

Modelling the Fault Correction Process
Proceedings of The Twelfth International Symposium on Software Reliability
Engineering, Hong Kong, 27-30 November, 2001, pp. 185-190.

Norman F. Schneidewind
Naval Postgraduate School

Abstract

In general, software reliability models have focused on modeling and predicting failure occurrence and have not given equal priority to modeling the fault correction process. However, there is a need for fault correction prediction, because there are important applications that fault correction modeling and prediction support. These are the following: predicting whether reliability goals have been achieved, developing stopping rules for testing, formulating test strategies, and rationally allocating test resources. Because these factors are related, we integrate them in our model. Our modeling approach involves relating fault correction to failure prediction, with a time delay between failure detection and fault correction, represented by a random variable whose distribution parameters are estimated from observed data.

1. Introduction

There is a need for greater emphasis on fault correction modeling and prediction in software reliability models. This need stems from the fact that the fault correction process is vital to ensuring high quality software. If we only address failure prediction, reliability assessment will be incomplete because it would not reflect the reliability of the software resulting from fault correction. In addition to achieving greater accuracy in reliability prediction, there are by-product benefits associated with fault correction prediction as follows:

- a. Predicting whether reliability goals have been achieved: If no predictions are made of the number of faults to be corrected, fault correction rate, and fault correction time, accurate prediction of reliability cannot be obtained.
- b. Providing stopping rules for testing as follows: (1) The predicted number of remaining faults is less than or equal to a specified critical value and (2) The fault correction rate asymptotically approaches zero.
- c. Prioritizing tests and allocating test resources: Software with high values of number of remaining faults and low fault correction rates are given high priority in testing and allocation of resources, such as personnel and computer time.

Our research hypothesis is that fault correction can be modeled 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.

Gokhale and colleagues 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]. However, this is not the same as the delay mentioned in section 2.1 below that is the result of a decision by the developing organization to defer fault correction until the removal of a fault becomes critical to the operation of the software. 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) [1]. They use a non-homogeneous Markov Chain to represent a non-homogeneous Poisson process to model failure detection and fault correction. Their approach is interesting and provides greater flexibility than analytical models like ours in a variety of fault correction scenarios. However, analytical models provide greater visibility of the relationships between factors that influence the fault correction process than do complex Markov Chain diagrams. In addition, their models have not been validated against real-world projects. In contrast, we have provided validation tests in Tables 1, 2, and 3 for the Shuttle.

The delayed S-shaped model has the interesting characteristic of an initial increasing failure intensity function, as the test team becomes familiar with the software, reaches a maximum, and then asymptotically approaches zero with test time, as it becomes more difficult to detect failures. Thus, this model gets its name from a *delay in failure detection* and not fault correction, because an assumption of the model is that faults are corrected immediately without introducing new ones [4].

This paper contains the following sections: 2. Fault Correction Prediction Model Components, 3. Applications, 4. Validation, and 5. Summary and Conclusions

2. Fault Correction Prediction Model Components

This section develops the equations of the components of the fault correction prediction model, where all references to “time” are elapsed or wall clock times. These components include the following: failure and fault correction counts, measures of progress in fault correction, and stopping rules for testing to achieve specified reliability goals.

2.1 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 [6]. 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 [6]. As pointed out by Xie, this assumption is too restrictive [9]. He suggests modeling the delay as an increasing function of test time. However, we have not found this to be the case in either the Shuttle or Goddard Space Flight Center (GSFC) 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. In order to improve the model, 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 intervals dT . This distribution was confirmed for the Shuttle, using a sample of 85 fault correction times and the Kolmogorov-Smirnov test, resulting in $p = 0$. In addition, Musa found that failure correction times were exponentially distributed for 178 failure corrections [5].

We assume that the fault correction *starts* when failures are detected. This assumption is related to the previous assumption of fault correction keeping 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 in the *current model*. Our current model is based on keeping the software updated with corrections in the current release. In addition, the model does not

include the possibility of introducing a fault when correcting one; there is no data available for the Shuttle regarding this factor. These are important factors, but they are beyond the scope of this effort; these factors will be addressed in a future effort, involving the use of GSFC data. Fault mitigation methods, such as fault tolerance, (the Shuttle has four active computers and a fifth backup) are reflected in the model as a reduction in the number of failures from what would be experienced with no fault tolerance; this result, in turn, decreases the model’s failure rate parameters and predicted number of failures.

It is well known that various human queue service times (e.g., supermarket checkout stands) can be approximated with an exponential distribution [3], 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 uses this type of queuing model in his failure correction process [5].

Due to the great variability in fault correction time that we have found in both the Shuttle and GSFC data, we emphasize predicting limits instead of expected values. For a given mean fault correction rate m , the cumulative probability distribution $F(dT)$ of the fault correction delay dT is used to specify upper and lower limits of dT . These limits are dT_U and dT_L , corresponding to F_U and F_L , respectively. The concept is to bound the delay time, for example at $F_U = .9$ and $F_L = .1$, 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 .80). The equation for $F(dT)$ for the exponential distribution, is given by (1):

$$F(dT) = 1 - \exp(-m(dT)). \quad (1)$$

Equation (1) is manipulated to produce equation (2), which would be used to compute the limits of dT , applying the specified limit values of $F(dT)$:

$$dT = (-\log(1 - F(dT)))/m. \quad (2)$$

In the examples in section 3, predictions that require $F(dT)$ are made using F_U and *not* F_L in order to provide conservative estimates (e.g., the probability is .90 that the number of corrected faults is less than or equal to its predicted value or .10 that it exceeds this value). The reason for this approach is to mitigate against overly optimistic predictions of number and rate of fault correction.

2.2 Number of Faults Corrected

The predicted number of failures detected $D(T)$, for $T > (s-1)$, is given by equation (3) [8]:

$$D(T) = (\alpha/\beta)[1 - \exp(-\beta((T-s+1)))] + X_{s-1}, \quad (3)$$

where the terms have the following definitions:

- α : failure rate at the beginning of interval s
- β : negative of derivative of failure rate divided by failure rate (i.e., relative rate of change of failure rate)
- T : cumulative test or operational time
- s : starting interval for using observed failure data in parameter estimation
- X_{s-1} : observed failure count in the range $[1, s-1]$.

The parameters α and β are obtained from maximum likelihood estimation techniques [8]. The parameter s is used in the optimal selection of failure data that involves selecting only the most relevant set of failure data for reliability prediction, with the result of producing more accurate predictions than would be the case if the entire set of data were used. The mean squared error criterion, applied to the differences between predicted and actual values in the observed range of the failure data, is used for selecting the optimal value of s . For all equations in which the term $T - s + 1$ appears, predictions are made for $T > (s - 1)$ [7, 8].

Using the assumption of section 2.1 that the number of corrected faults $C(T)$ has the same form as the number of detected failures $D(T)$ but with a variable delay dT , yields equation (4), for $T > (s - 1)$, [6]:

$$C(T) = (\alpha/\beta) [1 - \exp(-\beta((T-s+1) - dT))] + C_{s-1}, \quad (4)$$

where C_{s-1} is the observed fault correction count in the range $[1, s-1]$. Using equation (2), equation (4) becomes equation (5), where we would compute the upper and lower limits of $C(T)$:

$$C(T) = (\alpha/\beta) [1 - \exp(-\beta(T-s+1 + (\log(1 - F(dT))/m)))] + C_{s-1} \quad (5)$$

2.3 Proportion of Faults Corrected

A measure of progress in fault correction of detected failures, at time T , is given by the proportion of faults corrected, as expressed in equation (6):

$$r(T) = C(T)/D(T) \quad (6)$$

The ideal goal, of course, is to achieve a value of 1. However, the achievement of this goal is constrained by the amount of test time that is economically feasible to allocate to the software under test. This challenge will be addressed in section 3, when we consider stopping rules and prioritization of tests.

2.4 Number of Remaining Faults

The predicted number of remaining faults, after the correction process has been operative for time T , is given by equation (7):

$$N(T) = D(T) - C(T) \quad (7)$$

This equation is based on the assumption that all the faults that exist in the software have been predicted by $D(T)$. A

more conservative prediction is obtained by predicting the detected failures over the life of the software $D(T_L)$, as in equation (8) [8], and then using equation (9) as the predicted remaining faults.

$$D(T_L) = \alpha/\beta + X_{s-1} \quad (8)$$

$$N(T_L) = D(T_L) - C(T) \quad (9)$$

2.5 Time Required To Correct C Faults

In order to do informed scheduling of test resources, such as personnel and computer time, it is helpful to predict how much cumulative test time would be required to correct a given number of faults during testing. If equation (4) is solved for T , and $C(T)$ becomes a given number of faults C to correct, we obtain equation (10), the predicted time required to correct C faults during testing. If the delay is exponentially distributed, equation (2) would be substituted for dT in equation (10).

$$T_c = [\log[\alpha/(\alpha - \beta(C - C_{s-1}))]]/\beta + (s-1 + dT), \text{ for } \alpha > \beta(C - C_{s-1}) \text{ and } C \geq C_{s-1} \quad (10)$$

2.6 Fault Correction Rate

The fault correction rate is another useful measure of progress in fault correction. If the rate is decreasing and relatively high, it would be indicative of further gains in fault correction by continuing to test. On the other hand, if the rate is decreasing towards an asymptotic value, it would indicate that further testing would produce little gain in fault correction. The fault correction rate is obtained by taking the derivative of equation (5) to produce equation (11):

$$R(T) = \alpha[\exp(-\beta(T-s+1 + (\log(1 - F(dT))/m)))] \quad (11)$$

3. Applications

For the purpose of model development and validation, post release failure data from three Shuttle operational increments were used: OID, OIJ, and OIO. An operational increment is a software system comprised of modules and configured from a series of builds to meet Shuttle mission functional requirements. In addition, fault correction data, obtained from Shuttle build inspection files, were used. This section presents several applications of the model developed in section 2, involving reliability assessment and strategies for efficient testing. Because of the space restriction, we are unable to show the plots referred to in this section.

3.1 Predicting whether Reliability Goals Have Been Achieved

Because it is important to gear the fault correction process to the remaining faults, we specify inequality (12), which relates $N(T)$, the number of remaining faults at time T , to R_C , the critical value of remaining faults:

$$N(T) \leq R_C, \text{ or } N(T) = (D(T) - C(T)) \leq R_C \quad (12)$$

When equations (3) and (4) are substituted for $D(T)$ and $C(T)$, respectively, in inequality (12), and assuming that $X_{s-1} - C_{s-1} \cong 0$ because both X_{s-1} and C_{s-1} are small in the range $1, s-1$, we obtain inequality (13), the condition for maximum fault correction delay:

$$dT \leq [\log[1 + (\beta/\alpha)(\exp(\beta(T - s + 1)))(R_C)]]/\beta \quad (13).$$

The parameter R_C serves as a reliability threshold. A value of $R_C = 1$ would be appropriate for safety critical systems. If $R_C = 0$, $dT = 0$. This result makes sense because to achieve zero faults, faults must be corrected as soon as failures are detected. We feel that inequality (13) is a significant result, because it says that independent of the distribution of dT , (13) must be satisfied to meet the reliability requirement. Thus for a given value of R_C , we are able to specify the maximum fault correction delay dT that will meet the reliability goal. As a practical matter, the software engineer can control the development and maintenance process to constrain the fault correction delay to (13) by assigning test personnel with the appropriate skills and by allocating sufficient computer resources to the tests.

3.2 Stopping Rules for Testing and Prioritizing Tests and Test Resources

3.2.1 Remaining Faults and Fault Correction Rate.

The number of remaining faults $N(T)$, equation (7), can be used as a stopping rule for testing. We used an upper probability limit of .90, meaning that the probability is .90 that $N(T)$ is less than or equal to its ordinate values for a given test time, or .10 that these values are exceeded. A plot of multiple OIs would show how to prioritize tests (i.e., the OI requiring the most test time for a given R_C would have the highest priority). By priority, we mean the order of testing and allocation of personnel and computer resources. In addition, the plot would show when to stop testing (i.e., when the remaining faults equals R_C). Plotting the fault correction rate $R(T)$, equation (11), would also provide insight for when to stop testing and for prioritizing tests. However, we consider $N(T)$ more useful because it can be used with the critical value of remaining faults R_C -- a reliability threshold.

3.2.2 Reliability Improvement. In the previous section, we used absolute quantities (e.g., number of remaining faults) for developing test strategies and assessing reliability. In this section, we use the relative quantity $p(T)$, the proportion of faults remaining at time T , which is related to $r(T)$ from equation (6), the proportion of faults corrected at time T , by equation (14):

$$p(T) = 1 - r(T) = 1 - C(T)/D(T) \quad (14).$$

We use (14) because two software systems could have experienced different numbers of failures but have equal

numbers of remaining faults. In this case, the software with fewer failures would have achieved greater progress in reliability improvement, as measured by $p(T)$. However, the use of a threshold seems more intuitive when applied to $N(T)$. In practice, both measures could be used.

3.2.3 Test Scheduling. We can anticipate test requirements and do proactive test scheduling by using equation (10), the predicted amount of test time required to correct a given number of faults T_c . In addition, a plot of multiple software systems would show how the systems compare in test requirements.

4. Validation

Predictions were made for post release failures and fault correction, where OI, OIJ, and OIO experienced 13, 7, and 7 post release failures, respectively (post release failures are sparse for the Shuttle). To make fault correction predictions comparable across OIs, 7 failures were used for each OI (the first seven for OI). All data and predictions are in terms of 30-day intervals. This is the failure count interval used in previous Shuttle reliability predictions [8]. Although 7 failures may seem like a small sample size, it is representative of Shuttle post release reliability, and despite the small number of failures, the model is able to predict detected number of failures $D(T)$ fairly accurately (see Tables 1, 2, and 3).

For the purpose of validation, we ran three scenarios -- one each for OI, OIJ, and OIO -- shown in Tables 1, 2, and 3, respectively. This involved determining from the collected data when failures occurred and when faults were corrected. The actual delay between failure occurrence and fault correction was estimated by examining, manually, the Shuttle Discrepancy Reports (i.e., reports that document deviations between specified and observed software behavior) to determine the disposition of the fault (i.e., the release and release date on which the fault was corrected).

The event column of the tables shows when either a failure occurred or a fault was corrected. In some cases multiple failures or corrected faults occurred in the same interval; these occurrences are signified by the plural form in the event column. The next column shows the test time T when the events occurred followed by the actual values of cumulative number of failures detected $D(T)$, cumulative number of faults corrected $C(T)$, and number of remaining faults $N(T)$, the difference between $D(T)$ and $C(T)$. The next section of the tables shows the predictions for $D(T)$, $C(T)$, and $N(T)$. Notice at the top of each table the statement about the range of prediction, which is for $T > s-1$ (see section 2.2). For example, for

OID in Table 1, $s = 7$; therefore, the predictions start at interval $T = 7.43$.

Table 1: OID (Predictions for $T > s-1 = 6$)										
Event	T	Actual Values			Predictions			Squared Error		
		D(T)	C(T)	N(T)	D(T)	C(T)	N(T)	D(T)	C(T)	N(T)
Failure	4.53	1	0	1						
Failure	4.83	2	0	2						
Failure	5.70	3	0	3						
Failure	7.43	4	0	4	3.53	0.30	3.23	0.220	0.09	0.59
Failure	9.77	5	0	5	4.37	1.15	3.22	0.401	1.31	3.17
Corrections	10.86	5	4	1	4.74	4.53	0.22	0.066	0.28	0.62
Correction	10.87	5	5	0	4.75	4.53	0.22	0.065	0.22	0.05
Failure	12.73	6	5	1	5.37	5.16	0.21	0.401	0.03	0.63
Correction	14.60	6	6	0	5.97	5.77	0.20	0.001	0.05	0.04
Failure	17.50	7	6	1	6.85	6.67	0.19	0.022	0.44	0.66
Correction	18.17	7	7	0	7.05	6.87	0.18	0.002	0.02	0.03
Mean Square Error								0.147	0.31	0.72

Table 2: OIJ (Predictions for $T > s-1 = 8$)										
Event	T	Actual Values			Predictions			Squared Error		
		D(T)	C(T)	N(T)	D(T)	C(T)	N(T)	D(T)	C(T)	N(T)
Failure	3.57	1	0	1						
Failure	7.03	2	0	2						
Failure	7.47	3	0	3						
Failure	9.17	4	0	4	3.60	0.28	3.32	0.16	0.08	0.46
Failure	9.23	5	0	5	3.63	0.32	3.32	1.87	0.10	2.83
Corrections	10.60	5	2	3	4.28	2.99	1.29	0.51	0.99	2.92
Failure	13.20	6	2	4	5.38	4.13	1.25	0.38	4.54	7.57
Correction	13.66	6	3	3	5.56	5.32	0.24	0.20	5.36	7.61
Failure	17.17	7	3	4	6.75	6.56	0.20	0.06	12.65	14.47
Correction	24.06	7	4	3	8.47	8.34	0.13	2.16	18.86	8.25
Correction	24.27	7	5	2	8.51	8.39	0.13	2.29	11.47	3.51
Correction	37.43	7	6	1	10.31	10.25	0.06	10.92	18.05	0.89
Correction	37.44	7	7	0	10.31	10.25	0.06	10.93	10.56	0.00
Mean Square Error								2.95	8.27	4.85

Table 3: OIO (Predictions for $T > s-1 = 8$)										
Event	T	Actual Values			Predictions			Squared Error		
		D(T)	C(T)	N(T)	D(T)	C(T)	N(T)	D(T)	C(T)	N(T)
Failure	5.77	1	0	1						
Failure	5.90	2	0	2						
Failure	7.53	3	0	3						
Correction	9.07	3	1	2	3.20	1.06	2.14	0.04	0.00	0.02
Failures	11.47	5	1	4	3.62	1.49	2.13	1.90	0.24	3.50
Corrections	16.80	5	4	1	4.49	3.37	1.11	0.26	0.39	0.01
Failure	24.67	6	4	2	5.60	4.50	1.10	0.16	0.25	0.82
Corrections	29.40	6	6	0	6.18	6.09	0.09	0.03	0.01	0.01
Failure	36.17	7	6	1	6.92	6.84	0.08	0.01	0.71	0.86
Correction	45.77	7	7	0	7.79	7.73	0.06	0.63	0.54	0.00
Mean Square Error								0.43	0.31	0.75

The last section of the tables shows the squares of the differences between actual and predicted values for computing the Mean Square Error (MSE) at the bottom of the tables. We consider the MSE values for OID and OIO, Tables 1 and 3, respectively, to be sufficiently low to validate the predictions. However, we do not reach that conclusion regarding OIJ in Table 2. The discrepancy between predicted and actual results is due primarily to the long delay between failure detection and the *start* of fault correction that occurs because the failures were classified as non-critical on the current release and were not considered critical for one to three releases in the future. In contrast, “our model is based on keeping the software updated with corrections in the current release” (see section 2.1). For the same reason, we were unable to validate inequality (13), the maximum fault correction delay (see section 3.1). This is a lesson learned from the research that we will further address in section 5.

5. Summary and Conclusions

It is imperative to include fault correction in reliability modeling and prediction because without it, predictions will understate the reliability of the software. We have developed a number of equations for assessing the improvement in reliability resulting from fault correction. In addition, we have shown examples of how they apply to stopping rules for testing and prioritization of tests and test resources. We showed that the *number of remaining faults* is better than the *fault correction rate* for assessing reliability and prioritizing tests because the former can be used with a reliability threshold to predict how much testing would be required to meet the reliability goal.

We found that the most important factor in fault correction modeling is the delay between failure detection and fault correction. We modeled this factor, using the concept of a fault correction queuing service with exponentially distributed delay – a highly statistically significant empirical result based on Shuttle data.

We obtained good validation results for two of the three OIs evaluated. The third OI served as a lesson learned that we will apply to future work on the NASA GSFC software projects. We will investigate whether fault corrections are postponed and, if so, whether we can model this delay. This will be a challenge because, whereas failure detection is a machine process, fault correction is part human process (deciding when to implement a correction and analyzing how to make the correction) and part machine process (verifying the correction on a computer).

References

- [1] Swapna S. Gokhale, Teebu Phillip, and Peter N. Marinos, “A Non-Homogeneous Markov Software Reliability Model with Imperfect Repair”, Proceedings of the International Performance and Dependability Symposium, Urbana-Champaign, IL, 1996, 10 pages.
- [2] Swapna S. Gokhale, Peter N. Marinos, Michael R. Lyu, and Kishor S. Trivedi, “Effect of Repair Policies on Software Reliability”, Proceedings of Computer Assurance, Gaithersburg, MD, 1997, 10 pages.
- [3] Leonard Kleinrock, *Queuing Systems, Volume 1: Theory*, John Wiley & Sons, New York, 1975.
- [4] Michael R. Lyu (Editor-in-Chief), *Handbook of Software Reliability Engineering*, Computer Society Press, Los Alamitos, CA and McGraw-Hill, New York, NY, 1995, pp. 95-98.
- [5] John D. Musa, et al, *Software Reliability: Measurement, Prediction, Application*, McGraw-Hill, New York, 1987.
- [6] Norman F. Schneidewind, "Analysis of Error Processes in Computer Software", Proceedings of the International Conference on Reliable Software, IEEE Computer Society, 21-23 April 1975, pp. 337-346.
- [7] Norman F. Schneidewind, "Software Reliability Model with Optimal Selection of Failure Data", IEEE Transactions on Software Engineering, Vol. 19, No. 11, November 1993, pp. 1095-1104.
- [8] Norman F. Schneidewind, "Reliability Modeling for Safety Critical Software", IEEE Transactions on Reliability, Vol. 46, No.1, March 1997, pp.88-98.
- [9] Min Xie and M. Zhao, “The Schneidewind Software Reliability Model Revisited”, Proceedings of the Third International Symposium on Software Reliability Engineering, IEEE Computer Society Press, Los Alamitos, CA, Research Triangle Park, NC, October 7-10, 1992, pp. 184-192.