Department of Computer Science

On Input Profile Selection For Software Testing

Naixin Li and Yashwant K. Malaiya

Technical Report CS-94-109

March 15, 1994

Colorado State University

On Input Profile Selection For Software Testing *

Naixin Li Yashwant K. Malaiya

Computer Science Department
Colorado State University
Fort Collins, CO 80523
(303) 491-7031
malaiya@cs.colostate.edu

Abstract

This paper analyzes the effect of input profile selection on software testing using the concept of fault detectability profile. It shows that optimality of the input profile during testing depends on factors such as the planned testing effort and the error detectability profile. To achieve ultra-reliable software, selecting test input uniformly among different input domains is preferred. On the other hand, if testing effort is limited due to cost or schedule constraints, one should test only the highly used input domains. Use of operational profile is also needed for accurate determination of operational reliability.

1 Introduction

Significant effort is now being devoted to develop techniques to deliver reliable software. Methods proposed include well-controlled software development practice such as the cleanroom approach[16, 24], formal verification and testing. Cleanroom approach significantly reduces the number of faults introduced during the early phases of software life cycle, but it cannot totally avoid the problem of software faults and failures. Formal verification has been used for small programs but, in its current stage, cannot be applied to practical software which can be very large. In foreseeable future, achievement of reliable software will heavily rely on software testing.

During testing a program is executed with some inputs to see if the software operates as it is specified. It is impossible to exhaustively test a program due to the sheer size of

*This work was partly supported by BMDO and is monitored by ONR

software input space. Thus some approach must be used to select a small subset of the input space with the hope that the inputs from this subset are representatives for the whole input space and will be able to detect most, if not all of the software faults.

Several different approaches for software testing are used. For functional testing, input space is partitioned into domains based on the functions supported by the software. Every input from a domain is considered to be equivalent to every other input from the same domain as far as the software fault detection is concerned. Structural testing is based on the control flow of software code. One cannot have confidence in a piece of code unless it has been tested out. One should test all possible and reachable elements of a software if cost and time constraints allow. Many criteria have been proposed for structural testing including statement coverage, branch coverage, and data-flow based coverages.

Both functional and structural testing have their limitations. Neither of them assures that every possible fault will be found. Some coverage criteria can be too costly to be practical.

There is another category of testing termed random testing. In this approach, test input is selected randomly from the input space. Testing continues until either the deadline for release is met or it is estimated that the objective failure rate is reached. The advantage of random testing is the ease of selecting an input. However some kind of test oracle may be needed to efficiently verify that an output is valid.

The major purpose of testing is to increase the reliability of a software. During testing, if a fault is found, it will be fixed and hence the reliability is improved. Even if no faults are found and fixed for a period, our confidence about the software reliability is increased. The reliability growth exhibited during software testing has much to do with the selected test input. What really matters to the user, and also to the testing personnel, is the software's operational reliability, which depends on the software's quality as well as its operational usage. Since it is too difficult, if not impossible, to detect and fix all the faults in a software, it would be ideal if one can detect and fix faults that are more likely to result in failures during operational use. This gives rise to the idea of operational profile-based testing [18, 19] which involves partitioning input space into domains and selecting inputs from each domain based on its frequency during operational use. Musa has given detailed steps for the construction of operational profile and the associated test input selection [19]. Cobb and Mills [5] mentioned that operational profile based (usage) testing is 20 times more effective than coverage testing. We examine this aspect of testing in detail here.

Another purpose of software testing is to assess the software quality. The software failure data collected during software testing are used to drive software reliability growth models so that an estimation about the software's reliability can be made. The accuracy

of such estimation requires that the software should be exercised during testing phase in a similar way, or following the same input distribution, as the software in operational usage. Indeed, this is an assumption generally made for software reliability models [9]. If the input selection during testing phase is different in distribution from that in operation, some adjustment should be made to account for the differences. Musa et al [17] introduce a concept termed *test compression factor* for this purpose. In contrast with real operational use, input states for software during testing phase are generally not repeated or repeated with much lower frequency. Thus, actual test inputs are more effective in revealing errors than random sampling according to operational usage patterns. An simple example was given in [17] to illustrate the concept of test compression factor, which is quoted below.

> Assume that a program has only two input states, A and B. Input state A occurs 90 percent of the time; B, 10 percent. All runs take 1 CPU hr. In operation, on the average, it will require 10 CPU hr to cover the input space, with A occurring nine times and B, one. In test, the coverage can be accomplished in 2 CPU hr. The testing compression factor would be 5 in this case.

Based on some assumptions, Musa et al[17] computed that the test compression factor varies from 8 to 20 for softwares with the number of input states ranging from 10^3 to 10^9. Musa et al [17] also noted that equivalence partition testing can increase the test compression factor. A similar concept termed accelerating factor was used by Drake and Wolting [7]. Using repair data, they computed the value of acceleration factors for two terminal firmware systems to be 47805 and 45532. Observations [6, 8, 28] of significant correlation between structural coverage and error removal growth and work by Malaiya et al [13] also support that real testing can be more effective than random sampling over operational usage distribution. Data gathered by Hecht and Crane [10] indicate that code segments for rare conditions, like exception handling, have a much higher failure rate than normal code. Since such code segments are not readily exercised during software testing, relatively more errors (corresponding to higher error rates) are left undetected in such segments. When these segments happens to be executed in real operation, they are much more likely to result in a failure. This would suggest that substantial number of test cases should be directed towards rare conditions, which generally cannot be satisfied by operational usages testing.

We thus have two conflicting considerations. On one side, test input selection reflecting operational usages tends to capture errors that are more likely to result in a failure during operation; on the other side, it is believed that test input profile with more coverage (of code, path, rare conditions, etc.) should be more effective in error removal. Taking both of these aspects into consideration, what is the best overall test input selection scheme for enhancing the reliability of a software ? How can the knowledge of operational profile be

best used in software testing ? This paper tries to answer these questions.

2 Optimum Test Input Distribution

2.1 Input space with two domains

Let us start with a simple case which is analyzed and interpreted relatively easily. Assume we have a software whose operational profile is described by input space partition S1, S2, $|S1| \gg 1$, $|S2| \gg 1$, with op_1 (percent of) usage from S1 and op_2 (percent of) usage from S2. Obviously $op_1 + op_2 = 1$. There are exactly 2 faults in the software. Fault 1 can be detected only by inputs from S1, with detectability [15] of $d_1 = 1 - p_1$ in S1. Fault 2 can be detected only by inputs from S2, with detectability of $d_2 = 1 - p_2$ in S2. (If two faults are equally testable by S1 and S2, then the effect of testing on reliability growth is independent of the distribution of test input selection.) We also assume that all failures will be observed and debugging is perfect, that is, no new faults are introduced while a fault is fixed. Since both S1 and S2 are large enough, we will consider input selection from either of them as sampling with replacement, which will facilitate the calculation.

$$
\begin{aligned}
P_{s1} &= \text{Prob}\{\text{an input from S1 is processed properly after } n1 \text{ test runs from S1}\} \\
&= \text{Prob}\{\text{Fault 1 will not be encountered} \mid \text{it was not found in } n1 \text{ tests}\} \\
&\quad \times \text{Prob}\{\text{it was not found in } n1 \text{ tests}\} \\
&\quad + \text{Prob}\{\text{it will not be encountered} \mid \text{it was found in } n1 \text{ tests}\} \\
&\quad \times \text{Prob}\{\text{it was found in } n1 \text{ tests}\} \\
&= p \times p_1^{n1} + 1 \times (1 - p_1^{n1}) \\
&= 1 - p_1^{n1} + p_1^{n1+1}
\end{aligned}
$$

Similarly,

$$
\begin{aligned}
P_{s2} &= \text{Prob}\{\text{an input from S2 is processed properly after } n2 \text{ test runs from S2}\} \\
&= 1 - p_2^{n2} + p_2^{n2+1}
\end{aligned}
$$

Let $n1 + n2 = n$ be the total number of test runs. $n1 = k \times n$, $n2 = (1 - k) \times n$; where $0 \leq k \leq 1$ is the proportion of test inputs that are chosen from S1. Then the overall probability of a correct execution is given by,

$$P_{sys} = P_{s1} \times op_1 + P_{s2} \times op_2 = op_1(1 - p_1^{kn} + p_1^{kn+1}) + op_2(1 - p_2^{(1-k)n} + p_2^{(1-k)n+1}) \qquad (1)$$

Differentiating this with respect to k on both sides,

$$\frac{dP_{sys}}{dk} = op_1[-(n \ln(p_1))p_1^{kn} + (n \ln(p_1))p_1^{kn+1}] + op_2[(n \ln(p_2))p_2^{(1-k)n} - (n \ln(p_2))p_2^{(1-k)n+1}]$$

To obtain the optimal value of k, we equal the above to 0 and solve to get,

$$k_{opt} = \frac{1}{n} \times \frac{\ln(\frac{op_2 \ln(p_2)}{op_1 \ln(p_1)} \times \frac{p_2^n(1-p_2)}{1-p_1})}{\ln(p_1 p2)} \tag{2}$$

This gives the optimum proportion of test input which should be selected from S1 provided that we know the values of all the parameters p_1, p_2, op_1, op_2, and n. So the optimum test input distribution is not the same as the operational usage (in this case, $k \neq op_1$). Instead, it is a function of the operational profile as well as the individual fault detectabilities $(1-p_1)$ and $(1-p_2)$, and the planned amount of test effort in terms of the number of test inputs n.

It should be noticed that in Equation 2, the terms $op1$, $op2$, $p1$ and $p2$ occur within logarithmic functions. Thus k_{opt} is not as sensitive with respect to them as for n.

To explore the variation of k_{opt}, let us assume that $p_1 = p_2 = p$, i.e., the two faults have equal detectabilities, then the above equation can be reduced to:

$$k_{opt} = \frac{1}{2} + \frac{\ln(\frac{op_2}{op_1})}{2n \ln(p)} \tag{3}$$

Notice that the second term is negative when $op2 > op1$. From this equation, we can make the following observations:

- When $op_1 = op_2$, $k = 0.5$. This says that if inputs from two domains are used with equal chance during operation, they should be tested with equal chance. When $op_1 < op_2$, $k < 0.5$. That is, if the domain S1 is used less frequently than the domain S2, S1 should also be tested less compared to S2. Similar is true for the case $op_1 > op_2$. This is consistent with the suggested operational profile based testing, although the exact distribution for test input selection differs.

- For fixed input sample size n, smaller detectability $(1-p)$ implies k_{opt} closer to 0.5 i.e. more even distribution. So test input selection should also be based on the initial overall fault detectability or software reliability. Figure 1 plots the variation of k_{opt} with p, where $op1 = 20\%$, $op2 = 80\%$, curve A corresponds to 100 test inputs, curve B to 1000 and curve C to 10000.

- For fixed fault detectability $(1-p)$, larger value of n suggests more even distribution since k_{opt} is closer to 0.5 as shown in Figure 2. This tells us that the optimal distribution of input selection depends on how much testing is going to be spent. To test most effectively all the time, the test input distribution should vary as testing proceeds.

  For small n, and small value of $(1-p)$, the value of k_{opt} obtained from the above equation can be negative, which suggests that no test inputs should be chosen from S1 if the amount of testing is very limited.

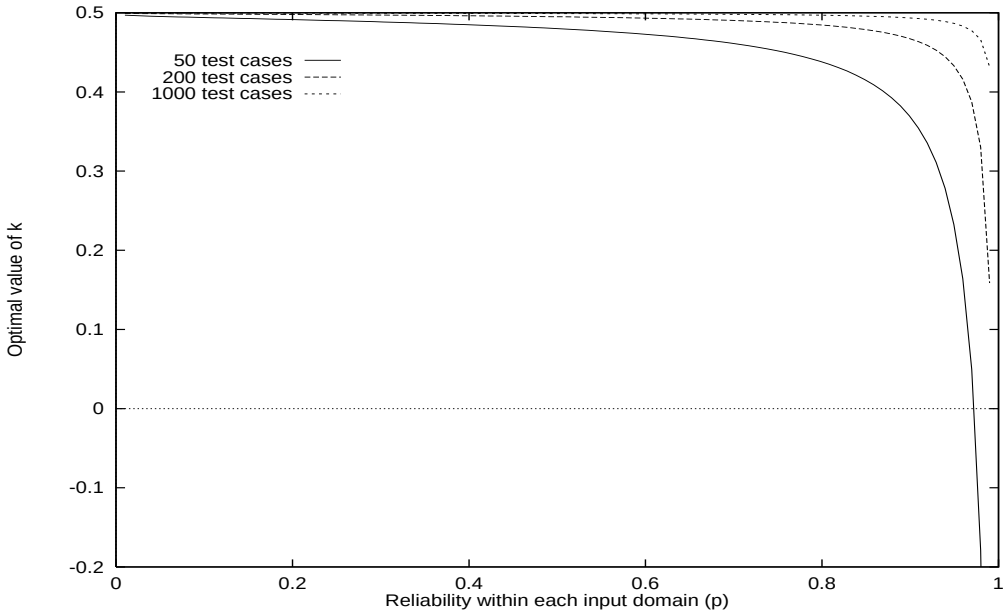

Figure 1: Variation of k with p (op1=20%, op2=80%)

As n approaches infinity, k_{opt} approaches 0.5. Which means that to achieve ultra-high reliability through extensive testing, we should select inputs with equal frequency from each domain. This may correspond to weighted random testing, because S1 and S2 may not have the same size.

2.2 Input space with multiple domains

In practice, there are several domains not just two. Typically the number of domains obtained during the construction of operational profile can be hundreds or even thousands for very large projects [18, 19]. For such cases, we can still get an analytical optimal distribution for test input selection. Lets assume that a program's input space consists of m domains with one fault associated with each domain with the same detectability $(1-p)$. In this case, the system operational reliability after n tests is described by :

$P_{sys} = 1 - (1-p)\sum_{i=1}^{m} op_i \times p^{k_i n}$

which is constrained by $\sum_{i=1}^{m} k_i = 1$.

Solving this, we obtain the optimal test input distribution given by the following values of k^i_{opt}:

$$k^i_{opt} = \frac{1}{m} + \frac{\ln(\frac{\prod_{j \neq i} op_j}{op_i^{m-1}})}{mn\ln(p)} \tag{4}$$

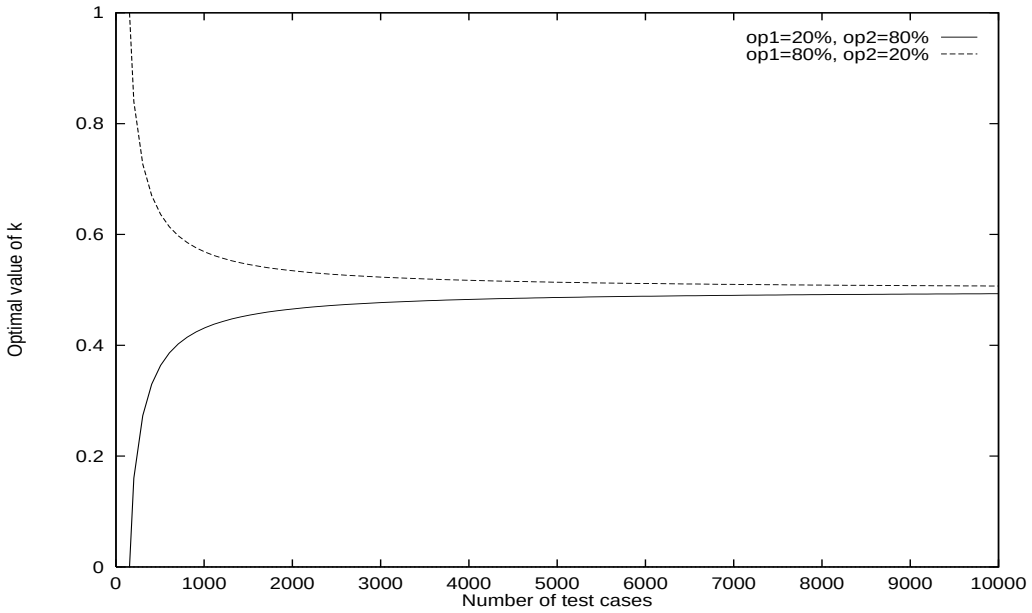

Figure 2: Variation of k_{opt} with n (p=0.99)

This has the same format as the earlier solution for the case of two partitions. The observations and conclusions in the previous section are thus still applicable.

3 Reliability Growth With Different Test Input Distributions

In this section, we will examine how reliability growth is affected by different test input distribution. These examples are given below to illustrate different reliability growth trend for different detectability profiles with different test input distributions.

Example 1. Figure 3 plots the reliability growth for a software consisting of two domains with one fault associated with each domain. Where X-axis is the number of test cases applied, Y-axis is the relative value of MTTF as given by the mean number of test cases to a failure, which is

$$MTTF = \frac{1}{1 - P_{sys}} \tag{5}$$

where P_{sys} can be computed using Equation 1. Curve for $k = 0.1$ describes the reliability growth using operational profile based testing, curve for $k = 0.01$ corresponds to testing using more biased input distribution, and the curve for $k = 0.5$ to uses even distribution between two input domains. For this example, we assume $op1 = 0.1$, $op2 = 0.9$, $1 - p = 0.01$.

From the plots, we can see that initially when the number of test input is small, more

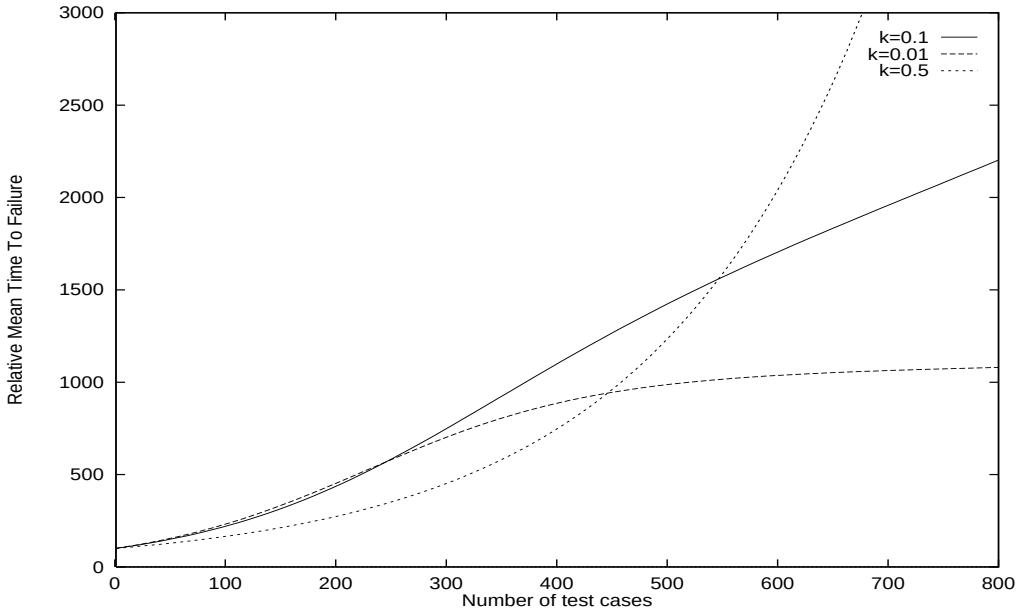

Figure 3: Variation of relative MTTF with n (one fault for each domain)

biased test input distribution gives MTTF. As more test inputs are exercised, the reliability growth curve favors the even distribution.

Example 2. Figure 4 plots the reliability growth for a system consisting of two domains with three faults associated with each domain. The X-axis and Y-axis are defined the same as above. The detectabilities for the three faults within each domain are $(1-p1) = 0.01$, $(1-p2) = 0.05$, $(1-p3) = 0.1$ respectively. The operational profile is described by $op1 = 0.01$, $op2 = 0.99$. The curve for $k = 0.01$ shows the result of testing with operational usage. The curve for $k = 0.001$ is more biased. While the Curve for $k = 0.1$ is less biased than operational usage. Again the curve for $k = 0.1$ uses an even distribution.

The plot shows that when the number of test is less than 450, usage-based testing is slightly better than more uniform testing. After this, uniform testing will be remarkably superior to usage-based testing.

Example 3. Figure 5 plots the reliability growth for a system consisting of four domains with one fault associated with each domain. Again the X-axis and Y-axis are defined the same as above. The values of the parameters used in this plot are: $op1 = 0.01$, $op2 = 0.1$, $op3 = 0.3$, $op4 = 0.59$, $(1-p) = 0.02$. The dashed curve in the plot corresponding to uniform testing. The solid curve reflects usage-based testing.

When the number of test input is less than 630, usage-based testing is superior to uniform testing. However, as more testing is involved, uniform testing becomes much more better than usage testing.

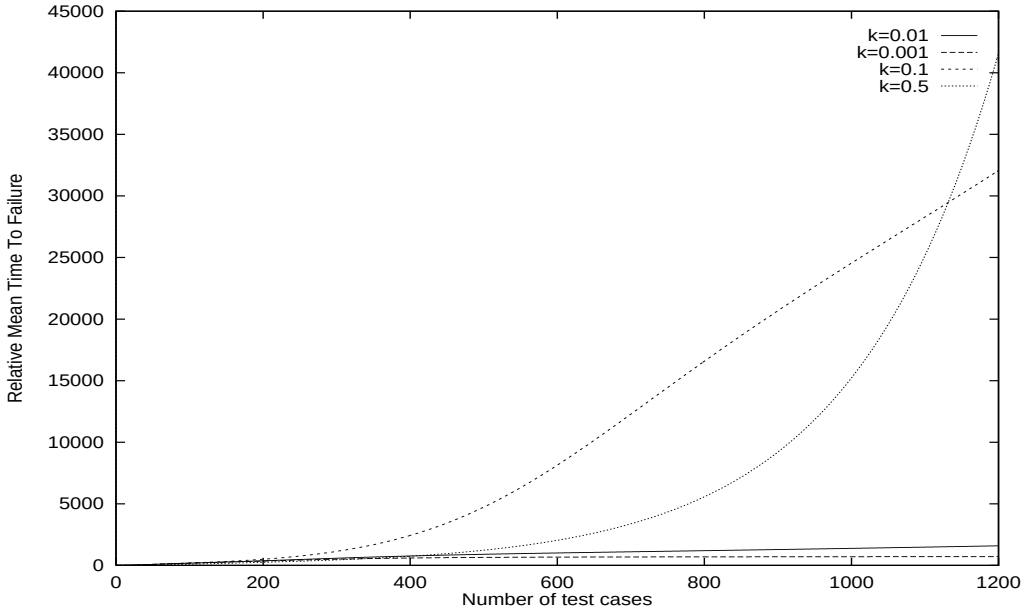

Figure 4: Variation of relative MTTF with n (3 faults in each domain)

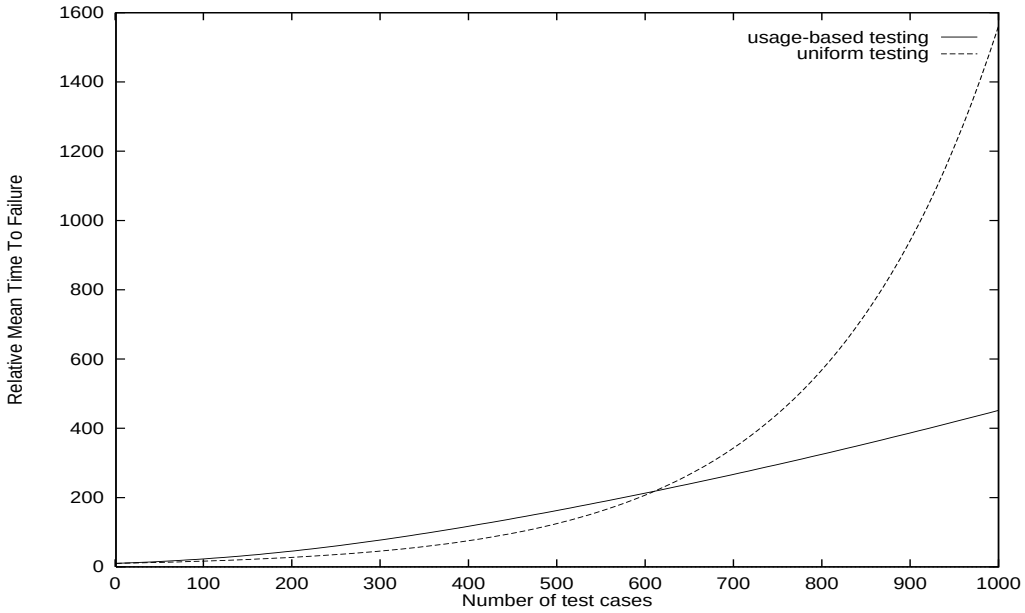

Figure 5: Variation of relative MTTF with n (p=0.98)

Although the number of domains, the number of faults associated with each domain, and the parameters vary, the general trend shown in the above three examples is the same. That is, testing should be more biased towards the frequently used domains if only a small number of test inputs is allowed. However, as more test inputs are executed, test inputs should be selected more uniformly among different domains.

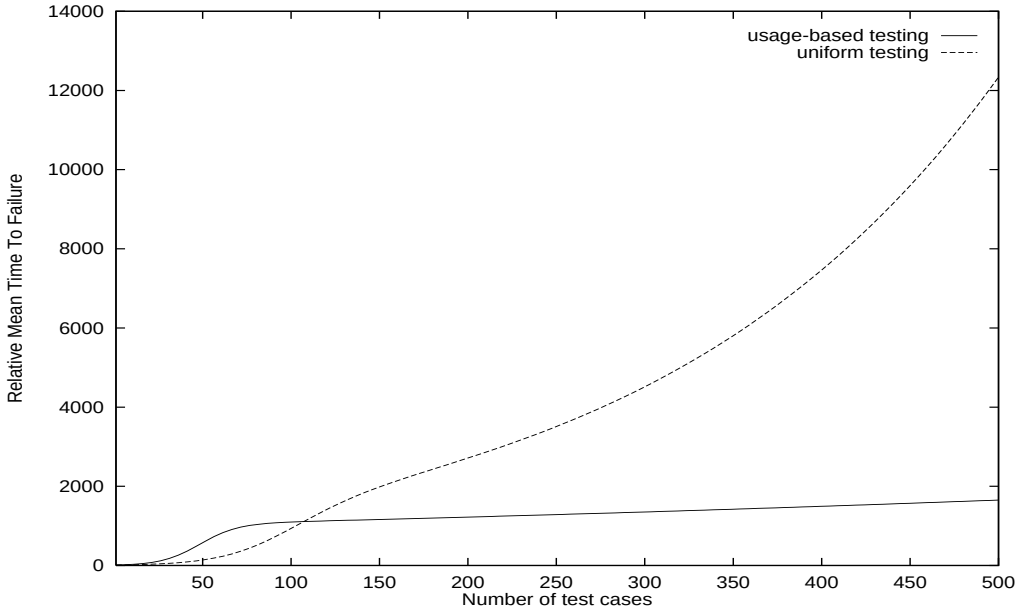

Figure 6: Variation of relative MTTF with n (op1=90%, op2=10%, p1=0.9, p2=0.99)

Example 4. Figure 6 plots the reliability growth for a system with 2 domains. There is one fault associated with each domain. The parameters are assumed as follows: $op1 = 90\%$, $op2 = 10\%$, $p1 = 0.9$, $p2 = 0.99$. One should notice here the detectabilities of faults are different and the detectability values are set in favor of usage based testing. However, even for this case, uniform testing gives better MTTF once the number of tests exceeds about 110.

4 Usage Testing vs. Coverage Testing

Adams' study of some real software [1] shows that the operational failure rates for different projects follow a similar distribution with the number of faults having a certain failure rate being inversely proportional to the failure rate. Figure 7 plots the relative detectability profiles from two projects and the average detectability profile for 9 other projects.

Cobb and Mills [5] used Adams' data in their computation and came to the conclusion that usage testing testing is about 20 times more effective than (statement) coverage testing.

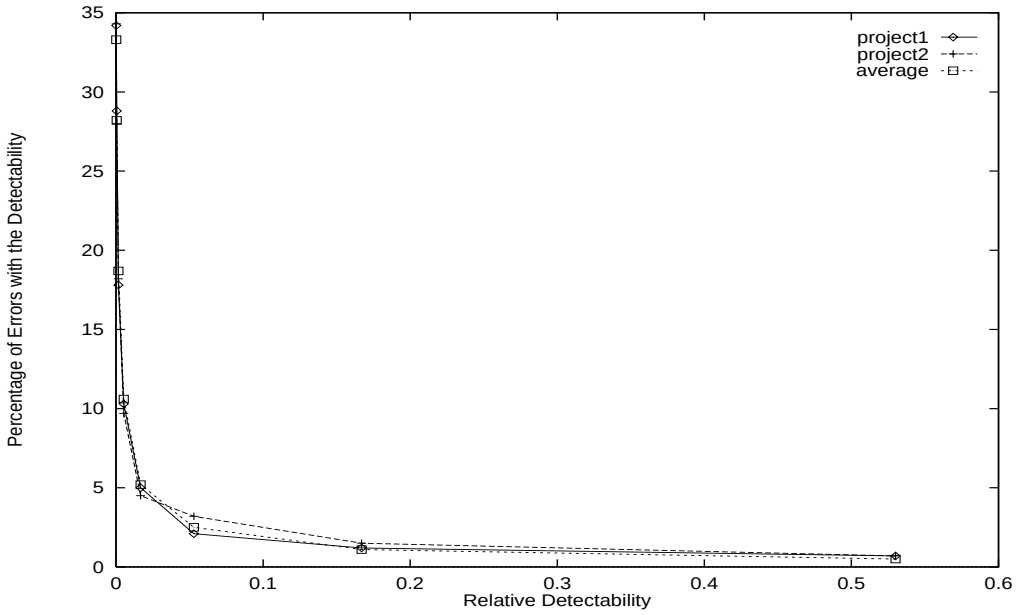

Figure 7: Detectability profile of errors for some operational software

If this is true in general, than there is no need to do coverage testing. However, a close examination suggests that some assumptions implied in the calculation may not hold.

Assumption 1: Usage testing distributes testing effort to faults according to the failure rates: faults with higher failure rates are tested with more effort than faults with lower failure rates. Coverage testing distributes test effort equally to every fault, so a major portion of testing effort is devoted to faults with small failure rate because a majority of faults have small failure rates according to Adam's data.

The fact is that each fault has a certain detectability associated with each input domain. The overall detectability profile of faults in a software depends *partly* on the software's input distribution. In general, bias in input distribution will make faults detectable by inputs from heavily used domains more testable. However, for an arbitrary input distribution, we can not claim that all the faults detectable by inputs from heavily used input domains are more testable than any faults detectable by inputs from less used domains. Both usage testing and coverage testing select test input randomly either to follow certain test input distribution or to achieve certain coverage level, so the error removal process is still dictated by the detectability profiles associated with each input selection. With operational profile based software testing, initially faults associated with heavily used input domains are exercised more often and well-testable faults associated with those domains get removed quickly. So during early software testing, input profile reflecting the software's operational usage *is* efficient. As testing proceeds further, however, the number of such faults with high testability diminishes and only hard to detect faults remain undetected. Now most faults that are

relatively more detectable are associated with less used input domains. Continued testing following the operational profile then becomes inefficient.

Assumption 2: Usage testing and coverage testing is equally effective in terms of the number of failures detected per test input.

The fact is that initially during testing, coverage testing and usage testing may have similar fault detecting ability since they exercise the program in similar way (each input will exercise some new features of the software). As testing proceeds, coverage testing would be more effective because it always tries to select test input such that some new part of a software will be exercised, hence coverage testing is likely to reveal more faults than usage testing for each test input on the average.

Ramsey and Basili noticed that the number of faults detected in a procedure are independent of the number of times the procedure is executed [23]. Piwowarski et al [22] observed that error removal rate and code coverage are closely related by a somewhat linear function. Vouk [28] suggested that the relation between software coverage and fault removal rate follows the Weibull distribution. Chen et al [4] suggested exclusion of the tests that do not contribute to any type of coverage from consideration when using traditional software reliability models. Malaiya et al [14] proposed a new model to relate test coverage to software reliability. These results support that, when the number of tests increases to the point where many additional test inputs based on operational usage do not contribute to more coverage, coverage testing should be more effective in fault detection than usage testing.

Assumption 3: The failure rate distribution remains the same when testing starts and after testing finishes. Adams' data [1] gives the distribution of failures collected from operational use.

For an untested software, the distribution of faults over different detectabilities would be more uniform. Trachtenberg [27] argue that the reason Adams' data follows Zipf's law may be because during software development in IBM, consciously or unconsciously, "the effort to prevent and remove each fault could have been expended in proportion to the fault's potential failure rate". Although no data is available to describe the failure rate distribution of faults for an untested software, it is reasonable to assume that such failure rate distribution must be more uniform in nature. The effect of such changes in software failure rate distribution during testing phases should also be taken into consideration.

As conclusion to this section, more detailed and careful analysis is needed to compare the relative effectiveness of usage testing vs. coverage testing. Quantitative evaluation of their effectiveness remains a problem and calls for more experimentation and experience to fully understand the testing processes.

5 Testing For Reliability

The operational profile of a software system can be used at different stages in the software's lifetime [17]. For the purpose of reliability certification or prediction, software test input selection should follow the software's operational profile [16]. Also, operational profile based testing can be efficient if very limited amount of testing is available. If our main objective of testing is fault removal, operational profile based testing must be supplemented by coverage based testing. Accurate operational profile of a software can be difficult and costly to obtain in some cases but is worth the effort if high reliability levels need to be certified. When accurate operational profile is available, other factors, such as the planned testing effort and the initial software quality also need to be considered because they also affect the effectiveness of testing. When testing a program, we must consider the software usage, but should not rely solely on it.

Like operational profile-based testing, coverage testing has its intuitive appeal. An ideal coverage criterion should be such that it is possible to generate tests manually or automatically to achieve the desired coverage. The number of inputs for a target coverage level should not be too large to be practical, and the chosen level of the measure should satisfy the critical reliability requirement. There should be a strong known correlation between reliability and the coverage measure so that one can accurately estimate and predict the reliability from the coverage measure and determine when testing can be stopped because certain coverage (and hence reliability) has already been reached.

Statement coverage (or block coverage) and branch coverage are the most used coverage measures in practice. Other coverages such as data flow coverages also becomes well-known. Tools are now available for collecting the coverage data of test inputs for some metrics: block, branch, c-use, p-use, all-use [12]. Some work is being done to study the test coverage growth and its relation to fault removal rate or software reliability achieved. For example, Ntafos [20, 21] compared the effectiveness of random testing with that of branch testing and all-uses testing, and observed that coverage testing is much more effective in revealing errors. Ramsey and Basili [23] empirically investigated the relationship between the number of tests and the test coverage. Sneed [25] reported his experience on comparing the effectiveness of branch coverage and data coverage in catching real bugs. Piwowarski et al [22] analytically derived a model characterizing the test coverage growth. Vouk [28] has derived a different model starting with different assumptions. Chen et al [4] proposed incorporating test coverage into traditional time-based software reliability models by filtering out the test efforts that contributes to no new coverage. Frankl and Weiss [8] empirically evaluated the fault exposing capability of branch coverage and data flow coverage criteria. Malaiya et al [14] suggested a hypothesis that different test coverage growths follow an logarithmic trend. Based on this

hypothesis, software fault removal rate and software reliability can be estimated directly from static test coverage measures. Still more empirical data and analytical studies correlating such coverage measures and reliability are needed.

It was noticed that faults are not evenly distributed among program modules. Static metrics have been used to predict error-prone modules. Usage information may be used to estimate the relative use frequencies of program modules or functions. Combination of these knowledges may be used to determine the reliability level for different modules. And then based on the reliability objective, different coverage measures and/or different coverage levels may be associated with different modules to achieve most efficient testing.

6 Conclusions

Our results show that the optimal test input profile for the purpose of defect removal depends on the operational profile and the defect detectability profile of the program. It is also related, to a larger extent, to the amount of testing planned. If only limited testing can be afforded, test input distribution should be more biased than the operational profile. For accurate estimation or prediction of software reliability, testing should be conducted according to the software's operational profile. However, if very high reliability is to be achieved through extensive testing, test inputs should be more evenly distributed among different input domains.

Coverage testing can be very effective in practice. Detailed investigations are needed in this area to examine and evaluate different coverage measures. Work is also needed to relate different coverage measures to software reliability growth. Since some modules are more error-prone than others, and some modules are more critical to a system's operation than others, a family of coverage measures may be chosen eventually to meet different reliability requirements for different modules or different systems.

We suggest that effective software testing requires the knowledge of operational profile, effectiveness of coverage measures, and error-proneness of program modules. However, before this is possible, further empirical and analytical researches are required to better understand these problems.

References

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

[2] B. Beizer, *Software Testing Techniques*, 2nd Edition, Van Nostrand Reinhold, 1990.

[3] J.R. Brown, and M. Lipow The Quantitative Measurement of Software Safety and Reliability, TRW Report QR 1776, August 1973.

[4] M.H. Chen, J.R. Horgan, A.P. Mathur and V.J. Rego, "A time/structure based model for estimating software reliability," *SERC-TR-117-P*, Purdue University, Dec. 1992.

[5] R. Cobb and H. Mills, Engineering Software under Statistical Quality Control, IEEE Software, November 1990, pp. 44-56.

[6] S.R. Dalal, J.R. Horgan and J.R. Kettenring, "Reliable Software and Communications: Software Quality, Reliability and Safety," *Proc. 15th Int. Conf. Software Engineering*, May 1993, pp. 425-435

[7] H.D.Drake and D.E.Wolting, Reliability Theory Applied to Software Testing" Hewlett-Packard Journal, April, 1987, pp. 35-39.

[8] P.G.Frankl and N.Weiss, "An Experimental Comparison of the Effectiveness of Branch Testing and Data Flow Testing," *IEEE Trans. Soft. Eng.*, Aug. 1993, pp. 774-787.

[9] A.L.Goel, "Software Reliability Models: Assumptions, Limitations, and Applicability", *IEEE Trans. Software Engineering,* Vol. December 1985, p. 1411-1423.

[10] H. Hecht and P. Crane, "Rare Conditions and Their Effect on Software Failures" Proc. Annual Reliability and Maintainability Symposium, 1994, pp. 334-337.

[11] Y. Levendal, Improving quality with a Manufacturing Process, IEEE Software, March 1991, pp. 13-25.

[12] M.R. Lyu, J.R. Horgan and S. London, "A coverage Analysis Tool for the Effectiveness of Software Testing" IEEE Int. Symp. on Software Reliability Engineering, 1993, pp. 25-34.

[13] Y. K. Malaiya, A. von Mayrhauser and P. Srimani, "An examination of Fault Exposure Ratio," to appear in *IEEE Trans. Software Engineering*, 1993.

[14] Y.K.Malaiya, N. Li, J.Bieman, R. Karcich and B. Skibbe, The Relation Between Software Test Coverage and Reliability, Technical Report, 1994.

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

[16] H.D. Mills, M. Dyer and R.C. Linger, Cleanroom Software Engineering, IEEE Software, Nov. 1986, pp. 19-24.

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

[18] J.D. Musa, The Operational Profile in Software Reliability Engineering: An Overview, Proc. of International Symposium on Software Reliability Engineering, October, 1992. pp. 140-154.

[19] J.D. Musa, Operational Profiles in Software Reliability Engineering, *IEEE Software,* March 1993, pp. 14-32.

[20] S.C. Natfos, "An Evaluation of Required Elemente Testing Strategies", 7th Int. Conf. on Software Engineering, March 1984.

[21] S.C. Ntafos, "On Required Element Testing" IEEE Trans. on Software Engineering, October 1984, pp. 795-803.

[22] P. Piwowarski, M. Ohba and J. Caruso, "Coverage Measurement Experience During Function Test," ICSE'93, pp. 287-300.

[23] J. Ramsey and V.R. Basili, "Analyzing the Test Process Using Structural Coverage," Proc. 8th international conference on Software Engineering, August 1985, pp. 306-3312.

[24] R. Selby, V. Basili, and F. Baker, "Cleanroom Software Development: An Empirical Evaluation," IEEE Trans. Software Engineering, SE-13(9), 1987, pp. 1027-1037.

[25] H.M. Sneed, "Data Coverage Measurement in Program Testing," *Workshop on Software Testing*, July 1986.

[26] M. Takahashi, and Y. Kamayachi, "An Empirical Study of a Model for Program Error Prediction," IEEE Trans.l Software Engineering, SE-15(1), 1989, pp. 82-86.

[27] M. Trachtenberg, "Why Failure Rates Observe Zipf's Law in Operational Software," IEEE Trans. Reliability, Vol. 41, No. 3, 1992, pp. 386-389.

[28] M.A. Vouk "Using Reliability Models During Testing With Non-operational Profiles," *Proc. 2nd Bellcore/Purdue workshop on issues in Software Reliability Estimation,* Oct. 1992, pp. 103-111