

## **Reliability Modeling for Safety Critical Software**

**Norman F. Schneidewind, Fellow IEEE**

IEEE Transactions on Reliability, Vol. 46, No.1, March 1997, pp. 88-98

Code IS/Ss  
Naval Postgraduate School  
Monterey, CA 93943, U.S.A.  
Voice: (831) 656-2719  
Fax : (831) 372-0445  
Internet: nschneid@nps.navy.mil

**Keywords:** software reliability prediction, safety critical software, risk analysis.

### **Summary and Conclusions**

We show how software reliability predictions can increase confidence in the reliability of safety critical software such as the *NASA Space Shuttle Primary Avionics Software System* (*Shuttle* flight software). This objective was achieved using a novel approach to integrate software safety criteria, risk analysis, reliability prediction, and stopping rules for testing. This approach is applicable to other safety critical software. We only cover the safety of the software in a safety critical system. The hardware and human operator components of such systems are not explicitly modeled nor are the hardware and operator induced software failures. Our concern is with reducing the risk of all failures *attributed* to software. Thus, our use of the word *safety* refers to *software safety* and not to system safety. By improving the reliability of the software, where the reliability measurements and predictions are *directly related to mission and crew safety*, we contribute to system safety.

*Remaining failures, maximum failures, total test time* required to attain a given *fraction of remaining failures*, and *time to next failure* are shown to be useful reliability measurements and predictions for: 1) providing confidence that the software has achieved safety goals; 2) rationalizing how long to test a piece of software; and 3) analyzing the risk of not achieving *remaining failure* and *time to next failure* goals. Having predictions of the extent that the software is not fault free (*remaining failures*) and whether it is likely to survive a mission (*time to next failure*) provide criteria for assessing the risk of deploying the software. Furthermore, *fraction of remaining failures* can be used as both an *operational quality* goal in predicting *total test time* requirements and, conversely, as an indicator of *operational quality* as a function of *total test time* expended.

Software reliability models provide one of several tools that software managers of the *Shuttle* flight software are using to provide confidence that the software meets required safety goals. Other tools are inspections, software reviews, testing, change control boards, and perhaps most important -- experience and judgement.

## **1. Introduction**

We propose that two categories of software reliability measurements (i.e., observed failure data used for model parameter estimation) and predictions (i.e., forecasts of future reliability using the parameterized model) be used in combination to *assist* in assuring the safety of the software in safety critical systems like the *Shuttle* flight software. The two categories are: 1) measurements and predictions that are associated with residual software faults and failures,

and 2) measurements and predictions that are associated with the ability of the software to survive a mission without experiencing a serious failure. In the first category are: *remaining failures*, *maximum failures*, *fraction of remaining failures*, and *total test time required to attain a given number or fraction of remaining failures*. In the second category are: *time to next failure* and *total test time required to attain a given time to next failure*. In addition, we define the risk associated with not attaining the required *remaining failures* and *time to next failure*. Lastly, we derive a quantity from the *fraction of remaining failures* that we call *operational quality*.

The benefits of predicting these quantities are: 1) they provide confidence that the software has achieved safety goals, and 2) they provide a means of rationalizing how long to test a piece of software (stopping rule). Having predictions of the extent that the software is not fault free (*remaining failures*) and its ability to survive a mission (*time to next failure*) are meaningful for assessing the risk of deploying safety critical software. In addition, with this type of information a software manager can determine whether more testing is warranted or whether the software is sufficiently tested to allow its release or unrestricted use. These predictions, in combination with other methods of assurance, such as inspections, defect prevention, project control boards, process assessment, and fault tracking, provide a quantitative basis for achieving safety and reliability goals [3].

*Risk* in the Webster's New Universal Unabridged Dictionary is defined as: "the chance of injury; damage, or loss" [19]. Some authors have extended the dictionary definition as

follows: "Risk Exposure=Probability of an Unsatisfactory Outcome\*Loss if the Outcome is Unsatisfactory" [2]. Such a definition is frequently applied to the risks in managing software projects such as budget and schedule slippage. In contrast, our application of the dictionary definition pertains to the risk of executing the software of a safety critical system where there is the chance of injury (e.g., astronaut injury or fatality), damage (e.g., destruction of the *Shuttle*), or loss (e.g., loss of the mission) if a serious software failure occurs during a mission. We have developed risk criterion metrics to quantify the degree of risk associated with such an occurrence.

Lockheed-Martin, the primary contractor on the *Shuttle* flight software project, is experimenting with a promising algorithm which involves the use of the *Schneidewind Software Reliability Model* to compute a parameter: *fraction of remaining failures* as a function of the archived failure history during test and operation [10]. Our prediction methodology uses this parameter and other reliability quantities to provide bounds on *total test time*, *remaining failures*, *operational quality*, and *time to next failure* that are necessary to meet *Shuttle* safety requirements. We also show that there is a pronounced asymptotic characteristic to the *total test time* and *operational quality* curves that indicate the possibility of big gains in reliability as testing continues; eventually the gains become marginal as testing continues. We conclude that the prediction methodology is feasible for the *Shuttle* and other safety critical systems.

We only cover the safety of the software in a safety critical system. The hardware and human operator components of such systems are not explicitly modeled nor are the hardware and operator induced software failures. However, in practice, these hardware-software interface and human operator-software interface failures may be very difficult to identify as such; these failures may be recorded as software failures. Our concern is with reducing the risk of all failures *attributed* to software. Thus, our use of the word *safety* refers to *software safety* and not to system safety.

Although *remaining failures* has been discussed in general as a type of software reliability prediction [13], and various stopping rules for testing have been proposed, based on costs of testing and releasing software [4, 5, 8, 17], *failure intensity* [12], and testability [18], our approach is novel because we integrate software safety criteria, risk analysis, reliability prediction, and a stopping rule for testing. For a system like the *Shuttle*, where human lives are at risk, we cannot use economic or time-to-market criteria to determine when to deploy the software. Although *failure intensity* has proven useful for allocating test effort and determining when to stop testing in commercial systems [12], this criterion is not directly related to software safety. In a safety critical system, the prediction of remaining failures and identification of the faults which cause them is more relevant to ensuring safety than the trend of failure intensity over time. The latent faults must be found and removed through additional testing, inspection, or other means, if the safety of the mission is not to be jeopardized. Furthermore, as we will show, *remaining failures*, along with *time to next failure*, can be

used as risk criteria. It is not clear how *failure intensity* could be a meaningful safety criterion.

Because *testability* attempts to quantify the probability of failure, if the code is faulty [18], this criterion has a relationship with reliability if we *know* that the code is faulty. However in the *Shuttle* and other safety critical software, our purpose is to *predict* whether the code is faulty. For safety critical software, we must use reliability measurements and predictions to assess whether safety and mission goals are likely to be achieved.

We first define two criteria for software safety. Then we apply these criteria to risk analysis of safety critical software, using the *Shuttle* flight software as an example. Next, we define and provide brief derivations for a variety of prediction equations that are used in reliability prediction and risk analysis; included is the relationship between *time to next failure* and *reduction in remaining failures*. This is followed by an explanation of the principal of *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. Then we show how the prediction equations can be used to integrate testing with reliability and quality. An example is shown of how the risk analysis and reliability predictions can be used to make decisions about whether the software is safe to deploy. Lastly we show validation results for a variety of predictions.

## ***Acronyms***

*OIA*: Shuttle operational increment A

*OIB*: Shuttle operational increment B

*OIC*: Shuttle operational increment C

*OID*: Shuttle operational increment D

## ***Assumptions*** [1]:

1. Faults that cause failures are removed.
2. As more failures occur and more faults are corrected, *remaining failures* will be reduced.
3. The *remaining failures* are "zero" for those OI's that were executed for extremely long times (years) with no additional failure reports; correspondingly, for these OI's, maximum failures equals total observed failures.
4. The number of failures detected in one interval is independent of the failure count in another.
5. Only "new" failures are counted (i.e., failures that are repeated as a consequence of not correcting a fault are not counted).

## ***Definitions***

o *Interval*: an integer time unit  $t$  of constant length defined by  $t-1 < t < t+1$ , where  $t > 0$ ; failures are counted in intervals (e.g., one failure occurred in *interval 4*) [1, 7].

o *Number of Intervals*: the number of contiguous integer time units  $t$  of constant length

represented by a positive real number (e.g., the predicted time to next failure is 3.87 *intervals*).

o *Operational Increment* (OI): a software system comprised of modules and configured from a series of builds to meet *Shuttle* mission functional requirements.

o *Time*: Continuous CPU execution time over an *interval* range.

*Severity Codes*:

1. Severe Vehicle or Crew Performance Implications.
2. Affects Ability to Complete Mission (Not a safety issue).
3. Workaround Available, Minimal Effect on Procedures.
4. Insignificant (Paperwork, etc.).
5. Not Visible to User.

### ***Nomenclature***

o *Predicted at time t*: a prediction made in the *interval t*.

o *Safety*: software safety; not system safety.

### ***Notation***

$\hat{\lambda}$	failure rate at the beginning of interval $s$
$\hat{\lambda}$	negative of derivative of failure rate divided by failure rate (i.e., relative failure rate)
$F(i)$	predicted failure count in the range $[1,i]$ ; used in computing $MSE_T$
$F_{ij}$	observed failure count during interval $j$ since interval $i$ ; used in computing $MSE_T$

$F(t)$	predicted failure count in the range $[1, t]$
$F_t$	given number of failures to occur after interval $t$ ; used in predicting $T_F(t)$
$F(t_1, t_2)$	predicted failure count in the range $[t_1, t_2]$
$F(4)$	predicted failure count in the range $[1, 4]$ ; maximum failures over the life of the software
$i$	current interval
$j$	next interval $j > i$ where $F_{ij} > 0$
$J$	maximum $j \neq t$ where $F_{ij} > 0$ .
$MSE_F$	mean square error criterion for selecting $s$ for failure count predictions
$MSE_r$	mean square error criterion for selecting $s$ for remaining failure predictions
$MSE_T$	mean square error criterion for selecting $s$ for time to next failure predictions
$p(t)$	fraction of remaining failures predicted at time $t$
$Q(t)$	operational quality predicted at time $t$ ; the complement of $p(t)$ ; the degree to which software is free of remaining faults (failures)
$r_c$	critical value of remaining failures; used in computing RCM $r(t_i)$
$r(t)$	remaining failures predicted at time $t$
$r(t_i)$	remaining failures predicted at total test time $t_i$
${}^a r(T_F, t)$	reduction in remaining failures that would be achieved if the software were executed for a time $T_F$ , predicted at time $t$

RCM $r(t_t)$	risk criterion metric for remaining failures at total test time $t_t$
RCM $T_F(t_t)$	risk criterion metric for time to next failure at total test time $t_t$
$s$	starting interval for using observed failure data in parameter estimation
$s^*$	optimal starting interval for using observed failure data, as determined by MSE criterion
$t$	cumulative time in the range $[1,t]$ ; last interval of observed failure data; current interval
$t_m$	mission duration (end time-start time); used in computing RCM $T_F(t_t)$
$t_t$	total test time (observed or predicted)
$T_F(t)$	time to next failure(s) predicted at time $t$
$T_F(t_t)$	time to next failure predicted at total test time $t_t$
$T_F(a_r, t)$	time to next $N$ failures that would be achieved if remaining failures were reduced by $a_r$ , predicted at time $t$
$T_{ij}$	time since interval $i$ to observe number of failures $F_{ij}$ during interval $j$ ; used in computing $MSE_T$
$X_i$	observed failure count in the range $[1,i]$
$X_{s-1}$	observed failure count in the range $[1,s-1]$
$X_{s,t}$	observed failure count in the range $[s,t]$
$X_{s,t_1}$	observed failure count in the range $[s,t_1]$
$X_t$	observed failure count in the range $[1,t]$

$X_{t_1}$  observed failure count in the range  $[1, t_1]$

## **2. Criteria for Safety**

If we define our safety goal as the reduction of failures that would cause loss of life, loss of mission, or abort of mission to an acceptable level of risk [11], then for software to be ready to deploy, after having been tested for total time  $t_t$ , we must satisfy the following criteria:

$$1) \text{ predicted remaining failures } r(t_t) < r_c, \quad (1)$$

where  $r_c$  is a specified critical value, and

$$2) \text{ predicted time to next failure } T_F(t_t) > t_m, \quad (2)$$

where  $t_m$  is mission duration.

For systems that are tested and operated continuously like the *Shuttle*,  $t_t$ ,  $T_F(t_t)$ , and  $t_m$  are measured in execution time. Note that, as with any methodology for assuring software safety, we cannot guarantee safety. Rather, with these criteria, we seek to reduce the risk of deploying the software to an acceptable level.

### **2.1 Remaining Failures Criterion**

Using *assumption 1* that the faults that cause failures are removed (this is the case for the *Shuttle*), *criterion 1* specifies that the residual failures and faults must be reduced to a level where the risk of operating the software is acceptable. As a practical matter, we suggest  $r_c = 1$ . That is, the goal would be to reduce the expected *remaining failures* to less than one before deploying the software. The reason for this choice is that one or more *remaining failures*

would constitute unacceptable risk for safety critical systems. This is the threshold used by the *Shuttle* software managers. One way to specify  $r_c$  is by failure severity level (e.g., *severity level 1* for life threatening failures). Another way, which imposes a more demanding safety requirement, is to specify that  $r_c$  represents *all* severity levels. For example,  $r(t) < 1$  would mean that  $r(t)$  must be less than one failure, *independent* of severity level.

If we predict  $r(t) \geq r_c$ , we would continue to test for a total time  $t' > t$  that is predicted to achieve  $r(t') < r_c$ , using *assumption 2* that we will experience more failures and correct more faults so that the *remaining failures* will be reduced by the quantity  $r(t) - r(t')$ . If the developer does not have the resources to satisfy the criterion or is unable to satisfy the criterion through additional testing, the risk of deploying the software prematurely should be assessed (see the next section). We know from Dijkstra's dictum that we cannot demonstrate the absence of faults [6]; however we can reduce the risk of failures occurring to an acceptable level, as represented by  $r_c$ . This scenario is shown in Figure 1. In *case A* we predict  $r(t) < r_c$  and the mission begins at  $t$ . In *case B* we predict  $r(t) \geq r_c$  and postpone the mission until we test for total time  $t'$  and predict  $r(t') < r_c$ . In both cases *criterion 2*) must also be satisfied for the mission to begin.

## **2.2 Time to Next Failure Criterion**

*Criterion 2* specifies that the software must survive for a time greater than the duration of the mission. If we predict  $T_F(t) \leq t_m$ , we would continue to test for a total time  $t'' > t$  that is predicted to achieve  $T_F(t'') > t_m$ , using *assumption 2* that we will experience more failures and

correct more faults so that the *time to next failure* will be increased by the quantity  $T_F(t'') - T_F(t_1)$ . Again, if it is infeasible for the developer to satisfy the criterion for lack of resources or failure to achieve test objectives, the risk of deploying the software prematurely should be assessed (see the next section). This scenario is shown in Figure 2. In *case A* we predict  $T_F(t_1) > t_m$  and the mission begins at  $t_1$ . In *case B* we predict  $T_F(t_1) \# t_m$  and postpone the mission until we test for total time  $t''$  and predict  $T_F(t'') > t_m$ . In both cases *criterion 1*) must also be satisfied for the mission to begin. If neither criterion is satisfied, we test for a time which is the greater of  $t_1'$  or  $t_1''$ .

### **3. Risk Assessment**

The amount of *total test time*  $t_1$  can be considered a measure of the degree to which software reliability goals have been achieved. This is particularly the case for systems like the *Shuttle* where the software is subjected to continuous and rigorous testing for several years in multiple facilities, using a variety of operational and training scenarios (e.g., by Lockheed-Martin in Houston, by NASA in Houston for astronaut training, and by NASA at Cape Kennedy). If we view  $t_1$  as an input to a risk reduction process, and  $r(t_1)$  and  $T_F(t_1)$  as the outputs, we can portray the process as shown in Figure 3, where  $r_c$  and  $t_m$  are shown as "risk criteria levels" of safety that control the process. While we recognize that *total test time* is not the only consideration in developing test strategies and that there are other important factors, like the consequences for reliability and cost, in selecting test cases [20], nevertheless, for the foregoing reasons, *total test time* has been found to be strongly positively correlated with

reliability growth for the *Shuttle* [15].

### **3.1 Remaining Failures**

We can formulate the mean value of the *risk criterion metric* (RCM) for *criterion 1* as follows:

$$\text{RCM } r(t_i) = (r(t_i) - r_c) / r_c = (r(t_i) / r_c) - 1 \quad (3)$$

We plot equation (3) in Figure 4 as a function of  $t_i$  for  $r_c=1$ , where *positive*, *zero*, and *negative* values correspond to  $r(t_i) > r_c$ ,  $r(t_i) = r_c$ , and  $r(t_i) < r_c$ , respectively. In Figure 4, these values correspond to the following regions: *UNSAFE* (i.e., above the X-axis predicted *remaining failures* are greater than the "safe" value); *NEUTRAL* (i.e., on the X-axis predicted *remaining failures* equal to the "safe" value); and *SAFE* (i.e., below the X-axis predicted *remaining failures* are less than the "safe" value).

This graph is for the *Shuttle operational increment OID*. In this example we see that at approximately  $t_i=57$  the risk transitions from the *UNSAFE* region to the *SAFE* region.

### **3.2 Time to Next Failure**

Similarly, we can formulate the mean value of the *risk criterion metric* (RCM) for *criterion 2* as follows:

$$\text{RCM } T_F(t_i) = (t_m - T_F(t_i)) / t_m = 1 - (T_F(t_i)) / t_m \quad (4)$$

We plot equation (4) in Figure 5 as a function of  $t_i$  for  $t_m=8$  days (a typical mission duration time for this OI), where *positive*, *zero*, and *negative* risk corresponds to  $T_F(t_i) < t_m$ ,  $T_F(t_i) = t_m$ , and  $T_F(t_i) > t_m$ , respectively. In Figure 5, these values correspond to the following regions:

*UNSAFE* (i.e., above the X-axis predicted *time to next failure* is less than the "safe" value); *NEUTRAL* (i.e., on the X-axis predicted *time to next failure* is equal to the "safe" value); and *SAFE* (i.e., below the X-axis predicted *time to next failure* is greater than the "safe" value).

This graph is for the *Shuttle operational increment OIC*. In this example we see that at all values of  $t_i$  the RCM is in the *SAFE* region.

#### **4. Approach to Prediction**

In order to support our safety goal and to assess the risk of deploying the software, we make various reliability and quality predictions. In addition, we use these predictions to perform tradeoff analysis between reliability and *total test time*. Thus, our approach is to use a software reliability model to predict the following: 1) *maximum failures, remaining failures*, and *operational quality* (as defined in the next section); 2) *time to next failure* (beyond the last observed failure); 3) *total test time* necessary to achieve required levels of *remaining failures* (fault) level, *operational quality*, and *time to next failure*; and 4) tradeoffs between increases in levels of reliability and quality with increases in testing.

#### **5. Prediction Equations**

The following prediction equations are based on the *Schneidewind Software Reliability Model* [1, 14, 15, 16], one of the four models recommended in the *AIAA Recommended Practice for Software Reliability* [1]. These equations use *assumptions 4-7* in the *Introduction*. We derive these equations in the next section. . We apply them to analyze the reliability of the *Shuttle* flight software. All predictions are mean values.

Because the flight software is run continuously, around the clock, in simulation, test, or flight, "time" refers to continuous execution time and *total test time* refers to execution time that is used for testing. Failure count intervals are equal to 30 days of continuous execution time. This interval is long because the *Shuttle* software is tested for several years; a 30 day interval length is a convenient for recording failures for software that is tested this long.

In the following equations, the parameter  $\acute{a}$  is the failure rate at the beginning of interval  $s$ ; the parameter  $\hat{a}$  is the negative of derivative of failure rate divided by failure rate (i.e., relative failure rate);  $t$  is the last interval of observed failure data;  $s$  is the starting interval for using observed failure data in parameter estimation that will result in the best estimates of  $\acute{a}$  and  $\hat{a}$  and the most accurate predictions [14];  $X_{s-1}$  is the observed failure count in the range  $[1, s-1]$ ;  $X_{s,t}$  is the observed failure count in the range  $[s, t]$ ; and  $X_t = X_{s-1} + X_{s,t}$ . These failure count interval relationships are shown in Figure 6; also shown is *total test time*  $t$ . Failures are counted against *operational increments* (OIs). Data from four *Shuttle* OI's, designated *OIA*, *OIB*, *OIC*, and *OID* are used in this analysis.

### **5.1 Cumulative Failures**

When maximum likelihood estimates are obtained for the parameters  $\acute{a}$  and  $\hat{a}$ , with  $s$  as the starting interval for using observed failure data, we obtain the predicted *failure count in the range  $[s, t]$* :

$$F_{s,t} = (\acute{a}/\hat{a}) [1 - \exp(-\hat{a}((t-s+1)))] \quad (5)$$

Furthermore, if we add  $X_{s-1}$ , the observed failure count in the range  $[1, s-1]$ , we obtain

predicted *failure count in the range*  $[1, t]$ :

$$F(t) = (\hat{\lambda}/\hat{\lambda}) [1 - \exp(-\hat{\lambda}((t-s+1)))] + X_{s-1} \quad (6)$$

### **5.2 Failures in an Interval Range**

If we set  $t/t_2$  and subtract  $X_{t_1} = X_{s-1} + X_{s,t_1}$ , the observed failure count in the range  $[1, t_1]$ , from equation (6), we obtain the predicted *failure count in the range*  $[t_1, t_2]$ :

$$F(t_1, t_2) = (\hat{\lambda}/\hat{\lambda}) [1 - \exp(-\hat{\lambda}((t_2-s+1)))] - X_{s,t_1} \quad (7)$$

### **5.3 Maximum Failures**

If we let  $t=4$  in equation (6), we obtain the predicted *failure count in the range*  $[1, 4]$  (i.e., *maximum failures* over the life of the software):

$$F(4) = \hat{\lambda}/\hat{\lambda} + X_{s-1} \quad (8)$$

### **5.4 Remaining Failures**

To obtain predicted *remaining failures*  $r(t)$  at time  $t$ , we subtract  $X_t = X_{s-1} + X_{s,t}$  from equation (8):

$$r(t) = (\hat{\lambda}/\hat{\lambda}) - X_{s,t} = F(4) - X_t \quad (9)$$

$r(t)$  can also be expressed as a function of *total test time*  $t_t$  by substituting equation (5) into equation (9) and setting  $t/t_t$ :

$$r(t_t) = (\hat{\lambda}/\hat{\lambda}) (\exp-\hat{\lambda}[t_t - (s-1)]) \quad (10)$$

### **5.5 Fraction of Remaining Failures:**

If we divide equation (9) by equation (8), we obtain *fraction of remaining failures*

predicted at time t:

$$p(t) = r(t)/F(4) \quad (11)$$

### **5.6 Operational Quality**

The *operational quality* of software is the complement of  $p(t)$ . It is the degree to which software is free of remaining faults (failures), using *assumption 1* that the faults that cause failures are removed. It is predicted at time t as follows:

$$Q(t) = 1 - p(t) \quad (12)$$

### **5.7 Total Test Time to Achieve Specified Remaining Failures**

The predicted *total test time* required to achieve a specified *number of remaining failures* at  $t$ ,  $r(t)$ , is obtained from equation (10) by solving for  $t_t$ :

$$t_t' [\log[\hat{\alpha}/(\hat{\alpha}[r(t)])]]/\hat{\alpha}^{(s+1)} \quad (13)$$

### **5.8 Time to Next Failure**

By substituting  $t_2 = t + T_F(t)$  in equation (7), setting  $t_1/t$ , defining  $F_t = F(t, t + T_F)$ , and solving for  $T_F(t)$ , we obtain the predicted *time for the next  $F_t$  failures* to occur, when the current time is  $t$ :

$$T_F(t)' [(\log[\hat{\alpha}/(\hat{\alpha}(X_{s,t} \% F_t))])/\hat{\alpha}] \& (t \& s \% 1) \quad (14)$$

for  $(\hat{\alpha}/\hat{\alpha}) > (X_{s,t} \% F_t)$

The terms in  $T_F(t)$  have the following definitions:

- t: Current interval;
- $X_{s,t}$ : Observed failure count in the range [s,t]; and
- $F_t$ : Given number of failures to occur after interval t.

We consider equations (5)-(11) and (14) to be predictors of reliability that are related to safety; equation (13) represents the predicted *total test time* required to achieve stated safety goals. If a quality requirement is stated in terms of *fraction of remaining failures*, the definition of  $Q$  as *Operational Quality*, equation (12), is consistent with the IEEE definition of quality: the *degree* to which a system, component, or process meets specified requirements [9]. For example, if a reliability specification requires that software is to have no more than 5% remaining failures (i.e.,  $p=.05$ ,  $Q=.95$ ) after testing for a total of  $t_t$  intervals, then a predicted  $Q$  of .90 would indicate the *degree* to which the software meets specified requirements.

### **5.9 Relating Time to Next N Failures and Remaining Failures Predictions**

Although we have shown the risk analysis and prediction equations for *remaining failures* and *time to next failure* separately, it would be useful to combine these quantities in one equation so that we can predict the effect on one quantity for a given change in the other. In particular we want to predict, at time  $t$ , the *time to the next N failures*,  $T_F(a_r, t)$ , that would be achieved if *remaining failures* were reduced by  $a_r$ . We use *assumption 1* that  $N=a_r$ ; that is, faults that cause failures are removed. When  $N=1$ , we have the familiar *time to next failure*. When  $N>1$ ,  $T_F(a_r, t)$  is interpreted as cumulative execution time for the  $N$  failures to

occur. Conversely, we want to predict, at time  $t$ , the *reduction in remaining failures*,  ${}^a r(T_F, t)$ , that would be achieved if the software were executed for a time  $T_F$ . This relationship is derived by using equation (10) and setting  ${}^a r = r(t_1) - r(t)$ ,  $t_1 = t_1 + {}^a t$ , and  $t_1/t$ , and solving for  ${}^a t/T_F({}^a r, t)$ :

$$T_F({}^a r, t) = (-1/\hat{\alpha}) [\log[1 - ((\hat{\alpha} {}^a r/\acute{\alpha})(\exp(\hat{\alpha}(t-s+1))))]] \quad (15)$$

for  $((\hat{\alpha} {}^a r/\acute{\alpha})(\exp(\hat{\alpha}(t-s+1)))) < 1$ .

Equation (15) is analogous to equation (14). Also,  ${}^a r$  in equation (15) is analogous to  $F_t$  in equation (14), if we use *assumption 1* that the faults that cause the  $F_t$  failures are removed, with a corresponding *reduction in remaining failures*. The two equations produce the same result for the same parameter values. Equation (15) has the advantage of being a simpler computation because it does not require the observed data vector  $X_{s,t}$ , which is used in equation (14). Also, equation (15) is convenient to use for trading off *time to next  $N$  failures* against *reduction in remaining failures*, and the effort and the *total test time* implicit in making the reductions.

We can invert equation (15) to solve for the *reduction in remaining failures* that would be achieved by executing the software for a time  $T_F$ .

$${}^a r(T_F, t) = (\acute{\alpha}/\hat{\alpha}) [\exp(-\hat{\alpha}(t-s+1))] [1 - \exp(-\hat{\alpha}(T_F))] \quad (16)$$

## **6. Criterion for Optimally Selecting Failure Data**

The first step in identifying the optimal value of  $s$  ( $s^*$ ) is to estimate the parameters  $\acute{\alpha}$  and  $\hat{\alpha}$  for each value of  $s$  in the range  $[1, t]$  where convergence can be obtained [1, 14, 16]. Then

the *Mean Square Error* (MSE) criterion is used to select  $s^*$ , the failure count interval that corresponds to the minimum MSE between predicted and actual failure counts ( $MSE_F$ ), *time to next failure* ( $MSE_T$ ), or *remaining failures* ( $MSE_r$ ), depending on the type of prediction. The first two were reported in [14]. In this paper we develop  $MSE_r$ .  $MSE_r$  is also the criterion for *maximum failures* ( $F(4)$ ) and *total test time* ( $t$ ) because the two are functionally related to *remaining failures* ( $r(t)$ ); see equations 9 and 13. We also show  $MSE_T$  because it is used in predictions that involve *time to next failure*:  $T_F(t)$ ,  $T_F(\hat{I}r, t)$ , and  ${}^a r(T_F, t)$ . Once  $\hat{a}$ ,  $\hat{\lambda}$ , and  $s$  are estimated from observed counts of failures, the foregoing predictions can be made. The reason MSE is used to evaluate which triple  $(\hat{a}, \hat{\lambda}, s)$  is best in the range  $[1, t]$  is that research has shown that because the product and process change over the life of the software, old failure data (i.e.,  $s=1$ ) are not as representative of the current state of the product and process as the more recent failure data (i.e.,  $s>1$ ) [14]. The optimal values of  $s$  ( $s^*$ ) that were used in the risk analysis and prediction examples are shown in Tables 1-4.

The *Statistical Modeling and Estimation of Reliability Functions for Software* (SMERFS) [7] is used for all predictions except  $t_r$ ,  $T_F({}^a r, t)$ , and  ${}^a r(T_F, t)$ , which are not implemented in SMERFS.

### **6.1 Mean Square Error Criterion for Remaining Failures**

Although we can never know whether additional failures may occur, nevertheless we can form the difference between two equations for  $r(t)$ : (9), which is a function of predicted *maximum failures* and the observed failures, and (10), which is a function of *total test time*,

and apply the MSE criterion. This yields the following Mean Square Error ( $MSE_r$ ) criterion for number of *remaining failures*:

$$MSE_r = \frac{\sum_{i=1}^t [F(i) - X_i]^2}{t+1} \quad (17)$$

where  $F(i)$  is the predicted *failure count in the range [1,i]* and  $X_i$  is the observed failure count in the range [1,i].

### **6.2 Mean Square Error Criterion for Time to Next Failure(s)**

The Mean Square Error ( $MSE_T$ ) criterion for *time to next failure(s)*, which was derived in [14], is given by equation (18):

$$MSE_T = \frac{\sum_{i=1}^{J+1} \left[ \frac{\log[\hat{\lambda}/(\hat{\lambda} - (X_{s,i} - F_{ij}))]}{\hat{\lambda} - (i+1)} \right]^2 T_{ij}}{(J+1)} \quad (18)$$

for  $(\hat{\lambda} - (i+1)) > (X_{s,i} - F_{ij})$

The terms in  $MSE_T$  have the following definitions:

$i$ : Current interval;

$j$ : Next interval  $j > i$  where  $F_{ij} > 0$ ;

$X_{s,i}$ : Observed failure count in the range [s,i];

$F_{ij}$ : Observed failure count during interval  $j$  since interval  $i$ ;  $T_{ij}$ : Time since  $i$  to observe number of failures  $F_{ij}$  during  $j$  (i.e.,  $T_{ij} = j - i$ )

$t$ : The last interval of observed failure data; and

$J$ : Maximum  $j \# t$  where  $F_{ij} > 0$ .

## 7. Relating Testing to Reliability and Quality

### 7.1 Predicting Total Test Time and Remaining Failures

We use equation (8) to predict *maximum failures* ( $F(4)=11.76$ ) for *Shuttle OIA*. Using given values of  $p$  and equation (11) and setting  $t/t_r$ , we predict  $r(t)$  for each value of  $p$ . The values of  $r(t_r)$  are the predictions of *remaining failures* after the OI has been executed for *total test time*  $t_r$ . Then we use the values of  $r(t_r)$  and equation (13) to predict corresponding values of  $t_r$ . The results are shown in Figure 7, where  $r(t_r)$  and  $t_r$  are plotted against  $p$  for *OIA*. Note that required *total test time*  $t_r$  rises very rapidly at small values of  $p$  and  $r(t_r)$ . Also note that the maximum value of  $p$  on the plot corresponds to  $t_r=18$  and that smaller values correspond to *future* values of  $t_r$  (i.e.,  $t_r > 18$ ).

### 7.2 Predicting Operational Quality

Equation (12) is a useful measure of the *operational quality* of software because it measures the degree to which faults have been removed from the software (using *assumption 1* that the faults that cause failures are removed), relative to predicted *maximum failures*. We call this type of quality *operational* (i.e., based on executing the software) to distinguish it from static quality (e.g., based on the complexity of the software).

Using given values of  $p$  and equations (11) and (12) and setting  $t/t_r$ , we compute  $r(t_r)$  and  $Q$ , respectively. The values of  $r(t_r)$  are then used in equation (13) to compute  $t_r$ . The

corresponding values of  $Q$  and  $t_t$  are plotted in Figure 8 as *Operational Quality* and *Total Test Time*, respectively for *OIA*. We again observe the asymptotic nature of the testing relationship in the great amount of testing required to achieve high levels of quality.

### **7.3 Predicting Time to Next Failure**

First, we show the actual *time to next failure* in Figure 9 for *OIA* on the solid curve that has occurred in the *execution time* range  $t=[1,18]$ , where one failure occurred at  $t=4, 14,$  and  $18$ , and two failures occurred at  $t=8$  and  $10$ . All failures were *Severity Level 3*: "Workaround available; minimal effect on procedures". The way to read the graph is as follows: If we take a given failure, *Failure 1*, for example, it occurs at  $t=4$ ; therefore, at  $t=1$  the *time to next failure*=3 ( $4-1$ ); at  $t=2$  the *time to next failure*=2 ( $4-2$ ); at  $t=4$  *Failure 1* occurs, so the *time to next failure*=4 ( $8-4$ ) now refers to *Failure 2*, etc. Next, using equation (14), we predict the *time to next failure*  $T_F(18)$  to be 4 (3.87 rounded) on the dashed curve. Based on the foregoing, this prediction indicates we should continue testing if  $T_F(18)=3.87 \# t_m$  (mission duration).

### **7.4 Predicting Tradeoffs of Time to Next N Failures with Reduced Remaining Failures**

By using equation (15), we can predict *time to next N failures*,  $T_F(a_r, t)$ , as a function of *reduction in remaining failures*,  $a_r$ . This is shown in Figure 10 for *OIA*, where, for example, with  $a_r=1$ , we predict  $T_F(1,18)=3.87$  (i.e., a *reduction in remaining failures* of 1 corresponds to achieving a *time to next failure* of 3.87 intervals from the current interval 18).

Conversely, by using equation (16), we predict *reduction in remaining failures*,  ${}^a r(T_F, t)$ , as a function of *time to next failure*,  $T_F$ . This is shown in Figure 11 for *OIA*, where, for example, with  $T_F=3.87$ , we predict  ${}^a r(3.87, 18)=1$  (i.e., executing *OIA* for a *time to next failure* of 3.87 intervals from the current interval 18 corresponds to achieving a *reduction in remaining failures* of 1). We provide further elaboration of these graphs in the next section.

### **8. Making Safety Decisions**

In making the decision about how long to test,  $t$ , we apply our safety criteria and risk assessment approach. We use Table 1 to illustrate the process. For  $t=18$  (when the last failure occurred on *OIA*),  $r_c=1$ , and  $t_m=8$  days (.267 intervals), we show *remaining failures*, *RCM for remaining failures*, *time to next failure*, *RCM for time to next failure*, and *operational quality*. These results indicate that safety *criterion 2* is satisfied but not *criterion 1* (i.e., *UNSAFE* with respect to *remaining failures*); also *operational quality* is low.

By looking at Figure 10 and Table 1, we see that if we reduce *remaining failures*  $r(18)$  by 1 from 4.76 to 3.76 (non-integer values are possible because the predictions are mean values), the predicted *time to next failure* that would be achieved is  $T_F(18)=3.87$  intervals. These predictions satisfy *criterion 2* (i.e.,  $T_F(18)=3.87 > t_m=.267$ ) but not *criterion 1* (i.e.,  $r(18)=4.76 > r_c=1$ ). Note also in Figure 10 and Table 1 that *fraction of remaining failures*  $p=1-Q=.40$  at  $r(18)=4.76$ . Now, if we continue testing for a total time  $t=52$  intervals, as shown in Figure 10 and Table 1, and reduce *remaining failures* from 4.76 to .60, the predicted *time to next 4.16 failures* that would be achieved is 33.94 (34, rounded) intervals. This

corresponds to  $t_t=18+34=52$  intervals. That is, if we test for an additional 34 intervals, starting at interval 18, we would expect to experience 4.16 failures. These predictions now satisfy *criterion 1* because  $r(52)=.60 < r_c=1$ . Note also in Figure 10 and Table 1 that *fraction of remaining failures*  $p=1-Q=.05$  at  $r(52)=.60$ . Using the converse of the relationship in Figure 10, provides another perspective, as shown in Figure 11, where we see that if we continue to test for an additional  $T_F=34$  intervals, starting at interval 18, the predicted *reduction in remaining failures* that would be achieved is 4.16 or  $r(52)=.60$ .

Lastly, Figure 12 shows the *Launch Decision*, relevant to the *Shuttle*, (or, generically, *the Deployment Decision*), where *remaining failures* are plotted against *total test time* for OIA. With these results in hand, the software manager can decide whether to deploy the software depending on factors such as predicted *remaining failures*, as shown in Figure 12, along with considering other factors such as the trend in reported faults over time, inspection results, etc.. If testing were to continue until  $t_t=52$ , the predictions in Figure 12 and Table 1 would be obtained. These results show that *criterion 1* is now satisfied (*i.e.*, *SAFE*) and *operational quality* is high. We also see from Figure 12 that at this value of  $t_t$ , further increases in  $t_t$  would not result in a significant increase in reliability and safety. Also note that at  $t_t=52$  it is not feasible to make a prediction of  $T_F(52)$  because the predicted *remaining failures* is less than one.

**Table 1 Safety Criteria Assessment**

**OIA**

$r_c=1$ <span style="float: right;"><math>t_m=8</math></span>									
days									
$t_t$	$\hat{a}$	$\hat{\lambda}$	$s^*$	$r(t_t)$	RCM	$s^*$	$T_F(t_t)$	RCM	Q
18	.534	.061	9	4.76	3.76	9	3.87	-13.49	.60
52	.534	.061	9	.60	-.40	9	*	*	.95

30 day Total Test Time and Time to Next Failure Intervals.

\* Cannot predict because predicted Remaining Failures is less than one.

## 9. Summary of Predictions and Validation

### 9.1 Predictions

Table 2 shows a summary of remaining and maximum failure predictions compared with actual failure data, where available, for *OIA*, *OIB*, *OIC*, and *OID*. Because we do not know the *actual* remaining and maximum failures, we use *assumption 3: remaining failures* are "zero" for those OI's (*B*, *C*, and *D*) that were executed for extremely long times (years) with no additional failure reports; correspondingly, for these OI's, we use *assumption 3* that *maximum failures* equals total observed failures.

Table 2

**Predicted Remaining and Maximum Failures versus Actuals**

	$t_t$	$s^*$	$\hat{\alpha}$	$\hat{\alpha}$	$r(t_t)$	Actual r	F(4)	Actual F
<b>OIA</b>	18	9	.534	.061	4.76	? <sup>A</sup>	11.76	7 <sup>A</sup>
<b>OIB</b>	20	1	1.69	.131	0.95	1 <sup>B</sup>	12.95	13 <sup>B</sup>
<b>OIC</b>	20	7	1.37	.126	1.87	2 <sup>C</sup>	12.87	13 <sup>C</sup>
<b>OID</b>	18	6	.738	.051	7.36	4 <sup>D</sup>	17.36	14 <sup>D</sup>

30 day Total Test Time Intervals

**Time of last recorded failure:**

- A. No additional failures have been reported after 17.17 intervals.
- B. The last recorded failure occurred at 63.67 intervals.
- C. The last recorded failure occurred at 43.80 intervals.
- D. The last recorded failure occurred at 65.03 intervals.

Table 3 shows a summary of *total test time* and *time to next failure* predictions compared with actual execution time data, where available, for *OIA*, *OIB*, *OIC*, and *OID*.

Table 3

**Predicted Total Test Time and Time to Next Failure versus Actuals**

	$s^*$	$t_t(r=1)$	Actual $t_t$	$t$	$s^*$	$T_F(t)$	Actual $T_F$
<b>OIA</b>	9	43.59	?	18	9	3.9	?
<b>OIB</b>	1	*	63.67	20	*	*	43.67
<b>OIC</b>	7	24.98	27.07	20	5	4.2	7.63
<b>OID</b>	6	56.84	58.27	18	5	6.4	6.2

30 day Total Test Time and Time to Next Failure Intervals.

\* Cannot predict because predicted Remaining Failures is less than one.

**Additional Predictions for OID:**

The following are additional predictions of total test time for OID that are not listed in Table 3:  $t_t(r=2)=43.35$ , Actual=45.17;  $t_t(r=3)=35.47$ , Actual=23.70.

Table 4 shows a summary of the predictions of *time to next failure* for a given *reduction in remaining failures* of 1 and the predictions of *reduction in remaining failures* for given *time to next failure* compared with actual execution time and failure data, where available, for *OIA*, *OIB*, *OIC*, and *OID*.

Table 4

**Predicted Tradeoffs of Time to Next Failure with Reduced Remaining Failures  
versus Actuals**

	t	s*	$\hat{a}$	$\hat{a}$	$T_F(\hat{r}=1,t)$	Actual	$(T_F,t)$	$\hat{r}(T_F,t)$	Actual
<b>OIA</b>	18	9	.534	.061	3.87	?	3.87	1.00	?
<b>OIB</b>	20	1	1.69	.131	*	43.67	43.67	.95	1.0
<b>OIC</b>	20	5	1.34	.096	4.16	7.63	7.63	1.58	1.0
<b>OID</b>	18	5	1.61	.137	6.35	6.20	6.20	.99	1.0

30 day Total Test Time and Time to Next Failure Intervals.

\* Cannot predict because predicted Remaining Failures is less than one.

## **9.2 Validation**

A total of 18 predictions were made across Tables 2, 3, and 4, where there was an actual value to compare: three  $r(t)$ , four  $F(4)$ , four  $t_r$ , two  $T_F(t)$ , two  $T_F(\hat{r},t)$ , and three  $\hat{r}(T_F,t)$ . The mean relative error (mean of (actual-predicted)/actual) of prediction is 22.92% and the standard deviation is 27.61%. In making these predictions we note both the sparsity of post-delivery failures and the extremely long test times for *Shuttle* flight software, as summarized in Table 5. See the *Appendix* for a listing of the failure data. Despite the fact that the *Schneidewind Software Reliability Model* uses optimal selection of failure data, and thus less than the full set of data, there must be a minimum number of failures to *start the parameter estimation process*, understanding that the model will then select the optimal value of  $s(s^*)$ . Thus, given the sparsity of the data, all failures in Table 5 were used in parameter estimation,

regardless of their severity. Furthermore, as described earlier, a more conservative risk assessment is produced if all categories of failures are included in the analysis.

**Table 5**

**Failure Distribution by Severity Code**

	<b>Severity 2</b>	<b>Severity 3</b>	<b>Severity 4</b>	<b>Maximum</b>	<b>Total</b>
	<b>Failures</b>	<b>Failures</b>	<b>Failures</b>	<b>Failures</b>	<b>Test Time</b>
<b>OIA</b>	0	7	0	7	18
<b>OIB</b>	5	8	0	13	64
<b>OIC</b>	3	6	2	13*	44
<b>OID</b>	6	8	0	14	66

30 day Total Test Time Intervals.

\* Unknown Severity for two failures

There are no post-delivery Severity 1 or 5 failures in the above Operational Increments.

APPENDIXObserved Failure Counts

(Interval i = 30 days execution time)

<u>i</u>	<u>OIA</u>	<u>OIB</u>	<u>OIC</u>	<u>OID</u>
1	0	1	0	0
2	0	1	0	0
3	0	1	0	0
4	1	2	0	0
5	0	1	0	3
6	0	0	2	1
7	0	0	1	0
8	2	2	3	1
9	0	1	1	0
10	2	0	0	1
11	0	2	0	1
12	0	0	0	0
13	0	1	1	2
14	1	0	1	0
15	0	0	0	0
16	0	0	0	0
17	0	0	1	0
18	1	0	0	1
19		0	0	0
20		0	1	0
21		0	0	0
22		0	0	0
23		0	0	0
24		0	0	1
25		0	0	0
26		0	0	0
27		0	0	0
28		0	1	0
29		0	0	0
30		0	0	0
<hr/>				
31-63		0		
64		1		
<hr/>				
31-43			0	
44			1	
<hr/>				
31-45				0
46				1
47-58				0
59				1
60-65				0
66				1
<hr/>				
Totals:				
	7	13	13	14

## Acknowledgments

We acknowledge the support provided for this project by Dr. William Farr, Naval Surface Warfare Center; Ms. Alice Lee of NASA; U.S. Marine Corps Tactical Systems Support Activity; and Mr. Ted Keller and Ms. Patti Thornton of Lockheed-Martin. We also acknowledge the helpful comments of the reviewers.

## References

- [1] Recommended Practice for Software Reliability, R-013-1992, *American National Standards Institute/American Institute of Aeronautics and Astronautics*, 370 L'Enfant Promenade, SW, Washington, DC 20024, 1993.
- [2] Barry W. Boehm, "Software Risk Management: Principles and Practices", *IEEE Software*, Vol. 8, No. 1, January 1991, pp. 32-41.
- [3] C. Billings, et al, "Journey to a Mature Software Process", *IBM Systems Journal*, Vol. 33, No. 1, 1994, pp. 46-61.
- [4] Siddhartha R. Dalal and Allen A. McIntosh, "When to Stop Testing for Large Software Systems with Changing Code", *IEEE Transactions on Software Engineering*, Vol. 20, No. 4, April 1994, pp. 318-323.
- [5] Siddhartha R. Dalal and Allen A. McIntosh, "Some Graphical Aids for Deciding When to Stop Testing", *IEEE Journal on Selected Areas in Communications*, Vol. 8, No.2, February 1990, pp. 169-175.
- [6] E. W. Dijkstra, "Structured Programming", *Software Engineering Techniques*, eds. J. N. Buxton and B. Randell, NATO Scientific Affairs Division, Brussels 39, Belgium, April 1970 pp. 84-88.
- [7] William H. Farr and Oliver D. Smith, *Statistical Modeling and Estimation of Reliability Functions for Software (SMERFS) Users Guide*, NAVSWC TR-84-373, Revision 3, Naval Surface Weapons Center, Revised September 1993.
- [8] Willa Ehrlich, et al, "Determining the Cost of a Stop-Test Decision", *IEEE Software*, March 1993, pp. 33-42.
- [9] *IEEE Standard Glossary of Software Engineering Terminology*, IEEE Std 610.12.1990, The Institute of Electrical and Electronics Engineers, New York, New York, March 30, 1990.
- [10] Ted Keller, Norman F. Schneidewind, and Patti A. Thornton "Predictions for Increasing Confidence in the Reliability of the Space Shuttle Flight Software", *Proceedings of the AIAA Computing in Aerospace 10*, San Antonio, TX, March 28, 1995, pp. 1-8.
- [11] Nancy G. Leveson, "Software Safety: What, Why, and How", *ACM Computing Surveys*, Vol. 18, No. 2, June 1986, pp. 125-163.
- [12] John D. Musa and A. Frank Ackerman, "Quantifying Software Validation: When to Stop Testing?", *IEEE Software*, Vol. 6, No. 3, May 1989, pp. 19-27.
- [13] John D. Musa, et al, *Software Reliability: Measurement, Prediction, Application*, McGraw-Hill, New York, 1987.
- [14] 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.
- [15] Norman F. Schneidewind and T. W. Keller, "Application of Reliability Models to the Space Shuttle", *IEEE Software*, Vol. 9, No. 4, July 1992 pp. 28-33.
- [16] 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.
- [17] Nozer D. Singpurwalla, "Determining an Optimal Time Interval for Testing and Debugging Software", *IEEE Transactions on Software Engineering*, Vol. 17, No. 4, April 1991, pp. 313-319.
- [18] Jeffrey M. Voas and Keith W. Miller, "Software Testability: The New Verification", *IEEE Software*, Vol. 12, No. 3, May 1995, pp. 17-28.
- [19] *Webster's New Universal Unabridged Dictionary*, Second Edition, Simon and Shuster, New York, 1979.
- [20] Elaine J. Weyuker, "Using the Consequences of Failures for Testing and Reliability Assessment", *Proceedings of the Third ACM SIGSOFT Symposium on the Foundations of Software Engineering*, Washington, D.C., October 10-13, 1995, pp. 81-91.

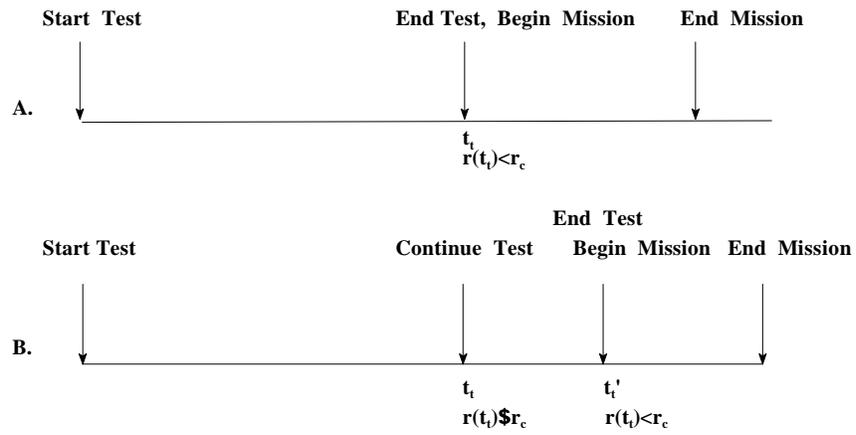


Figure 1. Remaining Failures Criterion Scenario

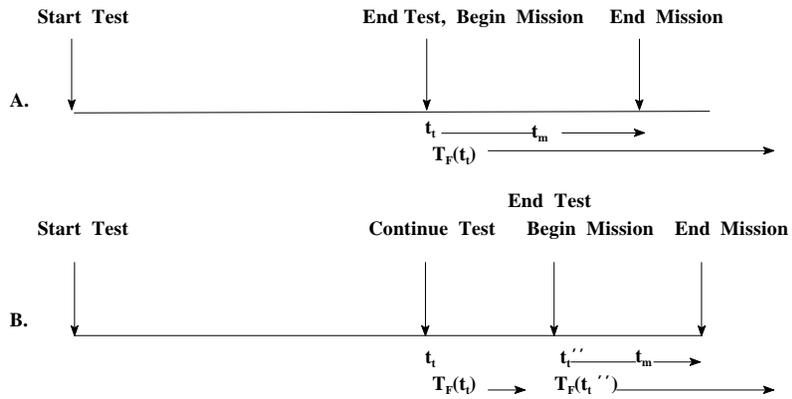


Figure 2. Time to Next Failure Criterion Scenario

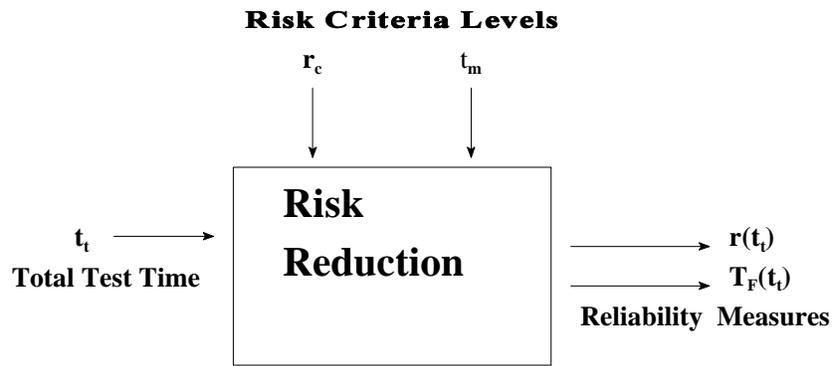


Figure 3. Risk Reduction Process

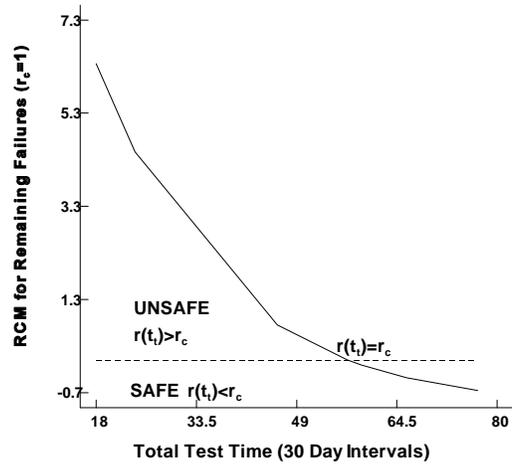


Figure 4. RCM for Remaining Failures, OID

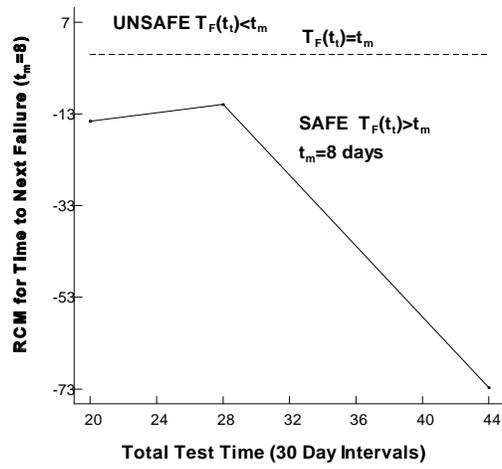
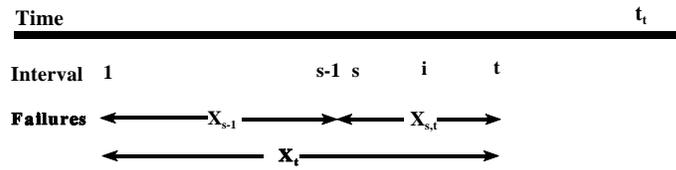
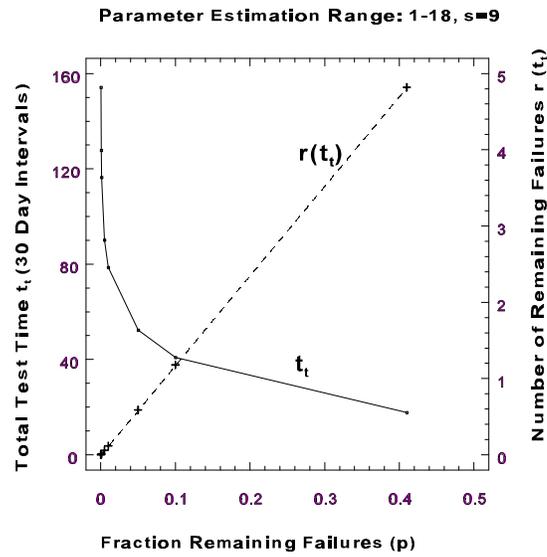


Figure 5. RCM for Time to Next Failure, OIC

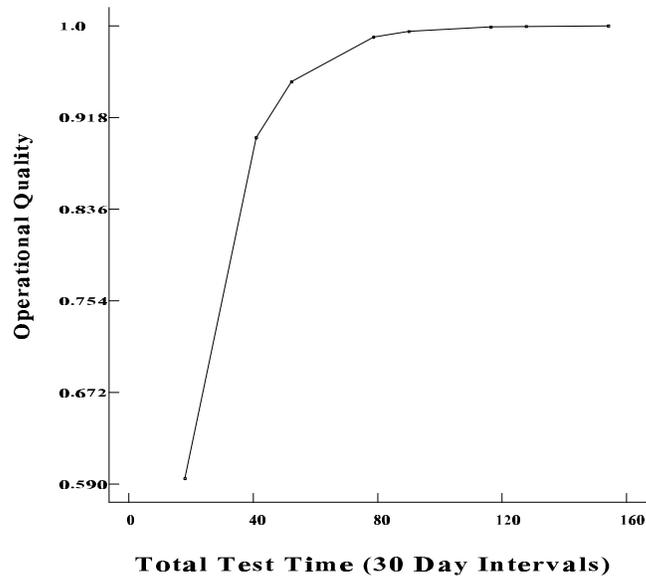


- i:** current interval
- s:** starting interval for using observed failure data in parameter estimation
- t:** the last interval of observed failure data
- $t_i$ :** total test time
- $X_{s-1}$ :** observed failure count in the range  $[1, s-1]$
- $X_{s,t}$ :** observed failure count in the range  $[s, t]$
- $X_t$ :** observed failure count in the range  $[1, t]$

Figure 6. Failure Count Interval Relationships



**Figure 7. Total Test Time & Remaining Failures vs. Fraction Remaining Failures, OIA**



**Figure 8. Operational Quality vs. Total Test Time, OIA**

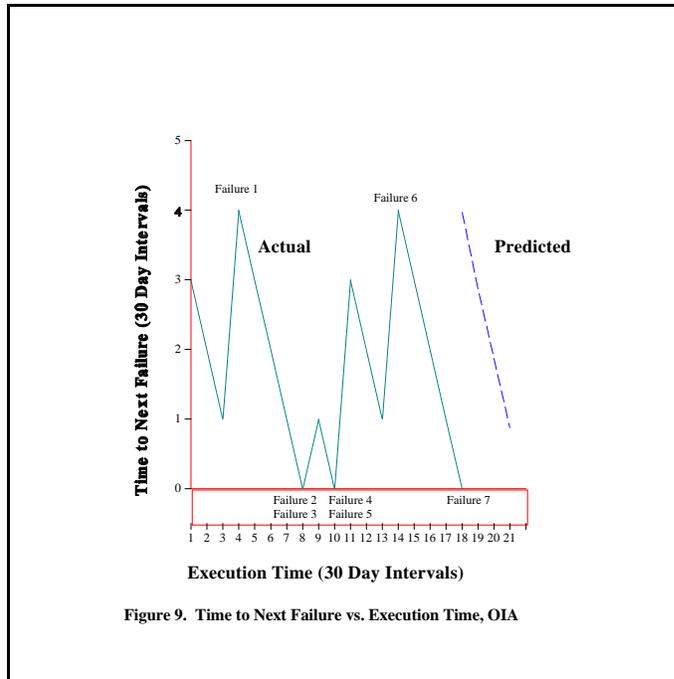


Figure 9. Time to Next Failure vs. Execution Time, OIA

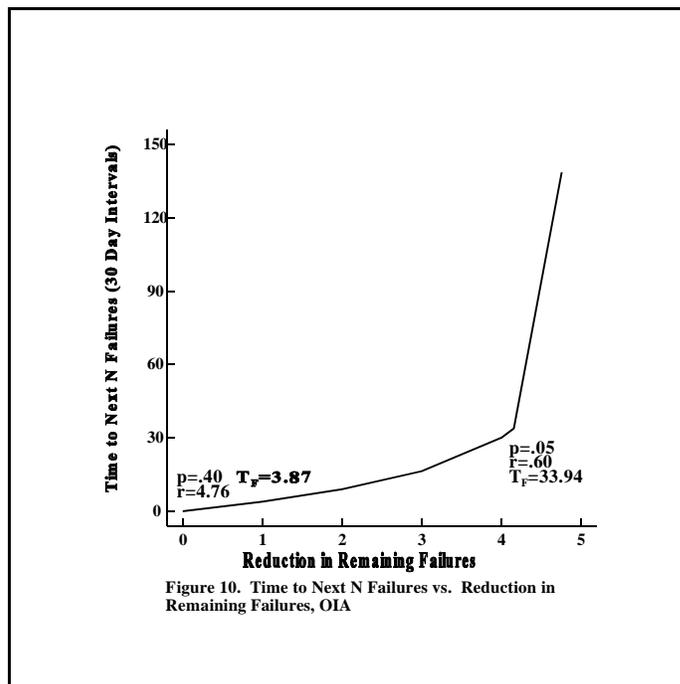


Figure 10. Time to Next N Failures vs. Reduction in Remaining Failures, OIA

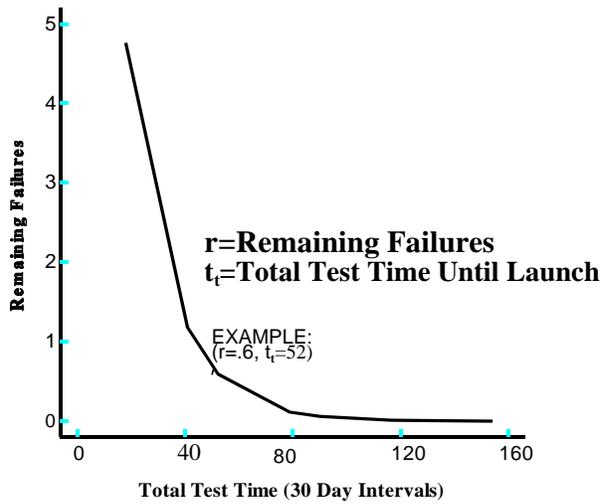
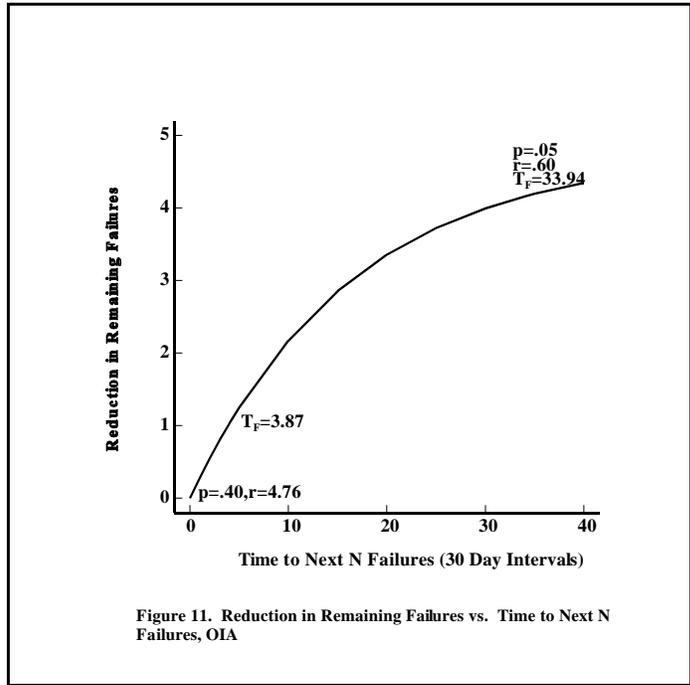


Figure 12. Launch Decision: Remaining Failures vs. Total Test Time, OIA