Using Reliability Models During Testing With Non-Operational Profiles*

M. A. Vouk
Computer Science Department, Box 8206
North Carolina State University
Raleigh, NC 27695-8206
Tel: 919-515-7886
e-mail: vouk@adm.csc.ncsu.edu

Extended Abstract

Operational profile is a set of relative frequencies of occurrence of the run categories associated with the product and its operational use. During operation system executes a series of runs which are selected from the available run categories at random (but according to the operational profile). Software (and system) reliability growth models model the fault removal process during product testing in order to make inferences about its behavior in operation. Practically all available software reliability models assume failure detection based on operational profiles. This assumption is usually violated during early software testing phases (e.g., unit testing and integration testing phases). Consequently, Software Reliability Engineering assessment and control of project or product quality growth during non-operational testing stages requires consideration of several factors, and interpretation of classical software reliability models becomes difficult and may be deceptive. Models specifically tailored for evaluation of non-operational testing are needed.

We have investigated a family of models which can be used to describe, and predict, failure detection process during non-operational software testing. The models are based on a family of testing models which we believe apply in the case of non-operational testing. The models combine the concepts of functional coverage and code coverage with the testing strategies observed in some practical systems. It is shown that the dynamics of the testing processes translates into a Weibull failure detection model. Weibull-type model using time as exposure was considered by, for example, Wagoner [Wag73] although not in the context of non-operational profile testing. Also, the Shick-Wolverton model can be interpreted as a special case of the Weibull model class [Shi73, Mus87].

Our models were originally developed and verified using the data from a multi-university

*Research supported in part by NASA Grant No. NAG-1-983
multi-version NASA sponsored program [Eck91, Vou90, Kel88]. Recently an attempt was made to use the models during the early testing stages of a very large telecommunications system. The results obtained in this industrial environment are very encouraging and appear to indicate that the models can be successfully used to monitor and control risks associated with software testing processes that occur during non-operational (accelerated?) testing of software in phases where operational testing is not feasible or is too expensive or too slow.

**Testing**

One, polarized, representation of the testing process that we assume takes place during non-operational testing is the "sandwich" or "layered" model. The other extreme is the "disjoint" or "functional group" model. Both models, in different mixes, are expected to be in operation during unit level, integration, and even system testing. A possible representation of the two cases is shown in Figure 1.

In the "layered" model each test case exercises a hierarchy of functions, the more fundamental functionalities being exercised more frequently due to their inclusion into most of the test-case paths. During this type of functional testing many sub-functions (simple and complex) are "exercised" with each test case and this in practice provides a continuous and repeated (re-) sampling of the whole program. In situations like this use of "classical" failure intensity decay models may be possible.

![Figure 1. A possible representation of the "Layered" (left) and "Functional Group" (right) Testing Models.](image-url)

In the "functional groups" model software functionalities exercised by the test cases are decoupled to a degree where they are practically independent of each other and each test case essentially...
exercises an independent function, so that, to a large extent, the testing process is akin to sampling of program functions without replacement. In situations like this failure intensity may not follow the "classical" decay profile and may exhibit modes. The differential equations that describe this latter process yield a Weibull failure intensity in the general case. A special instance of that model is the Rayleigh model.

The actual field progress of the non-operational testing process can be monitored using the cumulative failure profiles, different "flavors" of the failure intensity profiles, and different exposure metrics. We have experimented with instantaneous, average, modeled, weighted, etc., failure intensities. The "exposure" can be the CPU execution time, calendar execution time, fraction of the planned test cases executed, the number of test cases executed, coverage information in the classical sense (e.g., branch coverage, data-flow coverage), etc. When the models are applied to sub-systems of a larger system, failure intensity trends, absolute and relative intensity target values, and historical information obtained for all components are combined into an integrated model for evaluation of the non-operational testing process.

**An Illustration:**

**Rayleigh Fault Removal Model Based on the "Functional Groups" Testing Model**

Testing coverage \( C(M,S) \) is computed for a construct quantified through metric \( M \) and testing strategy \( S \) by the following expression

\[
C(M,S) = \frac{\text{No. of executed constructs for } M \text{ under } S}{\text{Total no. of constructs for } M \text{ under } S}
\]

Notice that the above expression does not allow for infeasible (unexecutable) constructs under a given testing strategy \( S \). Hence, in general, there may exist an upper limit on the value of \( C(M,S) \), \( C_{\text{max}} \leq 1 \). Classical examples of unit testing oriented control flow metrics are lines of code (executable), linear blocks of code, branches, paths, etc. Similarly, examples of data flow metrics are all definitions (in predicates and linear blocks), all uses (in predicates and linear blocks), all definition-use tuples, all definition-use paths, etc. However, the logic applies equally to coverage of planned test cases, planned execution time, etc.

**Some Assumptions**

(1) The coverage based testing and fault removal, in the first approximation, is similar to sampling without replacement.

To increase (fulfill) coverage \( C(M,S) \) of metric \( M \) we generate test cases using strategy \( S \) which would cover as many yet uncovered constructs as possible. The "re-use" of constructs through new paths that exercise at least one new construct (i.e. increase \( M \) by at
least 1) is effectively "ignored" by the metric M.

(2) The order of execution of test cases is ignored (and is assumed random) unless otherwise dictated by the testing strategy S.

(3) The fault detection rate with respect to coverage is proportional to the coverage, \( 1 \geq C(M,S) > 0 \).

(4) For each test set T generated under S and monitored through M, there is a minimal coverage, \( 1 \geq C(M,S) \geq C_{\text{min}} \geq 0 \), that results in a recorded change in M through execution of a (typical) test case from T.

In general, execution of a complete entry to exit path in a program will exercise more than one M construct of P.

(5) For each test set T generated under S and monitored through M, there is a maximum coverage, \( C_{\text{max}} \geq C(M,S) \geq 0 \), that can be achieved.

In general, execution of a complete entry to exit path in a program will exercise more than one M construct of P.

(6) The rate of fault detection with respect to coverage is also proportional to the effective number (or density) of residual faults, \( \varepsilon_r (\xi_r) \) detectable by metric M under strategy S.

Not all faults present in the code may be detectable under a given testing strategy. For instance, faults that require at least two passes through a loop may not be detectable by a strategy S that requires the test cases to cover all branches only once.

**Figure 2.** M-space of program P

Consider the metric space (M-space) offered by a program and schematically depicted in Figure 2. Let each square represent a construct of metric M (e.g. du-pair, a branch, a sub-path, a functionality, a test case). Let the dark blobs represent faults detectable through path synthesis of constructs of M. Different paths (jumps from one M square to another) from the Entry to the Exit
of program P will exercise some already "used" M-constructs, and some new ones. Only the latter matter as far as increments in the current coverage model are concerned. As the number of unexecuted M constructs shrinks the probability of "trapping" a M-detectable fault that remains increases. We assume that in general this probability is a function of the achieved coverage. In this particular example we assume that it is proportional to the achieved coverage. However, note that if strict sampling without replacement is assumed the probability mass function is hypergeometric [Tri82]. The hypergeometric software failure model has been studied by Tohma et al [e.g. Toh89].

Let each fault, X, have associated with it probability \( p(M,S,X) \) that it is detectable by M under S. Then

\[
\varepsilon_r = \sum_{i=1}^{E_r} p(M,S,X_i)
\]

where \( E_r \) is the actual number of faults remaining. Note that the effective number of faults (initial, detected, remaining, etc.) can be normalized over the total number of M constructs in program P (density).

![Figure 3. Probability of Detecting X by M under S](image)

To understand the concept consider the structure shown in Figure 3. Let the testing strategy be: "cover all branches at least once". Then the full branch coverage can be achieved in many ways. For example via paths:
(1-2-4-5-7, 1-3-4-6-7), or (1-2-4-6-7, 1-2-4-5-7), or (1-2-4-5-7, 1-2-4-6-7, 1-3-4-5-7), etc.

Only some of these path combinations detect the error in node 3 because they have to synthesize subpath 1346 (under the assumption that the use of B in 6 detects the error). The rest of the subpaths are error-insensitive. The probability of detecting this particular fault by branch testing could be estimated by taking the ratio of all path combinations that provide full branch coverage and detect the fault, and the total number of these combinations.

A Simple Model
From assumptions it follows that the fault detection rate with respect to coverage is

$$\frac{d\varepsilon_d}{dC} = k \varepsilon_r (C - C_{min})$$

where $\varepsilon_d$ is the effective number (or density) of detected faults, and $C$ is $C(M,S)$.

Under another simplifying assumption that fault correction is instantaneous and perfect (i.e., no fault generation, $\varepsilon_g = 0$), the effective number of corrected faults, $\varepsilon_c$, is equal to effective number of detected faults. So the effective number (or density) of residual faults is

$$\varepsilon_r = \varepsilon_T - \varepsilon_c$$

where $\varepsilon_T$ is the total (effective) number of (M-detectable) faults in program P at coverage $C=0$. Hence

$$\frac{d\varepsilon_c}{dC} = k (\varepsilon_T - \varepsilon_c)(C - C_{min})$$

Solution of this differential equation in the range $0 \leq C \leq 1$, under initial condition $\varepsilon_c = 0$ for $C \leq C_{min}$

$$\frac{d\varepsilon_c}{\varepsilon_T - \varepsilon_c} = k (C - C_{min})dC$$

yields

$$\ln(1 - \frac{\varepsilon_c}{\varepsilon_T}) = \frac{1}{2} k (C - C_{min})^2$$

or
\[ \varepsilon_T = \varepsilon_T[1 - e^{-\beta(C-C_{\text{min}})^2}] \]

This illustrates that the fault removal growth offered by structured based testing can be modeled by a variant of the Rayleigh distribution, i.e. a special case of the Weibull distribution.

**Stopping Rules**

Because the probability of "trapping" an error increases with the coverage, stopping before full coverage is achieved makes sense only if the residual error count has been reduced below a target value. This threshold coverage value will tend to vary even between functionally equivalent implementations. The model also shows that metric saturation can occur, but that additional testing may still detect errors. This is in good agreement with the experimental observations. For some metrics, such as fraction of planned test cases, this effect allows introduction of the concept of "hidden" constructs, re-evaluation of the "plan" and estimation of the additional "time" to target intensity in a manner similar to "classical" time-based models [e.g., Mus87].

**Experimental Results**

Some of the experimental results that support the Rayleigh nature of the model are shown in Figures 4 and 5. Other results, not shown here, support use of a more comprehensive Weibull model, with the Rayleigh model being only a special case.
In Figure 5 we see the experimental data and the model fit for the p-use coverage achieved with certification testing in one of the multi-version programs. The right intercept of the fitted curve with the y axis at C=1 can be an indicator of the number of remaining faults, although for acceptable confidence bounds the coverage has to be close to 1.

Another interesting property of the model is that it allows us to classify the relative power of the coverage metrics in a way that the usual hierarchies do not. The relative power estimate is specific to the application and the implementation in question. It is shown in Figure 6 where we plot the logarithm of

$$1 - \frac{\varepsilon_C}{\varepsilon_T} \leq e^{-\beta(C-C_{\min})^2}$$

with $\varepsilon_T=9$, the known total number of faults. The sharper the slope, $\beta$, of the curves and the smaller the $C_{\min}$, the higher is the potential of the metric for detecting faults. From the graph we see that the relative power of the depicted coverage metrics is

$\text{block : p-use : DUD-chains} \approx 1 : 2 : 6$
Figure 6. Relative power of block, p-use and dud coverage testing.

Work is in progress on an integrated model for process and quality (reliability and availability) description and control during non-operational testing of large multi-component software based systems.

References