JOURNAL OF AEROSPACE COMPUTING, INFORMATION, AND COMMUNICATION
Vol. 4, July 2007

Investigation of State Transition Model for Predicting Software Reliability

Norman Schneidewind*
Professor Emeritus, Naval Postgraduate School, Monterey CA 93942

We compare a state transition model that is developed in this paper with previously developed software reliability models with respect to prediction accuracy. Two NASA Space Shuttle software releases are used to make the comparison. Unexpectedly, the state transition model did not do well, compared to its opponents, in predicting reliability and remaining failures. Furthermore, a single parameter, homogeneous Poisson model we had developed, did not faire much better. Only the two parameter, non-homogeneous Poisson model produced acceptable prediction accuracy. This result is attributed to the ability of the non-homogeneous Poisson model to capture more of the variation in failure data and, thus, estimate model parameters more accurately.

I. Introduction

GENERALLY speaking, a software system is said to be "reliable" if it performs its specified functions correctly over a long period of time or under a wide variety of usage environments and operations. There are two commonly used approaches to software reliability analysis: time domain and input domain. In the time domain approach: What is the software's ability to perform correctly over time under a given operational environment? What is the likelihood of failures for a specified length of time? Software reliability growth models are commonly used to answer these questions by analyzing time-indexed failure data.

With the input domain approach: What is the software's ability to perform correctly for various different inputs? For a selected set of input states, what is the likelihood of software failures? Models based on repeated sampling are commonly used to address these issues by analyzing input states and failure data.

These two approaches provide two different perspectives of reliability. The time domain approach stresses the assessment and prediction of the overall reliability. The input domain approach provides valuable information that can be used to thoroughly test software products due to the use of well-defined input states.[11]

Our goal in this research is to analyze both approaches and compare their predictive accuracy with respect to a variety of reliability metrics. In the time domain arena, we look at software reliability models; in the input domain case, we investigate state transition models.

II. Prior Research on State Transition Models for Software

1. A Markov process consists of a set of objects (e.g., software modules), a set of states (e.g., reliability), and the times (e.g., software development time) when the objects and states exist. The probability of transitioning to the next state depends only on the present state.[1,6] We use a modified version of the Markov process that allows for the probability of transitioning to the next state to depend only the *immediate past state and the present state*. We model the probability of transitioning from state i to state j that tracks the changes in software reliability.

Received 06 May 2007; revision received; accepted for publication 22 May 2007.
* ieeelife@yahoo.com

2. It is fairly common to use a Markov process or its variants to model software characteristics. For example, in,[12] the authors report on their research as follows: they present an analytical model for estimating architecture-based software reliability, according to the reliability of each component, the operational profile, and the architecture of software. Their approach is based on Markov chain properties and state transformations to perform reliability analysis on heterogeneous software architectures. They demonstrated how this analytical model can be utilized to estimate the reliability of a heterogeneous architecture.
3. Software testing is directly tied to software reliability in the sense that each test causes a change in state based on the fact that the reliability has changed as a result of the tests, assuming that failures have been discovered and corrected. For example, according to[3] they consider a common testing scenario: a tester applies an input and then appraises the result. The tester then selects another input, depending on the prior result, and once again reappraises the next set of possible inputs. At any given time, a tester has a specific set of inputs to choose from. This set of inputs varies depending on the state of the software. This characteristic of software makes state-based models a logical fit for software testing: software is always in a specific state and the current state of the application governs what set of inputs testers can select. Thus, we feel the foregoing is an important reason for formulating a state transition model to track reliability changes that result from tests.
4. Fortunately, for the crew of the NASA Space Shuttle, the software has demonstrated high reliability. Unfortunately, with respect to conducting software experiments, the sample sizes are small, leading to the need to perform sensitivity analysis to verify the results. As stated by:[2] Sensitivity analysis is a mathematical methodology to compute changes in the system behavior due to changes in system parameters or variables. This is particularly important when parameters are calibrated using noisy or small data sets. Nevertheless, by mathematically quantifying the effects of parameter variations on the behavior of the model, one can evaluate the effect of such variations on the model performance.

   Two of our state transition model parameters are state and time. We vary these for eight and nine pairs of state-time failure count relationships on two Shuttle flight software releases to verify model reliability predictions.
5. An additional argument for the link between testing and state transition events is articulated by:[7] Automatic testing methods are supported with testing data and state-transition specifications. In automatic testing based on specifications, a test scenario, driven by the state-transition specifications, is conducted.

   Similarly, our test scenario and scenario results are implemented by combining test states and times with state transitions of reliability caused by the test process.

III. Objective of Research

The occurrence of failures accompanied by state changes at specific times in the evolution of software, is a rich mix for stimulating research into how these elements interact. To illustrate, if failures occur, the testing process is repeated (i.e., the state of the software changes). We can evaluate the system states and reliability by analyzing the failure data and test results.[4] This we do, as described below.

We are motivated to investigate the trace of reliability evolution fostered by reliability state transitions, as the states change as a function of the effectiveness of the software development process in each state. To accomplish this objective, we developed a state transition model to track reliability changes of the NASA Space Shuttle flight software. We think this type of model has a lot of attractiveness because one can predict the state changes that can yield either low or high reliability, and investigate the cause of each. An organization employing this model would then have an effective tool for monitoring and improving software quality.

We are also interesting in seeing whether a state transition model can more accurately predict a variety of reliability metrics than a software reliability model.

IV. Definitions

Empirical: value based on historical failure data
STM: Transition State Model
SSPM: Schneidewind Single Parameter Model

SSRM: Schneidewind Software Reliability Model

Note: unless otherwise indicated, definitions apply to STM

States
State i: number of failures $x(i, t)$ observed in the software at time t. A state is also equivalent to a test.
From State: i
To State: j
Time: t
$\lambda(i, j, t)$: rate of transitions from state i to state j at time t (failures per time)
$\lambda[(i-1), j, (t-1)]$: rate of transitions from state $(i-1)$ to state j at time $(t-1)$ (failures per time)

Probabilities
$\mathrm{P}(i, t)$: probability of being in state i at time t
$\mathrm{P}(j, t)$: probability of being in state j at time t
$\mathrm{P}(i, j, t)$: probability of transitioning from state i to j at time t
$\mathrm{P}_c(i, t)$: probability of failure correction in state i at time t

Reliabilities
$\mathrm{R}(i, t)$: reliability of software in state i at time t
$\mathrm{R}(j, t)$: reliability of software in state j at time t

$\mathrm{R}_r(i, t)$: revised reliability of software based on correcting failures in state i at time t
$\mathrm{R}_r[(i-1), (t-1)]$: revised reliability of software in based on correcting failures in state $(i-1)$ at time $(t-1)$
$\mathrm{R}_s(i, t)$: revised SSPM reliability of software based on correcting failures in state i at time t

$\mathrm{R}_e(i, t)$: empirical reliability of software in state i at time t
$\mathrm{R}_e[(i-1), (t-1)]$ empirical reliability of software in state $(i-1)$ at time $(t-1)$
$\mathrm{R}_{er}(i, t)$: revised empirical reliability of software in state i at time t, based on correcting failures
$\mathrm{R}_{er}[(i-1), (t-1)]$ revised empirical reliability of software in state $(i-1)$ at time $(t-1)$, based on correcting failures

Mean Values
$\mathrm{m}(i)$: mean number of failures in software when in state i
$\mathrm{m}_r(i)$: revised mean number of failures based on correcting failures in state i at time t
$\mathrm{m}(i, t)$: mean number of failures in software in state i at time t
$\mathrm{m}_r[(i-1), (t-1)]$: revised mean number of failures based on correcting failures in state $(i-1)$ at time $(t-1)$
$\mathrm{m}_e(i, t)$: empirical mean number of failures in software in state i at time t

Failure Counts
$\mathrm{x}(i, t)$: number of failures in state i at time t
$\mathrm{x}s[(i-1), (t-1)]$: number of failures in state $(i-1)$ at time $(t-1)$

Remaining Failures
$\mathrm{n}_f(i, t)$: number of failures in a program in state i at time t
$\mathrm{n}_f[(i-1), (t-1)]$: number of failures in a program in state $(i-1)$ at time $(t-1)$
n_{fr}: (i, t) revised number of failures in a program based on correcting failures in state i at time t (i.e., remaining failures)
n_{fr}: $[(i-1), (t-1)]$ revised number of failures in a program based on correcting failures in state $(i-1)$ at time $(t-1)$ i.e., remaining failures)

$r_r(i, t)$: *total* remaining failures in state i at time t
r_r : $[(i-1), (t-1)]$ *total* remaining failures in state $(i-1)$ at time $(t-1)$
$r_s(i, t)$: remaining failures in state i at time t for SSPM
$r_e(i, t)$: empirical remaining failures in state i at time t

Parameters
— α SSRM Failure rate at the beginning of interval s
— β SSRM Negative of derivative of failure rate divided by failure rate (i.e., relative failure rate
— s SSRM Starting interval for using observed failure data in parameter estimation
β: SSPM failure rate parameter

V. Equations

1. STM

We assume that the probability of transitioning from state i to state j at time t, given that the system has been in state $(i-1)$ at time $(t-1)$, and state i at time t, is the maximum of past and current probabilities, based on the concept that transitions will occur with maximum probability. For example, the software will transition from state i to state j dependent on in which state the software developer is producing the more reliable software. Note that the STM only covers the present and immediately preceding states. Then, the transition equation is given as:

$$\mathrm{P}(i, j, t) = \max\{[\mathrm{P}[(i-1), (t-1)], \{[\mathrm{P}[(i, t)]\} \tag{1}$$

The probability of being in state i at time t is assumed to be governed by a Poisson distribution, as follows:

$$\mathrm{P}(i, t) = \frac{\mathrm{m}(i, t)^{\mathrm{x}(i,t)} e^{-\mathrm{m}(i,t)}}{\mathrm{x}(i, t)!} \tag{2}$$

This means that the outcome of the state transition decision specified in equation (1) [see Fig. 1] will be determined by the maximum value of Eq. (1).

Now, using the state transition rate, we need Eq. (4) in order to predict the mean number of failures in state i at time t in Eq. (2). This is done in Eq. (3):

$$\mathrm{m}(i, t) = [\lambda(i, j, t)]^{*} t \tag{3}$$

Equation (4) gives us the rate of state transitions when in state i at time t, and Eq. (5) provides the rate when in state $(i-1)$ at time $(t-1)$. This means we expect the reliability to change according to this rate. For the Shuttle, failures (i.e., state changes) are recorded in time periods of 30 days. Note that the state transition rate is equivalent to MTTF. This means that we would expect to observe a decreasing MTTF associated with increasing reliability. We demonstrated this relationship across all Shuttle OIs from 9/1/83 to 3/5/97.[9]

$$\lambda(i, j, t) = \mathrm{x}\frac{(i, t)}{t} \tag{4}$$

$$\lambda[(i-1), j, (t-1)] = \frac{\mathrm{x}[(i-1), (t-1)]}{(t-1)} \tag{5}$$

Then, using Eqs. (4) and (5), we predict the mean number of failures in Eqs. (16) and (7), respectively:

$$\mathrm{m}(i, t) = \mathrm{x}(i, t) \tag{6}$$

And it follows that:

$$\mathrm{m}[(i-1), (t-1)] = \mathrm{x}[(i-1), (t-1)] \tag{7}$$

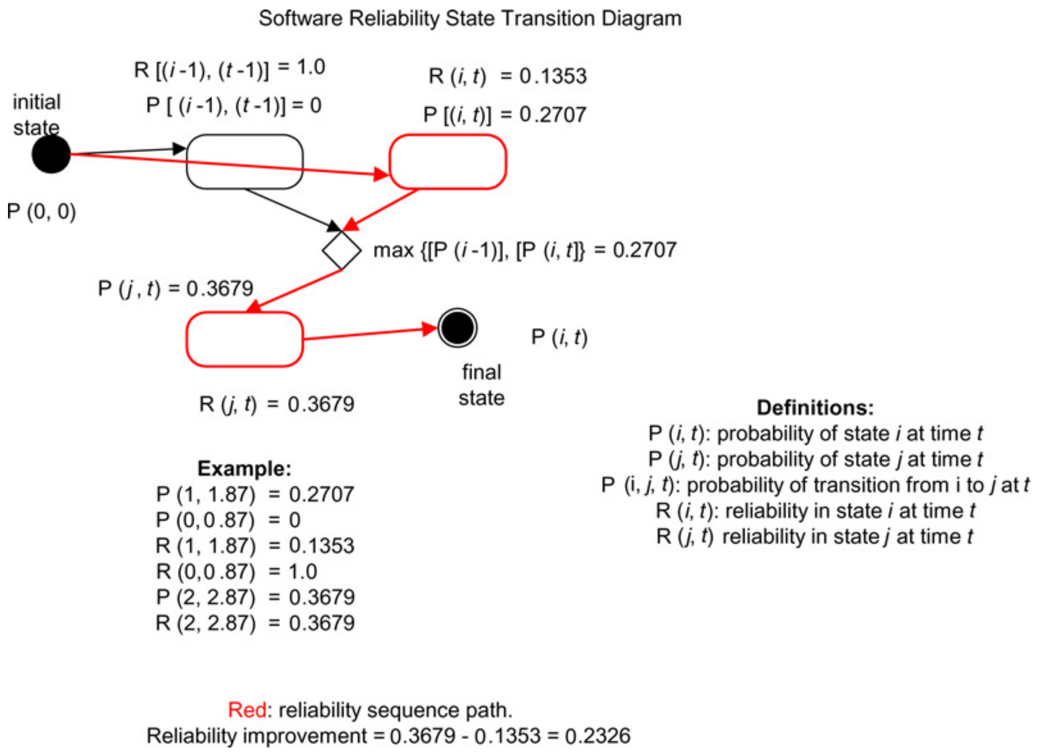

Software Reliability State Transition Diagram
R [(i -1), (t -1)] = 1.0
P [(i -1), (t -1)] = 0
R (i, t) = 0.1353
P [(i, t)] = 0.2707
initial state
P (0, 0)
max {[P (i -1)], [P (i, t]} = 0.2707
P (j , t) = 0.3679
P (i, t)
final state
R (j, t) = 0.3679
Definitions:
P (i, t): probability of state i at time t
P (j, t): probability of state j at time t
P (i, j, t): probability of transition from i to j at t
R (i, t): reliability in state i at time t
R (j, t) reliability in state j at time t
Example:
P (1, 1.87) = 0.2707
P (0,0 .87) = 0
R (1, 1.87) = 0.1353
R (0,0 .87) = 1.0
P (2, 2.87) = 0.3679
R (2, 2.87) = 0.3679
Red: reliability sequence path.
Reliability improvement = 0.3679 - 0.1353 = 0.2326

Fig. 1 Software reliability state transition diagram.

In order to implement Eq. (1), we need Eq. (2), with the mean number of failures inserted from Eqs. (6) and (7), as shown in Eqs. (8) and (9), respectively:

$$\mathrm{P}(i, t) = \frac{[\mathrm{m}(i, t)]^{x(i,t)} \mathrm{e}^{-[m(i,t)]}}{\mathrm{x}(i, t)!} \tag{8}$$

$$\mathrm{P}[(i-1), (t-1)] = \frac{\mathrm{m}[(i-1), (t-1)]^{\mathrm{x}[(i-1),(t-1)]} \mathrm{e}^{-\mathrm{m}[(i-1),(t-1)]}}{\mathrm{x}[(i-1), (t-1)]!} \tag{9}$$

2. SSPM

Next we develop the reliability for SSPM in state i at time t, $\mathrm{R}(i, t)$. According to,[10] $\mathrm{R}(i, t)$ is given by:

$$\mathrm{R}(i, t) = \mathrm{e}^{-\mathrm{m}(i,t)} \tag{10}$$

where the mean value of Eq. (10) is given by: $\mathrm{m}(i) = \mathrm{e}^{-\beta i}$, and β is the single parameter of this model.[10]

Note that we obtain the same result by setting the number of failures $\mathrm{x}(i, t)$in a Poisson distribution to zero [see Eq. (2)].

Then the reliability in state $(i-1)$ and time $(t-1)$ is predicted by:

$$\mathrm{R}[(i-1), (t-1)] = \mathrm{e}^{-\mathrm{m}[(i-1),(t-1)]} \tag{11}$$

3. STM and SSPM Failure Correction

We assume that the probability of correcting failures is proportional to the number of failures present in the software in state i at time t. Therefore, it follows that this probability at state i and time t is equal to:

$$\mathrm{P_c}(i, t) = \frac{[x(i, t)]}{n_f(i, t)} \tag{12}$$

Where $\mathrm{n_f}(i, t)$ is the number of failures in a program in state i at time t.

Then using Eq. (12) as a template, this probability in state $(i-1)$ at time $(t-1)$ is equal to:

$$\mathrm{P_c}[(i-1),(t-1)] = \frac{[\mathrm{x}\{(i-1),(t-1)\}]}{n_f[(i-1),(t-1)]} \tag{13}$$

Recognizing that the expected number of failures corrected is equal to probability of correction times number of failures, we can predict this metric in the two states, using Eqs. (12) and (13), in Eqs. (14) and (15), respectively:

$$\mathrm{n_c}(i,t) = \frac{[\mathrm{x}(i,t)]^2}{\mathrm{n_f}(i,t)} \tag{14}$$

$$\mathrm{n_c}[(i-1),(t-1)] = \frac{[\mathrm{x}\{(i-1),(t-1)\}]^2}{\mathrm{n_f}[(i-1),(t-1)]} \tag{15}$$

4. Empirical Reliability

We also want to compare reliability predictions, such as mean number of failures, with the empirical values. The empirical mean number of failures in state i at time t is computed in Eq. (16):

$$\mathrm{m_e}(i,t) = \frac{\mathrm{x}(i,t)}{\mathrm{n_f}(i,t)} \tag{16}$$

Furthermore, in order to compare predicted reliabilities with their empirical values, we need the latter in Eqs. (17) and (18), which compute empirical reliability in state i at time t and empirical reliability in state $(i-1)$ at time $(t-1)$, respectively:

$$\mathrm{R_e}(i,t) = \left(1 - \frac{\mathrm{x}(i,t)}{\sum_{i=1}^{\mathrm{n_f}(i,t)} \mathrm{x}(i,t)}\right) \tag{17}$$

$$\mathrm{R_e}[(i-1),(t-1)] = \left(1 - \frac{\mathrm{x}\{(i,t),[(i-1),(t-1)]\}}{\sum_{i=1}^{\mathrm{n_f}\{(i,t),[(i-1),(t-1)]\}} \mathrm{x}\{(i,t),[(i-1),(t-1)]\}}\right) \tag{18}$$

VI. Example Software State Transition Process

An example of a state transition process is shown in Fig. 1, where based on calculations made with a C++ program, we show how reliability can be improved in transition from a given development state to one with higher quality to produce an incremental gain in reliability. The example is based on failure data from the Shuttle OI6 software release. The reliability values are low because the predictions assume that the faults causing the failures have not been removed. Later, we will remove that assumption and show the revised reliability picture.

VII. Analysis of State Transitions

In order to see whether reliability state transitions are effective, we provide Figs. 2 and 3 for comparison. Also, we wish to compare STM against SSPM with respect to prediction accuracy. Two facts emerge from this comparison: 1) there is improvement in reliability in transiting from state i to state j and 2) prediction accuracy of STM is poor in absolute terms and when compared to SSPM. However, these comparisons are sans failure correction. Therefore, we are interested in comparing results once failures have been corrected.

A. Tracking Reliability with State Transitions

It is of interest to see what effect the state transition rate has on reliability because a rapidly changing rate could result in reliability volatility whereas a stable rate would indicate reliability quiescence. Figure 4 demonstrates that the rate –computed both in the immediate past state and in the present state—correlates with reliability. That is, when the rates change rapidly the effect on reliability is chaotic. In contrast, when the rates asymptotically approach zero, reliability stabilizes. The message for a software development organization is to use the transition rate as a barometer of reliability volatility. As noted previously, λ is equivalent to MTTF, so we are not surprised by the result of increasing MTTF accompanied by increasing reliability.

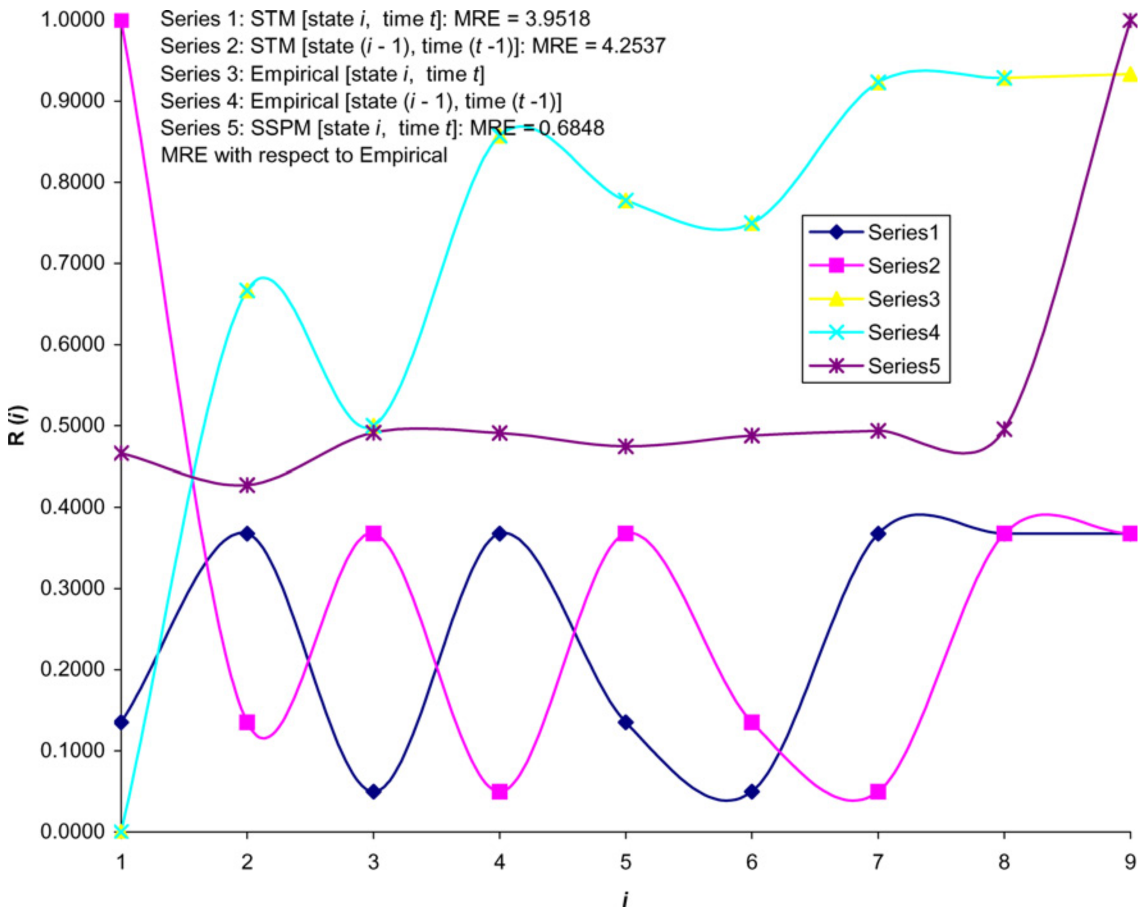


Fig. 2 NASA Space Shuttle OI6 reliability R(i) vs. state i.

B. Revising Reliability State Transitions Based on Correcting Failures

1. STM

To make the revisions in state transitions, first we recompute the number of failures in state i at time t and the number in state $(i-1)$ at time $(t-1)$ in Eqs. (19) and (20), respectively. The computation is made by subtracting the expected number of failures corrected from the number of failures in the program in a given state, at a given time (i.e., the computation yields remaining failures).

$$\mathrm{n_{fr}}(i,t) = \mathrm{n_f}(i,t) - \mathrm{n_c}(i,t) \tag{19}$$

$$\mathrm{n_{fr}}[(i-1),(t-1)] = \mathrm{n_f}[(i-1),(t-1)] - n_c[(i-1),(t-1)] \tag{20}$$

Once the revised number of failures is available, the mean values can be computed in Eqs. (21) and (22):

$$\mathrm{m_r}(i,t) = \mathrm{n_{fr}}(i,t) \tag{21}$$

$$\mathrm{m_r}[(i-1),(t-1)] = \mathrm{n_{fr}}[(i-1),(t-1)] \tag{22}$$

Then, our objective is reached in using Eqs. (23) and (24) to predict the revised reliabilities:

$$\mathrm{R_r}(i,t) = \mathrm{e_r^{-m(\mathit{i},\mathit{t})}} \tag{23}$$

$$\mathrm{R_r}[(i-1),(t-1)] = \mathrm{e_r^{-m}}[(i-1),(t-1)] \tag{24}$$

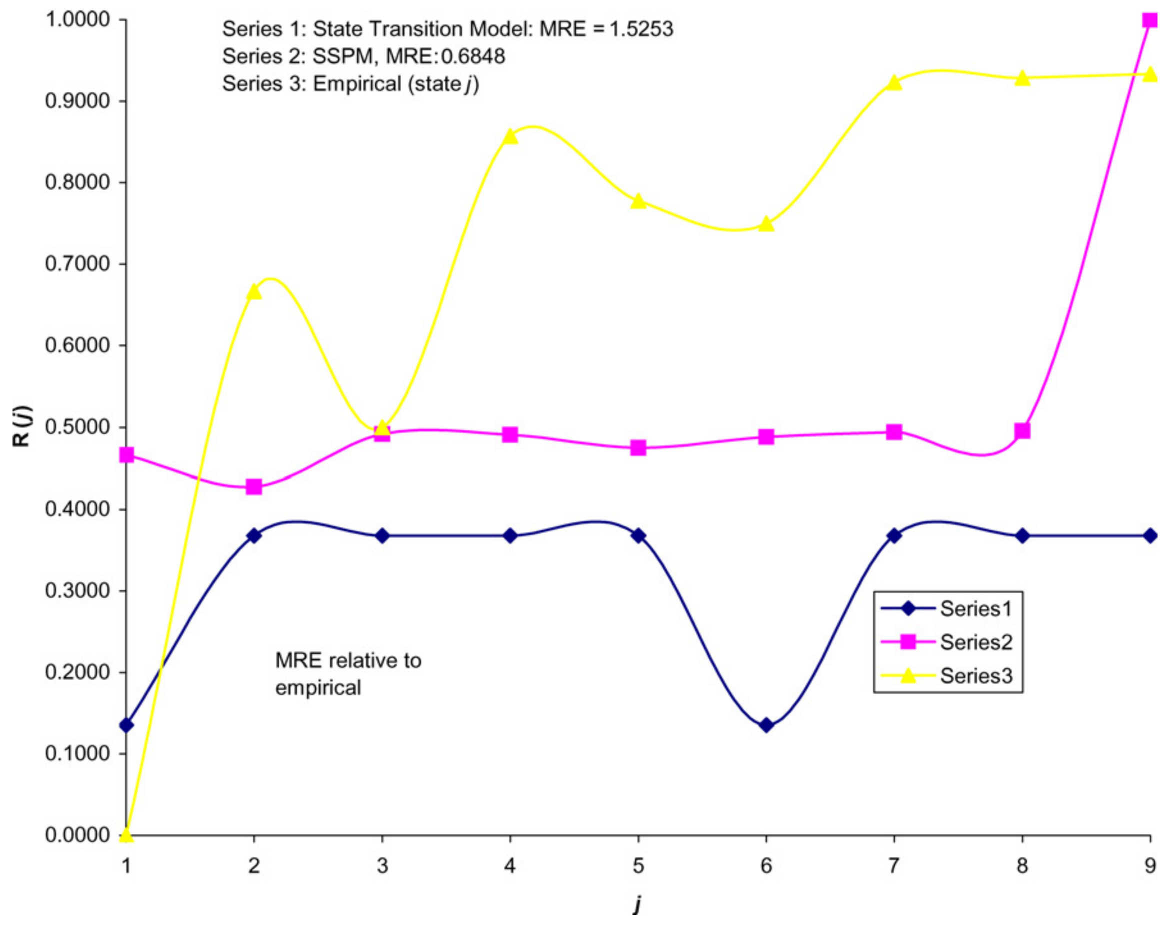


Fig. 3 NASA Space Shuttle OI6 reliability $R(j)$ vs. state j.

Now while predicting the number of failures in a given state and at a given time is important, it is crucial to predict the totals of these quantities. This we do in Eqs. (25) and (26), calling upon Eqs. (19) and (20, respectively):

$$\mathrm{r_r}(i,t) = \sum_{i,t} \mathrm{n_f}(i,t) \tag{25}$$

$$\mathrm{r_r}[(i-1),(t-1)] = \sum_{[(i-1),(t-1)]} \mathrm{n_f}[(i-1),(t-1)] \tag{26}$$

Then, based on Eqs. (27) and (28), we can compute revised empirical reliabilities, reflecting the condition of failure correction, as follows:

$$\mathrm{R_{er}}(i,t) = 1 - \frac{\mathrm{n_f}(i,t)}{\sum_{i,t} \mathrm{n_f}(i,t)} \tag{27}$$

$$\mathrm{R_{er}}[(i-1),(t-1)] = 1 - \frac{\mathrm{n_f}[(i-1),(t-1)]}{\sum_{i,t} \mathrm{n_f}[(i-1),(t-1)]} \tag{28}$$

2. *SSPM*

Also to be considered are the changes in reliability in SSPM, based on correcting failures, as given by equation (29), using the revised mean number of failures $^{m}_{r}(i,t)$ that is computed from revised parameter estimates for

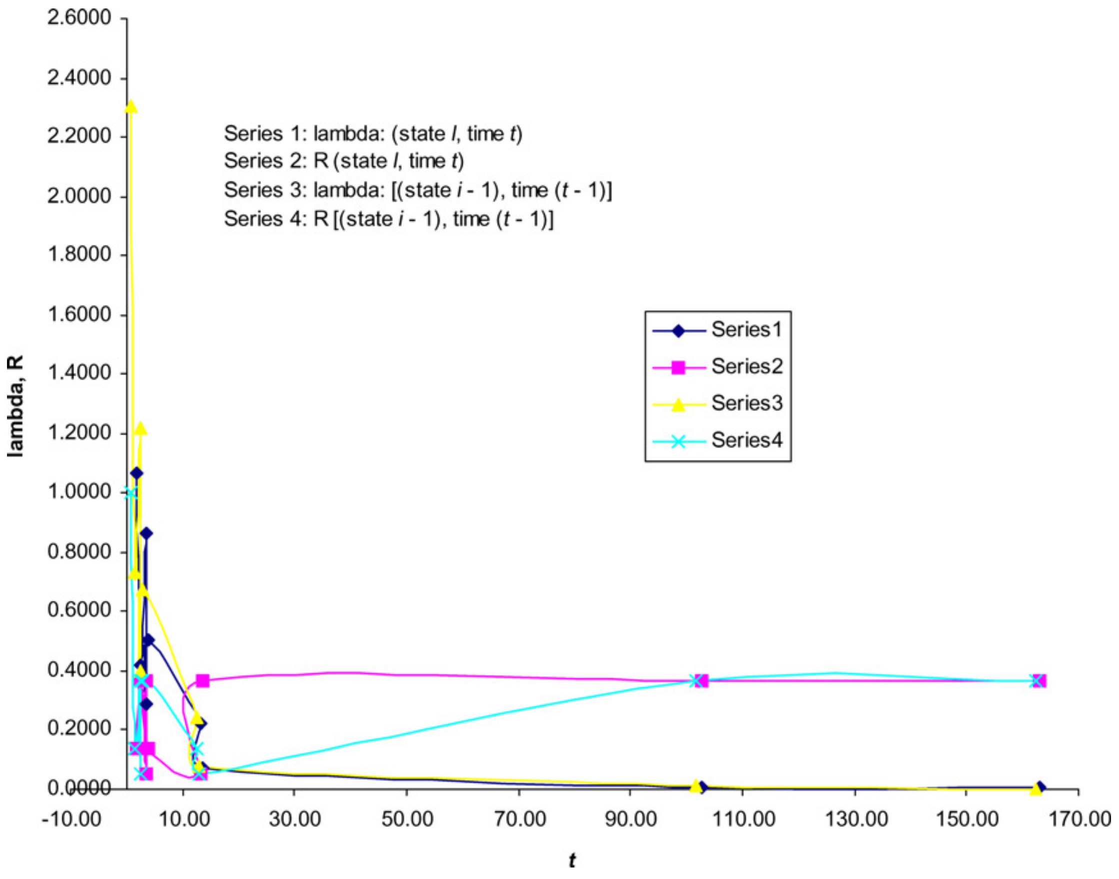


Fig. 4 NASA Space Shuttle OI6 state transition rate (lambda) and reliability R vs. time *t*.

SSPM:

$$\mathrm{R_s}(i,t) = \mathrm{e}_{\mathrm{r}}^{-\mathrm{m}}(i,t) \tag{29}$$

A composite picture of the improvements engendered by correcting failures is displayed in Fig. 5 for both STM and SSPM. Although neither model predicts with extreme accuracy, SSPM provides more accurate predictions, overall.

Now we want to predict remaining failures for SSPM[10] and compare it with the result from STM. This we do in equation (30):

$$r_s(i,t) = \left(\frac{1}{\beta}\right)\mathrm{e}^{-\beta * (i,t)} \tag{30}$$

Figure 6 shows how an analyst an analyst could use the STM to trace changes in reliability state as a function of failures corrected. Comparing Fig. 6 with Fig. 1, we see that, of course, Fig. 6 represents higher reliabilities, but the *incremental gains* in reliability generated by the transition is less because, as reliability increases, we reach diminishing returns in improvement.

C. Remaining Failures Prediction Assessment

Now then, to see how accurate our predictions of remaining failures are, we must compute the empirical remaining failures to have a basis of comparison. Assuming that failures discovered in state i at time t are corrected, we have the computation in Eq. (31):

$$\mathrm{r_e}(i,t) = \left[\sum_{i,t} \mathrm{x}(i,t)\right] - \mathrm{x}(i,t) \tag{31}$$

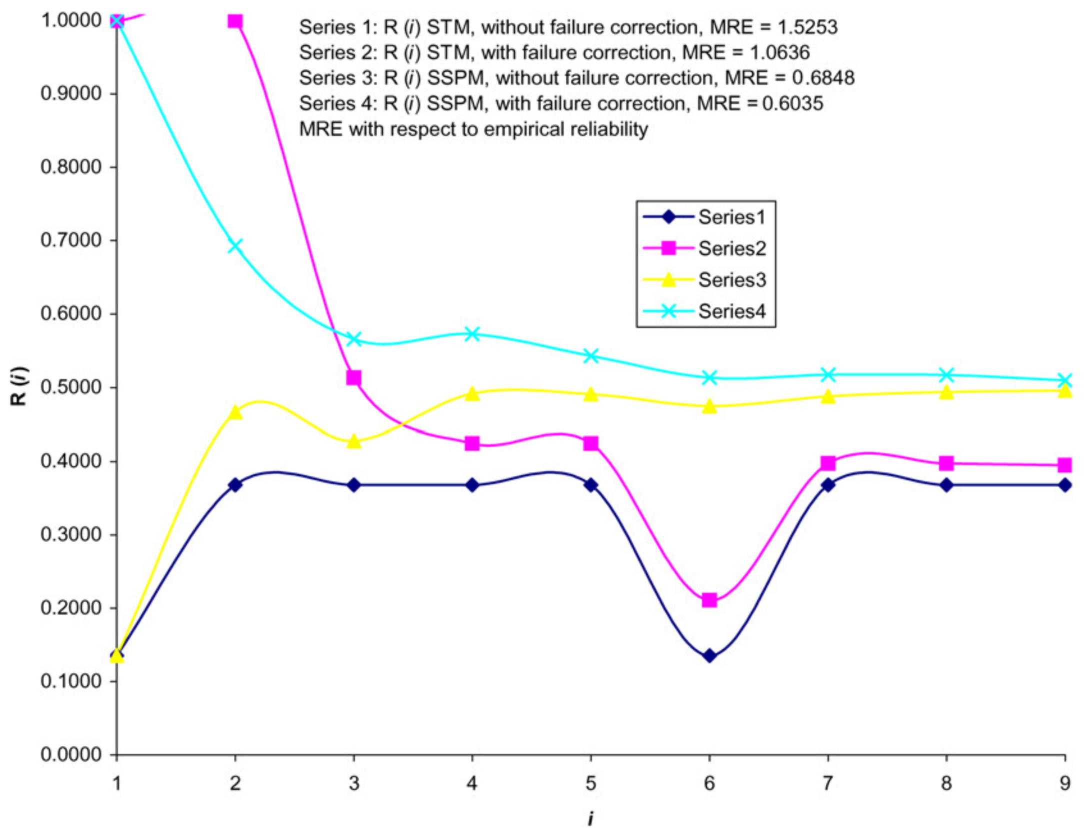
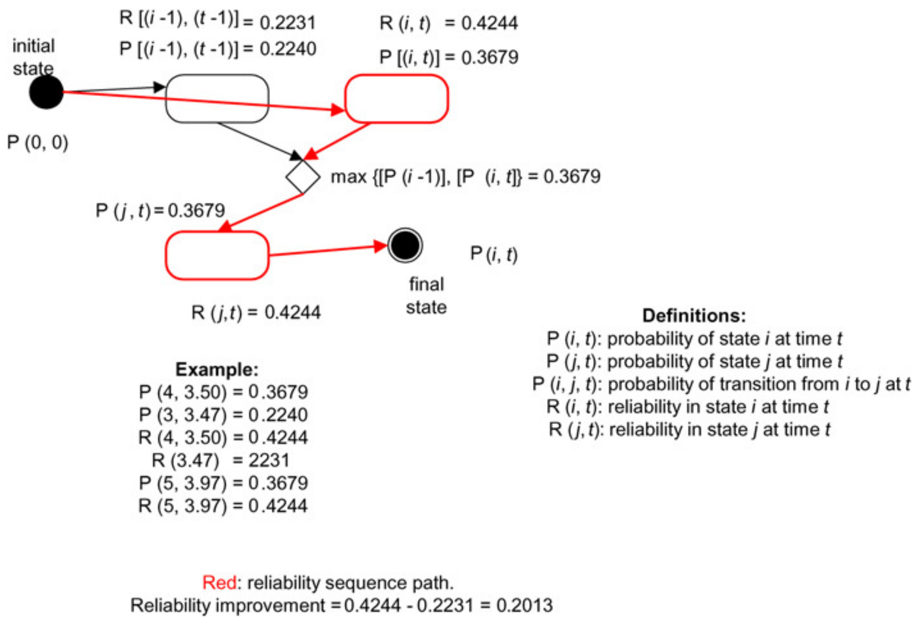


Fig. 5 NASA Space Shuttle reliability OI6 $R(i)$ before and after failure correction vs. reliability state i.



Definitions:
P (i, t): probability of state i at time t
P (j, t): probability of state j at time t
P (i, j, t): probability of transition from i to j at t
R (i, t): reliability in state i at time t
R (j, t): reliability in state j at time t

Example:
P (4, 3.50) = 0.3679
P (3, 3.47) = 0.2240
R (4, 3.50) = 0.4244
R (3.47) = 2231
P (5, 3.97) = 0.3679
R (5, 3.97) = 0.4244

Red: reliability sequence path.
Reliability improvement = 0.4244 - 0.2231 = 0.2013

Fig. 6 Revised software reliability state transition diagram (based on failure correction).

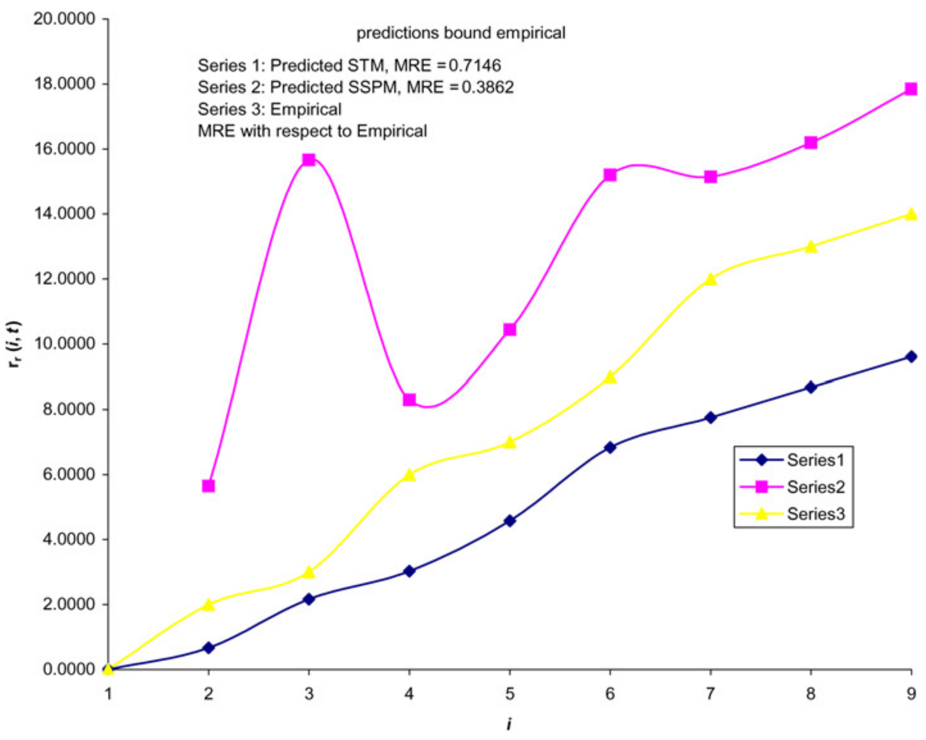


Fig. 7 NASA Space Shuttle OI6 remaining failures $r_r(i, t)$ in state i at time t vs. state i.

Note that in computing Eq. (31), it is not possible to know the *true* total number of failures that have occurred in the program until testing is complete (i.e., all the states and transition times have been explored). Thus, necessarily, we are limited to computing remaining failures based on the total failures observed *upto* a given state and time.

In Fig. 7 we show that the STM and SSPM predictions of remaining failures bound the empirical plot, with SSPM providing the greater accuracy. Thus, an organization could use this technique to provide limits on remaining failures, with confidence that the actual values would lie between the two plots.

VIII. Model Summaries

The prediction error rates of STM and SSPM are summarized in Table 1. We see that in response to our research objective, STM does *not* provide a superior model, based on the plots and Table 1. However, we caution that up to this point, results were obtained using only a single Shuttle operational increment. Now, we will obtain results using a second release, OI7, and compare results. Figure 8 shows that SSPM still predicts more accurately, but neither model shows reliability growth, as in the case of empirical reliability. We also note in comparing MRE in Table 1 with Fig. 8, that reliability has improved with OI7, which is what we would expect with a subsequent release.

In Fig. 9, we again observe the superiority of SSPM predictions, but we cannot claim that the accuracy is great relative to the empirical remaining failures. However, as in the case of OI6, the plot is useful it that the prediction provide a bounds on the actual values.

Frankly, we are puzzled as to why STM does not do better. It is based on the assumption of transition events being governed by the Poisson distribution. This seems to be a reasonable assumption, but other models may be more appropriate. This is a ripe area for further research.

1. SSRM

Since neither model projects reliability growth (i.e., reliability increases with state and time), what is needed is a reliability *growth* model! We now provide that model — the Schneidewind Software Reliability Model (SSRM). We justify its use by virtue of the fact that it has been used to predict reliability for the Shuttle flight software.[8] In order to

Table 1 Summary of model prediction error rates

Model	Prediction	State	Time	MRE
STM (OI6)	Reliability (No failure correction)	i	t	3.9518
SSPM (OI6)	Reliability (No failure correction)	i	t	0.6848
STM (OI6)	Reliability (with failure correction)	i	t	1.5253
SSPM (OI6)	Reliability (with failure correction)	j	t	0.6035
SSRM (OI6) ($s = 6$)	Reliability (No failure correction)	i	t	**0.1356**
STM (OI7)	Reliability (No failure correction)	i	t	1.0512
SSPM (OI7)	Reliability (No failure correction)	i	t	0.3945
SSRM (OI7) ($s = 6$)	Reliability (No failure correction)	i	t	**0.1070**
STM (OI6)	Remaining Failures	i	t	0.7146
SSPM (OI6)	Remaining Failures	i	t	**0.3862**

compare its predictions with those of STM and SSPM, we make SSRM reliability predictions using equation (32):[8]

$$\mathrm{R}(t) = \mathrm{e}^{-[(\alpha/\beta)[\mathrm{e}^{-\beta(t-s)} - \mathrm{e}^{-\beta(t-s+1)}]]} \tag{32}$$

Figure 10 shows that, indeed, SSRM demonstrates reliability growth for OI 6 and that its error rates are less than the other two models. In addition, we show a typical Shuttle mission duration of 8 days. Since the SSRM displays reliability growth, its predictions would be comforting in the mission duration range. SSRM parameters (see Definitions) were estimated for two cases: 1) nine intervals of failure counts and 2) six intervals of failure counts. As we would suspect, the former provided greater prediction accuracy.

Again, in Fig. 11, we see that SSRM shows reliability growth and, again, shows superior prediction accuracy compared with the other two models. As summarized in Table 1, the evidence is definitive that SSRM is the best choice, at least for the Shuttle releases OI6 and OI7.

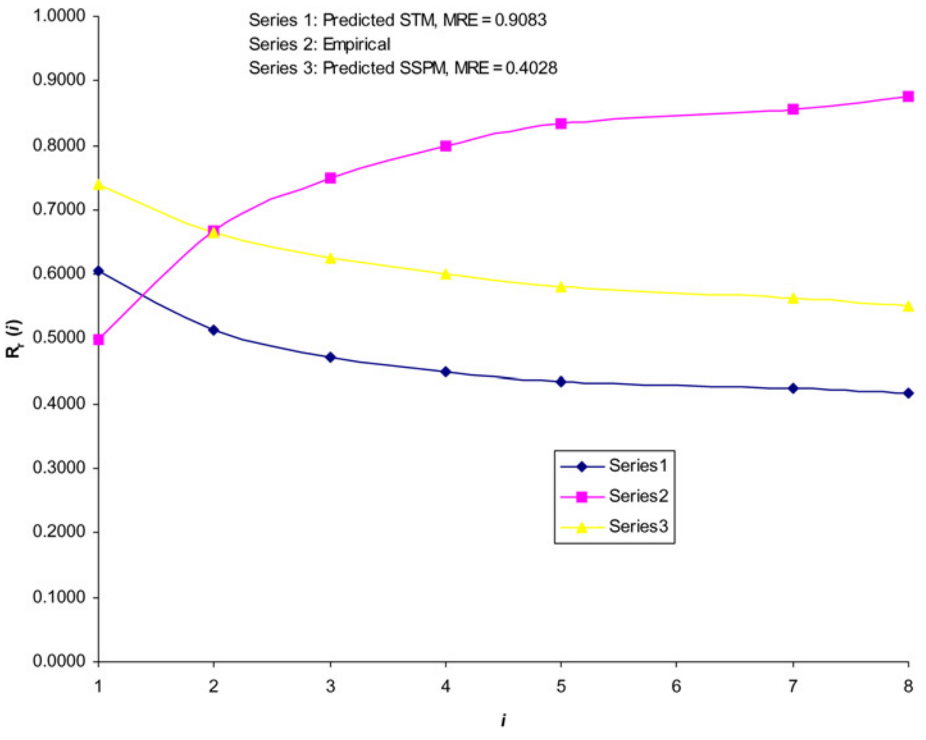


Fig. 8 NASA Space Shuttle OI7 Reliability R_r (i), after correcting failures, vs. State i.

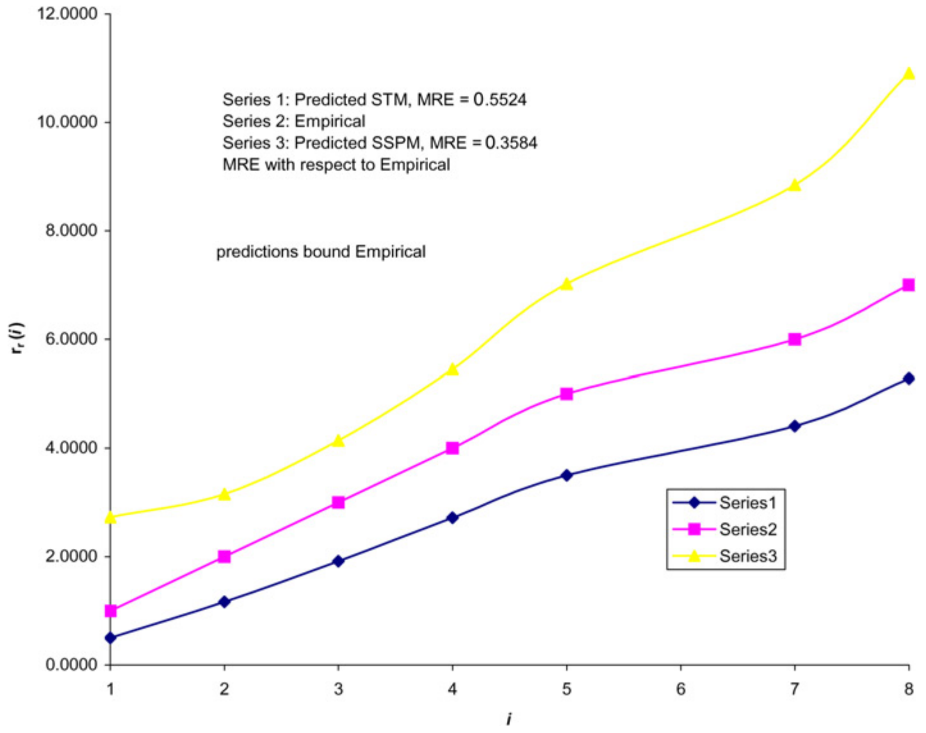


Fig. 9 NASA Space Shuttle OI7 remaining failures r_r (i) vs. state i.

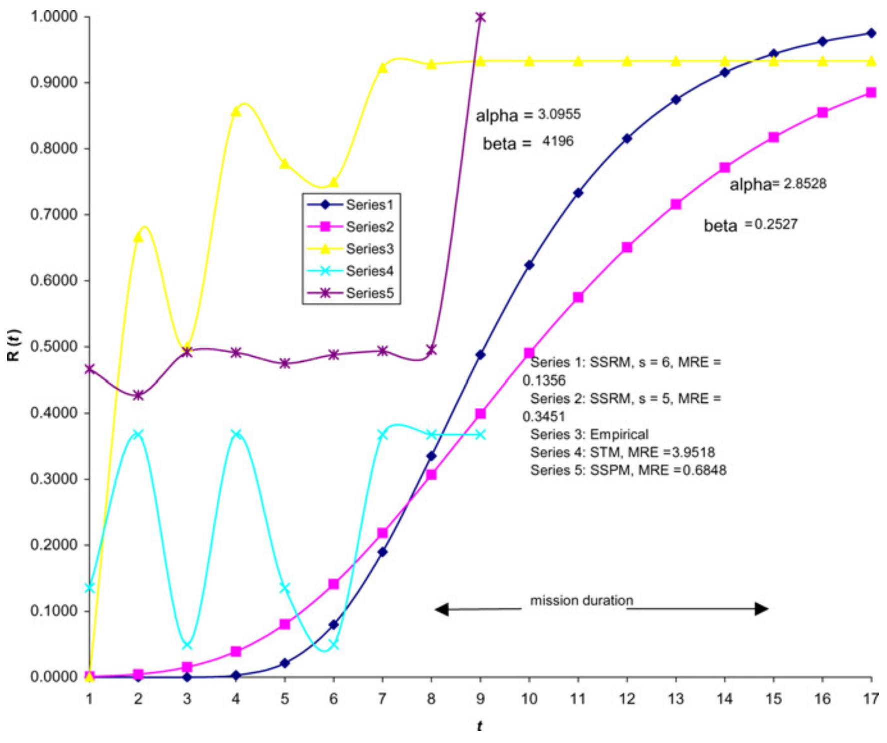


Fig. 10 NASA Space Shuttle OI6 reliability R(t) vs. time t.

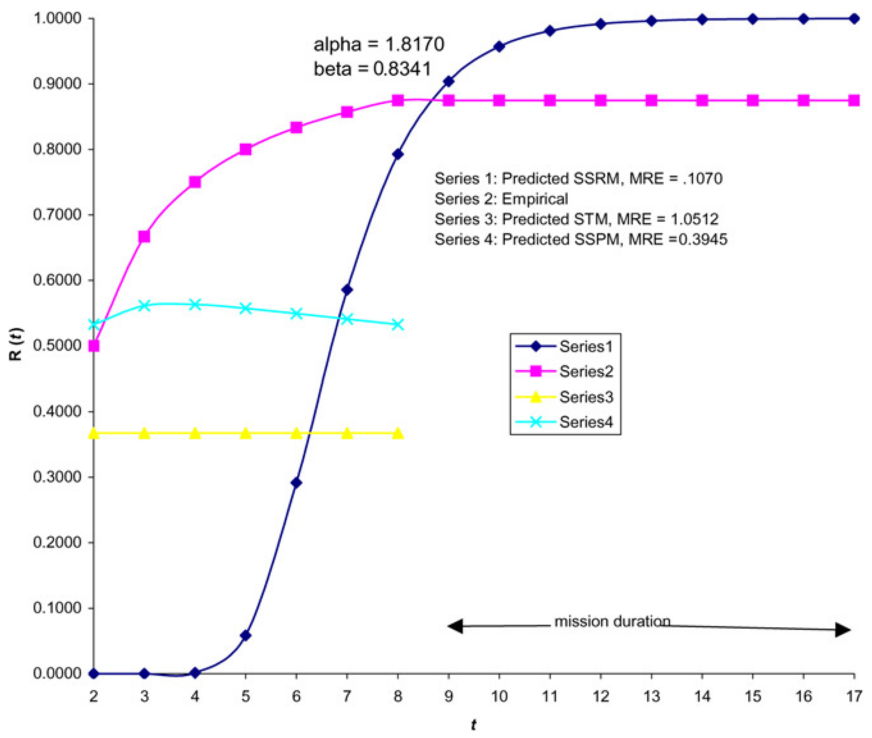


Fig. 11 NASA Space Shuttle OI7 reliability R(t) vs. time t.

IX. Conclusions

Based on the evidence (i.e., plots and Table 1), STM did not prove to be a viable model for predicting Shuttle reliability meters. This was a disappointment because at the outset it seemed to be a reasonable approach, but the counter intuitive result is the benefit of doing research! SSPM was not much better. Only SSRM provided acceptable (i.e., MRE $<$ 15%). The reason for the discrepancy in prediction accuracy is attributed to the fact that SSRM is based on the non-homogeneous Poisson failure occurrence process,[8] whereas STM and SSPM are governed by a homogeneous failure process.[10] A non-homogeneous process accounts for more of the variability in the failure data and, therefore, models based on it, can provide more accurate predictions. It is possible that STM—supported by a different failure process—could perform better. The IEEE Draft Standard for Software Reliability Prediction provides descriptions of a number of reliability models that practitioners and researches could experiment with to further investigate the potential of STM.[5]

References

[1]Bronson, R., and Naadimuthu, G., *Operations Research*, Second Edition, McGraw-Hill, 1997.

[2]Joao W. Cangussu, Raymond A. DeCarlo, Aditya P. Mathur, “Using Sensitivity Analysis to Validate a State Variable Model of the Software Test Process,” *IEEE Transactions on Software Engineering*, Vol. 29, No. 5, May 2003, pp. 430–443.

[3]Ibrahim K. El-Far and James A. Whittaker, Florida Institute of Technology, “Model-based Software Testing”, *Encyclopedia on Software Engineering*, edited by J.J. Marciniak, Wiley, 2001.

[4]Hong, S., Hwang, J., Kanp, B., Lee, M., Kim, H., and Kim, Y., “An Empirical Study on Reliability Evaluation for a Real Time Switching System,” Software Testing, Reliability and Quality Assurance, 1994. *Conference Proceedings*, Electronics and Telecommunications Research Institute, Taejon, Korea, Publication Date: 21–22 December 1994, page(s): 11–15, Meeting Date: 12/21/1994–12/22/1994, New Delhi, India.

[5]IEEE/AIAA P1633$^{\text{TM}}$/Draft 7, Draft Standard for Software Reliability Prediction, Prepared by the Software Reliability Engineering Working Group of the Definitions and Standards Committee of the Reliability Society, May 2007.

[6]*Handbook of Software Reliability Engineering*, edited by Michael R. Lyu, Published by IEEE Computer Society Press and McGraw-Hill Book Company, 1996.

[7]Nakajima, T., Bessho, Y., Yamanaka, H., Hirota, K., and Works, K., "Automatic Testing of Embedded Software Based on State-transition Requirement Specifications," *Electronics and Communications in Japan (Part II: Electronics)*, Vol. 86, No. 9, 18 August 2003, Wiley Periodicals, Inc. pp. 64–75.

[8]Norman F. Schneidewind, "Reliability Modeling for Safety Critical Software," *IEEE Transactions on Reliability*, Vol. 46, No.1, March 1997, pp.88–98.

[9]Norman F. Schneidewind, "Measuring and Evaluating Maintenance Process Using Reliability, Risk, and Test Metrics", *IEEE Transactions on Software Engineering*, Vol. 25, No. 6, November/December 1999, pp. 768–781.

[10]Norman F. Schneidewind, "A New Software Reliability Model", *The R & M Engineering Journal, American Society for Quality*, September 2006, Vol. 26, No. 3, pp. 6–22.

[11]Jeff Tian, "Integrating Time Domain and Input Domain Analyses of Software Reliability Using Tree-Based Models," *IEEE Transactions On Software Engineering*, Vol. 21, No. 12, December 1995, pp. 945–958.

[12]Wang, W.-L., Wu, Y., and Chen, M.-H., "An Architecture-Based Software Reliability Model," *Pacific Rim International Symposium on Dependable Computing*, prdc, p. 143, 1999.