Measurement of Failure Size in Software Testing Techniques

A.Vivek Yoganand[*], Deepan[**]

[*]Computer Science and Engg, VelTech MultiTech Engg College, Chennai, Tamilnadu, India.
[**] Computer Science and Engg, VelTech MultiTech Engg College, Chennai, Tamilnadu, India.

Abstract- Analyzing of failure size and the issues are attained through failure rate. Failure size is the probability of finding an input that causes a failure in the input domain. As testing progresses, failure size decreases due to debugging. The failure size at the termination of testing is called the attained failure rate. Using this measure, we compare the efficacies of partition testing, random testing, derive conditions that lead to the superiority of partition testing, and obtain optimal time allocations in partition testing. The core findings are presented in a decision tree to assist testers in test management.

Index Terms- Partition testing, Random testing, Reduction ratio, Optimization, Failure rate, Measurement of failure size, Code handling, Performance, Examination of code

I. INTRODUCTION

Testing can never completely identify all the defects within software. Instead, it furnishes a *criticism* or *comparison* that compares the state and behavior of the product where principles or mechanisms by which someone might recognize a problem.

A primary purpose of testing is to detect software failures so that defects may be discovered and corrected. Testing cannot establish that a product functions properly under all conditions but can only establish that it does not function properly under specific conditions.

The scope of software testing often includes examination of code as well as execution of that code in various environments and conditions as well as examining the aspects of code: does it do what it is supposed to do and do what it needs to do.

Defects and failures:

Not all software defects are caused by coding errors. One common source of expensive defects is requirement gaps, e.g., unrecognized requirements which result in errors of omission by the program designer. Requirement gaps can often be non-functional requirements such as testability, scalability, maintainability, usability, performance, and security.

Software faults occur through the following processes. A programmer makes an error (mistake), which results in a defect (fault, bug) in the software source code. If this defect is executed, in certain situations the system will produce wrong results, causing a failure. Not all defects will necessarily result in failures.

For example, defects in dead code will never result in failures. A defect can turn into a failure when the environment is changed. Examples of these changes in environment include the software being run on a new computer hardware platform, alterations in source data, or interacting with different software. A single defect may result in a wide range of failure symptoms.

Failure size relate with input. A very fundamental problem with software testing is that testing under *all* combinations of inputs and preconditions (initial state) is not feasible, even with a simple product. This means that the number of defects in a software product can be very large and defects that occur infrequently are difficult to find in testing.

More significantly, non-functional dimensions of quality ,usability, scalability, performance, compatibility, reliability—can be highly subjective; something that constitutes sufficient value to one person may be intolerable to another. In Partition testing which relocate the following test cases?

Test cases for input box accepting numbers between 1 and 1000 using Equivalence Partitioning:

1) One input data class with all valid inputs. Pick a single value from range 1 to 1000 as a valid test case. If you select other values between 1 and 1000 then result is going to be same. So one test case for valid input data should be sufficient.

2) Input data class with all values below lower limit. I.e. any value below 1, as a invalid input data test case.

3) Input data with any value greater than 1000 to represent third invalid input class.

So using equivalence partitioning you have categorized all possible test cases into three classes. Test cases with other values from any class should give you the same result.

Testing Technique Applications

Morasca & Serra-Capizzano[5] introduced an approach for comparing *testing technique applications* (TTA). They believed that the actual application of a testing technique to a program depends on a number of factors.

Hence, it is more meaningful to compare TTA. They have used two measures: number of failures caused by a TTA, and failure probability. They have derived a necessary and sufficient condition for comparing.

However, the whole approach is based on the philosophy of providing more attention to subdomains with higher failure sizes. The desirability of this approach is doubtful in certain circumstances. For example, consider the hypothetical situation of two subdomains in path testing: subdomain A with failure size 0.3, and subdomain B with failure size 0.1. There is a possibility that all failure causing inputs may vanish totally from subdomain A at a single debugging effort, which may not be the case with subdomain B. This result can happen if all the failure causing

inputs of subdomain A are connected with a problem in a particular path, and if that problem is fixed, all failure causing inputs vanish together.

On the contrary, failure causing inputs of B may correspond to three paths. Hence, the expected number of efforts required for B is likely to be larger than A. In this case, continuing to give more attention to subdomain A may not be logical. Zachariah & Rattihalli [2] is a comparison study which differs much from the earlier approaches. The approach in this study (henceforth referred to as the ZR approach) uses a new software reliability model to define the measure, and perform the comparison.

The details of this model and approach are presented in Section III as it involves a number of new concepts which once again forms the background of this paper. Compared to other approaches, the ZR approach introduced more parameters, which were of fundamental importance to the comparison study.

Also, the ZR model reflected the reality of software testing. However, the measure of comparison in Zachariah & Rattihalli [2] is not the best that could have been used. This point will be illustrated in Section IV. This paper brings in another measure from the ZR model of software reliability, and revisits efficacy studies on software testing strategies. This paper lists the possible measures that can be derived from the ZR model. They are compared using desirable properties, and the best measure is chosen. This measure is then used to perform the efficacy studies. In addition, this paper discusses the very important aspect of time allocation in PT. The paper concludes with some practical tips to testers in test management. These tips are presented through a decision tree.

II. Terminology, and General Assumptions

In this terminology, during testing, different inputs are selected following the selection criteria, and administered to the software. The output generated is compared with the desired output. If there is a difference between the two, then a failure is said to have occurred. The set of all such inputs capable of generating a failure is called the failure domain. The failure domain is unknown, and it is a subset of the input domain. At each failure, some changes are introduced into the software such that, when the same input that caused the failure is administered again, the software gives the desired output.

This process is known as rectification. At each rectification, a portion of the failure domain is eliminated. The collection of failure-causing inputs that are fixed exactly by a program change is referred to as a failure region (Zachariah & Rattihalli [2]). Due to the elimination of different failure regions across the testing, the failure domain is expected to shrink. However, improper rectifications can result in the increase of the cardinality of the failure domain.

In Random Testing (RT), each input has an equal probability of getting selected.

Hence, the ratio of the cardinalities of the failure domain to the input domain is the probability of encountering a failure causing input. We refer to this probability as the failure size. As testing progresses, the rectification and the resultant reduction in the cardinality of the failure domain causes the failure size to decrease.

The failure size at any point in time may be viewed as a random variable. Because the cardinality of the failure domain is unknown, its expected value is used as a measure of comparison.

The s-expected failure size at the termination of testing is called the attained failure size. This term is used as a measure of testing efficacy in this paper.

In partition testing (PT), the input domain is divided in a number of sub domains. We assume that these divisions are non-overlapping. In reality, only some of the subtypes like path testing and branch testing will have non-overlapping subdomains. However, we may overcome this difficulty by considering the intersections of overlapping subdomains as yet another subdomain.

Assumptions about the ZR model:

A. A piece of software is subjected to RT, with each input getting an equal probability of selection.

B. The failure counting process N(t) is a Nonhomogeneous Poisson Process with intensity function $\gamma(t)$ proportional to the -expected failure size at time t.

C. At each rectification, a -expected proportion p of the inputs is eliminated from the failure domain, and it does not add any new input to the failure domain.

A3 assumes perfect rectification, which need not always be true. It is possible to modify this model to accommodate this fact. Such modifications will complicate the model, and it will also complicate the estimation of parameters.

Such modifications are needed in certain types of applications, such as determining the optimal stopping time. However, it does not have significance to the comparison of testing strategies. Hence, we assume perfect rectification in this paper.

III. A New Measure of Efficacy in Software

A. Testing Strategies

The measure used for comparison in Zachariah & Rattihalli [2] was the expected number of failures detected up to time t. In efficacy studies, this measure may not be appropriate for certain types of analysis. For example, consider two testing strategies applied on two copies of the same software for the same duration: strategy A may be PT, and strategy B may be RT. It is possible that the number of failures realized under strategy A is greater than the number of failures realized under strategy B. At the same time, it is possible that the attained failure size under strategy A is also greater than what is attained under strategy B, indicating strategy A has achieved lesser towards the goal of failure free software, even though it has generated more failures. This phenomenon is possible because the reduction in the size of the failure domain is determined not only by the number of failures but also by the cardinality of the failure regions eliminated at each rectification.

As already explained, the denominator of the failure size is the cardinality of D, which remains constant. Hence, intensity function is directly proportional to the cardinality of the failure domain. Under RT this assumption is much more realistic instead of assuming the intensity function to be proportional to the

remaining number of errors, which has a hidden assumption that all errors are equally capable of generating failures. In reality, all errors cannot be equal in failure generation ability.

The second reason for this change is to avoid the terms error or fault, which do not have a clear definition. Frankl *et al.* [8] have pointed out that fault is characterized by its fix.

Whenever a program fails, the program is changed so that it will not fail on the same input. Because this change is not unique, we cannot uniquely define a fault.

Thus, better failure generating ability need not indicate better testing efficacy. Thus, we need to have a better measure that will carry the advantage of the ZR model and approach, but does not suffer from the drawback of the measure "expected number of failures."

B. Selection of New Measure

It is important to think about the various measures that may be used for comparing the strategies, evaluating them, and picking the best. This section is devoted to that purpose. There are four possible measures of interest that can be obtained from the ZR model.

Mean number of failures at time t,

$$m(t) = \frac{1}{p} \ln\{1 + \frac{bpft}{d}\} \quad (1)$$

E (Cardinality of F at time t),

$$E(|F|(t)) = f(e^{-p})^{m(t)}$$

Substituting for m(t) from (1), and simplifying, we get

$$E(|F|(t)) = \frac{f}{[1+\frac{bpft}{d}]} \quad (2)$$

E (Failure size of D at time t),

$s(t) = E($*Ratio of cardinality of failure domain to input domain at time t*$)$

$$= \frac{E(|F|(t))}{|D|} = \frac{E(|F|(t))}{d}$$
$$= \frac{f}{[d + bfpt]}$$

Reliability,

$R(x|t)$ = P (Software will not fail for an additional x units of time given that it was tested for t units of time)

= P (No failure occurring between time points t and $t + x$)

From the Poisson probability, it is given as

$$R(x|t) = e^{-\{m(t+x)-m(t)\}}$$

Table -I
Comparison of Measures

Criteria of comparison in the order of priority	Measure			
	m(t)	E(|F|(t))	s(t)	R(x|t)
Reflective ability	Poor	Good	Very Good	Very Good
Freedom from additional assumptions	Yes	Yes	Yes	No

Where m(t+x)-m(t) is the mean number of failures encountered in the interval (t, t+x). Substituting for m(t+x) and m(t) .Where simplifying from the equation (1) we get

$$R(x|t) = \left[d + \frac{bfp(t+x)}{d} + bfpt\right]^{-\frac{1}{p}} \quad ---(3)$$

These measures should be compared to identify the best among them. Measures can be compared by using the desirable properties as criteria of comparison. For this purpose, four desirable properties are identified.

1) Reflective ability: This property is the ability of the measure to reflect the status of the software in relation to freedom from failures.
2) Freedom from additional assumption: It is desirable that the measure is free from assumptions beyond the testing period. For example, m(t) has assumptions about the testing, and behavior of the software during testing. But $R(x|t)$ has not only assumptions about the testing time t, but also assumptions about the additional x units beyond the testing time t.
3) Freedom from additional information: It is also desirable that the measure is free from information beyond the testing time. In the case of $R(x|t)$ we need to know the value of x, the additional number of units in which the user expects the software to be failure free.
4) Mathematical tractability: This property is desired from the point of view of performing mathematical analysis. To facilitate the comparison of the measures stated earlier, these desirable properties are arranged in Table I below, in the order of priority. Naturally, reflective ability gets the highest priority. The next two priorities are given to freedom from additional assumption and freedom from additional information, because of practical considerations. The comparisons are shown in Table I. Explanations follow the table. In regard to reflective ability, except for m(t), all measures are rated good or very good.

With regard to the third, $R(x|t)$ has a drawback that x needs to be specified, and it may vary from user to user. The other three measures have no problem with this criterion. With regard to mathematical tractability, $R(x|t)$ is a poor measure because it has a complex mathematical expression? Hence, when we refer

the comparison table in relation to the priority, s(t) is the clear choice as the measure in efficacy studies. Having established the superiority of s(t) over m(t), let us also note that s(t) is a non-linear function of m(t) (from (8)).

Hence, m(t) cannot replace s(t). It is important to note that, unlike m(t), the desirability of a testing strategy is attached with smaller values of s(t). Observe that our measure of comparison s(t) is the expected failure size of D at time t, which is the expectation of the ratio of cardinality of the failure domain to the input domain at time t. We work with expected failure size because the actual failure size will be unknown. The expected failure size of D at the termination of testing is called the attained failure size.

Numerical Comparisons between PT (partition testing) and RT (random testing):

In this section, we will take up numerical examples to illustrate that there does not exist unconditional superiority with either of the testing strategies.

To perform this result, let us consider a newly developed random number generator providing values from the Uniform (0, 1) distribution.

Because the simulation experiment has a large number of computations over a large number of observations, it is desirable that generators work very fast. Assume that 2000 of the 10,000 inputs results in failure; these 2000 inputs constitute the failure domain.

Further, assume that we are able to test 10 inputs on a working day. Assume that testing continues for 3 days. Further assume that a proportion of 0.175 inputs are eliminated from failure domain at each rectification.

Thus, we have

d=10000, f=2000, p=0.175, b=10 and t=3

From these values we have $SR(3) = 0.098$

This value will be used for comparison with SP(.) in this section

Table -II
Comparison Between Pt And Rt For Various Time Allocations

Ex no.	t_1	t_2	$s_P(t_1,t_2)$	Remark
1	0.5	2.5	0.094	$s_P(.) < s_R(.)$
2	0.9	2.1	0.098	$s_P(.) = s_R(.)$
3	2	1	0.121	$s_P(.) > s_R(.)$

Partitioning on this input space is worth studying. The nature of the generated sequence depends on the seeds. The seeds are selected by the user who does not know its impact on the generated sequence, hence the selection is pseudo-random. This approach makes the cycle length and uniformity of the generated sequence unpredictable.

Seeds may be classified by different methods. Classifying numbers as odd or even is one way. If there are reasons to believe that odd numbers (even numbers) will behave in a similar way, then the distinction is useful. Such a partition will have two subdomains with equal cardinality. There are many other possible classification methods. Sometimes, multiples of 3 may be considered as one partition, and the rest as another. The partition strategy depends on the basic algorithm, and the parametric values that are used.

Still there is lot of unpredictability on the effect of partition. When new software is developed based on a new algorithm, nobody Can say for sure the effect of any type of partition. Comparison Studies suggested in this paper will help resolve such Issues.

Observation 1: PT can be better, worse, or the same as RT depending on the choice of .

Illustration: Consider the following two subdomains of the input domain described earlier. All the parametric and time values of this illustration are selected intentionally to reveal the three distinct possibilities stated in Observation 1.

Subdomain1: d_1=5000, f_1=500, p_1=0.1
Subdomain2: d_1=5000, f_1=1500, p_1=0.3

Table II gives the comparison between PT and RT when different time allocations are given to the same partition.

Observation 2: Even when the time spent on two partitions is equal, PT can be better, worse, or the same as RT depending on how the partitioning is done.

Illustration: Consider the examples in Table III. Each example represents a different type of partitioning. All the examples have t_1 =1.5 and t_2=1.5

All the parametric values of this illustration are selected intentionally to reveal the three distinct possibilities stated in Observation 2

We take p_1=0.1, p_2=0.3, f_1=500 and f_2=1500 in all the examples in the Table III.

This table shows that, even under the same time allocation, different kinds of partitions yield different results in comparison with RT. From the above observations, it is clear that unconditional superiority does not exist with either of the testing strategies.

Hence, in the next subsection, we pursue our investigation under specified conditions.

Table III
Comparison Between Pt And Rt For Various Partitions

This table shows that PT can be better, worse, or the same as RT depending on the choice of the t1.

Ex no.	d_1	d_2	$s_P(.)$	Remark
4	7100	2900	0.090	$s_P(.) < s_R(.)$
5	6300	3700	0.098	$s_P(.) = s_R(.)$
6	2100	7900	0.118	$s_P(.) > s_R(.)$

IV. SUMMARY AND CONCLUSION

This paper uses attained failure size for performing analysis n testing strategies. The advantages of this new approach are analyzed in detail. We have derived specific conditions that lead to the superiority of PT over RT.

A sure way of obtaining superior performance in PT over RT (when the subdomains have equal reduction ratios) is allocating time proportional to the cardinality of input domains.

Allocating time inversely proportional to the reduction ratios of input domains is a way of obtaining superior performance in PT when the subdomains have equal sizes, provided the weighted arithmetic mean of reduction ratios of failure subdomains (with respect to time) is greater than or equal to k times the RR of the failure domain.

Similarly, allocating time inversely proportional to the reduction ratios of input domains is a way of obtaining superior performance in PT when the subdomains have equal sized failure subdomains, provided the weighted arithmetic mean of reduction ratios of failure subdomains (with respect to time) is greater than or equal to k times the RR of the failure domain.

Further, a method of obtaining the optimal time allocations in PT is described, which is free from preconditions.

Still, a good amount of work needs to be done in this area. Incorporating the cost aspect of testing in this type of analysis could be an interesting study. Extending this method over nonuniform operational profiles is another important task.

REFERENCES

[1] P. J. Boland,H. Singh, andB. Cukic, "Comparing partition and random testing via. Majorization and Schur functions," IEEE Trans. Software Eng., vol. 29, no. 1, pp. 88–93, 2003.

[2] B. Zachariah and R. N. Rattihalli, "Failure size proportionalmodels and an analysis of failure detection abilities of software testing strategies," IEEE Trans. Reliability, vol. 56, no. 2, pp. 246–253, 2007.

[3] W. J. Gutjahr, "Partition testing vs. random testing: The influence of uncertainty," IEEE Trans. Software Eng., vol. 25, no. 5, pp. 661–674, 1999.

[4] A. W. Marshall and I. Olkin, Inequalities: Theory of Majorization and its Applications. : Academic Press, 1979.

[5] S. Morasca and S. Serra-Capizzano, "On the analytical comparison of testing techniques," in ISSTA'04, pp. 154–164.

[6] V. N. Nair, D. A. James, W. K. Ehrlich, and J. Zevallos, "A statistical assessment of some software testing strategies and application of experimental design techniques," Statistica Sinica, vol. 8, no. 1, pp. 165–184, 1998.

[7] S. C. Ntafos, "On comparisons of random, partition and proportional partition testing," IEEE Trans. Software Eng., vol. 27, no. 10, pp. 949–960, 2001.

[8] P. G. Frankl, R. G. Hamlet, B. Littlewood, and L. Strigini, "Evaluating testing methods by delivered reliability," IEEE Trans. Software Eng.,vol. 24, no. 8, pp. 586–601, 1998.

AUTHORS

First Author – Vivek Yoganand is a faculty member at VelTech Multitech Institute of Computer Engineering Education and Research, Chennai, India. He teaches Software Testing and Operating System Techniques. He received his B.E in Computer Science Engineering from Anna University, Chennai (India), and M.Tech in Multimedia Technology from SRM University, Chennai (India). His current research interest is in software testing, image processing and reliability.

Second Author – Deepan is a faculty member at VelTech Multitech Institute of Computer Engineering Education and Research, Chennai, India. He teaches Compilers and Programming Techniques. He received his B.E in Computer Science Engineering from Anna University, Chennai (India), and M.Tech in Computer Science from SRM University, Chennai (India). His current research interest is in Compiling, Random image Mining and Optimization of sematics.