

Reliability and Risk Analysis for Software That Must be Safe

Norman F. Schneidewind
Code SM/Ss
Naval Postgraduate School
Monterey, CA 93943
Voice: (408) 656-2719
Fax : (408) 656-3407
Internet: schneidewind@nps.navy.mil

Abstract

Remaining failures, total failures, test time required to attain a given fraction of remaining failures, and time to next failure are useful reliability metrics for: 1) providing confidence that the software has achieved reliability 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 a program quality goal in predicting test time requirements and, conversely, as an indicator of program quality as a function of test time expended. We show how these software reliability predictions can increase confidence in the reliability of safety critical software such as the NASA Space Shuttle Primary Avionics Software.

1. Introduction

We propose that two categories of software reliability measurements and predictions be used in combination to assist in assuring the safety of the software in safety critical systems like the *NASA Space Shuttle Primary Avionics Software System*. 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, total failures, fraction of remaining failures, and test time required to attain a given number or fraction of remaining failures*. In the second category are: *time to next failure and 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 *program 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. 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 program 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 objectives [BIL94].

Loral Space Information Systems, 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 testing and operation. [KEL95]. Our prediction methodology provides bounds on *test time, remaining failures, program quality, and time to next failure* that are necessary to meet *Shuttle* software safety requirements. We also show that there is a pronounced asymptotic characteristic to the *test time* and *program 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 applications.

We note that there have been claims of the infeasibility of quantifying the reliability of life-critical real-time software [BUT93] despite the fact that this has been done for the *Shuttle* as part of a Level Five Capability Maturity Model

process for several years [BIL94]. This process involves a combination of extremely long testing at various stress levels, short mission durations, and an approach to prediction that provides high confidence that there will be no life threatening failures occurring during a *particular mission*. This is accomplished by satisfying the software safety criteria described below.

Although *remaining failures* has been discussed in general as a type of software reliability prediction [MUS87], and various stopping rules for testing have been proposed, based on the economics of testing [DAL94] and a testability criterion [VOA95], our approach is novel because we integrate safety criteria, risk analysis, and a stopping rule for testing. Furthermore, we use reliability measurements and predictions to assess whether safety goals are likely to be achieved. Thus we advocate using safety analysis and reliability analysis synergistically in a mutually supportive way rather than treat these fields as disjoint and unrelated.

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 [LEV86], then for software to be ready to deploy, after having been tested for time t_2 , we must satisfy the following criteria:

- 1) predicted remaining failures $R(t_2) < R_c$,
where R_c is a specified critical value, and (1)
- 2) predicted time to next failure $T_F(t_2) > t_m$, (2)

where t_m is mission duration.

For systems that are tested and operated continuously like the *Shuttle*, t_2 , $T_F(t_2)$, and t_m are measured in execution time. Note that, as with any methodology for assuring software safety, we can't 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

On the assumption that the faults associated with 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. If we predict $R(t_2) > R_c$, we would continue to test for a total time $t_2' > t_2$ that is predicted to achieve $R(t_2') < R_c$, on the assumption that we

will experience more failures and correct more faults so that the remaining failures will be reduced by the quantity $R(t_2) - R(t_2')$. 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 can't demonstrate the absence of faults; 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 $R(t_2) < R_c$ and the mission begins at t_2 . In *case B* we predict $R(t_2) > R_c$ and postpone the mission until we test for time t_2' and predict $R(t_2') < R_c$. In both cases *criterion 2*) must also be satisfied for the mission to begin.

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, $R(t_2) < 1$ would mean that $R(t_2)$ must be less than one failure, *independent* of severity level. We use this approach in our examples.

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 $T_F(t_2) \leq t_m$, we would continue to test for a total time $t_2' > t_2$ that is predicted to achieve $T_F(t_2') > t_m$, on the assumption that we will experience more failures and correct more faults so that the time to next failure will be increased by the quantity $T_F(t_2') - T_F(t_2)$. 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 $T_F(t_2) > t_m$ and the mission begins at t_2 . In *case B* we predict $T_F(t_2) \leq t_m$ and postpone the mission until we test for time t_2' and predict $T_F(t_2') > t_m$. 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_2' or t_2'' .

3. Risk Assessment

The amount of test execution time t_2 can be considered a measure of the maturity of the software. This is particularly the case for systems like the *Shuttle* where the software is subjected to continuous and rigorous testing for several years. If we view t_2 as an input to a risk reduction process, and $R(t_2)$ and $T_F(t_2)$ as the outputs, we can portray the process as shown in Figure 3, where R_c and t_m are shown as "levels" of safety that control the process.

3.1 Remaining Failures

We can formulate the risk of *criterion 1* as follows:

$$\text{Risk } R(t_2) = (R(t_2) - R_c) / R_c = (R(t_2) / R_c) - 1 \quad (3)$$

We plot equation (3) in Figure 4 as a function of t_2 for $R_c = 1$, where *positive*, *zero*, and *negative* risk correspond to $R(t_2) > R_c$, $R(t_2) = R_c$, and $R(t_2) < R_c$, respectively, and the *UNSAFE* and *SAFE* regions are above and below the X-axis, respectively. This graph is for the *Shuttle operational increment OID*; an operational increment (OI) is comprised of modules and configured from a series of builds to meet mission functional requirements. In this example we see that at approximately $t_2 = 57$ the risk transitions from the *UNSAFE* region to the *SAFE* region.

3.2 Time to Next Failure

Similarly, we can formulate the risk of *criterion 2* as follows:

$$\text{Risk } T_F(t_2) = (t_m - T_F(t_2)) / t_m = 1 - (T_F(t_2) / t_m) \quad (4)$$

We plot equation (4) in Figure 5 as a function of t_2 for $t_m = 8$ days (a typical mission duration time for this OI), where *positive*, *zero*, and *negative* risk corresponds to $T_F(t_2) < t_m$, $T_F(t_2) = t_m$, and $T_F(t_2) > t_m$, respectively, and the *UNSAFE* and *SAFE* regions are above and below the X-axis, respectively. This graph is for the *Shuttle operational increment OIC*. In this example we see that at all values of t_2 the risk 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 make tradeoff analysis between reliability and test time (cost). Thus our approach to reliability prediction is the following: 1) Use a software reliability model to predict *total failures*, *remaining failures*, and *operational quality*; 2) Predict the *time to next failure* (beyond the last observed failure); 3) Predict the *test time* necessary to achieve required levels of *remaining failures* (fault) level, *operational quality*, and *time to next failure*; and 4) Examine the tradeoff 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, one of the four models recommended in the *AIAA Recommended Practice for Software Reliability* [AIA93]. It is not our purpose to derive these equations because they have been derived in other publications [AIA93, SCH93, SCH92, SCH75]. Rather we apply the equations to analyze the reliability of the *Space Shuttle Primary Avionics Software*.

Because the flight software is run continuously, around the clock, in simulation, test, or flight, "time" refers to continuous execution time and test time refers to execution time that is used for testing.

In these predictions, the parameter α is the failure rate at the beginning of interval s ; the parameter β is the failure rate per failure; t is the last observed count interval; s is the index of time intervals for selecting the optimal subset of failure data that will result in the best estimates of α and β and the most accurate predictions [SCH93]; X_{s-1} is the number of failures observed from 1 through $s-1$; $X_{s,t}$ is the number observed from s through t ; and $X_t = X_{s-1} + X_{s,t}$. Failures are counted against *operational increments* (OIs). Data from four I OI's, designated *OIA*, *OIB*, *OIC*, and *OID* are used in this analysis.

5.1 Maximum Failures

The predicted maximum number of failures over the life of the software ($t = \infty$) is:

$$F(\infty) = \alpha / \beta + X_{s-1} \quad (5)$$

5.2 Remaining Failures

On the assumption that the faults corresponding to failures are removed from the software after the failures occur, the maximum number of remaining failures, predicted at time t is:

$$R(t) = (\alpha / \beta) - X_{s,t} = F(\infty) - X_{s,t} \quad (6)$$

$R(t)$ can also be expressed as a function of test time t_2 as follows:

$$R(t_2) = (\alpha / \beta) (\exp - \beta [t_2 - (s-1)]) \quad (7)$$

5.3 Fraction of Remaining Failures:

$$p = R(t) / F(\infty) \quad (8)$$

5.4 Operational Quality

The *operational quality* of software is defined as:

$$Q = 1 - p \quad (9)$$

We recognize that there are many attributes of software quality such as maintainability and useability. Attributes like these are difficult to quantify and measure. Our Q is easy to quantify and measure: it represents the quality of the software *during execution*, as measured by the absence of failures.

5.5 Test Time to Achieve Specified Remaining Failures

The predicted test time required to achieve a specified number of remaining failures, where $R(t_2)$ is the specified number of remaining failures at t_2 , is:

$$t_2 = [\log [\alpha / (\beta [R(t_2)])]] / \beta + (s-1) \quad (10)$$

5.6 Time to Next Failure

The predicted time for the next F_t failures to occur, when the current time is t , is given by:

$$T_F(t) = [(\log [\alpha / (\alpha - \beta (X_{s,t} + F_t)]) / \beta) - (t - s + 1)]$$

for $(\alpha / \beta) > (X_{s,t} + F_t)$ (11)

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

t : Current interval;

$X_{s,t}$: Cumulative number of failures observed in the range s, t , and

F_t : Given number of failures to occur after interval t .

We consider equations (5)-(9) and (11) to be measures of reliability; equation (10) represents the test time required to achieve stated reliability goals. Although we consider Q , equation (9), a measure of reliability, we call it *Operational Quality*, and not *Operational Reliability*, to avoid possible confusion with the formal definition of *reliability*: the ability of a system or component to perform its required function under stated conditions for a specified period of time [IEE90]. If a quality requirement is stated in terms of *fraction of remaining failures*, the definition of Q as *Operational Quality* is consistent with the IEEE definition of quality: the degree to which a system, component, or process meets specified requirements [IEE90].

6. Criterion for Optimally Selecting Failure Data

Prior to making predictions, the parameters α and β are estimated using the method of maximum likelihood for each value of s in the range $1, t$ where convergence can be obtained [AIA93, SCH93, SCH 75]. 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 failure (MSE_T), or remaining failures (MSE_R), depending on the type of prediction. The first two were reported in [SCH93]. In this paper we develop MSE_R . MSE_R is also the criterion for *total failures* and *test time* because the two are functionally related to *remaining failures* (see equations 6 and 10). We also show MSE_T because it is used in our *time to next failure* predictions. Failure count intervals are equal to 30 days of continuous execution time. Once α , β , and s are estimated from observed counts of failures, the foregoing predictions can be made. The reason MSE is used to evaluate which triple (α, β, 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., $s>1$) [SCH93].

The *Statistical Modeling and Estimation of Reliability Functions for Software* (SMERFS) [FAR93] is used for all predictions except t_2 , which is 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 $R(t)$: (6), which is a function of predicted total failures and the observed failures, and (7), which is a function of test time, and apply the MSE criterion. This yields the following Mean Square Error (MSE_R) criterion for number of remaining failures:

$$MSE_R = \frac{\sum_{i=s}^t [F(i) - X_i]^2}{t - s + 1} \quad (12)$$

where $F(i)$ is the predicted cumulative failures at time i and X_i is the cumulative observed failures at time 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 [SCH93], is given by equation (13):

$$MSE_T = \frac{\sum_{i=s}^{J-1} [\log [\alpha / (\alpha - \beta (X_{s,i} + F_{ij}))] / \beta - (i - s + 1)] - T_{ij}}{(J - s)}^2$$

$$\text{for } (\alpha/\beta) > (X_{s,i} + F_{ij}) \quad (13)$$

The terms in MSE_T have the following definitions:

- i: Current interval;
- j: Next interval $j > i$ where $F_{ij} > 0$;
- $X_{s,i}$: Cumulative number of failures observed in the range s, i ;
- F_{ij} : Number of failures observed during j since i ;
- T_{ij} : Time since i to observe number of failures F_{ij} during j (i.e., $T_{ij} = j - i$);
- t: Upper limit on parameter estimation range; and
- J: Maximum $j \leq t$ where $F_{ij} > 0$.

7. Relating Testing to Reliability and Quality

7.1 Predicting Test Time and Remaining Failures

We use equation (5) to predict *total failures* ($F(\infty) = 11.76$) for *OIA*. Using given values of p , equation (8), we predict $R(t_2)$ for each value of p . The values of $R(t_2)$ are the predictions of *remaining failures* after the OI has been executed for time t_2 . Then we use the values of $R(t_2)$ and equation (10) to predict corresponding values of t_2 . The results are shown in Figure 6, where $R(t_2)$ and t_2 are plotted against p for *OIA*. Note that required test time t_2 rises very rapidly at small values of p and $R(t_2)$. Also note that the maximum value of p on the plot corresponds to $t_2 = 18$ and that smaller values correspond to *future* values of t_2 (i.e., $t_2 > 18$).

7.2 Predicting Operational Quality

Equation (9) is a useful measure of the *operational quality* of software because it measures the degree to which faults have been removed from the software, relative to predicted *total 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 (8) and (9), we compute $R(t_2)$ and Q , respectively. The values of $R(t_2)$ are then used in equation (10) to compute t_2 . The corresponding values of Q and t_2 are plotted in Figure 7 as *Quality* and *Execution 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 8 for *OIA* on the solid curve that has occurred in the 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*. Next, using equation (11), we predict the *time to next failure* $T_r(18)$ to be 4 (3.87 rounded) on the dashed curve. Based on the foregoing, this prediction indicates we should continue testing if $T_r(18) = 3.87 \leq t_m$ (mission duration).

8. Making Safety Decisions

In making the decision about how long to test t_2 , we apply our safety criteria and risk assessment approach. We use Table 1 and Figure 9 to illustrate the process. For $t_2 = 18$ (when the last failure occurred on *OIA*), $R_s = 1$, and $t_m = 8$ days (.267 intervals), we show *remaining failures*, *risk of remaining failures*, *time to next failure*, *risk of time to next failure*, and *operational quality* in Table 1. 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. With these results in hand, the software manager could choose to continue to test. If testing were to continue until $t_2 = 52$, the predictions in Table 1 and annotated on Figure 9 would be obtained. These results show that *criterion 1* is now satisfied (i.e., *SAFE*) and *operational quality* is high. We also see that at this value of t_2 , further increases in t_2 would not result in a significant increase in reliability and safety.

9. Summary of Predictions and Validation

9.1 Predictions

Table 2 shows a summary of remaining and total failure predictions compared with actual failure data, where available, for *OIA*, *OIB*, *OIC*, and *OID*. Because we do not know the *actual* remaining and total failures, we assume the 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 assume that total failures equals the total observed failures.

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

9.2 Validation

A total of 13 predictions were made across Tables 2 and 3, where there was an actual value to compare: three $R(t)$, four $F(\infty)$, four t_2 , and two $T_F(t)$. The mean relative error (mean of (actual-predicted)/actual) of prediction is 23.13% and the standard deviation is 28.86%. In making these predictions we note both the sparsity of post-delivery failures and the extremely long test times for *Shuttle* flight software as tabulated below. As stated previously, we used the conservative approach of using all failures -- independent of severity code -- in satisfying our safety criteria.

Number of Failures by Severity Code:

1. Severe Vehicle or Crew Performance Implications: 0
2. Affects Ability to Complete Mission (Not safety issue): 14
3. Workaround Available, Minimal Effect on Procedures: 29
4. Insignificant (Paperwork, etc.): 2
5. Not Visible to User: 0
6. Unknown severity: 2

Total test time for 47 failures to occur: 192 thirty day intervals.

10. Conclusions

Software reliability models provide one of several tools that software reliability managers of the *Space Shuttle Primary Avionics 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. We have shown how to apply these models; the approach would seem to be applicable to other safety critical systems. We encourage practitioners to apply these methods.

11. References

- [AIA93] 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.
- [BIL94] C. Billings, et al, "Journey to a Mature Software Process", IBM Systems Journal Vol. 33, No. 1, 1994, pp. 46-61.
- [BUT93] Ricky W. Butler and George B. Finelli, "Infeasibility of Quantifying the Reliability of Life-Critical Real-Time Software", IEEE Transactions on Software Engineering, Vol. 19, No. 1, January 1993, pp. 3-12.
- [DAL94] S. R. Dalal and A. 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.
- [FAR93] 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.
- [IEE90] IEEE Standard Glossary of Software Engineering Terminology, IEEE Std 610.12.1990.
- [KEL95] 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.
- [LEV86] Nancy G. Leveson, "Software Safety: What, Why, and How", ACM Computing Surveys, Vol. 18, No. 2, June 1986, pp. 125-163.
- [MUS87] John Musa, et al, Software Reliability: Measurement, Prediction, Application, McGraw-Hill, New York, 1987.
- [SCH93] 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.
- [SCH92] 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.
- [SCH75] 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.
- [VOA95] Jeffrey M. Voas and Keith W. Miller, "Software Testability: The New Verification", IEEE Software, Vol. 12, No. 3, May 1995, pp. 17-28.

Table 1
Safety Criteria Assessment

$R_c=1$ $t_m=8$ days					
t_2	$R(t_2)$	RiskR(t_2)	$T_r(t_2)$	Risk $T_r(t_2)$	Q
18	4.76	3.76	3.87	-13.49	.60
52	0.60	-40	*	*	.95

t_2 : 30 day test intervals. * Can't predict because predicted Remaining Failures is less than one.

Table 2
Predicted Remaining and Total Failures versus Actuals

	t_2	$R(t_2)$	Actual R	$F(\infty)$	Actual F
OIA	18	4.76	7 ^A	11.76	7 ^A
OIB	20	.95	1 ^B	12.95	13 ^B
OIC	20	1.87	2 ^C	12.87	13 ^C
OID	18	7.36	4 ^D	17.36	14 ^D

t_2 : 30 day intervals. $R(t_2)$: Predicted Remaining Failures at Test Execution Time t_2 $F(\infty)$: Predicted Total Failures.

Time of last recorded failure: A. 17.17 (No additional failures have been reported). B. 63.67 C. 43.80 D. 65.03

Table 3
Predicted Test Time and Time to Next Failure versus Actuals

	$t_2(R=1)$	Actual t_2	t	$T_r(t)$	Actual T_r
OIA	43.59	?	18	3.87	?
OIB	*	63.67	20	*	43.67
OIC	24.98	27.07	20	4.16	7.63
OID	56.84	58.27	18	6.35	6.20

t_2 : 30 day intervals. $t_2(R=1)$: Predicted Total Test Time to reach one Remaining Failure.

$T_r(t)$: Predicted Time to Next Failure at Execution Time t. * Can't predict because predicted Remaining Failures is less than one.

Additional Predictions for OID: $t_2(R=2)=43.35$, Actual=45.17; $t_2(R=3)=35.47$, Actual=23.70.

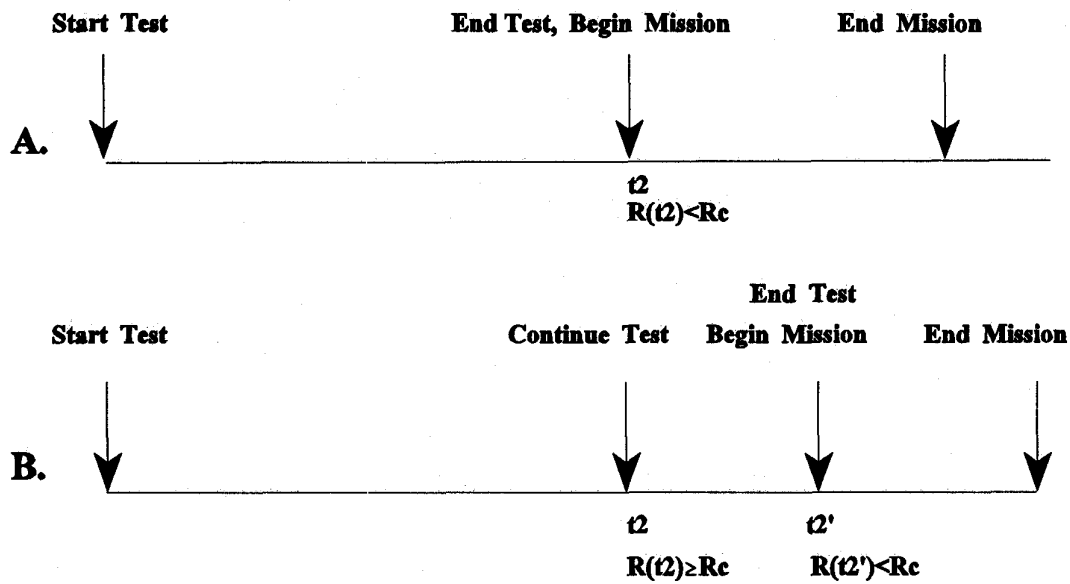


Figure 1. Remaining Failures Criterion Scenario

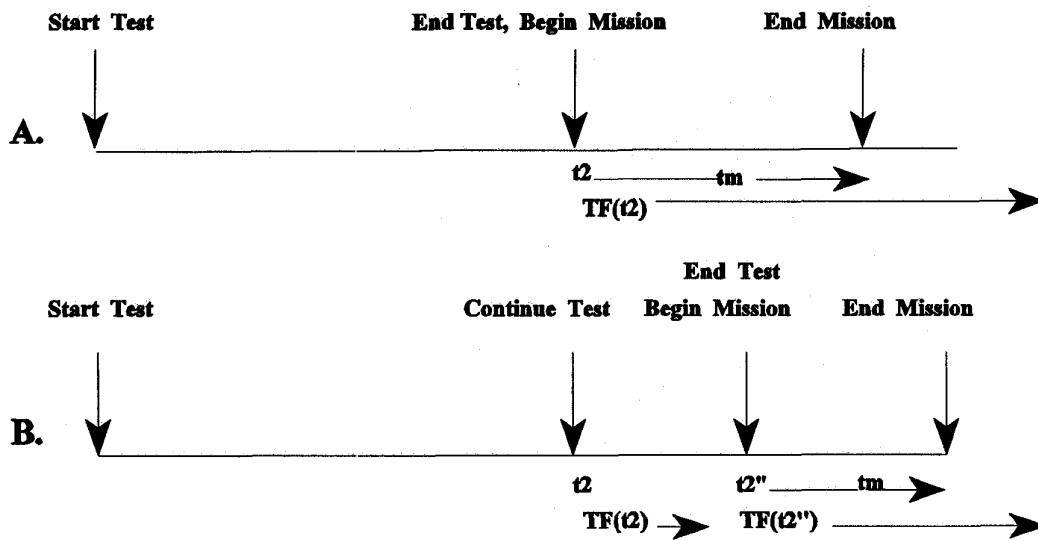


Figure 2. Time to Next Failure Criterion Scenario

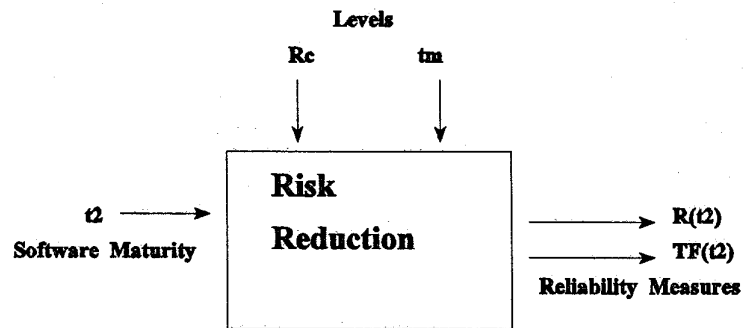


Figure 3. Risk Reduction

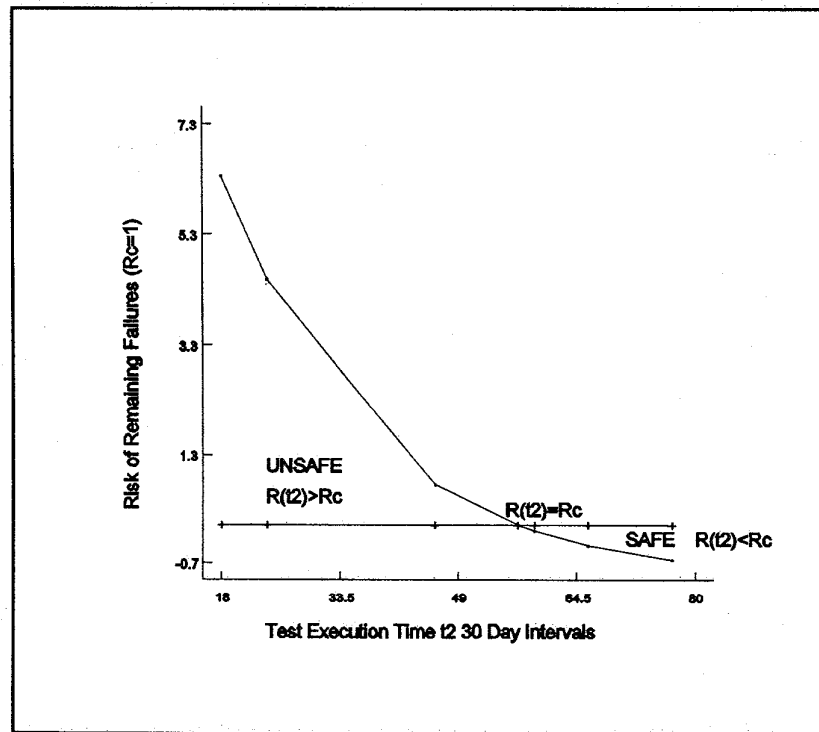


Figure 4. Risk of Remaining Failures

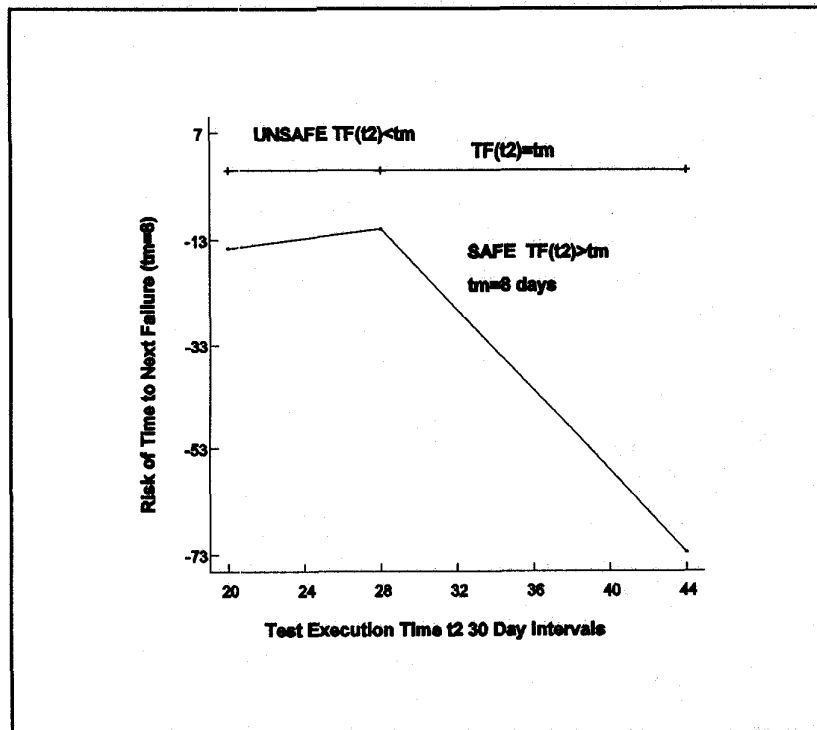


Figure 5. Risk of Time to Next Failure

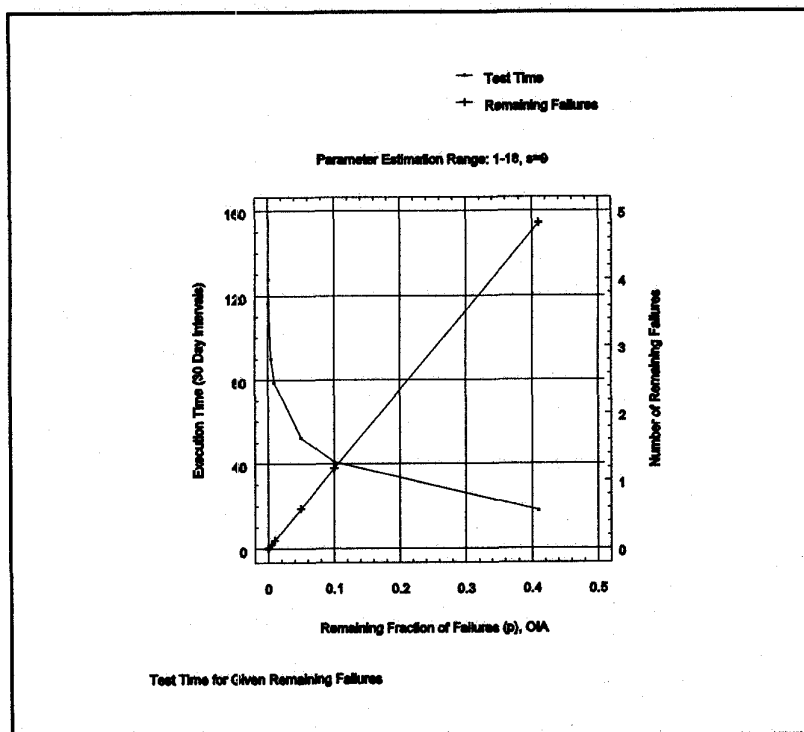


Figure 6. Test Time and Remaining Failures versus Fraction Remaining Failures

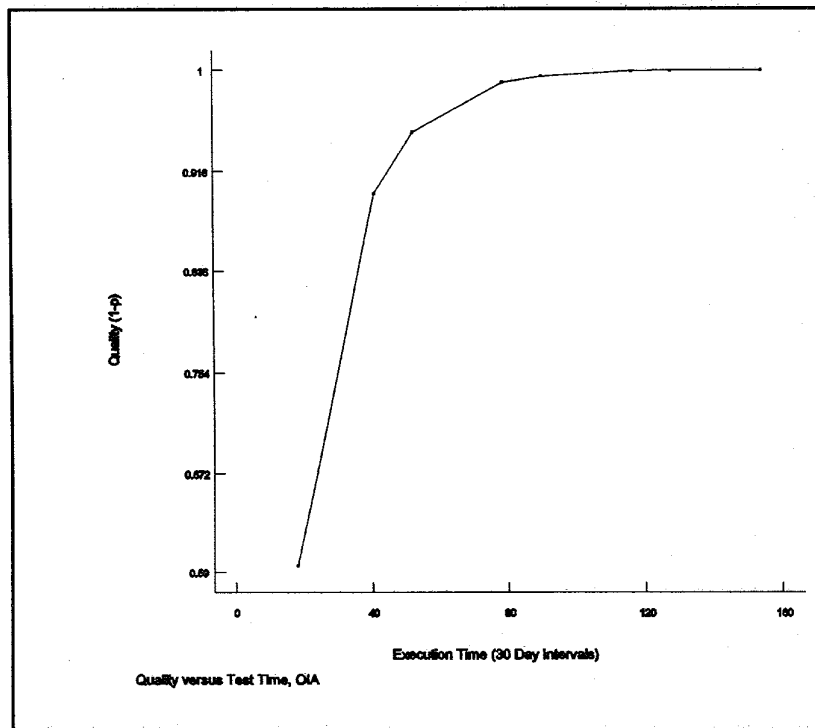


Figure 7. Operational Quality versus Test Time

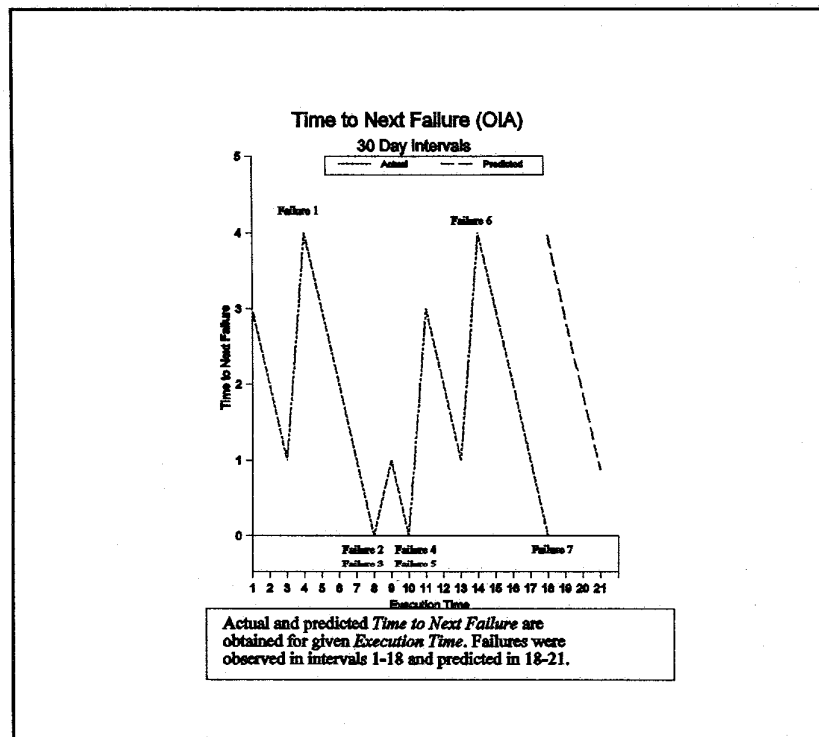


Figure 8. Time to Next Failure versus Execution Time

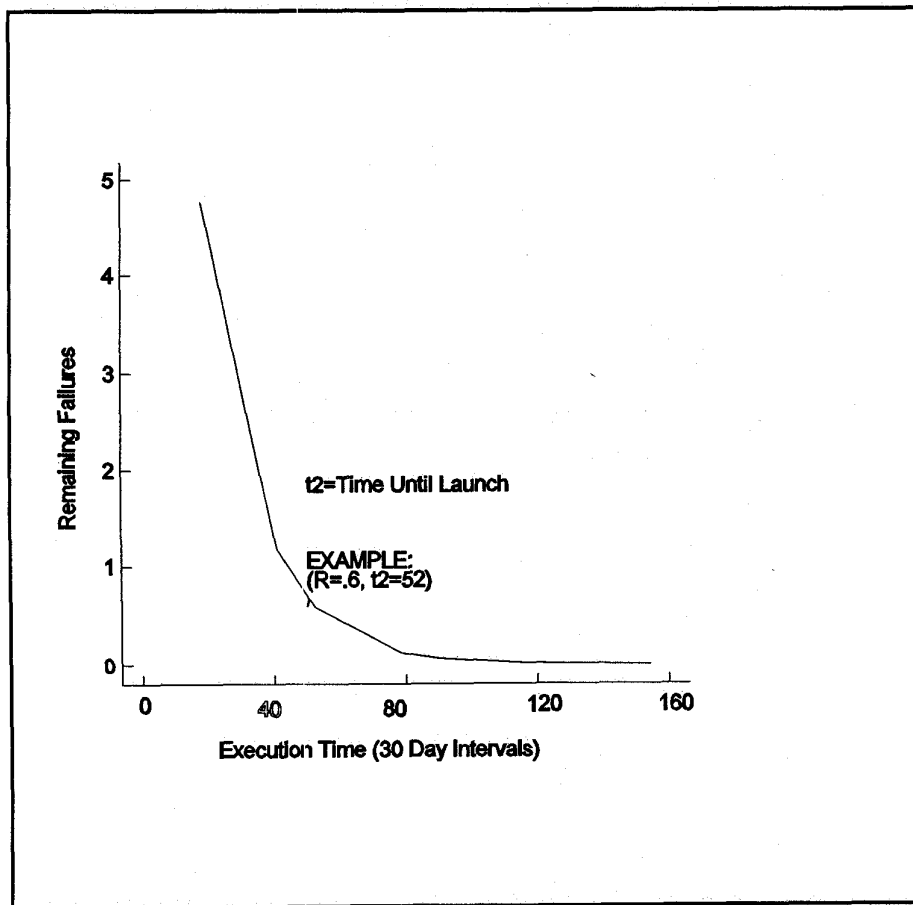


Figure 9. Launch Decision: Remaining Failures versus Test Time, OIA