The Nature of Fault Exposure Ratio *

Y. K. Malaiya, A. von Mayrhauser and P. K. Srimani

Department of Computer Science
Colorado State University
Ft. Collins, CO 80523
malaiya@cs.colostate.edu

Abstract

The fault exposure ratio K is an important factor that controls the per-fault hazard rate, and hence the effectiveness of testing. 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, suggesting that real testing is more efficient than random testing. Data sets from several different projects are analyzed. Models for the two factors controlling K are suggested, which jointly lead to the logarithmic model.

1 Introduction

The software reliability models are needed for measuring and projecting reliability. While many of the models are purely empirical, some of the models are based on some specific assumptions about the fault detection/removal process. 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 are the easiest to explain. Using this model the expected number of faults $\mu(t)$ detected in a duration t may be expressed as

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

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 data collected by Musa [1] shows that the number of fault introduced during the debugging process is only about 5%. Thus β_0 may be estimated as the initial number of faults. It has been observed [9] 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 [27, 28].

The estimation of the other parameter β_1 is more complex. Musa et. al. have defined a parameter, K, called **fault exposure ratio**, which can be obtained by normalizing the per-fault hazard rate with respect to the software size and the instruction execution rate. 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} failure/fault.

What factors affect K? This question is of considerable significance. If we can accurately model the behavior of K, there are two ways in which the debugging process can be better planned.

- When both parameters can be known apriori, optimal resource allocation can be done even before testing begins.
- In the early phases of testing, the failure intensity values observed contain considerable noise [11]. The use of reliability growth models in the early phases can sometimes result in grossly incorrect projection. The accuracy can be enhanced by using apriori parameter values in such cases.

*This work was partly supported by an SDIO/IST contract monitored by ONR.

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) may not vary much since it is highly correlated with program size [1]. Musa has also argued that K should be independent of program size [2]. Mayrhauser and Teresinki [3] 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 [4]. This paper suggests that K depends on fault density and also on testing strategies. First, an analytical examination of random testing is presented. As faults with higher testability [5, 6, 19] are likely to be exposed earlier, it is shown that K should decline with time when random testing is done. Some real data sets appear to support this; however, it is observed that when the fault density is sufficiently low, a reversal in the behavior occurs. 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. The implications of these observations for ordinary as well as for ultra-reliable software are discussed.

2 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 doesn't really restrict the applicability of our analysis; if it is not true, this would merely result in some change in the parameter values [7].

The software being tested is executed repeatedly. 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* [1] 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

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

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 $\frac{T_s}{T_L} = F$, 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 [29]. For a *branch-dominated* program, F would be smaller than one since during a single execution, many branches would not be executed. Using Equation 3 we can write Equation 2 as

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

Let us define a new parameter

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

Using this Equation 4 can be rewritten as

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

The equation 5 above 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 Equation 6 is K/T_L. Thus K (or K_s) directly controls the efficiency of the testing process.

2.1 Time Invariant K

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

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

This may be expressed in a more familiar form as follows:

$$N(0) - N(t) = N(0)(1 - e^{-t\frac{K}{T_L}}) \quad (8)$$

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

$$\beta_0 = N(0), \quad and \quad \beta_1 = \frac{K}{T_L} \quad (9)$$

2.2 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:

$$k_i = Prob\{\text{a random input tests for } f_i\} \quad (10)$$

The probability that a random input will expose f_i is

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

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

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

which has the solution

$$P_{f_i}(t) = P_{f_i}(0) e^{-\frac{k_i}{T_s} t} \quad (13)$$

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

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

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

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

Hence by Equation 6, the overall hazard rate is also given by

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

Comparing Equations 14 and 16, we get

$$K = \frac{1}{F} \frac{\sum_{i=1}^{M} k_i P_{f_i}(t)}{\sum_{i=1}^{M} P_{f_i}(t)} \quad (17)$$

This suggests that K would in general be a function of time, unless all k_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 k_i for all faults. Using (13) we have,

$$K(t) = \frac{1}{F} \frac{\sum_{i=1}^{M} k_i e^{-t\frac{k_i}{T_s}}}{\sum_{i=1}^{M} e^{-t\frac{k_i}{T_s}}} \quad (18)$$

A fault with a larger value of k_i is likely to be exposed and removed earlier. In other words, the weights of larger k_i in the numerator of (18) will decline faster. Hence $K(t)$ will decline monotonically with time. If we assume that all the faults $f_1, f_2, \ldots, f_M$ exist at time $t = 0$, then $N(0) = M$ and the maximum value of K is given by

$$K(0) = \frac{1}{M} \sum_{i=1}^{M} \frac{k_i}{F} \quad (19)$$

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

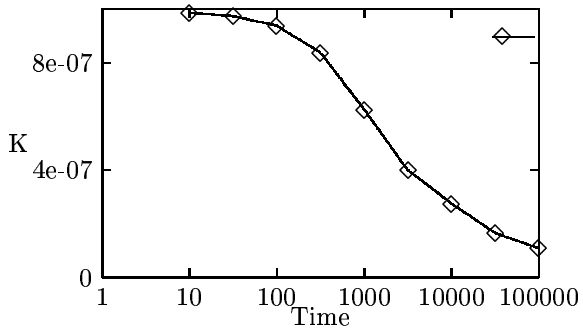
$$\begin{matrix} K(t) \\ t \to \infty \end{matrix} = \frac{1}{F} \frac{k_{min} e^{-t\frac{k_{min}}{T_s}}}{e^{-t\frac{k_{min}}{T_s}}} = \frac{k_{min}}{F} \quad (20)$$

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

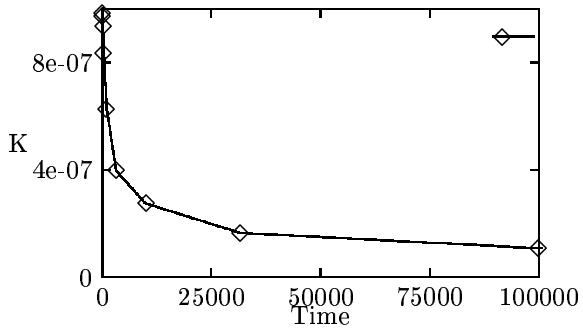
In general $K(t)$ is controlled by the *detectability profile* of the object under test. The concept of detectability profile was introduced by Malaiya and Yang [12] and is applicable to both hardware and software. The detectability profile (DP) for an object is defined as

$$\Pi = \{\pi_{d_1}, \pi_{d_2}, \cdots, \pi_{d_{max}}\} \quad (21)$$

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. 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. Finnelli



(a) With logarithmic time scale



(b) With linear time scale

Figure 1: Variation of K for Example 1

[20] have given DP of two software packages. The example below illustrates variation of $K(t)$ with time.

Example 1: Consider a software system contains 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 $K(t)$, as given by equation (18), can be plotted as shown in Figures 1(a and b). The first uses a logarithmic x-axis; the second uses a linear 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.

The estimation of the detectability profile is hard for large combinational logic blocks [14], 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, Figure 1(b) suggests that it may be approximately modeled as

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

Substitution of Equation 22 into 6 yields

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

which has the solution

$$N(t) = N(0)(1 + at)^{-\frac{K(0)}{aT_L}} \tag{23}$$

If $a \ll 1/t$, the above equation can be approximated by equation (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

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

If $D(t)$ denotes the fault density at time t, then $D(t) = \frac{N(t)}{I_s Q_s}$ and we can express K in terms of fault density as follows:

$$K(D) = K(0) \left(\frac{D}{D(0)} \right)^{\frac{aT_L}{K(0)}} \tag{24}$$

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.

Somewhat similar assumptions were used by Nakagawa and Hanata [6] 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.

3 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.

3.1 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 $(t_0, m_o), (t_1, m_1), \cdots (t_f, m_f)$ 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. The other approach is to use equation (7). Using $N(t_i)$ and $N(t_{i+1})$, $K(t)$ for the duration (t_i, t_{i+1}) can be estimated as

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

This is the computational approach we have used. The group size must be large enough to filter out the short term noise. Let us first consider data set T1 compiled by Musa [1]. This software system with 21.7 thousand object instructions was found to contain 136 faults. The collection of this 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 per second, the overall value of K is given by $K = 1.9 \times 10^{-7}$. Figures 2(a and b) show 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, normalized values of K have been plotted to allow comparison with other data sets, which also use normalized values

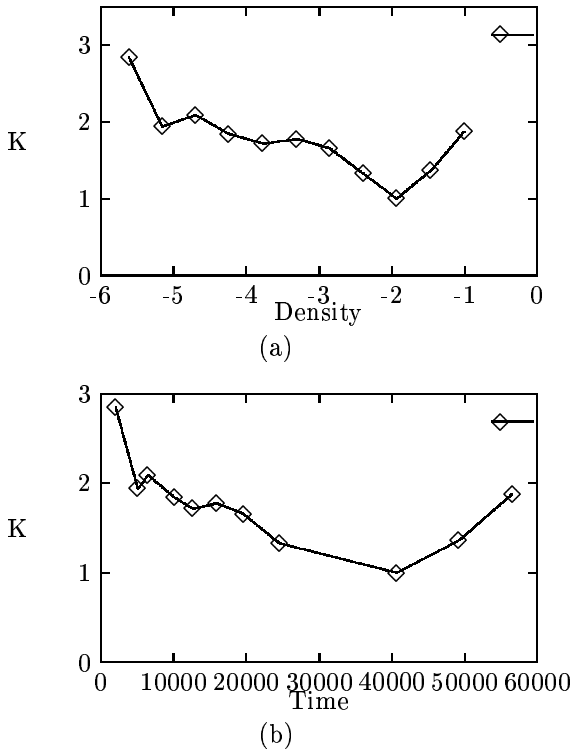
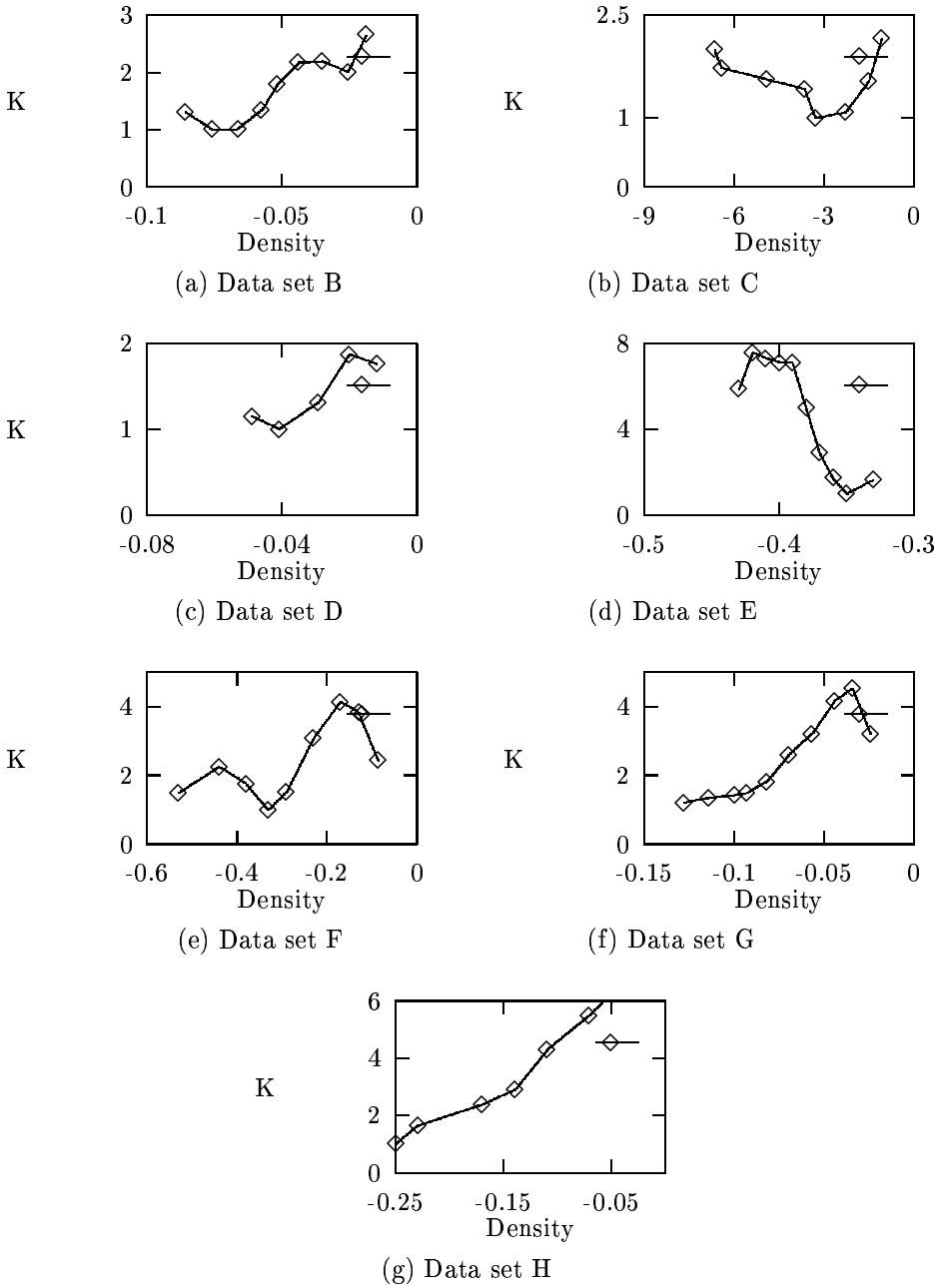


Figure 2: Variation of K (normalized) with density and time for data set A in Table 1

of K. Figure 2a shows that K generally declines with time except for the last two points. Figure 2b shows that K would similarly decline as the fault density falls. 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 1. 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 of K with density is plotted in Figure 3(a–g) and Figure 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 Figure 3 use data from several sources in USA and Japan. Figures 3 (a), (c), (e), (f) and



(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

Figure 3: Variation of K with density

Data Set	Instructions (K Object code)	Faults Detected	Initial Density (est)	Trend
A	21.7	136	6.89	Min at 2
B	2400	231	0.11	Rising
C	35	279	8.77	Min at 3
D	5268	328	0.07	Rising
E	180	101	0.62	Falling
F	800	481	0.66	Rising
G	3480	535	0.17	Rising
H	160	46	0.32	Rising
I	4	27	7.43	Min at 4
J	4	24	6.60	Rising
K	4	21	5.78	Min at 4
L	4	27	7.43	???
M	4	27	7.43	Rising

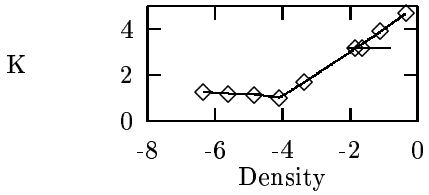
Table 1: Trends for different data sets

(g) show 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. Figure 2 and Figure 3(b) show that K first declines to a minimum at densities of about 1.9 and 2.5 respectively and then start rising. Figure 3(d) shows a decline which is a noticeable exception. These suggest the following:

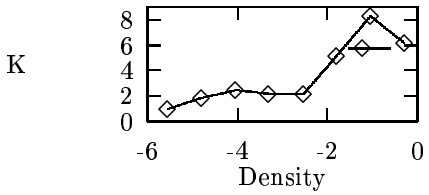
- At higher fault density K declines, whereas at lower fault densities K appears to rise as testing progresses.
- The change of behavior appears to occur in the vicinity of fault density about 2, although it is likely to vary from system to system.

Some additional plots are given in Figure 4(a–d). These are for five Japanese students testing their own compiler project. These are very small software (about 1000 lines) packages. The initial fault density is within the range normally encountered in industry [1]. For such a small project, we would not expect to see a specific trend. It is interesting to note that Figure 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 our previous observation. The plot of Figure 3(d) does not show a trend that can be easily interpreted, while Figure 3(e) show a rising trend.

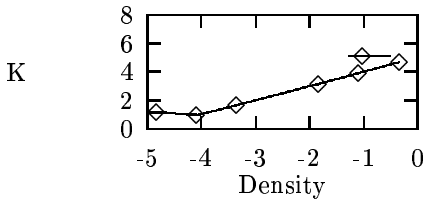
An explanation of this phenomenon can be found by re-examining our assumption about testing being random. The data suggests that at low fault densities, actual testing is significantly more efficient than random testing. Since there are only a few faults left, the testers are more likely to examine the situations which may not have been considered before. Testing



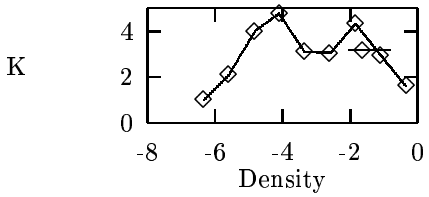
(a) Data set I



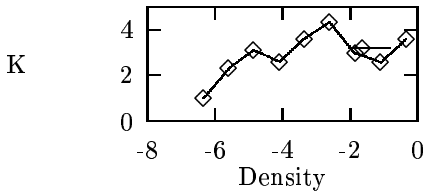
(b) Data set J



(b) Data set K



(c) Data set L



(d) Data set M

Figure 4: Variation of K with density for five variations of the same project

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.

3.2 Deterministic Testing:

To see how K would be affected by non-randomness, consider deterministic testing when the location of possible faults is known. Let us also assume that application of each test would reveal 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 equation (1), we have

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

or

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

Thus, K would rise as N falls. The situation assumed for Equation 26 would be an extreme case. The 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.

3.3 K for different data sets:

Figure 4 gives the values of K for several data sets, as obtained from the table given by Musa [1]. In order to have a valid comparison we have chosen the x-axis to be the average fault density for all the data sets. sets. Figure 5 suggests that there may be other factors affecting K; however, it again supports the earlier observation that K declines with declining fault density until a $D_{K_{min}}$ of about 1.5, and then it rises sharply as density approaches zero.

4 A Model Including Both Effects

The Equation (22) suggests that K should decline in accordance with a factor $\frac{1}{1+at}$. On the other hand Equation (27) suggests that K should rise as it is

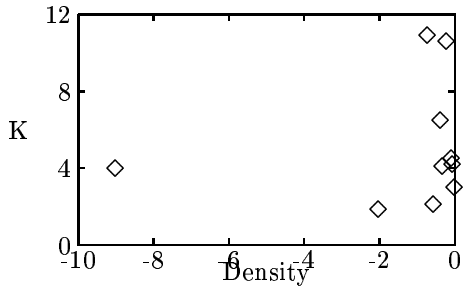


Figure 5: Scatterplot of K vs. density for different projects as given by Musa et. al.

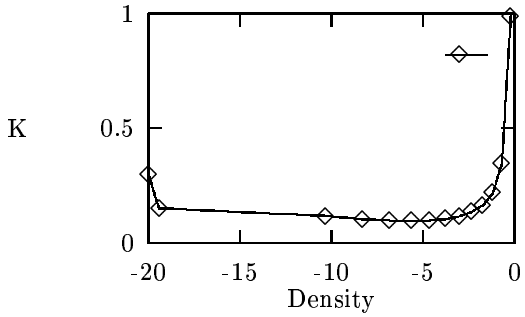


Figure 6: The Computed values of K for the logrithmic model

Figure 6: Computed values of K for the logarithmic model

proportional to $1/N$. It would be interesting to hypothesize that the behavior of K actually is given as a product of two factors, i.e.,

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

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

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

which has a solution

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

Rearranging terms, we get,

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

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:

$$\beta_0 = \frac{g}{T_L} = \frac{K(0)N(0)}{T_L} \quad and \quad \beta_1 = a \tag{33}$$

It has been shown [18] that the logarithmic model works better than other 2-parameter models. 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 Figure 6. It shows a remarkable resemblance to Figure 5.

5 Conclusion

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

- At higher fault densities, the value of K would generally 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 Figure 4 suggest, a decreasing K may never be observed, suggesting that the second factor (Equation 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 [18] or corrected when the fault density falls below $D_{K_{min}}$.
- The reliability growth models [16] generally assume random testing. Our examination suggests that testing becomes more efficient at very low densities and thus ultra-reliable software is more achievable than is thought [17].
- If we assume that during the operational phase, the inputs are effectively random choices from the operational profile, then some faults have less probability of being exposed than we would otherwise think. 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).

Acknowledgements: We would like to thank Li Naixing for his assistance in this paper. We are also grateful to Professor Tohma for providing us with some of his technical reports and Prof. Gustafson for providing us with Oeff's thesis.

References

[1] J.D. Musa, A. Iannino and K. Okumoto, **Software Reliability: Measurement, Prediction, Application**, McGraw-Hill, 1987.

[2] J.D. Musa, "Rationale for Fault Exposure Ratio K," **ACM SIGSOFT Software Engineering News**, pp. 79, July 1991.

[3] A. von Mayrhauser and J.A. Teresinki, "The Effects of Static Code Metrics on Dynamic Software Reli-

ability Models," **Proc. of Symposium on Software Reliability Engineering**, pp. 19.1-19.13, April 1990.

[4] J.M. Keables, "Program Structure and Dynamic Models in Software Reliability: Investigation in a Simulated Environment," Ph.D. Dissertation, Computer Science Dept., Illinois Institute of Technology, 1991.

[5] B. Littlewood, "A Bayesian Differential Debugging Model for Software Reliability," Proceedings of **COMPSAC**, pp. 511-517, ctober 1980.

[6] Y. Nakagawa and S. Hanata, "An Error Complexity Model for Software Reliability Measurement," Proceedings of **Int. Conf. on Software Engineering**, pp. 230-235, 1989.

[7] M. Ohba and X.M. Chou, "Does Imperfect Debugging Affect Software Reliability Growth?," Proceedings of **Int. Conf. on Software Engineering**, pp. 237- 244, 1989.

[8] B. Littlewood, "Stochastic reliability growth: a model for fault removal in computer programs and hardware designs," **IEEE Trans. Reliability**, Vol. R-30, pp. 313–320, October 1981.

[9] G.A. Kruger, "Validation and Further Application of Software Reliability Growth Models," **Hewlett-Packard Journal**, pp. 75-79, April 1989.

[10] Y.K. Malaiya, S. Sur, N. Karunanithi and Y.C. Sun, "Implementation Considerations for Software Reliability," Proceedings of **Software Reliability Symposium**, pp. 6.27-6.30, June 1990.

[11] Y.K. Malaiya and P. Verma, "Testability Profile Approach to Software Reliability," in **Advances in Reliability and Quality Control** (Ed. M.H. Hamza), Acta Press, pp. 67-71, December 1988.

[12] Y.K. Malaiya and S. Yang, "The Coverage Problem for Random Testing," Proceedings of **Int. Test Conference**, pp. 237-242, October 1984.

[13] K. Wagnor, C. Chin and E. McCluskey, "Pseudorandom Testing," **IEEE Trans. Comput.**, Vol. C-36, pp. 332-343, March 1987.

[14] P.G. Ryan and W. Kent Fuchs, "Partial Detectability Profiles," Proceedings of **Int. Conf. on CAD**, November 1990.

[15] Y.K. Malaiya, N. Karunanithi and P. Verma, "Predictability Measures for Software Reliability Models," Proceedings of **COMPSAC**, pp. 7-12, October 1990.

[16] Y.K. Malaiya and P.K. Srimani (Eds.), **Software Reliability Models**, IEEE Computer Society Press, December 1990.

[17] R.W. Butler and G.B. Finelli, "The Infeasibility of Experimental Quantification of Life-Critical Software Reliability," Proceedings of **ACM SIGSOFT '91 Conference on Software for Critical Systems**, pp. 66-76, December 1991.

[18] Y.K. Malaiya, N. Karunanithi and P. Verma, "Predictability of Software Reliability Models," To appear in **IEEE Trans. on Reliability**, 1992.

[19] A. von Mayrhauser and J.M. Keables, "A Data Collection Environment for Software Reliability Research," Proceedings of **Int. Symp. on Software Reliability Engineering**, pp. 98-103, 1981

[20] G.B. Finnelli, "Results of software error-data experiments," Proceedings of **AIAA/AHS/ASEE Aircraft Design, Systems and Operations** Conference, pp. 1–5, September 1988.

[21] M. Matsumoto, K. Inoue, T. Kituno, and K. Tori, "Experimental evaluation of software reliability models," Proceedings of **IEEE FTCS-18**, pp. 148–153, June 1988.

[22] Y. Tohma et. al., "Structural approach to the estimations of the number of residual software fault based on hypergeometric distribution," **IEEE Trans. Software Engineering**, pp. 345–355, March 1989.

[23] Y. Tohma et. al. ,"Parameter estimation of the hypergeometric distribution model for real test/debug data," Technical report # 90 1002, department of Computer Science, Tokyo Inst. of Technology, 1990.

[24] M. Ohba, "Software reliability analysis models," **IBM Journal of Research & Development**, Vol. 28, pp. 428–443, July 1984.

[25] N.D. Singpurwalla and R. Soyer, "Assessing software reliability growth using a random coefficient autoregressive process and its ramifications," **IEEE Trans. Software Engineering**, Vol. SE-11, pp. 1456–1464, December 1985.

[26] P. M. Misra, "Software reliability analysis," **IBM Systems Journal**, Vol. 22, pp. 262–270, March 1983.

[27] M. Takahashi and Y. Kamayachi, "An empirical study of a model for program error prediction," **Proc. Int. Conf. on Software Engineering**, pp. 330–336, August 1985.

[28] T.M. Khoshgoftar and J.C. Munson, "The live of code metric as a predictor of program faults: a critical analysis," **Proc. COMPSAC '90**, pp. 408-413.

[29] B.S. Oeff, "The effect of program structures on error rates using Unix system utilities," MS Thesis, Kansas State University, 1992.