JOURNAL OF AEROSPACE COMPUTING, INFORMATION, AND COMMUNICATION
Vol. 6, May 2009

Updated Software Reliability Metrics

Norman Schneidewind*
Naval Postgraduate School, Pebble Beach, CA 93953

DOI: 10.2514/1.40883

The Institute of Electrical and Electronics Engineers' *Standard Dictionary of Measures of the Software Aspects of Dependability* (982.1) has been scheduled for updating. This standard was issued several years ago. Since then new software reliability metrics have been developed and evaluated. In addition, modifications of metrics in the standard have developed and evaluated. The objective of this paper is to describe, evaluate, and apply the new and modified metrics, using failure data from several releases of the NASA Space Shuttle flight software. Recognizing that users of standards have other applications, the methodology, equations, and prediction plots are explained so that reliability engineers can apply the metrics to their applications. The metrics are assessed from two standpoints: 1) identify metrics that support a specified purpose (e.g., demonstrate reliability growth) and 2) use these metrics to identify software releases that, based on reliability predictions, are ready to deploy and identify which software requires additional testing. Prediction accuracy is computed for all metrics and the metrics are compared based on the results.

I. Objective

Our objective is to introduce new and modified metrics – new in the sense that they were not included in the Institute of Electrical and Electronics Engineers' (IEEE) *Standard Dictionary of Measures of the Software Aspects of Dependability* (982.1) [1]. (In plain English, this standard is about software reliability metrics!) Modifications are made to selected original metrics to enhance their usability. In addition, we examine the justification of assumptions that support the validity of the metrics. Also, as well as the metrics themselves, there are the trends in metrics, which indicate whether reliability growth is being achieved, that we add to the metrics toolkit. We propose that the metrics we describe and analyze be included in the next version of the standard. To assess the validity of predictive metrics, we compute the mean relative error (MRE) between predicted and actual values.

Where predictive reliability metrics are introduced or modified, we use the Schneidewind Software Reliability Model (SSRM) [2]. Other models recommended in the *IEEE/AIAA Recommended Practice on Software Reliability* [3] could be used. In software reliability analysis, there are various time values: failure time, time *when* a prediction is made, time for which a prediction is *made*, etc. We designate all of these as T_i, where i identifies an event (e.g., failure i) or an interval of test or operational time of the software.

To validate the metric computations, we compared C++ program results with Excel computation results for the Shuttle releases [operational increments (OIs)]. The computations were not considered validated until the C++ program and Excel computations matched.

Received 8 September 2008; accepted for publication 15 March 2009.
* Naval Postgraduate School, 2822 Raccoon Trail, Pebble Beach, CA 93953. Ieeelife@yahoo.com.

II. Reliability Metric Assumptions

A. Independence of Successive Failures

Some researchers [4] claim that the assumption of independence of successive software failures in applying software reliability models is inappropriate. Whether this is true depends on the kind of testing that is conducted. Sometimes different test scenarios are grouped according to high-level functionalities and a series of related test runs are conducted. In addition, input data may be chosen to increase the testing effectiveness, that is, to detect as many faults as possible. As a result, once a failure is observed, series of related test runs are conducted to help isolate the cause of failure. This would also be the case during debugging when successive failures could be dependent because debugging is a fine-grained search for specific faults that may be violating the specification. On the other hand, the failure data that drives software reliability models is obtained during system tests of program functions. Given the enormity of the input and program space, it is unlikely that two faults could cause two successive failures to be related [5]. This is particularly the case if random selection of inputs is used in system testing. Rather than make an assumption that may turn out to be erroneous, the data should be subjected to an autocorrelation test, using a statistical package. For example, to test the assumption of independence of successive failures, we computed and plotted the autocorrelation functions for several National Aeronautics and Space Administration (NASA) space systems, using time to next failure T_i. The autocorrelation function is defined in Eq. (1) [6]:

$$\text{Autocorrelation}\ (T_i, \Delta t) = \text{Correlation}(T_i, T_i + \Delta t), \tag{1}$$

where Δt represents the lag between values of T_i. For example, $\Delta t = 1$ represents the series T_i, T_{i+1}, T_{i+2}, etc., $\Delta t = 2$ represents the series T_i, T_{i+3}, T_{i+5}, etc. When computing the autocorrelation function, confidence intervals of the function are produced to see when the function, plotted for various lags, falls outside the intervals. When this is the case, a high degree of correlation is indicated. In Figs. A1 to A4 in the Appendix, for the NASA Space Shuttle flight software there is no significant correlation, which indicates that the independence assumption is justified for these data. However, in the case of the NASA satellite project JM1 in Fig. A5, there is significant autocorrelation for $\Delta t = 1$. Therefore, it would not be appropriate to use a model to predict the series T_i, T_{i+1}, T_{i+2}, etc. As there is no significant correlation at other values of Δt, predictions could be made, for example, of series T_i, T_{i+5}, T_{i+9}, etc.

A word about accounting for the passage of time: in some cases, time is measured at an instance in time, for example, a prediction of time to next failure. In other cases, time is measured in intervals, for example, failures that occur in the interval $(T_{i+1} - T_i)$.

III. New Software Reliability Metrics

A. Time Between Failures Trend

If the trend of a series of time between failures increases, a reliability growth is suggested, as expressed in Eq. (2):

$$M_{i+1} = (T_{i+2} - -T_{i+1}) \quad > \quad M_i = (T_{i+1} - -T_i) \tag{2}$$

B. Trend Analysis

A method is needed to ascertain whether the trend in a series like Eq. (2) indicates reliability growth. One such method by Bates [7] is:

$$U_i = \frac{\sum_{i=1}^{N_i} M_i - ((N_i/2)(T_i))}{T_i\left(\sqrt{\frac{N_i}{12}}\right)} \tag{3}$$

where N_i is the actual cumulative number of failures at interval i, T_i is the time during which the N_i failures occur, and M_i is the series being examined. With $M_i = (T_{i+1} - T_i)$, increasing positive values of U_i indicate reliability growth [5].

C. Predicted Software Reliability

Strangely, software reliability was not included in 982.1. Using SSRM, reliability is predicted in Eq. (4):

$$R(T_i) = \exp(-(\alpha/\beta)\{\exp[-\beta(T_i - s + 1)] - \exp[-\beta(T_i - s + 2)]\}) \tag{4}$$

where α is initial failure rate, β is rate of change of failure rate, T_i is the time for which the prediction is made, and s is the first time interval at which failure data are used in the estimation of parameters α and β.

D. Actual Software Reliability

In addition to predicted reliability, we can compute the actual reliability $R_a(T_i)$, based on failures observed in interval i, x_i, in relation to the total cumulative number of failures observed at interval t, X_t, in Eq. (5):

$$R_a(T_i) = 1 - (x_i/X_t) \tag{5}$$

E. Reliability Required to Meet Mission Duration Requirement

The original 962.1 does provide metrics for meeting the mission duration requirement by predicting the time to next failure and seeing whether the prediction *exceeds* the mission duration. Another approach is to predict the reliability at the mission duration T_m plus mission start, or launch, time T_s (nominally the last test time), and see whether the result meets the required reliability *during* the mission. This is accomplished by reformulating Eq. (4) in Eq. (6).

$$R(T_s + T_m) = \exp[-(\alpha/\beta)(\exp\{-\beta[(T_s + T_m) - s + 1]\} - \exp\{-\beta[(T_s + T_m) - s + 2]\})] \tag{6}$$

F. Rate of Change of Software Reliability

In addition to the predicted software reliability, its rate of change is also important to identify the amount of test or operational time at which the rate of change is maximum. Beyond this time, increases in reliability yield diminishing returns, although additional reliability may be warranted to meet reliability requirements. The rate of change is formulated by differentiating Eq. (4) and is given in Eq. (7):

$$\frac{\mathrm{d}[R(T_i)]}{\mathrm{d}(T_i)} = \alpha R(T_i)\exp[-\beta(T_i - s + 1)] - \exp[-\beta(T_i - s + 2)] \tag{7}$$

G. Parameter Ratio

In [3] it has been demonstrated that the parameter ratio (PR) $= \beta/\alpha$ from SSRM can accurately rank the reliability of a set of software modules or releases, *before* extended effort is involved in making reliability predictions, by just using the result of parameter estimates. That is, increasing values of PR are associated with increasing values of reliability. The reason is – referring to the definitions above – high values of β mean that the failure rate decreases rapidly and small values of α mean that the failure rate decreases from a low starting value. The two parameters in concert, computed in PR, leads to increasing reliability, as can be seen by examining Eq. (4).

H. Software Restoration Time

When software fails and the fault that was the culprit is corrected, the question is 'How long will it take to restore the system to the specified reliability?'[†] This metric was suggested by Harold Williams, Editor of *The R & M Engineering Journal*, American Society for Quality. This metric can be obtained by solving Eq. (4) for T_i – the restoration time – specifying $R(T_i)$ as the required reliability when the system has been restored. The result is Eq. (8):

$$T_i = \left(-\frac{1}{\beta}\right)\log\left\{\frac{-\log[R(T_i)\beta]}{\alpha[1 - \exp(-\beta)]}\right\} + (s - 1) \tag{8}$$

I. Predicted Cumulative Failures

In 982.1, there is no prediction of cumulative failure, which is a fundamental reliability growth measure [i.e., $F(T_i)$ will increase at a decreasing rate if reliability growth is present). Therefore, cumulative failures are predicted

[†] Email correspondence with the author.

(using SSRM) [2] in Eq. (9).

$$F(T_i) = \left(\frac{\alpha}{\beta}\right)\{1 - \exp[-\beta(T_i - s + 1)]\} + X_{s-1}, \tag{9}$$

where $F(T_i)$ uses the definitions: T_i is the time when $F(T_i)$ failures are predicted to occur and X_{s-1} is the observed failure count in the range $(s - 1, T_i)$.

J. Fault Correction Rate and Delay

Our approach to fault correction prediction is to relate it to failure prediction, introducing a delay, dT, between failure detection and the completion of fault correction (i.e., fault correction time) [8]. We assume that the rate of fault correction is proportional to the rate of failure detection. In other words, we assume that fault correction keeps up with failure detection, except for the delay $d(T_i)$ in correcting fault i. If this assumption is not met in practice, the model will underestimate the remaining faults in the code. Thus, the model provides a lower bound on remaining faults (i.e., the remaining faults would be no less than the prediction). Using this assumption, the cumulative number of faults corrected by time T_i, N_{ci}, would have the same form as the cumulative number of failures $F(T_i)$ that have been detected by time T_i, but delayed by the interval $d(T_i)$. The fault correction rate for fault i is modeled in Eq. (10), where x_i is the number of faults corrected in interval i:

$$c_i = \frac{x_i}{(T_{i+1} - T_i)} \tag{10}$$

We use a random variable to model the delay $d(T_i)$. For the Space Shuttle, $d(T_i)$ was found to be exponentially distributed with mean fault correction time $1/m_i$, where m_i is the mean fault correction rate in interval i in Eq. (11).

$$m_i = \sum_{i=1}^{i} \left\{ \left[\frac{x_i}{(T_{i+1} - T_i)} \right] \middle/ N_{ci} \right\} \tag{11}$$

This distribution was confirmed for the Shuttle, using a sample of 85 fault correction times and the Kolmogorov–Smirnof test, resulting in $p = 0$. In addition, Musa found that failure correction times were exponentially distributed for 178 failure corrections [5].

The great variability in fault correction time that we found in both the Shuttle and Goddard Space Flight Center data means we emphasize predicting limits instead of expected values. For a given mean fault correction rate m_i, the cumulative probability distribution $F(dT_i)$ of the fault correction delay dT_i is used to specify an upper limit of $d(T_i)$. The concept is to bound the delay time, for example at $F(dT_i) = 0.99$, and to use this limit in the fault correction delay prediction. Thus, when making predictions, there would be high confidence that the actual delay is within the limit (e.g., probability of 0.01). The equation for $F(dT_i)$ for the cumulative exponential distribution, when using m_i, computed in Eq. (11) is:

$$F(dT_i) = 1 - \exp[-(m_i)(dT_i)] \tag{12}$$

Eq. (12) is manipulated to produce Eq. (13), which is used to compute the limit of $d(T_i)$, using the specified limit $F(dT_i)$:

$$d(T_i) = \frac{\{-\log[1 - F(dT_i)]\}}{m_i} \tag{13}$$

K. Cumulative Number of Faults Corrected

Knowing the correction rate for fault i from Eq. (10) and the time between failures from Eq. (2), assuming these times are equal to the times between faults, we can predict the cumulative number of faults corrected, N_{ci}, at interval i in Eq. (14) [8].

$$N_{ci} = \sum_{i=1}^{N_{ci}-1} [c_i(T_{i+1} - T_i)] \tag{14}$$

L. Proportion of Faults Corrected

Now having predicted the number of cumulative faults corrected in Eq. (14) and using the cumulative number of actual failures N_i, observed at interval i, and assuming the number of faults equals the number of failures, we compute the proportion of faults corrected at interval i in Eq. (15) [8]:

$$P_{ci} = \frac{N_{ci}}{N_i} \tag{15}$$

M. Predicted Failure Rate

It is important to have a prediction of failure rate that can take into account *future* test or operational time, for example, mission duration. The predicted failure rate is the derivative of the predicted cumulative failures in Eq. (9). The result is Eq. (16) [9]:

$$f(T_i) = \frac{\mathrm{d}[F(T_i)]}{\mathrm{d}(T_i)} = \alpha \exp\{\exp[-\beta(T_i - s + 1)]\} \tag{16}$$

N. Predicted Number of Failures in Interval *i*

In addition to predicting cumulative failures, which aggregates failure count, a fine-grain prediction can be applied to the interval i. With this prediction, failures can be tracked from interval to interval to see whether any anomalies occur. Using SSRM [2], the prediction is made in Eq. (17):

$$m(T_i) = \left(\frac{\alpha}{\beta}\right)\{\exp[-\beta(T_i - s + 1)] - \exp[-\beta(T_{i+1} - s + 1)]\} \tag{17}$$

O. Predicted Normalized Number of Failures in Interval *i*

Although Eq. (17) is very useful as a predictor of software quality, large values of $m(T_i)$ could simply be the result of large programs producing large numbers of failures! Therefore, we can normalize $m(T_i)$ by the size of the program S, in thousand lines of code (KLOC), as shown in Eq. (18).

$$M(T_i) = \frac{m(T_i)}{S} \tag{18}$$

P. Predicted Maximum Number of Failures (at $T_i = \infty$)

It is important to predict the number of failures over the life of the software. Software is crucial to the economy and infrastructure of a nation, so it is seldom discarded. Rather, it is maintained and upgraded. Thus, the prediction of total number of failures over the life of the software is highly relevant. To ensure that we have a conservative prediction of this metric, infinity is used as its life in Eq. (9), which results in Eq. (19).

$$F(\infty) = \left(\frac{\alpha}{\beta}\right) + X_{s-1} \tag{19}$$

Q. Predicted Maximum Number of Remaining Failures

Additionally, the predicted maximum number of remaining failures is an excellent indicator of residual faults and failures that remain after testing is complete. This metric is computed by subtracting the cumulative number of failures X_t observed at the previous test time t, from Eq. (19). This is done in Eq. (20).

$$RF(T) = \left(\frac{\alpha}{\beta}\right) + X_{s-1} - X_t \tag{20}$$

R. Predicted Operational Quality

According to the former manager of the Shuttle flight software development, predicted operational reliability (1 – fraction remaining failures) is an excellent managerial tool for assessing the overall quality of the software because it indicates – on a fractional (percentage) basis – the extent of fault and failure removal [10]. This metric is computed by using Eqs. (19) and (20), which results in Eq. (21):

$$Q(t) = 1 - \left[\frac{RF(t)}{F(\infty)}\right] \tag{21}$$

S. Probability of x_i Failures

It is time now to address the probability that failures will occur because this metric provides the software developer with a measure of risk of operating the software. Most failure processes during test fit the Poisson process [5]. Thus, the probability of x_i failures occurring during interval i is formulated:

$$P(x_i) = \frac{[(m_i)x_i \exp(-m_i)]}{x_i!} \tag{22}$$

where m_i is the mean number of failures in interval i, computed as a cumulative value:

$$m_i = \frac{x_i}{\sum_{i=1}^{i} x_i} \tag{23}$$

The reason for computing Eq. (23) as shown, rather that summing to the total number of failures, is that the latter quantity would not be known at the time of making the computation. Thus, we sum to the last known interval i.

T. Predicted Number of Faults Remaining

Once the *maximum number of failures over the life of the software* and the *cumulative number of faults corrected* have been predicted, the *number of faults remaining to be corrected at interval* i can be predicted using Eq. (24), assuming one-to-one correspondence between faults and failures. To make the prediction, we call upon Eq. (14) (cumulative number of faults corrected, N_{ci}) and Eq. (19) (maximum number of failures, $F(\infty)$:

$$R_{ci} = F(\infty) - N_{ci} \tag{24}$$

U. Predicted Fault Correction Quality

Then, having predicted the *number of faults remaining to be corrected* in Eq. (24), the *fault correction quality* at interval i can be predicted in Eq. (25), where higher values correspond to higher fault correction quality:

$$Q_{ci} = 1 - \left[\frac{R_{ci}}{F(\infty)}\right] \tag{25}$$

V. Weighted Failure Severity

Up to this point nothing has been said about failure severity. We have been treating failures as if they were equal in severity. Of course, they are not. In the following formulation, we develop a weighted severity metric for a software release. Designating s_i as the severity of fault i, s_m as the maximum *value* of s_i (*minimum* severity), w_r as the severity weight of software release r, x_i as the number of failures of severity s_i, and N as the number of failures that have occurred on release r, w_r is computed in Eq. (26).

For example, $s_i = 1$, 2, 3, 4, and 5, where $s_i = 1$ is the most severe, and $s_i = 5 = s_m$ is the least severe. The higher the value of w_r, the lower the quality of the software release. Table 1 shows the definition of the failure code used in the computation of weighted failure severity:

$$w_r = \sum_{i=1}^{N} \left(x_i * \left\{ 1 - \left[\frac{(s_i - 1)}{s_m} \right] \right\} \right) \Big/ N \tag{26}$$

Table 2 is a compilation of the definitions of new metrics, as expressed in the above equations.

Table 1 Definition of failure severity code

Code	Definition
s_1	Loss of life or system
s_2	Affects ability to complete mission objectives
s_3	Workaround available, therefore minimal effects on procedures – mission objectives met
s_4	Insignificant violation of requirements or recommended practices, not visible to user in operational use
s_5	Cosmetic issue which should be addressed or tracked for future action, but not necessarily a present problem

Table 2 Definition of new software reliability metrics

Metric	Purpose	Data requirement	Parameter estimates
Time between failures trend, M_i	Demonstrate reliability growth	Failure time, T_i	
Trend analysis, U_i	Demonstrate reliability growth	M_i, T_i; Actual cumulative failures, N_i	
Software reliability, $R(T_i)$	Predict reliability	Prediction time, T_i	Failure rate parameters, α, β, s
Reliability to meet mission duration, $R(t_s + t_m)$	Demonstrate required reliability	Mission start time, t_s; Mission duration, t_m	α, β, s
Actual reliability, $R_a(T_i)$	Assess empirical (i.e., historical) reliability	Number of failures in interval i, x_i; total number of failures at interval t, X_t	
Rate of change of reliability, d[$R(T_i)$]/d(T_i)	Identify T_i where gain in $R(T_i)$ is maximum	$R(T_i)$, T_i	α, β, s
Parameter ratio	Rank reliability of releases		α, β
Software restoration time, T_i	Predict time when software will return to its specified reliability $R(T_i)$	$R(T_i)$	α, β, s
Predicted cumulative failures, $F(T_i)$	Demonstrate reliability growth	T_i	α, β, s; observed failures in the range $1, s-1$, X_{s-1}
Fault correction rate, c_i	Predict rate of fault correction	T_i, T_{i+1}, x_i	
Fault correction delay, $d(T_i)$	Predict delay in correcting faults	Specified limit, $F(\mathrm{d}T_i)$; mean fault correction rate, m_i	
Cumulative number of faults corrected, N_{ci}	Assess progress in fault correction	c_i ;T_i, T_{i+1}	
Proportion of faults corrected, P_{ci}	Identify variation in fault correction by fault i	N_{ci}, cumulative number of failures, N_i	
Predicted failure rate, $f(T_i)$	Predict *future* failure rate	T_i	α, β, s
Predicted number of failures in interval i, $m(T_i)$	Predict failures on fine-grained basis	T_i, T_{i+1}	α, β, s
Predicted normalized number of failures in interval i, $M(T_i)$	Include program size in failure predictions	$m(T_i)$; program size, S	
Predicted maximum number of failures, $F(\infty)$	Predict failures over the life of the software		α, β, X_{s-1}
Predicted maximum number of remaining failures, $RF(t)$	Predict residual failures over the life of the software		α, β, X_{s-1}; number of observed failures, X_t
Predicted operational quality, $Q(t)$	Predict overall quality of software	$RF(t)$, $F(\infty)$	
Probability of x_i failures, $P(x_i)$	Assess risk of operating software	Mean number of failures in interval i, m_i, x_i	
Predicted number of faults remaining, R_{ci}	Determine whether the software is ready to deploy	$F(\infty)$, N_{ci}	
Predicted fault correction quality, Q_{ci}	Assess fault correction process	R_{ci}, $F(\infty)$	

IV. Modified Software Reliability Metrics

A. Actual Mean Time to Failure

Whereas 982.1 computes the mean value of $(T_{i+1} - T_i)$ over the total $(T_{i+1} - T_i)$ and cumulative failure count, we can obtain a refined assessment by computing a value for each failure i, as in Eq. (27):

$$M_a(T_i) = \frac{\sum_{i=1}^{N_i}(T_{i+1} - T_i)}{N_i} \tag{27}$$

where N_i is the number of cumulative failures at failure i.

B. Predicted Mean Time to Failure

The original 982.1 used SSRM to make this prediction. We have found that Eq. (28) will result in less error (MRE) than the original model:

$$M_p(T_i) = \sum_{i=1}^{N_i} \frac{(T_{i+1} - T_i)}{F(T_i)}, \tag{28}$$

where $F(T_i)$ is the predicted cumulative failures from Eq. (9).

Reliability growth would be demonstrated by an increasing $M_a(T_i)$ and $M_p(T_i)$, as a function of test time T_i.

C. Actual Failure Rate

Although 982.1 includes an incremental failure rate computed by dividing incremental failures by incremental test or operational time, we now include an actual failure rate, Eq. (29) designed to demonstrate reliability growth, if it exists:

$$f(x_i, T_i) = \frac{\sum_{i=1}^{i} x_i}{T_i}, \tag{29}$$

where x_i is the failure count in time interval i and T_i is the time when x_i failures have been observed. Thus, it can be seen that Eq. (29) computes the failure rate on a *cumulative* basis.

Table 3 is a compilation of modified software reliability metrics, based on the above equations.

V. Reliability Metrics Prediction Results

In this section we show results from predictions, using the equations that have been presented and selected OIs of the Shuttle. Each subsection is dedicated to the purpose of the metrics, as identified in Tables 2 and 3.

A. Demonstrate Reliability Growth

1. Trend Analysis, U_i

Figure 1 demonstrates reliability growth by virtue of the trend metric U_i increasing in the positive direction. A reliability engineer can use this kind of plot to test for reliability growth, for various reliability data, such as failure count – the technique is not limited to time-to-next failure trend.

Table 3 Definition of modified software reliability metrics

Metric	Purpose	Data requirement	Parameter estimates
Actual mean time to failure, $M_a(T_i)$	Assess empirical (i.e., historical) reliability and demonstrate reliability growth	T_i, T_{i+1}; actual cumulative failures, N_i	Does not apply
Predicted mean time to failure, $M_p(T_i)$	Predict future reliability and demonstrate reliability growth	T_i, T_{i+1}; predicted cumulative failures, $F(T_i)$	Does not apply
Actual failure rate, $f(x_i, T_i)$	Estimate empirical (i.e., historical) failure rate	T_i; number of failures in interval i, x_i	Does not apply

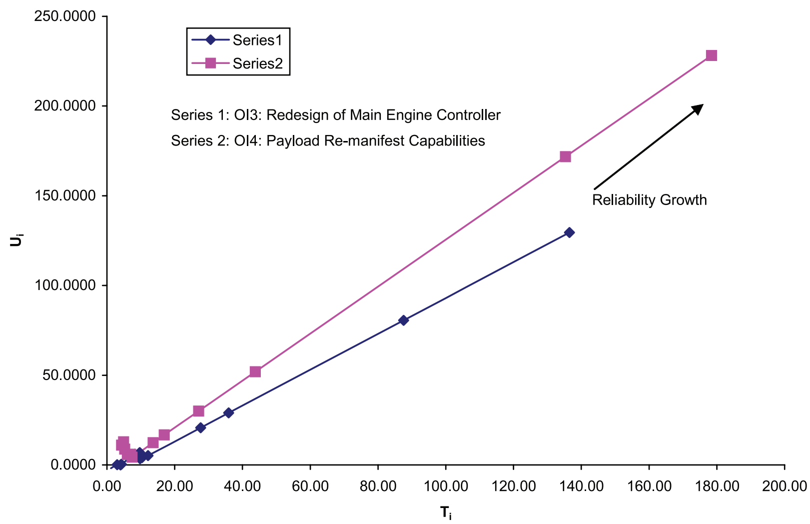

Fig. 1 Time to next failure trend U_i versus test time T_i.

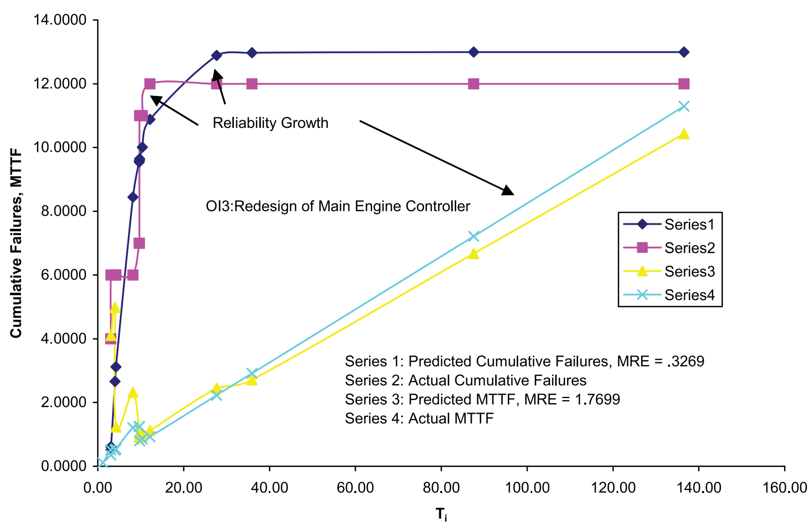

Fig. 2 Shuttle OI3: cumulative failures and MTTF versus test time T_i.

2. Cumulative Failures, $F(T_i)$, Mean Time to Failure

Figure 2 shows reliability growth from another perspective: $F(T_i)$ asymptotically approaches a maximum value as a function of test time T_i and mean time to failure (MTTF) increases monotonically as a function of T_i. We also see that, based on MRE, $F(T_i)$ is a more accurate predictor of reliability than MTTF for *these* data. The practical application of these plots is see whether $F(T_i)$ and MTTF behave as shown in Fig. 2. If they do not, the software development process should be investigated to determine the cause of excessive faults.

B. Predict Reliability, Demonstrate Required Reliability and Predict Reliability Restoration Time

1. Reliability, $R(T_i)$, Reliability to Meet Mission Duration, $R(T_s + T_m)$, Rate of Change of Reliability $d[R(T_i)]/d(T_i)]$ Reliability Restoration Time, T_i

It is important to predict reliability and to predict the reliability that would be achieved for the duration of the mission. For this purpose, the mission duration relevant for a given application should be used. In Fig. 3, $T_m = 0.50$

(0.50 months or 15 days for a typical Shuttle mission is used). Furthermore, it is of interest to identify the test time at which maximum reliability is achieved. In Fig. 3, this is accomplished by plotting the rate of change of reliability. This test time corresponds to the point of maximum payoff of reliability versus test time (i.e., cost). Of course, additional test time may be required to achieve the required reliability, but at diminishing returns to the investment in testing. If the required reliability for the mission duration is not achieved, the software should be subjected to additional testing to eliminate more faults.

If the specified reliability has been temporarily violated because of software failure, the restoration time – the operational time necessary to restore software reliability to its required value – can be predicted. In Fig. 4, we show the restoration time as a function of specified reliability for several OIs. As Fig. 4 shows, latter releases require more restoration time. This is probably caused by the increasing functional complexity of the software across releases, which reflects that each Shuttle release contains all the functionality of previous releases plus the added functionality

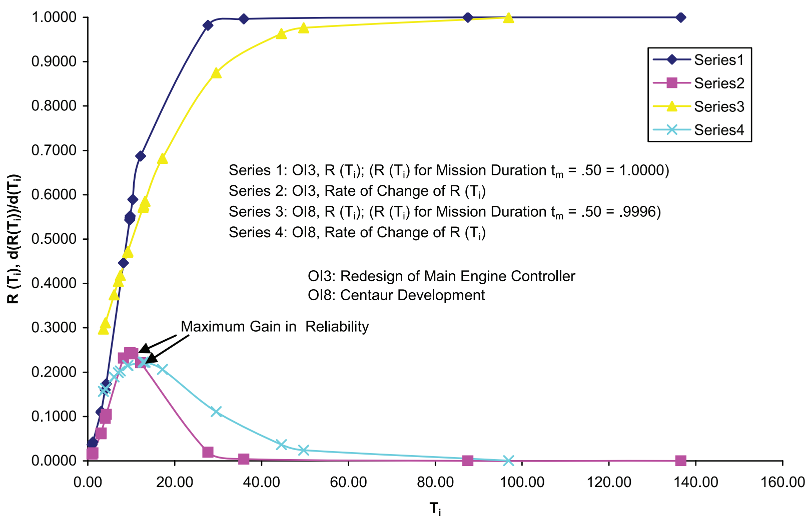

Fig. 3 Reliability $R(T_i)$ and rate of change of reliability $d[R(T_i)]/d(T_i)$ versus test time T_i.

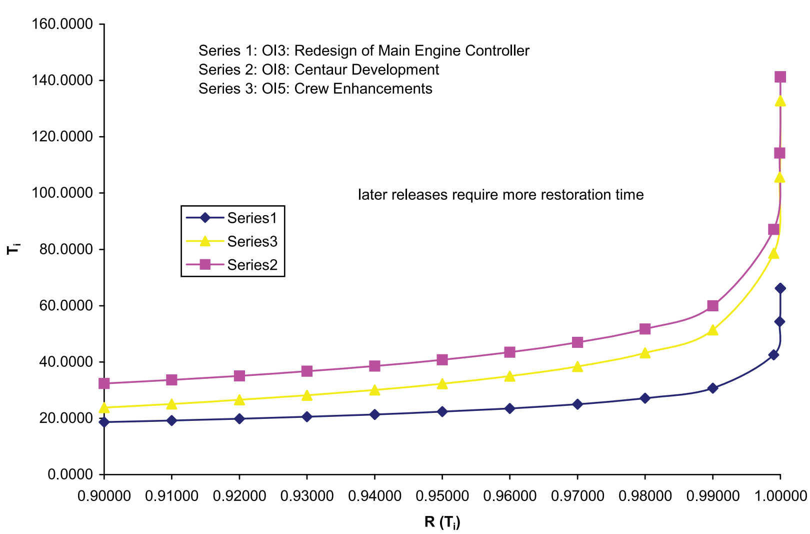

Fig. 4 Restoration time T_i versus reliability $R(T_i)$.

of the current release. This plot would be used to predict whether the restoration time is acceptable for recovering from a failure, based on the specified reliability.

C. Predict and Assess Progress in Fault Correction

1. Fault Correction Rate, c_i, Fault Correction Delay, $d(T_i)$, Proportion of Faults Corrected, P_{ci}

Figure 5 demonstrates that, at the maximum fault-correction rate, the fault correction delay, and the proportion of faults corrected stabilize (i.e., assume constant values) as a function of test time. This is a valuable relationship because a reliability engineer can make these plots and identify the test time in which maximum progress is being made in correcting faults.

2. Predicted Number of Faults Remaining, R_{ci}

Figure 6 indicates that the greater functional complexity of later releases means the predicted number of remaining faults is higher. This plot is useful for indicating whether the software should be deployed on a mission. If the remaining faults are too high, as in the case of OI5, additional testing should be conducted until the remaining faults are predicted to be acceptable (e.g., $R_{ci} = 1$).

3. Predicted Fault Correction Quality, Q_{ci}

In Fig. 7, the predicted fault correction quality provides an overall assessment of the fault-correction process. This metric increases as the number of faults remaining to be corrected decreases. If this plot does not asymptotically approach a maximum with increasing test time, this would indicate an unstable correction process. In this case, the cause of the problem would be identified and corrected, such as inadequate test cases. We observe that correction quality becomes worse in a later release because of the aforementioned increase in functional complexity.

D. Perform Fine-grained Reliability Analysis

1. Predicted Number of Failures in Interval i, $m(T_i)$, Predicted Normalized Number of Failures in Interval i, $M(T_i)$

While reliability metrics based on cumulative values, such as cumulative failures, are useful for demonstrating reliability growth, they do not provide a focused prediction of reliability in each test time interval i. For this purpose, we use $m(T_i)$. Now, while useful, $m(T_i)$ does not account for the size of the software. Therefore, a companion prediction is the normalized failures in the interval i, $M(T_i)$.

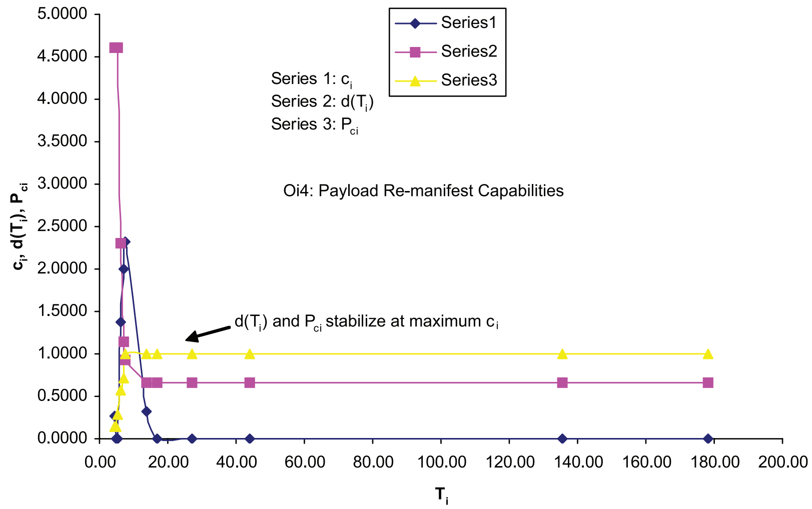

Fig. 5 OI4: Fault correction rate c_i, fault correction delay $d(T_i)$, and proportion of faults corrected P_{ci} versus test time T_i.

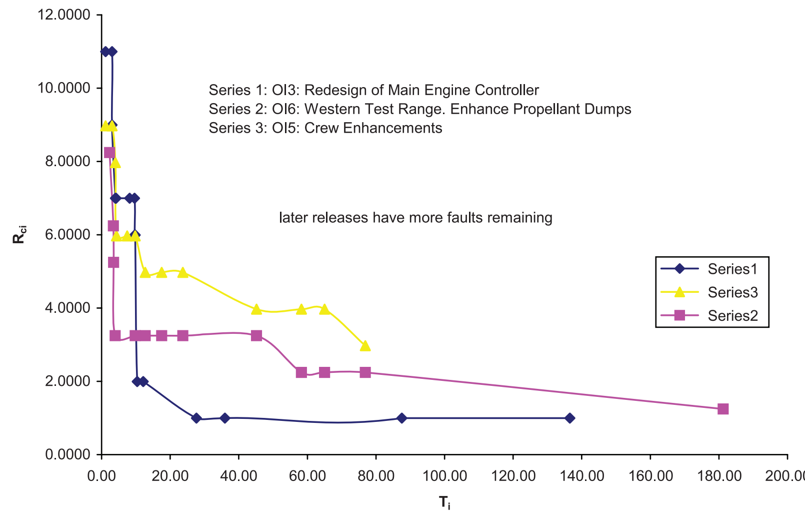

Fig. 6 Number of faults remaining to be corrected R_{ci} versus test time T_i.

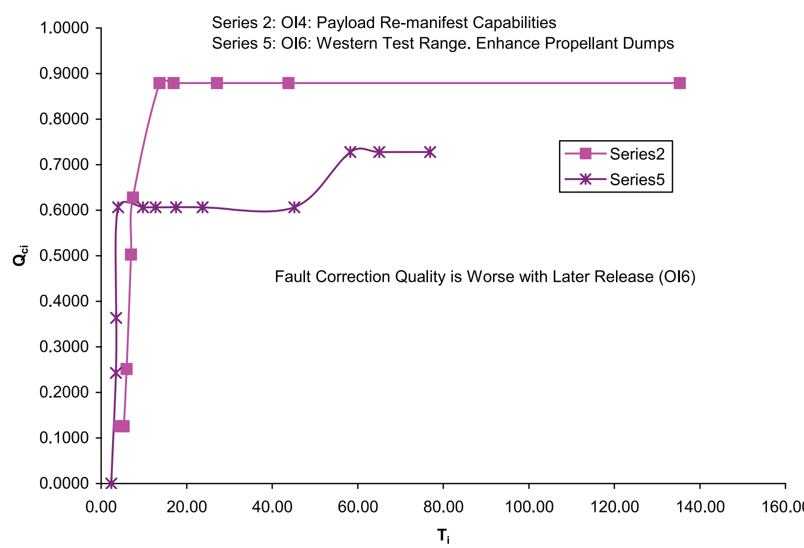

Fig. 7 Fault correction quality Q_{ci} versus test time T_i.

Figure 8 demonstrates that normalized predicted failures in interval i can vary considerably as a function of test time. The main point of the variation from a reliability standpoint is the test time when $M(T_i)$ begins to stabilize (i.e., decreases towards zero). The increased functionality of OI8 versus OI3 means OI8 stabilizes later in the test time. This plot is a tool in the arsenal of the reliability engineer for identifying the test time at which it is no longer cost-effective to continue testing. These times are $T_i = 30$ and 45 for OI3 and OI8, respectively.

2. Actual Failure Rate, $f(x_i, T_i)$, Predicted Failure Rate, $f(T_i)$

Other metrics that allow us to focus on fine-grained analysis are actual and predicted failure rate, with the distinction that the former estimates failure rate based on empirical (i.e., historical) failure data and the latter predicts *future* failure rate based on estimating model parameters, using empirical data. We do not have the "future" available for comparing the results produced by the two metrics, so we necessarily compute MRE over the empirical failure data

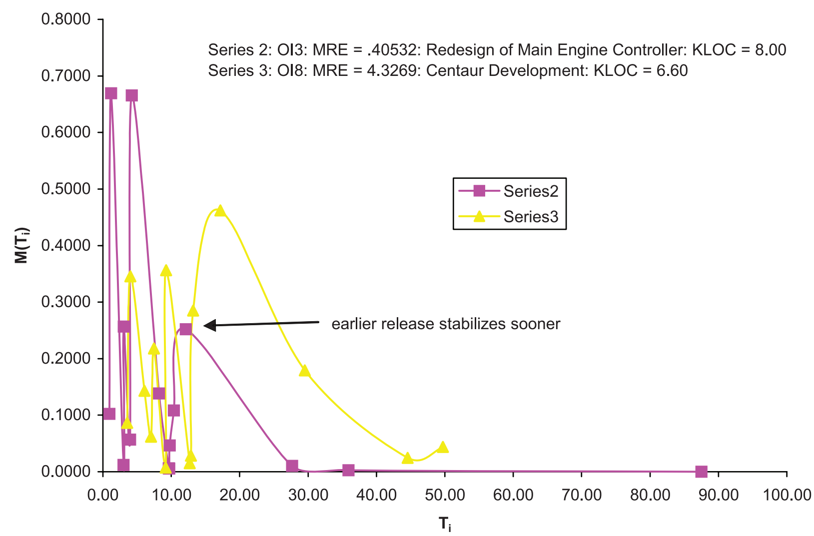

Fig. 8 Normalized predicted failures in interval $iM(T_i)$ versus test time T_i.

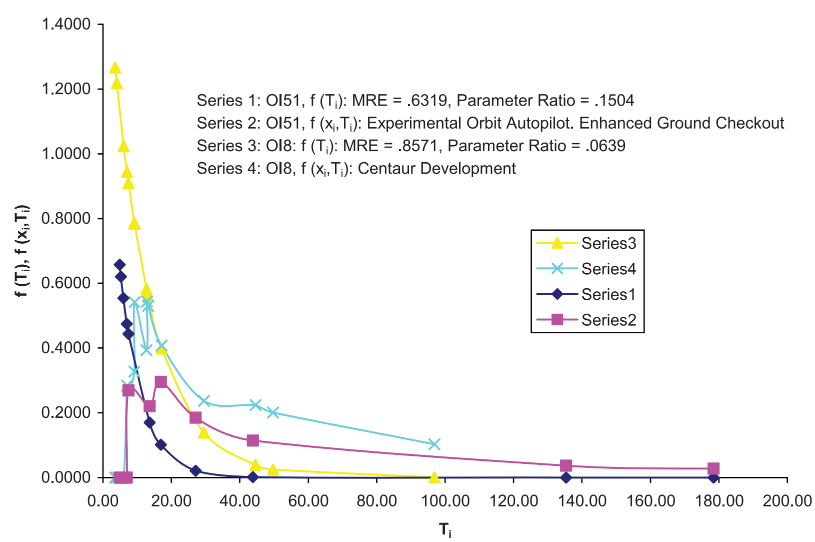

Fig. 9 predicted failure rate $f(T_i)$ and actual failure rate $f(x_i, T_i)$ versus test time T_i.

range. These values of MRE are shown on Fig. 9, which indicates that OI51 has better prediction accuracy than OI8. Also shown is PR, the higher values of which are associated with higher values of reliability (i.e., lower failure rate; see above). This is the case in Fig. 9 where OI51 has both lower predicted and lower actual failure rates than OI8.

An important application of these plots is that, for both OIs, the predictions monotonically decrease, but this is not the case for the actual rates. In the latter case, the failure rate can temporarily increase and then decrease, which reflects changes to the software that are made over test time. Interestingly, this phenomenon is accounted for in the Yamada S-shaped model that allows for an increase in failure rate [11]. If the reliability engineer has an application with this failure data characteristic, the Yamada model could be used, which is described in [3].

E. Assess Reliability Risk

1. *Probability of x_i Failures in Interval i, $P(x_i)$*

To assess the risk to reliability of the incidence of failures, we predict the probability of failures occurring in test time interval i. Of course, the threat to reliability would occur in *operational time* and not in test time. However, the idea of testing is to emulate, to the extent feasible, operational conditions. Therefore, with respect to realistic operational testing in Fig. 10, the software would not be released for operational use at test time $T_i = 4.90$. Rather, we would continue testing until there are no longer any spikes in the plot, at which time the software could be deployed.

F. Track Number of Faults Corrected

1. *Cumulative Number of Faults Corrected, N_{ci}*

By tracking the cumulative number of faults corrected over test time we can determine whether this function reaches a maximum asymptotic value early or late in test time. The former is preferable because it indicates an

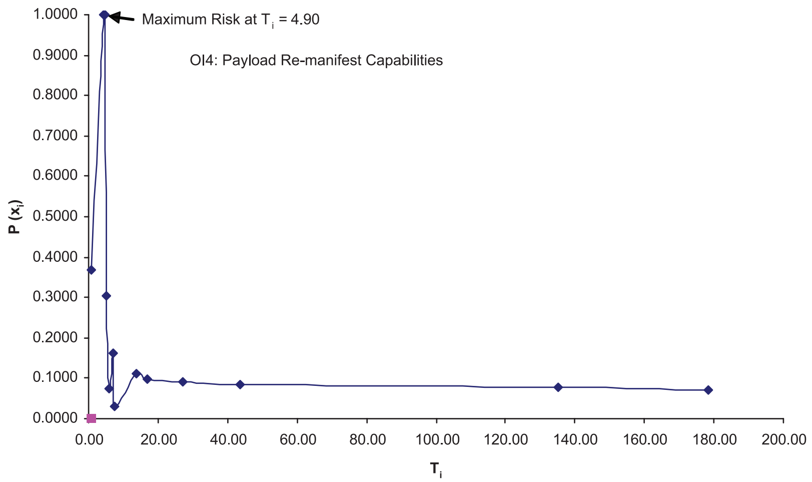

Fig. 10 OI4: Probability of x_i failures in interval i $P(x_i)$ versus test time T_i.

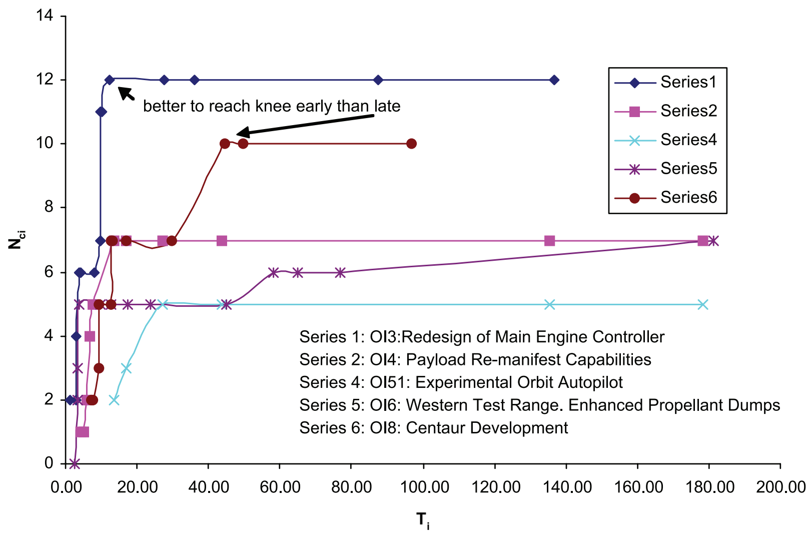

Fig. 11 Predicted number of faults corrected N_{ci} versus test time T_i.

Table 4 Summary of software reliability metrics attributes

Metric	Operational increment					
	OI3	OI4	OI5	OI51	OI6	OI8
	Value and MRE (relative to actual value)					
Weighted failure severity, w_r	0.7000	**0.6000**	0.6750	**0.6000**	0.7429	0.7200
Reliability, $R(T_i)$, standard deviation	0.3634 MRE = 4.4606	0.2219 MRE = *0.3329*	0.2012 MRE = 0.3380	**0.1930** MRE = ***0.3054***	0.2302 MRE = 0.3268	0.2461 MRE = 0.8652
Reliability to meet mission duration, $R(t_s + t_m)$, $t_m = 0.50$	**1.0000**	**1.0000**	0.9989	**1.0000**	**1.0000**	0.9996
Maximum rate of change of reliability, $d[R(T_i)]/d(T_i)$	**0.2433**	0.2216	0.1699	0.2092	0.2124	0.2228
Parameter ratio	0.1430	0.1437	0.1115	**0.1504**	0.1213	0.0639
Software restoration time, T_i, standard deviation	19.7805	17.7343	39.7789	**8.7628**	25.0460	43.1555
Predicted cumulative failures, $F(T_i)$, standard deviation	6.3457 MRE = *0.3269*	2.1980 MRE = 0.6938	3.4076 MRE = ***0.2719***	**1.7592** MRE = 0.5519	2.5764 MRE = *0.2831*	5.0151 MRE = *0.5125*
Fault correction rate, c_i, maximum	**50.0000**	2.3256	8.6957	0.3226	33.3333	16.6667
Fault correction delay, $d(T_i)$, standard deviation	0.4965	1.7283	0.5969	0.5696	**0.4564**	0.6765

(continued)

Table 4 Summary of software reliability metrics attributes (Continued)

Metric	Operational increment					
	OI3	OI4	OI5	OI51	OI6	OI8
Cumulative number of faults corrected, N_{ci}	See Figure 11					
Proportion of faults corrected, P_{ci}	See Figure 5					
Predicted failure rate, $f(T_i)$, maximum	3.6376 MRE = **0.5874**	1.2129 MRE = 0.8608	0.7645 MRE = 0.6139	**0.6573** MRE = 0.7721	0.8648 MRE = 0.6566	1.2665 MRE = 0.8571
Predicted number of failures in interval i, $m(T_i)$, maximum	5.3530 MRE = 4.0532	3.3322 MRE = 1.4593	1.5659 MRE = 5.1715	**0.5206** MRE = **1.1251**	2.6196 MRE = 310.5204	3.0495 MRE = 4.3269
Predicted normalized number of failures in interval i, $M(T_i)$, maximum	0.6691	0.2923	0.2654	**0.1450**	0.2977	0.4621
Predicted maximum number of failures, $F(\infty)$	12.9952	7.9611	10.9714	**6.6475**	8.2457	15.6395
Predicted maximum number of remaining failures, RF(t)	0.9952	**0.9611**	2.9714	1.6475	1.2457	5.6395
Predicted operational quality, $Q(t)$	**0.9234**	0.8793	0.7292	0.7522	0.8489	0.6394
Probability of x_i failures, $P(x_i)$, minimum	0.0398	0.0712	0.0536	0.2388	0.0536	**0.0067**
Predicted number of faults remaining, R_{ci}	See Figure 6					
Predicted fault correction quality, Q_{ci}	See Figure 7					

accelerated fault correction process. This principle is illustrated in Fig. 11, in which OI3 produces the knee of the plot earlier than OI8. In other words, this is a method for evaluating test effectiveness.

VI. Summary of Reliability Metric Results

We have defined and analyzed a myriad of reliability metrics, so it is necessary to document the major results in Table 4 to identify: 1) the metrics that have the least variability across test time, as measured by the standard deviation; 2) the metrics that have the greatest predictive validity, as measured by MRE; and 3) the releases (OIs) that have the smallest predictive error, as measured by MRE. In some cases it is appropriate to use other measures, such as maximum fault-correction rate. Some metrics do not appear in Table 4. These are metrics that are evaluated better using a plot. In these cases, the reader is referred to the relevant figure. The best values per OI are in bold and the best values per metric are in italics.

Although there is not a great deal of consistency in the results, we conclude that: 1) OI51 is the release with the most consistent "best" metrics (e.g., low standard deviation for reliability) and 2) for metrics where MRE can be computed, the metric predicted cumulative failures has the lowest prediction error. This is because cumulative functions smooth irregularities in the data. A reliability engineer could use this approach to: 1) identify software that is ready to deploy (OI51) and software that is not ready to deploy (OI8) and 2) rank reliability metrics by their predictive validity, using MRE.

VII. Conclusions

We have described and evaluated many metrics. There were no dominant metrics in the set with respect to producing the most desirable prediction (i.e., maximum fault-correction rate). Nor were there dominant metrics with respect to minimum prediction error, although metrics based on aggregated failure values, such as cumulative failure, provided marginally more accurate predictions. In contrast, we were able to identify a software release that generally yielded better predictions and less error than other releases. Based on these results, we conclude that the reliability engineer should evaluate several, if not many, metrics to ensure that those metrics appropriate for a given application can be identified. Furthermore, the evaluated metrics should be used to predict reliability for the various software releases to determine which software is ready to be deployed and which software requires further testing.

Appendix

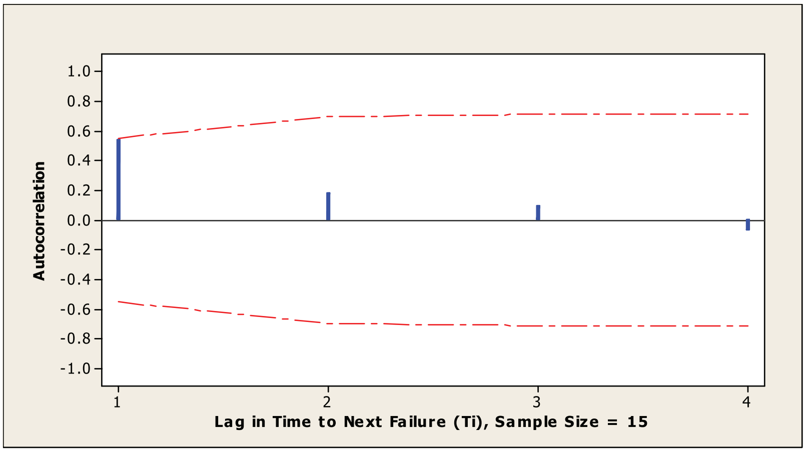

Fig. A1 Shuttle OI3: T_i 5% confidence intervals autocorrelation.

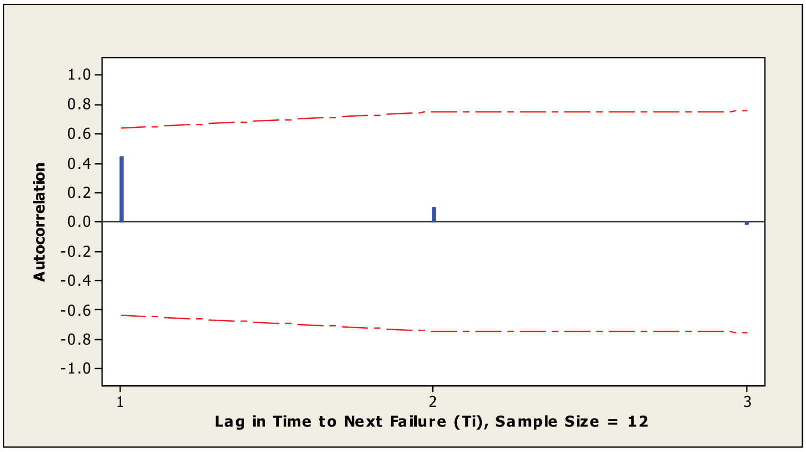

Fig. A2 Shuttle OI4: T_i 5% confidence intervals autocorrelation.

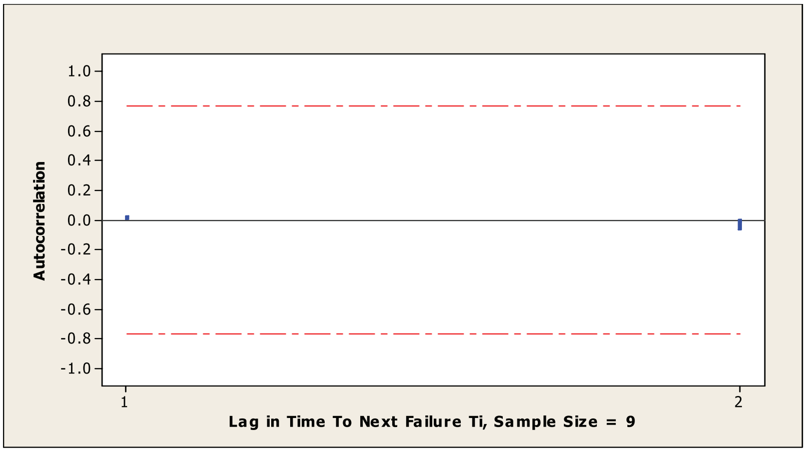

Fig. A3 Shuttle OI5: T_i 5% confidence intervals autocorrelation.

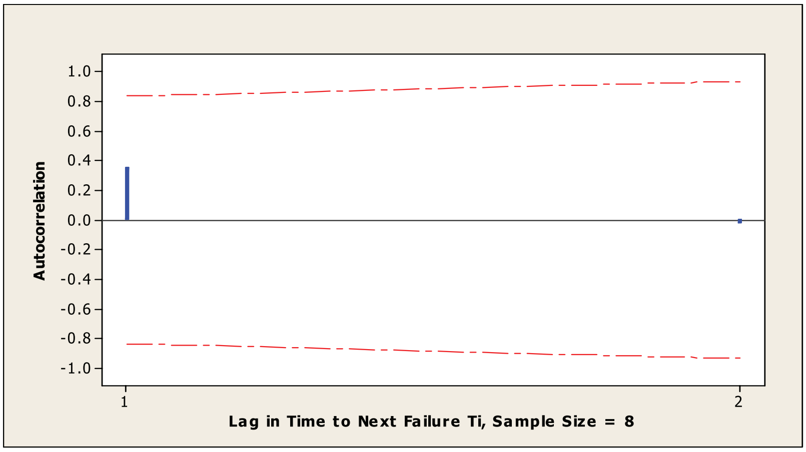

Fig. A4 Shuttle OI6: T_i 5% confidence intervals autocorrelation.

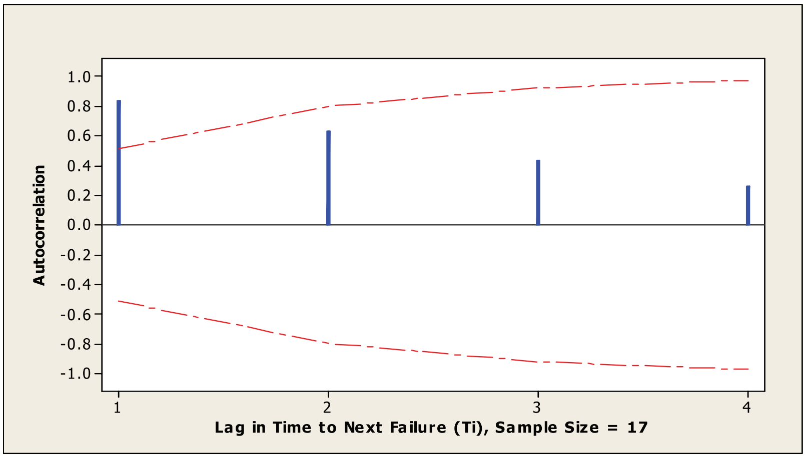

Fig. A5 Satellite JM1: T_i 5% confidence intervals autocorrelation.

References

[1] IEEE, *IEEE Std 982.1 Standard Dictionary of Measures of the Software Aspects of Dependability*, Institute of Electronics and Electrical Engineers, Los Alimatos, CA, May 2006.

[2] Schneidewind, N. F., "Reliability Modeling for Safety Critical Software," *IEEE Transactions on Reliability*, Vol. 46, No. 1, 1997, pp. 88–98.
doi: 10.1109/24.589933

[3] IEEE/AIAA, *IEEE/AIAA P1633™, Recommended Practice on Software Reliability*, Institute of Electronics and Electrical Engineers, Los Alimatos, CA, June 2008.

[4] Goseva-Popstojanova, K., and Trivedi, K., "Failure Correlation in Software Reliability Models," *10th International Symposium on Software Reliability Engineering*, Institute of Electronics and Electrical Engineers, 1999, pp. 232–241.

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

[6] Lyu, M. R. (ed.), *Handbook of Software Reliability Engineering*, IEEE Computer Society Press and McGraw-Hill, New York, NY, 1996.

[7] Bates, G. E., "Joint Distribution of Time Intervals for the Occurrence of Successive Accidents in a Generalized Polya Scheme," *Annals of Mathematical Statistics*, Vol. 26, No. 4, pp. 705–720.
doi: 10.1214/aoms/1177728429

[8] Schneidewind, N. F., "Modelling the Fault Correction Process," *Proceedings of The Twelfth International Symposium on Software Reliability Engineering, Hong Kong*, IEEE Computer Society Press, Washington, DC, 27–30 November 2001, pp. 185–190.

[9] Schneidewind, N. F., "Reliability Modeling for Safety Critical Software," *IEEE Transactions on Reliability*, Vol. 46, No. 1, March 1997, pp. 88–98.

[10] Keller, T., Schneidewind, N. F., and Thornton, P. A, "Predictions for Increasing Confidence in the Reliability of the Space Shuttle Flight Software", *Proceedings of the AIAA Computing in Aerospace 10, San Antonio, TX*, AIAA, Washington, DC, March 28 1995, pp. 1–8.

[11] Min Xie, *Software Reliability Modelling*, World Scientific, Hackensack, 1991.

Michael Hinchey
Associate Editor