An Examination of Fault Exposure Ratio

Yashwant K. Malaiya, *Senior Member, IEEE,* Anneliese von Mayrhauser, *Member, IEEE,* and Pradip K. Srimani, *Senior Member, IEEE*

Abstract—**The fault exposure ratio K is an important factor that controls the per-fault hazard rate, and hence the effectiveness of testing of software. The paper examines the variations of K with fault density which declines with testing time. Because faults get harder to find, K should decline if testing is strictly random. However, it is shown that at lower fault densities K tends to increase; we explain this by using a hypothesis: real testing is more efficient than strictly random testing especially at the end of the test phase. Data sets from several different projects (in USA and Japan) are analyzed. If we combine the two factors, e.g., shift in the detectability profile and the nonrandomness of testing, then the analysis leads us to the logarithmic model which is known to have superior predictive capability.**

Index Terms—**Detectability, fault density, fault exposure ratio, testing, predictive capability, software reliability.**

I. Introduction

THE SOFTWARE reliability models are needed for measuring and projecting reliability. While some of the models like the recent neural network approach [1] are purely empirical, many of them are based on some specific assumptions about the fault detection/removal process [2]. The parameters of these models thus have some interpretations and thus possibly may be estimated using empirical relationships using static attributes. The two parameters of the exponential model [3] are the easiest to explain. Using this model the expected number of faults $\mu(t)$ detected in a duration t may be expressed as

$$\mu(t) = \beta_0(1 - e^{-\beta_1 t}). \tag{1}$$

Here β_0 represents the total number of faults that would be eventually detected and β_1 is the *per-fault hazard rate,* which is assumed constant for the exponential model. The total hazard rate at any instant (i.e., the failure intensity) is given by the total number of faults multiplied by the per-fault hazard rate. The available data [3] show that the number of fault introduced during the debugging process is only about 5%. Thus β_0, the total number of faults that will be found, may be estimated as the initial number of faults. It has been observed [4] that in an organization, the defect density (measured in defects/thousand lines of code) does not vary significantly at the beginning of the system test phase and thus may be estimated from past experience. This allows β_0 to be estimated with acceptable accuracy. Empirical methods to estimate defect density using programmer skill, etc., have also been proposed [5], [6].

The estimation of the other parameter β_1 is more complex. Musa *et al.* [3] have defined a parameter K, called *fault exposure ratio,* which can be obtained by normalizing the per-fault hazard rate with respect to the *linear execution frequency,* which is the ratio of the instruction execution rate and the software size. For 13 software systems they found that K varies from 1.41×10^{-7} to 10.6×10^{-7}, with the average value equal to 4.20×10^{-7} failures/fault.

Identifying what factors affect K is of considerable significance. If we can accurately model the behavior of K, there are three ways in which the software reliability engineering will be affected.

- When the process parameters are known *a priori,* optimal resource allocation can be done even before testing begins. Early planning can be crucial to the success of the project.
- In the early phases of testing, the failure intensity values observed contain considerable noise [7]. The use of reliability growth models in the early phases can sometimes result in grossly incorrect projection. The accuracy can be enhanced by using *a priori* parameter values in such cases.
- Residual defect density can be measured accurately.

Musa *et al.* have speculated that K may depend on program structure in some way. However, they suspected that for large programs, the "structuredness" (as measured by decision density) averages out and hence does not vary much from program to program [3]. Musa has also argued that K should be independent of program size [8]. Mayrhauser and Teresinki [9] have suggested that K may depend on testability, as measured by static metrics like "loopiness" and "branchiness" of the program. However, because of lack of sufficient data, the results are not yet conclusive [10].

We analytically examine random testing in the next section. As faults with higher testability [11]–[13] are likely to be exposed earlier, it is next shown that K should decline with time when random testing is done. Such behavior is observed in some read data sets; however, it is also observed that when the fault density is sufficiently low, a reversal in the behavior occurs [14]. At low densities K rises as density declines. This can be explained by observing that the random testing assumption becomes invalid at low fault densities. In Section IV it is shown that one hypothesis leads to logarithmic model; it has been shown in [15] that the logarithmic model indeed is the best among the two-parameter models with statistical significance.

Manuscript received December 1992; revised July 1993. This work was partly supported under an SDIO/IST Contract monitored by ONR. Recommended by F. Bastani.

The authors are with the Department of Computer Science, Colorado State University, Ft. Collins, CO 80523.

IEEE Log Number 9213116.

II. Analysis of Random Testing

In this section we examine random testing analytically. We assume that when the software is tested, each test is selected randomly. We also assume that the debugging is perfect and no new fault is generated. The latter assumption does not really restrict the applicability of our analysis; if it is not true, this would merely result in some change in the model parameter values [16].

The software being tested is executed a number of times. During each execution one *input* (set of input information or actions) is accepted and one *output* (set of output information or actions) is generated. Let $N(t)$ be the expected number of defects present in the system at time t. According to our assumptions above, $N(t)$ will monotonically decrease as faults are exposed and removed. Let T_s be the average time needed for a single execution, which is very small compared with the overall testing duration. Let K_s be the *fraction of existing faults exposed during a single execution.* Then

$$\frac{dN(t)}{dt}T_s = -K_s N(t). \tag{2}$$

It would be convenient to replace T_s with something which can be easily estimated. Let T_L be the *linear execution time* [3] which is defined as the total time needed if each instruction in the program was executed once and only once. It can be estimated using

$$T_L = I_s \cdot Q_x \frac{1}{r} \tag{3}$$

where I_s is the number of source statements, Q_x is the number of object (machine level) instructions per source instructions, and r is the object instruction execution rate of the computer being used. Let us consider the ratio $T_s/T_L = F$, where F is a parameter depending on the program structure. If a program is *loop-dominated*, i.e., if the program execution involves a large number of loops, then F may be greater than one because of higher node visitation frequency. For a *branch-dominated* program, F may be smaller than one since during a single execution, many branches would not be executed. Using (3) we can write (2) as

$$\frac{dN(t)}{dt} = -\frac{K_s}{F}\frac{N(t)}{T_L}. \tag{4}$$

Here it would be convenient to use a parameter K such that

$$K = \frac{K_s}{F}. \tag{5}$$

Here K is the same as the *fault exposure ratio* defined by Musa *et al.* in [3]. Using this (4) can be rewritten as

$$\frac{dN(t)}{dt} = -\frac{K}{T_L}N(t). \tag{6}$$

Equation (5) suggests that K may depend on the program structure. However, for a program with higher loop domination, both F and K_s would have higher values. This is because when larger number of loops are executed during a single execution, the faults associated with looping would have higher probability of being exposed. Similarly it can be argued that in a program with higher branch domination, both F and K_s would have lower values. Thus the value of K may not vary much with the program structure.

It should be noted that the per-fault hazard rate as given in (6) is K/T_L. Thus K (or K_s) directly controls the efficiency of the testing process.

A. Time-Invariant K

If we assume that K is time-invariant, then the above equation has the following solution:

$$N(t) = N(0)e^{-(K/T_L)t}. \tag{7}$$

This may be expressed in amore familiar form as follows:

$$N(0) - N(t) = N(0)(1 - e^{-t(K/T_L)}). \tag{8}$$

The left side of this equation corresponds to $\mu(t)$, the mean value function, as given by (1). Thus the parameters β_0 and β_1 have the following interpretations:

$$\beta_0 = N(0) \qquad \text{and} \qquad \beta_1 = \frac{K}{T_L}. \tag{9}$$

B. Detectability of Different Faults

Let us assume that the system is subject to possible faults $f_1, f_2, \cdots, f_M$. A randomly selected test may or may not test for a specific fault f_i. Let us define *detectability* d_i of a fault f_i in the following way:

$$d_i = \text{Prob}\{\text{a random input tests for } f_i\}. \tag{10}$$

The detectability of a fault would depend on how frequently the instructions containing the fault get executed [17], and the probability that the final output will be affected by the execution of those instructions. The probability that a random input will expose f_i is

$$\text{Prob}\{\text{the input tests for } f_i \text{ and } f_i \text{ is present}\} = d_i P_{f_i}(t) \tag{11}$$

where $P_{f_i}(t)$ is the probability that f_i is present. If λ_i is the hazard rate $-(dP_{f_i}/dt)$ for fault f_i, we have

$$d_i P_{f_i}(t) = \lambda_i(t)T_s = -T_s\frac{dP_{f_i}}{dt} \tag{12}$$

which has the solution

$$P_{f_i}(t) = P_{f_i}(0)e^{-(d_i/T_s)t}. \tag{13}$$

The overall hazard rate due to all M faults is given by

$$\lambda(t) = \sum_{i=1}^{M}\lambda_i(t) = \frac{1}{T_s}\sum_{i=1}^{M} d_i P_{f_i}(t). \tag{14}$$

Since $N(t)$ is the expected number of faults at time t, we have

$$N(t) = \sum_{i=1}^{M}\text{Prob}\{f_i \text{ is present at } t\} = \sum_{i=1}^{M} P_{f_i}(t). \tag{15}$$

Hence by (6), the overall hazard rate is also given by

$$\lambda(t) = -\frac{dN(t)}{dt} = \frac{K}{T_L}N(t) = \frac{K}{T_L}\sum_{i=1}^{M} P_{f_i}(t). \tag{16}$$

Comparing (14) and (16), we get

$$K = \frac{1}{F}\frac{\sum_{i=1}^{M} d_i P_{f_i}(t)}{\sum_{i=1}^{M} P_{f_i}(t)}. \tag{17}$$

This suggests that K would in general be a function of time, unless all d_i are equal, i.e., all the faults have the same detectability. Equation (17) states that the overall K is proportional to the weighted average of d_i for all faults and hence K can be regarded as a measure of average detectability for the software. Using (13) we have

$$K(t) = \frac{1}{F}\frac{\sum_{i=1}^{M} d_i e^{-t(d_i/T_s)}}{\sum_{i=1}^{M} e^{-t(d_i/T_s)}}. \tag{18}$$

A fault with higher detectability is likely to be exposed and removed earlier. The faults with lower detectability will remain causing $K(t)$ to decline with time. This can be verified by numerically plotting the value of $K(t)$ in (18). If we assume that all the faults $f_1, f_2, \cdots, f_M$ exist at time $t = 0$, then $N(0) = M$ and the maximum value of K is given by

$$K(0) = \frac{1}{M}\sum_{i=1}^{M}\frac{d_i}{F}. \tag{19}$$

The fault with the smallest value of d_i controls the minimum value of K.

$$\underset{t\to\infty}{K(t)} = \frac{1}{F}\frac{k_{\min}e^{-t(k_{\min}/T_s)}}{e^{-t(k_{\min}/T_s)}} = \frac{k_{\min}}{F} \tag{20}$$

where $k_{\min} = \min_{1\le i\le M}\{d_i\}$ = the detectability of the *least detectable* fault. Thus the value of K will range from initial average of d_i/F, as given by (19) and the smallest value of d_i/F, as given by (20). Equation (20) is also true when several faults with equal values of d_i have the smallest testability.

In general $K(t)$ is controlled by the *detectability profile* of the object under test. The detectability profile (DP) for an object is defined as

$$\Pi = \{\pi_{d_1}, \pi_{d_2}, \cdots, \pi_{d_{\max}}\} \tag{21}$$

where π_{d_i} is the total number of faults with detectability d_i. The DP is arranged such that $d_1 \le d_2 \le \cdots \le d_{\max}$, i.e., π_{d_1} denotes the number of faults with least detectability and $\pi_{d_{\max}}$ denotes the number of faults with maximum detectability. The concept of detectability profile was introduced by Malaiya and Yang [18] and is applicable to both hardware and software. If the detectability profile is known, then the expected fault coverage achieved by random (with replacement) or pseudo-random (without replacement) can be calculated. It can be shown that at high fault coverage levels only the lowest components of the DP (i.e., hardest to test faults) are significant. The DP of several hardware components has been evaluated. Adams [19] and Finnelli [20] have given DP's of software packages. The example below illustrates variation of $K(t)$ with time.

Example 1: Consider a software system containing faults with detectabilities $d_1 = 0.2 \times 10^{-8}$, $d_2 = 0.2 \times 10^{-7}$, $d_3 = 2 \times 10^{-7}$, $d_4 = 2 \times 10^{-6}$, $d_5 = 2 \times 10^{-5}$. Also assume that the detectability profile is $\{10, 50, 25, 5, 1\}$ and the value of T_L is 5×10^{-2}. Then $K(t)$ can be plotted as shown in Fig. 1(a) and (b); the line marked by A shows $K(t)$ as given by (18) and the line marked by B shows $K(t)$ as given by (22). The first uses a logarithmic x-axis; the second uses a linear scale.

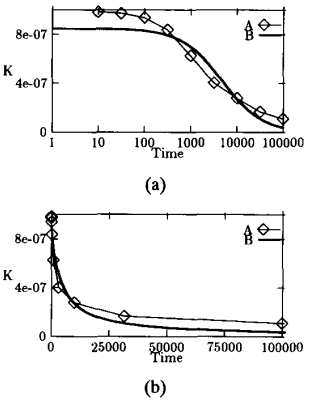


Fig. 1. *A*: variation of K for Example 1, *B*: approximation. (a) With logarithmic time scale. (b) With linear time scale.

While the above example uses values arbitrarily chosen for illustration, it shows that $K(t)$ may be regarded approximately constant for a short duration. It may change by an order of magnitude if t is sufficiently large. As shown below, such behavior has indeed been experimentally observed for some real situations.

Somewhat similar assumptions were used by Nakagawa and Hanata [12] for their error complexity model. They proposed a classification of errors into three classes which may be regarded as an approximation of the detectability profile by considering only three major components. Their model is different from other reliability growth models in that it requires recording not only the number of faults detected but also the distribution of the detected faults into the three classes.

C. A Useful Approximation for $K(t)$

The estimation of the detectability profile is hard for large combinational logic blocks [21], it would be even harder for large software systems. However, we have observed that the general shape of $K(t)$ appears to remain the same for different detectability profiles. $K(t)$ is a complicated function of time; however, Fig. 1(b) suggests that it may be approximately modeled as

$$K(t) = \frac{K(0)}{1+at}, \qquad \text{where} \quad a > 0. \tag{22}$$

Substitution of (22) into (6) yields

$$\frac{dN}{dt}T_L = -N\frac{K(0)}{1+at}$$

which has the solution

$$N(t) = N(0)(1+at)^{-[K(0)/aT_L]}. \tag{23}$$

If $a \ll 1/t$, the above equation can be approximated by (7) which describes the exponential model. Because of the

assumed decreasing testing efficiency, $N(t)$ will asymptotically approach a minimum value. Also, from the above two equations we have

$$K(t) = K(0)\left(\frac{N}{N(0)}\right)^{[aT_L/K(0)]}.$$

If $D(t)$ denotes the fault density at time t, then $D(t) = N(t)/I_sQ_x$ and we can express K in terms of fault density as follows:

$$K(D) = K(0)\left(\frac{D}{D(0)}\right)^{[aT_L/K(0)]}. \quad (24)$$

Thus if our assumptions at the beginning of this section are valid, K should vary with fault density. Then, we can make the following observations:

- If testing is random, the faults get harder and harder to expose near the end of the test phase.
- If inputs are applied randomly during the operational phase, the remaining faults have less probability of being encountered than we would have thought.

III. Evidence from Real Data

If the assumptions stated in the previous section are valid, K should decline monotonically (neglecting the "noise" that is always present) as the fault density decline. We now put this to test using real data sets. The number of useful data sets is still limited. Still, the observations noted below allow us to get a better understanding of the debugging process. We will first examine the variation of K with fault density (or time) for the same software system. Next, we will look at variation of K across different software systems.

A. Variation of K for Individual Data Sets

A dynamic test data set can be examined for variation in K with time or fault density. A grouped data set is of the form $(t_0, m_0), (t_1, m_1), \cdots, (t_f, m_f)$ where m_i is the total number of faults detected by time t_i. The experimentally obtained data sets contain considerable noise (short-term randomness), which must be filtered out by appropriate smoothing.

One way of estimating $K(t)$ at different times would be to divide the data set into several subsets. Curve fitting would yield values of β_1 for different subsets. The values of $K(t)$ can then be obtained using (5). However, curve fitting with a limited number of points would lead to inaccurate estimation of β_1 and hence $K(t)$. The other approach is to use (7). Using $N(t_i)$ and $N(t_{i+1})$, $K(t)$ for the duration (t_i, t_{i+1}) can be estimated as

$$K(t) = -\frac{T_L}{t_{i+1} - t_i} \ln\left(\frac{N(t_i)}{N(t_{i+1})}\right). \quad (25)$$

This is the computational approach we have used since we found that it results in more stable estimates. The group size must be large enough to filter out the short-term noise. Let us first consider data set $T1$ compiled by Musa [3]. This software system with 21.7 thousand object instructions was found to contain 136 faults. The collection of these data was carefully controlled. The overall exponential model parameters have been calculated as $\beta_0 = 142$ and $\beta_1 = 0.35 \times 10^{-4}$. If we take the CPU instruction execution rate to be 4×10^6/s, the overall value of K is given by $K = 1.9 \times 10^{-7}$. Fig. 2(a) and (b) shows the variation of K with time and fault density, respectively. At the points examined, K ranges between 3.6×10^{-7} and 1.3×10^{-7}. However, values of K, normalized with respect to its minimum value have been plotted to allow comparison with other data sets, which also use normalized values of K. Fig. 2(a) shows that K would decline as the fault density falls. Fig. 2(b) shows that K generally declines with time except for the last two points. Except for the last two points corresponding to lowest densities, the behavior is in accordance with the analysis in the previous section. Note that points corresponding to higher density occur earlier in time. As the discussion below suggests, the last two points indicate a reversal in the trend.

Let us now examine a few other data sets given in Table I. While the size is known for all of them, execution rates are not. This does not present a problem since we are interested in relative values of K. The variation of normalized values

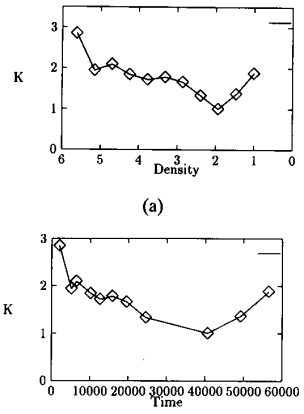


Fig. 2. Variation of K (normalized) with density and time for data set A in Table I.

TABLE I
Data Sets Used for Study of K

Data Set	Instructions (K Object Code)	Faults Detected	Initial Density (Est.)	Trend
A [3]	21.7	136	6.89	min. at 2
B [22]	2400	231	0.11	rising
C [23]	35	279	8.77	min. at 3
D [23]	5268	328	0.07	rising
E [24]	180	101	0.62	falling
F [25]	800	481	0.66	rising
G [26]	3480	535	0.17	rising
H [23]	160	46	0.32	rising
I [27]	4	27	7.43	min. at 4
J [27]	4	24	6.60	rising
K [27]	4	21	5.78	min. at 4
L [27]	4	27	7.43	???
M [27]	4	27	7.43	rising

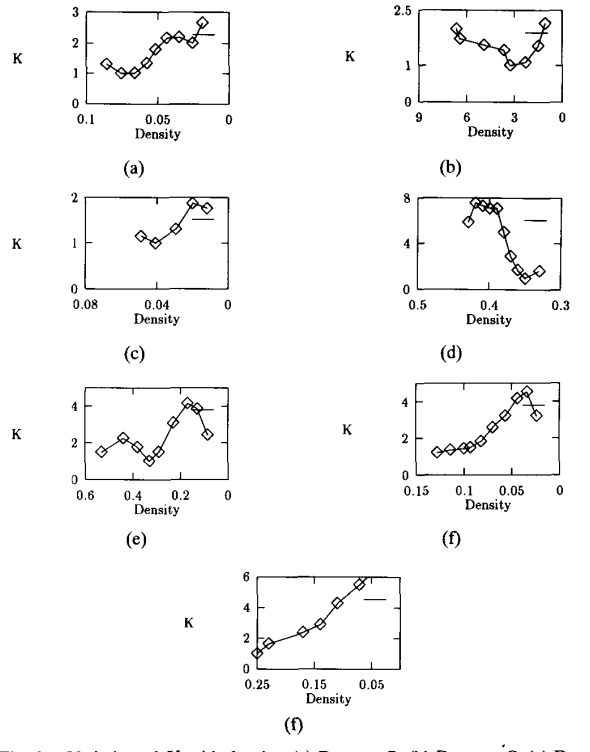


Fig. 3. Variation of K with density. (a) Data set B. (b) Data set C. (c) Data set D. (d) Data set E. (e) Data set F. (f) Data set G. (g) Data set H.

of K with density is plotted in Figs. 3(a)–(g) and 4(a)–(d). We assume that about 10% of the faults are still present when testing ends. There exists no accurate way to measure fault density at very low values. As we are primarily interested in observing the trend, we have used estimated values of defect densities.

The plots of Fig. 3 use data from several sources in USA and Japan. Fig. 3(a), (c), (e), (f), and (g) shows a rise of K as fault density declines. For all of these, the initial defect density is less than 1.0 per thousand object instructions. Figs. 2 and 3(b) appear to suggest that K first declines to a minimum at densities of about 1.9 and 2.5, respectively, and then starts rising. Fig. 3(d) shows a decline which is a noticeable exception. These observations seem to suggest the following:

- At higher fault density K declines, whereas at lower fault densities K appears to rise as testing progresses.
- The change of behavior appears to occur in the vicinity of fault density about 2, although it is likely to vary from system to system.

Some additional plots are given in Fig. 4(a)–(d). These are for five Japanese students testing their own compiler project [27]. Very small software (about 1000 lines) packages are used. The initial fault density is within the range normally encountered in industry [3]. For such a small project, we would not expect to see a specific trend. It is interesting to note that Fig. 3(a), (b), and (c) do show a specific trend. The value of K is stable or declines slightly until a density of about 3 to 4 and then it starts rising. This is consistent with out previous observation. The plot of Fig. 3(d) does not show a trend that can be easily interpreted, while Fig. 3(e) shows a rising trend.

An explanation of this phenomenon can be found by re-examining our assumption about testing being random. The data suggest that at low fault densities, actual testing is significantly more efficient than random testing. Since there are only a few faults left, the testers are more likely to examine the situations which may not have been considered before. Testing becomes more and more directed. The assumption of randomness of testing should be regarded as an approximation which may be valid during the early phases of testing.

B. Deterministic Testing

To see how K would be affected by nonrandomness, consider deterministic testing when the location of possible faults is known. Let us also assume that application of each test has the same likelihood of revealing the presence or absence of a new fault. In this situation, we can assume

$$\frac{dN}{dt} = -C \tag{26}$$

where C is a constant. Comparing this with (6), we have

$$K = T_L \cdot C \cdot \frac{1}{N} \tag{27}$$

or

$$K(D) = \frac{T_L \cdot C}{I_s}\left(\frac{1}{D}\right). \tag{28}$$

Thus K would rise as N falls. The situation assumed for (26) would be an extreme case. Equation (28) explains why K would start rising as the extreme situation is approached.

It should be remembered that when we describe the variation in K, we are examining the process at a very low level of granularity. We should thus not expect K to vary in a precise and smooth manner. Some of the peaks and valleys in K probably occur because of switching to new test suites.

C. K for Different Data Sets

We have seen that K varies with time as testing progresses. This suggest that the value of K may depend on the phase as reflected by the current fault density. It would be interesting to see if this dependence on fault density is also observed across separate projects. Fig. 5 gives the values of K for several data sets, as obtained from the table given by Musa [3]. In order to have a valid comparison we have chosen the x-axis to be the average fault density for all the data sets. Fig. 5 suggests that K declines with declining fault density until a $D_{K_{\min}}$ of about 2, and then it rises sharply as density approaches zero. On a preliminary examination, one might regard the isolated data point on the left-hand side or the two points with K greater than 8 to be outliers. However, based on the preceding discussion these points can be regarded as representing the expected behavior. This is supported by the discussion in the next section. Note that K may also depend on program evolution and the use of the operational profile. It would require further study to quantify this dependence.

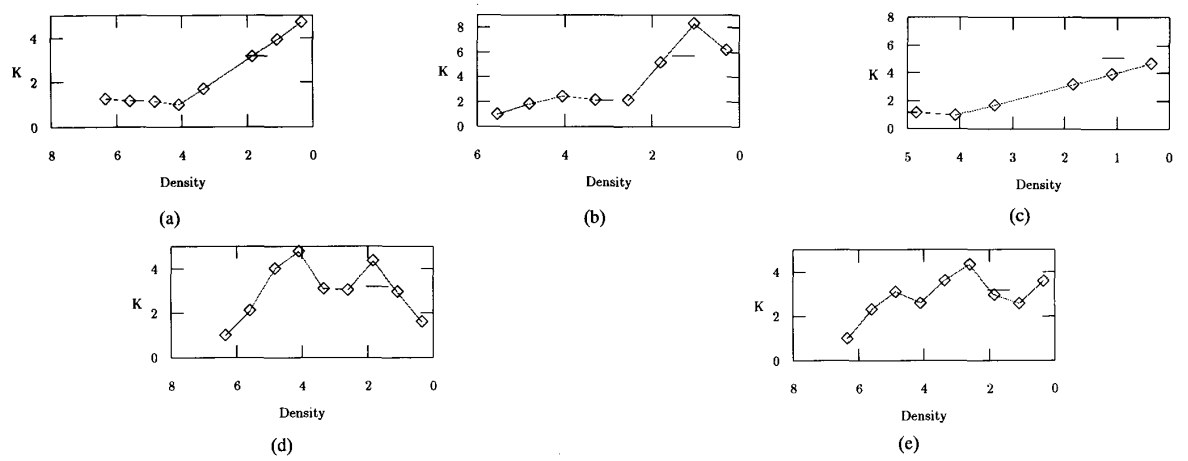


Fig. 4. Variation of K with density for five variations of the same project. (a) Data set I. (b) Data set J. (c) Data set K (d) Data set L. (e) Data set M.

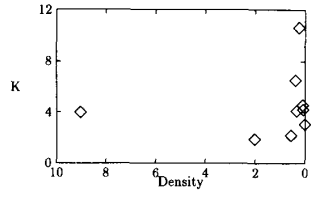


Fig. 5. Scatterplot of K versus density for different projects as given by Musa *et al.*

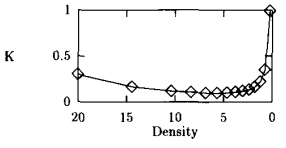


Fig. 6. Computed values of K for the logarithmic model.

IV. A Model Including Both Effects

In the above discussion we have identified two factors affecting K. Because faults get harder to find, (22) suggests that K should decline in accordance with a factor $1/(1+at)$. On the other hand, (27) suggests that K should rise as it is proportional to $1/N$, because effectiveness of testing increases as compared with random testing. It would be interesting to hypothesize that the behavior of K actually is given as a product of two factors, i.e.,

$$K(t) = \frac{g}{N(t)(1+at)} \tag{29}$$

where g is a parameter. It can be seen that the parameter $g = K(0)N(0)$. Substituting (29) into (6), we get

$$\frac{dN}{dt} = -\frac{1}{T_L}\frac{g}{(1+at)} \tag{30}$$

which has a solution

$$N = N(0) - \frac{g}{aT_L}\ln(1+at).$$

Rearranging terms, we get

$$N(0) - N = \frac{g}{aT_L}\ln(1+at). \tag{31}$$

This corresponds to the logarithmic model

$$\mu(t) = \beta_0 \ln(1+\beta_1 t) \tag{32}$$

where $\mu(t)$ is the mean value function. The parameters have the following correspondence:

$$\beta_0 = \frac{g}{aT_L} = \frac{K(0)N(0)}{aT_L}, \quad \text{and} \quad \beta_1 = a. \tag{33}$$

It has been shown [15] that the logarithmic model works better than other two-parameter models. The result has been shown to be statistically significant. If we assume that a debugging process for a system with $N(0) = 200$ is exactly described by a logarithmic model with $\beta_0 = 60$ *and* $\beta_1 = 1$, we can calculate the values of K at different densities. The values are plotted in Fig. 6. It shows a remarkable resemblance to Fig. 5. Here it should be noted that the parameter "a" introduced in (22) depends on the detectability profile. If this parameter can be estimated *a priori*, both β_0 and β_1 can be estimated, provided both $K(0)$ and $N(0)$ can be empirically obtained [28]. Empirical estimation of the parameter "a" remains an important problem.

V. Conclusion

The above examination of real data, along with mathematical analysis, suggests the following hypotheses:

- K represents the average detectability of the software under the test strategy being used.

- At higher fault densities, the value of K would sometimes fall as testing progresses. This is because faults with high detectability are found earlier and thus the remaining faults get harder and harder to find.
- At lower densities, the effective value of K starts to rise. This is caused by the fact that at this density range, testing is more efficient than what the assumption of random testing implies.

In the data sets examined, $D_{K_{\min}}$ appears to be in the region of 2–4 faults/1000 object instructions (i.e., about 8–16/KSLOC). As some of the plots of Fig. 4 suggest, a decreasing K may never be observed, suggesting that the second factor (28) may dominate from the beginning. It probably depends on the testing approach used. Additional data sets need to be examined to validate the hypothesis suggested and to arrive at an accurate model for $K(D)$ in the two regions. A preliminary model is suggested here in the form of equations (22) and (27). Taken together they may provide an explanation for why the logarithmic model works better. If the hypothesis is indeed valid, it has major consequences for both ordinary and ultra-reliable software.

- Some reliability growth models must be switched [15] or corrected when the fault density falls below $D_{K_{\min}}$. This would not be needed for the logarithmic model, which is a significant advantage.
- The reliability growth models [29] generally assume random testing. As (6) and (7) suggest, the fault detection rate drops exponentially and thus random testing would be very inefficient near the end of testing. However, our observations suggest, as given by (30), that the fault detection rate drops only according to an inverse linear relationship. This is the behavior modeled by the logarithmic model

$$\lambda = \frac{d\mu}{dt} = \frac{\beta_0\beta_1}{1+\beta_1 t}.$$

  Thus lower fault densities are more easily achievable than assumed under random testing.
- If we assume that during the operational phase, the inputs are effectively random choices from the operational profile, then the probability of faults with low densities being exposed would be significantly less than during testing. In other words, the last phases of testing may represent a highly accelerated form of exercising than during actual operation.

The last observation arises from the fact that if inputs are indeed random, then even at low-fault densities, K would be low in accordance with (18).

ACKNOWLEDGMENT

The authors wish to thank N. Li for his assistance in this paper. They also wish to thank Prof. Tohma for providing them with some of his technical reports. They are grateful to J. D. Musa and other anonymous reviewers for their detailed comments which greatly improved the presentation.

REFERENCES

[1] N. Karunanithi, D. Whitley, and Y. K. Malaiya, "Using neural networks in reliability prediction," *IEEE Software,* vol. 9, no. 4, pp. 53–59, July 1992.
[2] C. V. Ramamoorthy and F. B. Bastani, "Software reliability: Status and perspectives," *IEEE Trans. Software Eng.,* vol. SE-8, no. 4, pp. 354–371, Apr. 1982.
[3] A. Iannino, J. D. Musa, and K. Okumoto, *Software Reliability: Measurement, Prediction and Application.* New York: McGraw-Hill, 1987.
[4] G. A. Krueger, "Validation and further application of software reliability growth models," *Hewlett Packard J.,* pp. 75–79, Apr. 1989.
[5] M. Takahashi and Y. Kamayachi, "An empirical study of a model for program error prediction," in *Proc. 8th Int. IEEE Conf. on Software Engineering,* pp. 330–336, Aug. 1985.
[6] T. M. Khosgoftar and J. C. Munson, "Predicting software development errors using software complexity metrics," *IEEE J. Selected Areas in Communication,* vol. 8, no. 2, pp. 253–264, Feb. 1990.
[7] Y. K. Malaiya and P. K. Verma, "Testability profile and approach to software reliability," in M. H. Hamza, Ed., *Advances in Reliability and Quality Control.* Acta Press, Dec. 1988, pp. 67–71.
[8] J. D. Musa, "Rationale for fault exposure ratio K," *Software Eng. Notes,* vol. 16, no. 3, pp. 78–79, 1991.
[9] A. von Mayrhauser and J. A. Teresinki, "The effects of static code metrics toon dynamic software reliability models," in *Proc. Symp. on Software Reliability Engineering,* pp. 19.1–19.13, Apr. 1990.
[10] J. M. Keables, "Program structure and dynamic models in software reliability: investigation in a simulated environment," Ph.D. dissertation, Illinois Institute of Technology, 1991.
[11] B. Littlewood, "A Bayesian differential debugging model for software reliability," in *Proc. COMPSAC,* Oct. 1980, pp. 511–517.
[12] Y. Nakagawa and S. Hanata, "An error complexity model for software reliability measurement," in *Proc. Int. Conf. on Software Engineering,* 1989, pp. 230–235.
[13] A. von Mayrhauser and J. M. Keables, "A data collection environment for software reliability research," in *Proc. Int. Symp. on Software Reliability Engineering,* 1991, pp. 98–105.
[14] Y. K. Malaiya, A. von Mayrhauser, and P. K. Srimani, "The nature of fault exposure ratio," in *Proc. Int. Symp. on Software Reliability Engineering* (Raleigh, NC, Oct. 1992), pp. 23–32.
[15] Y. K. Malaiya, N. Karunanithi, and P. Verma, "Predictability of software reliability models," *IEEE Trans. Reliab.,* vol. 41, no. 4, pp. 539–546, Dec. 1992.
[16] M. Ohba and X. M. Chou, "Does imperfect debugging affect software reliability growth," in *Proc. Int. Conf. on Software Engineering,* 1989, pp. 230–235.
[17] W. W. Everett, "An extended execution time software reliability model," in *Proc. Int. Symp. on Software Reliability Engineering* (Raleigh, NC, Oct. 1992), pp. 4–13.
[18] Y. K. Malaiya and S. Yang, "The coverage problem for random testing," in *Proc. Int. Test Conf.,* Oct. 1984, pp. 237–242.
[19] E. A. Adams, "Optimizing preventing service to software systems," *IBM J. Res. Devel.,* vol. 28, pp. 2–14, Jan. 1984.
[20] G. B. Finnelli, "Results of software error-data experiments," in *Proc AIAA AHS ASEE Aircraft Design, Systems and Operations Conf.,* Sept. 1988, pp. 1–5.
[21] P. G. Ryan and W. K. Fuchs, "Partial detectability profiles, in *Proc. Int. Conf. on CAD,* Nov. 1990.
[22] P. M. Misra, "Software reliability analysis," *IBM Syst. J.,* vol. 22, pp. 262–270, Mar. 1983.
[23] M. Ohba, "Software reliability analysis models," *IBM J. Res. Devel.,* vol. 28, pp. 428–443, July 1984.
[24] N. D. Singpurwalla and R. Soyer, "Assessing software reliability growth using a random coefficient autoregressive process and its ramifications," *IEEE Trans. Software Eng.,* vol. SE-11, pp. 1456–1464, Dec. 1985.
[25] Y. Tohma, "Structural approach to the estimations of the number of residual software fault based on hypergeometric distribution," *IEEE Trans. Software Eng.,* vol. 15, pp. 345–355, Mar. 1989.
[26] ——, "Parameter estimation of the hypergeometric distribution model for real test/debug data," Tech. Rep. TR 90 1002, Tokyo Institute of Technology, Dep. Computer Sci., 1990.
[27] M. Matsumoto, K. Inoue, and K. Tori, "Experimental evaluation of software reliability models," in *Proc. IEEE Fault Tolerant Computing Sumposium (FTCS-18),* June 1988, pp. 148–153.

[28] L. Naixing and Y. K. Malaiya, "Empirical estimation of fault exposure ratio," Tech. Rep., Dep. Computer Sci., Colorado State Univ., 1993.
[29] Y. K. Malaiya and P. K. Srimani, Eds., *Software Reliability Models: Theoretical Developments, Evaluation and Applications.* IEEE Computer Society Press, Dec. 1990.

Yashwan K. Malaiya (S'76–M'78–SM'89) received the B.Sc. degree from the Government Degree College, Damoh, India, the M.Sc. degree from the University of Sangor, Sangor, India, and the M.Sc. Tech. degree from BITS, Pilani, India. In 1978 he received the Ph.D. degree in electrical engineering from Utah State University, Salt Lake City.

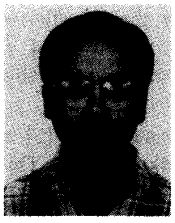
He was with the State University of New York, Binghamton, from 1978 to 1982. He is a Professor in the Computer Science Department and also in the Electrical Engineering Department at Colorado State University, Ft. Collins. He has published widely in the areas of fault modeling, software and hardware reliability, testing, and testable design. He has also been a Consultant to industry. He was a General Chair of the 24th International Symposium on Microarchitecture and the 6th International Conference on VLSI Design. He is the General Chair of the 4th International Symposium on Software Reliability Engineering. He has been the Chair of TC on Microprogramming and Microarchitecture. He is the Chair of the Software Test Subcommittee of TTTC and a Vice-Chair of the TCSE Subcommittee on Software Reliability Engineering.

Dr. Malaiya is a Member of the IEEE Computer Society TAB Executive Committee, and a member of the IEEE Computer Society Awards Committee. He has co-edited the IEEE Computer Society technical series books *Software Reliability Models, Theoretical Developments, Evaluation and Applications* and *Bridging Faults and IDDQ Testing.* He was a Guest Editor of special issues of *IEEE Software* and *IEEE Design and Test.*

Anneliese von Mayrhauser (M'93) received the Dipl. Inf. degree in Informatik from Technische Universität Karlsruhe, and the M.A. as well as Ph.D. degrees in computer science, from Duke University, Durham, NC.

She was with Illinois Institute of Technology, Chicago, and in 1991 joined Colorado State University, Ft. Collins, where she is now an Associate Professor on Computer Science. She is the author of *Software Engineering: Methods and Management* and of more than 60 conference and journal articles. She also works in industry on selective consulting projects.

Currently the second Vice President of IEEE Computer Society, Conferences and Tutorials, and a member of the Board of Governors, Dr. von Mayrhauser has recently been the Chair of TC on Software Engineering. She has been actively involved in several IEEE Computer Society activities. She was the General Chair of the 2nd International Symposium on Software Reliability Engineering, a Program Chair of the IEEE Computer Society International Software Metrics Symposium, and of the 4th International Symposium on Software Reliability Engineering.



Pradip K. Srimani (M'87–SM'90) received the B.Sc. degree in physics, the B. Tech., as well as the M. Tech. degrees in radiophysics and electronics, and the Ph.D. degree in computer science, all from the University of Calcutta, Calcutta, India, in 1970, 1973, 1975, and 1978, respectively.

He has served on the faculty of the Indian Statistical Institute, Calcutta, Gesselschaft für Mathematik und Datenverarbeitung, Bonn, Germany, The Indian Institute of Management, Calcutta, and Southern Illinois University, Carbondale. Currently, he is a Professor of Computer Science at the Colorado State University, Ft. Collins. His research interests include reliable systems, parallel algorithms, fault-tolerant computing, networks, and graph theory applications. He is a member of the Editorial Board of *International Journal of Simulation.*

Dr. Srimani is currently the Associate Editor-in-Chief of IEEE Computer Society Press, and is a member of the Editorial Board of IEEE SOFTWARE MAGAZINE. He is a Vice-Chair for Tutorials of the IEEE Computer Society CT Board. He is a member of ACM.