Applying Reliability Models to the Space Shuttle

Schneidewind, Norman F.
Keller, Ted W.

http://hdl.handle.net/10945/45143

Downloaded from NPS Archive: Calhoun
Applying Reliability Models to the Space Shuttle

NORMAN F. SCHNEIDEWIND, Naval Postgraduate School
TED W. KELLER, IBM Federal Services Company

Real project experience shows that reliability models can predict reliability and help develop test strategies. This case study reports on IBM's approach to the space shuttle's on-board software.

Reliability models are a powerful tool for predicting, controlling, and assessing software reliability. In combination, these functions let an organization determine if the reliability goals it sets for its software have been met.

At a recent conference, many practitioners reported on the increasing and increasingly successful use of reliability measurement. One of the most important examples was its use on the US space shuttle. The space-shuttle case study is an excellent example of how a real project team can evaluate candidate reliability models and select the models that best match the software's failure history.

In this article, we share the experience of a team at IBM Federal Services Company in Houston, which evaluated many reliability models, tried to validate them for use on this project, and selected the Schneidewind model to predict the reliability of the shuttle's on-board system software for the National Aeronautics and Space Administration.

The approach reported here is experimental — it is not current practice throughout IBM. The output of the Schneidewind model is used by this shuttle project only, and only to add confidence to low failure probabilities that are based on formal certification processes. However, the IBM team has judged that this application successfully models the software's failure history.

The techniques the IBM team used to apply a reliability model and the underlying concepts should be of value to others who must perform similar studies and model evaluations.

USING MODELS TO PREDICT

Three separate but related functions comprise an integrated reliability program:
Predictions is the activity of estimating the future failure rate, number of failures, time to next failure, and mean time to failure, where a failure is "the inability of a system or system component to perform a required function within specified limits." 

Control is the activity of comparing predictions with predefined goals and flagging software that fails to meet those goals.

Assessment is the activity of determining what action to take when software fails to meet goals. For example, you might intensify inspection or testing, redesign the software, or revise the process. Formulating test strategies, which is also part of assessment, involves determining the priority, duration, and completion date of testing and allocating personnel and computer resources to testing.

Satisfying assumptions. To ensure that statistical modeling successfully predicts reliability, you must thoroughly understand precisely how the predictions are to be interpreted and applied and by whom. Business and military decisions could vary significantly in response to perceptions of reliability, based on interpretations of the predictions and their credibility.

To validate a model's appropriateness for an application, you must address each assumption the model makes. For example, the Schneidewind model assumes that a system is modified only in immediate response to an observed failure. It assumes that the process used to correct the code is constant, implying that for each error corrected there is an inherent, fixed probability of introducing additional errors. It also assumes that all code in a program is homogeneous from the standpoint of execution history.

For many systems that are sequentially upgraded, these assumptions appear at first to represent significant incompatibilities. However, as this case study illustrates, these restrictions can be accommodated by carefully analyzing the elements of a complete software system and its associated processes.

systems as components. To apply your data to a reliability model, consider breaking your systems and processes into smaller elements that can more accurately satisfy assumptions and constraints. If you think of each software version as a combination of code subsets that have a known failure history and homogeneous execution history, you can more readily accommodate a model's assumptions.

The IBM team used this approach to deal with the Schneidewind model's assumptions. The shuttle's Primary Avionics Software Subsystem is modified frequently, using a constantly improving process, to add or change capabilities. More than 15 versions of PASS have been released to NASA since 1980, each an upgrade of the preceding version.

Figure 1 shows one way to depict a sequentially upgraded system. Code developed for a system's first release is the original version, labeled version A in Figure 1. Because all the code in version A was released for the first time, the subset of the total system that was new was in fact the entire version. This subset is labeled new-code subset a in Figure 1; all of subset a begins operation when version A is released.

When the system is updated and rereleased as version B, only the lines of code that were modified or added are included in new-code subset b. The remainder of subset a is carried over from version A and has been in use in its unchanged form since version A's release. The new-code subset b, on the other hand, begins operation when version B is released.

All of version B is carried over to version C, unless it is modified, in which case...
it becomes part of new-code subset c. Subset c begins operation when version C is released. This process applies to each successive system release.

**New-code subsets.** As Figure 1 illustrates, each new version of PASS contains code that has been carried forward from previous versions plus code generated for that version. The team found that they can more adequately satisfy a model's assumptions by applying the model independently to the new-code subset of each version. New code developed for a particular version satisfies the Schneidewind model's assumptions:

- It is developed with a nearly constant process.
- It does, as an aggregate, build up the same shelf life and execution history.
- Unless it is subsequently changed to add capability (thereby becoming new code in the next version), it is changed only to correct faults.
- An absolutely essential requirement for this approach is an accurate code-change history, so every failure can be attributed only to the version in which it was first introduced. This lets you build a separate failure history for each release's new code.

**Estimating execution time.** The team estimates the execution time of PASS segments by analyzing test-case records of digital flight simulations as well as records of actual shuttle operations. They count test-case executions as operational executions only if the simulation fidelity matches actual operational conditions very closely. They never count prerelease test execution time for the new code actually being tested as operational execution time.

You can eliminate the tedious manual activity of estimating execution time and increase the accuracy of your estimates if your environment can be designed to automatically log execution time. In many installations, however, this luxury may be impractical, as in the case of the shuttle.

**MODEL APPLICATION**

The team uses each new-code subset's failure and operational execution time histories to generate a reliability prediction for the new code in each version. This approach places every line of code in PASS into one new-code subset or another, depending on the version for which it was written. PASS is represented as a composite system of separate new-code components, each with an execution history and a reliability, connected in series.

By comparing calculated and actual failure histories, the IBM team evaluated several ways to represent a composite system mathematically.

In this case, the team used the failure data for each of six dates between 1986 and 1989 to obtain six PASS reliability predictions using the Schneidewind model. For each of the six predictions, they computed the predicted mean time between failure by assuming that the next failure did in fact occur on the predicted date. They then compared each prediction to the actual mean time between failures as of that date.

The Schneidewind model appears to provide the most accurate fit to the 12 years of failure data from this project. For all six dates, the Schneidewind model's reliability predictions were about 15 percent less than the actual average time between failures. On the basis of the accuracy and consistency of these predictions relative to other models, the IBM team selected this statistical method to model PASS reliability.

**Validation.** The team uses each new-code subset's failure and operational execution time histories to generate a reliability prediction for the new code in each version. This approach places every line of code in PASS into one new-code subset or another, depending on the version for which it was written. PASS is represented as a composite system of separate new-code components, each with an execution history and a reliability, connected in series.

By comparing calculated and actual failure histories, the IBM team evaluated several ways to represent a composite system mathematically.

In the end, they judged a standard statistical expression to best fit the actual failure data. This expression describes the probability of an event based on a serial relationship of multiple elements that each have different probabilities.

In other words, it represents the failure prediction for the overall system as the reciprocal of the sum of the reciprocals of the failure predictions of each individual element. The composite \( T_{PASS} \) measure — the estimated average execution time until the next failure — is computed by

\[ T_{PASS} = \frac{1}{T_{T_a}} + \frac{1}{T_{T_b}} + \frac{1}{T_{T_c}} + \ldots \]

where \( T_{T_a}, T_{T_b}, T_{T_c} \) and so on are estimates of the time until the next failure for each new-code subset, which is determined by applying the model to each subset individually.

Because you must assign code to a new-code subset only when it fails, you need not perform the unreasonable task of breaking down the entire system, line by line, into subsets. You must know which subset a line belongs to only when it is defective. This is what makes this approach so feasible.
you can demonstrate that you can, with high confidence, predict a lower bound on reliability within a specified environment.

If you can use historical failure data at a series of previous dates (and you have the actual data for the failure history following those dates) you should be able to compare the predictions to the actual reliability and evaluate the model’s performance.

You should take all these factors into consideration when establishing validation criteria. This will also significantly enhance the credibility of your predictions among those who must make decisions on the basis of your results.

**Analysis.** After much study and analysis, the IBM team concluded that the Schneidewind model’s conservative performance was due to:

- Conservative execution-time estimates for each version.
  - We plan to improve the accuracy of execution-time estimates and use the model’s weighting capabilities to more heavily weight the most recent failure history in the predictions.

- Slight process improvements implemented during the development cycle of some versions (violating the model’s assumption of a constant process).

- Detection and removal of latent faults before they became failures in execution (violating the implied assumption that the software is corrected only when a failure is encountered). The model’s prediction, which is based on an assumed fault density remaining in the software until the next failure occurs, will underpredict the time to next failure if the fault density decreases between failures.

The IBM team is applying the model to predict a conservative lower bound for PASS reliability. They are also performing independent statistical analyses using the same failure data to compute 95 percent upper and lower confidence intervals. The analyses, which use various classical statistical formulas, have further confirmed both the reasonableness and the relative conservatism of the model’s results. You can find formulas appropriate for your application in any standard statistics textbook.

**RELIABILITY AND TESTING**

If you don’t have a testing strategy, test costs are likely to get out of control. Without a strategy, each module you test may get an equal amount of resources. You must treat modules unequally! Allocate more test time, effort, and funds to modules with the highest predicted number of failures.

You can use a reliability model to predict failures, $F(t_1,t_2)$, during the interval $t_1,t_2$, where $t_1$ could be execution time or tester labor time for a single module (in this case, $t$ means execution time). We predict failures at $t_1$ for a continuous interval that has endpoints at $t_1$ and $t_2$.

We recommend that you allocate test time to your modules in proportion to $F(t_1,t_2)$.

You update the model’s parameters and predictions, according to the actual number of failures, $X_{0,t_1}$, during the time interval $0,t_1$. As Figure 2 shows, we predict $F(t_1,t_2)$, at $t_1$ during $t_1,t_2$, on the basis of the model and $X_{0,t_1}$. In Figure 2, $t_\alpha$ is the total available test execution time for a single module, $F(t_1,t_2)$, the total available test-execution time (or $t$), $t_1$, is allocated for each module $i$ across $n$ modules.

**Example of use.** We now provide an example in which we used the interval 0.20 to estimate $\alpha$ and $\beta$ for each module and the equations would differ, you could use other reliability models to help allocate test resources.

First you establish two parameters, $\alpha$ and $\beta$, which you estimate by applying the model to $X_{0,t_1}$. Once the parameters have been established, you can predict four quantities that help allocate test resources.

1. Number of failures during $0,t_r$:
   
   $F(t) = (\alpha/\beta)(1 - \exp(-\beta t))$  
   
   (1)

2. Using this quantity, you can predict the number of failures during $t_1,t_2$:
   
   $F(t_1,t_2) = (\alpha/\beta)(1 - \exp(-\beta t_2)) - X_{0,t_1}$
   
   (2)

3. You can also predict the maximum number of failures during the software’s life ($t = \infty$):
   
   $F(\infty) = \alpha/\beta$
   
   (3)

4. Then, using this quantity, you can predict the maximum remaining number of failures at $t$:
   
   $R(t) = (\alpha/\beta) - X_{0,t} = \alpha - \alpha/\beta = \alpha(1 - 1/\beta)$
   
   (4)

So, given $n$ modules, you should allocate test-execution time, $T_i$, for each module $i$ according to:

$T_i = F(t_1,t_2) \times (n)(t_2-t_1)$

(5)

In this equation, although you are using the predicted failures for a single module, $F(t_1,t_2)$, the total available test-execution time (or $t$), $t_2$, is allocated for each module $i$ across $n$ modules.

Figure 2. Reliability prediction time scale. We predict $F(t_1,t_2)$, at $t_2$ during the time interval $t_1,t_2$, based on the model and $X_{0,t_1}$. On the time scale, $t_\alpha$, is total available test time for a single module.
Failures during the interval 20,20+T, is shown in the last column as X(20,20+T).

If you compare Table 1 with Table 2, you see that additional failures could occur in module 1 (12.95 - (12 + 0) = .95 failures) and module 2 (12.50 - (11 + 1) = .50 failures), according to the predicted maximum number of failures F(∞). That is, for these modules,

\[ X(20,20+T) < F(∞). \]

The actual F(∞) is known only after all testing is complete; it is not known at 20+T. 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. John Musa and A. Frank Ackerman use failure intensity as a criterion for determining when to stop testing.

When to stop testing? Our recommended approach to deciding when to stop testing uses reliability prediction to estimate the minimum testing time \( t_2 \) (or the interval \( 0, t_2 \)) needed to reduce the predicted maximum number of remaining failures to \( R(t_2) \).

To do this, we subtract Equation 1 from Equation 3, set the result equal to \( R(t_2) \), and solve for \( t_2 \):

\[
t_2 = \frac{\ln \left( \frac{a}{\beta} \right)}{b} \quad (6)
\]

where \( R(t_2) \) can be established from

\[
R(t_2) = (p)(a/\beta) \quad (7)
\]

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

Substituting Equation 7 in Equation 6 gives

\[
t_2 = \frac{\ln \left( \frac{1}{(1-p)} \right)}{b} \quad (8)
\]

You can use this result to determine when to stop testing a given module. Figure 3 plots the results for module 1 and module 2 for various values of \( p \). From Equation 8 and Figure 3 you can derive (for \( p = .001 \)), the data in Table 3:

- The total minimum test time \( t_2 \) from time 0 to reach essentially 0 remaining failures: .1 percent, corresponding to the predicted remaining failures: 12.95 \times .001 = .01295 for module 1 and 12.50 \times .001 = .01250 for module 2 (from Equation 7 and Table 2).
- The additional test time beyond 20+T: 52.9 - 7.6 = 45.3 for module 1 and 49.0 - 14.4 = 34.6 for module 2.

<table>
<thead>
<tr>
<th>TABLE 1</th>
<th>OBSERVED FAILURES AND PARAMETERS FOR 0,20 INTERVAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Failures</td>
<td>( \alpha )</td>
</tr>
<tr>
<td>Module 1</td>
<td>12</td>
</tr>
<tr>
<td>Module 2</td>
<td>11</td>
</tr>
<tr>
<td>Module 3</td>
<td>10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TABLE 2</th>
<th>TEST-RESOURCE ALLOCATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>( F(∞) ) failures</td>
<td>( F(20,30) ) failures</td>
</tr>
<tr>
<td>Module 1 predicted actual</td>
<td>12.95</td>
</tr>
<tr>
<td>Module 2 predicted actual</td>
<td>12.50</td>
</tr>
<tr>
<td>Module 3 predicted actual</td>
<td>10.81</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TABLE 3</th>
<th>TIME NEEDED TO REACH &quot;ZERO&quot; REMAINING FAILURES (( p = .001 ))</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total test time (in periods)</td>
<td>Additional test time (in periods)</td>
</tr>
<tr>
<td>Module 1</td>
<td>52.9</td>
</tr>
<tr>
<td>Module 2</td>
<td>49.0</td>
</tr>
</tbody>
</table>

The actual \( F(∞) \) is known only after all testing is complete; it is not known at 20+T. 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. John Musa and A. Frank Ackerman use failure intensity as a criterion for determining when to stop testing.

When to stop testing? Our recommended approach to deciding when to stop testing uses reliability prediction to estimate the minimum testing time \( t_2 \) (or the interval \( 0, t_2 \)) needed to reduce the predicted maximum number of remaining failures to \( R(t_2) \).

To do this, we subtract Equation 1 from Equation 3, set the result equal to \( R(t_2) \), and solve for \( t_2 \):

\[
t_2 = \frac{\ln \left( \frac{a}{\beta} \right)}{b} \quad (6)
\]

where \( R(t_2) \) can be established from

\[
R(t_2) = (p)(a/\beta) \quad (7)
\]

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

Substituting Equation 7 in Equation 6 gives

\[
t_2 = \frac{\ln \left( \frac{1}{(1-p)} \right)}{b} \quad (8)
\]

You can use this result to determine when to stop testing a given module. Figure 3 plots the results for module 1 and module 2 for various values of \( p \). From Equation 8 and Figure 3 you can derive (for \( p = .001 \)), the data in Table 3:

- The total minimum test time \( t_2 \) from time 0 to reach essentially 0 remaining failures: .1 percent, corresponding to the predicted remaining failures: 12.95 \times .001 = .01295 for module 1 and 12.50 \times .001 = .01250 for module 2 (from Equation 7 and Table 2).
- The additional test time beyond 20+T: 52.9 - 7.6 = 45.3 for module 1 and 49.0 - 14.4 = 34.6 for module 2.
ACKNOWLEDGMENTS

We thank E. Gomez, D.O. Hamilton, and J.K. Orr of the IBM Federal Services Co., Houston, Texas, for their roles in the space-shuttle reliability study. This research was supported by William Farr of the Naval Surface Warfare Center and Raymond Paul of the Army Operational Test and Evaluation Command.

REFERENCES


Norman F. Schneidewind is a professor of information sciences at the Naval Postgraduate School and director of the computer laboratories in his department. His interests are software engineering and networks. He is chairman of the IEEE Standard for Software Quality Metrics working group and vice chairman for standards for the IEEE Computer Society's Technical Committee on Software Engineering and edits the Standards department for Computer Software.

Schneidewind received a BS in electrical engineering from the University of California at Berkeley, an MS in electrical engineering and an MS in computer science, both from San Jose State University, and an MS in engineering and a PhD in operations research from the University of Southern California. He is a member of Eta Kappa Nu and Tau Beta Pi engineering honor societies, Sigma Xi research society, and a senior member of IEEE.

Ted W. Keller is a senior technical staff member at IBM Federal Services Co., where he manages the organization responsible for integrating schedules and commitments, configuration control, control boards, quality and reliability metrics, and certification of flight readiness for the space shuttle's Primary Avionics Software System.

Keller received a BS in aerospace engineering from the University of Tennessee. He is co-vice chair of the AIAA Space-Based Observations Systems Committee on Standard's Software Reliability Working Group and a member of the IEEE Computer Society.

Address questions about this article to Schneidewind at the Naval Postgraduate School, Code AS/SS, Monterey, CA 93943; internet n.schneidewind@compmail.com.

- The actual amount of test time required, starting at 0, for the "last" failure to occur: 64 for module 1 and 44 for module 2. This quantity comes from data, not from prediction. You don't know that it is necessarily the last failure. You only know that it was the last after 64 periods (1,910 days) for module 1 and 44 periods (1,314 days) for module 2.

So, your stopping rule for module 1 is \( t_1 = 52.9 \) periods; for module 2, \( t_2 = 49.0 \) periods.

Applying these methods and models is feasible today, but we recommend that you combine your use of a model with an evaluation of its assumptions and constraints, a validation of its predictions, and an understanding of how to interpret its predictions. Doing these things will lend credibility to your results.

---

**Figure 3.** Execution time needed to reach the desired fraction of remaining failures.