Computer Science
Technical Report

Colorado State University

Fault Exposure Ratio Estimation and Applications*

Li Naixin
Microsoft Corp.
One Microsoft Way
Redmond WA 98052
naixinli@microsoft.com

Yashwant K. Malaiya
Computer Science Dept.
Colorado State University
Fort Collins, CO 80523
malaiya@cs.colostate.edu

Technical Report CS-96-130

Computer Science Department
Colorado State University
Fort Collins, CO 80523-1873

Phone: (970) 491-5792 Fax: (970) 491-2466
WWW: http://www.cs.colostate.edu

*This research was supported by a BMDO funded project monitored by ONR

Fault Exposure Ratio Estimation and Applications*

Li Naixin
Microsoft Corp.
One Microsoft Way
Redmond WA 98052
naixinli@microsoft.com

Yashwant K. Malaiya
Computer Science Dept.
Colorado State University
Fort Collins, CO 80523
malaiya@cs.colostate.edu

ABSTRACT

One of the most important parameters that control reliability growth is the *fault exposure ratio* (FER) identified by Musa et al. It represents the average detectability of the faults in software. Other parameters that control reliability growth are software size and execution speed of the processor which are both easily evaluated. The fault exposure ratio thus presents a key challenge in our quest towards understanding the software testing process and characterizing it analytically.

It has been suggested that the fault exposure ratio may depend on the program structure, however the *structuredness* as measured by decision density may average out and may not vary with program size. In addition FER should be independent of program size. The available data sets suggest that FER varies as testing progresses. This has been attributed partly to the non-randomness of testing.

In this paper, we relate defect density to FER and present a model that can be used to estimate FER. Implications of the model are discussed. This model has three applications. First, it offers the possibility of estimating parameters of reliability growth models even before testing begins. Secondly, it can assist in stabilizing projections during the early phases of testing when the failure intensity may have large short-term swings. Finally, since it allows analytical characterization of the testing process, it can be used in expressions describing processes like software test coverage growth.

1 Introduction

The defect removal process plays a critical role in achieving software with the desired reliability level. The software defects occur largely because of the imperfections of the human design and coding process. In order to plan and control the test phase of a software development project, the manager has to assess how the defect removal process is proceeding and when the defects or the defect encounter rate would be expected to be below an acceptable standard.

We can regard the process as being governed by two parameters.

1. *The initial number of defects.* This would depend on the factors that influence the occurrence of the defects in the first place as well as defect removal that may already have occurred before the current test phase, for example during unit testing. The purpose of the current test phase would be to reduce the number of defects to a sufficiently low level.
2. *Effectiveness of the defect removal process.* If the defects are found and removed faster, the target reliability levels will be achieved earlier. This effectiveness would depend on some obvious factors, like the computational performance of the CPU. It would also depend on how the tests are selected and how the remaining defects are distributed.

*This research was supported by a BMDO funded project monitored by ONR

While test effectiveness will vary from test to test, the test manager would like to look at the overall trend. This requires a statistical characterization of the process. This is done by use of appropriate models.

While many of the reliability growth 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

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

Here β_0^E represents the total number of faults that would be eventually detected and β_1^E is the per fault hazard rate which is assumed to be constant for exponential model. The data collected by Musa [1] shows that the number of additional faults introduced during the debugging process is only about 5%. Thus β_0^E may be estimated as the initial number of faults.

It has been observed [8] that in an organization, the defect density (measured in defects/thousand lines of code) at the beginning of the system test phase does not vary significantly and thus may be estimated with acceptable accuracy. Empirical methods to estimate defect density using programmer skill etc. have also been proposed [9, 10, 16].

The estimation of the other parameter β_1^E is more complex. Musa et al. have defined a parameter K, called **fault exposure ratio** (FER) 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 the overall FER varies from 1.41×10^{-7} to 10.6×10^{-7}, with the average value equal to 4.2×10^{-7} failure/fault. Once we know the value of K, β_1^E can be estimated using,

$$\beta_1^E = \frac{K}{T_L} \tag{2}$$

where T_L is the linear execution time [1], given by $T_L = I_s Q_r \frac{1}{r}$; I_s is the number of source lines of code; Q_r is the average object instructions per source statement; r is the CPU instruction execution rate.

If $N(t)$ is the total number of defects still present at time t, we can show that [4],

$$\frac{dN(t)}{dt} = -\frac{K}{T_L} N(t) \tag{3}$$

Thus the defect finding rate $-\frac{dN(t)}{dt}$ is proportional to the fault exposure ratio. Note that in the above equation, the effect of the software size and the instruction execution rate of the CPU has been taken into account separately in the term T_L.

If we wish to characterize the defect removal process accurately, we need to identify the factors that control FER. Once we have done that, we can empirically estimate FER and hence the defect removal rate. We will also be able to assess the potential techniques for enhancing the defect removal rate. The researchers have considered the following as possible factors.

Program Structure: Musa et al. [1] have speculated that FER may depend on the program structure in some way. However, they suggested that for large programs, the "structuredness" (as measured by decision density) may be average out. [1]. von Mayrhauser and Teresinki [11] have suggested that FER may depend on static metrics like "loopiness" and "branchiness" of the program. However, because of lack of sufficient data, the results are not conclusive [12]. Malaiya et al. [3] have identified a factor that depends on the program structure, but they show that its affect is counterbalanced by another factor, and thus FER should be relatively unchanged with program structure. Further studies are needed to see if the program structure has any influence on the FER.

Program Size: Musa has also presented reasons why FER should be independent of program size [2].

Testing effectiveness: Malaiya et al. [3] have suggested that test strategies cause FER to vary. Since real testing is not random but directed, it becomes more effective in comparison with random testing as testing progresses. Their analysis of several data sets supports this hypothesis.

In this paper we relate FER to the stage of testing as discussed in the next section.

A study of the factors affecting FER 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 *a priori*, the testing time needed to achieve target reliability can be calculated and hence, optimal resource allocation can be done even before testing begins.
- In early phases of testing, the failure intensity values observed contain considerable noise. 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.

In this paper we refer to two well known SRGMs, the *exponential* model referred above and the *logarithmic* model. Both are 2-parameter models and are thus comparable. The exponential model offers the advantage that the physical meaning of its two parameters is easily understood, however studies have shown that in most cases the logarithmic model provides much better predictive accuracy. It is possible to regard the exponential model as a convenient approximation of the logarithmic model. Both models have been used in industry and by researchers. There are other models that work well in specific cases, however they are not considered here.

In this work we are primary concerned with the relation between FER and initial fault density D_0. Starting with the logarithmic SRGM model, we obtain a model giving variation of FER with defect density D. We next discuss how this model can be used in software reliability engineering.

2 Variation of FER with Fault Density

The detectability of a fault is defined as the probability that the fault is detected by a randomly selected test input [3]. For truly random testing, faults with high detectability tends to be detected earlier in time, so FER should in general decline with time. However experiments with real reliability data indicates a reversal of this trend at low defect densities [3]. One possible explanation for this phenomenon is that at the later stages of the testing phase testing becomes more and more directed, rather than being random. The testers have a good idea of what part of the functionality has not yet been exercised, and they would tend to use tests that are likely to be more effective. Another possibility is that some faults may block access to some other faults, and thus removal of some faults may allow more faults to become accesible.

The logarithmic model, has been shown to have higher predictive capability in general than other two-parameter software reliability models by several researchers [14, 1, 7]. Malaiya et al [7] have shown using a number of diverse data sets that the superiority of the logarithmic model is statistically significant. The model is characterized by these equations.

$$\mu(t) = \beta_0^L \ln(1 + \beta_1^L t) \tag{4}$$

$$\lambda(t) = \frac{\beta_0^L \beta_1^L}{1 + \beta_1^L t} \tag{5}$$

where $\lambda(t)$ is the failure intensity at time t, $\mu(t)$ is the mean value function and β_0^L and β_0^L are the two parameters for the logarithmic model.

In [3] it is shown that the general trend observed for FER is consistent with what would be expected for a process obeying the logarithmic model.

In order to relate FER to the test stage, we need a quantitative metric that measures the *stage* of testing. The actual amount of testing done in phases like unit testing and integration testing can very significantly for different

projects. Thus if we only use the testing time in a specific phase, it would not take into account the debugging that has taken place in the prior phases. Here we use *defect density* as a measure of the test stage, because it is independent of the project to project variation of the specific stage at which a given test phase begins.

An expression for FER in terms of the testing time is derived in [3]. Here we will obtain an expression for FER in term of the fault density D, which is a function of the testing time t. Taking logarithm on both sides of Equation 5, we get

$$\ln(\lambda(t)) = \ln(\beta_0^L \beta_1^L) - \ln(1 + \beta_1^L t) \tag{6}$$

Using Equation 4

$$\ln(\lambda(t)) = \ln(\beta_0^L \beta_1^L) - \frac{1}{\beta_0^L}\mu(t) \tag{7}$$

Also, by definition,

$$D(t) = \frac{N(t)}{I_s} \tag{8}$$

Since,

$$\lambda(t) = -\frac{dN(t)}{dt} \tag{9}$$

Equation 3 yields,

$$K(t) = T_L \frac{\lambda(t)}{N(t)} \tag{10}$$

Using Equation 8 and Equation 10 we obtain,

$$\ln(K(t)D(t)) = \ln(T_L \frac{\lambda(t)}{I_s}) = \ln(\frac{I_s Q_r \frac{1}{r}}{I_s}\lambda(t)) \tag{11}$$

Reorganizing,

$$\ln(\lambda(t)) = \ln(K(t)D(t)) - \ln(\frac{Q_r}{r}) \tag{12}$$

Since $\mu(t)$ can also be expressed as,

$$\mu(t) = N_0 - N(t) = N_0 - I_s D(t) \tag{13}$$

substituting Equation 12 and Equation 13 into Equation 7, we obtain,

$$\ln(K(t)D(t)) - \ln(\frac{Q_r}{r}) = \ln(\beta_0^L \beta_1^L) - \frac{1}{\beta_0^L}(N_0 - I_s D(t)) \tag{14}$$

Rearranging, we get,

$$\ln(K(t)D(t)) = \ln(\beta_0^L \beta_1^L) + \ln(\frac{Q_r}{r}) - \frac{N_0}{\beta_0^L} + \frac{I_s}{\beta_0^L} D(t) \tag{15}$$

Both defect density D and the FER vary as testing proceeds. The exact process of defect detection and removal depends on the testing strategy, CPU speed, etc. Including other factors in the parameters, we will express FER as a function of D. Other factors will be manifested through variation in D. Let us rewrite the above equation as:

$$\ln(KD) = \ln(\alpha_0) + \alpha_1 D \tag{16}$$

where the parameters α_0 and α_1 are given by,

$$\alpha_0 = \frac{\beta_0^L \beta_1^L Q_r}{r} e^{-\frac{N_0}{\beta_0^L}} \tag{17}$$

$$\alpha_1 = \frac{I_s}{\beta_0^L} \tag{18}$$

This allows us to write FER as,

$$K = \frac{\alpha_0}{D} e^{\alpha_1 D} \tag{19}$$

To illustrate the nature of Equation 19, let us assume that a debugging process for a system with $N_0 = 200$ is exactly described by a logarithmic model with $\beta_0^L = 60$ and $\beta_1^L = 1$. We can calculate the values of FER at different densities. The values are plotted in Figure 1 along with the fitted model of Equation 19. Figure 2 is a plot of $\ln(KD)$ against D. These plots can be used to compare actual variation in FER in a specific case against the behavior predicted by Equation 19.

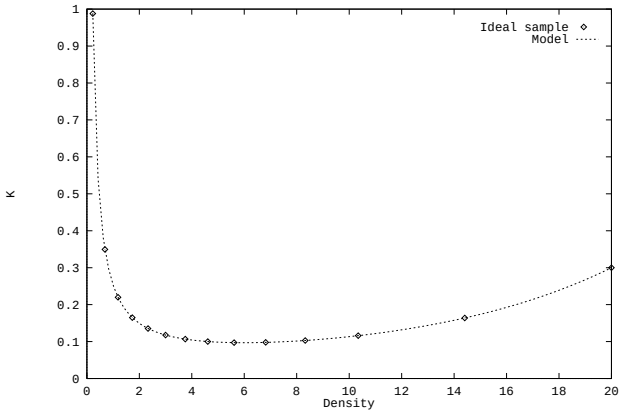

Figure 1: FER against D for an ideal case

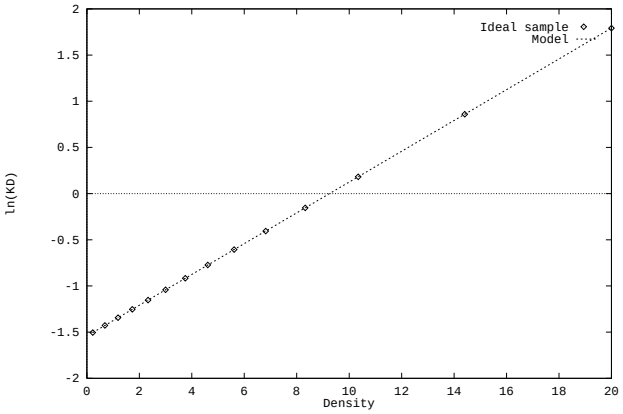

Figure 2: ln(KD) against D for an ideal case

3 Estimation of Fault Exposure Ratio

Adams [13] noticed that software's failure rates in operational phase had a distribution which observes Zipf's law, *the failure rate of a fault i is inversely proportional to a power of i, when faults are ranked by decreasing failure rate* [15]. Trachtenberg [15] proposed a software reliability model based on Zipf's law which fitted Adam's reliability data with an very high correlation coefficient.

If we regard the testing phase as a compressed form of operational use [1], then the failure rates for different defects during the testing phase may also have similar distribution for different software projects. Further we

suspect that at the beginning of system testing phase, the detectability profiles [3] of software projects may have similar shapes in accordance with the Zipf's law.

If the testing activity is conducted in the same way, FER would be primarily determined by the detectability profile. In case of truly random testing, FER would be an weighted average of the detectability of each individual fault [3].

Here we will obtain a form of Equation 19 that will relate the FER to the initial defect density. The exponential model assumes that FER does not change over time. The exponential model can be considered to be an approximation of the logarithmic model. Let us define the overall *exponential model based FER (EFER)*, represented by $\hat{K}$, as the value obtained by fitting the exponential model to the test data. If we would calculate the EFER during the test phase using the exponential model, we should expect it to vary with the defect density present at the beginning i.e. at t=0. Here we obtain a mapping of our model described by Equation 19 on to the exponential model that relates the average defect density and EFER.

Let us assume that Equation 19 is applicable when when instead of instantaneous values K and D, we use their average values $\hat{K}$ and $\hat{D}$, averaged over the test duration. Then we can write:

$$\hat{K} = \frac{\alpha_0}{\hat{D}} e^{\alpha_1 \hat{D}} \tag{20}$$

In Musa et al [1], the values of EFER were computed assuming the exponential model. According to the exponential model, from Equation 1,

$$N_0 - \mu(t) = \beta_0^E e^{-\beta_1^E t} \tag{21}$$

or,

$$N(t) = \beta_0^E e^{-\beta_1^E t} \tag{22}$$

Since $N(t)/I_s = D(t)$, $N_0/I_s = D_0$, we have

$$D(t) = D_0 e^{-\beta_1^E t} \tag{23}$$

Let us assume that t varies from 0 to T such that $D(T) = D_0/10$, then

$$T = -\frac{1}{\beta_1^E} \ln(0.1) = \frac{2.303}{\beta_1^E} \tag{24}$$

Now $\hat{D}$, the average value of $D(t)$ from 0 to T is given by

$$\hat{D} = \frac{1}{T} D_0 \int_0^T e^{-\beta_1^E t} dt = \frac{D_0}{\beta_1^E T} (1 - e^{-\beta_1^E T}) \tag{25}$$

Substituting using Equation 24, we get

$$\hat{D} = \frac{D_0}{2.303}(1 - 0.1) = 0.39 D_0 \tag{26}$$

Since Equation 26 is a simple linear expression, D_0 may be used instead of $\hat{D}$ in an empirical analysis.

This allows us to rewrite Equation 20 as

$$\hat{K} = \frac{\alpha_0}{0.39 D_0} e^{0.39 \alpha_1 D_0} \tag{27}$$

renaming the parameters we get,

$$\hat{K} = \frac{\alpha_0^{'}}{D_0} e^{\alpha_1^{'} D_0} \tag{28}$$

In Equation 19 and Equation 28, the parameters α_0 and $\alpha_0^{'}$ represent the scale factors for K and $\hat{K}$ respectively. To see the significance of α_1 or $\alpha_1^{'}$, let us solve for the minimum value of K. Taking the derivative of the RHS of Equation 19 and equating it to zero, we get,

$$\begin{aligned} \frac{\partial K}{\partial D} &= \alpha_0\left(\frac{\alpha_1 e^{\alpha_1 D}}{D} - \frac{e^{-\alpha_1 D}}{D^2}\right) \\ &= 0 \\ yielding \;\; D &= \frac{1}{\alpha_1} \end{aligned} \tag{29}$$

Substituting this in Equation 19, we get

$$K_{min} = \alpha_0 \alpha_1 e \tag{30}$$

Thus $1/\alpha_1$ gives the minimum of the curve. The available data suggests that it occurs at about $D_0 = 5$. This corresponds to the minimum at approximately $D = 5 \times 0.39 = 2$. Further studies are needed to see if the two parameters vary with variation in the testing practices.

Table 1, obtained using [1], relates initial defect density and overall fault exposure ratio. Here 10% of the total initial defects are assumed to be present by the end of the testing phase.

Table 1: $\hat{K}$ vs. D_0 [1]
($\hat{K}$ in units of 10^{-7})

Data	Size (K)	D_0	$\hat{K}$
T1	21.7	6.89	1.87
T2	27.7	2.14	2.15
T3	23.4	1.79	4.11
T4	33.5	1.74	10.6
T5	2445	0.374	4.2
T6	5.7	14.08	3.97
T16	126.1	0.357	3.03
T19	61.9	0.675	4.54
T20	115.35	20.89	6.5

Using the form as given in Equation 16 and using the data given in Table 1, we get the following estimates of the parameters $\alpha_0^{'}$ and $\alpha_1^{'}$:

$$\alpha_0^{'} = 3.058 \times 10^{-7} \tag{31}$$

and,

$$\alpha_1^{'} = 0.195 \tag{32}$$

when the defect density is expressed in terms of per one thousand lines of object code. The correlation coefficient value was 0.89 which indicates that the Equation 28 can satisfactorily describes the relation between $\hat{K}$ and D_0. If $\hat{D}$ was used in place of D_0, we would get $\alpha_0 = 1.205 \times 10^{-7}$ and $\alpha_1 = 0.492$ using Equations 20 and 26.

Figure 3 depicts the relation between $EFER$ and D_0 for both the real data and the model with parameters evaluated from the real data. Each point represents a different data set. Figure 4 depicts $ln(\hat{K}D_0)$ against D_0. Note that the lower density values correspond to the later parts of testing.

Both the data analyzed and our model suggest that until a defect density value of about 5, the EFER declines gradually and then at lower defect densities starts rising.

The data sets in Table 1 were collected at AT&T. High quality data sets that are available in the literature are still limited. In [4], several other data sets were examined. When the variation of K within one AT&T data

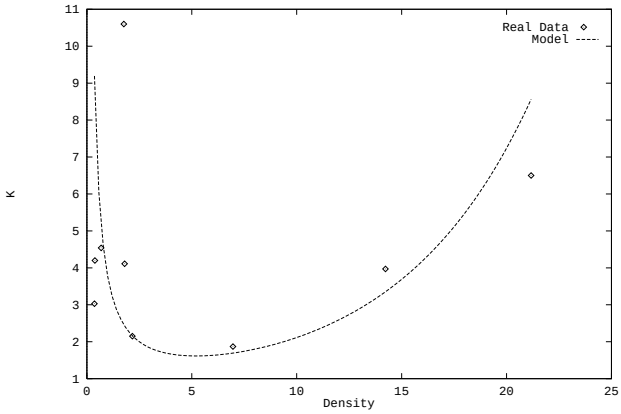

Figure 3: EFER against D_0: model compared with the real data

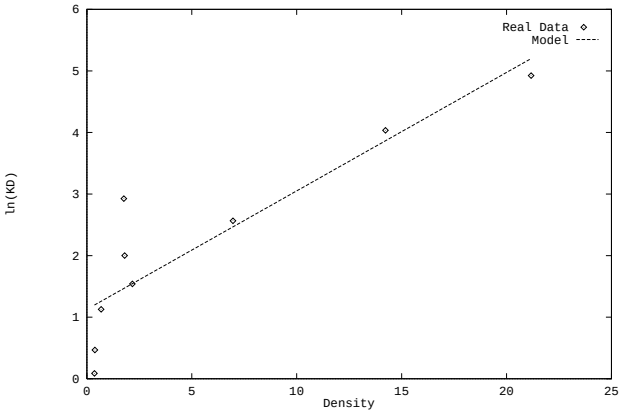

Figure 4: $ln(\hat{K}D_0)$ against D_0: model compared with the real data

set was examined, it showed a minimum at density value 2. Three data sets from IBM with D_0 values of 0.11, 0.07 and 0.32 all exhibited a rising value of K and one data set with D_0 equal to 8.77 showed a minimum at about 3. Two data sets from Japanese industry with D_0 values equal to 0.66 and 0.17 showed K rising. These are in confirmation with the model presented here. One AT&T data set did not match out model. The fact that the logarithmic SRGM has good predictive capability, also supports our model indirectly, since it leads to the logarithmic SRGM.

If more failure data are available, the value of α_0 and α_1 can be determined more exactly.

It should be noted that the defect density shown in Table 1 was calculated using the object instruction count. It might be possible to get even better correlation between FER and D_0 if we consider only the projects using high-level languages, or only the projects which used assembly languages.

4 Fault Exposure Ratio in Software Reliability Engineering

FER plays an important role in software reliability growth. A manager can use it to plan the test resources need to achieve the desired quality level, even before testing begins. In the early stages of testing, only a limited number of data points are available, which are not often enough to establish the long term trend. This makes parameter estimation for SRGMs unstable. The FER can be used to stabilize the early projections. Use of coverage information for reliability is just emerging. For coverage based modeling, use of FER can be used for initial estimates of the parameters involved.

4.1 A priori planning

It is relatively straightforward to use FER to estimate the parameters of the exponential model, as illustrated in this example.

Example 1: Assume there is a software system whose initial defect density is estimated to be 16 defects per thousand lines of source code. It has 50,000 lines of source code. The machine to be used for testing has MIPS rating of 16. The source to object instruction ratio is 4. Here we will calculate the total expected testing CPU time needed to achieve a failure intensity objective of 2×10^{-4}.

The initial defect density D_0 is

$$D_0 = 16/4 = 4 \; per \; thousand \; object \; lines \tag{33}$$

The total expected number of faults is

$$\beta_0^E = 50 \times 16 = 800 \tag{34}$$

The average fault exposure ratio EFER is

$$\frac{\alpha_0^{'}}{4} e^{\alpha_1^{'} \times 4} = \frac{3.089 \times 10^{-7}}{4} e^{0.192 \times 4} = 1.66 \times 10^{-7} \tag{35}$$

The linear execution time can be estimated as,

$$T_L = \frac{I_s Q_r}{r} = \frac{50000 \times 4}{16000000} = 0.0125 \tag{36}$$

The per fault hazard rate can be obtained as,

$$\beta_1^E = \frac{\hat{K}}{T_L} = \frac{4.167 \times 10^{-7}}{0.0125} = 1.33 \times 10^{-5} \tag{37}$$

Thus we have the following exponential model describing the failure process during system testing,

$$\mu(t) = 800(1 - e^{-0.0000133t}) \tag{38}$$

and,

$$\lambda(t) = 0.0267 e^{-0.0000133t} \tag{39}$$

where t is measured in CPU seconds. Solving this for the failure intensity objective, we get $t = 82.95$ CPU hours. By the time testing is terminated, there will be about 785 failures encountered. □

Notice that this is obtained before real system testing starts. When enough actual failure data from system testing phase is available, one might consider to use real data and the logarithmic model to get a more accurate projection.

A possible procedure to estimate the parameters of the logarithmic model can be based on the results obtained in [3]. A model relating FER with time as described by Equation 40 was proposed which characterizes the variation of FER with time t.

$$K(t) = \frac{K_0 N(0)}{N(t)(1 + at)} \tag{40}$$

where a is a parameter depending on the "detectability profile" of the software [3].

From this model we can derive the well-known logarithmic software reliability growth model with the following interpretation for the parameters:

$$\beta_0^L = \frac{K_0 N_0}{a T_L} \tag{41}$$

$$\beta_1^L = a \tag{42}$$

The problem of estimating the parameter a need to be further investigated.

An alternative way to estimate the parameters of the logarithmic model is to first estimate the parameters of the exponential model, and then use them to obtain the parameters of the logarithmic model. An approximate procedure is given in [5]

4.2 Stabilizing parameters in early test stages

When failure intensity is plotted against time, it shows a lot of noise superimposed over the long term trend. When sufficient number of data points are available, we can extract the long-term trend as described by a suitable SRGM. However at the beginning of the test phase, only a few data points are available. Any projections based on these are significantly influence by noise. It would be very desirable for a manager to be able to plan the rest of the test phase based on early data, and thus we need techniques to stabilize parameter estimation.

Stabilization can use a combination of static metrics with actual test data.

When the exponential model is being used, the following possible stabilization technique can be used. Let us assume that we are at the stage where no clear reliability growth trend (i.e. decline in failure intensity) is yet apparent.

1. Obtain average of failure intensity. This provides an estimate of $\lambda_0 = \beta_0^E \beta_1^E$.
2. Empirically estimate FER and hence β_1^E.
3. Obtain an estimate for β_0^E as λ_0 / β_1^E.

For the logarithmic model, we can use results from [3], where the parameter β_0^L was related to initial fault exposure ratio K_0 and β_1^L. If we can estimate K_0, the initial failure intensity λ_0 can be evaluated through,

$$\lambda_0 = \frac{K_0 N_0}{T_L} \tag{43}$$

Let $t = 0$ in Equation 5,

$$\lambda_0 = \beta_0^L \beta_1^L \tag{44}$$

Thus Equation 4 can be rewritten as,

$$\mu(t) = \frac{\lambda_0}{\beta_1^L} \ln(1 + \beta_1^L t) \tag{45}$$

Thus if we can estimate K_0, the logarithmic model will have only one unknown parameter β_1^L. This can be used to stabilize the use of the logarithmic model in the very early phases of testing.

4.3 Coverage based modeling

In [17], a model is presented that computes the *defect coverage* C^0 in terms of a software test coverage measure C^i which may be one of *block coverage*, *branch coverage*, *p-use coverage* etc.

$$\begin{aligned} C^0 &= p_0^i \ln[1 + p_1^i (exp(p_2^i C^i) - 1)] \\ & \quad i = 1 \quad to \quad 4 \qquad C^0 \leq 1 \end{aligned} \tag{46}$$

It was shown that this model confirms with experimental data available.

Equation 46 gives us a three-parameter model for defect coverage in terms of a measurable test coverage metric. It is possible to approximate Equation 46 by a linear relation, but it would be valid for only a small range. Because three parameters are involved, the fitted values of the parameters can be very sensitive to the initial values.

This approach models coverage of an enumerable (like a branch or a p-use) just like coverage of a defect. The superscript 0 indicates defects and superscript i indicates one of the test enumerables. The first parameter of Equation 46 then is [17],

$$p_0^i = \frac{K^0(0)}{a^0 T_L} \tag{47}$$

Estimation of a^i remains an open problem. The third parameter is given by,

$$p_2^i = \frac{a^i T_L}{K^i(0)} \tag{48}$$

This allows the possibility of empirically estimating two of the parameters. These values then can be used as initial estimates. This reduces the problem of obtaining parameter values that are stable, i.e. do not depend greatly on the initial values assumed when numerical curve fitting is done.

5 Concluding Remarks

We have presented an empirical model which allows us to estimate the fault exposure ratio and hence the parameter β_1^E of the exponential model. Since estimation of β_0^E can already be done satisfactorily, we can now use the exponential model before testing begins.

The fault exposure ratio FER describes the effectiveness of the testing process. Besides the defect density, FER may also depends on the testing strategy and perhaps the individual software structure. These may vary from project to project. However, within the same organization, these might not very significantly for different projects and thus FER may be estimated directly from D. More accurate estimation of FER can be done if the parameters α_0 and α_1 are obtained by fitting Equation 19 to the data from similar projects. When there is no previous data available within an organization, $\hat{K}$ can be estimated using parameter values for $\alpha_0^{'}$ and $\alpha_1^{'}$ from a variety of projects from other organizations.

Just as size can be used to estimate the number of total expected defects fairly accurately, the model here provides an estimate of K or $\hat{K}$ using D_0. Further research is needed to identify and quantify the effect of other factors so that K or $\hat{K}$ may be estimated more accurately, just as the frequency of specification changes, etc. can enhance the accuracy of estimating the total number of defects [9].

The model can be refined further when additional data is available. If there is data available to relate K to D, then we can estimate $\hat{K}$ and hence the parameter λ_0 of the logarithmic model (ref. Equation 45) at the beginning of system testing phase. Estimation of the other parameter β_1^L for the logarithmic model requires further investigation.

We have also presented approaches for using FER for enhancing the accuracy of software reliability analysis techniques. These approaches promise better accuracy in both time based and coverage based modeling.

References

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

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

[3] Y. K. Malaiya, A. von Mayrhauser and P. K. Srimani, The Nature of Fault Exposure Ratio, Proc. International Symposium on Software Reliability Engineering, October 1992, pp. 23-32.

[4] Y. K. Malaiya, A. von Mayrhauser and P. K. Srimani, An Examination of Fault Exposure Ratio, IEEE Trans. Software Engineering, Nov. 1993, pp. 1087-1094.

[5] Y. K. Malaiya, Early Characterization of the Defect Removal Process, Proc. 9th Annual Software Reliability Symposium, May 1991, pp. 6.1-6.4.

[6] Y. K. Malaiya, A. von Mayrhauser and P.K. Srimani, The Constant Per Fault Hazard Rate Assumption, Proc. 2nd Bellcore/Purdue Workshop on Issues in Software Reliability Estimation, October, 1992, pp. 1-9.

[7] Y. K. Malaiya, N. Karunanithi and P. Verma, Predictability of Software Reliability Models, IEEE Trans. Reliability, December 1992, pp. 539-546.

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

[9] M. Takahashi and Y. Kamayachi, An Empirical Study of a Model for Program Error Prediction, in Software Reliability Models, IEEE Computer Society, 1991. pp. 71-77.

[10] T. M. Khoshgoftar and J. C. Munson, The Line of Code Metric as a Predictor of Program Faults: a Critical Analysis, Proc. COMPSAC'90, pp. 408-413.

[11] A. von Mayrhauser and J. A. Teresinki, The Effects of Static Code Metrics on Dynamic Software Reliability Models, Proc. of Symposium on Software Reliability Engineering, April, 1990, pp. 19.1-19.13.

[12] 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.

[13] E. N. Adams, Optimizing Preventive Service of Software Products, IBM Journal of Research and Development, vol. 28, no. 1, January 1984, pp.2-14.

[14] W. H. Farr, A survey of Software Reliability Modeling and Estimation, Naval Surface Weapons Center, TR 82-171, Sept. 1983.

[15] M. Trachtenberg, Why Failure Rates Observe Zipf's Law in Operational Software, IEEE Trans. Reliability, vol. 41, no. 3, September 1992, pp. 386-389.

[16] N. Li and Y.K. Malaiya, ROBUST: A Next Generation Software Reliability Engineering Tool, Proc. IEEE Int. Symp. on Software Reliability Engineering, pp. 375-380, Oct. 1995.

[17] Y.K. Malaiya, N. Li, J. Bieman, R. Karcich and B. Skibbe, The Relationship between Test Coverage and Reliability, Proc. Int. Symp. Software Reliability Engineering, Nov. 1994, pp.186-195.