't be surprised when RedHat make a report in 2020 and found that 95% of companies view OSS as strategically significant to their business [7]. The quality and performance of OSS is a very critical topic in the literatures [9,10,11,12]. Many researchers propose different methods and approaches to assess and evaluate OSS. Software reliability is one of the most significant metrics used to evaluate OSS in the software development process [13] because of many reasons such as [14]: 1) It is used to determine the failure or approval of the software product, and 2) Reliability prediction allows us to be sure that the software product delivers to the customers perfectly. Therefore, there are different models are used to assess the reliability of OSS, which are classified into two types [15]:

- Deterministic models analyze the program source code, use the result as input, and do not combine any random events or values. For example, Halstead Model and McCabe's Cyclomatic Complexity V(G) Models [16].
- Probabilistic models describe error correction and failure cases as random events. For example, Error Seeding, Reliability Growth, Failure Rate, Input Domain, Program Structure[16].

In this paper, we aim to achieve many objectives are: 1) Make an overview of OSS reliability models categories, 2) Select different types of OSS reliability models based on different methodologies, and 3) Determine the optimal model for reliability assessment to solve the critical issue about which model is the best one for reliability assessment. To achieve the research objectives: 1) Discuss deterministic and probabilistic OSS reliability models, 2) Explain two deterministic models with examples, 3) Assess OSS reliability for three versions of GNU's Not Unix Network Object Model Environment (GNOME) projects using six probabilistic models, and 4) Compare the selected models through different well-known evaluation criteria to estimate the performance of the selected models. The experimental results showed that SRGM-4, based on imperfect debugging methodology, has the most reliability value for the first version of GNOME projects, and SRGM-6, based on Gompertz distribution methodology, has the most reliability value for the last two versions of GNOME projects. However, SRGM-6 does not evaluate the best reliability value for the first version of GNOME projects but is not the worst one, so we can say that SRGM-6 is the most optimal model for reliability assessment.

To present content in an organized manner the remaining parts of the paper are as follows: Section 2 presents and discusses the related work in OSS assessment reliability. Section 3 describes the most popular implemented software reliability models in detail. Section 4 shows an experimental study for the most popular selected models using different criteria over different datasets. Finally, section 5 contains the paper's conclusion and identifies directions for the future in this area.

2. Related Work

AS mentioned before, software reliability growth models (SRGMs) are classified into deterministic and probabilistic models. Both Table 1 and Table 2 show a more detailed picture for SRGMs.

Deterministic models contain two sub-categories: the Halstead model and McCabe's Cyclomatic Complexity models, and both will discuss in detail in the next section. On the other hand, probabilistic models contain ten sub-categories [16]:

- Error Seeding: in the testing phase, error seeding models are an indicator of the correctness measure. Mills model is an example that measures indigenous errors' number by seeding some pseudo errors in the program. It uses hypergeometric distribution for removed and induced errors, and its probability is the following:

$$p(k; N+n1, n1, r) = \frac{\binom{n1}{k}\binom{n}{k-1}}{\binom{N+n1}{r}} \tag{1}$$

where N is indigenous errors, *n1* is induced errors, r is removed errors' number during debugging, k is induced errors' number in r removed errors, and r – k is indigenous errors' number in r removed errors.

Table 1. Software reliability growth models categories[16].

Deterministic Models	Halstead Model - McCabe's Cyclomatic Complexity
Probabilistic Models	Error Seeding – Reliability Growth – Failure Rate – Input Domain – Program Structure – Curve Fitting – Execution Path – Non-Homogenous Poisson Process (NHPP) Models – Bayesian Models – Markov Models

Table 2. Deterministic models subcategory[16].

Error Seeding	Mills Model - Lipow Model
Reliability Growth	Duane Growth - Gompertz Growth Curve - Wall and Ferguson Model - Logistic Growth Curve - wagoner's Weibull - Hyperbolic Growth Curve
Failure Rate	JMD Model - S-W Model - JMGD Model - MGP Model - Go Imperfect Debugging - MSW Model
Input Domain	Basic Input-Domain Model - Input Domain-based Stochastic Model - Nelson Model
Program Structure	Cheung's user-oriented Markov - Littlewood Markov Structure
Curve Fitting	Estimation of errors - Estimation of time-between failures - Estimation of change - Estimation of failure-rate
Execution Path	Shooman Decomposition Model
NHPP Models	Delayed S-Shaped Growth Model - Go NHPP - Hyperexponential Growth Model - Inflection S-Shaped Growth Model
Bayesian Models	littlewood-Verrall Model - Basu and Ebrahimi Model - Cid and Achcar Model - Zagueira Model
Markov Models	Linear Death with Perfect debugging - Linear Death with Imperfect debugging - death with perfect debugging - Nonstationary linear - Nonstationary linear birth-and-death

- Reliability Growth: Over the last years, many SRGMs have developed each of them has specific limitations and assumptions such as the correlation between time and failure rate change. For example, Logistic growth curve estimates the encountered cumulative faults during the debugging by the following equation:

$$m(t) = \frac{k}{1+ae-\beta^t} \tag{2}$$

where *m(t)* is the cumulative faults noticed during the time interval (t), *and k*, *a*, and *B* are parameters calculated using regression analysis.

- Failure Rate estimates the reliability using the rate of failure for the software program. For example, Jelinski-Moranda deeutrophication (JMD) considers the faults' occurrence as random events, and the impact of all failures are the same on the reliability such that it calculates the fault rate of the software program λ, for the ith time interval using the following function:

$$\lambda(ti) = \phi[N-(-1)] \tag{3}$$

where ϕ is constant of proportionality, and N is period between the (i-1) st and ith faults.

- Input Domain: In input domain models, the most important indicator for estimating the reliability is the execution of the input state. For example, the Basic input-domain model needs the input and output domain to be known to measure the reliability, so the reliability determines the probability of successful run(s) selected randomly from the input space.
- Program Structure: Program structure models are used to estimate the interdependence of modules and their reliability. Every module act as a node, so the software's reliability depends on the individual nodes' reliability.
- Curve Fitting analyzes the relationship between the failure rate and software reliability, the remaining number of errors, the amount of modification done in the software. For example, the Estimation of errors model uses a linear regression model to estimate the fault's number in the software program given by the following equation:

$$N = \sum_i a_i \; x_i \tag{4}$$

where N is faults' number in the program, xi is ith error factor, and ai is coefficients.

- Execution Path: This model calculates the reliability based on:
  - The selected arbitrary path.
  - the required time for implementing the path.
  - The failure causes by this path.

For example, the Shooman decomposition model represents the implementation of the software program using structured programming, which consists of different independent paths. The following equation used to calculate the complete number of failures in N tests:

$$n_f = \sum_{i=1}^{k} fiqi \tag{5}$$

and the system fails probability on each run of case *i* calculated by:

$$qi = \lim_{n\to\infty} \frac{nf}{N} \tag{6}$$

where *N* is test cases' numbers, *k* is paths' number, *qi* is probability of errors of each run of case *i*, *fi* is selection probability of case *i*, and *nf* is total failures' number in *N* test.

- NHPP Models: In any program, the faults are represented as NHPP so, these models calculate the number of faults that should be expected using the mean value function for a specific interval of time. For example, Delayed S-shaped NHPP model detects and isolates errors using the following mean value function:

$$m(t) = a[1 - (1 + bt)exp\,(-bt)] \tag{7}$$

where a is number of expected faults, and b is error detection and isolation rate.

- Bayesian models: these models rely on Bayesian statistics, which require evidence to update the previous failure rate distribution and the important parameter using a likelihood function.
- Markov Models: These models assume the performance of the software in the future depends on the current state only, not on the past errors or history. Also, the fault rate is symmetrical to the remaining faults in the program at any random time t.

Because the Software complexity metrics represent significant criteria to estimate the software product, these criteria include effort, complexity, code size, time consumption, etc. We can test the program's complexity factors using Halstead software science, so Govil [17] checks Halstead metrics for palindrome code implemented in C, C++, PHP, Java, and Python. The metrics results for Python language have less effort, difficulty, the required time to program than other programming languages. The rapid development in programming and the utilization of many framework interfaces make programs more complex, so measuring software complexity has become an urgent matter, so Hariprasad, Vidhyagaran, etc., [18] check Halstead metrics for simple code implemented in C++ and Python. The evaluated metrics based on the number of operators and operands show that code implemented in python has less effort, expected bugs, and the required time to program than C++ language. The software complexity metrics evaluate the software's characteristics in all its aspects. Abdulkareem, and Abboud [19] compute the complexity level for different programming languages like C, JavaScript, and Python using the Halstead model. The Halstead metrics results show that Python is simple and user-friendly. Nevertheless, Java is modern, powerful, and is more complicated than other programming languages. The quality of the source code can be measured using cyclomatic complexity such that researchers found a relationship between cyclomatic complexity and line of code also between cyclomatic complexity and testing paths of the code [20]. On the other hand, many probabilistic models estimate OSS reliability as a result so, in the previous years' many research papers published in this direction. Zhu and Pham [21] proposed a reliability model for OSS and didn't ignore the fault-dependent relation between current releases and previous releases. Estimate reliability for two OSS Juddi and Apache datasets. The proposed model evaluates reliability more accurately than another seven models used in the comparison. Aggarwal [22] thought in The OSS development process, the faults in the recent version were divided into two parts. Part one for the faults in the previous versions will remain in the current version. In addition, new faults will be introduced because of the modification of the code. Consequently, they proposed a reliability model for multi-release OSS with imperfect debugging to estimate reliability for Apache with three versions of datasets. The comparison criteria show that the proposed model has more accurate results than other models.

3. Methodology

This paper presents assessment reliability for OSS with deterministic and probabilistic models as the following:

- Deterministic models:

This category of models analyzes the program source code, computes a list of metrics that reflect the degree of the code complexity [16]. There are two deterministic models:

  - The Halstead model [19] calculates a list of metrics such as program length, program volume calculated from (14) to (20). Also, we represent an example for it as illustrated in Fig.1, Table 3, and Table 4.
  - McCabe's Cyclomatic Complexity Model [23,24] calculates the complexity of the code based on the number of branches in the code using (21). Also, Fig.2 shows an example of it.

- Probabilistic models:

We present six models based on three different methodologies:

- Perfect debugging methodology [25]: is the detection process of the faults without incorporating additional faults. This methodology contains only one case.
  - Case 1 [25]: the function of fault content represented by *a(N),* which equals constant *a*. This case means debugging process will not cause new faults. SRGM-1 is the only model under this case. Therefore here,

$$a(N) = a \tag{8}$$

- Imperfect debugging methodology [25]: is the possibility of introducing new faults while fixing the previous one. This methodology contains four cases.
  - Case 2 [25]: the number of faults is a linear function of the user's number as the following equation:

$$a(N) = a(1 + \alpha N) \tag{9}$$

SRGM-2 is the only model under this case.

  - Case 3 [25]: the faults' number introduced exponentially related to the number of users according to the following equation:

$$a(N) = ae^{\alpha N} \tag{10}$$

SRGM-3 is the only model under this case.

  - Case 4 [25]: we assume the introduction rate of new faults like the removed faults number function in the software as the following equation:

$$a(N) = a + \alpha m(N) \tag{11}$$

SRGM-4 is the only model under this case.

  - Case 5 [25]: we assume that the new faults are generated exponentially for every detected fault according to the following equation:

$$a(N) = c + a(1 - e^{-\alpha N}) \tag{12}$$

SRGM-5 is the only model under this case.

- Gompertz distribution methodology: one type of mathematical model used as growth model [26]. SRGM-6 based on the following two assumption:
  - The number of initial faults based on Poisson distribution with a parameter *a*.
  - The failures occur at independent random times based on the Gompertz distribution function with the cumulative function:

$$F(t) = 1 - e^{(1-e^{bt})} \tag{13}$$

4. The implemented Models

In this section, we will discuss the implemented models in both deterministic and probabilistic categories, and for this purpose, this section consists of two parts:

4.1. Deterministic implemented models

- Halstead Model [19]: Halstead model measures the complexity of the software using operators and operands from source code. It is relied on the following metrics which are *n1* is distinct number of operators, *n2* is distinct number of operands, *N1* is Total number of operators' occurrences, and *N2* is total number of operands' occurrences. Based on previous measurements the Halstead model calculates the metrics as the following:

- Program Length (*N*) is the number of total operands and operators in the program evaluated by

$$N = N1 + N2 \tag{14}$$

- Program Vocabulary (*n*) is the complete number of discrete operands and operators in the program evaluated by

$$n = n1 + n2 \tag{15}$$

- Program Volume (*V*) is the implementation size of any algorithm evaluated by

$$V = N\ \log_2(n) \tag{16}$$

- Program Difficulty (*D*) is ability to learn and understand the program evaluated by

$$D = \frac{n1}{2} * \frac{N2}{n2} \tag{17}$$

- Program Effort (*E*) is the required efforts needed for learning and implementing the program evaluated by

$$E = D * V \tag{18}$$

- Number of Bugs (*B*) is the predictable number of bugs in the program comparable to the effort evaluated by

$$B = V/3000 \tag{19}$$

- Time (*T*) is the required time to write the program comparable to the effort evaluated by

$$T = \frac{E}{S} \tag{20}$$

Fig.1 shows a Python code example to evaluate Halstead model metrics listed above. Both Table 3 and Table 4 clarify the result of applying Halstead metrics to the code.

```
def is_odd(num):
  if num & 1:
      return true
  return false
  for i in range (1,100):
      print (str(i) +" : "+ str (is_odd (i)))
```

Fig.1. Python code example [19].

Table 3. Operands and operators of python code [19].

Distinct		Total
Operators	n1 = 4 [':' '&' ',' '+']	N1 = 7
Operands	n2 = 5 ['is_odd' 'num' 'True' 'False']	N2 = 9

Table 4. The results of Halstead for python code [19].

Metrics	**Value**
Program Length (*N*)	N = 7 + 9 = 16
Program Vocabulary (*n*)	n = 4 + 5 = 9
Program Volume (*V*)	V = 16 $\log_2$ 9 = 50.719
Program Difficulty (D)	D = 4/2 * 9/5 = 3.6
Program Effort (*E*)	E = 3.6 * 50.719 = 182.588
Number of Bugs (*B*)	B = 50.719/ 3000 = 0.017
Time (*T*)	T = 182.588 / 18 = 10.144

- McCabe's Cyclomatic Complexity Model [23,24]: It was developed in 1976 by McCabe to estimates the complexity of the program by calculating the total number of decisions (branches) in a method such as if, while, for, and switch cases. It is one of the most important metrics used for measuring the source code complexity. It calculated using the equation:

$$VG = e - n + 2 \tag{21}$$

where *e* is the edges' number, *n* is the nodes' number. For example, Fig.2 describes control flow graph (CFG) for insertion sort algorithm and shows that, *V(G)* = 11 – 10 + 2 = 3

4.2. Probabilistic implemented models:

There are different SRGMs used to assess OSS reliability [27]. We select six models to be practical experience. Every model has its mean value function m(t) as the following [25,27]:

- Mean value function for SRGM1 evaluated by

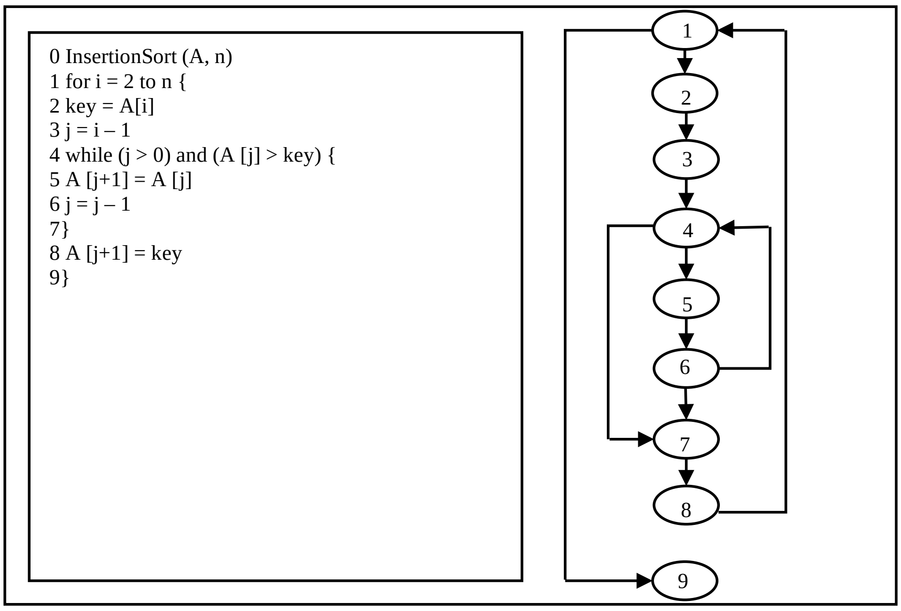
$$m1(t) = a(1 - e^{-bN}) \tag{22}$$

- Mean value function for SRGM2 evaluated by

$$m2(t) = a(\alpha + (1 - \tfrac{\alpha}{b})(1 - e^{-bN})) \tag{23}$$

- Mean value function for SRGM3 evaluated by

$$m3(t) = \frac{ab}{\alpha + b}(e^{\alpha N} - e^{-bN}) \tag{24}$$



```
0 InsertionSort (A, n)
1 for i = 2 to n {
2 key = A[i]
3 j = i – 1
4 while (j > 0) and (A [j] > key) {
5 A [j+1] = A [j]
6 j = j – 1
7}
8 A [j+1] = key
9}
```

Fig.2. Insertion sort algorithm and control flow graph for the algorithm [24].

- Mean value function for SRGM4 evaluated by

$$m4(t) = \frac{a}{1-\alpha}(1 - e^{-b(1-\alpha)N}) \tag{25}$$

- Mean value function for SRGM5 evaluated by

$$m5(t) = (a + c)(1 + e^{-bN}) - \frac{ab}{b-\alpha}(e^{-\alpha N} - e^{-bN}) \tag{26}$$

- Mean value function for SRGM6 evaluated by

$$m6(t) = a(1 - e^{(1-e^{bt})}) \tag{27}$$

where every mean value function has a list of parameters that refer to a specific term illustrated in Table 5 [25].

Table 5. Parameters' terms of Mean value functions [25].

Parameter	Term
t	Calendar time.
m, m(t)	Expected number of removed faults in time interval (0,1].
N, N(t)	The total number of users in time interval (0,1]. Calculated by the equation: - $N(t) = N' \frac{1 - e^{-(p+q)t}}{1 + \frac{q}{p} e^{-(p+q)t}}$
a	Initial number of faults in a software.
b	Removal rate for faults.
p	coefficient of innovation.
q	coefficient of imitation.
N'	Total potential software's users' number.
α	Proportion of error generation.

5. Experimental Results and Analysis

In this section, we present the employed dataset, hardware and software used, evaluation criteria, and experimental results.

5.1. Datasets

To check the efficiency of the selected models, we use three versions of GNU's Not Unix Network Object Model Environment (GNOME) OSS projects called GNOME 2.0, GNOME 2.2, and GNOME 2.4 respectively as illustrated in Table 6 [28]. Each version has two columns, one for the time in weeks from releasing each project and the second one for the detected faults in each week.

Table 6. Official public releases detected faults for GNOME [28].

GNOME 2.0		GNOME 2.2		GNOME 2.4	
Weeks from release	Detected bugs	Weeks from release	Detected bugs	Weeks from release	Detected bugs
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				

5.2. *Hardware and Software*

A. *Hardware*

We use a computer device with the following specification: 1) processor: Intel(R) Core (TM)i5-6200U 2) installed memory (RAM): 8.00 Gigabyte.

B. *Software*

We install windows 10 Pro as an operating system also MATLAB version R2017b.

5.3. *Evaluation Criteria*

There are four comparison criteria for the implemented models, namely, Sum of Square Error (SSE), Mean Square Error (MSE), R-square (R2), and reliability[29]. These criteria show how much the predictive and fitting performance of the selected models[29].

- SSE is estimated by the following equation [29]:

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

where M_y represents fault value that observed, and $M(T_{aj})$ represents the total number of faults that expected by time t.

- MSE is estimated by the following equation [26]:

$$MSE = \frac{\sum_{j=1}^{k}\left(M_j - M(t_j)\right)^2}{k} \tag{29}$$

where Mj, and $M(t_j)$ as mentioned before in the equation of SSE, and k refers to the sample size.

- R^2 is estimated by the following equation [26]:

$$R^2 = R1 - \frac{\sum_{j=1}^{k}\left(M(t_j) - M_j\right)^2}{\sum_{j=1}^{k}\left(M_j \; - \sum_{j=1}^{k}(M_j)/k\right)^2} \tag{30}$$

where $M_{j,}$ $M(t_j)$, and k as mentioned before in the equation of MSE.

- Reliability is estimated by the following equation [30]:

$$Maximize\ R_i = \frac{M_i(t)}{a_i} \tag{31}$$

where R_i represents the reliability for i release, M_i represents mean value function for the model, and a_i represents initial number of faults in the software.

Least Squares Estimation (LSE) estimates the software reliability models' parameters and evaluates the fitting and predictive execution. If the values of SSE and MSE are small, the predictive execution of the models is good [29]. Furthermore, the value of R2 has specific range from 0 to 1 [31]. If the value of R2 is large, the predictive execution of the models is good [29].

5.4. *Results and analysis*

A. *Mean value functions' parameters estimations*

For all GNOME dataset versions, every model has its mean value function with parameters needed to be estimated accurately to calculate software's reliability according to the equation number (31). LSE estimates these parameters for all models for all GNOME dataset versions as illustrated in Table 7, Table 8, and Table 9. First column in every table estimates the number of initial faults in the software. This parameter has high effect in software's reliability according to equation number (31). For GNOME 2.0 dataset SRGM-4 has the least value of parameter (a) with little difference from the rest models, and for GNOME 2.2 dataset, GNOME 2.4 dataset SRGM-6 has the least value of parameter (a) with a noticeable difference from the rest models.

Table 7. All Parameters of models' estimation for GNOME 2.0 dataset

Model	a	b	N	p	q	α	c
SRGM1	93.999	0.0703	42.1875	0.0133	0.1411	-	-
SRGM2	89.9898	0.4279	37.3464	0.0028	0.1006	-0.0073	-
SRGM3	84.9934	1.0835	36.4863	0.0012	0.1079	-0.0032	-
SRGM4	84.1547	0.1278	51.2903	0.0070	0.1251	0.0272	-
SRGM5	85.8475	0.3206	61.2329	0.0027	0.1223	0.0090	75.0091
SRGM6	92.4121	0.0551	-	-	-	-	-

Table 8. All Parameters of models' estimation for GNOME 2.2 dataset

Model	a	b	N	p	q	α	c
SRGM1	93.9774	0.0098	84.9181	0.0690	0.1396	-	-
SRGM2	88.1189	0.0996	10.7125	0.0593	0.1273	-0.0123	-
SRGM3	92.6471	4.5875	69.9598	0.0002228	-0.0066	-1.5860	-
SRGM4	85.1207	0.0124	105.3229	0.0501	0.1344	-0.3048	-
SRGM5	84.1026	1.8507	3.6011	0.0254	0.2426	0.1025	29.9909
SRGM6	50.7522	0.1019	-	-	-	-	-

Table 9. All Parameters of models' estimation for GNOME 2.4 dataset

Model	a	b	N	p	q	α	c
SRGM1	93.9978	0.0224	36.8368	0.0487	0.1261	-	-
SRGM2	70.0151	0.1989	8.3471	0.0370	0.0868	-0.0069	-
SRGM3	78.8783	0.2725	9.9767	0.0217	0.0389	-0.0487	-
SRGM4	73.5267	0.0242	47.6860	0.0502	0.0899	0.1603	-
SRGM5	81.5043	0.6123	2.7647	0.0344	0.0784	-0.0515	74.7220
SRGM6	51.2277	0.0763	-	-	-	-	-

B. *Goodness of fit for all GNOME dataset versions*

Four evaluation metrics used to estimate the predicative performance for the models as illustrated in Table 10, Table 11, and Table 12. Each table contains the goodness of fit for specific GMOME version dataset as the following:

- GNOME 2.0 dataset:

SRGM-2 has the lowest SSE value and the highest R-Square value, so it has the highest predictive performance compared with the rest models because the estimated value for this model is the closest to the actual value. As for reliability value, SRGM-1 is the lowest one because this model uses the perfect debugging methodology. This methodology assumes an illogical assumption that there is no probability of introducing new faults during the debugging process, and SRGM-4 has the highest reliability value because it has the lowest estimated initial number of faults. Additionally, SRGM-4 uses the imperfect debugging methodology, which has the logical assumption that there is a probability of introducing new faults at least only one fault during the debugging process.

- GNOME 2.2 dataset:

SRGM-5 has the lowest SSE value and the highest R-Square value, so it has the highest predictive performance compared with the rest models because the estimated value for this model is the closest to the actual value. As for reliability value, SRGM-1 is the lowest one because this model uses the perfect debugging methodology. This methodology assumes an illogical assumption that there is no probability of introducing new faults during the debugging process, and SRGM-6 has the highest reliability value because it has the lowest estimated initial number of faults. Additionally, SRGM-6 uses Gompertz distribution methodology, which permits both decreasing or increasing failure rates according to the shape parameters.

- GNOME 2.4 dataset:

SRGM-4 has the lowest SSE value and the highest R-Square value, so it has the highest predictive performance compared with the rest models because the estimated value for this model is the closest to the actual value. As for reliability value, SRGM-1 is the lowest one because this model uses the perfect debugging methodology. This methodology assumes an illogical assumption that there is no probability of introducing new faults during the debugging process, and SRGM-6 has the highest reliability value because it has the lowest estimated initial number of faults. Additionally, SRGM-6 uses Gompertz distribution methodology, which permits both decreasing or increasing failure rates according to the shape parameters.

Table 10. All models Goodness of fit for GNOME 2.0 dataset

Model	SSE	MSE	R-Square	Reliability
SRGM1	101.199	5.622	0.993	0.8991
SRGM2	96.7272	5.6898	0.9937	0.9845
SRGM3	96.7439	5.6908	0.9936	0.9802
SRGM4	97.9777	5.7634	0.9936	0.9962
SRGM5	101.4119	6.3382	0.9933	0.9933
SRGM6	155.7741	7.4178	0.9898	0.9359

Table 11. All models Goodness of fit for GNOME 2.2 dataset

Model	SSE	MSE	R-Square	Reliability
SRGM1	45.3156	3.4858	0.9887	0.5567
SRGM2	44.8107	3.7342	0.9888	0.6754
SRGM3	56.1775	4.6815	0.9859	0.5704
SRGM4	44.6103	3.7175	0.9888	0.6158
SRGM5	42.3749	3.8523	0.9894	0.6197
SRGM6	70.1503	4.3844	0.9824	1.0000

Table 12. All models Goodness of fit for GNOME 2.4 dataset

Model	SSE	MSE	R-Square	Reliability
SRGM1	61.6485	4.4035	0.9861	0.5612
SRGM2	54.9020	4.2232	0.9876	0.8158
SRGM3	55.6580	4.2814	0.9874	0.6854
SRGM4	54.8757	4.2212	0.9876	0.7375
SRGM5	55.0842	4.5904	0.9876	0.6674
SRGM6	79.2473	4.6616	0.9821	1.0000

C. *Fitting figures for all GNOME dataset versions*

It is known that the more time increased, the more bugs detected, so all listed figures below show a direct correlation between time as x-axis and bugs as y-axis. These figures clarify how much the models' estimated values are close to the actual values in the dataset. Fig.3, and Fig.4 show fitting results for GNOME 2.0 dataset. Fig.5, and Fig. 6 show fitting results for GNOME 2.2 dataset. Fig.7 shows fitting results for GNOME 2.4 dataset.

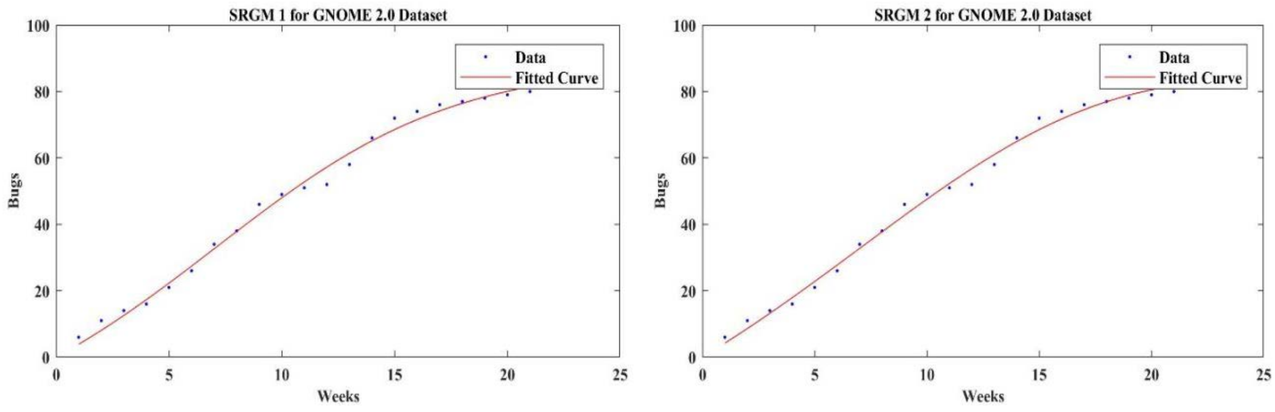


Fig.3. Fitting results for GNOME2.0 dataset part 1

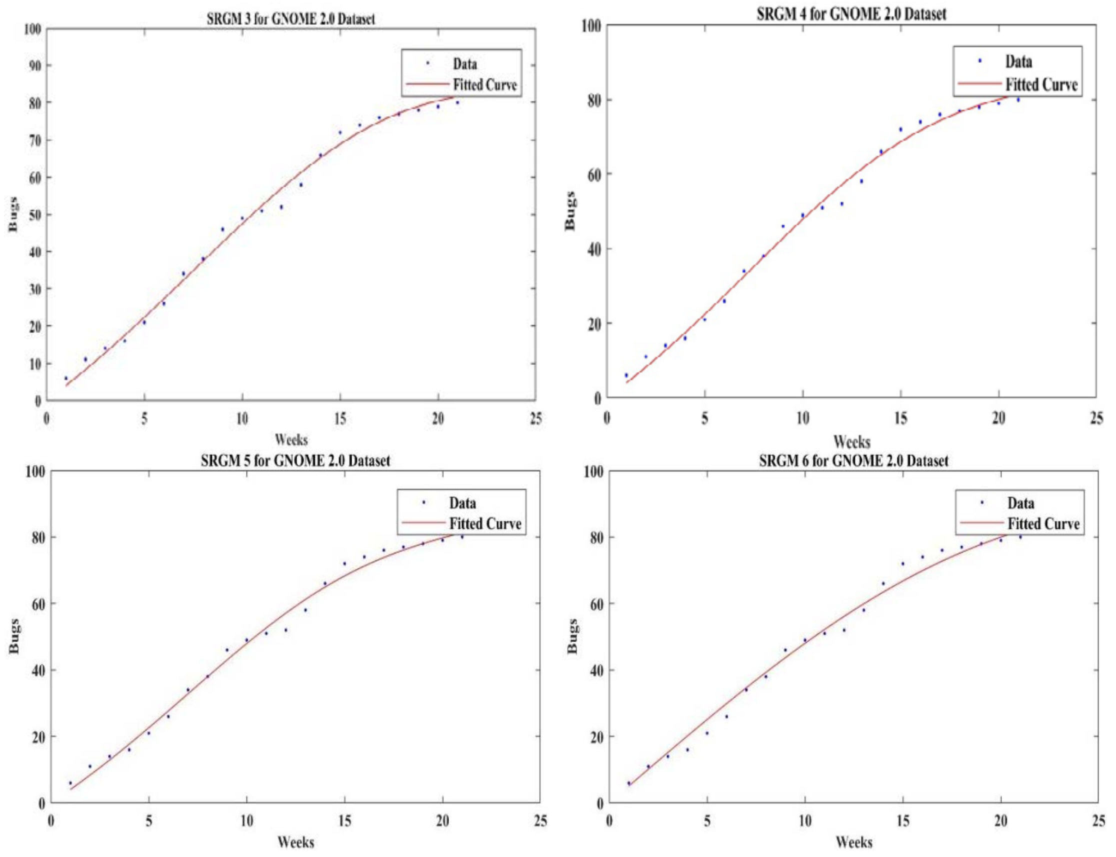


Fig.4. Fitting results for GNOME2.0 dataset part 2

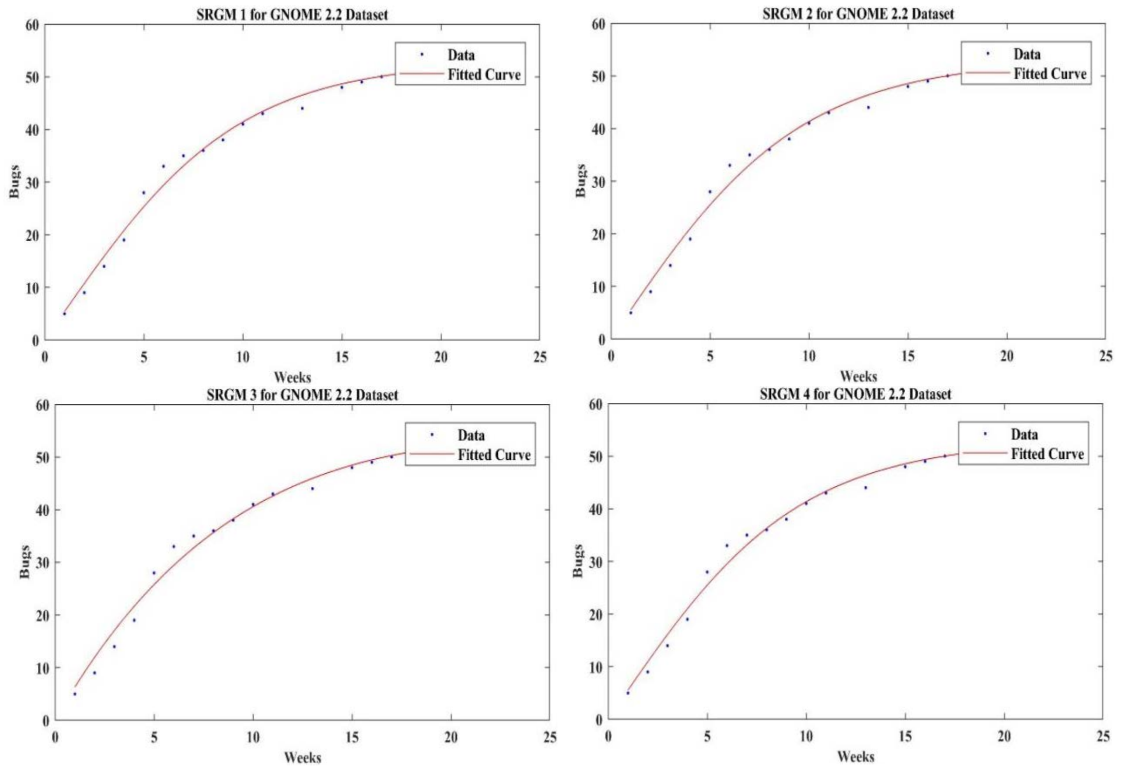


Fig.5. Fitting results for GNOME2.2 dataset part 1

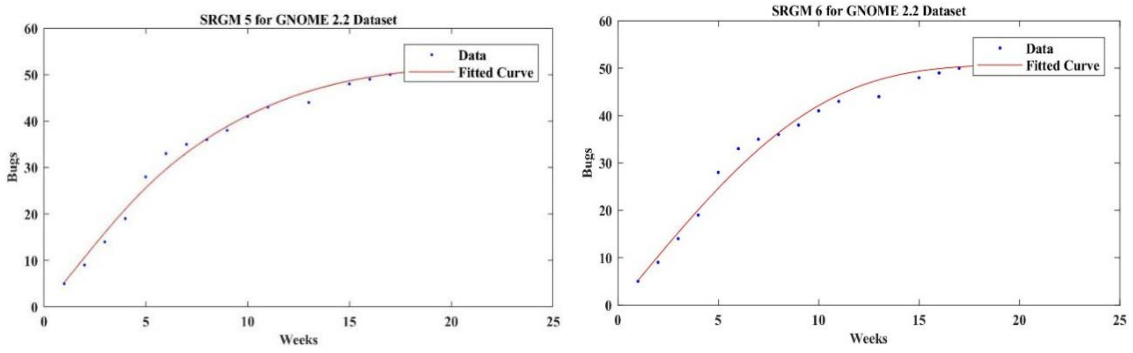


Fig.6. Fitting results for GNOME2.2 dataset part 2

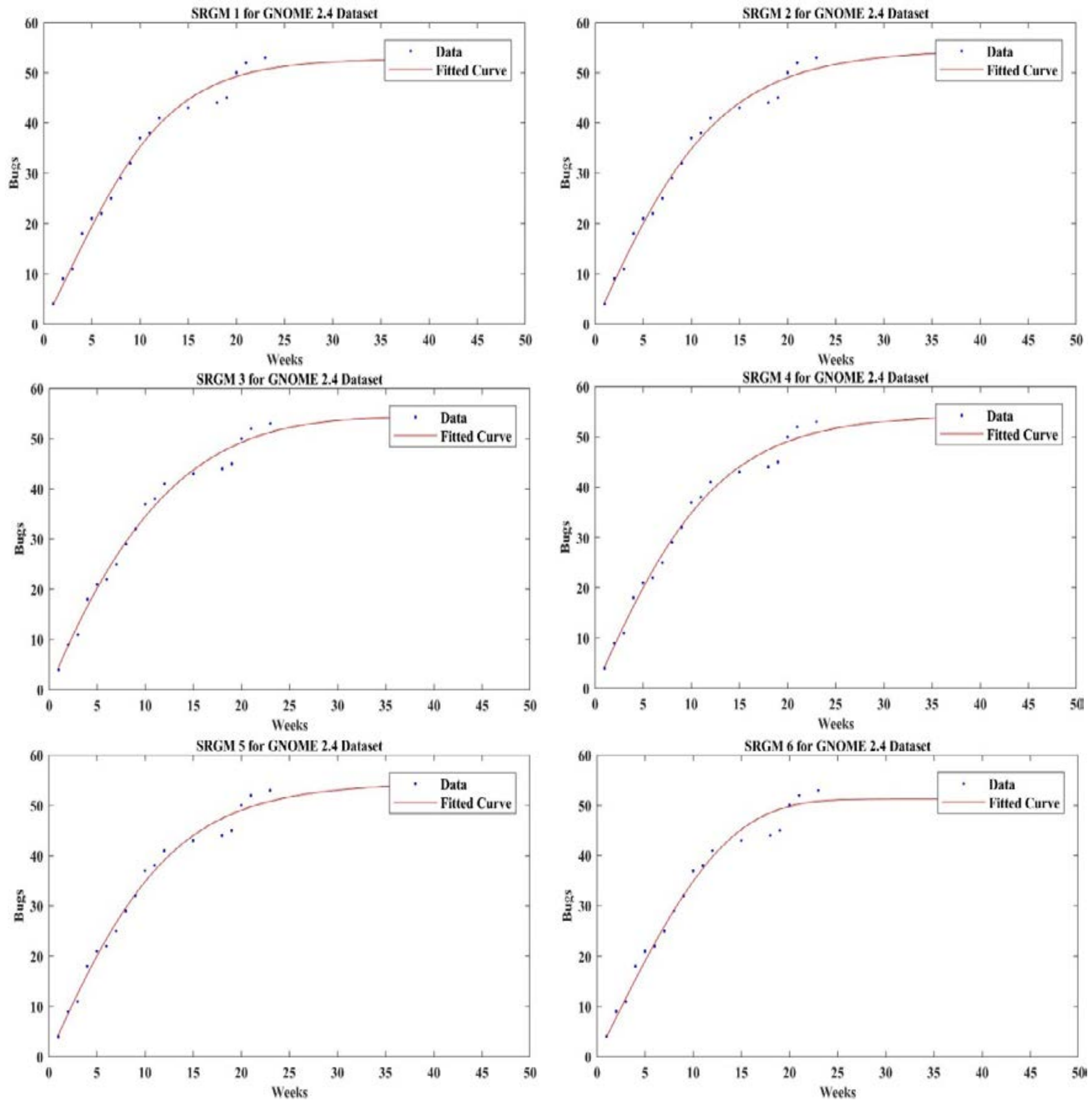


Fig.7. Fitting results for GNOME2.4 dataset

6. Conclusion and Future Work

Our research focused on evaluating the Halstead model and McCabe's Cyclomatic Complexity model. There are two examples of deterministic models. They have metrics used to calculate the degree of complexity in the code. On the other hand, we assessed the reliability of GNOME open-source software using six probabilistic models based on three different methodologies. SRGM-1 is based on perfect debugging. From SRGM-2 to SRGM5 are based on imperfect debugging. SRGM-6 is based on Gompertz distribution. LSE was used to estimate software reliability models' parameters and made comparison criteria between these models by SSE, MSE, R-Square, and reliability. Both SRGM-2, SRGM-5, and SRGM-4 have the highest performance compared with the other models. SRGM-1 has the lowest reliability value for all three versions of the GNOME dataset. SRGM-4 has the highest reliability value for GNOME 2.0, and SRGM-6 has the highest reliability value for GNOME 2.2 and GNOME 2.4. We can say that SRGM-6 is the most optimal model for reliability assessment. In the future, we will be extended this work with different optimization techniques to estimate mean value functions' parameters accurately with fewer values for SSE and MSE.

References

[1] 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.

[2] F. Akowuah, J. Lake, X. Yuan, E. Nuakoh, H. Yu,"Testing the security vulnerabilities of openemr 4.1. 1: a case study,"Journal of Computing Sciences in Colleges, vol. 30, no. 3, pp. 26-35,2015.

[3] W. Yu and L. Chen, "The Application of Computer Softwares in Chemistry Teaching," Int. J. Educ. Manag. Eng., vol. 2, no. 12, pp. 73–77, 2012.

[4] R.K. Behera, S.K. Rath, S. Misra, M. Leon, A. Adewumi," Machine learning approach for reliability assessment of open-source software," in International Conference on Computational Science and Its Applications, pp.472-482,2019.

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

[6] M. Sirshar, A. Ali, S. Ibrahim, "A Comparitive Analysis Between Open Source and Closed Source Software in Terms of Complexity and Quality Factors," Dec 2019.

[7] RiskSense, "The Dark Reality of Open Source," 2020, [Online]. Available: https://cdn2.hubspot.net/hubfs/5840026/Q2-2020 Open-Source Spotlight Report/Spotlight_OpenSource.pdf.

[8] D. K. Moulla, A. Abran, and Kolyang, "Duration Estimation Models for Open-Source Software Projects," Int. J. Inf. Technol. Comput. Sci., vol. 13, no. 1, pp. 1–17, 2021.

[9] Kluitenberg, HF," Evaluating Quality of Open-Source Components,"2018.

[10] A. Alami, Y. Dittrich, A. Wasowski, "Influencers of quality assurance in an open-source community," in 2018 IEEE/ACM 11th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE), pp. 61–68,2018.

[11] A. Al Hussein, "An Object-Oriented Software Metric Tool to Evaluate the Quality of Open-Source Software," IJCSNS Int. J. Comput. Sci. Netw. Secur., vol. 17, no. 4, pp. 345–351, 2017.

[12] 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, pp.699-704,2018.

[13] L. V. Utkin, F. P. A. Coolen, "A robust weighted SVR-based software reliability growth model," Reliability Engineering \& System Safety, vol. 176, pp. 93–101, 2018.

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

[15] N. Shakhovska, V. Yakovyna, N. Kryvinska," An improved software defect prediction algorithm using self-organizing maps combined with hierarchical clustering and data preprocessing," in International Conference on Database and Expert Systems Applications, pp.414-424,2020.

[16] P. Kumar, L. K. Singh, 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.

[17] N. Govil, "Programming Languages for Analyzing Software Complexity," in 2020 4th International Conference on Trends in Electronics and Informatics (ICOEI) (48184), pp. 939–943, 2020.

[18] T. Hariprasad, G. Vidhyagaran, K. Seenu, C. Thirumalai, "Software complexity analysis using halstead metrics,"in 2017 International Conference on Trends in Electronics and Informatics (ICEI), pp. 1109–1113, 2017.

[19] A.S. Abdulkareem, J.A. Abboud," Evaluating Python, C++, JavaScript, and Java Programming Languages Based on Software Complexity Calculator (Halstead Metrics)," in IOP Conference Series: Materials Science and Engineering, vol. 1076, no. 1, pp.012046,2021

[20] H. Liu, X. Gong, L. Liao, B. Li, "Evaluate How Cyclomatic Complexity Changes in the Context of Software Evolution," in 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC), vol. 2, pp. 756–761, 2018.

[21] M. Zhu, 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] G.A. Aggarwal," Multi Release Reliability Growth Modeling for Open-Source Software Under Imperfect Debugging," in System Performance and Management Analytics, pp.77-86,2019.

[23] M. Dorin, "Coding for inspections and reviews," inProceedings of the 19th International Conference on Agile Software Development: Companion, pp.1-3, 2018.

[24] Y. Tashtoush, M. Al-maolegi, B. Arkok, "The Correlation among Software Complexity Metrics with Case Study," International Journal of Advanced Computer Research, vol. 4, no. 2, pp. 414, 2014.

[25] N. Gandhi, N. Gondwal, A. Tandon, "Reliability Modeling of OSS Systems based on Innovation-Diffusion Theory and Imperfect Debugging,"in ICITKM, pp. 53–58, 2017.

[26] A. El-Gohary, A. Alshamrani, and A. N. Al-Otaibi, "The generalized Gompertz distribution," Appl. Math. Model., vol. 37, no. 1–2, pp. 13–24, 2013, doi: 10.1016/j.apm.2011.05.017.

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

[28] X. Li, Y. F. Li, M. Xie, 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.

[29] 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.

[30] B. Pachauri, A. Kumar, and J. Dhar, “Reliability analysis of open-source software systems considering the effect of previously released version,” Int. J. Comput. Appl., vol. 41, no. 1, pp. 30–37, 2019, doi: 10.1080/1206212X.2018.1497575.

[31] Javaid Iqbal,"Analysis of Some Software Reliability Growth Models with Learning Effects", International Journal of Mathematical Sciences and Computing, Vol.2, No.3, pp.58-70, 2016.

Authors’ Profiles

Islam Saied: received his BSc. in Computer Science from Menoufia University, Faculty of Computers, and Information in 2012. His research interest includes Software Engineering, Image Processing, Machine Learning, and Security.

Hany M. Harb: received the BSc. and MSc. in computer engineering from Ain Shams University, Faculty of engineering in 1978 and 1981, respectively and received Ph.D. in Distributed Systems from IIT, USA in 1986, MSOR from IIT, USA in 1987. His research Interest includes Software engineering, Distributed Systems, Computer Networks, Machine learning, Artificial Intelligence and Expert Systems, Neural Nets, Data Mining and Warehousing, Semantic Web and Semantic Web Services, Cloud Computing.

Hamdy M. Mousa: received the BSc. and MSc. in Electronic Engineering and Automatic control and measurements from Menoufia University, Faculty of Electronic Engineering in 1991 and 2002, respectively and received his Ph.D. in Automatic control and measurements Engineering (Artificial intelligent) from Menoufia University, Faculty of Electronic Engineering in 2007. His research interest includes intelligent systems, Natural Language Processing, privacy, security, embedded systems, GSP applications, intelligent agent, Bioinformatics, Robotics.

Mohammed G. Malhat: received his BSc., MSc., and PhD in Computer Science from Menoufia University, Faculty of Computers, and Information in 2010, 2014, and 2019, respectively. His research interest includes Distributed Systems, Data Mining, Machine Learning, Data Privacy, and Security, Cloud Computing, intelligent agent, Software engineering.

How to cite this paper: Islam S. Ramadan, Hany M. Harb, Hamdy M. Mousa, Mohammed G. Malhat, "Reliability Assessment for Open-Source Software Using Deterministic and Probabilistic Models", International Journal of Information Technology and Computer Science(IJITCS), Vol.14, No.3, pp.1-15, 2022. DOI: 10.5815/ijitcs.2022.03.01