IMPROVEMENTS IN ESTIMATING SOFTWARE RELIABILITY FROM GROWTH TEST DATA

David J. Dwyer, 99G* and Paul D'Onofrio, 10G**
BAE Systems, Inc., Nashua, NH

Key Words: *Duane, Musa Basic Model, Musa-Okumoto Logarithmic Law, reliability growth, software reliability*

SUMMARY & CONCLUSIONS

John Musa's first book on Software Reliability Engineering [1] advises the analyst to use the Musa Basic Law or the Musa-Okumoto Logarithmic Law to estimate failure intensity, depending on which provides the best fit to the data. He refered to the papers by Tractenberg [2] and Downs [3] as providing a foundation for these models. We propose using Musa's basic model in combination with an approach Duane [4] and Codier [5] used to estimate failure intensity (aka instantaneous failure rate) for software. Musa noted in his last book [6] that the Basic law is optimistic in estimating residual errors and that the logarithmic law is pessimistic because it implies infinite errors.

There is currently a problem in deciding which law to use for software, Basic or Logarithmic, other than which has a higher correlation coefficient. We recommend that Musa's basic law be used but that the line be drawn using a method that reflects the cumulative nature of the statistics. This approach is based on both Downs' paper and that presented by E. O. Codier at the 1968 Annual Symposium on Reliability which described how to draw the line for Duane growth plots. Codier argued that when reliability growth data is plotted:

- "The latter points, having more information content, must be given more weight than earlier points",
- "The normal curve fitting procedure of drawing the line through the 'center of gravity' of all the points should not be used." and
- "Unless the data is exceptionally noisy, the best procedure is to start the line on the last data point and seek the region of highest density of points to the left of it."

With regard to Musa basic plots, the region of highest density would be to the right of the last point, not to the left of it. It should be noted here that the IEEE Recommended Practice on Software Reliability [7] for the application of Duane states that "Least squares estimates for *a* and *b* of the straight line on log–log paper can be derived using recommended practice linear regression estimates". But Codier's recommendations (above) have been shown [8] to result in a more accurate measure of MTBF for hardware. This would be just as true for the application of Duane growth plots for software as for hardware. This paper shows even greater improvements when Codier's methods are applied to Musa's Basic model for Software.

ISSN 1559-9388 (online version), ISSN 1559-9396 (CD-ROM version).

The early paper by Thomas Downs referred to the curved lines that are fitted to operational profile data as "convex", not logarithmic, because there is no firm basis for calling the distribution logarithmic. There are examples in physics of exponential decay, as in the half life of a radioactive element, that follow a logarithmic curve, but no such mechanism exists to justify fitting a log function to software test data. The recommended method avoids defining the optimistic and pessimistic extremes of the curves as linear, logarithmic or any other specific shape. It simply draws a line that follows the changing slope of the points naturally as originally proposed by Codier. This paper develops a methodology for calculating failure intensity from the slope of the resulting line and from the cumulative failure rate at the final data point. It avoids the optimism of the Basic law and the pessimism of the Logarithmic law as well as the decision of which to use.

1 INTRODUCTION

1.1 Background

Downs' paper showed what happens when failure rate is plotted against failure number for two software operations tested at different rates. The result is a curve with two distinct segments, differing in slope. This implies that the number of operations will determine the number of segments, that the number is never infinite and so the line not logarithmic. But then, what is it? It is segmented and, at the point that reliability growth testing stops, the plot is a straight line. Downs also identified other sources of nonlinearity.

This paper describes a spreadsheet method for applying Codier's recommendations to software reliability growth data that will mitigate the errors from the Basic or Logarithmic model. This method automatically weighs the cumulative points (each successive point weighing more) up to, but not including the last one, calculates their center of gravity, draws a line from the center-of-gravity point through the last point, and calculates an instantaneous MTBF.

Applying the conventions of Tractenberg and Downs, the following notations are used:

λ_i	– failure intensity or instantaneous failure rate
λ_{cum}	– cumulative failure rate after some number of faults, 'j' are detected
j	– the number of faults [aka "error sites"] removed by time 'T'
T	– test time during which 'j' faults occur
Φ	– constant of proportionality between λ and j
Pr	– probability'
c	– number of paths affected by a fault
M	– total number of paths.

2 RELIABILITY GROWTH MODEL DEVELOPMENT

2. 1 Linear Test Methodology

Tractenberg simulated software with errors spaced throughout the code in six different patterns and tested this simulation in four different ways. He defined the following fundamental terms:

- an “error site” is a mistake made in the requirements, design or coding of software which, if executed, results in undesirable processing.
- “error rate” is the rate of detecting new error sites during system testing or operation.
- “uniform testing” is a condition wherein, during equal periods of testing, every instruction in a software system is tested a constant amount and has the same probability of being tested.

The results of his simulation testing showed that the error rates were linearly proportional to the number of remaining error sites when all error sites have an equal detection probability. The result would be a plot of failure rate that would decrease linearly as errors were corrected. An example of non-linear testing examined by Tractenberg was the common practice of function testing wherein each function is exhaustively tested, one at a time. Another non-linear method he examined was testing to the operational profile, or “biased testing”. The resulting plot of function testing is an error rate that is flat over time (or executions). With regard to the use of Musa’s linear model where the testing was to the operational profile, Tractenberg stated:

> “As for the applicability of the linear model to operational environments, these simulation results indicate that the model can be used (linear correlation coefficient > 0.90) where the least used functions in a system are run at least 25% as often as the most used functions.”

2.2 Effects of Biased Testing and Fault Density

Downs investigated the issues associated with random vs. biased testing. His approach assumes the following:

The execution of software takes the form of execution of a sequence of paths.

- ‘*c*’, the actual number of paths affected by an arbitrary fault, is unknown and can be treated as a random variable.
- Not all paths are equally likely to be executed in a randomly selected execution [operational] profile.

Downs said that very little is known about the nature of the random variable of the number of paths affected by an arbitrary fault. Downs stated:

- In the operational phase of many large software systems, some sections of code are executed much more frequently than others are. In addition,
- Faults located in heavily used sections of code are much more likely to be detected early.

If a plot of failure rate is made vs. the number of faults, a “convex” curve should be obtained. This paper is concerned with the problem of estimating the reliability of software during a structured test and then predicting what it would be during actual use. To do this, the testing must:

- resemble the way the software will finally be used, and
- the predictions must include the effects of all corrective actions implemented, not just be a cumulative measure of test results.

The execution profile defines the probabilities with which individual paths are selected for execution. This study will assume the following:

- The input data that is supplied to the software system is governed by an execution profile which remains invariant in the intervals between fault removals
- The number of paths affected by each fault is a constant.

- Downs showed that the error introduced by the second approximation is insignificant in most real software systems.

Downs derived the following Lemma:

> "If the execution profile of a software system is invariant in the intervals between fault removals", then the software failure rate in any such interval is given by

$$\lambda = -r \log[\Pr\{\text{a path selected for execution is fault free}\}] \qquad (1)$$

where r = the number of paths executed over unit time.

Downs then shows that for software containing N initial faults, (randomly distributed over M logic paths), the probability that an arbitrarily selected path is fault free is $(1 - c/M)^N$. Then the initial failure rate, λ_0, is given by:

$$\lambda_0 = -Nr\,\mathbf{log}(1 - c/M) \qquad (2)$$

Removal of faults from the software affects the distribution of the number of faults in a path. The manner in which this distribution is affected depends upon the way in which faults are removed from paths containing more than one fault. If, for instance, it is assumed that execution of a path containing more than one fault leads to the detection and removal of only one fault, then the distribution of the number of faults in a path will cease to be binomial after the removal of the first fault. This is because under such an assumption, considering that all paths are equally likely to be selected, those faults occurring in paths containing more than one fault are less likely to be eliminated than those occurring in paths containing one fault only. If, on the other hand, it is assumed that execution of a path containing more than one fault leads to detection and removal of all faults in that path, then all faults have an equal likelihood of removal and the distribution of the number of faults in a path will remain binomial.

Fortunately, in relation to software systems which are large enough for models of the type Downs developed, the discussion contained in the above paragraph has little relevance. This follows from the fact that, in large software systems, the number of logic paths, M, is an extremely large number, implying that the parameter c/M is a very small number. This in turn implies that the number of paths containing more than one fault is a very small fraction of M. Consequently, even if it cannot be assumed that all faults are removed from each path whose execution leads to system failure, the assumption that each fault has equal probability of removal will lead to very little error. Hence, in such cases, a very good approximation is obtained by assuming that the distribution of the number of faults in a path remains binomial as faults are removed.

Downs showed that after the removal of j faults, the failure rate is:

$$\lambda_j = (N - j)\mathbf{\Phi}, \qquad (3)$$

where j = the number of corrected faults, and

$$\mathbf{\Phi} = -r\,\mathbf{log}(1 - c/M) \qquad (4)$$

Downs inserted errors in software then tested two portions of the software at different rates, simulating two "operations". The resulting plot was not logarithmic, but showed two segments with distinct slopes, one for each of the two operations. Real world software has many more than two, but never an infinite number of operations. The curve from Downs' paper is shown in Figure 1.

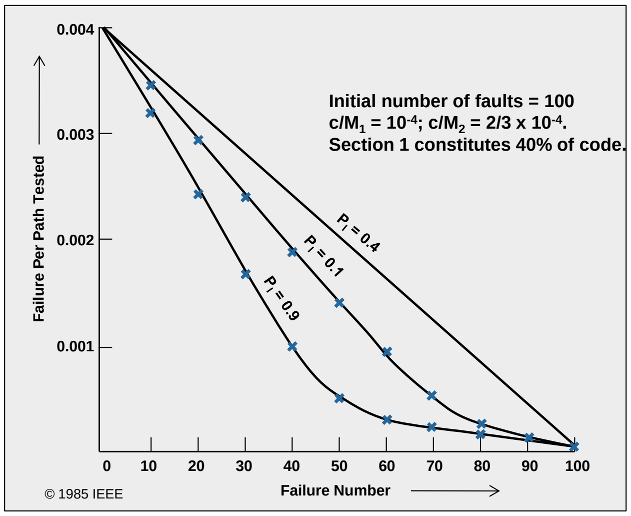

Figure 1. Failure rate plotted against failure number for a range of non-uniform testing profiles.

2.3 Recommended Approach

It can be seen from the forgoing discussion that there are several sources of nonlinearity in software reliability test plots, the most obvious being the operating profile. And it is obvious that the practice of having to decide whether to use a linear model or a logarithmic model is flawed. Furthermore, there is never an infinite number of remaining errors! What if the correlation coefficient is the same for both Musa Basic and Logarithmic? What is the real underlying distribution shape?

We should note that the last plotted data point includes all of the information up to that time. Our approach must be able to calculate current reliability based on the cumulative reliability at the last point and account for the non-linearities up to that point. These plotting criteria were described by E. O. Codier, in his paper for hardware reliability growth presented at the 1968 Annual Symposium on Reliability as listed at the beginning of this paper.

There were three parameters of interest for Codier: (λ_c, ΣF and ΣH), where λ_c is cumulative failure rate. λ_I is instantaneous failure rate, now commonly referred to as failure intensity and is calculated from the first three parameters. The measure for cumulative failure rate is Σ(Failures)/Σ(Hours), and is calculated at the last point on the graph, which includes the most and latest (most 'fixes' incorporated) data point.

Failure intensity, λ_i, is defined as the time derivative of 'F'. The following shows Codier's derivation for failure intensity for hardware which also applies to software:

$$\begin{aligned}
\lambda_c &= F / T \\
&= kT^{(-m)} \\
F &= kT^{(1-m)} \\
\lambda_i &= \partial F / \partial T \quad \text{//(failure intensity)} \\
&= k(1-m)T^{(-m)} \\
\lambda_i &= (1-m)\lambda_c
\end{aligned}$$

It should be noted that the use of Duane for software reliability estimates should also follow Codier's guidelines for drawing line, except that we want to plot cumulative failure rate vs failure number. This allows us to estimate the total number of errors remaining in the software. For this, we use Downs' formula (3) where the subscript '*j*' represents failure number and is substituted for Codier's '*c*' to avoid confusion with Down's '*c*' (number of paths affected by a fault):

$$\lambda_j = \Sigma j / \Sigma T \quad \text{//cumulative failure rate}$$

$$= (N - j)\Phi \text{ //from (3)}$$

$$j = T(N - j)\Phi \qquad (5)$$

$$\lambda_i = \partial j / \partial T \text{ // failure intensity} \quad (6)$$

$$= (N - j)\Phi + T(-\partial j / \partial T)\Phi \qquad (7)$$

From (3):

$$\lambda_i = \lambda j + T(-\lambda_i)\Phi \qquad (8)$$

$$\lambda_i(1 + T\Phi) = \lambda_j \quad \text{//collecting variables}$$

$$\lambda_i = \lambda_j / (1 + T\Phi) \qquad (9)$$

As we stated initially, we must draw the growth line through the last point and through the weighted center of gravity of the remainder of the points, not by drawing a least squares fit through all the points. This applies to both Downs' plots and to the proposed modified Musa Basic plot.

These criteria, along with the equations above, constitute our method to satisfy our requirements. We will literally find the 'center of gravity' by assigning 'weight' (='*n*' for the *n*-th point) to the points (except for the last one), each successive point having more weight than the one before it. We will then draw a line through the center of gravity of those weighted points and through the last point. When we have done that, we will have complied with Codier's guidelines for drawing a line through cumulative data but applied them to software reliability calculations.

2.4 Comparison of Least Squares Fit, logarithmic fit and the Effects of Weighing the Latter Points More

We next want to compare three methods, linear fit using least squares, logarithmic fit and Codier's approach. Table 1 shows some actual software reliability test data. This table shows failure times and weights assigned based on failure number. These "weights" will be used to calculate the center of gravity of all but the last point in a way similar to mechanical center of gravity calculations.

Figure 2 shows a typical least squares fit to the data from Table 1 where failure number is plotted against cumulative failure rate along with the recommended fit based on Codier's recommendations. Notice that the least squares fit would predict 5.88 errors total, but seven failures have already occurred. It should be noted that this least squares fit is also the recommended approach for the application of Duane plots to software reliability data in the IEEE Std 1633, reference 7, and this paper takes issue with that.

Failure Number =	Failure Time	Cumulative Failure Rate
Weight = j	= t	λc = j/t
1	56	0.01785714
2	224	0.00892857
3	560	0.00535714
4	1400	0.00285714
5	2072	0.00241313
6	2240	0.00267857
7	4424	0.00158228

Table 1. Software Reliability Test Data.

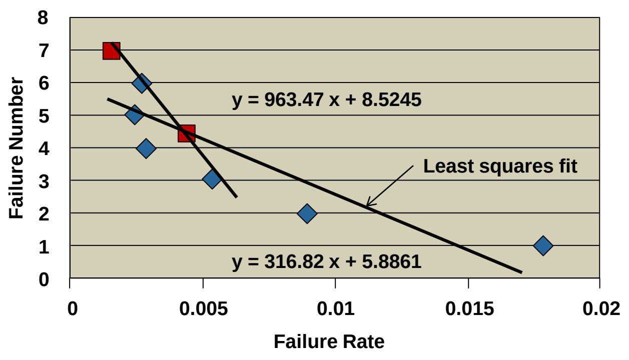

Figure 2. Comparison of Least Squares fit with recommended method.

A similar problem happens when a logarithmic curve is fitted to the same data. This is shown in Figure 3. The result of that would be an estimate for an infinite number of errors!

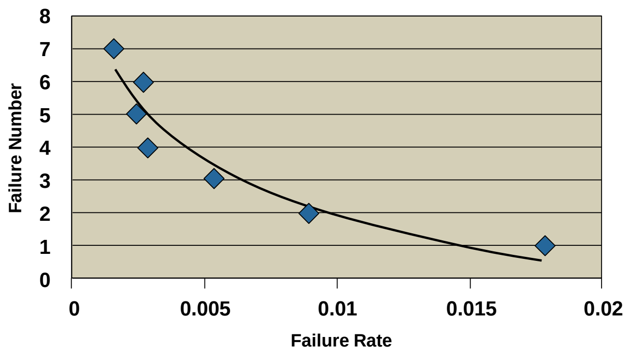

Figure 3. Logarithmic fit.

The method for calculating failure intensity was described in the derivation previously shown and depends only on the slope of the line ($1/\Phi$), the cumulative failure rate at the last point and the time "*T*" during which the "*j*" failures occur.

The coordinates for the weighted center of gravity of the first six points is shown below, along with that for the last point. These two points define the growth line. Table 2 shows the calculation method for weighing the points.

Weight = j	j x λc	j x j
1	0.017857	1
2	0.017857	4
3	0.016071	9
4	0.011429	16
5	0.012066	25
6	0.016071	36
7		
Σ (1-6) = 21	0.091351	91

Table 2. Calculation Method.

Table 3 shows the calculation of the weighted center of gravity of the first six points along with the location for the last point. These two points define the recommended growth line.

CGx =	0.09135/21 =	0.00435
CGy =	91/21 =	4.33333
Final x =		0.00158
Final y =		7

Table 3. Calculation of Center of Gravity.

This is a modification to the common process of making a least-squares fit through all of the points. For example, the normal curve fitting procedure of drawing the line through the "center of gravity" of all the points should not be used. Dhillon [9] describes one way to do a weighted fit of the points, although he does not go through the last point. Dhillon describes the least squares fit, aka, “trend line”, and the “weighted” least squares fit: [3, p150].

“If the plotted points are not independent, then applying a weight to the cumulative number of failures at each point is a reasonable way to improve accuracy of these estimates. This technique assigns greater weight to the most recent data point. This method is based on the assumption that each data point is plotted m number of times at that point.”

It can be seen from the graph that the y intercept is 8.5245, so there are (8.5245 - 7) = 1.5245 errors remaining. We could fit a log plot through all points, (except for the weighted CG), but that would imply infinite errors remaining! We could fit a trend line through all the points (except the weighted CG) but that would not reflect the trend in the later points. A least squares trend-line through the points has an intercept at y = 5.8861, meaning there would be -1.1 errors left in the software! Only the method we describe here provides reasonable results.

3. STUDY RESULTS

This paper shows that by implementing Codier’s approach to line drawing for software reliability test data, that we can avoid deciding whether a log plot or a linear plot should be chosen. We can draw the line through the last point and the center of density of the remaining points and have one approach that works whether the data seems to be a straight line or a “convex” curve. For software reliability test, significant error is avoided when Codier’s original recommendations for line drawing for reliability growth are implemented when using Musa’s Basic Law.

REFERENCES

[1] John D. Musa, Anthony Iannino, Kazuhira Okumoto, “Software Reliability,” McGraw-Hill, 1987

[2] Martin Trachtenberg, “The Linear Software Reliability Model and Uniform Testing,” IEEE Transactions on Reliability, 1985, pp 8-16.

[3] Thomas Downs, “An Approach to the Modeling of Software Testing with Some Applications,” IEEE Transactions on Software Engineering, Vol. SE-11, No. 4, April, 1985, pp 375-38.

[4] J. T. Duane, “Learning Curve Approach to Reliability Monitoring”, *IEEE Transactions on Aerospace*, Volume 2 Number 2, April 1964

[5] Ernest O. Codier, “Reliability Growth in Real Life”, *Proceedings, 1968 Annual Symposium on Reliability*, New York, IEEE, January 1968, pp 458-469.

[6] John D. Musa, “Software Reliability Engineering: More Reliable Software Faster and Cheaper,” AuthorHouse, 2004

[7] IEEE Std 1633™-2008, “IEEE Recommended Practice on Software Reliability”

[8] Dave Dwyer, Ed Wolfe, Jonathan Cahill, “Improvements In Automated Reliability Growth Plotting”, *Annual Reliability and Maintainability Symposium*, January 2009

[9] Balbir S. Dhillon, “Reliability Engineering in Systems Design and Operation”, 1983, Van Nostrand Company Inc.

* **DAVID J. DWYER** is a Principal Software Engineer and has worked both as a reliability engineer and as a software engineer while at BAE Systems over the last 38 years. He is also an adjunct faculty member at Rivier College where he has taught a graduate course in Software Reliability as part of the Computer Science curriculum. Before working at BAE Systems, he was a flight instructor at the University of Illinois for 3 ½ years. He has a B.S. in Physics (Providence College, 1963), M.S. in Electrical Engineering (Northeastern University, 1980) and M.S. in Computer Science (Rivier

College, 1999). He has presented several papers on Software and Hardware Reliability including “Improvements in Automated Reliability Growth Plotting and Estimations” (2009 RAMS Symposium), “Software Reliability Estimations/Projections, Cumulative & Instantaneous” (2004 RAMS Symposium), “Reliability Test Planning for One Shot Systems” (1987 RAMS Symposium), “Hardware Reliability Growth Estimations and Projections – What is Valid and What is Not” (ASQ NEQC 56th Conference, October 18, 2006), and “Sweaty Days = Shorter Runways” (Private Pilot Magazine, June 1972).

** **PAUL D’ONOFRIO** is the Reliability Engineering Section Manager for the Electronic Combat Solutions Business Area of BAE Systems Electronic, Intelligence & Support (EI&S) in Nashua, New Hampshire. He has been with BAE Systems EI&S for over 12 years. Paul holds a Bachelor of Science in Engineering (Boston University), and a Master of Science in Computer Science (Rivier College, 2010). Paul joined Sanders, a Lockheed Martin Company, in 1997, where he started as a Reliability, Maintainability, & Safety Engineer on various heritage Merlin programs, and later proceeded into Reliability Engineering leadership roles on the F-22 Electronic Warfare Production and Repair & Support Programs. Prior to his current role, Paul had been serving as a Reliability Product Assurance functional manager for the Electronic Warfare Systems Engineering Organization.