Rescaling Reliability Bounds for a New Operational Profile

Peter G Bishop
Centre for Software Reliability and Adelard
City University, Northampton Square
London, EC1V 0HB, UK
+44 (0)20-7490-9467
pgb@csr.city.ac.uk, pgb@adelard.co.uk

ABSTRACT
One of the main problems with reliability testing and prediction is that the result is specific to a particular operational profile. This paper extends an earlier reliability theory for computing a worst case reliability bound. The extended theory derives a re-scaled reliability bound based on the change in execution rates of the code segments in the program. In some cases it is possible to derive a maximum failure rate bound that applies to *any* change in the profile. It also predicts that (in principle) a "fair" test profile can be derived where the reliability bounds are relatively insensitive to the operational profile. In addition the theory allows unit and module test coverage measures to be incorporated into an operational reliability bound prediction. The implications of the theory are discussed, and the theory is evaluated by applying it to two example programs with known faults.

Categories and Subject Descriptors
D.2.4 [**Software/Program Verification**]: reliability, statistical methods; D.2.5 [**Testing and Debugging**] coverage testing

General Terms
Reliability, Theory, Experimentation

Keywords
Operational profile, operational reliability testing, worst case bounds.

1. INTRODUCTION
One of the main problems with reliability testing is that the result is specific to a *particular operational profile*. This paper extends an earlier worst case bound reliability theory to allow the bound to be re-scaled for a new operational profile. The paper will first show that the original worst case bound theory (for continuous time) also applies to discrete tests, and then present the extensions of the theory for re-scaling the bound to a new operational profile.

ISSTA '02, July 22-24, 2002, Rome, Italy

The implications of the theory are discussed, and the theory is evaluated by applying it to example programs with known sets of faults.

2. WORST CASE BOUND THEORY
The observed reliability of a system containing design faults is based on three main factors:

- the number of faults
- the size and location of faults
- the input distribution (operational profile)

This is illustrated in figure 1 below.

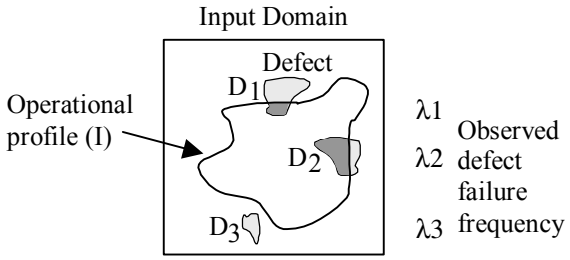

Figure 1. Operational profile and software failure rates

While there are many methods for estimating the likely number of software defects *N,* there is no way to establish the failure frequencies λ_1 .. λ_N for unknown software defects under a given input distribution *I*. However the theory developed in [2] can place a worst case bound on the failure rate for all the defects based on the amount of usage time. The theory makes the relatively standard reliability modelling assumptions that:

a) removing a fault does not affect the failure rates of the remaining faults

b) the random failure frequencies of the faults can be represented by λ_1 .. λ_N, which do not change with time (i.e. the input distribution *I* is stable)

c) any fault exhibiting a failure will be detected and corrected immediately

The basic idea behind the model is very simple; once the software has been operating for some time, faults with the highest failure frequencies are likely to be removed, while faults with low failure frequencies only make a small contribution to the residual software failure frequency. Thus for a given number of test executions, *T*, there is a worst case λ which maximises the probability of the failure on the next test.

Put more formally, using the assumptions given above, for a defect i with a probability of failure per program test of λ_i, the probability of observing a failure after T prior tests is:

$$P(T_i = T+1 \mid \lambda_i) = \lambda_i (1-\lambda_i)^T \quad (1)$$

where T_i is the test where the defect is detected (and removed).

Differentiating with respect to λ_i, it can be shown that the maximum failure probability after T tests occurs when:

$$\lambda_i = \frac{1}{T+1}$$

Substituting back into (1) and rearranging, the upper bound on the probability of failure after T tests is:

$$P(T_i = T+1) \le \frac{1}{T}\left(1-\frac{1}{T+1}\right)^{T+1} \quad (2)$$

There is a standard result that $\left(1+\frac{x}{n}\right)^n \to e^x$ as $n \to \infty$.

As the e^x asymptote is an upper bound, it follows that, for all T:

$$P(T_i = T+1) < \frac{e^{-1}}{T}$$

If we define $\theta_i(T)$ as $P(T_i=T+1)$, the probability of failure per test after T tests, then:

$$\theta_i(T) < \frac{1}{eT} \quad (3)$$

This bound is *independent of the defect failure probability*. Figure 2 below illustrates this independence for different values of λ_i.

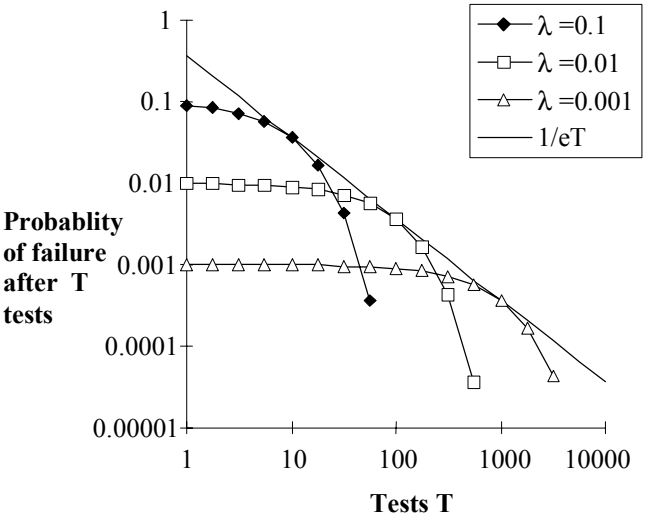

Figure 2. Illustration of the worst case bound

It is clear that, regardless of the value of λ_i, the probability of failure per test after T tests, $\theta_i(T)$, is bounded by $1/eT$.

In the worst case, the failures of all faults are disjoint, so we can sum the bounds for all N faults to derive a worst case bound for the probability of program failure after T tests, $\theta(T)$, i.e.:

$$\theta(T) < \frac{N}{eT} \quad (4)$$

This is the discrete equivalent of the continuous time bound equation in [2] where the bound on failures per unit time after a test time t is shown to be exactly N/et (or alternatively the bound on the MTTF after a test time of t, MTTF(t), is exactly et/N).

The result is surprising because it permits long-term reliability predictions to be made at t=0. If the model assumptions apply and we can estimate the number of faults N at the time of release (e.g. using estimation methods such as [3], [4], [7], [9]) the reliability growth can be bounded at *any time in the future*. Note that the theory does not tell us when (or even if) the faults will be found, but it does set a quantitative bound on the probability of program failure after testing and this bound always decreases with increasing tests (or operating time).

Figure 3 shows the application of the theory to the field reliability data of a commercial teleswitch [5]. The reliability bound is based on an estimate for N that is 50% more than the number of detected faults reported in [5]. The bars represent the measurement uncertainty in taking the average of several times to failure to estimate the current MTTF.

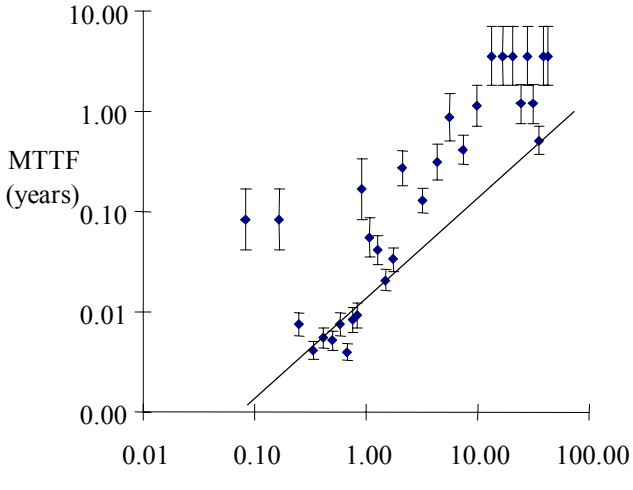

Figure 3. Reliability growth of a commercial teleswitch

The effect of violating theory assumptions (a) and (c) were examined in [2]. Both correction-induced faults and variations in the operational profile can alter the bound and cause a "saw-tooth" pattern of failure rates. It was however also shown that the worst bound should still hold over time periods that are longer than the short term fluctuations caused by changes in operational profile.

However the theory cannot predict what the reliability bound will be after a major change in the input profile (e.g. from development testing to operation). For example, in figure 3 above, there is a sudden drop in reliability after 0.1 years of usage. This might be due to a transition from field trials to full operational use—a point where there could be a dramatic change in the operational profile and where the standard reliability bound

prediction does not apply. The remainder of this paper describes an extension to the worst case bound theory that allows the reliability bound to be "re-scaled" to predict the effect of a transition to a different operational profile.

Note that this extension to the theory predicts the change in reliability bound immediately after the change in profile (with no fault correction). If detected faults are corrected immediately in subsequent operation then, given sufficient usage time with an unchanged profile, the bound will eventually fall back to the original bound of *N/et*.

3. EXTENDED THEORY

The extended theory makes some additional assumptions:

1) Each fault is localised to a single code segment, i.e. is in a "basic block".
2) The externally observed failure rate is proportional to the execution rate of the faulty code segment.
3) There is a constant probability of a fault existing in any line of executable code.
4) The operational profile *I* can be characterised by the distribution of code segment executions in a program *Q*.

The final assumption requires some further explanation. The operational profile *I* is characterised by a sequence of input values to the program (e.g. one input vector per test execution). Given sufficient program test cycles, we assume that an input profile *I* results in a specific profile of execution rates *Q* over the program segment, i.e. $Q = \{q(1), q(2), ...\}$ for the segments in the program. Note that $q(j)$ can exceed unity as segments inside subroutines and loops can be executed many times per test.

3.1 Scaling the bound for a different execution rate profile

From equation (4) we have shown that the failure probability per test is bounded by:

$$\theta_i(T) < \frac{1}{eT}$$

If fault *i* is located in a single segment *j* (assumption 1) then the faulty code will be executed $q(j)$ times per test. If we assume the failure probability of the faulty code per test is proportional to $q(j)$ (assumption 2), then for a new execution profile Q' in subsequent operation, the failure probability per test under the new profile becomes:

$$\theta'_i(T) < \frac{q'(j)}{q(j)} \frac{1}{eT} \qquad (5)$$

This assumption about scaling is open to challenge. It implies that if $q(j)$ is unchanged then λ_j cannot change. However λ_j can change without changing $q(j)$ if failure-causing values are increased and failure-avoiding values are decreased. Assumption (4) implies that this does not occur.

In the worst case, the failure probability increases linearly with $q'(j)$ if all failures are disjoint, but there is a limit—the failure probability of a fault cannot exceed unity, i.e. strictly:

$$\theta'_i(T) < \min\left(1, \frac{q'(j)}{q(j)} \frac{1}{eT}\right)$$

If this occurs, then equation (5) over-estimates the effect of increasing $q'(j)$, so the linear scaling assumption is conservative. A similar issue arises when the same segment is executed several times per program test (e.g. in a loop). Multiple segment failures in the same test will not increase the observed failures, i.e. the failures are not disjoint. Again, the assumption of linearity over-estimates the scaling effect, so equation (5) is conservative.

If we assume that the probability of a fault per line is constant (assumption 3), the probability that a fault is located in segment *j* is:

$$P(fault)_j = L(j)/L \qquad (6)$$

where $L(j)$ is the length of the basic block of code, and *L* is the length of all blocks in the program, $\Sigma L(j)$. Using this assumption, we can compute a mean value for the re-scaled bound of a program fault under Q':

$$\overline{\theta}_i'(T) < \sum \frac{q'(j) \cdot L(j)}{q(j) \cdot L} \cdot \frac{1}{eT}$$

In the worst case, the failures of the *N* individual faults in the program are disjoint, so the rescaled bound on the failure probability of the whole program is:

$$\overline{\theta}'(T) < \sum \frac{q'(j) \cdot L(j)}{q(j) \cdot L} \cdot \frac{N}{eT} \qquad (7)$$

The original bound under the test profile Q, was:

$$\theta(T) < \frac{N}{eT}$$

So the mean scale-up factor (S) for the bound under the new profile Q' relative to the bound under the test profile *Q* is:

$$S(Q', Q) = \overline{\theta}'(T) / \theta(T)$$

Hence:

$$S(Q', Q) = \sum \frac{q'(j)}{q(j)} \cdot \frac{L(j)}{L} \qquad (8)$$

From equation (8) it follows that the scale factor is always unity when $Q'=Q$, and in general, overall scale-factor is simply the length-weighted mean of the scale factors for individual segment executions.

3.2 Impact on failure rate bounds

What are the implications of this? Let us take a simple example: a one level binary tree where there are 10 lines of code in each segment, i.e. $L(j)=10$. This tree has three segments: a root segment and two branch segments. We will now define a set of execution rates *Q* for the segments in the table below.

Table 1. Execution rates (even branch split)

Segment *j*	L(*j*)	q(*j*)
0 (root)	10	1
1 (branch)	10	0.5
2 (branch)	10	0.5

Now let us define an alternative set of execution rates Q'.

Table 2. Execution scale factors (uneven branch split)

Segment *j*	L(*j*)	q′(*j*)
0 (root)	10	1
1 (branch)	10	0.99
2 (branch)	10	0.01

In the following table we compute the scale-up terms, $L(j)\cdot q'(j)/q(j)$, according to equation (8) for two cases where:

- the test and operation profiles are the same (test and operation both equal to Q)
- the test profile is Q and the operation profile is Q'

Table 3. Comparison of scale factors (balanced test profile)

Segment j	Test q(j)	q′(j)	Operation using Q L(j).q(j)/q(j)	Operation using Q' L(j).q′(j)/q(j)
Root 0	1	1	10	10
Branch 1	0.5	0.99	10	19.8
Branch 2	0.5	0.01	10	0.2
Sum			30	30
S =Sum/L			1	1

The final scale factor S is derived by dividing the length-weighted scale-factor by the program length L. As expected, if we use the same profile for test and operation the scale-up factor is unity, i.e. we get the same bound for the failure probability.

Perhaps more surprisingly, there is *no increase* in the bound when a different execution rate profile is used. This occurs if the program is "fairly" tested so that the scale-up of the mean failure bound in one branch is exactly matched by the scale-down in the other branch. Assume for example that there are two alternative branch segments j and k with segment execution rates $q(j)$ and $q(k)$, if we change the profile so that all the execution flows down j then $q'(j) = q(j) + q(k)$ and $q'(k)=0$. In the segments were "fairly" tested under Q, then the re-scaled bounds will be identical to the original ones. This will occur when:

$$\frac{q(j)+q(k)}{q(j)}L(j) = L(j) + L(k)$$

Rearranging this occurs when:

$$\frac{q(j)}{q(k)} = \frac{L(j)}{L(k)} \qquad (9)$$

This condition is satisfied in the example shown in Table 3 as the branch lengths and execution rates are equal.

Now let us look at the converse situation where the test profile is very uneven. Let us suppose that the asymmetric Q' was used for testing. We can compute the scale factors when using profiles Q' and Q in subsequent operation, as shown in the following table.

Table 4. Comparison of scale factors (uneven test profile)

Segment j	Test profile q′(j)	q(j)	Operation under Q′ L(j)q′(j)/q′(j)	Operation under Q L(j)q(j)/q′(j)
Root 0	1	1	10	10.0
Branch 1	0.99	0.5	10	5.1
Branch 2	0.01	0.5	10	500
Sum			30	515.1
S = Sum/L			1	17.2

Using the same profile for test and operation still results in a scale factor of unity, but the scale factor increases markedly when a different profile is used in operation (i.e. 17 rather than 1). The Q' profile is very "unfair" test profile. The ratio of branch execution rates is 99 while the ratio of the branch lengths is unity. If the low usage branch becomes more heavily used under the new profile, this increases the maximum failure rate bound for the overall "program" of three branches.

In the converse case where a well-tested branch becomes more heavily used in operation, the scale factor can be *less* than unity. For example, if Q had branch execution rates of 0.3 and 0.7 and Q' had execution rates of 0.0 and 1.0, it can be shown that the scale factor is 0.81.

Let us now apply this idea of "fair testing" to a more complex program flow structure. For a program graph with multiple branches, the execution rates should be apportioned to the length of code "dominated" by that branch, i.e. include sub-branches that can only be executed from that branch. This illustrated in the figure 4 below.

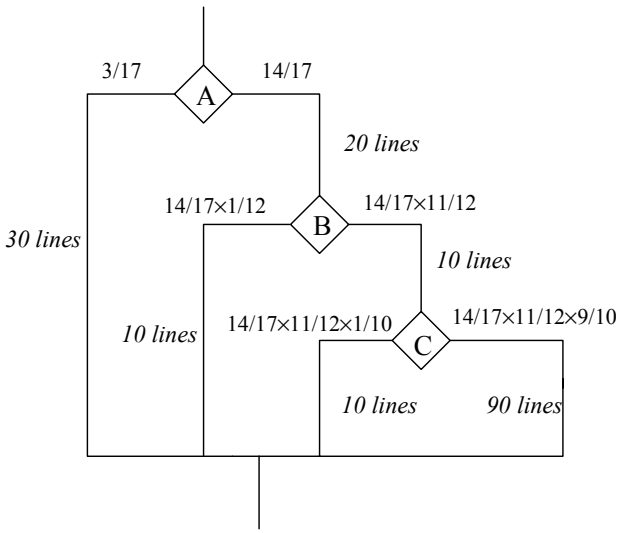

Figure 4. Fair test profile example

Node A dominates 30 lines of code in the lefthand branch and 140 lines in the righthand branch, so the execution rate is split in the ratio 30:140, i.e. 3/17 to the left and 14/17 to the right. At node B, the ratios of code dominated are 10:110 so the input execution rate of 14/17 is further split (1/12 to the left, 11/12 to the right). Finally at node C, the ratio is 10:90, the execution rate at node C is further split (1/10 to the left, 9/10 to the right).

Let us now compute the scale up factors for some different execution profiles, namely:

- Shortest path — all the executions flow down A.left
- Longest path — all the executions flow down A.right, B.right, C.right
- Even—execution rate split evenly at each node

The execution rates of the test profile and new profiles are shown in the table below.

Table 5. Fair test profile and alternative profiles

Code segment (j)	$q(j)$	$q'(j)$		
	fair	shortest	longest	even
A.left	0.176471	1	0	0.5
A.right	0.823529	0	1	0.5
B.left	0.068627	0	0	0.25
B.right	0.754902	0	1	0.25
C.left	0.07549	0	0	0.125
C.right	0.679412	0	1	0.125

The scale factors computed for the new execution profiles are shown below.

Table 6. Scale factors for alternative profiles

Segment (j)	q′(j)/q(j)*L(j)		
	shortest	longest	even
A.left	170	0	85.00
A.right	0	24.29	12.14
B.left	0	0	36.43
B.right	0	13.25	3.31
C.left	0	0	16.56
C.right	0	132.47	16.56
Sum	170	170	170
S (Sum/L)	1	1	1

This demonstrates that fair testing is theoretically possible for an arbitrary branch structure, but unfortunately, it is unlikely to be possible for programs in general. Equation (9) does not explicitly address cases where the segment is in a subroutine or a loop. In addition, fair apportionment may be prevented because the branch decisions may not be independent. These issues are discussed below.

3.2.1 Impact of loops

Increasing the number of loop iterations in the Q' profile will increase $q'(j)$ in all segments within the loop, and hence increase the failure intensity, so the scale factor cannot be unity. "Fair testing" is only feasible if *all* program loops have a fixed number of iterations, so that q(j) and q'(j) loop segments are scaled by the same factor and unit scale factor can be preserved under a different profile. On the other hand, it may be possible to define a "nearly fair" test profile where loops are executed the maximum number of times. In this case, the scale factor for a a different profile could be less than unity.′

3.2.2 Impact of subroutines

In the case of calls to a subroutine, substituting the length of the subroutine L_{sub} into the flow graph branch is not appropriate as the mean failure probability bound for the subroutine is associated with the *total* execution rate of the subroutine over all calls. The bound for all calls is:

$$\bar{\theta}_{sub} < \frac{L_{sub}}{L}\frac{N}{eT}$$

If a particular program segment calls a subroutine at a rate q_{call}, under the scaling assumption the mean bound on the failure probability for that call statement in the segment is:

$$\bar{\theta}_{call}(T) < \frac{q_{call}}{q_{sub}}\frac{L_{sub}}{L}\frac{N}{eT}$$

where q_{sub} is the total number of executions of the subroutine per program test. Hence the failure probability per test of the subroutine has to be apportioned between the call statements, i.e. the length substituted into the flow graph is:

$$L_{sub}\, q_{call}/q_{sub}$$

But as $q_{sub} = \Sigma q_{call}$ over all segments that call the subroutine, a fair apportionment of execution rates is difficult. Any change in q_{call} in one segment will affect q_{sub} and hence "unbalance" the apportionments in all other segments that call the routine. One possible approximation for including the effect of subroutines when calculating a balanced profile is to use L_{sub}/n_{call} for the length at each call, where n_{call} is the number of different calls to the subroutine that exist in the source code. This allows the rates to be apportioned independently and will be reasonably accurate if all q_{call} rates are similar.

3.2.3 Dependency between decision nodes

Even if some algorithm can be devised that can compute a balanced execution profile for a complete program, it is unlikely that a test profile can be constructed that can achieve it, because there are dependencies between decision nodes. Take the following C code example:

```
if (a > 10) action1a() else action1b();
if (a > 20) action2a() else action2b();
```

The dominated branches from (a > 10) are action1a() and action1b(). Selecting values of a to ensure the first decision node is balanced may make it difficult to ensure the dominated branches of next decision node are balanced. If for example, a balanced profile for the first decision node requires (a > 10) to be true 50% of the time and (a > 20) to be true 90% of the time, we have a contradiction as (a >10) is always true when (a > 20).

The same problem can exist when there are dependencies between subroutines.

3.3 Incorporating additional test data

While it may be infeasible to devise a balanced profile by testing the whole program, unit and module test can improve the coverage of infrequently used code segments and hence improve the "fairness" of the execution profile. Assuming that segment j has been executed x(j) times in module testing and $q(j)\cdot T$ times in program testing under profile Q, it can be shown that the scale factor under a new profile Q′ is:

$$S(Q',Q,x) = \sum \frac{L(j)}{L} \cdot \frac{q'(j)}{q(j) + x(j)/T} \qquad (10)$$

It can be seen that equation (10) is a simple extension of equation (8) that includes an extra term for module test coverage. It can be seen that extra tests always reduce the reliability bound, but the maximum improvement is achieved where $q(j)$ is small.

3.4 Estimating the maximum scale factor relative to a test profile

The previous analysis indicates that it is theoretically possible to design "fair" test profiles where the worst case failure probability is insensitive to changes in the profile in subsequent operation. However this only applies to very artificial program structures, and it is unlikely that a perfectly balanced profile can be achieved. The situation could be improved if individual subroutines are tested independently at the module test stage. These results could be incorporated into the integrated test profile using equation (10) to make it more "fair". But even with such improvements, it may be impractical to achieve a fair test profile as we still have to allow for subroutines and loops, so it would be useful if we could derive the worst-case scale-up for an arbitrary test profile. We have shown in equation (8) that the scale-up of reliability bound when using an operational profile Q′ rather than a test profile Q is:

$$S(Q',Q) = \sum \frac{L(j)}{L} \cdot \frac{q'(j)}{q(j)}$$

We know that, for each code segment, there is some maximum execution rate, $q(j)_{max}$. For non-looping unconditional code segments, the code segment can only be executed once per program test, so $q(j)_{\max}$ is unity. For code in loops or subroutines, $q(j)_{max}$ will be the maximum loop count or call count. There are also cases where the maximum rate is much less than unity, e.g. defensive code ($q(j)_{max} = 0$) and initialisation code ($q(j)_{max} = 1/T$).

So to determine the maximum scale-up factor we need to identify a worst-case execution rate profile $Q_{\max}$ that maximises S. The worst-case profile would actually be a single path (K) through the program tree such that:

$$S(Q_{\max},Q) = \sum (q(k)_{\max} L(k) / q(k)) / L$$

is maximised, i.e. where k represents all segments in path K, and $q(k)_{\max}$ is the maximum possible execution rate of the segment. This could be a hard problem to solve in general. An exact solution requires an in-depth analysis of the program structure to find all possible paths, and we should only identify paths that are *feasible*. Take, for example, the following C code fragment:

```
1.    if (val > HILIM)
2.                    {val = HILIM};
3.    if (val < LOLIM)
4.                    {val = LOLIM};
```

It is clear that val cannot satisfy both conditions (i.e. it cannot be high and low at the same time), so a worst case path K that executes segments 2 and 4 is not feasible.

Since it is difficult to determine which segments k are included in the worst-case path, we can set a conservative upper bound by summing over *all* program segments, i.e.:

$$S(Q',Q) < \sum (q(j)_{\max} L(j) / q(j)) / L \qquad (11)$$

where j is any segment in the program. This bound should apply regardless of the profile Q' used in subsequent operation.

3.5 Handling non-executed code segments

If a code segment i is never executed in T tests, the observed execution $q(i)$ rate is zero. Clearly any $q(i)$ of zero would result in an infinite scale factor using equation 8 then $q'(i)$ is non-zero. Ideally, this should never occur as all executable segments should be covered (e.g. by including extra module tests using equation 10). However, there is an alternative way of handling non-executed segments. We can split the segments into two classes indexed by c (covered segment) and u (uncovered segment). A fault in an uncovered segment could have a maximum failure probability of unity when the segment is executed. However a failure can only occur if a fault is present and the segment is executed, hence the maximum expected failure intensity after T tests is:

$$\bar{\theta}_u'(T) \le N \frac{L(u)q'(u)}{L}$$

while for covered segments, the bound on the expected failure intensity is:

$$\bar{\theta}_c'(T) \le \frac{N}{eT} \cdot \frac{L(c)q'(c)}{L \cdot q(c)}$$

It follows that for an uncovered segment, the equivalent execution rate that should be used in equation 8 to derive the scale factor is:

$$q(u) = \frac{1}{eT} \qquad (12)$$

4. EXPERIMENTAL EVALUATION OF THE SCALING THEORY

The theory presented in the previous sections suggests that programs tested with a "fair" profile should be less sensitive to profile changes than programs tested with an "unfair" profile. We have also given a method for bounding the worst-case change in the failure rate for any new profile given knowledge of the execution profile under test.

To assess whether this occurs in practice, and also to check the conservatism of the bounding formula, we derived the scale-up factors for a range of distributions and compared them with the scaled worst-case bounds predicted by equation (7) for two different programs.

4.1 Application to the PODS TRIPV program

The scale-up evaluation used the TRIPV program used in the PODS experiment [1]. This program computes a reactor power value from a number of analogue inputs and then performs a reactor trip decision calculation The original Fortran TRIPV program was converted to C to facilitate testing on a modern computer. The program was then instrumented to measure segment coverage and was tested over 10 000 program cycles with the following sets of random input data:

- uniform random (where all analogue input values have equal probability)
- normal (where the input values are taken from a normal distribution so the central values are most probable)
- inverse normal (a "bath-tub" distribution where the outer edge values are the most probable)

The segment execution rate profiles for the program under different input profiles are shown in figure 5 below. To aid comparison, the segments denote by *j* on the x axis are ordered by decreasing execution rate (for the case where the input data is uniform random data as depicted by the solid line in figure 5). Note that only the least executed segments are displayed in the figure. The highest execution rate segments tend to have very similar values (e.g. fixed loops and code on the main program path are always executed and hence the rate per program execution is constant regardless of the input profile).

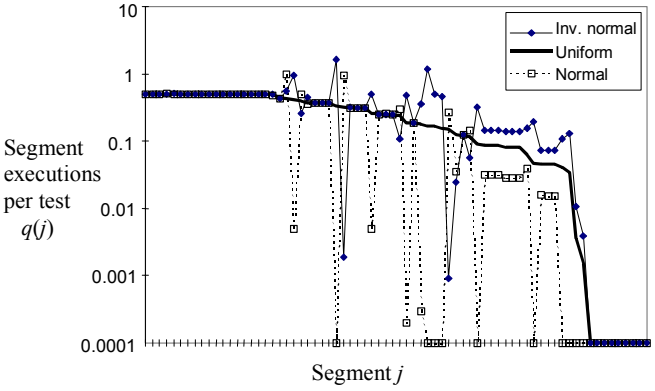

Figure 5. Segment execution rates /cycle for different input distributions (averaged over 10 000 program cycles)

As we can see from figure 5, the inverse normal input profile tends to produce the highest execution rates for most segments while the normal input distribution has a larger proportion of low (and zero) execution rates. This is not too surprising—most of the boundary conditions occur near the edge of the input range, so boundary checking code segments are activated more often with the inverse normal distribution. Conversely the normal distributions tend to select central values, so boundary values are selected less frequently.

Some segments are not executed at all. In most of these cases, segment execution is impossible (i.e. they are defensive checks against infeasible values) but in the case of the normal distribution, 6 feasible code segments are never executed (out of 140 feasible segments). For these non-executed segments we used equation (12) to derive an equivalent execution rate of 0.00037 per test (i.e. $1/eT$, where T=10 000).

4.1.1 Sensitivity to changes in input profile

Figure 5 suggests that the inverse normal input distribution is the "fairest" as it tends to have a higher execution rate for most segments. However there are exceptions where the rates can be dramatically lower than the rate under a random distribution, and as we have noted earlier, it is the least executed segments that have the dominant effect.

To assess which distribution is the fairest (i.e. least sensitive to changes in profile), we computed the scale-up factors for pairs of input distributions used in the TRIPV coverage experiment (one as the initial test profile, the other as the operational profile). Equation (8) was used to compute the scale-up factors, and the results are shown in the table below.

Table 7. Scale factors for different combinations of test and operational profiles (PODS)

Test profile	Operational profile		
	uniform	inverse normal	normal
uniform	1.00	1.21	0.90
inverse normal	3.20	1.00	6.17
normal	114.86	345.97	1.00

It can be seen that tests with uniform random data are the least sensitive to change. Perhaps surprisingly there is a scale-up factor of *less* than unity when normal data is used as the operational profile. This "scale-down" effect can occur with an "unfair" profile used in operation, because the average program path can be shorter than the average under the test profile (and hence, on average, encounters fewer faults).

It can be seen that the normal input distribution is the most sensitive to change. This is to be expected as exception conditions at extreme input values are relatively poorly tested (and in some cases not at all). We can also see that the greatest scale-up occurs when we test with normal data but have an operational profile that is inverse normal. In this case, the scale-up factor can be between one and two orders of magnitude.

4.1.2 Maximum scale-up prediction

Equation (11) gives the maximum scale-up relative to a specific test profile. The program was analysed to determine the maximum possible execution rate for each segment, $q(\mathrm{j})_{\max}$, to account for the execution of loops and multiple calls to subroutines. This was comparatively easy to derive as the program has a simple structure with fixed loops and a fixed number of calls to subroutines. We also had to set $q(\mathrm{i})_{max}$ to zero for non-executable defensive code, and also set $q(\mathrm{j})_{max} = 0.0001$ for initialisation code (since it is executed once in 10 000 tests). In the following table, the worst case scale factor predictions for different test profiles are compared with the observed scale-up factors achieved with different operational profiles.

Table 8. Actual scale factors vs. predicted maxima

Test profile	Scale-up factor	
	Max actual scale-factor	Max predicted scale factor
uniform	1.21	6.6
inverse normal	6.17	10.0
normal	345.97	1059.3

We do not know if there are other operational profiles that result in greater scale factors and there is some conservatism in the maximum scale factor prediction. Nevertheless, the actual scale factors are less than the predicted maxima and the general shape of the prediction mirrors the empirically observed scale-up.

While the uniform random input distribution has the smallest scale-up bound, this distribution is not necessarily the optimum test pattern for a program. Achieving a fair distribution over the code segments requires knowledge of the decision points within each input range and it is likely that each input variable will need a different distribution to maximise fairness. To assess whether the uniform random profile was the "fairest", we performed further segment coverage tests with intermediate distributions, namely "shallow" versions of the inverse normal and normal distributions. The table below shows the maximum scale factor prediction for all distributions.

Table 9. Predicted maximum scale-up: different test profiles

Test profile	Maximum scale-up
Inverse normal	10.0
Inverse (shallow)	5.0
Uniform	6.6
Normal (shallow)	745.2
Normal	1059.3

This suggests that (for this program at least) a slight bias towards the extremes of the input range can improve the overall fairness of the execution rates over the code segments. This seems intuitively reasonable as quite a lot of the program logic is associated with range checks that occur at the extremes of the input range. It is likely that further tuning of the distributions on individual inputs could improve fairness even further.

We can compare these scale-up bound predictions against the mean reliability growth for the 12 known faults in PODS. The mean failure probability of a program after T tests is modelled by:

$$\overline{\theta}(T) = \sum \lambda_i \exp(-\lambda_i T) \qquad (13)$$

for all faults i in the program, where λ_i is the probability of failure per test of fault i. This assumes that the failures of all 12 are disjoint (the worst case).

The values of λ_i for all 12 faults in the PODS TRIPV program were established by performing long term failure rate measurements under uniform, normal and inverse normal ("bathtub") input profiles with a single fault activated for each measurement. We then computed the mean failure probability of the set of faults after different numbers of tests T for each profile using equation (13). This is effectively a virtual reliability growth experiment that makes use of the known failure rates of the faults. The growth curves are shown in figure 6 together with the worse case bound prediction, N/eT, for the failure probability from equation (4). This bound is independent of the test profile provided that the profile is not changed.

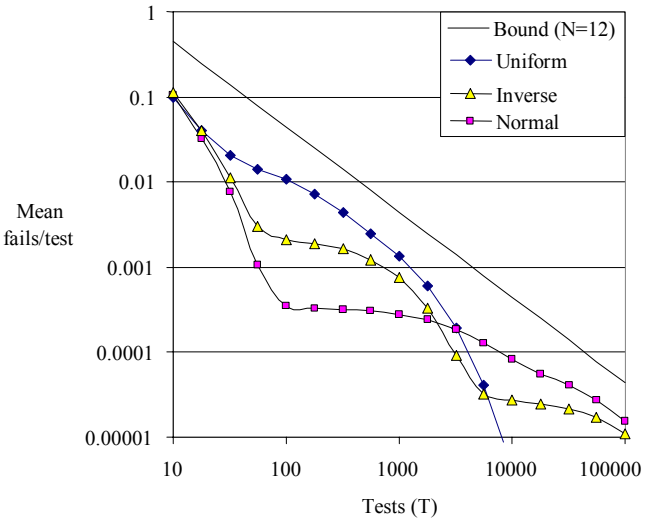

Figure 6. Reliability growth (different profiles) vs worst case bound

It can be seen that, regardless of the test profile chosen, the failure probability is always below the predicted worst case bound.

We can also compute the effect of switching to a different profile after T tests. If at time T, a new profile is used, the mean failure probability will be:

$$\overline{\theta'}(T) = \sum \lambda_i' \exp(-\lambda_i T) \qquad (14)$$

where λ'_i is the failure rate (per program test) of the fault under the new profile.

In figure 7 below we show the failure rate under the "normal" test profile and the effect of switching to the "uniform" profile after T tests for subsequent operation.

The "Test" plot shows the mean reliability growth if the "normal" input profile is used for testing (equation 13). The "Operation" plot shows the mean reliability if the profile is switched to the "uniform random" profile (equation 14) after T tests. Figure 7 also shows:

- the standard bound (equation 4) under the test profile
- the scaled bound for a switch to the uniform random profile (from Table 7 and equation 14)
- the maximum scaled bound (Table 8 and equation 11) that applies to a switch to any other profile.

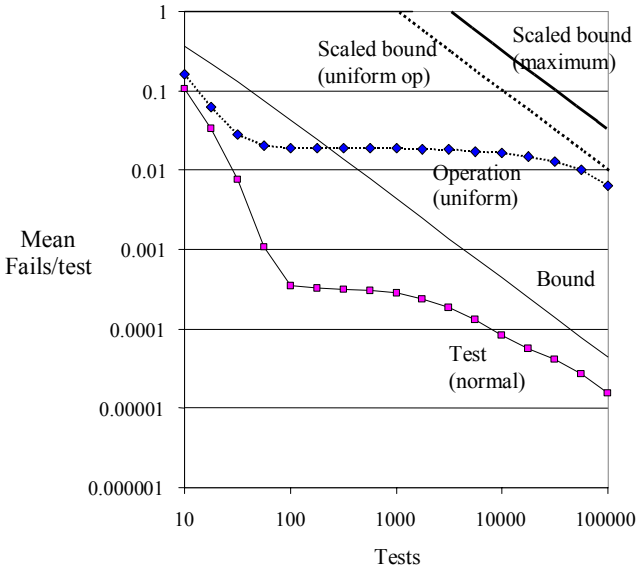

Figure 7. Change in reliability: "normal" test profile followed by "uniform" operational profile

It can be seen that there is a very large increase in failure probability when the input profile is changed to uniform random, but the failure rates are within the predicted bounds.

We repeated this virtual reliability growth experiment using "uniform random" as the test profile, and switching to the "normal" profile for subsequent operation (see figure 8).

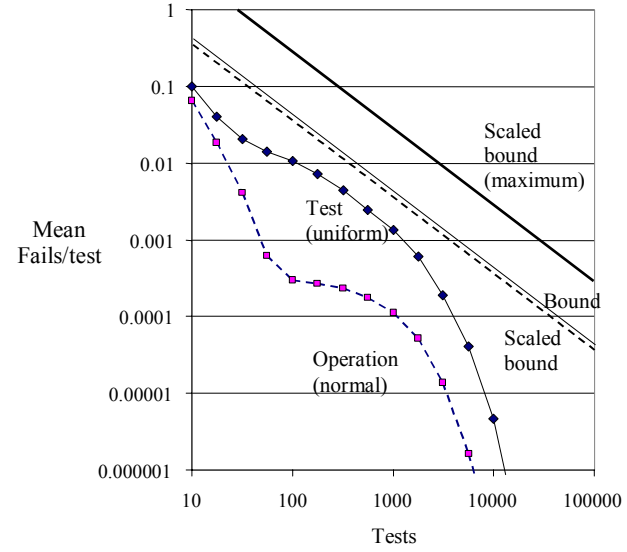

Figure 8. Change in reliability: "uniform" test profile followed by "normal" operational profile

As before, the original, scaled and maximum bounds are plotted on the same graph. It can be seen that when the profile is switched to the "normal" input profile the failure probability actually *decreases*. This is reflected in the re-scaled bound for a "normal" profile which also decreases slightly. As discussed earlier a reduced scale factor is possible when the well-tested segments are heavily used in operation. So, as predicted by the theory, the "fairer" test profile produces a program that is less sensitive to changes in the operational profile.

In both virtual experiments, the reliability bound equation (4), is successful in bounding the reliability growth under a fixed profile, and the scale-up bound equations (8 and 11) successfully bound the reliability changes when a new operational profile is used.

4.2 Application to PREPRO

The same type of experimental evaluation was applied to the PREPRO example. This was an offline program from the European Space Agency that had been used in an earlier reliability experiment [8]. This program computes the performance of an antenna array. This antenna array is specified in a special language which is parsed by the PREPRO program. A test generator was available that could generate random antenna descriptions of varying complexity. We instrumented this program and measured the test coverage under two different profiles (P1 and P2) produced by a modified random test case generator that could vary the occurrence rates of different antenna features.

Using equation (8), the predicted scale-ups for PREPRO were:

Table 9. Scale factors for the PREPRO example

Test profile	Operational profile	Predicted scale factor
P1	P2	1.1
P2	P1	3.8

We could not compute a maximum scale factor as the program was recursive, so the segment execution rates were unbounded.

We measured the failure rate λ_i for 28 known faults in PREPRO under profiles P1 and P2. As in the PODS example, we then computed the mean failure probability expected when tested under one profile, and the expected failure probability if there is a switch to a new profile after time *T*.

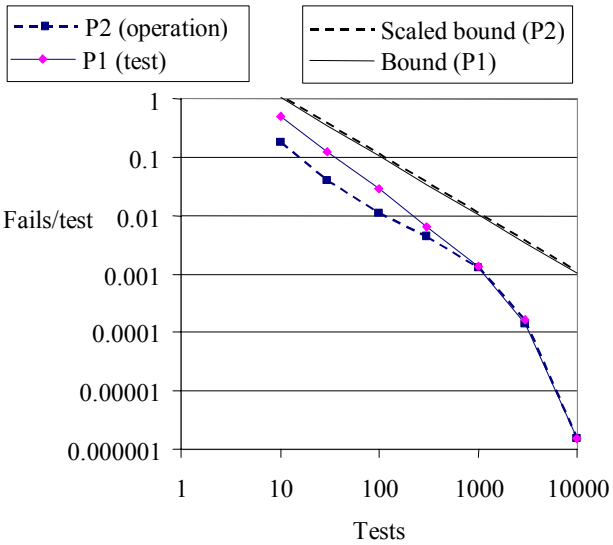

Figure 9. PREPRO evaluation: (P1 test, P2 operation)

In figure 9, we show the reliability growth when the program is tested under the "fairer" profile P1 and the impact of changing to profile P2. We also show the predicted reliability bounds under

the test profile P1 and the scaled bound under the operational profile P2.

It can be seen that the failure rate under P2 is *less* than under the original profile (an effect also observed in PODS). This is consistent with the relatively small scale-up factor of 1.1 predicted by the scale-up theory.

For the converse case where the "unfair" profile P2 is used for testing and P1 is used in operation the predicted scale factor *S*(P1,P2) is 3.8 so we would expect a significant increase in observed failure probability when profile P2 is used in place of P1. Figure 10 shows the results of using P2 for testing then switching to profile P2.

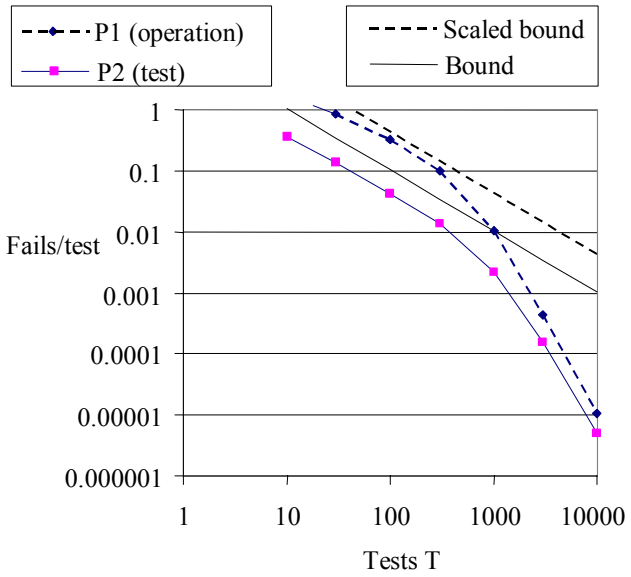

Figure 10. PREPRO evaluation: (P2 test, P1 operation)

So both the PODS the PREPRO results are consistent with the reliability bound scaling theory.

5. APPLICABILITY OF RESULTS

This paper has extended the reliability bound theory developed in [2]. In the extended theory:

- The reliability bound prediction in [2] can be re-scaled for a different operational profile provided the program segment execution rates can be measured under both profiles.
- The scaling theory can include module test information as well as tests applied to the entire program.
- If it is possible to set an upper bound on the execution rates of all program segments, it is possible to compute a scaled-up reliability bound that would apply to *whatever operational profile is chosen*.
- For some program structures it is possible to identify a totally "fair" test profile where there is *no change* in the reliability bound *whatever operational profile is used*. But in practice it is unlikely that perfectly fair test profiles can be achieved for realistic programs.

The experimental evaluation appears to be consistent with the predictions of the theory, but the theory needs to be tested on a range on different software examples. In addition the assumptions necessary to develop the re-scaling theory could be viewed as extreme and unlikely to be applicable in practice. It will be necessary to check the impact of violations of the underlying assumptions. Some of the effects of assumption violations are discussed below:

- Assumption (1) states that faults are localised to basic block. This assumption makes it easy to re-scale the failure intensity of a faulty block for a new profile. In practice, failures due to a single fault can be spread over many blocks, e.g. due to incorrectly initialised data. This may not have a major effect on the scaling prediction as the scale factor is actually a weighted average *over all blocks*. A hypothetical fault that causes failures in all blocks should have a scale factor equal to the mean scale factor. By comparison, scaling for localised faults can depart from the average (although the scaling factors averaged over all faults should be close to the predicted mean value). On this basis, we would expect *less* variation in scale factors for non-localised faults. So violations of assumption (1) do not necessarily invalidate the results.
- Assumption (2) that the probability of failure per test is proportional to segment execution rate is questionable. If the segment is executed several times per test, the scaling could result in a failure probability greater than unity. However in these cases, the predicted scale-up is greater than the actual scale-up so the scale bound is conservative.
- Assumption (3) that faults are equally likely in all lines of code is also debatable. It is known that fault density can increase in complex code modules. However this may not matter; the scale factor is an average over the entire program flow graph. Uneven distribution of the faults will only have an effect if the fault density is correlated with the scale-up factors for individual segments.

We should also note that the computed scale factor is a *mean* not a maximum. There is a finite probability that all *N* faults are located in the segment with the worst scale-up factor. However, the probability that all faults are located in the highest scale-up segment decreases rapidly as *N* increases. This effect would also be less severe if the faults were non-localised.

The limited analysis above suggests that assumption violations may not have a major effect (or are conservative), but further analysis is needed.

Current software engineering practice favours the use of a realistic operational profile [6] for testing the integrated system. If this profile is correctly defined then the reliability achieved in the field should match that observed in development. However our theory indicates that if the expected operational profile is "unfair" then the field reliability could be extremely sensitive to changes in the operational profile (in the PODS TRIPV evaluation, changes of several orders of magnitude were predicted and observed). A radical alternative to realistic testing would be the use of "fair testing" as this would result in profile-independent reliability bounds. However there are considerable technical barriers to the identification, feasibility and implementation of perfectly fair test profiles and the theory is based on assumptions that have not been evaluated on realistic programs. So it would be undesirable to abandon realistic testing in favour of "nearly fair" random testing.

However the theory could help to reduce sensitivity to changes in the profile during operation. Possible strategies are listed below.

- Design tests that execute the *least used* program segments (to fill "holes" in the segment execution profile). The theory indicates that it is the least used paths that pose the greatest risk when the profile changes. Additional testing of the least used and unused code segments should reduce sensitivity to changes in operational profile.
- Include module test results using equation (10) when estimating the reliability bound. Module tests can be used to achieve a fairer execution coverage.
- Estimate the sensitivity of the test profile by using maximum scale-up formula (equation 11). The sensitivity of the bound could then be assessed as part of the software acceptance process.
- Monitor the execution profile during operation. If the operational and test execution profiles are known, the worst case reliability bound can be adjusted to accommodate the difference using equation (7).

The theory could also be relevant to module testing. To limit sensitivity to changes in reliability in operation, module tests should be as fair as possible and, from equation (10), the *number* of segment executions should be increased beyond that needed to achieve full coverage.

6. SUMMARY AND CONCLUSIONS

The reliability bound theory developed in [2] has been extended to include changes in operational profile. The theory can predict the change in the reliability bound for a specific operational profile and for a worst case profile. Perhaps the most surprising aspect of the theory is that, in principle, there can be "fair" test profiles that produce programs that are insensitive to changes in the operational profile. An experiment assessment of achieved reliability using different test and operational profiles appears to be consistent with this theory.

Given the novelty of the theory, the assumptions and practical testing constraints, we do not think fair testing can replace tests using a realistic profile. However, the results of the theory can be used to assess and potentially reduce the sensitivity of the delivered program to unexpected changes in the operational profile.

To develop the theory it was necessary to make a number of additional assumptions about the behaviour of faults that are debatable. However a limited analysis of the assumptions suggest that the results of the theory seem to be relatively robust to assumptions violations.

Further work is needed to evaluate the theory on different software examples and to assess the impact of departures from the theory assumptions. It would also be useful to explore the feasibility of deriving and implementing "fair" test profiles for realistic programs.

7. ACKNOWLEDGMENTS

This research was undertaken for the UK Nuclear Industry C&I Research Programme under Scottish Nuclear (now British Energy Generation UK) contract PP/114163/HN. The paper reporting the research was produced under the EPSRC research interdisciplinary programme on dependability (DIRC). I also wish to thank Prof. Bev Littlewood, Prof. Lorenzo Strigini and Dr. Mourad Oussalah at the Centre for Software Reliability for their constructive comments.

8. REFERENCES

[1] P.G. Bishop et al, "PODS a Project on Diverse Software", IEEE Trans. Software Engineering, Vol. SE-12, No. 9, 929-940, pp. 929–940, 1986

[2] P.G. Bishop, R.E. Bloomfield, "A Conservative Theory for Long-Term Reliability Growth Prediction", IEEE Trans. Reliability, vol. 45, no. 4, pp 550-560, Dec. 1996

[3] R.E. Bloomfield, A.S.L. Guerra, "Process Modelling to Support Dependability Arguments", DSN 2002 Washington, DC, 23-26 June, 2002

[4] J.R. Gaffney, "Estimating the Number of Faults in Code", IEEE Trans. Software Engineering, vol. SE-10, no. 4, 1984

[5] M. Kaaniche, K. Kanoun, M. Cukier and M. Bastos Martini, "Software Reliability Analysis of Three Successive Generation of a Switching System", LAAS Report no 94.030, 1994.

[6] J.D. Musa, "Operational profile in software reliability engineering", IEEE Software 10(2) pp.:14-42, 1993

[7] Y.K. Malaiya and J. Denton, "Estimating the Number of Residual Defects", HASE'98, 3rd IEEE Int'l High-Assurance Systems Engineering Symposium, Maryland, USA, November 13-14, 1998

[8] A Pasquini, A N Crespo and P Matrella, "Sensitivity of reliability growth models to operational profile errors", IEEE Trans. Reliability, vol. 45, no. 4, pp 531–540, Dec. 1996

[9] K. Yasuda, "Software Quality Assurance Activities in Japan", *Japanese Perspectives in Software Engineering,* 187-205, Addison-Wesley, 1989