A HEURISTIC APPROACH FOR SOFTWARE RELIABILITY PREDICTION: THE EQUALLY-WEIGHTED LINEAR COMBINATION MODEL

Michael R. Lyu
ECE Department
The University of Iowa
Iowa City, IA 52242

Allen Nikora
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, CA 91109

Abstract

This paper proposes a heuristic approach to addressing the software reliability modeling problem. The heuristic approach is based on a linear combination of three popular software reliability models. A simple, predetermined combination is suggested by assigning equal weights to each component model for the final delivery of the software reliability prediction. In a preliminary examination, this Equally-Weighted Linear Combination (ELC) Model is judged to perform well when applied to three published software failure data sets. We further present five other sets of software failure data taken recently from major projects at the Jet Propulsion Laboratory, and apply the ELC model as well as six other popular models for a detailed comparison and evaluation. A number of statistical techniques are used to determine the applicability of these software reliability models. Our evaluation results indicate that the proposed ELC Model not only performs better than all the other models by a wide margin, but also enjoys favorable properities that practitioners would like to see in their software reliability modeling practices. These properties include: simplicity, low-risk, ease of application, and insensitivity to data noise.

1. Introduction

Since publication of the first software reliability model almost 20 years ago [1][2], over 40 of them are now known to exist in the literature [3]. It is likely that many more unpublished models are in use. It has been shown that there is no best software reliability model for every case under all circumstances [4]-[6], and the practitioners are encouraged to apply as many models as possible. However, in a realistic situation, there are significant technical difficulties and managerial resistance associated with the application of multiple software reliability models and tracking each model during the entire software life cycle. The large number of available models presents potential users with a confusing array of choices. Lacking sufficient guidance to select a small subset of the available models applicable to their development efforts, many potential users abandon the idea of incorporating reliability modeling as a management technique. In other words, the users are left in a dilemma as to which software reliability models to choose, which procedures to apply, and which prediction results to trust, while contending with varying software development practices.

The main difficulty in software reliability engineering practice is to analyze the particular context in which reliability measurement is to take place so as to decide *a priori* which model is likely to be trustworthy. No single model can guarantee its predictive validity across various projects. Even if a user knows that past predictions supplied by a model have been close to the actual behavior of an early part of a particular data set, he or she could not be assured that this particular model will perform well as development progresses and more information is added to the data set.

Another difficulty of modeling and prediction practice is the lack of published software failure data sets that can be used to refine and validate models. Most failure data is classified as proprietary information, making it very difficult to obtain. Moreover, since software failure data reflects the quality of the software development process, the developing party tends to be reluctant to provide outside organizations such revealing information. This makes it a formidable task to obtain enough data to compare software reliability models and predictions across various types of software projects.

We shall describe a low-overhead, low-risk, and high-performance approach to addressing the software reliability prediction problem, and provide five new sets of software failure data to validate the proposed approach.

The motivation, formalization, and advantage of this approach will be presented. For each project contributing data, the mission and development characteristics will be described. Finally, the overall evaluations and observations on the applicability of this empirically-developed model will be discussed.

2. An Equally-Weighted Linear Combination Software Reliability Model

In a recent study [5], we have evaluated 15 well-known software reliability models according to six different qualitative evaluation criteria: 1) model validity, 2) ease of measuring parameters, 3) quality of assumptions, 4) applicability, 5) simplicity, and 6) insensitivity to noise. As a result, a candidate set of six models was considered acceptable under the evaluation criteria. These models include: Jelinski-Moranda Model (JM) [1], Goel-Okumoto Model(GO) [7], Musa-Okumoto Model (MO) [8], Duane Model (DU) [9], Littlewood Model (LM) [10], and Littlewood-Verrall Model (LV) [11]. However, it was also observed that there was no best model for all cases. Neither was there a model whose overall performance was outstanding. A tie among JM, GO, MO, and LV was observed to make better predictions than DU and LM for the limited data set.

Given that no single reliability model's performance was outstanding, and that we did not have enough resources to keep track of all the existing software reliability models, we developed the following heuristic algorithm as a practical approach to form a simple, low-risk, high-performance linear combination model:

a. Identify a candidate set of models (called component models)

b. Select models that tend to cancel out in their biased (if any) predictions.

c. Keep track of the software failure data with all the component models.

d. Equally weight the selected component models to form a linear combination model

e. Each time when a prediction is called for, apply the arithmetic average (mean value) of the predictions from the component models, or select their middle prediction value (median value), for the final prediction.

We further recommend GO, MO, and LV as the three component models to form an Equally-Weighted Linear Combination (ELC) Model. The arithmetic average of each component model's prediction is taken as the ELC prediction. in other words,

$$ELC = \frac{1}{3} GO + \frac{1}{3} MO + \frac{1}{3} LV$$

There are good reasons for choosing these component models:

a. Their prediction validity has been observed in our recent investigation [5] [6]. In fact, they are judged to perform well by many practitioners, and they have been widely used.

b. They represent different categories of models: GO (similar to JM and LM) represents the the exponential shape non-homogeneous Poisson process (NHPP), MO represents the logarithmic shape NHPP model, and LV represents the inverse-polynomial shape Bayesian model.

c. Their predictive biases tend to cancel: GO tends to be optimistic, LV tends to be pessimistic, and MO might go either way.

It has been proposed in [4] that rather than predicting software reliability by using only one model, a *meta-predictor* could be formed to take a linear combination of two (or more) predictions with the weights chosen in some optimal way (e.g., posterior probabilities). A Bayesian interpretation of "prequential likelihood" as *a posteriori* was suggested, which was dynamically calculated in a short time window to determine the weight assignments.

The approach we propose here is a similar but simpler one: the weight assignments are constant and equal for all component models. The motivation of this approach is to reduce the risks of relying on any particular model, while preserving the simplicity of the prediction process. It was observed that good performance of a model in a project for a period of time does not guarantee good prediction of the model for a later time in the same project [4]-[6], and switching among models was suggested when changes of data behaviors were detected. However, we are concerned that the detection of data behavior changes to guide model selection is itself *a priori*, which implies another level of uncertainty and complexity. Therefore, a more straightforward approach suggested by the ELC Model is to take a static and equal weight for the selected models, rather than to dynamically change the weights of each component model for the linear combination. The main theme of our approach is a combination of prediction and modeling: get more accurate predictions while keeping the model simple.

3. Model Evaluation Criteria and Preliminary Results

3.1 Evaluation Criteria for Model Performance

In order to compare different models objectively and quantitatively, four formally defined measures have been adopted. These measures, including Accuracy, Bias, Trend, and Noise, represent various quantities for the quality of software reliability measurement from a particular model. For detailed discussion of each measure, refer to [4] [12] [13] and [14].

- *Accuracy*:

Defined as prequential likelihood (PL) function in [12] as follows. Let the observed data be a sequence of times between successive failures, denoted by $t_1, t_2, \ldots, t_{i-1}$. The objective is to use the data to predict the future unobserved T_i. More precisely, we want a good estimate of $\tilde{F}_i(t)$, defined as $P(T_i<t)$, i.e., the probability that T_i is less than a variable t. The predictive distribution $\tilde{F}_i(t)$ for T_i based on $t_1, t_2, \ldots, t_{i-1}$ will be assumed to have a pdf (probability density function)

$$\tilde{f}_i(t) = \frac{d}{dt} \tilde{F}_i(t).$$

For such one-step-ahead predictions of $T_{j+1}, \ldots, T_{j+n}$, the prequential likelihood is

$$PL_n = \prod_{i=j+1}^{j+n} \tilde{f}_i(t_i)$$

Since this measure is usually very close to zero, we take its logarithmic value for comparisons. The resulting number is always negative. The more negative it gets, the more inaccurate the prediction it represents.

- *Bias*:

Defined as the Kolmogorov Distance [15] of the following sequence of transformations:

$$u_i = \tilde{F}_i(t_i).$$

Each of which is a probability integral transform of the observed t_i using the previously calculated predictor $\tilde{F}_i$ based upon $t_1, t_2, \ldots, t_{i-1}$. To identify the direction a model is biased toward, we use the notation that a positive number means that the model tends to be optimistic, while a negative one represents a pessimistic model. This is achieved by examining u_i's in the *u-plot* [4] to see whether they are below (optimistic) or above (pessimistic) the line of unit slope through the origin. In any case, the smaller the absolute value of the number is, the less bias the model exhibits.

- *Trend*:

Defined as the Kolmogorov Distance of the following sequence of transformations:

$$x_i = -ln(1 - u_i)$$

This measure represents the consistency of the model's bias. A small value means that the model is more adaptable to changes in the data behavior, and hence it could achieve a better performance.

- *Noise*:

Defined as $\sum_i \left| \frac{m_i - m_{i-1}}{m_{i-1}} \right|$, where m_i is the predicted median of T_i.

Again, small values represent less noise in the predictive behavior of the model, indicating better smoothness.

To compare several models for a data set, we use an evaluation algorithm that gives the rank of each model for each measure first, and then equally weighs the ranks of the four measures by summing them up. The models with a lower overall sum are judged better than those with a higher sum. In case there is a tie, the model with a better accuracy measure is ranked higher since this measure is considered more important than the other three measures. It is recognized, however, that different weights for these measures might be applied. Moreover, the value of each measure should be examined in case some "wild" measure might totally disqualify a model in that measurement. Nevertheless, we decide to use this simple ranking algorithm without elaborating the details of each measure, since such elaborations might involve subjective judgement calls which could be themselves biased. We will be using this strategy for model comparisons throughout this paper, where the modeling procedure is facilitated by the SRMP tool [4], and the SMERFS tool [16].

3.2 Preliminary Results by Using Three Published Data Sets

Some approaches in using multiple models for data application could be found in [17]. In order to obtain preliminary results for the performance of the proposed ELC Model, three sets of published data [4] [18] were applied. The results are presented in Tables 1-3.

Data Set 1							
Measure	JM	GO	MO	DU	LM	LV	ELC
Accuracy	-764.3(7)	-762.7(6)	-755.4(1)	-759.6(5)	-757.0(3)	-759.2(4)	-756.3(2)
Bias	.1837(7)	.1486(4)	.0822(2)	.1655(6)	.1063(3)	-.1493(5)	.0803(1)
Trend	.1237(6)	.1280(7)	.0632(1)	.0888(4)	.0724(3)	.1060(5)	.0646(2)
Noise	8.126(7)	7.233(6)	3.782(3)	3.081(1)	7.180(5)	3.093(2)	4.196(4)
Rank	(7)	(6)	(1)	(5)	(3)	(4)	(2)

RECOMMENDED MODELS: 1. MO 2. ELC

Table 1: Model Comparisons for Data Set 1 in [18]

Data Set 2							
Measure	JM	GO	MO	DU	LM	LV	ELC
Accuracy	-466.3(4)	-466.6(6)	-465.6(2)	-467.6(7)	-466.3(4)	-465.1(1)	-465.6(2)
Bias	.1107(2)	.1258(3)	-.1632(6)	.2063(7)	.1011(1)	-.1608(5)	-.1557(4)
Trend	.1109(6)	.1153(7)	.0825(4)	.0524(2)	.1091(5)	.0497(1)	.0804(3)
Noise	4.120(6)	3.870(5)	2.773(3)	1.984(1)	4.662(7)	2.028(2)	2.831(4)
Rank	(6)	(7)	(3)	(5)	(4)	(1)	(2)

RECOMMENDED MODELS: 1. LV 2. ELC

Table 2: Model Comparisons for Data Set 2 in [18]

Data Set 3							
Measure	JM	GO	MO	DU	LM	LV	ELC
Accuracy	-811.1(2)	-811.2(4)	-811.1(2)	-814.3(7)	-811.3(5)	-812.7(6)	-810.8(1)
Bias	.0835(5)	.0761(3)	.0586(1)	.0994(7)	.0829(4)	-.0845(6)	.0640(2)
Trend	.0623(4)	.0663(6)	.0487(2)	.0740(7)	.0602(3)	.0630(5)	.0467(1)
Noise	5.384(7)	5.209(5)	4.088(3)	2.426(1)	6.002(7)	3.714(2)	4.224(4)
Rank	(3)	(4)	(2)	(7)	(5)	(6)	(1)

RECOMMENDED MODELS: 1. ELC 2. MO

Table 3: Model Comparisons for Data Set 3 in [18]

It can be seen from these tables that the proposed ELC Model performs relatively well compared with the other models. It is ranked either number 1 or 2 for all the three data sets. In fact, none of the individual measures obtained from this model is ranked worse than the average (4). This implies that the model not only performs well in general, but also behaves satisfactorily in each individual category of measures. Results from this preliminary investigation gave us enough confidence for further application of this model. In the following sections, we incorporate software failure data taken from five projects at the Jet Propulsion Laboratory (JPL) to make a more detailed assessment of this model.

4. Project Descriptions and Development Characteristics

In this section, we briefly describe the JPL projects from which software failure data was collected for examination. Specifically, the following information is given:

a. Project name and brief description of project goals.

b. Functional description of software developed for the project.

c. Size of the software, measured in uncommented source lines of code, and the type of language in which the software was developed.

d. Brief description of the software failure reporting mechanism used during development. The information tracked by the reporting system is described as well as that which is not.

e. Assumptions made about the development effort in applying the models.

4.1 Voyager

Originally proposed as a grand tour of the outer gas giants, Voyager was implemented as two identical spacecraft that were launched within two months of each other in mid-1977. These spacecraft were designed to provide close-up views of the Jovian and Saturnian systems. Voyager 2 would examine in more detail objects and planetary features identified by Voyager 1 as meriting greater attention. In addition, there were also options for Voyager 2 to fly past Uranus and Neptune; these options were exercised in January 1986 and August 1989 respectively.

Voyager was one of the first outer planetary spacecraft in which a significant fraction of the functionality was provided by software. The software was divided among three subsystems: the Command and Control Subsystem (CCS), the Flight Data Subsystem (FDS), and the Attitude and Articulation Control Subsystem (AACS). All three subsystems can be classified as real-time embedded systems.

These three subsystems were all implemented in assembly language. Among the three subsystems, there are approximately 14,000 uncommented source lines of code. There are no failure-history records surviving from subsystem-level software development that could have been used in the modeling effort. After subsystem-level software development, the CCS, FDS, and AACS were integrated into the spacecraft at the Spacecraft Assembly Facility (SAF). During the time of spacecraft system testing, software failures were reported via the Problem/Failure Reporting (PFR) system. Among the items reported by this system are:

a. Time of failure.

b. Failure type (hardware, software, manufacturing flaw, etc.)

c. Subsystem in which the failure occurred.

It is important to note that the following items were not systematically recorded, and were unavailable for use in the modeling effort:

a. Execution times between successive failures, or comparable information (e.g., total time spent testing during a calendar interval).

b. Operational profile information (e.g., functional area being tested, referenced to requirements or design documentation; subsystem being tested; points at which the testing method may have changed.)

It is therefore necessary to assume that the amount of time spent testing in a unit interval of calendar time was relatively constant, and that the testing method used during spacecraft system test did not vary significantly. Largely because of the lack of operational profile information, we decided to model the reliability of the spacecraft flight software as a whole, rather than attempt to separately model the reliability of the individual subsystems. It is also assumed that changes to the software under test that were caused by the imposition of additional requirements or design changes affected only a small fraction of the software.

4.2 Galileo

Originally planned as the Jupiter Orbiter Probe (JOP) follow-on to Voyager, Galileo became a flight project at the start of fiscal year 1977. Unlike previous outer solar system missions, Galileo was intended to remain in the neighborhood of a gas giant for an extended interval. This would allow observation of time variations in planetary and satellite features in addition to the single observation opportunities afforded by Voyager fly-by type missions. Galileo was launched in October of 1989, and will reach the Jovian system in 1995.

The three flight computers of Voyager were reduced to two aboard Galileo. The CCS and FDS were combined into the Command and Data Subsystem (CDS). Although its functionality was expanded, the AACS remained as a single subsystem. The CDS functionality remained essentially the same as the combined Voyager CCS and FDS functionality, although more telemetry rates and modes were added as well as the capability of having more sequences executing concurrently. As with Voyager, the flight software can be characterized as real-time, embedded software.

Originally, it was intended that the Galileo flight software be written in the high-level language HAL/S. This goal was partially achieved in that the AACS flight software, comprising approximately 7000 source lines of uncommented code, was written in this language. However, the 15,000 lines of CDS flight software were written in assembly language. As with Voyager, the results presented in the paper are from a specific period during spacecraft system integration and test. Failure history collection during this period was very similar to that for Voyager; so were the modeling assumptions.

4.3 Galileo CDS

In addition to the spacecraft system integration data described above, failure history data during subsystem-level software integration testing was available for the Galileo CDS. We were able to collect the following information for one particular phase of integration testing:

a. Date of failure.

b. Software version in which failure occurred.

Because one of the authors had been involved in CDS development as a tester, we were also able to reconstruct some elements of the testing profile which had not been systematically recoreded. For the CDS integration testing, the number of hours per week during which testing occurred was nearly constant through this phase. After testing week 34, the functional areas of the software were executed roughly the same amount of time every calendar week. We therefore decide to use "failures per week" as the unit of failure intensity in our modeling activities.

4.4 Magellan

Magellan, a derivative of the Venus Orbiting Imaging Radar (VOIR) proposed in the early 1980s, was intended to provide a detailed map of the Cytherian surface. Launched in October of 1989, Magellan achieved orbit around Venus in August 1990. It is currently carrying out its mission as planned.

Magellan was designed with the intention of using flight spare subsystems and components from Voyager and Galileo. The flight computers were derivatives of the Galileo CDS and AACS - in the case of the CDS, approximately 75% of the Galileo flight software wound up being reused for Magellan. The AACS, however, was largely rewritten.

The size of the CDS and AACS software was approximately the same in Magellan as it was in Galileo. However, the Magellan AACS was implemented largely in assembly language. The failure history data collected for our modeling effort came from spacecraft system integration and test. The data available to us was very similar to what was available during this phase for Voyager and Galileo. Consequently, the same assumptions about spacecraft integration were made for Magellan as for Voyager and Galileo.

4.5 Alaska SAR

The Alaska SAR Facility, installed on the Fairbanks campus of the University of Alaska, is a facility for tracking and acquiring data from polar-orbiting Earth resources satellites. The software for the Alaska SAR facility is written in C, Fortran, EQUEL, and OSL. It consists of 103,000 uncommented source lines of code, 14,000 lines of which are reused code.

The failure data reported here was obtained from the development organization's anomaly reporting system during software integration and test. The items reported were quite similar to those reported by previously mentioned projects using the PFR system. Since information regarding both (1) execution time between successive failures and (2) operational profile information were not available, it was therefore necessary to assume that the amount of time spent testing in a unit interval of calendar time was relatively constant, and that the testing method used during software integration test did not vary significantly. Largely because of the lack of operational profile information, we decide to model the reliability of the facility as a whole, rather than attempt to separately model and reliability of the individual components.

5. Final Results of Model Performance

In this section, we compare the applicability of the seven competing models to the JPL failure data in terms of predictive quality, biasedness, trend, and sensitivity to noise. The model's performance with respect to these criteria is identified and rated across the ensemble of the failure data sets.

5.1 Model Evaluations Using Project Data

The first data set analyzed came from the Voyager Project. In the following modeling exercise, we translated the original failure count data to time-between-failure data by assuming failures discovered in a time interval were equally distributed in that interval. Results of these four measures against the seven models are presented in Table 4. Numbers in parenthesis represent the rank of each model under a certain measure in that row. It can be seen that for this data set, the ELC Model and the LV Model are the best models.

Voyager Flight Software (133 data points/starting data–2)							
Measure	JM	GO	MO	DU	LM	LV	ELC
Accuracy	-894.7(7)	-573.7(4)	-571.5(3)	-586.6(5)	-829.9(6)	-549.1(1)	-554.0(2)
Bias	.2994(6)	.2849(4)	.2849(4)	.2703(3)	.2994(6)	-.0793(1)	.2084(2)
Trend	.0995(6)	.0965(4)	.0957(3)	.2551(7)	.0994(5)	.0876(2)	.0872(1)
Noise	40.00(7)	13.81(3)	9.225(2)	8.402(1)	30.00(6)	24.51(5)	15.15(4)
Rank	(7)	(4)	(3)	(5)	(6)	(1)	(1)

RECOMMENDED MODELS: 1. LV 2. ELC

Table 4: Model Comparisons for the Voyager Data

Galileo Flight Software (224 data points/starting data–24)							
Measure	JM	GO	MO	DU	LM	LV	ELC
Accuracy	-1074(3)	-1075(5)	-1078(6)	-1098(7)	-1074(3)	-1051(2)	-1019(1)
Bias	.3378(4)	.3378(4)	.3379(6)	.1944(1)	.3382(7)	-.2592(3)	.1991(2)
Trend	.4952(4)	.4954(5)	.5041(7)	.4618(3)	.4954(5)	.1082(1)	.2781(2)
Noise	2.607(3)	2.593(2)	2.395(1)	4.541(5)	2.624(4)	23.33(7)	17.72(6)
Rank	(3)	(4)	(7)	(5)	(6)	(2)	(1)

RECOMMENDED MODELS: 1. ELC 2. LV

Table 5: Model Comparisons for the Galileo Flight Data

Galileo CDS Flight Software (360 data points/starting data–152)							
Measure	JM	GO	MO	DU	LM	LV	ELC
Accuracy	-665.4(4)	-659.5(3)	-681.1(6)	-728.5(7)	-665.4(4)	-641.7(2)	-641.1(1)
Bias	.1731(2)	.1731(2)	.1635(1)	.1755(6)	.1732(4)	-.2587(7)	.1732(4)
Trend	.3505(4)	.3425(3)	.4683(6)	.4687(7)	.3505(4)	.2613(1)	.2855(2)
Noise	5.382(6)	5.143(5)	3.238(4)	2.829(2)	5.382(6)	2.570(1)	2.853(3)
Rank	(4)	(3)	(5)	(7)	(6)	(2)	(1)

RECOMMENDED MODELS: 1. ELC 2. LV

Table 6: Model Comparisons for the Galileo CDS Subsystem

Magellan Flight Software (197 data points/starting data–50)							
Measure	JM	GO	MO	DU	LM	LV	ELC
Accuracy	-627.1(4)	-627.1(4)	-627.1(4)	-616.0(1)	-627.1(4)	-622.9(3)	-619.1(2)
Bias	.2968(3)	.2968(3)	.2968(3)	.1858(1)	.2969(5)	-.3483(7)	.2140(2)
Trend	.2399(4)	.2399(4)	.2399(4)	.2180(3)	.2399(4)	.1429(2)	.1400(1)
Noise	1.007(1)	1.007(1)	1.007(1)	2.003(5)	1.009(4)	5.563(7)	4.260(6)
Rank	(3)	(3)	(3)	(1)	(6)	(7)	(2)

RECOMMENDED MODELS: 1. DU 2. ELV

Table 7: Model Comparisons for the Magellan Data

Alaska SAR Ground Software (367 data points/starting data–67)							
Measure	JM	GO	MO	DU	LM	LV	ELC
Accuracy	-915.7(1)	-915.8(4)	-915.7(1)	-925.5(7)	-915.7(1)	-920.5(6)	-916.2(5)
Bias	.3023(1)	.3023(1)	.3023(1)	.4249(7)	.3023(1)	-.3672(6)	-.3434(5)
Trend	.0606(2)	.0615(4)	.0620(5)	.0918(6)	.0606(2)	.1009(7)	.0586(1)
Noise	1.587(2)	1.590(4)	1.395(1)	1.650(5)	1.589(3)	3.183(7)	2.220(6)
Rank	(1)	(4)	(3)	(6)	(2)	(7)	(5)

RECOMMENDED MODELS: 1. JM 2. LM

Table 8: Model Comparisons for the Alaska SAR Data

Secondly, for the Galileo Project, results of the ELC Model compared with the other six competing models are presented in Table 5. It can be seen that for this data set, the ELC Model performs the best.

The next one is Galileo CDS subsystem. Results of the ELC Model compared with the other six competing models are presented in Table 6. It can be seen that for this data set, the ELC Model still performs the best.

The fourth prject is the Magellan spacecraft testing data. Results of the ELC Model compared with the other six competing models are presented in Table 7. It can be seen that for this data set, the ELC Model performs second only to the DU Model.

Finally, for the Alaska SAR project, results of the ELC Model compared with the other six competing models are presented in Table 8. This is the only data set that the ELC Model does not perform very well. It is to be noted, however, that the failure data from this project represents an early stage of software integration testing. The current failure data may differ from those from a later stage during which a more stable testing and operational environment will be set up.

5.2 The Overall Assessment

Tables 9 and 10 list the performance comparisons for all the eight data sets presented in this paper. The overall comparison is done by using all four measures in Table 9, or by using the prequential likelihood measure (the "Accuracy") alone in Table 10, since it is judged to be the most important one. In general, we consider a model as being satisfactory if and only if it is ranked 3 or better out of the 7 models for a particular project. To extend this idea, we define a "handicap" value, which is calculated by subtracting 3 (the "par" value) from the rank of a model for each data set before its ranks being summed up in the overall evaluation. (Or subtract 24 from the "Sum of Rank" row in Tables 9 and 10.) A negative handicap value represents good overall performance.

Summary of Model Ranking Using All Four Measures							
Data Set	JM	GO	MO	DU	LM	LV	ELC
Table 1	(7)	(6)	(1)	(5)	(3)	(4)	(2)
Table 2	(6)	(7)	(3)	(5)	(4)	(1)	(2)
Table 3	(3)	(4)	(2)	(7)	(5)	(6)	(1)
Voyager	(7)	(4)	(3)	(5)	(6)	(1)	(1)
Galileo	(3)	(4)	(7)	(5)	(6)	(2)	(1)
CDS	(4)	(3)	(5)	(7)	(6)	(2)	(1)
Magellan	(3)	(3)	(3)	(1)	(6)	(7)	(2)
ASAR	(1)	(4)	(3)	(6)	(2)	(7)	(5)
Sum	34	35	27	41	38	30	15
"Handicap"	+10	+11	+3	+17	+14	+6	-9
Total Rank	(4)	(5)	(2)	(7)	(6)	(3)	(1)

Table 9: Overall Model Comparisons Using All Four Measures

Summary of Model Ranking Using the Accuracy Measure							
Data Set	JM	GO	MO	DU	LM	LV	ELC
Table 1	(7)	(6)	(1)	(5)	(3)	(4)	(2)
Table 2	(4)	(6)	(2)	(7)	(4)	(1)	(2)
Table 3	(2)	(4)	(2)	(7)	(5)	(6)	(1)
Voyager	(7)	(4)	(3)	(5)	(6)	(1)	(2)
Galileo	(3)	(5)	(6)	(7)	(3)	(2)	(1)
CDS	(4)	(3)	(6)	(7)	(4)	(2)	(1)
Magellan	(4)	(4)	(4)	(1)	(4)	(3)	(2)
ASAR	(1)	(4)	(1)	(7)	(1)	(6)	(5)
Sum	32	36	25	46	30	25	16
"Handicap"	+8	+12	+1	+22	+6	+1	-8
Total Rank	(5)	(6)	(2)	(7)	(4)	(2)	(1)

Table 10: Overall Model Comparisons by the Accuracy Measure

There are several interesting and important points that we can observe from these summary tables:

a. As can be seen, there is a strong correlation between these two comparison strategies. In either case, the ELC Model is considered better than any other single model.

b. The ELC Model is superior for all the data set except the last one (Alaska SAR). By evaluating the handicap value, it is noted that the ELC Model performs extremely well across different category of data. In fact, it is the only model which can achieve a negative "handicap" in the overall evaluation for both evaluation schemes. Moreover, it beats all other models by a significant number of "strokes."

c. By averaging the predictions of the three famous models, the ELC Model appears to be less sensitive to potential data noise than its component models or any other single model. It not only performs well in the cpu-time based failure data (the first three sets), but also behaves well in the non-cpu-time based failure data (the last five sets).

d. The ELC Model is rather consistent. Most other models seem to perform well for a few data sets but poorly for other data sets, and the fluctuation in performance is significant. The ELC Model predicts consistently well, and it usually outperforms its component models.

e. In parallel experiments, we have also applied a linear combination of four models (including JM), and linear combinations of two models (picking two from GO, MO, LV). Although all the combined models do show improvement over their component models, none of them performs better than the ELC Model proposed in this paper.

6. Conclusions

It is clear that there is no "best" model for all cases. A number of models should be applied to each project. However, if we really want to select the "best" model within a number of models, the ELC model or a similar approach would be a recommended candidate. The ELC Model performs significantly better over the entire ensemble of data sets than do the individual models. This model is composed of three credible software reliability models. We feel that this combination is appropriate to development efforts having the same process characteristics as those described in our modeling practice. We believe one of the factors contributing to the improved performance of the ELC Model is the fact that one of the models chosen tends to be biased toward optimistic predictions, while another model is biased toward pessimistic predictions. These biases tend to cancel one another as the individual models are combined.

This model also maintains the characteristic of being simple: it applies an equal weight over its component models for the final prediction. We certainly can apply a more complicated approach by dynamically changing such weights during model application, but it is our opinion that unless a significant improvement has been identified to justify this more complicated approach, the ELC Model is an attractive choice due to its simplicity as well as fairness in treating the component models. A variation of the ELC Model can be generated by selecting the median value from the component models rather than using their averages. This could prevent a "wild" prediction from one of the component model to dominate the average. On the other hand, such a "wild" prediction could prove to be valid. In any case, we are currently exploring these alternative approaches.

We have to agree that the JPL project data presented in this paper contain a lot of noise irrelevant to the modeling concern for software reliability. However, since no other project information about the usage of the software is available, we prefer to present the data sets in the form in which they were collected instead of trying to make an (inevitably) artificial effort to smooth them. Nevertheless, since the data was collected from a formal data collection mechanism, we do believe they are a valid representation (at least in the statistical sense) of software testing and operation activities for these projects. In that sense, model comparisons based on these data should be considered meaningful regarding the predictive validity of each model. The evaluation results of the proposed ELC Model using our project data should provide a high enough merits for its further application and investigation.

Acknowledgement

The research described in this paper was carried out at the University of Iowa under a faculty starting fund, and at the Jet Propulsion Laboratory, California Institute of Technology, through the Director's Discretionary Fund.

References

[1] Z. Jelinski and P.B. Moranda, "Software Reliability Research," *Statistical Computer Performance Evaluation*, W. Freiberger, Ed. New York: Academic, 1972, pp. 465-484.

[2] M. Shooman, "Operational Testing and Software Reliability During Program Development," in *Res. 1973 IEEE Symposium on Computer Software Reliability*, New York, April 30-May 2, 1973, pp. 51-57.

[3] J.D. Musa, A. Iannino, and K. Okumoto, *Software Reliability - Measurement, Prediction, Application*, McGraw-Hill, New York, 1987.

[4] A.A. Abdel-Ghaly, P.Y. Chan, and B. Littlewood, "Evaluation of Competing Software Reliability Predictions," *IEEE Trans. Software Eng.*, vol. SE-12, pp. 950-967, Dec. 1986.

[5] M. Lyu, "Measuring Reliability of Embedded Software: An Empirical Study with JPL Project Data," in *Proceedings of the International Conference on Probabilistic Safety Assessment and Management*, Beverly Hills, California, pp. 493-500, Feb. 4-7, 1991.

[6] S. Brocklehurst, P.Y. Chan, and B. Littlewood, "Recalibrating Software Reliability Models," *IEEE Trans. on Software Eng.*, vol. SE-16, No. 9, pp. 458-470, April 1990.

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

[8] J.D. Musa and K. Okumoto,"A Logarithmic Poisson Execution Time Model for Software Reliability Measurement," *Proceedings Seventh International Conference on Software Engineering*, Orlando, pp. 230-238, 1984.

[9] J.T. Duane," Learning Curve Approach to Reliability Monitoring," *IEEE Trans. Aerospace*, vol. AS-2, pp. 563-566, 1964.

[10] B. Littlewood,"Stochastic Reliability Growth: A Model for Fault-Removal in Computer Programs and Hardware Designs," *IEEE Trans. Rel.*, vol. R-30, pp. 313-320, Oct. 1981.

[11] B. Littlewood and J.L. Verrall, "A Bayesian Reliability Growth Model for Computer Software," *J. Roy. Statist. Soc. C*, vol. 22, pp. 332-346, 1973.

[12] A.P. Dawid, "Statistical Theory: The Prequential Approach," *J. Roy. Statist. Soc. A*, vol. 147, pp. 278-292, 1984.

[13] A.P. Dawid, "The Well-Calibrated Bayesian"(with discussion, *J. Amer. Statist. Ass.*, vol. 77, pp. 605-613, 1982.

[14] A. Iannino, J.D. Musa, K. Okumoto, and B. Littlewood, "Criteria for Software Reliability Model Comparisons," *IEEE Transactions on Software Engineering*, vol. SE-10, No. 9, pp. 687-691, November 1984.

[15] M.G. Kendall and A. Stuart, *The Advanced Theory of Statistics*. London: Griffin, 1961.

[16] W.H. Farr, and O.D. Smith, *Statistical Modeling and Estimation of Reliability Functions for Software (SMERFS) User's Guide*, NSWC TR 84-373, Revision 1, December 1988.

[17] J.B. Bowen, "Application of a Multi-Model Approach to Estimating Residual Software Faults and Time Between Failures," *Quality and Reliability Engineering Int.*, vol 3, pp. 41-51, 1987.

[18] J.D. Musa, "Software Reliability Data," Data and Analysis Center for Software, Rome Air Development Center, Rome, NY, Tech. Rep.