Abstract


Keywords: Application of software reliability models, reliability prediction, testing strategies, Space Shuttle.

Introduction

The American Institute of Aeronautics and Astronautics, through its Software Reliability Working Group, and the American National Standards Institute have produced a Recommended Practice for Software Reliability, ANSI/AIAA R-013-1992 for software reliability estimation and prediction. By ballot this document was approved by AIAA and ANSI as a recommended practice on February 23, 1993. This document contains recommended procedures for implementing and using software reliability models and a set of recommended models. The effort also involves the establishment of a national software reliability database to be located at the Rome Laboratory, Griffiss Air Force Base, Rome, New York. A "Call for Participation" for this database project appeared in the October 1991 issue of Computer Magazine, p.94. This document will be proposed as an ANSI recommended practice. The working group was chaired by Dave Siefert of ATT-NCR with Ted Keller, IBM-FSD and George Stark of Mitre Corporation, as vice chairs.

This paper presents "Appendix F - Software Reliability Measurement Case Studies: Using Software Reliability Models for Developing Test Strategies" from the Recommended Practice.

Software reliability models provide the software manager with a powerful tool for predicting, controlling and assessing the reliability of software. In combination, these functions allow an organization to determine whether its reliability goals have been met. We show how the recommended practice can be applied to use reliability models in such important processes as predicting reliability, detecting anomalous conditions in software, and developing strategies to bring software into conformance with goals. The Space Shuttle Primary Avionics Software Subsystem is used as an example.

Allocating Test Resources

It is important for software organizations to have a strategy for testing; otherwise, testing costs are likely to get out of control. Without a strategy, each module you maintain may be treated equally with respect to allocation of resources. You need to treat your modules unequally! That is, allocate more test time during testing, effort and funds to the modules which have the highest predicted number of failures, \( F(t_1,t_2) \), during the interval \( [t_1,t_2] \), where \( t_1,t_2 \) could be execution time or labor time (of maintainers) for a single module. In the remainder of this section, "time" means execution time. Use the convention that you make a prediction of failures at \( t_1 \) for a continuous interval with end-points \( t_1 \) and \( t_2 \).

The following sections describe how a reliability model can be used to predict \( F(t_1,t_2) \). The testing strategy is the following:

Allocate test execution time to your modules during testing in proportion to \( F(t_1,t_2) \).

You update model parameters and predictions based on observing the actual number of failures, \( X_{t_1,t_1} \), during \( t_1,t_1 \). This is shown in Figure 1, where you predict \( F(t_1,t_2) \), using the model and \( X_{t_1,t_1} \). In this figure, \( t_0 \) is total available test time for a single module. Note that you could have \( t_2 = t_0 \) (i.e., the prediction is made to the end of the test period).

\[
\begin{array}{cccc}
1 & t_1 & t_2 & t_0 \\
X_{t_1,t_1} & F(t_1,t_2) \\
\end{array}
\]

Figure 1. Reliability prediction time scale

Based on the updated predictions, you may want to reallocate your test resources during testing (i.e., test execution time). Of course, it could be disruptive to your organization to reallocate too frequently. So, you could predict and reallocate at major milestones (e.g., major upgrades). Using the Schneiderwold software reliability model [2], the Space Shuttle Primary Avionics Software Subsystem, and failure data from the AIAA Software Reliability Database [3] as an example, the process of using prediction for allocating test resources is developed. Two parameters, \( \alpha \) and \( \beta \), which will be used in the following equations, are estimated by applying the model to \( X_{t_1,t_1} \). Once
the parameters have been established, you can predict various quantities that will assist you in allocating test resources, as shown in the following equations:

1. Number of failures during t1:

\[ F(t) = \frac{\alpha}{\beta} [1 - \exp(-\beta(t-s+1))] + X_s \]  

where \( 1 \leq s \leq t \) is the starting failure count interval determined by a mean square error criterion and \( X_s \) is the actual cumulative failure count in \( s, t \).

2. Using (1) and Figure 1, you can predict number of failures during \( t_1, t_2 \):

\[ F(t_1, t_2) = \frac{\alpha}{\beta} [1 - \exp(-\beta(t_2-s+1))] - X_{s,t} \]  

where \( X_{s,t} \) is the cumulative failure count in \( s, t \).

3. Also, you can predict maximum number of failures during the life \( t = \infty \) of the software:

\[ F(\infty) = \frac{\alpha}{\beta} + X_1 \]  

4. Using (3), you can predict the maximum remaining number of failures at \( t \):

\[ R(t) = \frac{\alpha}{\beta} - X_s \]  

where \( X_s \) is the cumulative failure count in \( s, t \).

Given \( n \) modules, allocate test execution periods \( T_i \) for each module \( i \) according to the following equation:

\[ T_i = \frac{F_i(t_1, t_2) * (n)(t_2-t_1)}{\sum_{i=1}^{n} F_i(t_1, t_2)} \]  

In (5), note that although predictions are made using (2) for a single module, the total available test execution time \( (\alpha)(t_2-t_1) \) is allocated for each module across \( n \) modules. You use the same interval \( 1, 20 \) for each module to estimate \( \alpha \) and \( \beta \) and the same interval \( 20, 30 \) for each module to make predictions, but from then on a variable amount of test time \( T_i \) is used depending on the predictions.

Tables 1 and 2 summarize the results of applying the model to the failure data for three Space Shuttle modules (operational increments). The modules are executed continuously, 24 hours per day, day after day. For illustrative purposes, each period in the test interval is assumed to be equal to 30 days. After executing the modules during \( 1, 20 \), the SMERFS [1] program was applied to the observed failure data during \( 1, 20 \) to obtain estimates of \( \alpha \) and \( \beta \). The total number of failures observed during \( 1, 20 \) and the estimated parameters are shown in Table 1.

Equations (2), (3), (4) and (5) were used to obtain the predictions in Table 2 during \( 20, 30 \). The prediction of \( F(20,30) \) led to the prediction of \( T_i \), the allocated number of test execution time periods. The number of additional failures that were subsequently observed, as testing continued during \( 20, 20 + T_i \), is shown as \( X(20, 20 + T_i) \). Since there may be remaining failures, \( R(T) \) is predicted from (4) and shown in Table 2. The predicted remaining failures indicate that additional testing is warranted. Note that the actual total number of failures \( F(\infty) \) would only be known after all (i.e., extremely long test time) testing is complete and was not known at \( 20 + T \). Thus you need additional procedures for deciding how long to test to reach a given number of remaining failures. A variant of this decision is the stopping rule (when to stop testing?). This is discussed in the following section.

**Making Test Decisions During Testing**

In addition to allocating test resources, you can use reliability prediction to estimate the minimum total test execution time \( t_2 \) (i.e., interval \( 1, t_2 \)) necessary to reduce the predicted maximum number of remaining failures to \( R(t_2) \). To do this, subtract equation (1) from (3), set the result equal to \( R(t_2) \), and solve for \( t_2 \):

\[ t_2 = \frac{1}{\beta} \log \left( \frac{\alpha}{\beta} \right) \]  

where, by using (3), \( R(t_2) \) can be expressed as:

\[ R(t_2) = (\alpha + X_s)(p) \]  

where \( p \) is the desired fraction (percentage) of remaining failures at \( t_2 \).

Equation (6) is plotted for Module 1, Module 2, and Module 3 in Figure 2 for various values of \( p \). You can use (6) as a rule to determine when to stop testing a given module during testing.

Using (6) and Figure 2 you can produce Table 3 which tells you the following: the total minimum test execution time \( t_2 \) from time 0 to reach essentially 0 remaining failures (i.e., at \( p = .001 \) or 0.1\%), predicted remaining failures are .01295, .01251, and .01165 for Module 1, Module 2 and Module 3, respectively (see (6) and Table 2); the additional test execution time beyond \( 20 + T \) shown in Table 2; and the actual amount of test time required, starting at 0, for the "last" failure to occur (this quantity comes from the data and not from prediction). You don't know that it is necessarily the last; you only know that it was the last after 64 periods (1910 days), 44 periods (1314 days), and 66 periods (1951 days) for Module 1, Module 2 and Module 3, respectively. So, \( t_2 = 52.9, 54.0 \) and 63.0 periods would constitute your stopping rule for Module 1, Module 2 and Module 3, respectively. This procedure allows you to exercise control over software quality.

**Summary**

We have shown how to use a software reliability model for failure prediction, allocation of test resources during testing based on failure prediction, and a criterion for terminating testing based on prediction of remaining failures. These elements comprise a strategy for assigning priorities to modules for testing.
### Table 1

**Observed Failures and Model Parameters**

<table>
<thead>
<tr>
<th>Module</th>
<th>$X(1,20)$ failures</th>
<th>$\alpha$</th>
<th>$\beta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module 1</td>
<td>12</td>
<td>1.6915</td>
<td>.1306</td>
</tr>
<tr>
<td>Module 2</td>
<td>11</td>
<td>1.7642</td>
<td>.1411</td>
</tr>
<tr>
<td>Module 3</td>
<td>10</td>
<td>1.3403</td>
<td>.1151</td>
</tr>
</tbody>
</table>

### Table 2

**Allocation of Test Resources During Testing**

<table>
<thead>
<tr>
<th>Module</th>
<th>F($\infty$)</th>
<th>F(20,30)</th>
<th>R(T)</th>
<th>T</th>
<th>X(20,20+T)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>failures</td>
<td>failures</td>
<td>failures</td>
<td>periods</td>
<td>failures</td>
</tr>
<tr>
<td>Module 1</td>
<td>12.95</td>
<td>.693</td>
<td>.950</td>
<td>7.0</td>
<td>12.51</td>
</tr>
<tr>
<td>Predicted</td>
<td>Actual</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Module 2</td>
<td>12.51</td>
<td>1.140</td>
<td>.507</td>
<td>11.6</td>
<td>11.4</td>
</tr>
<tr>
<td>Predicted</td>
<td>Actual</td>
<td>13</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Module 3</td>
<td>11.65</td>
<td>1.125</td>
<td>.646</td>
<td>11.4</td>
<td>11.4</td>
</tr>
<tr>
<td>Predicted</td>
<td>Actual</td>
<td>14</td>
<td>1</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

### Table 3

**Test Time $t_2$ Required to Reach "0" Remaining Failures**

$p = .001$

<table>
<thead>
<tr>
<th>Module</th>
<th>$t_2$</th>
<th>Additional Test Time</th>
<th>Last Failure Found</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>periods</td>
<td>periods</td>
<td>periods</td>
</tr>
<tr>
<td>Module 1</td>
<td>52.9</td>
<td>45.9</td>
<td>64</td>
</tr>
<tr>
<td>Module 2</td>
<td>54.0</td>
<td>42.4</td>
<td>44</td>
</tr>
<tr>
<td>Module 3</td>
<td>63.0</td>
<td>51.6</td>
<td>66</td>
</tr>
</tbody>
</table>
References


Execution Time to Reach Remaining Failure Fraction $p$

![Graph showing execution time to reach remaining failure fraction $p$.](image.png)

Figure 2. $p$: Remaining Failure Fraction