物件導向軟體可靠度評估工具：CARATS*

Object-Oriented Software Reliability Assessment Tool: CARATS

陳建嘉，黃瀚萱，黃士維，林楚迪，黃慶育
國立清華大學資訊工程學系
Chien-Chia Chen, Hen-Hsen Huang, Shih-Wei Huang, Chu-Ti Lin, and Chin-Yu Huang
Department of Computer Science
National Tsing Hua University
Hsinchu, Taiwan

Abstract

With the growth of software project scale, how to deliver reliable software products on time becomes a critical issue. Although many related software reliability theories have been proposed in the past few decades, most of software reliability analysis processes still depend on the powerful computations of the general-purposed numerical software. Hence, to develop a user-friendly special-purposed software reliability assessment tool is extremely meaningful for both research and business application. For these reasons, this paper presents the design and implement of **C**omputer-**A**ided **R**eliability **A**ssessment **T**ool for **S**oftware (CARATS). CARATS is an object-oriented software reliability assessment tool using software reliability growth models (SRGMs) with period failure count data and neural networks to assessment the software reliability. Due to the characteristics of the special-purposed and object-oriented design, CARATS can analyze the software reliability easily. Besides, it is more flexible to adopt different SRGMs than traditional tools.

1. Introduction

The techniques of software reliability analysis developed based on hardware reliability incipiently. As the significance of software grows rapidly, software reliability gets more and more attentions. In the past few decades, many software reliability modeling theories have been proposed, which can be classified into two main types: the deterministic model and the probabilistic model [1].

The deterministic model assesses software reliability by analyzing the program texture, such as the number of distinct operators and operands and the number of assembly instructions in a program. This type of model does not involve any random event. On the other hand, the probabilistic model treats the failure occurrences and removals as probabilistic event. This type of model can be classified into different groups [1]:

- error seeding
- failure rate
- curve fitting
- reliability growth
- nonhomogeneous Poisson process(NHPP)
- Markov structure

In this paper, we focus on NHPP probabilistic model only. The software reliability growth theory is based on the different characteristics between hardware and software since software will become more reliable after appropriate testing and debugging phase. Hence, we can describe the historical failure data gathered from the testing phase by NHPP models, and these models can represent the software reliability growth pattern.

Modern methods used to estimate the cumulative number of failures occurred up to a specific time must rely on numerical analysis software mostly. However, either operation or execution of general-purposed numerical analysis software is very inconvenient to analyze software reliability systematically since those software tools must be compatible with general numerical problems.

In this research, we implement a special-purposed assessment tool, **C**omputer-**A**ided **R**eliability **A**ssessment **T**ool for **S**oftware (CARATS), based on SRGM. CARATS integrates several most popular SRGMs, such as GO model, delay S-shaped model, Rayleigh model, power model, inflection S-shaped model, and so on [1-8]. By inputting failure data in plain text format to this tool, CARATS will systematically estimate parameters of selected SRGMs automatically. Both software reliability diagrams and numerical data will be shown based on the estimates.

In addition to the use of SRGMs, we show another novel prediction method—prediction of software reliability by using neural networks, which has been widely used in many fields, such as machine learning, stock prediction, and adaptive filters. Actually, neural networks can be treated as a black box, that is, neural networks can learn to fit almost any periodical curves by giving enough training data. The concept of predicting software reliability by using neural networks has been proposed many years ago, but no advanced applications were presented yet. Hence, we integrate neural networks into this assessment tool.

In the rest of this paper, five SRGMs and two parameter evaluation methods are briefly outlined in

* 本研究接受國科會編號：NSC94-2815-C-007-028-E 研究計畫經費補助

Section 2. In addition, Section 2 also shows how to predict software reliability by using neural networks. The implementation details of CARATS are shown in Section 3. Besides, Section 4 presents a set of real software reliability assessment results. Finally, some conclusions are given in Section 5.

2. Background

In this section, we briefly go through the technical backgrounds of CARATS. We only mention the fundamental theories of our implementations very shortly.

2.1 Software Reliability Growth Model

We have mentioned that the concept of software reliability comes from the concept of hardware reliability. Compared with software, the physical characteristics make the reliability of hardware decrease gradually with time. Eventually, it is economically impractical or too unreliable to continue in service. However, software cannot be treated as hardware because software does not wear out as time goes by. In other words, software reliability grows with the proceeding of testing and debugging.

The process of estimating the reliability of specific software through SRGMs consists of: (i) gathering historical failure data in the testing phase, and (ii) evaluating suitable value of selected SRGMs parameters based on the given failure data.

Traditionally, there are two common types of failure data: time-domain and interval-domain data. The time-domain data involve the individual times for each occurred failure or the times between two succeeded failures, so we also call this kind of failure data "time between failure (TBF)" data format. The interval-domain data count the cumulative number of failures occurred in a fixed period. Hence, we call this kind of failure data "period failure count (PFC)" data format.

2.1.1 Goel-Okumoto NHPP Model

The Goel-Okumoto NHPP model is known as GO model, which was proposed by Goel and Okumoto in 1979 [1-5]. The GO model is characterized by the following mean value function:

$$m(t) = N\,\{1 - \exp(-\varphi t)\}\,, \tag{1}$$

where N is the number of initial faults in the software, and φ is the fault detection rate.

2.1.2 Delayed S-Shaped Model

The delayed S-shaped model was proposed by Yamada in 1984 [1, 3-6]. This model is characterized by the following mean value function:

$$m(t) = N\,\{1 - (1 + \rho\, t)\exp(-\rho\, t)\}\,, \tag{2}$$

where N is the number of initial faults in the software, and ρ is the fault removal (failure detection and fault isolation) rate parameter.

2.1.3 Rayleigh Model

The Rayleigh model is a member of the family of the Weibull distribution [7] and is characterized by the following mean value function:

$$m(t) = N\,\{1 - \exp(-\lambda t^2)\}\,, \tag{3}$$

where N is the number of initial faults in the software, and λ is the failure rate parameter.

2.1.4 Power Model

The power model was proposed by Crow in 1974 [8]. This model is characterized by the following mean value function:

$$m(t) = N \cdot t^{\lambda}\,, \tag{4}$$

where N is the scale parameter that can be treated as the number of failures in the software, and λ is the shape parameter.

2.1.5 Inflection S-Shaped Model

The inflection S-shaped model was proposed by Ohba in 1984 [1, 3-6]. This model is characterized by the following mean value function:

$$m(t) = N\,\frac{1 - \exp(-\varphi\, t)}{1 + \psi \cdot \exp(-\varphi\, t)}\,, \tag{5}$$

where N is the number of initial faults in the software, φ is the failure detection rate, and ψ is the inflection parameter.

2.2 Parameters Evaluation

In Section 2.1, we introduced several popular SRGMs used in CARATS. After formulating these mathematical models, we still have to determine the parameters of each model. There are two famous methods to determine parameters: least squares estimation (LSE) and maximum likelihood estimation (MLE) [1, 4-5, 9-12].

We can evaluate suitable parameters of selected SRGM by minimizing the least square sum as the following:

$$S = \sum_{k=1}^{n} [m_k - m(t_k)]^2\,, \tag{6}$$

where m_k is the cumulative number of failures consumed in time $(0,\, t_k]$, and $m(t_k)$ is the cumulative number of failures estimated by the given model.

Compared with LSE method, MLE method is much more complex. The likelihood function is defined as the following:

$$L = \prod_{k=1}^{n} \frac{[m(t_k) - m(t_{k-1})]^{(m_k - m_{k-1})}}{(m_k - m_{k-1})!} \cdot \exp[-(m(t_k) - m(t_{k-1}))] \tag{7}$$

where m_k is the cumulative number of failures observed in $(0,\, t_k]$, and $m(t_k)$ is the cumulative number of failures estimated by the given model. Taking the logarithm of the likelihood function in (7), we have

$$log(L) = \sum_{k=1}^{n} (m_k - m_{k-1}) \cdot \log[m(t_k) - m(t_{k-1})]$$

$$- \sum_{k=1}^{n} [m(t_k) - m(t_{k-1})] - \sum_{k=1}^{n} \log[(m_k - m_{k-1})!]. \quad (8)$$

By replacing $m(t_k)$ with the selected model formula and solving the partial differential equations, we can determine the value of each parameter.

However, these two methods discussed above are not suitable for our object-oriented design due to the lack of scalability, so we can also use a model independent alternative, which will be shown in Section 2.3.

2.3 Prediction of Software Reliability Using NN

In this section, we present a parameter-free way to predict software reliability. Hence, we do not have to spend extra computing cost on determining the value of parameters.

Neural networks are constructed by variable number of neurons, and each one has its bias and weight. In the training process, neurons adjust their biases and weights to reach a given goal, and the final outputs are evaluated by a specific activation function, which is to limit the amplitude of output of a neuron [13]. After training with representative historical data, neural networks can predict the software reliability growth model.

3. System Description

The UML class diagram of CARATS is shown in Fig. 1, which consists of five sub-diagrams.

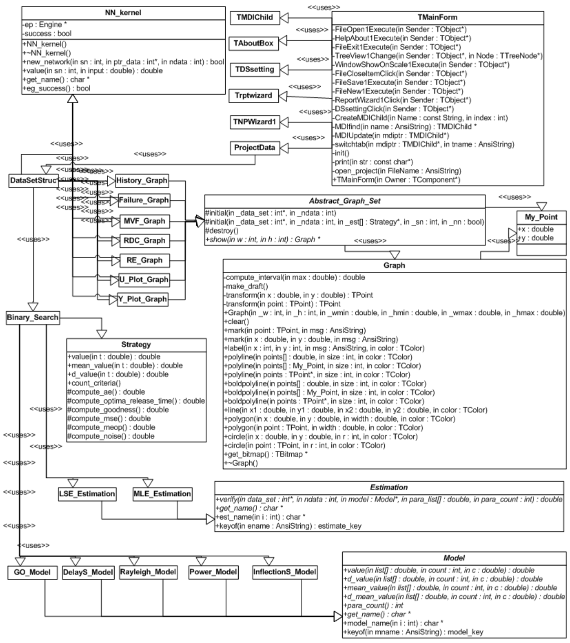


Fig. 1 CARATS UML class diagram.

3.1 Model Abstraction Module

This module abstracts software reliability growth models. By extending this module, CARATS can adopt different SRGMs easily, which shows its flexibility. The related UML class diagram of this module is shown in Fig. 2.

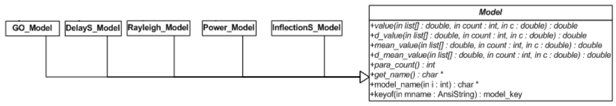


Fig. 2 UML diagram for model abstraction module.

3.2 Parameters Estimation Module

This module is responsible for the parameters estimation. In order to adopt newly extended SRGMs, we use numerical method to find out the answer instead of calculating and solving the equations case by case. Fig. 3 shows the related UML class diagram of this module

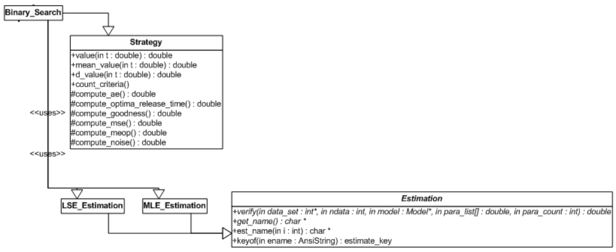


Fig. 3 UML diagram for parameters estimation module.

3.3 Neural Networks Prediction Module

In CARATS, we treat neural networks as a kind of model. First, we create and train the neural networks based on given PFC data, and then neural networks return the estimated number of failures by time t after training. Fig. 4 illustrates the related UML class diagram of this module.

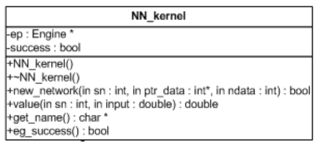


Fig. 4 UML diagram for NN prediction module.

3.4 Graph Abstraction Module

This module is the data visualizing routine, i.e., after fitting curves, this module will translate the numerical data into graphical data to enhance readability. The related UML class diagram of this module is given in Fig. 5.

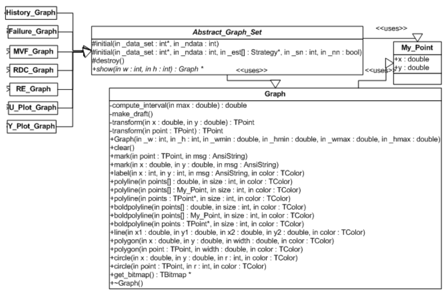


Fig. 5 UML diagram for graph abstraction module.

3.5 User Interface and Internal Data Structure

A well-designed interface improves usability. These graphical user interfaces help user to communicate with the internal data structure easier. Besides, internal data structure includes some routines that relay the requests to lower layer in order to execute real calculations. Fig. 6 shows the related UML class diagram of this module.

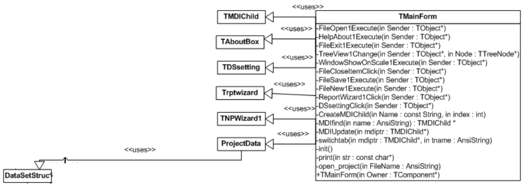


Fig. 6 UML diagram for user interface and internal data structure.

4. Function Description and Results

Figures 7-12 show some execution screenshots and simulation results of CARATS.

4.1 New Project Creation

CARATS supports multiple data sets in a project and each can have different analysis strategies, such as predicting by different models or evaluating model parameters by different estimation methods. Beginners can perform this tool by following the project creation wizard of CARATS. Fig. 7 shows three of seven steps in the project creation wizard.

4.2 Overall Picture

Users can select the desired diagram from the treeview in the left side as shown in Fig. 8.

4.3 Viewing and Adjusting Data Set Settings

After loading data sets and finishing the initial estimation, users can adjust settings by means of the estimation results and read the brief report from the settings dialog as shown in Fig. 9. The "Iterations" in the Fig. 9(c) is the number of iterations that CARATS spent before models being converged. And the rest five are some criteria to evaluate the model performance [1, 9, 10, 12, 14]. Besides, users can also get the optimal release time based on the desired release criteria [9-12, 15].

4.4 Simulation Results

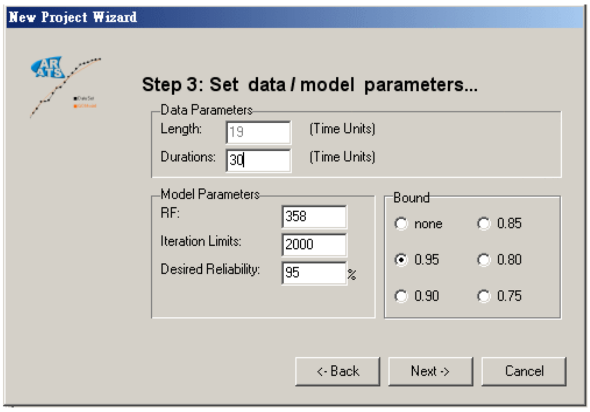
Fig. 10 shows the cumulative number of failures of Ohba's data set. Fig. 11 shows the performance comparisons based on the parameters estimated by LSE. Each simulation result has its own diagram independently. However, we only consider figures for LSE as illustrations due to the limitation of space.

4.5 Report Generator

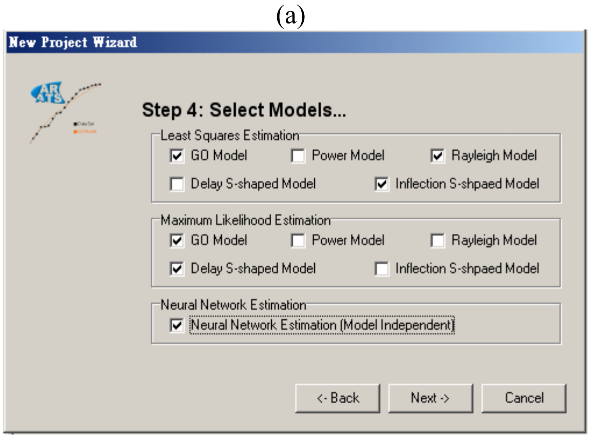
CARATS can generate a very detailed reliability analysis report as shown in Fig. 12, which contains all of the estimation results and figures. Weekly estimations report can be used to verify the reliability growth. All of these report items can be customized.

5. Conclusions

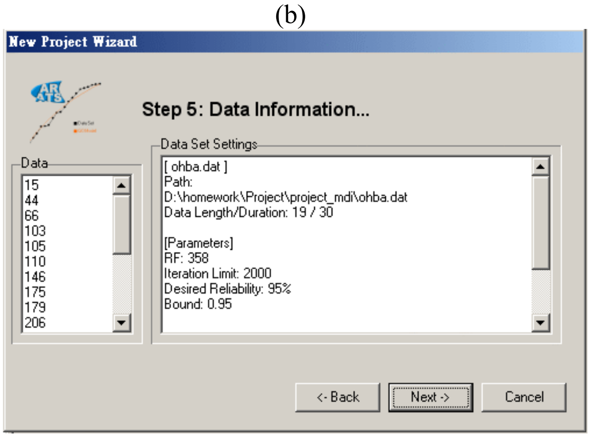
In this paper, we present a user-friendly software reliability assessment tool, CARATS, which can be used to measure a software product through its development process. CARATS is expected to have widely impact due to its abundant functionality. It supports several famous SRGMs and introduces a rarely implemented predicting technology, neural networks. Furthermore, CARATS can also suggest optimal release time based on the desired release criteria. In early days, to analyze and predict software reliability is difficult since integrated assessment tools are very rare. Nowadays, CARATS provides a much more elegant solution for systematic software reliability prediction.



(a)



(b)



(c)

Fig. 7 Project Creation Wizard

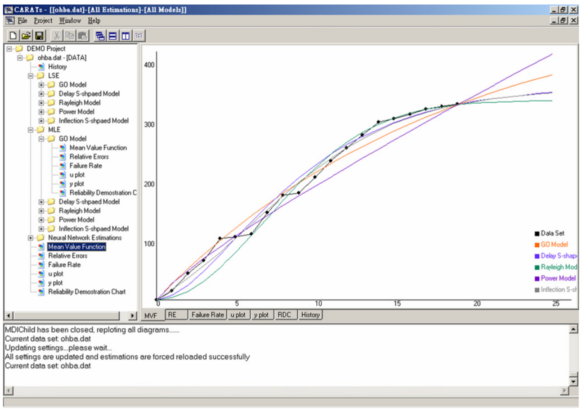
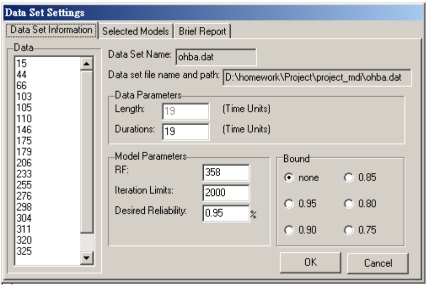
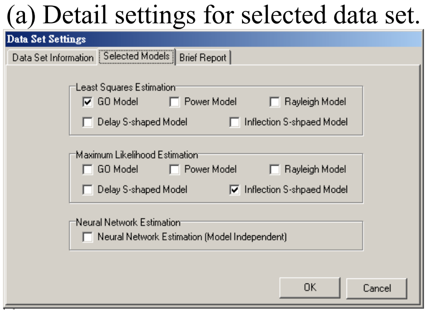


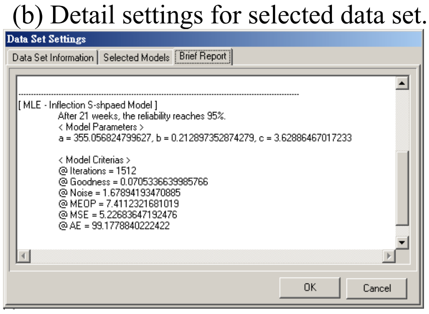
Fig. 8 Execution screenshot.



(a) Detail settings for selected data set.



(b) Detail settings for selected data set.



(c) Brief reports for selected data set.

Fig. 9 Viewing and adjusting settings.

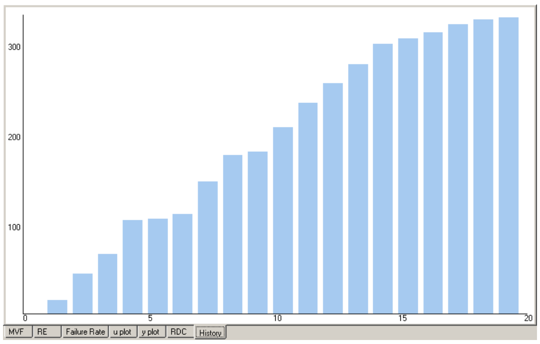
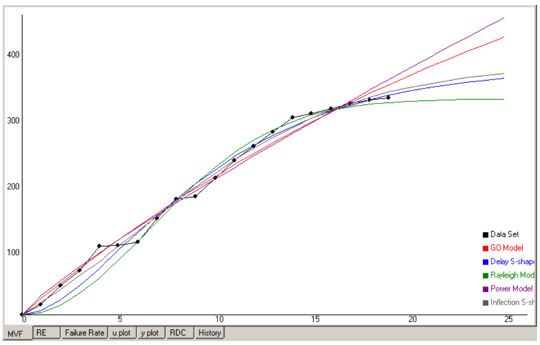
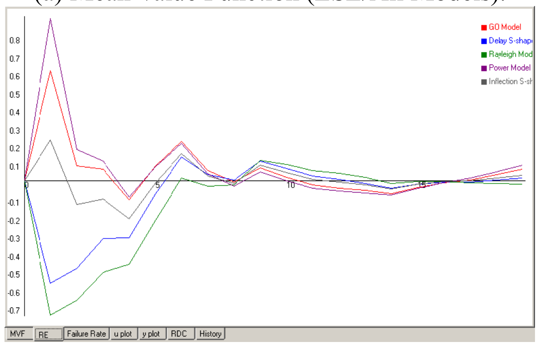


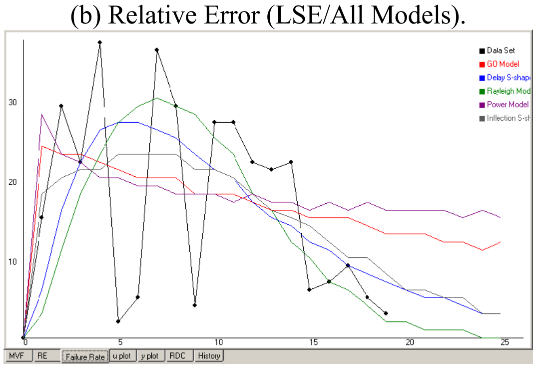
Fig. 10 Cumulative number of failures for Ohba's data set.



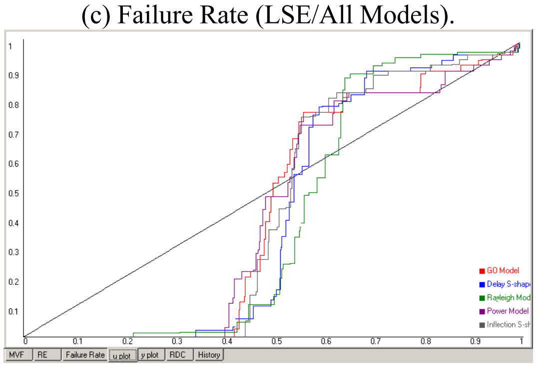
(a) Mean Value Function (LSE/All Models).



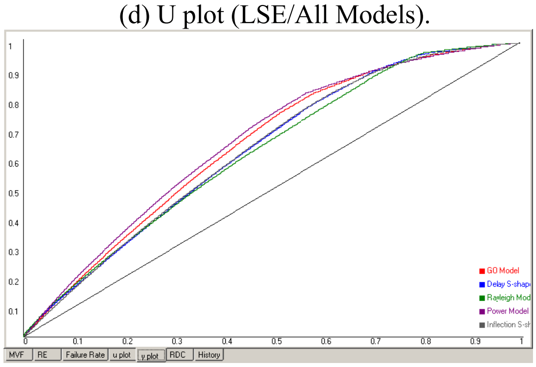
(b) Relative Error (LSE/All Models).



(c) Failure Rate (LSE/All Models).

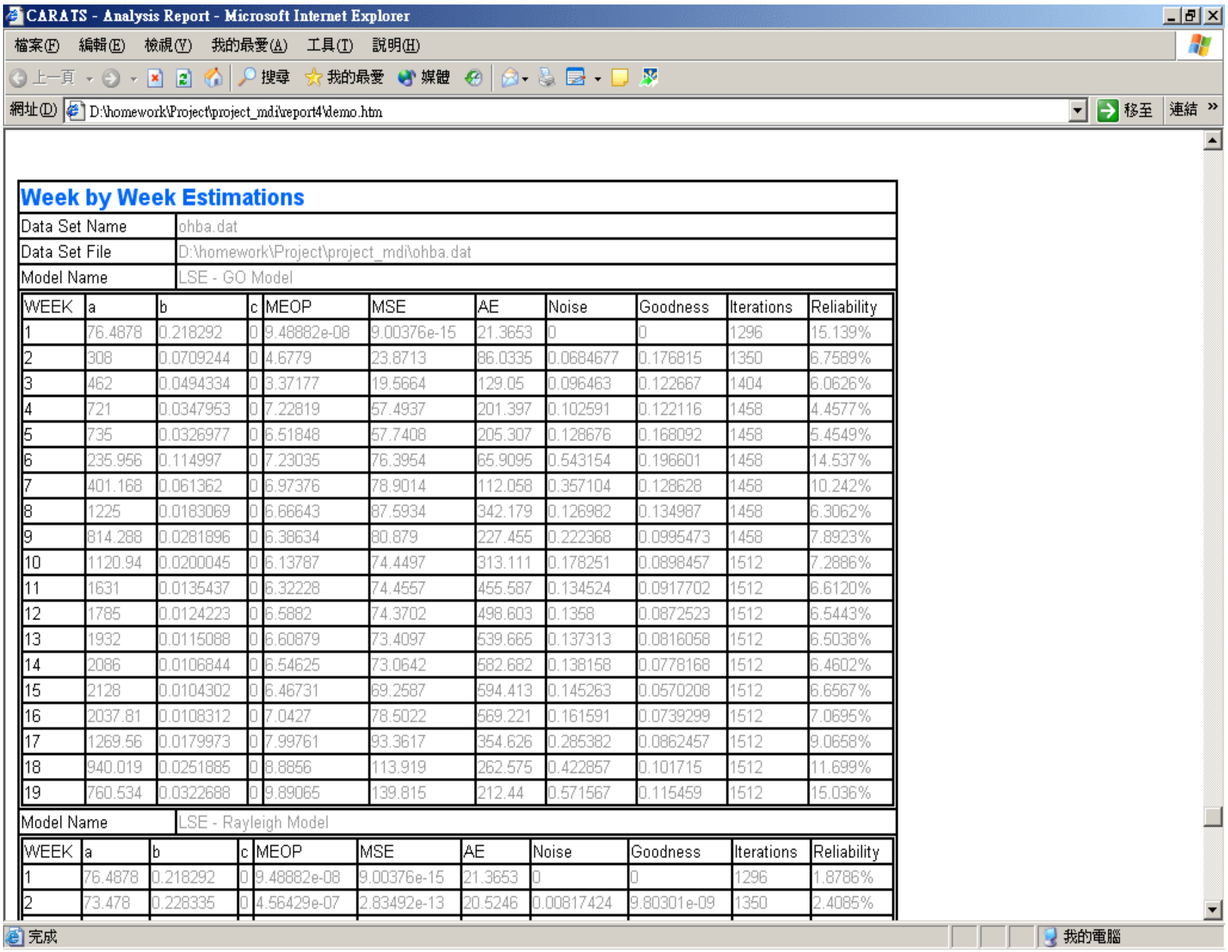


(d) U plot (LSE/All Models).



(e) Y plot (LSE/All Models).

Fig. 11 Simulation results for Ohba's data set.



Week by Week Estimations

Data Set Name	ohba.dat
Data Set File	D:\homework\Project\project_mdi\ohba.dat
Model Name	LSE - GO Model

WEEK	a	b	c	MEOP	MSE	AE	Noise	Goodness	Iterations	Reliability
1	76.4878	0.218292	0	9.48882e-08	9.00376e-15	21.3653	0	0	1296	15.139%
2	308	0.0709244	0	4.6779	23.8713	86.0335	0.0684677	0.176815	1350	6.7589%
3	462	0.0494334	0	3.37177	19.5664	129.05	0.096463	0.122667	1404	6.0626%
4	721	0.0347953	0	7.22819	57.4937	201.397	0.102591	0.122116	1458	4.4577%
5	735	0.0326977	0	6.51848	57.7408	205.307	0.128676	0.168092	1458	5.4549%
6	235.956	0.114997	0	7.23035	76.3954	65.9095	0.543154	0.196601	1458	14.537%
7	401.168	0.061362	0	6.97376	78.9014	112.058	0.357104	0.128628	1458	10.242%
8	1225	0.0183069	0	6.66643	87.5934	342.179	0.126982	0.134987	1458	6.3062%
9	814.288	0.0281896	0	6.38634	80.879	227.455	0.222368	0.0995473	1458	7.8923%
10	1120.94	0.0200045	0	6.13787	74.4497	313.111	0.178251	0.0898457	1512	7.2886%
11	1631	0.0135437	0	6.32228	74.4557	455.587	0.134524	0.0917702	1512	6.6120%
12	1785	0.0124223	0	6.5882	74.3702	498.603	0.1358	0.0872523	1512	6.5443%
13	1932	0.0115088	0	6.60879	73.4097	539.665	0.137313	0.0816058	1512	6.5038%
14	2086	0.0106844	0	6.54625	73.0642	582.682	0.138158	0.0778168	1512	6.4602%
15	2128	0.0104302	0	6.46731	69.2587	594.413	0.145263	0.0570208	1512	6.6567%
16	2037.81	0.0108312	0	7.0427	78.5022	569.221	0.161591	0.0739299	1512	7.0695%
17	1269.56	0.0179973	0	7.99761	93.3617	354.626	0.285382	0.0862457	1512	9.0658%
18	940.019	0.0251885	0	8.8856	113.919	262.575	0.422857	0.101715	1512	11.699%
19	760.534	0.0322688	0	9.89065	139.815	212.44	0.571567	0.115459	1512	15.036%

Model Name	LSE - Rayleigh Model

WEEK	a	b	c	MEOP	MSE	AE	Noise	Goodness	Iterations	Reliability
1	76.4878	0.218292	0	9.48882e-08	9.00376e-15	21.3653	0	0	1296	1.8786%
2	73.478	0.228335	0	4.56429e-07	2.83492e-13	20.5246	0.00817424	9.80301e-09	1350	2.4085%

Fig. 12 Weekly estimations report.

References

[1] H. Pham, *Software Reliability*: Springer-Verlag, 2000.

[2] A. L. Goel and K. Okumoto, "Time-Dependent Error Detection Rate Model for Software Reliability and Other Performance Measures," *IEEE Trans. Reliability* R-28, 206-211 (1979).

[3] P. N. Misra, "Software Reliability Analysis," *IBM Systems Journal*, Vol. 22, No. 3, pp. 262-270, 1983.

[4] M. Xie, *Software Reliability Modeling*: World Scientific Publishing Company, 1991.

[5] C. Y. Huang, M. R. Lyu, and S. Y. Kuo, "A Unified Scheme of Some Non-Homogenous Poisson Process Models for Software Reliability Estimation," *IEEE Trans. on Software Engineering*, Vol. 29, No. 3, pp. 261-269, Mar. 2003.

[6] M. Ohba, "Software Reliability Analysis Models," *IBM J. Research & Development*, Vol. 28, No. 4, pp. 428-443, Jul. 1984.

[7] S. H. Kan, *Metrics and Models in Software Quality Engineering*: Addison-Wesley, 1994.

[8] L. H. Crow: "Reliability Analysis for Complex, Repairable Systems," *Reliability and Biometry Statistical Analysis of Lifelength* pp. 379-410: SIAM, Philadelphia, 1974.

[9] M. R. Lyu, *Handbook of Software Reliability Engineering*: McGraw Hill, 1996.

[10] J. D. Musa, A. Iannino, and K. Okumoto, *Software Reliability, Measurement, Prediction and Application*: McGraw Hill, 1987.

[11] J. D. Musa, *Software Reliability Engineering: More Reliable Software, Faster Development and Testing*: McGraw Hill, 1999.

[12] C. Y. Huang, and S.Y. Kuo, "Analysis of Incorporating Logistic Testing-Effort Function Into Software Reliability Modeling", *IEEE Transactions on Reliability*, Vol. 51, No. 3, pp. 261-270, Sep. 2002.

[13] S. Haykin, *Neural Networks A Comprehensive Foundation 2nd Edition*: Prentice Hall, 1999.

[14] M. R. Lyu and A. Nikora, "Applying Software Reliability Models More Effectively," *IEEE Software*, pp. 43-52, Jul. 1992.

[15] C. Y. Huang and M. R. Lyu, "Optimal Release Time for Software Systems Considering Cost, Testing-Effort, and Test Efficiency," *IEEE Trans. on Reliability*, Vol. 54, No. 4, pp. 583-591, Dec. 2005.