JOURNAL OF AEROSPACE COMPUTING, INFORMATION, AND COMMUNICATION
Vol. 5, September 2008

# Software Sequential Testing Model

Norman F. Schneidewind*
*Naval Postgraduate School, Monterey, CA, 93942*

DOI: 10.2514/1.34543

**A risk-driven reliability model and testing process is developed that borrows concepts from classical sequential testing methodology which is used for hardware. The model is adapted to software. Both consumer and producer risk are considered, reflecting the fact that the consumer (e.g., customer) and producer (e.g., contractor) have different perspectives concerning what they consider to be tolerable risks of software failure. Similarly, there is also a differentiation based on what the consumer and producer consider to be acceptable reliability. Test rules are specified for determining at each decision point in testing whether the software and the model prediction accuracy are acceptable. In addition, the test rules serve as stopping criteria for testing. Both empirical and predicted quantities are assessed. Based on experience in using the model, lessons learned are provided with the objective of improving the model and process for future applications. This model and test scenario is applied to a real application involving the NASA Space Shuttle flight software. The model and test scenario can be tailored to commercial applications, as well.**

## I. Model and Process Basics

SOFTWARE test scenarios involve the comparison of the software's actual outputs, resulting from test scenario execution, with its expected outputs, as documented by Whittaker [1]. The model actual outputs presented here are empirical values of risk and reliability and the expected outputs are represented by specified threshold values of risk and reliability.

In addition, risk and reliability predictions provide stopping rules for testing. The foundation for these concepts of software testing is based on classical methods addressed to hardware [2], but with significant modifications to tailor the models to software testing and reliability. The classical methods of sequential testing, involving the concepts of consumer and producer risks [2], are very useful for structuring a testing and reliability model. These concepts are however lacking in the literature on software testing [3]. Software testing emphasizes techniques such as statement coverage, decision coverage, branch coverage, and data flow coverage [3]. The classical methods are not entirely satisfactory for software because they are based on testing large quantities of homogeneous hardware items. This is not the case with software where, in many cases, one-of-a kind of software system is developed and tested. The classical methods therefore require modification to be applicable to software. Another important facet of the risk and reliability process is to evaluate not only the software but also the model that predicts software risk and reliability as well. If the model cannot predict accurately, the predictions cannot be used and we must try to validate another model.

## II. Safety Critical Software Considerations

Since the example application is about the Shuttle software, it is important to consider the risk and reliability requirement of this type of software. To assist in making informed acceptance decisions, software risk analysis and reliability prediction are integrated to provide a comprehensive approach to implementing test rules designed to

Received 11 September 2007; accepted for publication 20 March 2008. 
* Professor Emeritus of Information Sciences, ieeelife@yahoo.com, 2822 Raccoon Trail, Pebble Beach, CA 93953, USA.

reducing risk and increasing reliability. This approach is applicable to all software, and in particular, it is critical for certifying safety critical software because achieving improvements in the reliability of software contributes to system safety [4]. In addition, for this type of software, it is critical to have a feedback mechanism during testing to indicate when to continue to test and when to stop testing. Important feedback criteria are level of risk, reliability, and reliability growth. This approach was inspired by the feedback mechanism concept of Cangussu et al. [5] in which a test manager was used to monitor the difference between observed reliability and reliability predicted by a model. The difference is fed back into the test process to control the next step in testing. In my case, the differences between observed and required risk and reliability are used to control the test process.

## III. Other Reliability Testing Methods

Reliability testing can be conducted at a macro or microlevel. This model uses the former in which the concern is about the big picture of risk, failure occurrence, and reliability, and how to mitigate risk and increase reliability in sequential test scenarios. In the microview of testing, however, the focus is on methods that deal with the specifications, code, and data flow to produce effective fault removal in a cost-efficient manner. Specification-based testing produces test cases based on inputs, outputs, and program states. Code-based testing addresses computation results, predicate coverage, and control flow coverage. In data flow-based testing, test cases are produced to cover the execution space between where variables are declared and where they are used. Yet another method is mutation testing in which mutants of the original code are produced by introducing faults into program statements and observing the resulting execution behavior [6].

Lyu provides a brief description of some of the important white box testing methods: white box testing uses the structure of the software to measure the quality of testing. Other testing schemes include statement coverage, decision coverage, and data-flow coverage. Statement coverage testing constructs test cases such that each statement or a basic block of code is executed at least once. Decision coverage constructs test cases such that each decision in the program is covered at least once. A decision is covered if, during some execution, it evaluates to true and in the same or another execution it evaluates to false [7].

It appears that none of these methods is superior to the others in all cases and that their effectiveness and efficiency are application dependent. Selected tests at the microlevel should be combined with a macrolevel approach, to provide a comprehensive attack on the software risk and reliability problem. The approach used here is to model testing at the micro-level (i.e., white box testing) to provide failure count input to the macrolevel model (i.e., black box testing). The process does not have to stop there. The approaches can be used synergistically by feeding black box testing risk and reliability predictions to white box testing so that the latter will have an assessment of likely operational risk and reliability. Then, the white box strategy would be adjusted to focus testing on the highest risk and lowest reliability software.

## IV. Software and Model Performance

In the analysis and evaluation of test results the engineer must be careful to distinguish between software performance and model performance. Therefore, before making predictions with the model, one way in which the engineer could assess whether its prediction accuracy is unacceptable or not is to see whether the accuracy goals have been met after two tests. If this is not the case, try to validate another model. If the predictions are acceptable, proceed with the risk and reliability tests.

With regard to model performance, it is assumed that the model will perform in future operational time as it has during test time. Of course, this may not be the case, but it is the best we can do until the future is reached when we can compare actual risk and reliability with the predicted quantities. Continuing this bootstrapping process will continually refine prediction accuracy as more failure data are collected. In addition, confidence in the model can be built by conducting multiple tests to train the model to improve its prediction accuracy.

## V. Test Rules

One of the most difficult aspects of testing is to answer the question: “when to stop testing?” Myers suggests that testing should be stopped when we have discovered and corrected a given number of faults [8]. While this approach is certainly better than stopping when money and time run out, and it is indirectly related to reliability, criteria directly

related to risk and reliability are suggested. With this approach, the stopping rule to achieving acceptable levels of risk and reliability can be keyed. This concept is embodied in the test rules.

Test rules should also include the criticality of the software being tested. This factor is mentioned by Schmid et al. [9], where the authors state: "Many commercial products are not fully prepared for use in high assurance situations. In spite of the criticality of these applications, there currently exists a dearth of software assurance techniques to asses the robustness of both the application and the operating system under strenuous conditions. The testing practices that ordinary commercial products undergo are not thorough enough to guarantee reliability. High assurance applications require software components that can function correctly even when faced with improper usage or stressful environmental conditions".

The aim of this method is to guarantee reliability by using a model and test schema that require the software to pass several reliability (and risk) checks before they can be certified. "Improper usage" is reflected in the rate of failure incidence in the model and "stressful environmental conditions" is included by imposing the most stringent test conditions on safety critical software.

## VI. NASA Space Shuttle Application

The feasibility of applying the sequential reliability test concepts to the Shuttle flight software using the Schneidewind software reliability model [10] is investigated. Any software reliability growth model (SRGM) would suffice for this purpose.

An assumption of SRGMs is that reliability will increase with time, as faults are removed as they are discovered. Thus, a sufficiently long test time is used to: 1) collect failure data in order to estimate the model parameters and 2) allow reliability growth to take place (e.g., reliability reaches an acceptable level). Once 1) and 2) have been accomplished, the reliability of the software for the specified mission duration $t_{\mathrm{m}}$ can be predicted. The first step is to define model quantities.

### A. Definitions

*1. Risk*

According to NASA-STD-8719.13A [11], "risk is a function of: the possible frequency of occurrence of an undesired event, the potential severity of resulting consequences, and the uncertainties associated with the frequency and severity". This broad definition is used to encompass the specific model definition of risk as the probability (i.e., frequency of occurrence) of failures (i.e., undesired event), with failure count $r$ (i.e., potential severity), and variance of probability and failure count (i.e., uncertainties) occurring on a software release.

Safety critical: an application in which high risk and low reliability would jeopardize the safety of the crew and mission

*2. Actual (Empirical) Quantities*

$\mu_c(t, r_c)$: actual consumer software risk at time $t$ when $r_c$ failures have occurred
$\mu_p(t, r_p)$: actual producer software risk at time $t$ when $r_p$ failures have occurred
$P_{\mathrm{ac}}(t, r_c)$: actual consumer software probability of $r_c$ failures at time $t$
$P_{\mathrm{ap}}(t, r_p)$: actual producer software probability of $r_p$ failures at time $t$
$R_{\mathrm{ac}}(t, r_c)$: actual consumer software reliability computed over time $t$ and failure count $r_c$
$R_{\mathrm{ap}}(t, r_p)$: actual producer software reliability computed over time $t$ and failure count $r_p$
$\rho_c$: actual consumer software reliability growth
$\rho_p$: actual producer software reliability growth

*3. Consumer Estimated or Predicted Quantities*

$\alpha(t, r_c)$: consumer software risk: probability of consumer predicted failures: probability of accepting bad software at time $t$ when $r_c$ failures have occurred
$r_c(t)$: number of consumer software failures whose faults have been removed at time $t$
$m_c(t)$: consumer software predicted mean number of failures predicted to occur at time $t$
$L_m$: maximum allowable consumer risk for all values of $t$
$R_c(t)$: consumer software predicted reliability at time $t$

$R_{cs}$: specified minimum consumer software reliability
$\rho_{cp}$: predicted consumer software reliability growth
$\rho_{pp}$: predicted producer reliability growth

*4. Producer Estimated or Predicted Quantities*
$\beta(t, r_p)$: producer software risk: probability of producer predicted failures: probability of accepting bad software at time $t$ when $r_p$ failures have occurred
$r_p(t)$: number of producer software failures whose faults have been removed at time $t$
$m_p(t)$: producer software predicted mean number of failures predicted to occur at time $t$
$R_p(t)$: producer software predicted reliability at time $t$
$R_{ps}$: specified minimum producer software reliability, where $R_{ps} \leqslant R_{cs}$ (i.e., to favor the consumer in mission and safety critical applications).

### B. Risk Analysis

Next, the consumer and producer risk equations for the Shuttle are developed. These equations are used in the test rules and scenario and in the risk plots. Because there are several streams of failure data available for a given software release (i.e., operational increment (OI)), one failure stream can be used for the consumer and another for the producer. The logic of this is that the producer would deliver software with a certain reliability that the consumer would attempt to continue to increase by removing more faults. The next step is to formulate the risk equations (1–4):

$$\text{Predicted consumer risk} = \alpha(t, r_c) = [(m_c(t)^r e_c^{-m(t)})/r_c!]r_c \tag{1}$$

$$\text{Predicted producer risk} = \beta(t, r_p) = [(m_p(t)^r e_p^{-m(t)})/r_p!]r_p \tag{2}$$

$$\text{Actual consumer Risk} = \mu_c(t, r_c) = P_{ac}(t, r_c)r_c \tag{3}$$

$$\text{Actual producer Risk} = \mu_p(t, r_p) = P_{ap}(t, r_p)r_p \tag{4}$$

### C. Reliability Analysis

In addition to risk, the second component of the model is reliability. Using a reliability growth model that has been used on the Shuttle, the equations that are used in the test rules and scenario are developed. The general form of consumer and producer reliability at time $t$ is given by Eq. (5) [10]

$$R(t) = \exp -[a(e - b(t - s + 1))] \tag{5}$$

where $a$, $b$, and $s$ are model parameters estimated from the Shuttle failure data.

### D. Reliability Growth

For safety critical systems such as the Shuttle, it is important to demonstrate reliability growth, as contributing to the safety of the crew and mission. As pointed out by Musa et al. [12], it may be necessary for an organization to demonstrate the reliability of its product "as delivered". For example, there could be a test where the consumer "buys off" the product from the producer. In this case, particularly for safety critical software, the test model and scenario must enforce a high standard of reliability (and risk) before the product is accepted.

As the Shuttle uses a reliability growth model, test rules are conditioned to capture this important characteristic. To compute reliability growth quantitatively, Tian [13] suggests that reliability growth can be measured by the purification level $\rho$, i.e., the ratio between the number of defects removed during testing over the total defects at the beginning of testing. He states that the purification level captures overall reliability growth and testing effectiveness. The objective of my model's use of purification level is to produce tests that have high testability (i.e., use tests that will cause failures to be detected and faults to be exposed and removed).

## VII. Test Rules

Test rules are based on: 1) mandatory risk, reliability, risk prediction accuracy, reliability prediction accuracy, and reliability growth; and 2) desirable reliability growth. Although important, desirable reliability growth is not

considered as important as the other criteria. This is a subjective judgment that the model user might want to change. In addition, as indicated in Fig. 1, the test scenario consists of two complete tests for the software to be accepted or rejected. This is another feature that the model user could change.

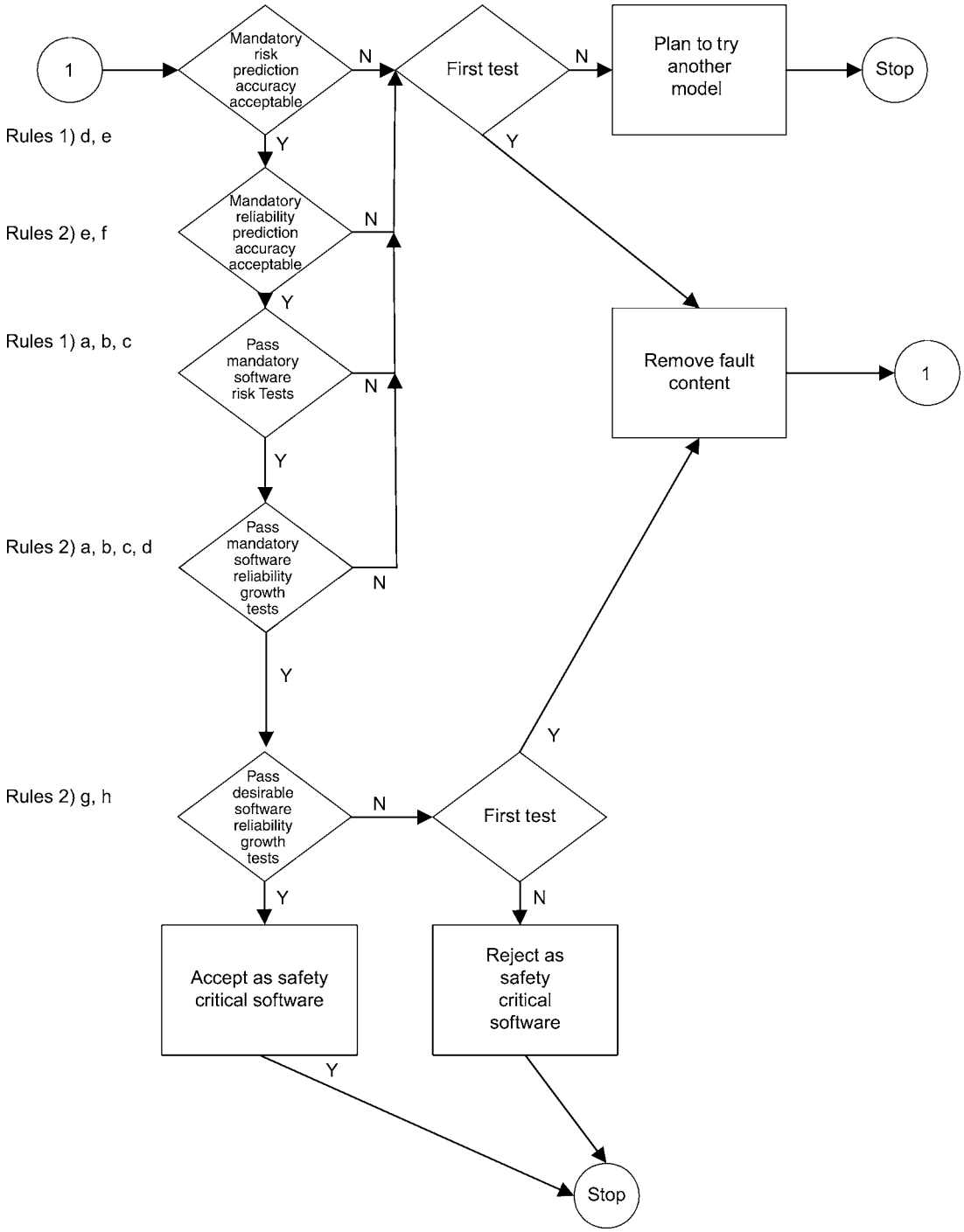


**Fig. 1 Shuttle risk-driven testing and reliability process.**

Accept software if the following software rules evaluate to "true". Accept model if the following model rules evaluate to "true".

For all values of time $t = t_{\mathrm{m}}$ (mission duration):

1) *Risk*:

*Mandatory for software*

Predicted and actual consumer and producer risks less than the limit:

a. Consumer risk, $\alpha(t, r_c) < L_{\mathrm{m}}$;
b. Producer risk, $\beta(t, r_p) < L_m$;
c. Actual consumer risk, $\mu_c(t, r_c) < L_m$, actual producer risk, $\mu_p(t, r_p) < L_m$;

*Mandatory for prediction model*

Consumer and producer risk prediction errors less than the limits:

d. Consumer prediction error: $[\mu_c(t, r_c) - \alpha(t, r_c)]^2 <$ (consumer risk mean error + 3 standard deviations);
e. Producer prediction error: $[\mu_p(t, r_p) - \beta(t, r_p)]^2 <$ (producer risk mean error + 3 standard deviations).

2) *Reliability*

*Mandatory for software reliability growth*

Predicted and actual consumer and producer software reliabilities exceed the thresholds:

a. Predicted consumer reliability: $R_c(t) > R_{\mathrm{cs}}$;
b. Predicted producer reliability: $R_p(t) > R_{\mathrm{ps}}$;
c. Actual consumer reliability: $R_{\mathrm{ac}}(t) > R_{\mathrm{cs}}$;
d. Actual producer reliability: $R_{\mathrm{ap}}(t) > R_{\mathrm{ps}}$.

*Mandatory for prediction model*

Consumer and producer reliability prediction errors less than the limits:

e. Consumer prediction error: $[R_{\mathrm{ac}}(t, r_c) - R_c(t, r_c)]^2 <$ (consumer reliability mean error + 3 standard deviations);
f. Producer prediction error: $[R_{\mathrm{ap}}(t, r_p) - R_p(t, r_p)]^2 <$ (producer risk mean error + 3 standard deviations).

*Desirable for software reliability growth*

Predicted consumer and producer software reliability growths exceed the actual growths:

g. Predicted consumer reliability growth: $\rho_{\mathrm{cp}} > \rho_c$;
h. Predicted producer reliability growth: $\rho_{\mathrm{pp}} > \rho_p$.

The Shuttle test rules, based on using risk, reliability, and reliability growth criteria, are shown in Fig. 1.

## VIII. NASA Space Shuttle Application Example

### A. Model Parameters

Based on actual Shuttle release failure data, the following parameters in the example problem are specified:

$t$: consumer test time = 1,…,25; consumer prediction range = 26,…,45

$t$: producer test time = 1,…,36; producer prediction range = 37,…,45

$t_m$: = desired mission duration = 8 (45 – 37)

Test time, above, can be different for consumer and producer because each may choose to test a different amount of time, dependent on their risk and reliability objectives. For example, the producer may have more resources that the consumer to do testing and, therefore, test for a longer time, and, in addition, is being paid by the consumer to do testing. This difference leads to different prediction ranges, given the end of the mission at $t = 45$. Of course, the mission duration must be the same for consumer and producer. A mission duration of 8 days is typical for the Shuttle.

$r_c$: mean of consumer failure distribution = 0.2400 failures

$r_p$: mean of producer failure distribution = 0.1818 failures

Since $r_c$ is needed in the computation of consumer risk for the prediction range $t = 26, \ldots, 45$, and there are no historical data available for this range, this quantity was predicted. Likewise, $r_p$ for producer risk in the prediction range $t = 37, \ldots, 45$ was predicted.

### B. Model Limits and Thresholds

$L_m$: risk limit = 0.500 000

What is the basis for this risk limit? Admittedly, it is somewhat subjective, but it is based on the following consideration: risk is the (probability of $r$ failures) $x$ ($r$ failure count). For this software, it is reasoned that the

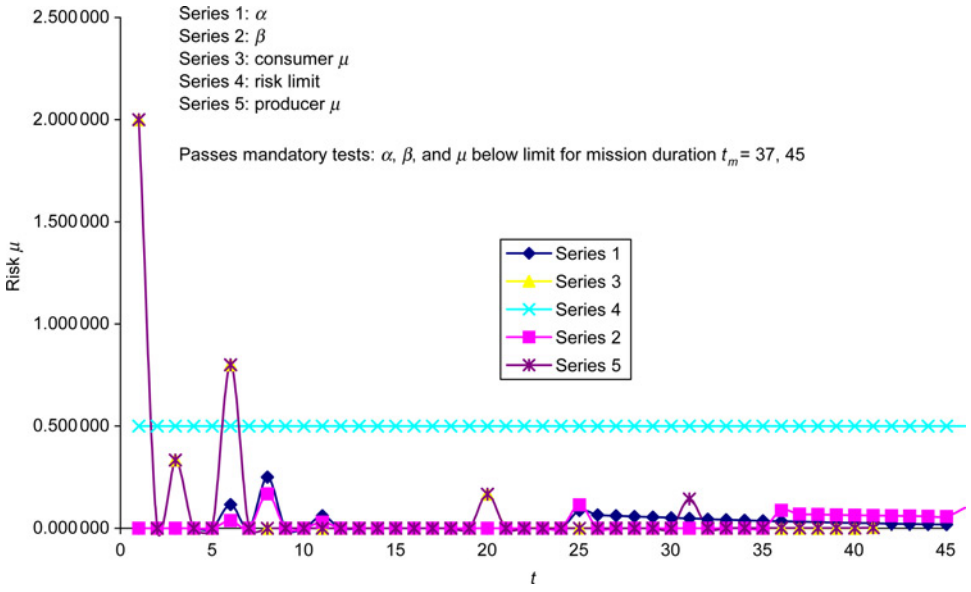


**Fig. 2 Shuttle test: consumer risk(alpha), producer risk (beta), and actual risk ($\mu$) vs test time $t$.**

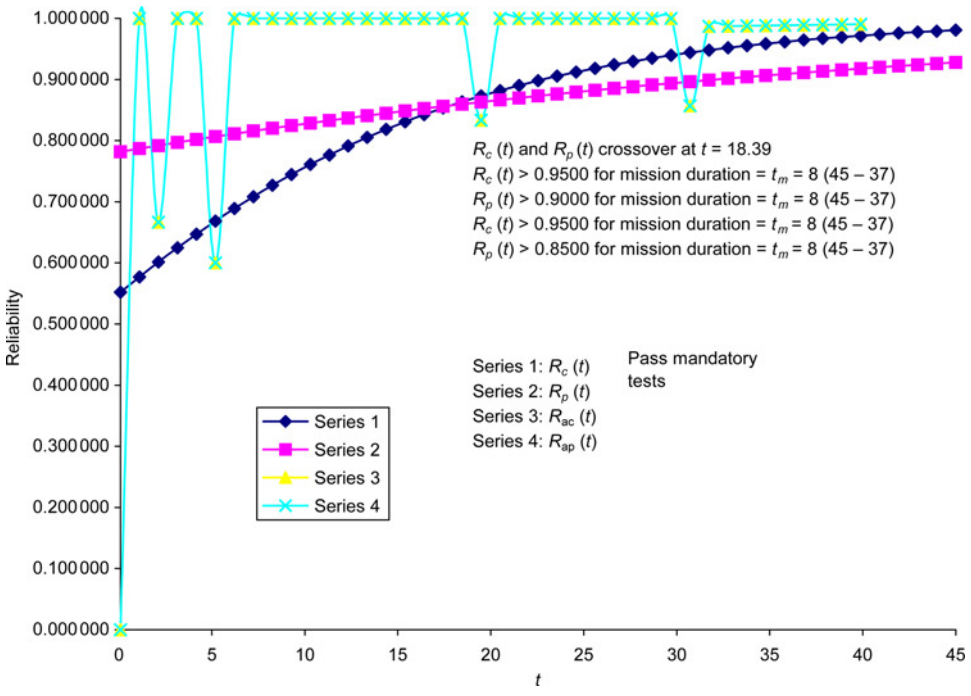


**Fig. 3 Shuttle first test: consumer reliability $R_c(t)$, producer reliability $B_p(t)$, actual consumer reliability $R_{ac}(t)$, and actual producer reliability $R_{ap}$ vs test time $t$.**

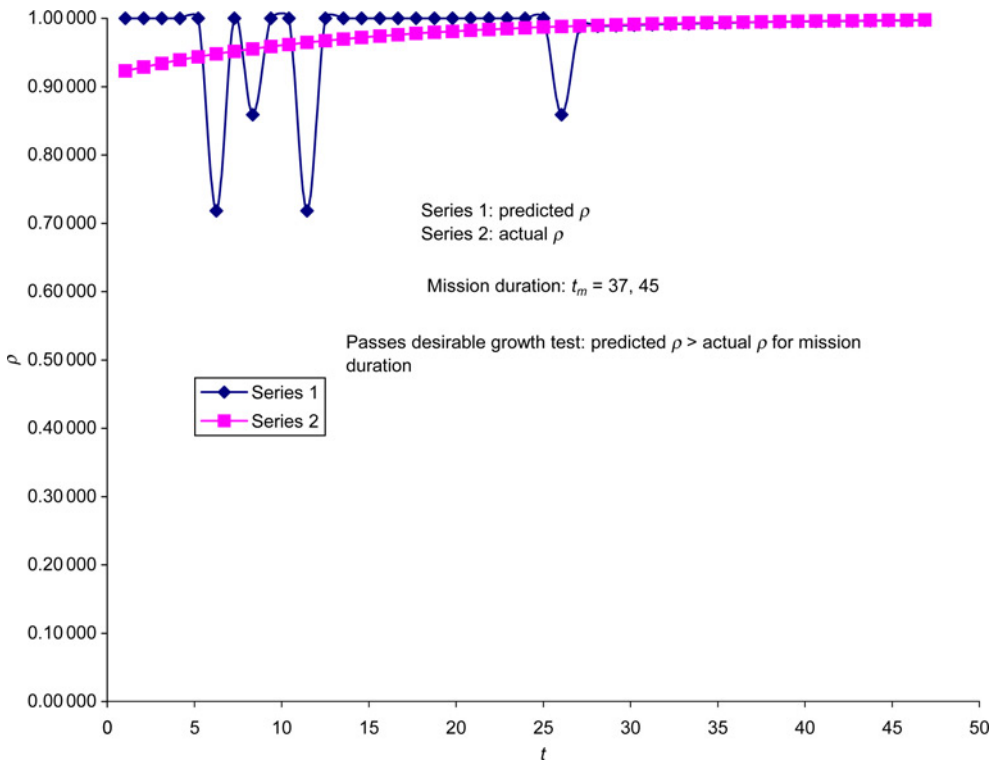


**Fig. 4 Shuttle first test: consumer reliability growth ($\rho$) vs test time $t$.**

**Table 1 Shuttle test results**

| Step | Test rules | Outcome | Action | Details |
|---|---|---|---|---|
| 1. Mandatory risk prediction accuracy | 1) d, e | Pass | Go to step 2 | $S_{\mathrm{Rc}} \leqslant (E_{\mathrm{Rc}} + 3\sigma)$ for $t = t_m$ $S_{\mathrm{Rp}} \leqslant (E_{\mathrm{Rp}} + 3\sigma)$ for $t = t_m$ |
| 2. Mandatory reliability prediction accuracy | 2) e, f | Pass | Go to step 3 | $Sr_{\mathrm{c}} \leqslant (E_{\mathrm{rc}} + 3\sigma)$ for $t = t_m S_{\mathrm{rp}} \leqslant (E_{\mathrm{rp}} + 3\sigma)$ for $t = t_m$ |
| 3. Mandatory software risk | 1) a, b, c | Pass | Go to step 4 | See Fig. 2 |
| 4. Mandatory reliability growth | 2) a, b, c, d | Pass | Go to step 5 | See Fig. 3 |
| 5. Desirable software reliability growth | 2) g, h | Pass | Accept software as safety critical | See Fig. 4 |

Definitions:
Consumer software risk error: $S_{\mathrm{Rc}}$
Consumer software mean risk error: $E_{\mathrm{Rc}}$
Producer software risk error: $S_{\mathrm{Rp}}$
Producer software mean risk error: $E_{\mathrm{Rp}}$
Consumer software reliability error: $Sr_{\mathrm{c}}$
Consumer software mean reliability error: $E_{\mathrm{rc}}$
Producer software reliability error: $S_{\mathrm{rp}}$
Producer software mean reliability error: $E_{\mathrm{rp}}$
$t_m$: Mission duration

probability is 0.5 for $r = 1$, or $L_m = 0.5$. The reason for six decimal places in the risk limit is that actual, consumer, and producer risks can be very small, as you can see in Fig. 2.
$R_{cs}$: specified minimum consumer reliability $= 0.9500$
$R_{ps}$: specified minimum producer reliability, where $R_{ps} < R_{cs} = 0.9000$

My choice of reliability thresholds is based on the criticality of this mission to the safety of the crew and Shuttle, and the need to impose a higher threshold for consumer reliability because, ultimately, the consumer has responsibility for the success of the mission.

### C. Results from Shuttle Test

Key figures and summarized results are provided in Table 1. The first key figure, Fig. 2 indicates that mandatory risks tests have been passed. The second key figure, Fig. 3 indicates that mandatory reliability tests have been passed. The third key figure, Fig. 4 demonstrates that the desirable reliability growth test for the consumer has been passed. The producer test was also passed, but its plot is not shown because the result is almost identical to the consumer test. The final outcome, as noted in Table 1, is that this safety critical software has been accepted.

## IX. Conclusions

1) For safety critical systems such as the Shuttle, it is important to demonstrate low risk, high reliability, and reliability growth. Thus, the model and test scenarios include all of these criteria.
2) The detailed analysis required by my model and test process provides a great deal of insight into the complex interrelationships among consumer and producer risk and reliability.

## References

[1] Whittaker, J. A., "What is Software Testing? And Why is it so Hard?", *IEEE Software*, Vol. 17, No. 1, Jan.–Feb. 2000, pp. 70–79.
[2] David, L. K., and Lipow, M., *Reliability: Management, Methods, and Mathematics*, Prentice-Hall, Upper Saddle River, NJ, 1962.
[3] Horgan, J. R., and Mathur, A. P., "Software Testing and Reliability", *Handbook of Software Reliability Engineering*, edited by M. M. Lyu, Computer Society Press, Los Alamitos, CA, 1996, pp. 531–563.
[4] Keller, T., and Schneidewind, N. F., "A Successful Application of Software Reliability Engineering for the NASA Space Shuttle", *Software Reliability Engineering Case Studies, International Symposium on Software Reliability Engineering*, Albuquerque, NM, pp. 71–82, Nov. 1997.
[5] Cangussu, J. W., Mathur, A. P., and DeCarlo, R. A., "Feedback Control of the Software Test Process through Measurements of Software Reliability", *Proceedings of the 12th International Symposium on Software Reliability Engineering*, 2001. ISSRE 2001, pp. 232–241, Nov. 2001.
[6] Jursto, N., Moreno, A. M., Vegas, S., and Solari, M., "In Search of What We Experimentally Know about Unit Testing", *IEEE Software*, Vol. 23, No. 6, Nov./Dec. 2006, pp. 72–79.
[7] Lyu, M. R., "An Integrated Approach to Achieving High Software Reliability", *Proceedings of the Aerospace Conference*, Vol. 4, pp. 123–136, March 1998, Snowmass at Aspen, CO, USA.
[8] Myers, G. J., *The Art of Software Testing*, Wiley, New York, 1979.
[9] Schmid, M., Ghosh, A., and Hill, F., "Techniques for Evaluating the Robustness of Windows NT Software", *DARPA Information Survivability Conference and Exposition*, Hilton Head, SC, Jan. 2000.
[10] Schneidewind, N. F., "Reliability Modeling for Safety Critical Software", *IEEE Transactions on Reliability*, Vol. 46, No.1, March 1997, pp. 88–98.
[11] "Software Safety", NASA Technical Standard, NASA-STD-8719.13A, Sep. 1997.
[12] Musa, J. D., Iannino, A., and Okumoto, K., *Software Reliability: Measurement, Prediction, Application*, McGraw–Hill, New York, 1987.
[13] Tian, J., "Integrating Time Domain and Input Domain Analyses of Software Reliability Using Tree-Based Models", *IEEE Transactions On Software Engineering*, Vol. 21, No. 12, Dec. 1995, pp. 945–958.
[14] Fenton, N. F., and Pfleeger, S. L., *Software Metrics: A Rigorous and Practical Approach*, 2nd ed., PWS Publishing, Boston, MA, 1997.

Michael Hinchey
*Associate Editor*