# Applying Reliability Models More Effectively

MICHAEL R. LYU, *University of Iowa*
ALLEN NIKORA, *Jet Propulsion Laboratory, Caltech*

◆ *Combining the results of individual models may give more accurate predictions than using component models alone, providing a general reliability method across projects.*

More than 40 software-reliability models have been created since the first one appeared in 1972.[1] As new projects arose, new reliability models were created to suit them. Now software engineers have a plethora of reliability models, none of which work optimally across projects.

Consequently, a major difficulty in software measurement is analyzing the context in which measurement is to take place to determine beforehand which model is likely to be trustworthy. Because software development and operation involve many intricate human activities and because software failure patterns are uncertain, such determinations are difficult — if not impossible. Also, project data varies considerably and often does not comply with a model's underlying assumptions.

Thus, practitioners have no reliable way of knowing in advance which model is likely to produce the most trustworthy predictions.

Instead of developing more detailed — and potentially more complicated —models, we chose to focus on using existing models more effectively. To this end, we have developed a set of linear combination models that combine the results of single, or component, models. As measured by statistical methods for determining a model's applicability to a set of failure data, a combination model tends to have more accurate short-term and long-term predictions than a component model.

After evaluating these models using both historical data sets and data from recent Jet Propulsion Laboratory projects, we have found that they are consistently satisfactory. To make it easier to apply reliability models and to form combination models, we are developing a tool to automate many reliability-measurement tasks.

0740-7459/92/0700/0043/$03.00 © IEEE

### WELL-KNOWN RELIABILITY MODELS

The traditional software-reliability model is a set of techniques that apply probability theory and statistical analysis to software reliability. A reliability model specifies the general form of the dependence of the failure process on the principal factors that affect it: fault introduction, fault manifestation, failure detection and recovery, fault removal, and operational environment. The primary goal of these models is to assess current reliability and forecast future reliability.

We believe the component models from two reliability-measurement tools — Statistical Modeling and Estimation of Reliability Functions for Software and Software Reliability Modeling Programs — are the most frequently used.

SMERFS is the only tool that allows multiple time-domain and interval-domain parameter-estimation procedures. It is available free from William Farr, Naval Surface Warfare Center, Code K-52, Dahlgren, VA 22448. SRMP is the only tool that offers methods to analyze prediction quality. It provides users with graphics (like *u*-plots and *y*-plots) and statistics (like prequential likelihood figures) that help detect the bias and noise of the prediction errors made by each model it implements. SRMP is available from Reliability and Statistical Consultants, 5 Jocelyn Rd., Richmond, Surrey TW9 2TJ UK.

The models these tools offer are

- ♦ *Jelinski-Moranda* (JM): One of the earliest models, it assumes failures occur purely randomly and that all faults contribute an equally to total unreliability. When a failure occurs, it assumes that the fix is perfect; thus, the program's failure rate improves by the same amount at each fix. (Z. Jelinski and P. Moranda, "Software Reliability Research," in *Statistical Computer Performance Evaluation*, W. Freiberger, Academic Press, New York, 1972, pp. 465-484.) — *SMERFS, SRMP*
- ♦ *Bayesian Jelinski-Moranda* (BJM): Essentially the same as JM, this model uses a Bayesian inference scheme rather than maximum likelihood. (A. Abdel-Ghaly, P. Chan, and B. Littlewood, "Evaluation of Competing Software Reliability Predictions," *IEEE Trans. Software Engineering*, Sept. 1986, pp. 950-967.) — *SMERFS*
- ♦ *Schneidewind* (SM): Similar to JM, this model's philosophy is that the error-detection process changes as testing progresses and that recent error counts are usually more useful than earlier counts in predicting future counts. (N. Schneidewind, "Analysis of Error Processes in Computer Software," *SigPlan Notice*, June 1975, pp. 337-346.) — *SMERFS*
- ♦ *Geometric* (GM): A variation of JM, this model does not assume a fixed, finite number of program errors, nor does it assume that errors are equally likely to occur. (P. Moranda, "Event-Altered Rate Models for General Reliability Analysis," *IEEE Trans. Reliability*, Dec. 1979, pp. 376-381.) — *SMERFS*
- ♦ *Generalized Poisson* (PM): Similar to JM, except within the error-count framework. (R. Schafer et al., "Validation of Software Reliability Models," Tech. Report RADC-TR-79-147, Rome Air Development Ctr., Rome, N.Y., 1979.) — *SMERFS*

## MODELING STRATEGY

To create a combination model for improving reliability measurement, we recommend the following procedure:

1. Identify a basic set of models (the component models). If you know the project testing environments, select models whose assumptions are closest to the real environments.

2. Select models whose prediction biases tend to cancel out. A prediction bias is either pessimistic or optimistic.

3. Separately apply each component model to the failure data.

4. Apply certain criteria to weigh the selected component models and form one or more linear combination models for final predictions. The weights can be either static or dynamic.

In general, this approach is expressed as a mixed distribution:

$$\hat{f}_i(t) = \sum_{j=1}^{n} \omega_i^j \hat{f}_i^j(t)$$

where $n$ is the number of models, $\hat{f}_i^j(t)$ is the predictive probability density function of the $j$th component model, given that you have made $i - 1$ observations of times between successive failures, and

$$\sum_{j} \omega_i^j = 1$$

for all *i*s.

The linear combination model tends to preserve the features inherited from its component models. Also, because each component model performs reliability calculations independently, the combination model remains fairly simple. The component models are plugged into the combination model only at the last stage for final predictions.

Selecting appropriate component models is, of course, important to the success of the combination model. The parameter-estimation method you select to implement the component models may, to a certain extent, affect the combination model's prediction validity. We recommend using component models from two reliability-measurement tools: Statistical Modeling and Estimation of Reliability Functions for Software and Software Reliability Modeling Programs. Contacts for acquiring these tools and the reliability models they include are given in the boxes above and on the facing page.

We felt that the GO, MO, and LV component models of these tools were the best candidates for our linear combination models. We selected them because in our recent investigations, we found that their predictions were valid.[2] Other practitioners have also found that they perform well, and they are widely used.[3] Another reason is that they represent different model categories. GO, which is similar to JM and SM, represents the exponential-shape NHPP model, MO represents the logarithmic-shape NHPP model, and LV represents the inverse-polynomial-shape Bayesian model. Finally, at least with the data set we analyzed, the biases of these models tend to cancel out. GO tends to be optimistic, LV tends to be pessimistic, and MO might go either way.

## COMBINATION MODELS

From the GO, MO, and LV component models, we formed four combination models. The goal of each is to reduce the risk of relying on a specific model, which may produce grossly inaccurate predictions, while retaining much of the simplicity of using the component models:

♦ *Equally Weighted Linear Combination.* This model is the simplest combination to form. Each component model has a con-

♦ *Goel-Okumoto* (GO): Similar to JM, except it assumes the failure rate improves continuously in time. (A. Goel and K. Okumoto, "Time-Dependent Error-Detection Rate Model for Software Reliability and Other Performance Measures," *IEEE Trans. Reliability*, Aug. 1979, pp. 206-211.) — *SMERFS, SRMP*

♦ *Musa-Okumoto* (MO): Similar to GO, except it attempts to consider that later fixes have less effect on program reliability than earlier ones. (J. Musa and K. Okumoto, "A Logarithmic Poisson Execution Time Model for Software Reliability Measurement," *Proc. Int'l Conf. Software Eng.*, IEEE CS Press, Los Alamitos, Calif., 1984 pp. 230-238.) — *SMERFS, SRMP*

♦ *Yamada Delayed S-Shape* (YM): Similar to GO, except it accounts for the learning period that testers go through as they become familiar with the software at the start of testing. (S. Yamada, M. Ohba, and S. Osaki, "S-Shaped Reliability Growth Modeling for Software Error Detection," *IEEE Trans. Reliability*, Dec. 1983, pp. 475-478.) — *SMERFS*

♦ *Littlewood* (LM): Similar to JM, except it assumes that different faults have different sizes (contribute unequally to unreliability), which is more realistic. Larger faults tend to be removed earlier, causing a "law of diminishing returns" in debugging. (B. Littlewood, "Stochastic Reliability Growth: A Model for Fault Removal in Computer Programs and Hardware Designs, " *IEEE Trans. Reliability*, Oct. 1981, pp. 313-320.) — *SRMP*

♦ *Littlewood Nonhomogeneous Poisson Process* (LNHPP): Similar to LM but assumes a continuous change in failure rate, rather than discrete jumps, when fixes take place. (D. Miller, "Exponential Order Statistic Models of Software Reliability Growth, *IEEE Trans. Software Eng.*, Jan. 1986, pp. 12-24.) — *SRMP*

♦ *Littlewood-Verrall* (LV): Lets the size of failure-rate improvement at a fix vary randomly, representing the uncertainty about fault size and the efficacy of the fix. (B. Littlewood and J. Verrall, "A Bayesian Reliability Growth Model for Computer Software," *J. Royal Statistics Soc. C*, Vol. 22, pp. 332-346.) — *SMERFS, SRMP*

♦ *Keiller-Littlewood* (KL): Similar to the LV model but has a different mathematical form for reliability growth. (P. Keiller et al., "Comparison of Software Reliability Predictions," *Proc. IEEE Int'l Symp. Fault-Tolerant Computing*, IEEE CS Press, Los Alamitos, Calif., 1983, pp. 128-134.) — *SRMP*

♦ *Brooks and Motley* (BM): The BM binomial and Poisson models attempt to consider that not all of a program is tested equally during a testing period and that only some portions of the program may be available for testing during its development. (W. Brooks and R. Motley, Analysis of Discrete Software Reliability Models, Tech. Report RADC-TR-80-84, Rome Air Development Ctr, Rome, N.Y., 1980.) — *SMERFS*

♦ *Duane* (DU): Developed for hardware burn-in testing, in which defective system components are detected and replaced in the early days of use. Once again, the model assumes that the failure rate changes continuously in time. (L. Crow, "Confidence Interval Procedures for Reliability Growth Analysis," Tech. Report 197, US Army Materiel Systems Analysis Activity, Aberdeen, Md., 1977.) — *SMERFS, SRMP*

stant, equal weight. The arithmetic average of all component models' predictions is taken as the ELC model prediction

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

These weightings remain constant and unchanged throughout the modeling process. This model follows a strategy similar to that of a Delphi survey, in which authorities working independently are asked for an opinion on a subject, and an average of the results is taken.

♦ *Median-Oriented Linear Combination.* The MLC model does not rely on the arithmetic mean for prediction as in ELC. Instead, it selects the component model whose predicted value lies between optimistic and pessimistic values. The justification for this approach is that the median might be more moderate than the mean in some cases, since it can better tolerate an erroneous prediction that is far away from the others.

♦ *Unequally Weighted Linear Combination.* The ULC model is similar to the MLC model except that optimistic and pessimistic predictions contribute to the final prediction. The prediction is not determined solely by the median value. Here we use weightings similar to those in the Program Evaluation and Review Technique:

$$\mathrm{ULC} = \frac{1}{6}O + \frac{4}{6}M + \frac{1}{6}P$$

where $O$ represents an optimistic prediction, $P$, a pessimistic prediction, and $M$, the median prediction.

♦ *Dynamically Weighted Linear Combination.* In the DLC model, we assume that the applicability of any individual model to the project data may change as testing progresses and therefore that the component models' weights will change according to changes in a model's applicability. Here, we use changes in prequential likelihood — a measure that denotes a model's accumulated accuracy — to assign weights to the component models, which could be taken over a few or many time frames. As a baseline, we formed the simplest DLC model by choosing an observation window of one time frame before each prediction as the reference in assigning weights.

## SELECTING MODELS FOR COMPARISON

To compare the combination models' performance, we selected a subset of the component models in SMERFS and SRMP that ranked the highest in the following criteria:

♦ *Model validity.* We viewed this criterion as the most important because we can quantitatively define it; other measures tend to be more subjective. We adopted four measures to rate model validity:[4,5]

1. *Accuracy.* We defined accuracy as the prequential likelihood measure, in which the observed data is 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 $T_i$ — a random variable to denote the time to the *i*th failure. Our goal is to get a good estimate of $\hat{F}_i(t)$, which is the probability that $T_i$ is less than a specific time value $t$. We assume that the predictive distribution $\hat{F}_i(t)$ for $T_i$ based on $t_1, t_2, \ldots t_{i-1}$ will have a probability density function of

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

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

$$\mathrm{PL}_n = \prod_{i=j+1}^{j+n} \hat{f}_i(t_i)$$

Since this measure is usually very close to zero, we take its logarithmic value for comparison. The resulting number is always negative. Given several models that use the same data set, the model with the largest value gives the most accurate prediction.

### RECENT PROJECTS FROM THE JET PROPULSION LABORATORY

**Voyager.** The Voyager 1 and 2 spacecraft were developed during the mid-1970s and launched in mid-1977. Both spacecraft flew past Jupiter and Saturn. Voyager 2 continued exploring the outer solar system by flying past Uranus in 1986 and Neptune in 1989.

The Voyagers were one of the first spacecraft in which software provided a large part of the functionality. This software, approximately 14,000 lines of uncommented assembly language, was divided among three real-time embedded subsystems — the Attitude and Articulation Control Subsystem, the Command and Control Subsystem, and the Flight Data Subsystem.

The failure data we analyzed comes from spacecraft-system testing, at which point the AACS, CCS, and FDS had been integrated into the spacecraft. Among the items recorded on the problem/failure reports during system test are time of failure, failure type, and subsystem in which the failure occurred. Roughly 9.5 faults per thousand lines of code were discovered during system test.

**Galileo.** Launched in 1989, Galileo was developed as a Jupiter orbiter carrying an atmospheric probe. As with the Voyagers, a large fraction of Galileo's functionality was provided by software. Galileo contains an AACS and a Command and Data Subsystem. Approximately 7,000 uncommented source lines of HAL/S were implemented for the AACS. As with the Voyagers, the failure data comes from spacecraft-system testing. An estimated 10.2 faults per thousand lines of code were detected.

**Galileo CDS.** Failure data for the Galileo Command and Data Subsystem during one phase of subsystem-level integration testing was available for analysis. Because one of us had been involved in this testing effort, we could reconstruct some elements of the testing profile.

For example, we knew that the hours of testing per week was nearly constant throughout the two testing stages. In addition, the main functional areas of the software received roughly the same amount of testing every calendar week.

This information made the failure data more accurate than that for other projects. About 15,000 source lines of assembly language were developed for the CDS. During integration testing, roughly 10.1 faults per thousand lines of code were discovered.

**Magellan.** A large portion of the on-board software for the Magellan Venus radar mapper is derived from Galileo's software. Like Galileo, Magellan has an AACS and a CDS — the number of uncommented source lines of code for each is roughly the same as that for Galileo. As with Galileo and the Voyagers, the failure data comes from the spacecraft-system test period. An estimated 8.0 faults per thousand lines of code were detected during testing.

**Alaska SAR.** The Alaska Synthetic Aperture Radar facility, installed on the Fairbanks campus of the University of Alaska, is a facility for tracking and acquiring data from Earth resources satellites in high-inclination orbits. Totaling about 103,000 uncommented source lines of code, the software is written in a mixture of C, Fortran, Equel, and OSL. About 14,000 lines were reused from previous efforts. We obtained the failure data presented here from the development organization's anomaly reporting system during software integration and test.

As with the other projects, we *assume* the test time per unit interval of calendar time was relatively constant, and the testing method remained constant, since this information was not systematically recorded. Largely because of this lack of information, we decided to model the reliability of the facility as a whole, rather than attempt to model the component reliabilities. For the part of system testing that we analyzed, about 3.6 faults per thousand lines of code were discovered.

2. *Bias.* This measure is the Kolmogorov distance — the maximum absolute vertical deviation — between the perfect prediction line of slope 1 and a plot of the following transformation:

$$u_i = \hat{F}_i(t_i)$$

which is the probability integral transform of the observed $t_i$ using the previously calculated predictor $\hat{F}_i$ based on $t_1, t_2, \ldots t_{i-1}$. A positive number means that the model tends to be optimistic; a negative one means the model tends to be pessimistic. To derive this measure, we examine $u_i$s in the $u$-plot (Sarah Brocklehurst and Bev Littlewood describe the $u$-plot on p. 36) to see whether they are above (optimistic) or below (pessimistic) the line of unit slope through the origin. In any case, the smaller the number's absolute value, the less the model is biased in either direction.

3. *Trend.* Defined as the Kolmogorov distance of the following sequence of transformations:

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

$$y_i = \frac{\sum_{j=1}^{i} x_j}{\sum_{j=1}^{n} x_j}$$

where $i$ is less than or equal to $n$. 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's behavior, and hence it could achieve a better performance.

4. *Noise.* Defined as

$$\sum_i \left| \frac{r_i - r_{i-1}}{r_{i-1}} \right|$$

where $r_i$ is the predicted failure rate $1/(T_i)$. Again, small values represent less noise in the model's prediction behavior, indicating more smoothness. A noise measure of ∞ indicates that the model has predicted a zero failure rate.

♦ *Ease of measuring parameters.* This criterion concerns the number of parameters a model requires and the difficulty in estimating them. Easily measured parameters not only reduce measurement cost but also tend to help reliability engineers successfully interpret the model's physical significance, which can provide feedback to software development.

♦ *Quality of assumptions.* The assumptions a model is based on should be as close to real project testing and operation as possible. If the assumption is testable, it should be supported by data to validate it. If it is not testable, it should be examined for logical consistency.

♦ *Capability.* This criterion refers to the model's ability to estimate reliability-related quantities for software systems, including present reliability, expected date of reaching a reliability objective, and cost required to reach that objective.

♦ *Applicability.* Applicability refers to the usefulness of the model in different development environments, operational environments, and life-cycle phases. It should be evaluated in different size, struc-

**TABLE 1**
**MODEL COMPARISONS FOR MUSA DATA SET 3**

| Measure | JM | GO | MO | DU | LM | LV | ELC | ULC | MLC | DLC |
|---|---|---|---|---|---|---|---|---|---|---|
| Accuracy | –811.1 | –811.2 | –811.1 | –814.3 | –811.3 | –812.7 | –810.8 | –810.8 | –811.1 | –809.1 |
| | (4) | (7) | (4) | (10) | (8) | (9) | (2) | (2) | (4) | (1) |
| Bias | .0835 | .0761 | .0586 | .0994 | .0829 | –.0845 | .0640 | .0594 | .0586 | .0649 |
| | (8) | (6) | (1) | (10) | (7) | (9) | (4) | (3) | (1) | (5) |
| Trend | .0623 | .0663 | .0487 | .0740 | .0602 | .0630 | .0467 | .0474 | .0480 | .0462 |
| | (7) | (9) | (5) | (10) | (6) | (8) | (2) | (3) | (4) | (1) |
| Noise | 5.384 | 5.209 | 4.088 | 2.426 | 6.002 | 3.714 | 4.224 | 4.196 | 4.073 | 3.901 |
| | (9) | (8) | (5) | (1) | (10) | (2) | (7) | (6) | (4) | (3) |
| Rank | (6) | (8) | (4) | (9) | (9) | (6) | (4) | (3) | (2) | (1) |

**TABLE 2**
**MODEL COMPARISONS FOR THE GALILEO CDS SUBSYSTEM**

| Measure | JM | GO | MO | DU | LM | LV | ELC | ULC | MLC | DLC |
|---|---|---|---|---|---|---|---|---|---|---|
| Accuracy | –643.0 | –639.3 | –681.1 | –728.5 | –643.0 | –612.3 | –618.7 | –626.9 | –681.1 | –606.1 |
| | (6) | (5) | (8) | (10) | (6) | (2) | (3) | (4) | (8) | (1) |
| Bias | .1783 | .1783 | .1700 | .1748 | .1784 | –.2581 | .1732 | .1599 | .1700 | .1845 |
| | (6) | (6) | (2) | (5) | (8) | (10) | (4) | (1) | (2) | (9) |
| Trend | .3450 | .3408 | .4262 | .4282 | .3450 | .2426 | .2855 | .3072 | .4261 | .2618 |
| | (6) | (5) | (9) | (10) | (6) | (1) | (3) | (4) | (8) | (2) |
| Noise | 4.042 | 3.908 | 2.673 | 2.287 | 4.042 | 2.564 | 2.853 | 2.958 | 2.672 | 11.19 |
| | (8) | (7) | (4) | (1) | (8) | (2) | (5) | (6) | (3) | (10) |
| Rank | (8) | (6) | (6) | (8) | (10) | (1) | (1) | (1) | (4) | (5) |

tural, functional, and application domains.

♦ *Simplicity.* Simplicity is generally a desirable feature for most mathematical models. The simpler the model, the easier it is to gather project-specific data, select generic parameters from a database, and understand and interpret the modeling results.

♦ *Insensitivity to noise.* Reliability data generally contains information that is irrelevant to the modeling process. A model is appealing if it can make accurate measurements even when failure data is incomplete or contains uncertainties.

After applying the seven evaluation criteria,[2] we found that JM, GO, MO, DU, LM, and LV ranked high enough to warrant further study. We used these six component models plus our four combination models to evaluate prediction validity.

## EVALUATING MODEL PERFORMANCE

To assess the prediction validity of our combination models, we evaluated their performance, plus the performance of the six component models we selected, using first three data sets from John Musa's reliability data compiled in 1980[6] and then data from recent projects at the Jet Propulsion Laboratory. We then evaluated the models in terms of all the data.

To compare models, we first determined each model's rank for each measure. We then equally weighed the ranks by summing them. The models with a lower overall sum were better than those with a higher sum. Of course, others might apply different weights to each measure, and there can always be a "wild" measure that might totally disqualify a model. Nevertheless, we used this simple ranking algorithm without expanding the details of each measure, since such elaborations might involve subjective judgments that could themselves be biased.

**Musa data sets.** Table 1 shows the results from Musa's data set 3, which contains 207 data points. We began predictions at data point 60 so that we would have a small but reasonable set of data points (1–59) for parameter estimation.

The numbers in each row represent the computed measure under each criterion; the ranks are in parentheses. We arrived at the values in Rank (the last row) by summing them.

As the table shows, the combination models performed relatively well compared with the six component models. We obtained similar results for Musa's other data sets.

**JPL data sets.** We collected failure data from recent JPL projects, which are described in the box on the facing page. The data we collected was based mainly on calendar times. The following information, which would have been useful, was not available because it was not routinely recorded:

♦ Execution times between successive failures or comparable information, such as the total time spent testing during a calendar interval.

♦ Operational profile information (like functional area being tested), referenced to requirements or design documentation, the subsystem being tested, and the points at which the testing method may have changed.

In general, data based on calendar time tends to be noisy and might not comply with most of the reliability models' assumptions. We present it to show circumstances typical of actual practice.

Table 2 shows comparisons when we

## TABLE 3
## SUMMARY OF MODEL RANKING FOR EACH DATA SET USING ALL FOUR VALIDITY MEASURES

| Data | JM | GO | MO | DU | LM | LV | ELC | ULC | MLC | DLC |
|---|---|---|---|---|---|---|---|---|---|---|
| Musa data set 1 | (10) | (9) | (1) | (6) | (8) | (6) | (4) | (2) | (3) | (5) |
| Musa data set 2 | (9) | (10) | (6) | (7) | (8) | (1) | (4) | (5) | (2) | (2) |
| Musa data set 3 | (6) | (8) | (4) | (9) | (9) | (6) | (4) | (3) | (2) | (1) |
| Voyager | (10) | (7) | (6) | (7) | (9) | (2) | (2) | (4) | (5) | (1) |
| Galileo | (5) | (7) | (10) | (6) | (9) | (4) | (1) | (3) | (8) | (2) |
| Galileo CDS | (8) | (6) | (6) | (8) | (10) | (1) | (1) | (1) | (4) | (5) |
| Magellan | (5) | (5) | (8) | (1) | (9) | (10) | (1) | (5) | (4) | (3) |
| Alaska SAR | (1) | (5) | (1) | (9) | (3) | (10) | (8) | (7) | (3) | (6) |
| Sum of rank | 54 | 57 | 42 | 53 | 65 | 40 | 25 | 30 | 31 | 25 |
| Handicap | +22 | +25 | +10 | +21 | +33 | +8 | −7 | −2 | −1 | −7 |
| Total rank | (8) | (9) | (6) | (7) | (10) | (5) | (1) | (3) | (4) | (1) |

## TABLE 4
## SUMMARY OF MODEL RANKING FOR EACH DATA SET USING ACCURACY MEASURE ONLY

| Data | JM | GO | MO | DU | LM | LV | ELC | ULC | MLC | DLC |
|---|---|---|---|---|---|---|---|---|---|---|
| Musa data set 1 | (10) | (9) | (2) | (8) | (6) | (7) | (5) | (4) | (3) | (1) |
| Musa data set 2 | (7) | (9) | (4) | (10) | (7) | (1) | (4) | (4) | (3) | (2) |
| Musa data set 3 | (4) | (7) | (4) | (10) | (8) | (9) | (2) | (2) | (4) | (1) |
| Voyager | (10) | (7) | (6) | (8) | (9) | (2) | (3) | (4) | (5) | (1) |
| Galileo | (5) | (7) | (9) | (10) | (5) | (4) | (2) | (3) | (8) | (1) |
| Galileo CDS | (6) | (5) | (8) | (10) | (6) | (2) | (3) | (4) | (8) | (1) |
| Magellan | (6) | (6) | (6) | (2) | (6) | (5) | (3) | (4) | (6) | (1) |
| Alaska SAR | (2) | (6) | (2) | (10) | (2) | (9) | (8) | (7) | (2) | (1) |
| Sum of rank | 50 | 56 | 41 | 68 | 49 | 39 | 30 | 32 | 39 | 9 |
| Handicap | +18 | +24 | +9 | +36 | +17 | +7 | −2 | 0 | +7 | −23 |
| Total rank | (8) | (9) | (6) | (10) | (7) | (4) | (2) | (3) | (4) | (1) |

applied the four measurements of the model validity criterion for the Galileo Command and Data Subsystem's flight software, which is a representative data set. (Results from all the data sets would take too much space.) The data contains 358 points, and the starting point is 152. As the table shows, the ELC and ULC models ranked the highest.

**Combined data sets.** Tables 3 and 4 list the performance comparisons for all eight data sets we investigated. In Table 3, we used the four model validity measures; in Table 4, we used only the accuracy measure, since we thought it was the most important and would give a more detailed breakdown of performance.

We considered a model satisfactory if and only if its ranking was 4 or better. We arrived at the values in Handicap by subtracting 4 (the par rank) from the rank of a model for each data set before we added up its rankings in the overall evaluation (another way is to subtract 32 from Sum of rank). A negative handicap means that the model's overall performance was satisfactory for the eight data sets.

These tables illustrate several important points:

♦ In general, the combination models perform better than the component models. In Table 3 (all criteria), the only acceptable models (those with a negative handicap) are the combination models. In Table 4 (accuracy criterion alone), the three acceptable models are also combination models. The handicap values of the combination models usually beat those of the component models by a significant margin.

♦ When the predictions from GO, MO, and LV are weighted or averaged, the combinational models are less sensitive to potential data noise than component models. This is true with data based on both execution and calendar time. Across all project data for the four accuracy criteria, the combination models sometimes outperform all their component models and never perform worse than the worst component model.

● The DLC and ELC models perform more consistently than the other models. Most other models seem to perform well for a few data sets but poorly for other data sets — and the fluctuation is significant. The ELC model's performance is due to its equal weighting, which preserves GO's, MO's, and LV's good properties. On the other hand, since the DLC model is allowed to change its weightings dynamically, according to the outcome of the accuracy measure, it can consistently produce the best accuracy measure for almost every data set. This consistency suggests that if you use whatever accuracy measures you deem the most important as the weighting criterion in forming the DLC model, you will get the best results.

## EXTENSIONS AND ALTERNATIVES

You can extend or alter our basic approach in the following ways:

♦ Extend the DLC model by increasing the size of the observation window from one time frame to $N$ time frames. The DLC model consistently produces the best accuracy measure, but with only one observation window, it might fail to note a global measurement trend. Thus, a natural extension is to enlarge the window.

♦ Try to apply models other than GO, MO, and LV as component models. If some models perform well in a particular data set, they should be the candidate component models to form a combination model.

♦ Use more than three models as component models. We believe that the more component models you apply, the better the prediction. However, more computations are required, and the returns may diminish as more models are added.

♦ Apply alternative weighting schemes that are based on project criteria and engineering judgments. Our approach is flexible enough that you can decide how you want to form a combination model.

♦ Use the combination models themselves as component models to form another combination model.

♦ As the original assumptions behind each model become lost through the layers of linear combinations, a distribution-free (nonparametric) modeling technique may emerge.

In our investigation, the most promising approach was to extend the DLC model. We considered a DLC model with a fixed $N$ window, DLC/F, and a DLC model with a sliding $N$ window, DLC/S. Figure 1 shows how the two models differ.

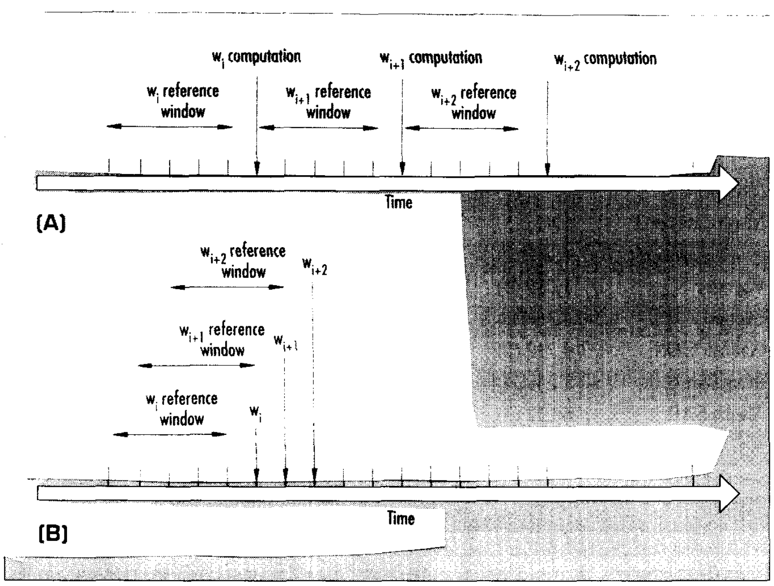


***Figure 1.*** *(A) The DLC model with a fixed-size observation window and (B) the DLC model with a sliding-size observation window.*

In the DLC/F window, the weight assignments for each model are based on changes in the accuracy measure over the last $N$ observations. The weight assignment for each model remains fixed for the next $N$ predictions. At the end of that time, the weights are recomputed according to the changes in accuracy over the last $N$ observations. To compute the weight of a component model, you first determine the amount of change in component model $A$'s accuracy measure over the last $N$ observations. You then identify component model $B$, the component model whose accuracy measure changed the most. The unnormalized weight for $A$ is simply the ratio of the change in its accuracy measure to the change in $B$'s accuracy measure.

In the DLC/S model, you recompute the weight assignments for each model at each data point, using changes in the accuracy measure over the last $N$ observations as the basis for determining each model's weight. To compute weights for component models, the procedure is the same as that in the DLC/F model.

Figure 2 summarizes the accuracy measure of the DLC/F and DLC/S type models, normalized with respect to the number of measured points in each data set before being summed up for the eight data sets.

As Figure 2 shows, the DLC/S model is generally superior to the DLC/F model. This result is not surprising, since DLC/S allows the observing window to advance dynamically as step-by-step prediction moves ahead. In general, the accuracy of the DLC/F model deteriorates when the window becomes larger. The DLC/S model's performance, on the other hand, improves when the window becomes larger, but only slightly larger. We found that a window size of three to four time frames is optimal.

Of course, the best window size depends on your development environment, testing scheme, and operational profile, but, in general, the window size should be fewer than five time frames, since the model is then able to catch fast shifts in model applicability among the component models.

The accuracy measure in Figure 2 is

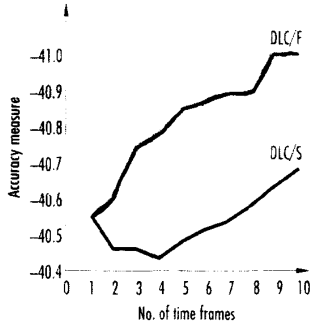


***Figure 2.*** *Summary of the DLC/F and DLC/S models for windows up to 10 time frames.*

## TABLE 5
## SUMMARY OF MODEL RANKING FOR LONG-TERM PREDICTIONS USING MEAN SQUARE ERROR

| Data | GO | MO | LV | ELC | DLC |
|---|---|---|---|---|---|
| **Musa data set 1** | **2,117 (5)** | **687.4 (4)** | **567.7 (3)** | **266.7 (2)** | **169.7 (1)** |
| Musa data set 2 | 1,455 (5) | 1,421 (4) | 246.1 (1) | 930.5 (2) | 955.7 (3) |
| **Musa data set 3** | **480.0 (2)** | **253.2 (1)** | **2,067 (5)** | **745.5 (3)** | **779.8 (4)** |
| Voyager | 1,089 (4) | 782.9 (2) | 5,283 (5) | 130.1 (1) | 876.7 (3) |
| **Galileo** | **4,368 (4)** | **4,370 (5)** | **539.3 (1)** | **2,171 (3)** | **1,791 (2)** |
| Galileo CDS | 4,712 (5) | 3,073 (3) | 4,318 (4) | 1,322 (2) | 1,141 (1) |
| **Magellan** | **3,247 (4)** | **3,248 (5)** | **219.5 (1)** | **1,684 (3)** | **1,354 (2)** |
| Alaska SAR | 60.22 (3) | 60.12 (1) | 104.45 (5) | 68.44 (4) | 60.15 (2) |
| **Sum of MSEs** | **17,528.5** | **13,896.6** | **13,345.0** | **7,317.3** | **7,128.3** |
| Sum of rank | 32 | 25 | 25 | 20 | 18 |
| **Overall rank** | **(5)** | **(4)** | **(3)** | **(2)** | **(1)** |

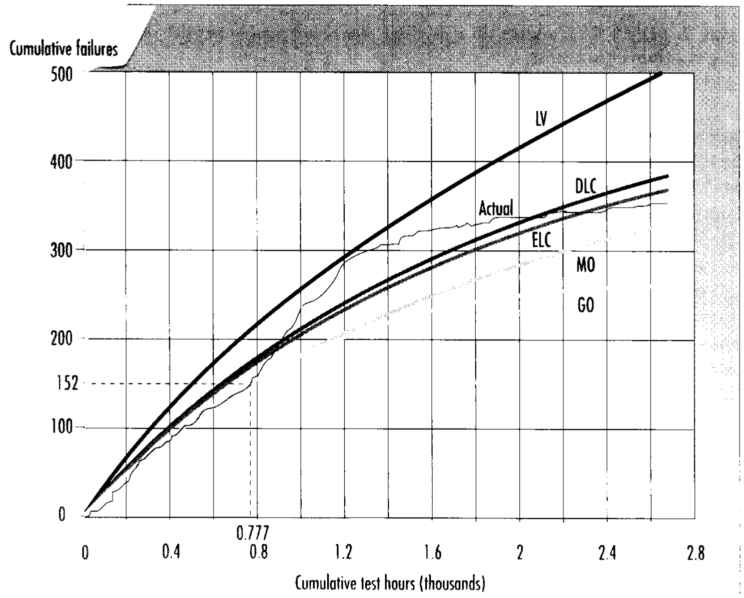


*Figure 3. Long-term predictions for JPL's Galileo Command and Data Subsystem.*

the prequential likelihood, but other accuracy measures, such as the Akaike information criterion — a criterion to denote how close a prediction is to the actual data[7] — or mean square error, are also feasible. The main strength of the DLC models is that they combine component models in a way that lets the output be fed back for model adjustment.

The fundamental approach of the linear combination models is simple. However, by applying more complicated procedures, we risk losing the individual model's assumptions about the physical process. It then becomes harder to get insight into the process of reliability engineering. Most reliability models view software as a black box, from which to observe failure data and make predictions. In that context, our combination models do not degrade any properties assumed in current reliability-modeling practices.

### LONG-TERM PREDICTIONS

Our results showed that the combination models performed well in making step-by-step predictions — in which you can adjust the model's parameters for each prediction — but we also wanted to determine how they performed in making long-term predictions, say 20 failures ahead. For this evaluation, we selected the ELC and DLC models and compared them with the GO, MO, and LV component models. Figure 3 shows the prediction curve for each model for the Galileo CDS data.

We used the first 152 data points in the project, or up to 777 cumulative test hours as indicated by the dashed line, to estimate each model's parameters. Immediately following this estimation stage is the prediction stage. For the Galileo CDS, these two stages follow the project's natural breakdown into two testing stages.

For the DLC model, we computed model preferences and weights in the estimation phase, and fixed the weight assignments in the prediction phase.

LV's prediction curve is too pessimistic, and GO's and MO's are too optimistic. In fact, all three curves for the component models are out of the actual project data curve (the line labeled Actual). ELC and DLC, on the other hand, compensate these extremes and make rather reasonable long-term predictions.

To show quantitative comparisons of long-term predictions, we use mean square error instead of prequential likelihood. Prequential likelihood is more appropriate for comparing step-by-step predictions, while the mean square error provides a more widely understood measure of the distance between actual and predicted values. The mean square error is defined as

$$\text{MSE} = \frac{\sum_{i=1}^{N} \left|\hat{y}_i - y_i\right|^2}{N}$$

where $N$ is the total number of predicted points in the prediction phase, and $\hat{y}_i$ and $y_i$ are the predicted and actual number of failures, respectively.

Table 5 shows the summary of long-term predictions. The values under Sum of MSEs and Sum of ranks show that the ELC and DLC models generally perform

better than component models. Even though the component models make a better prediction than the ELC and DLC models on several occasions, they also perform significantly worse on others. The ELC and DLC models, on the other hand, never make the worst long-term predictions.

## AUTOMATING RELIABILITY MEASUREMENT

Selecting component models to form combination models can be tedious and computation intensive. We are in Phase 1 of a three-phase effort to develop a tool, called Computer-Aided Software Reliability Estimation, which will automate most reliability-measurement tasks.

Figure 4 shows CASRE's architecture. You can find many of its functions in current reliability-measurement tools, but no other tool lets you combine the results of several models in addition to executing one model. Feedback from model evaluation helps you identify a model or combination of models best suited to the failure data being analyzed. Also, CASRE's I/O facility, the user interface, and the measurement procedures are greatly enhanced over those in existing tools.

Figures 5 and 6, two screen dumps from CASRE, show that you have many choices of models and evaluation criteria, yet the selection operation remains fairly simple.

CASRE's major functions are

♦ *Data modification.* CASRE lets you create new failure-data files, modify existing files, and perform global operations on files. You can also select appropriate smoothing techniques or apply data transformations to the failure data being analyzed. You can plot the modified input data, use it as input to a reliability model, or write it to a new file for later use.

♦ *Failure-data analysis.* You can display the failure data's summary statistics, including the data's mean, median, and variance and 25- and 75-percentile cutoffs.

♦ *Modeling and measurement.* CASRE has two modeling functions: As Figure 4 shows, you can execute either single component models on a data set or several models and combine their results. Through model evaluation, you can determine how well a

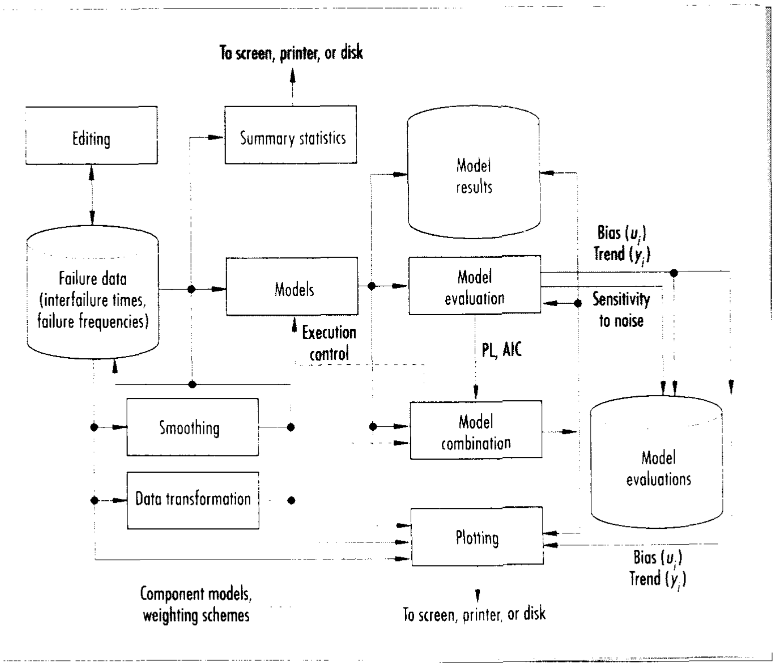
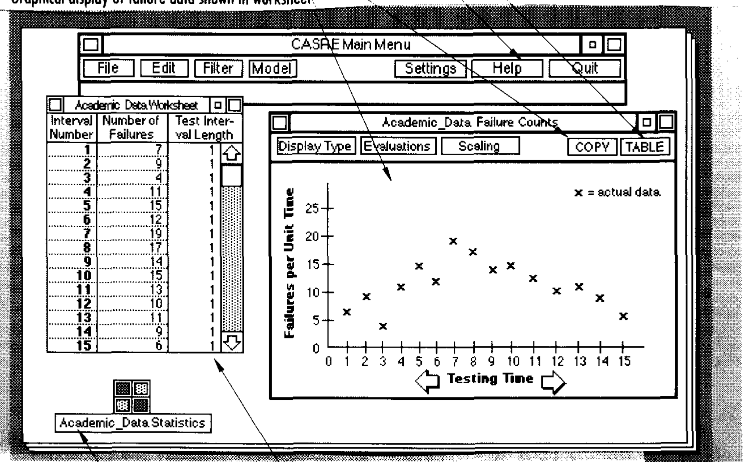


*Figure 4. Architecture of CASRE, a tool to automate the selection of component models to form a combination model. PL is prequential likelihood; AIC is Akaike Information Criterion.*



Academic Data Worksheet

| Interval Number | Number of Failures | Test Interval Length |
|---|---|---|
| 1 | 7 | 1 |
| 2 | 9 | 1 |
| 3 | 4 | 1 |
| 4 | 11 | 1 |
| 5 | 15 | 1 |
| 6 | 12 | 1 |
| 7 | 19 | 1 |
| 8 | 17 | 1 |
| 9 | 14 | 1 |
| 10 | 15 | 1 |
| 11 | 13 | 1 |
| 12 | 10 | 1 |
| 13 | 11 | 1 |
| 14 | 9 | 1 |
| 15 | 6 | 1 |

Displays a new copy of the graph on screen

Press to invoke help system

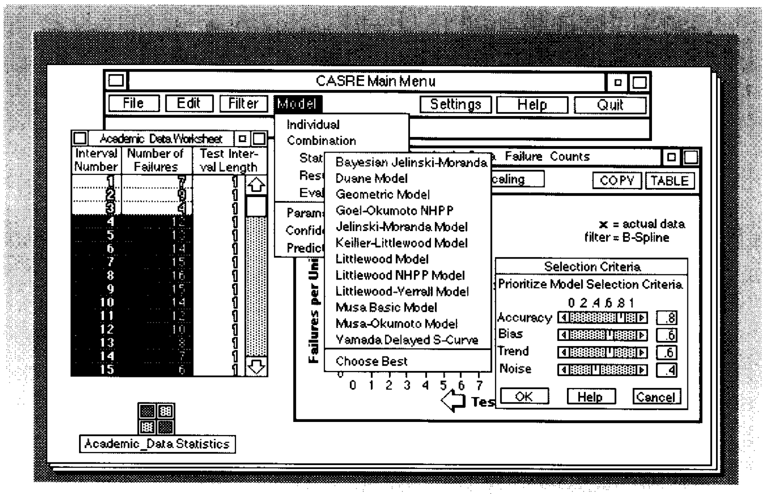
Displays content of graphical display in tabular form

Graphical display of failure data shown in worksheet

Editable, worksheet-like display of failure data

Iconified display of statistics summarizing failure data; comes up in iconified form to minimize screen clutter

*Figure 5. CASRE's initial display of failure data.*



Academic Data Worksheet

| Interval Number | Number of Failures | Test Interval Length |
|---|---|---|
| 1 | 7 | 1 |
| 2 | 9 | 1 |
| 3 | 4 | 1 |
| 4 | 12 | 1 |
| 5 | 13 | 1 |
| 6 | 14 | 1 |
| 7 | 15 | 1 |
| 8 | 16 | 1 |
| 9 | 15 | 1 |
| 10 | 14 | 1 |
| 11 | 12 | 1 |
| 12 | 10 | 1 |
| 13 | 8 | 1 |
| 14 | 7 | 1 |
| 15 | 6 | 1 |

Selection Criteria

Prioritize Model Selection Criteria

0 .2 .4 .6 .8 1

Accuracy .8

Bias .6

Trend .6

Noise .4

*__Figure 6.__ Selecting the best models with CASRE. To specify the criteria by which you will judge a model to be the best, move the slide bars on the Selection Criteria panel (lower right corner) to set the relative weights of four criteria.*

model applies to the data.

♦ *Results display.* CASRE graphically displays model results of interfailure times, cumulative failures, failure intensities, and the reliability-growth curve. You can plot both actual and estimated quantities on the same figure. Plots also include user-specified confidence limits and control over the plotted range of data.

In a windowing environment, you can display multiple plots. You can then either print the plots, save them as a disk file, or feed them to other software, such as a spreadsheet. The plotting function also produces graphics from the model evaluation's output, which indicate the degree and direction of model bias and the way in which the bias changes over time.

The combination models we have proposed show promising results compared with traditional single models. Our approach is also flexible, letting you select models that best suit the failure data. CASRE automates significant portions of the work, making software-reliability measurement even simpler. For instance, CASRE will let users run the combination models we have described just by selecting them from a menu.

Users can also form their own combination models, save them as part of the tool's configuration, and run them in current or subsequent sessions.

We recognize that much more work needs to be done to gain confidence that the combination models consistently outperform component models. We urge you to apply different data sets to these models and to compare resulting predictions across a variety of projects.

We have not addressed how models can more accurately describe software development and testing, although we realize that this area is of increasing concern. Because the detailed information we would require for such an investigation is not available, we decided our work was better confined to evaluating how to use existing models more effectively.

We hope someday to address how to develop models that can more accurately describe software development. ♦

## ACKNOWLEDGMENTS

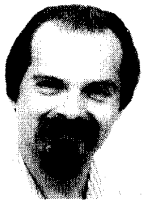
We thank William Farr of the Naval Surface Warfare Center for the use of SMERFS and Bev Littlewood of the City University of London for permission to use SRMP. The research described in this article was done at the University of Iowa under a faculty starting fund and at the Jet Propulsion Laboratory, California Institute of Technology, under a NASA contract through the director's discretionary fund. CASRE's implementation is being supported by the Air Force Operational Test and Evaluation Center under Task Order RE-182, Amendment 655, Proposal 80-3417.

## REFERENCES

1. J. Musa, A. Iannino, and K. Okumoto, *Software Reliability — Measurement, Prediction, Application,* McGraw-Hill, New York, 1987.
2. M. Lyu, "Measuring Reliability of Embedded Software: An Empirical Study with JPL Project Data," *Proc. Int'l Conf. Probabilistic Safety Assessment and Management,* Elsevier, New York, Feb. 1991, pp. 493-500.
3. AIAA Working Group, "Recommended Practice for Software Reliability," American Inst. of Aeronautics and Astronautics, Washington, DC (to appear).
4. A. Dawid, "Statistical Theory: The Prequential Approach," *J. Royal Statistics Soc. A,* Vol. 147, pp. 278-292.
5. A. Abdel-Ghaly, P. Chan, and B. Littlewood, "Evaluation of Competing Software Reliability Predictions," *IEEE Trans. Software Engineering,* Sept. 1986, pp. 950-967.
6. J. Musa, "Software Reliability Data," tech. report, Rome Air Development Center, Griffis AFB, N.Y., 1980.
7. H. Akaike, "Prediction and Entropy," Tech. Report 2397, Mathematics Research Ctr., Univ. of Wisconsin, Madison, 1982.

**Michael R. Lyu** is an assistant professor of electrical and computer engineering at the University of Iowa. His research interests include software engineering, software reliability, fault-tolerant computing, and distributed-systems engineering.

Lyu received a BS in electrical engineering from the National Taiwan University, an MS in electrical and computer engineering from the University of California at Santa Barbara, and a PhD in computer science from the University of California at Los Angeles. He is a vice chair of the subcommittee on software reliability engineering of the IEEE Computer Society's Technical Committee on Software Engineering.



**Allen Nikora** is a member of the Jet Propulsion Laboratory's software product assurance section. His research interests include software-reliability measurement, software safety, and software-system development methodologies.

Nikora received a BS in engineering and applied science from the California Institute of Technology and an MS in computer science from the University of Southern California. He is pursuing a PhD in computer science at USC. He is a member of the IEEE Computer Society and the IEEE Reliability Society.

Address questions about this article to Lyu at ECE Dept., Univ. of Iowa, Iowa City, IA 52242; Internet lyu@eng.uiowa.edu; or Nikora at JPL, M/S 125-233, 4800 Oak Grove Dr., Pasadena, CA 91109; Internet bignuke@spa1.jpl.nasa.gov.