JOURNAL OF AEROSPACE COMPUTING, INFORMATION, AND COMMUNICATION
Vol. 7, September 2010

Software Testing and Reliability Strategies

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

DOI: 10.2514/1.49220

Software testing strategies are developed based on fault removal—deterministic and random. Based on literature search, it is appears that this is the first research that has evaluated these strategies. Deterministic fault removal is determined by the ranking of deviations of the time of fault occurrence from statistical control chart upper limit. Random removal of faults is selected by random number generation. Counterintuitively, random fault removal proved to be the better strategy based on improvement in reliability metrics across a series of tests. Contrariwise, deterministic fault removal was better when the cost effectiveness of fault removal was considered. Fault and failure data from three diverse projects were used to make the strategy assessment: NASA Space Shuttle flight software, Japanese University application, and a database application.

I. Introduction

THERE is a need for greater emphasis on fault correction and removal modeling in developing software test strategies and reliability models [1] (fault correction is the process of correcting the error that cased the fault and fault removal means to delete or modify the faulty code). This need stems from the fact that the fault correction process is vital to ensuring high quality software. If we only address failure prediction, strategy and reliability assessment will be incomplete because it would not reflect the reliability of the software resulting from fault correction and removal. The benefits to be achieved with fault correction and removal prediction are the following:

a) Determining whether reliability growth goals have been achieved by predicting number of faults corrected and removed, fault correction rate, and fault correction time.
b) Providing stopping rules for testing as follows: 1) reliability metrics cannot be improved further as testing progresses using various test strategies and 2) the fault correction rate has attained a maximum value.
c) Prioritizing fault removal based on statistical control charts to identify outlier values of fault counts.

In addition, using random selection of faults to prioritize fault removal. We model fault correction with a function that has the same form as the failure detection function but with a random delay that accounts for fault correction time. In practice, it would be possible to predict how much delay in the correction process could be tolerated in order to meet reliability goals at a given time in test or operation [1].

Gokhale et al. have addressed the issue of delayed fault correction, when the delay is caused by the queuing of faults to be removed or by the presence of latent faults that are difficult to remove [2]. They also model the possibility of imperfect fault repair (i.e., a fault may not be entirely corrected or a new fault may be inserted during the repair operation) [2].

They use a non-homogeneous Markov Chain to represent a non-homogeneous Poisson process to model failure detection and fault correction. We agree that it is desirable to model new fault insertion when faults are corrected and

Received 15 February 2010; accepted for publication 10 May 2010.
* Fellow of the IEEE, IEEE Congressional Fellow, US Senate 2005, ieeelife@yahoo.com

removed but only if the model can be validated. We do not have access to this type of data in making our strategy assessments.

A. Objectives

1) Using fault and failure data from the Space Shuttle flight software [3], Japanese University application [4], and a database application [4], we are interested in assessing which of the following two software testing and reliability methods would result in the maximum increase in reliability and cost effectiveness:
   a) correcting and removing faults deterministically based on ranking of deviations from quality control limits.
   b) correcting and removing faults based on random selection of faults.
2) Determine whether there are differences in testing and reliability results based on software project: Space Shuttle flight software, Japanese University application, and database application.

B. Contribution

Many testing strategies are based on coverage of the code of some sort—path, branch, loop, statement [5]. In contrast, our contribution is the development of strategies that are based on fault removal. In our search of the literature, we could not find other research that uses this strategy.

C. Software Projects and Failure Data Sets

The following data sets were used in our research:

Space Shuttle flight software: the software controls ascent, orbit, and decent of the vehicle. The system has 400 K source lines of code that are continuously executed in simulators, astronaut training facilities, and in flight. Time to failure is recorded in days.

Japanese University application: the software system is a compiler project in a university in Japan and the system has about 1000 lines of code. The execution time is in seconds. The system is a PL/1 database application software consisting of approximately 1,317,000 lines of code. The execution time is in hours.

We chose these data sets to provide breadth of applications, so that the analysis results would not be biased towards a particular application domain. The failure data sets are shown in the Appendix. Except for the Shuttle, we do not have severity information for these data. We use severity as one criterion in prioritizing fault removal for the Shuttle, but are unable to include this factor in assessing test strategies for the other applications.

As can be seen in the Appendix, the three projects do not have identical failure characteristics. Therefore, the sequence of fault removals designed to provide different testing strategies, cannot be identical. Thus, results will be compared *within* each project. That is, compare the reliability achieved with deterministic fault removal vs. random fault removal for the Shuttle. Then do the same for the other two projects.

II. Quality Control

The $\bar{x}$-chart statistical quality control chart is used to monitor and control software reliability [6]. The $\bar{x}$-chart provides monitoring and control of variations around the mean difference in time of fault occurrence $\Delta\bar{t}$, as a function of test i, and resultant variations around the mean reliability $\bar{R}$ and other reliability metrics. The difference in t is used rather than t because the latter is a cumulative metric and would have no meaning on a control chart. The upper control limit (UCL) for Δt is computed as:

$$\text{UCL} = \Delta\bar{t} + S \tag{1}$$

where S is the standard deviation. Only one standard deviation is used because the distribution of t is highly skewed. The quality control process is shown in Fig. 1. There are three things to note about how faults are identified for removal. One is that both deterministic fault removal, based on the statistical control chart, and random fault removal are used in Fig. 1. Second is that, of course, in actual testing it would not be feasible to reinsert faults into the code after having removed them in order to initiate random fault removal, however, in assessing testing strategies we can "reset" the code to its initial state of no faults removed in order to invoke random fault removal. Third, faults are removed both deterministically and randomly by identifying the faults associated with the value of $\Delta\bar{t}$ that has been selected either by the UCL or random number.

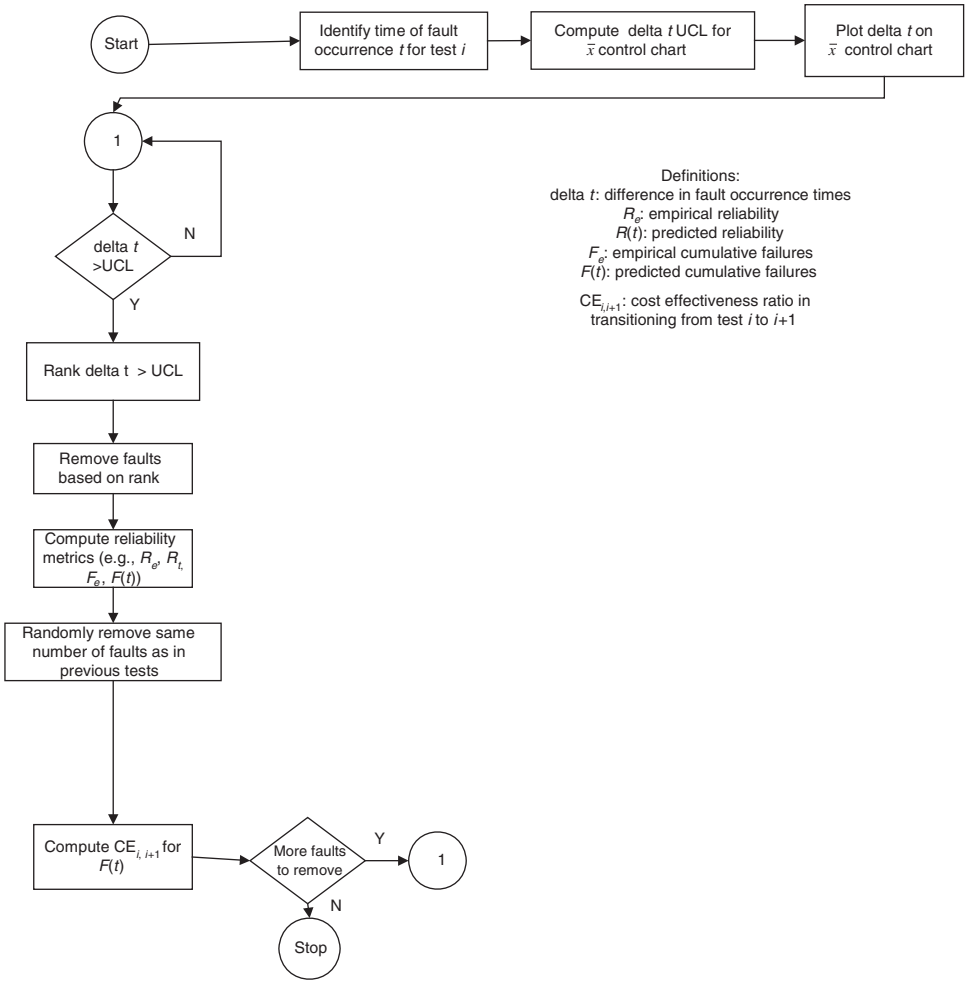

Fig. 1 Software reliability control process.

In the following sections, we discuss a number of metrics—cost and reliability—that are used to assess the efficacy of the test strategies.

III. Cost-Effectiveness Analysis

Cost effectiveness analysis can be used in situations like safety critical applications where the benefits, although real, are difficult to quantify. For example, the reliability of the Shuttle flight software is obviously important for the survival of the mission and the safety of the crew but putting a monetary value on the loss of the Shuttle and crew like life insurance policies may not be appropriate [7]. In the case of software reliability, we will measure "cost" in terms of the difference in test time t from test i to test $i + 1$ required to achieve a change in cumulative failures from $F(t)_i$ to $F(t)_{i+1}$. Then the cost effectiveness ratio is computed as:

$$\mathrm{CE}_{i,i+1} = \frac{t_{i+1} - t_i}{F(t)_i - F(t)_{i+1}} \quad \text{for } F(t)_i - F(t)_{i+1} > 0 \tag{2}$$

This evaluation is made for predicted cumulative failures because the objective is to assess the cost-effectiveness of deterministic vs. random fault removal strategies, as fault removal progresses across a series of tests; $F(t)$ will readily reveal progress in fault removal. Note that the *lower* the CE, the better the fault removal strategy. The reason is that with a low CE, we would achieve a large difference in fault removal for a small investment in difference in

test time. Because there could be considerable variability in CE across test plans, the mean values are computed and compared for the various strategies.

A. Computing the Rate of Return on Software Investment

Even with a safety critical system like the Shuttle, it is possible to apply the principles of engineering economics to compute the rate of return of the investment in software based on the testing strategies we evaluate [8]. Each use of contractor development and test resources involves an investment in making the software safe. We can compute the values of these investments and discount them to the beginning of the development and test process for a software release, like Shuttle release OI3. The rate of return is the value r that equates the timephased investments in test strategies to the initial investment. If C is the initial investment and $c(t)$ is the investment in test time t for a given strategy, r is the rate that satisfies the following equation:

$$C = \frac{c(1)}{(1+r)^1} + \cdots + \frac{c(t)}{(1+r)^t} + \cdots + \frac{c(n)}{(1+r)^n} \tag{3}$$

The values of C and $c(t)$ can be determined from the annual value of the contractor's contract, as we show later.

IV. Reliability Metrics

Several metrics are used in assessing test strategies, including cumulative failures and reliability. For each metric, both empirical and predictive metrics are computed. Cumulative failures are used in Eq. (2) rather than reliability because the latter can have the value 1.0 in going from test i to test $i + 1$. Both empirical and predicted metrics are computed in order to compute prediction error statistics that can be used to assess the accuracy of predictions. Reliability metrics are computed by a C++ program we wrote that cannot be provided due to the paper space limitation.

Empirical cumulative failures are computed by Eq. (4) and empirical reliability by (5).

$$F_e = \sum_{i=1}^{n} x_i \tag{4}$$

where x_i is the number of faults and failures occurring on test i.

$$R_e = 1 - \left(\frac{x_i}{\sum_{i=1}^{n} x_i} \right) \tag{5}$$

Reliability metrics predictions are made using the Schneidewind Software Reliability Model [9], a model recommended in [10]. Other recommended models could also be used.

Cumulative failures are predicted by Eq. (6) and reliability is predicted by (7).

$$F(t) = \frac{\alpha}{\beta}[1 - e^{-\beta(t-s+1)}] + X_{s-1} - \left[\frac{\alpha}{\beta}[e^{-\beta(t-s+1)} - e^{-\beta(t-s+2)}] \right] \tag{6}$$

$$R(t) = e \tag{7}$$

where, in this case, t is the test or operational time for which the prediction is made, α is the initial failure rate, β is the rate of change of failure rate, s is the first failure count interval that is used for estimating α and β, and X_{s-1} is the observed failure count in the range $[1, s-1]$.

Another reliability metric we use to assess test strategies is the mean number of failures in the interval $t, t+1$:

$$m(t) = \left(\frac{\alpha}{\beta} \right) [e^{-\beta(t-s+1)} - e^{-\beta(t-s+2)}] \tag{8}$$

V. Software Reliability Risk

It has been shown in [11] that the software reliability model parameter ratio $PR = \beta/\alpha$ is highly associated with the risk of deploying Shuttle software. By risk we mean that the lower the ratio, the higher the probability that the software will *not* meet expectations in operations. The reason for this result is that a low β means a low rate of change of failure rate over test or operational time, and a high value of α means a high initial failure rate. Therefore, we use this ratio as one of the factors in evaluating the effectiveness of testing strategies. A high ratio across a series of fault removal strategies would imply decreasing risk of software deployment.

VI. Fault Correction 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). We assume that the rate of fault correction is proportional to the rate of failure detection [12]. In other words, we assume that fault correction keeps up with failure detection, except for the delay dT. 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 number of faults corrected at time T, $C(T)$, would have the same form as the number of failures detected at time T, $D(T)$, but delayed by the interval dT. Originally, we used a constant dT, which was estimated from the empirical data [12]. As pointed out by Xie and Zhao, this assumption is too restrictive [13]. He suggests modeling the delay as an increasing function of test time. However, we have not found this to be the case in the Shuttle data, where the fault correction time appears to be primarily a function of the difficulty of the correction and independent of when the correction occurs.

It is well known that various human queue service times (e.g., supermarket checkout stands) can be approximated with an exponential distribution [14], and can be modeled as a birth-death process, where in our case a birth is a detected failure and a death is a corrected fault. Musa et al. uses this type of queuing model in his failure correction process [15]. Therefore, in order to improve the prediction, we use a random variable for the delay dT. For the Shuttle, this variable was found to be exponentially distributed with mean fault correction time $1/m$, where m is the mean fault correction rate in the interval dT. This distribution was confirmed for the Shuttle, using a sample of 85 fault correction times and the Kolmogorov–Smirnof test, resulting in $p = 0$ [1]. In addition, Musa et al. found that failure correction times were exponentially distributed for 178 failure corrections [15]. We have no information about the correction time distribution for the Japanese University and database applications, but we assume the exponential distribution holds in order to provide consistency of analysis across the three systems. We assume that fault correction starts when failures are detected. This assumption is based on the need—for most faults—to keep fault correction and removal current with failure detection. In some cases, a software developer may choose to postpone a non-critical fault correction for several releases because it has obtained a waiver to not make the correction in the current release. We do not attempt to model this human, case-by-case decision process. Our model is based on keeping the software updated with corrections in the current release. In addition, as stated previously, our assessment of test strategies does not include the possibility of introducing a fault when correcting one.

Because the possibility of variability in fault correction times, we emphasize prediction limits instead of expected values. For a given mean fault correction rate m, the cumulative probability distribution $F(\mathrm{dT})$ of the fault correction delay dT is used to specify upper and lower limits of dT. These limits are dTU and dTL, corresponding to the upper limit FU and lower limit FL. The concept is to bound the delay time, for example, at FU $= 0.9$ and FL $= 0.1$, and to use these limits in the fault correction predictions. Thus, when making predictions, there would be high confidence that actual values lie within the limits (e.g., probability of 0.80). The equation for $F(\mathrm{dT})$ for the exponential distribution, is given by (9):

$$F(\mathrm{dT}) = 1 - \exp(-(m)(\mathrm{dT})) \tag{9}$$

Equation (9) is manipulated to produce Eq. (10), which is used to compute the limits of dT, applying the specified limit values of $F(\mathrm{dT})$:

$$\mathrm{dT} = (-\log(1 - F(\mathrm{dT})))/m \tag{10}$$

where m is estimated in Eq. (11), in which x_i is the number of failures detected, corrected, and removed in test i, assuming that the number of failures detected is equal to the number of faults in the code, Δt_i is the difference in test time between test i and test $i + 1$, and N is the number of tests.

$$m = \frac{\sum_{i=1}^{N} (x_i / \Delta t_i)}{N} \tag{11}$$

VII. Fault Correction Rate

The fault correction rate is another useful measure of progress in fault correction and removal. If the rate reaches a maximum, then the strategy corresponding to the maximum is the best strategy [1].

$$C(t) = \alpha[\exp(-[\beta(t - s + l - (\log(1 - F(\mathrm{dT})))/m)])] \tag{12}$$

As in the case of the fault correction delay, limits on $F(\mathrm{dT})$ (e.g., 0.90 and 0.10) are applied to Eq. (12).

VIII. Proportion of Remaining Faults

The predicted proportion of remaining faults, $P(i)$ for type of test i (e.g., deterministically remove two faults), is given in Eq. (13), where F_e is the empirical cumulative failures and $N(i)$ is the number of failures removed by test i [1]:

$$P(i) = (F_e - N(i))/F_e \tag{13}$$

IX. Prediction Error Analysis

Because the error in prediction could influence the evaluation of test strategies, we use the absolute value of the mean relative error (MRE) between predicted and empirical values to make this assessment [5].

X. Results of Applying Test Strategies

In reporting on the results of applying the test strategies, we show the results in the following categories: statistical quality control, predictions and error analysis, cost-effective analysis, risk assessment, rate of return of testing investment, and fault correction rate. In addition to reporting results by topic, results are shown by project. The test strategies are as follows: 1) do not remove faults; 2) based on the ranking, with respect to deviation of the difference in fault occurrence time from the UCL, remove two, four, and five faults; and 3) randomly remove two, four, and five faults.

A. Statistical Quality Control

1. NASA Space Shuttle

To set the stage for evaluating test strategies, in Fig. 2 we show how statistical quality control is used to monitor quality and to identify the high priority faults for removal. Once these faults are removed, the process continues in Fig. 1 with the evaluation of the software sans two faults. Iterative application of the control chart resulted in identifying two, four, and five faults for removal. Then a selection process was applied to the tests to remove the same number of faults randomly.

B. Predictions and Error Analysis

1. NASA Space Shuttle

Ideally, we prefer to have the best test strategy (e.g., highest predicted reliability) associated with the lowest prediction error with respect to empirical reliability. Fortunately this does occur in Fig. 3 where the best strategy and minimum error occur when four failures are removed randomly. A similar result emerges in Fig. 4 where the best strategy and lowest error occur when five faults are removed randomly. Thus, for the Shuttle and reliability predictions, random fault removal emerges as the desirable strategy.

Although Table 1 shows inconsistent results among the strategies for the various metrics, the key metrics are reliability and cumulative failures, wherein random fault removal (bolded) is the best.

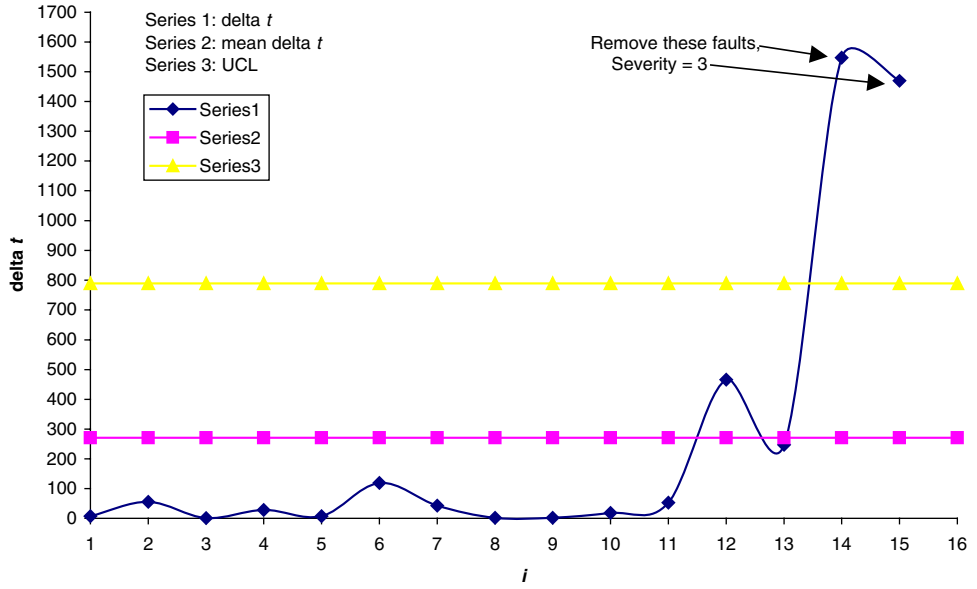

Fig. 2 NASA Space Shuttle OI3 control chart for difference in time of fault occurrence delta t vs. test number i.

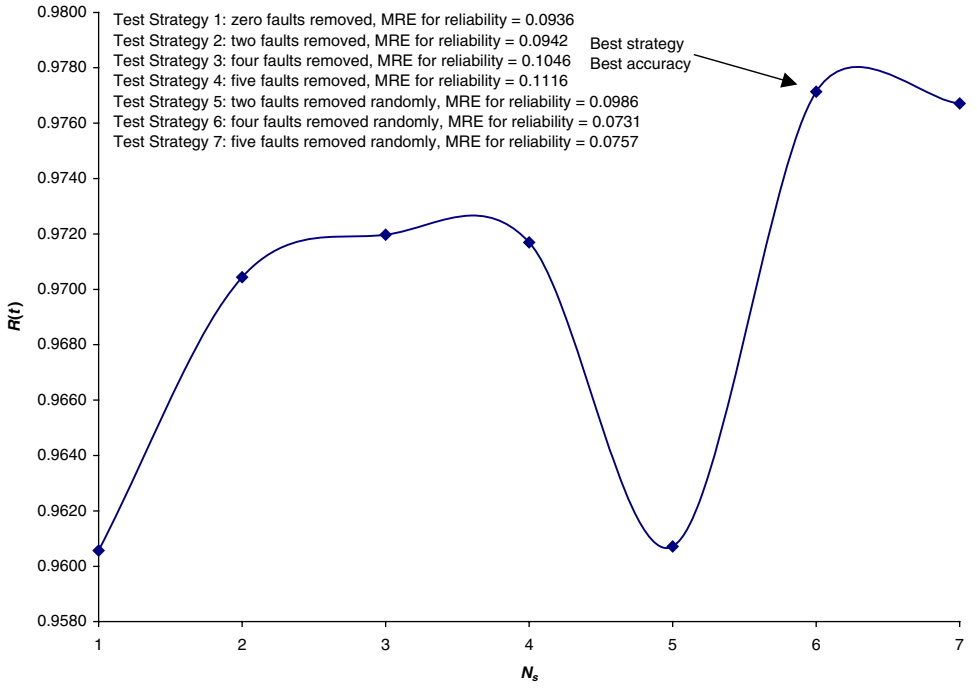

Fig. 3 NASA Space Shuttle OI3 mean of predicted reliability $R(t)$ vs. test strategy N_s.

2. Japanese University System

For this system, the best strategy based on predicted reliability is to remove five faults randomly, as shown in Fig. 5. However, as in the case of the Shuttle, the best strategy is not the lowest error alternative. Because there is not a significant difference in prediction errors among the strategies, it seems best to opt for the best test strategy. A composite view of the results is shown in Table 2, where all of the reliability metrics are documented and the best

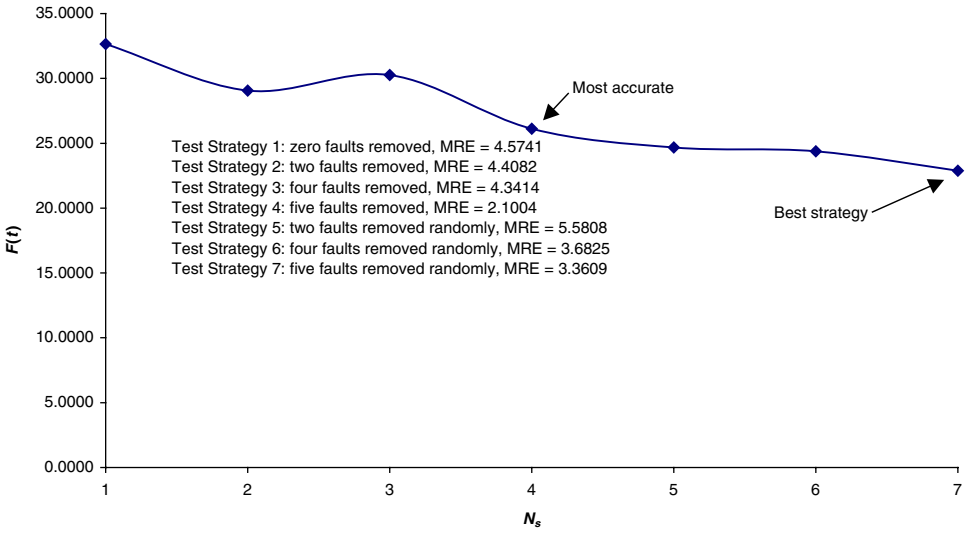

Fig. 4 NASA Space Shuttle OI3 mean of predicted cumulative failures $F(t)$ vs. test strategy N_s.

Table 1 NASA Space Shuttle test strategy results (mean values)

Type of test	$F(t)$	$R(t)$	R_e	$C(t)$ L	$C(t)$ U	m(t)	m	PR	dTU	dTL	$P(i)$
Zero faults removed	31.4092	0.9606	**0.9375**	0.0760	0.0000	0.0468	0.2724	0.0396	8.4530	0.3868	1.0000
Two faults removed	25.8568	0.9704	0.9286	0.0506	0.0000	0.0333	0.3142	0.0518	7.3283	0.3353	0.8182
Four faults removed	23.9680	0.9720	0.9236	0.0441	0.0000	0.0309	0.3708	0.0576	6.2103	0.2842	0.6364
Five faults removed	23.9676	0.9717	0.9091	0.0428	0.0000	0.0309	**0.4061**	0.0576	**5.6694**	**0.2594**	**0.5455**
Two fault removed rand	29.4175	0.9607	0.9333	**0.1449**	0.0000	0.0463	0.1490	0.0396	15.4585	0.7073	0.8182
Four faults removed rand	20.1339	**0.9771**	0.9333	0.0780	0.0000	**0.0249**	0.1490	0.1042	15.4537	0.7071	0.6364
Five faults removed rand	**19.1273**	0.9767	0.9286	0.0428	0.0000	0.0253	0.1601	0.1042	14.3828	0.6581	**0.5455**

Definitions: $F(t)$: cumulative failures, $R(t)$: predicted reliability, R_e: empirical reliability, $C(t)$ L: lower limit of fault correction rate, $C(t)$ U: upper limit of fault correction Rate, $m(t)$: failures in interval t, m: correction rate, PR: parameter ratio, dTU: upper limit of fault correction delay, dTL: lower limit of fault correction delay, $P(i)$: proportion of faults remaining.

values are bolded. The table indicates that there is no one test strategy that is superior for all metrics. For the key metrics of predicted reliability and number of failures in a test interval, removing faults *randomly*, within the feasible test budget, would be a reasonable strategy.

3. Database System

It was not possible to duplicate the same type of tests in this system as was the case for the Shuttle and the Japanese system: remove two, four, and five faults deterministically, followed by removing the same number of faults randomly. The reason is that the database system has faults in large clumps, as can be seen in Table A1 in the Appendix. Therefore, removing one or two faults had a miniscule effect on the relisbility metrics. As a consequence, we were forced to remove a large number of faults for some of the tests as can be seen in Table 3. The results are decidedly mixed with no type of test being dominant. Thus, random tests would be in order to avoid the greater expense of deterministic testing.

C. Cost-Effective Analysis

1. NASA Space Shuttle

Using the predicted cumulative failures $F(t)$ for computing the cost-effectiveness ratio over the test strategies, we plot the mean values of CE in Fig. 6 for the Shuttle. Interestingly, we observe the following: although the previous analyses indicated that randomly removing faults was the best strategy, when viewed from a CE standpoint, removing

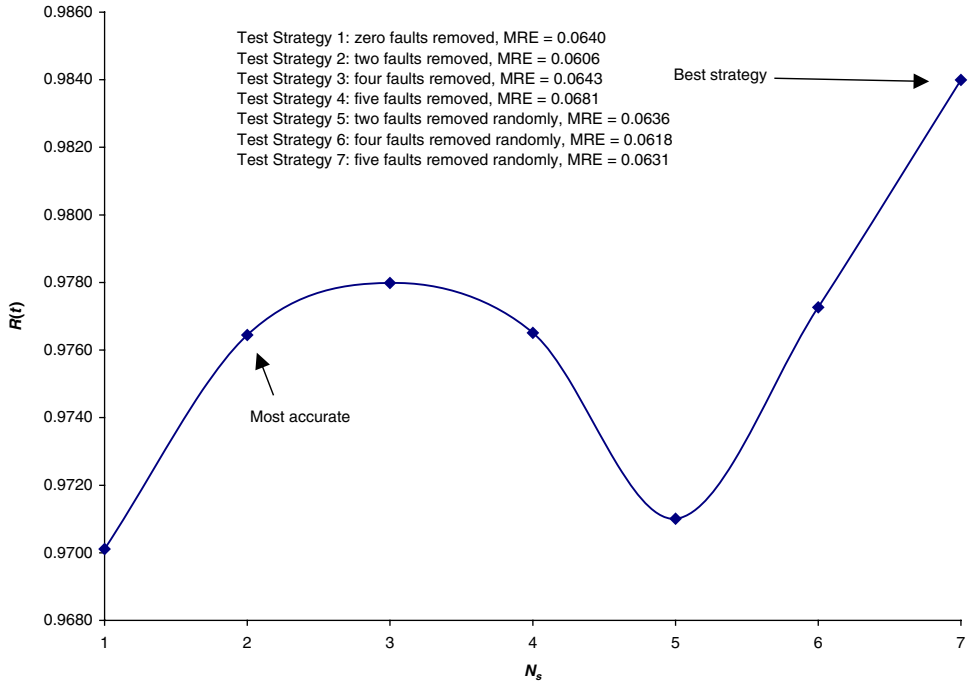

Fig. 5 Japanese University System: predicted reliability $R(t)$ vs. type of test N_s.

Table 2 Japanese test strategy results (mean values)

Type of Test	$F(t)$	$R(t)$	R_e	$C(t)$ L	$C(t)$ U	$m(t)$	m	PR	dTU	dTL	$P(i)$
Zero faults removed	37.9144	0.9701	0.9524	0.0000	0.0638	0.0326	**0.2139**	0.0411	**10.7631**	**0.4925**	1.0000
Two faults removed	35.2651	0.9764	**0.9558**	0.0000	**0.1568**	0.0251	0.1162	**0.0444**	19.8129	0.9066	0.9259
Four faults removed	33.2861	0.9780	0.9412	0.0000	0.1154	0.0232	0.1267	**0.0444**	18.1718	0.8315	0.8519
Five faults removed	33.3730	0.9765	0.9375	0.0000	0.1043	0.0248	0.1332	0.0442	17.2910	0.7912	**0.8148**
Two faults removed rand	35.9289	0.9710	0.9500	0.0000	0.0632	0.0314	0.2076	0.0411	11.0926	0.5076	0.9259
Four faults removed rand	**33.2763**	0.9773	0.9444	0.0000	0.0484	0.0241	0.2093	**0.0444**	11.0033	0.5035	0.8519
Five faults removed rand	32.3559	**0.9840**	0.9412	0.0000	0.0352	**0.0169**	0.2012	**0.0444**	11.4464	0.5238	**0.8148**

Definitions: $F(t)$: cumulative failures, $R(t)$: predicted reliability, R_e: empirical reliability, $C(t)$ L: lower limit of fault correction rate, $C(t)$ U: upper limit of fault correction Rate, $m(t)$: failures in interval t, m: correction rate, PR: parameter ratio, dTU: upper limit of fault correction delay, dTL: lower limit of fault correction delay, $P(i)$: proportion of faults remaining.

Table 3 Database aplication test strategy results (mean values)

Type of Test	$F(t)$	$R(t)$	R_e	$C(t)$ L	$C(t)$ U	$m(t)$	m	PR	dTU	dTL	$P(i)$
Zero faults removed	308.4563	0.2291	**0.9474**	7.1668	7.9244	6.7230	**8.4671**	**0.0100**	**0.2719**	**0.0124**	1.0000
Two faults removed	306.4563	0.2291	**0.9474**	7.1573	7.8212	6.7537	8.4407	**0.0100**	0.2728	0.0125	0.9878
Four faults removed	306.4563	0.2291	**0.9474**	7.1673	7.9304	6.7230	8.4143	**0.0100**	0.2737	0.0125	0.9817
36 faults removed	278.7308	0.2906	0.9444	6.5028	7.2173	6.0976	8.1810	**0.0100**	0.2815	0.0129	0.9024
38 faults removed randomly	**266.5204**	**0.3754**	0.9412	**10.5580**	**11.7472**	**5.9425**	8.0026	0.0078	0.2877	0.0132	**0.8841**

Definitions: $F(t)$: cumulative failures, $R(t)$: predicted reliability, R_e: empirical reliability, $C(t)$ L: lower limit of fault correction rate, $C(t)$ U: upper limit of fault correction Rate, $m(t)$: failures in interval t, m: correction rate, PR: parameter ratio, dTU: upper limit of fault correction delay, dTL: lower limit of fault correction delay, $P(i)$: proportion of faults remaining.

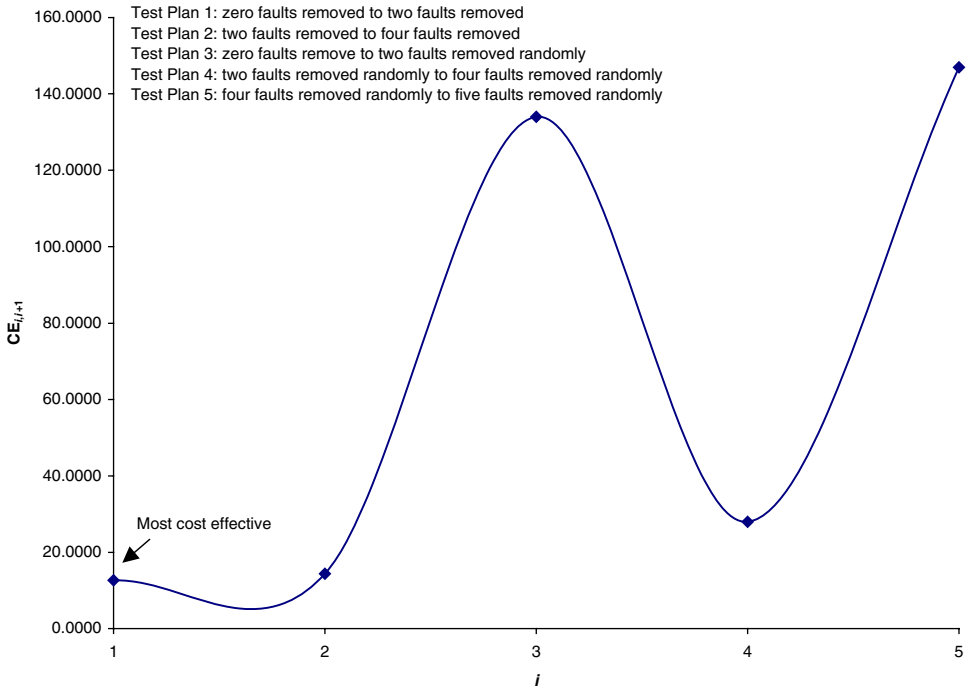

Fig. 6 Mean value of cost effectiveness ratio $CE_{i,i+1}$ vs. test plan i.

two faults deterministically is the best plan. Based on this finding, the development organization should decide which is more important for a test strategy: high reliability (i.e., randomly removing faults) or low CE (i.e., removing faults deterministically).

2. Japanese University System

Similar to the finding in Fig. 6 for the Shuttle, Fig. 7 for the Japanese University System, shows that removing faults deterministically is the most cost effective strategy. Again, an organization needs to make a choice between reliability and cost effectiveness.

3. Database Application

There is insufficient change in cumulative failures across test strategies for this application to develop a trace of CE. However, we did find that the mean CE does progress from 0.1103, when transitioning from four faults removed deterministically to 36 faults removed in this manner, to a mean of 0.0299 when changing from 36 faults removed deterministically to 38 faults removed randomly. Thus, this result is contrary to the previous findings for the other two projects. However, given the small sample size, we do not consider this result significant.

D. Risk Assessment

1. NASA Space Shuttle and Japanese University System

Figure 8 shows that for both the Shuttle and the Japanese system, minimum risk is achieved with randomly removing five faults, albeit this result is more pronounced for the Shuttle. This adds further evidence that, from a reliability and risk standpoint, the preponderance of evidence is on the side of removing faults randomly. However, if the development organization is primarily interested in cost-effectiveness, its choice is to remove faults deterministically (Figs 5 and 7). Of course, the specific number of faults to remove would vary by project and type of software. In addition, in view of the greater effort involved in developing control charts to identify candidates for removal, random selection of faults to remove is a reasonable policy.

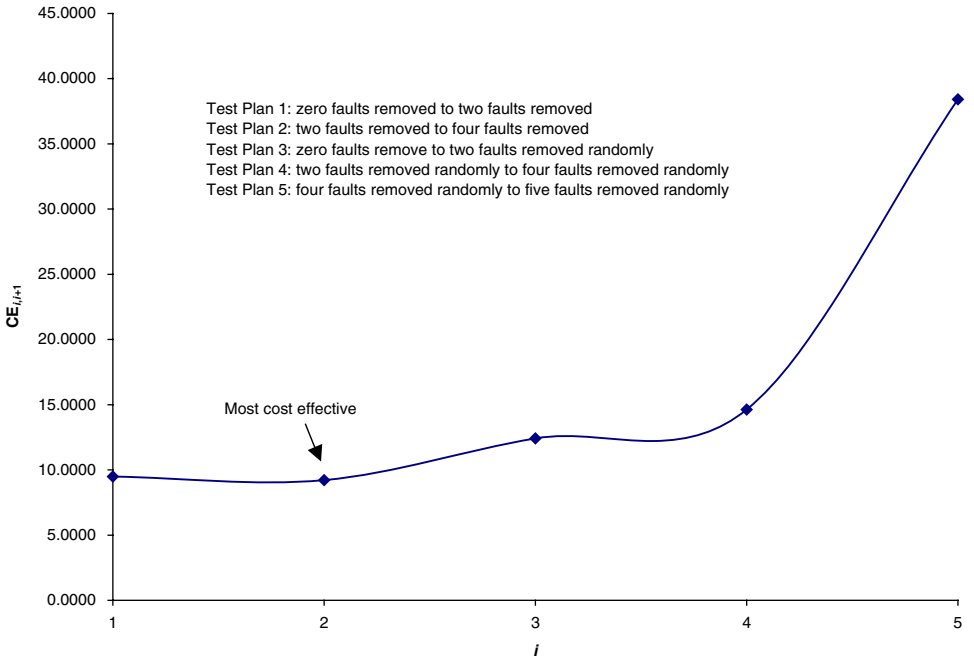

Fig. 7 Japanese University System: mean value cost effectiveness ratio $CE_{i,i+1}$ vs. test plan i.

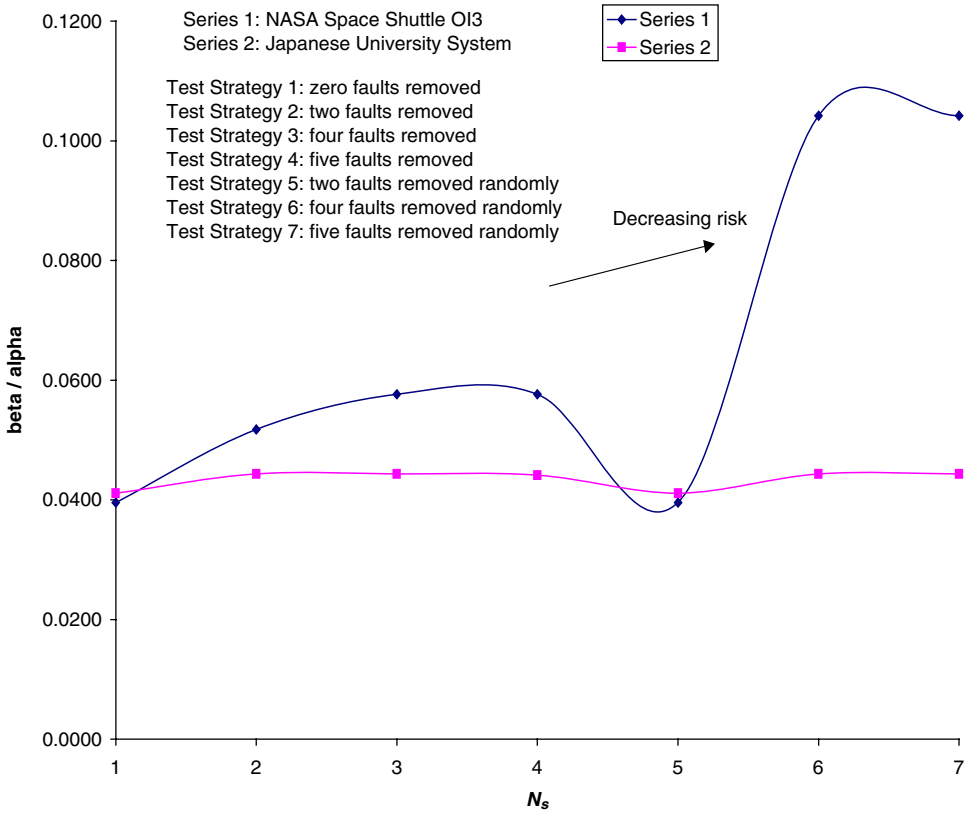

Fig. 8 Parameter ratio (beta/alpha) vs. test strategy N_s.

E. Rate of Return of Testing Investment

1. NASA Space Shuttle

The annual software development contract for the Shuttle flight software is $C = \$\,35$ M. On a daily basis, realizing that the Shuttle development and testing process is continuous, this amounts to $c(t) = \$\,95{,}890$ (365 days per year). Therefore, the value of t days of testing is $c(t)\$\,95{,}890\,t$. Using Eq. (3), we computed a rate of return $r = 13.90\%$ when no faults have been removed. This is a very good rate of return considering the return available in money markets. Then we computed r for the various testing strategies, and interestingly there was no change in r except for $r = 13.89\%$ when two faults were removed randomly. This result implies that for Shuttle release OI3 return on investment in testing is independent of the type of test.

F. Fault Correction Rate

1. NASA Space Shuttle, Japanese University System, and Database Application

The best strategies with respect to fault correction rate limits are where the plots are maximum in Figs. 9 and 10. This occurs at removing two faults randomly for the Shuttle, removing two faults deterministically for the Japanese System, and removing two faults randomly for the database application. Again, although the evidence is obviously not overwhelming in the case of random fault removal, the weight of the evidence suggests that an organization would not go wrong by using this test strategy.

G. Conclusions

We used three sets of fault data comprising a range of applications and modeled several test strategies for each application. Overall, random removal of faults proved to be the best strategy in terms of resultant reliability metrics. This result is reinforced by the fact that random fault removal is less expensive than deterministic removal. In addition, this result is consistent across the three applications.

Another conclusion is that for our test assessment strategy to be effective, fault removal must be sufficiently large to produce significant differences in the reliability metrics across test strategies. Initially, this was not the case

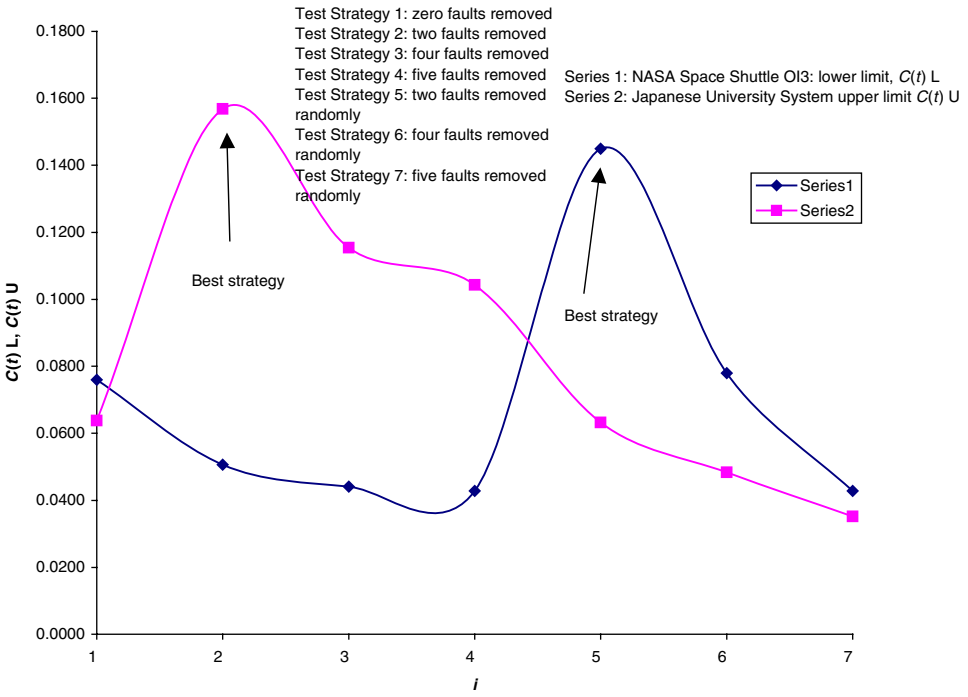

Fig. 9 Correction rate limits $C(t)$L and $C(t)$U vs. test strategy i.

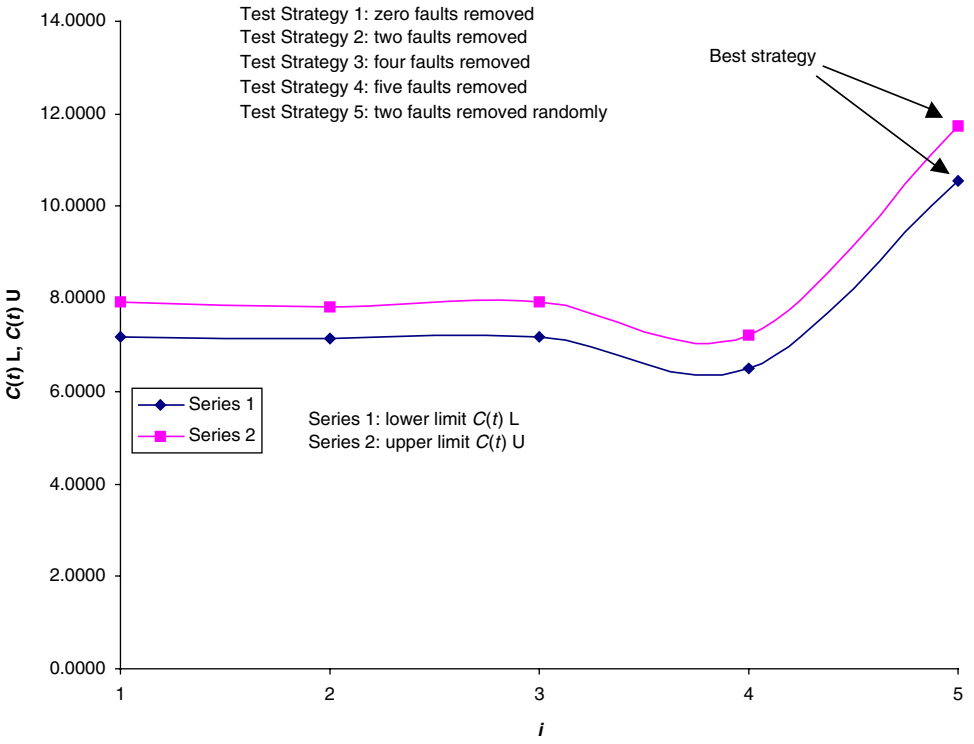

Fig. 10 Database application fault correction rate limits, $C(t)$L and $C(t)$U, vs. test strategy i.

in dealing with the database application. In retrospect, it would have been better to remove faults based on equal proportion of faults removed across applications.

Because the MREs for reliability predictions were the lowest for any of the reliability metrics, we have greater confidence in test strategies identified by reliability than for other metrics.

References

[1] Schneidewind, N. F., "Modelling the Fault Correction Process," *Proceedings of The 12th International Symposium on Software Reliability Engineering*, IEEE Publications, Piscataway, NJ, 27–30 Nov. 2001, pp. 185–190.

[2] Gokhale, S. S., Marinos, P. N., Lyu, M. R., and Trivedi, K. S., "Effect of Repair Policies on Software Reliability," *Proceedings of Computer Assurance*, IEEE Publications, Piscataway, NJ, 1997, p. 10.

[3] Keller, T., and Schneidewind, N. F., "A Successful Application of Software Reliability Engineering for the NASA Space Shuttle," *Software Reliability Engineering Case Studies, International Symposium on Software Reliability Engineering, November 3*, IEEE Publications, Piscataway, NJ, 4 Nov 1997, pp. 71–82.

[4] *Handbook of Software Reliability Engineering*, edited by Lyu, M. R., Published by IEEE Computer Society Press and McGraw-Hill Book Company, Piscataway, NJ, 1996.

[5] Fenton, N. F., and Pfleeger, S. L., *Software Metrics: A Rigorous & Practical Approach*, 2nd ed., PWS Publishing Company, Boston, MA, 1997.

[6] Levine, D. M., Ramsey, P. P., and Smidt, R. K., *Applied Statistics for Engineers and Scientists*, Prentice-Hall, Upper Saddle River, NJ, 2001.

[7] Boardman, A. E., Greenberg, D. H., Vining, A. R., and Weimer, D. L., *Cost-Benefit Analysis: Concepts and Practice*, 3rd ed., Prentice-Hall, Upper Saddle River, NJ, 2006.

[8] Lindeburg, M. R., *Engineering Economic Analysis: An Introduction*, Professional Publications, Inc., Belmont, CA, 2001.

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

[10] "AIAA Recommended Practice for Software Reliability," American Institute of Aeronautics and Astronautics, 1993.

[11] Schneidewind, N. F., "Predicting shuttle software reliability with parameter evaluation," *Innovations Systems Software Engineering*, Springer–Verlag London Limited 2007.
[12] Schneidewind, N. F., "Analysis of Error Processes in Computer Software," *Proceedings of the International Conference on Reliable Software*, IEEE Publications, Piscataway, NJ, 21–23 April 1975, pp. 337–346.
[13] Xie, M., and Zhao, M., "The Schneidewind Software Reliability Model Revisited," *Proceedings of the 3rd International Symposium on Software Reliability Engineering*, IEEE Computer Society Press, Los Alamitos, CA, Research Triangle Park, NC, 7–10 Oct 1992, pp. 184–192.
[14] Kleinrock, L., *Queuing Systems, Volume 1: Theory*, Wiley, New York, 1975.
[15] Musa, J. D., Iannino, A., and Okumoto, K., *Software Reliability: Measurement, Prediction, Application*, McGraw-Hill, New york, 1987.

Appendix

Table A1 Failure data sets

Shuttle		Data base application		Japanese University	
Time to failure (days)	Cumulative failure count	Execution time (hours)	Cumulative failure count	Execution time (seconds)	Cumulative failure count
29	2	2.45	15	36	1
36	2	4.90	44	40	2
91	4	6.86	66	45	3
92	6	7.84	103	52	4
120	6	9.52	105	54	5
127	6	12.89	110	102	6
246	6	17.10	146	108	8
289	7	20.47	175	172	9
291	7	21.43	179	197	10
293	7	23.35	206	200	11
311	7	26.23	223	242	12
364	8	27.67	255	326	13
830	8	30.93	276	491	14
1078	9	34.77	298	514	19
2626	10	38.61	304	537	20
4096	11	40.91	311	580	21
		42.67	320	677	22
		44.66	325	678	24
		47.65	328	732	25
				779	26
				794	27

Michael Hinchey
Associate Editor