

# A Markov Model for Software Code Construct Coverage and Fault Detection

Michael Grottke

Chair of Statistics and Econometrics  
University of Erlangen-Nuremberg  
Lange Gasse 20, D-90403 Nuernberg, Germany  
*Michael.Grottke@wiso.uni-erlangen.de*

## Abstract

In this paper, variations of a model bridging the gap between classical software reliability models and failure models designed for systematic testing are described. The expected values of the numbers of code constructs in the different states defined are formulated as non-homogeneous Markov chains.

## 1 Basic Markov model for software code construct coverage

A basic proposition of software reliability engineering is that the failure data analyzed has been collected while testing the software according to its operational profile. On the one hand, this requirement makes sure that reliability estimations and predictions based on the test data also hold for the user environment. On the other hand, the derivations of classical software reliability models explicitly or implicitly rely on the assumption that the testers show a certain behavior that is related to operational testing. Piwowarski, Ohba and Caruso (1993), for example, demonstrate that the mean value function of the well-known Goel-Okumoto model (cf. Goel and Okumoto (1979)) arises when code constructs (e.g., code blocks) are sampled with replacement by equally-sized test cases. This setup resembles operational testing with a homogeneous operational profile.

The vast majority of industrial organizations actually employs systematic testing techniques (e.g., equivalence partitioning). One goal of such strategies is to test as much of the software under test within the given time and budget constraints. Therefore, they try to avoid repeated executions of code constructs already exercised before. It is clear that classical software reliability models, which are based on operational testing, may not be appropriate for modelling the shape of the software failure pattern encountered by testers following a systematic approach to testing.<sup>1</sup>

This gives rise to the question what a model for failure data collected during systematic testing should look like. A highly simplifying approach is proposed by Rivers and Vouk (1998): They assume that no code construct is exercised twice. Since systematic testing techniques cannot accomplish perfect avoidance of repeated code construct executions, a more realistic model would have to allow for *partial* redundancy in code construct sampling.

For that end, we distinguish between three different states a code construct may take: “untested” ( $U$ ), “already tested with the possibility of being tested again in future” ( $T$ ) and “tested and eliminated from further consideration” ( $E$ ). Let  $G_{A,i-1}$  denote the number of code constructs residing in state  $A$  before execution of the  $i^{th}$  test case, where  $A \in \{U, T, E\}$ . We make the following assumptions about how code coverage is attained:

1. The program under test consists of  $G$  code constructs. At the beginning of testing, all constructs are in state  $U$ , i.e.,  $G_{U,0} = G$ .
2. Per test case,  $p$  constructs are sensitized on average.

---

<sup>1</sup>This argument does not imply the notion that systematic testing techniques actually are *better* (i.e., more efficient in detecting the faults that matter most to the customers) than operational testing. They are *different* - and these differences show in the shape of the software failure pattern; hence the need for different models.

3. For each of the constructs residing in either state  $U$  or state  $T$  at the beginning of the execution of the  $i^{th}$  test case the probability of being exercised by the test case is the same,  $\frac{p}{G_{U,i-1}+G_{T,i-1}}$ .
4. On average, a constant fraction  $r$  ( $0 \leq r \leq 1$ ) of those constructs exercised by a test case changes to (or stays in) state  $T$  and may be tested again in future. The other constructs are eliminated and take state  $E$ .

According to this setup, the *expected values* of the numbers of code constructs in each of the three states after the  $i^{th}$  test case has been executed are determined by the expected values of these quantities after the execution of the  $(i-1)^{th}$  test case. Therefore, the sequence of the expected values forms a Markov chain:

$$\begin{pmatrix} E(G_{U,i}) \\ E(G_{T,i}) \\ E(G_{E,i}) \end{pmatrix} = \begin{pmatrix} 1 - z_i & 0 & 0 \\ z_i r & 1 - z_i(1 - r) & 0 \\ z_i(1 - r) & z_i(1 - r) & 1 \end{pmatrix} \begin{pmatrix} E(G_{U,i-1}) \\ E(G_{T,i-1}) \\ E(G_{E,i-1}) \end{pmatrix} \quad (1)$$

The expected transition probabilities from state  $A$  to state  $B$ ,  $\pi_{AB,i}$  ( $A, B \in \{U, T, E\}$ ), depend on the probability that one specific construct of those constructs expected not to have been eliminated during the first  $(i-1)$  test cases is sensitized by the  $i^{th}$  test case,  $z_i = \frac{p}{E(G_{U,i-1})+E(G_{T,i-1})}$ . In figure 1, the structure of this Markov model is depicted.

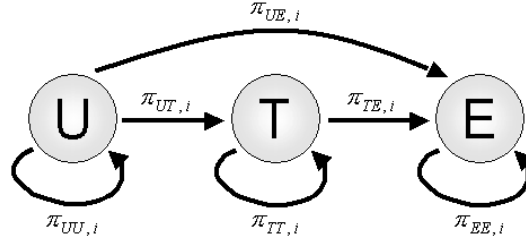


Figure 1: Structure of the basic Markov model

We are interested in  $\frac{E(G_{T,i})+E(G_{E,i})}{G}$ , the expected code coverage achieved as a function of the number of test cases executed,  $i$ . It can be shown that  $\kappa(i)$ , the continuous approximation of this function, takes the form<sup>2</sup>

$$\kappa(i) = \begin{cases} 1 - (1 - \frac{p}{G}(1 - r)i)^{\frac{1}{1-r}} & \text{if } 0 \leq r < 1 \text{ and } i \leq [\frac{p}{G}(1 - r)]^{-1} \\ 1 - \exp(-\frac{p}{G} \cdot i) & \text{if } r = 1 \end{cases} \quad (2)$$

This result confirms that the Markov model is a generalization of the approaches by Piwowarski et al. and by Rivers and Vouk: For  $r = 1$ , i.e., if constructs are never eliminated, the exponential model derived by Piwowarski et al. is obtained. Assuming perfect avoidance of redundancy ( $r = 0$ ), on the other hand, leads to a linear relationship between the number of test cases executed and code coverage achieved, like in the setup proposed by Rivers and Vouk. Apart from these extremes, the model also contains the more realistic cases of partial redundancy in code construct sampling.

## 2 Extended Markov models for fault detection

The Markov model discussed in the last section can be extended to include fault detection and correction. This requires the distinction between faulty and correct code constructs. Let there be six different states a code construct may take: “untested and correct” ( $UC$ ), “untested and faulty” ( $UF$ ), “tested and correct” ( $TC$ ), “tested and faulty” ( $TF$ ), “eliminated and correct” ( $EC$ ) and “eliminated and faulty” ( $EF$ ). Consequently, the four assumptions of the basic model have to be reformulated as follows:

1. The program under test consists of  $G$  code constructs. At the beginning of testing,  $u_0$  of these constructs are in state  $UF$  (i.e., they are untested and faulty); the remaining  $(G - u_0)$  constructs are in state  $UC$ .

<sup>2</sup>For details concerning this derivation as well as the following ones please refer to Grottko (2001).

2. Per test case,  $p$  constructs are sensitized on average.
3. For each of the constructs residing in one of the states  $UF$ ,  $UC$ ,  $TF$  and  $TC$  at the beginning of the execution of the  $i^{th}$  test case the probability of being exercised by the test case is the same,  $\frac{p}{G_{UF,i-1}+G_{UC,i-1}+G_{TF,i-1}+G_{TC,i-1}}$ .
4. On average, a constant fraction  $r$  ( $0 \leq r \leq 1$ ) of those constructs exercised by a test case may be tested again in future. If such a construct is faulty after the test case execution, it changes to (or stays in) state  $TF$ ; if it is correct after the test case execution, it moves to (or remains in) state  $TC$ . The other constructs are eliminated and take state  $EF$  or state  $EC$ , respectively.

The rephrasing of the assumptions lays the foundation for the model extension, but it does not change expected code coverage growth. Equation (2) still holds true. However, these assumptions are not sufficient for determining  $N - E(G_{UF,i}) - E(G_{TF,i}) - E(G_{EF,i})$ , the expected number of failure occurrences as a function of the number of test cases executed, as well as  $\mu(i)$ , its continuous approximation. Additional assumptions have to be made.

Several existing models, e.g. the ENHPP framework by Gokhale et al. (1996), suggest that executing a faulty construct may only result in a failure occurrence if the construct is exercised *for the first time*. Within the Markov model such a proposition can be formulated as follows:

5. When a code construct at which a fault is located is exercised for the first time, the fault causes a failure with activation probability  $s$  ( $0 < s \leq 1$ ). The fault is then removed instantaneously and perfectly. If no failure occurs during the first execution of the code construct, then the fault will not be detected until the end of testing.

In the structure of the resulting Markov model shown in figure 2(a) “transitions” within one state have been omitted in order to maintain a clear diagram.

From the five model assumptions given follows the mean value function

$$\mu(i) = u_0 s \kappa(i) = \begin{cases} u_0 s \left[ 1 - \left( 1 - \frac{p}{G} (1-r) i \right)^{\frac{1}{1-r}} \right] & \text{if } 0 \leq r < 1 \text{ and } i \leq \left[ \frac{p}{G} (1-r) \right]^{-1} \\ u_0 s \left[ 1 - \exp \left( -\frac{p}{G} \cdot i \right) \right] & \text{if } r = 1 \end{cases} \quad (3)$$

This result fits well with intuition: If fault detection is possible at the first execution of a faulty construct only, then the number of failure occurrences is proportional to the expected level of code coverage attained.

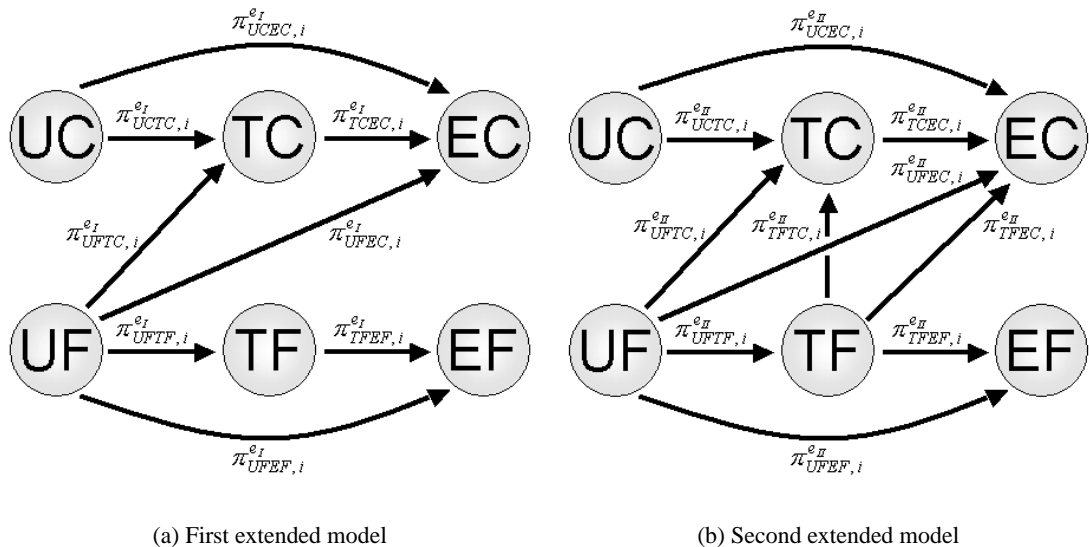


Figure 2: Structures of the two extended Markov models

However, a code construct can be exercised several times with variables taking different values. Therefore, a fault may well be revealed even if the faulty code construct has already been covered before. To account for that, assumption 5 of the first extended model is replaced by the following one:

5. When a code construct at which a fault is located is exercised for the first time or repeatedly, the fault causes a failure with constant activation probability  $s$  ( $0 < s \leq 1$ ). The fault is then removed instantaneously and perfectly.

This proposition adds two more transitions with transition probabilities larger than zero to the Markov structure of the expected numbers of code constructs in the different states (cf. figure 2(b)). The corresponding mean value function takes the form

$$\mu(i) = \begin{cases} u_0 \frac{s}{1-r+rs} \left[ 1 - \left( 1 - \frac{p}{G}(1-r)i \right)^{\frac{1-r+rs}{1-r}} \right] & \text{if } 0 \leq r < 1 \text{ and } i \leq \left[ \frac{p}{G}(1-r) \right]^{-1} \\ u_0 \left[ 1 - \exp \left( -\frac{p}{G}si \right) \right] & \text{if } r = 1 \end{cases} . \quad (4)$$

Clearly, for  $0 < s < 1$  the expected number of failure occurrences is not proportional to the expected code coverage achieved. Rather, as shown by Grottke (2001), in this case the derivative of  $\frac{\mu(i)}{\kappa(i)}$  with respect to the number of test cases executed is positive. This means that as testing proceeds relatively more failures occur per percentage of newly gained code construct coverage. Furthermore, the expected number of faults detected at full coverage according to this model variation,  $u_0 \frac{s}{1-r+rs}$ , is larger than the corresponding value in the first extended model,  $u_0s$ . The reason for these properties of the second model variation lies in the fact that it allows for positive effects of redundancies in sampling code constructs: Faults in those constructs already covered before may still be detected.

### 3 Conclusions

The model presented in this paper contains setups related to operational testing as well as systematic testing. It also includes for the more realistic intermediate cases of partial redundancy in sampling code constructs.

### Acknowledgements

The research described in this paper was done in the course of the project PETS. This project is supported by the European Community in the framework of the specific program for research, technological development and demonstration on a user friendly society (1998-2002), the ‘‘IST Program’’. The author is solely responsible for this paper. It does not represent the opinion of the Community.

### References

- Goel, A. L. and K. Okumoto (1979). Time-dependent error-detection rate model for software reliability and other performance measures. *IEEE Trans. Reliability* 28, 206–211.
- Gokhale, S. S., T. Philip, P. Marinos, and K. S. Trivedi (1996). Unification of finite failure non-homogeneous poisson process models through test coverage. In *Proc. Seventh International Symposium on Software Reliability Engineering*, White Plains, pp. 299–307.
- Grottke, M. (2001). Modelling structural coverage and the number of failure occurrences with non-homogeneous markov chains. Technical Report 41/2001, Chairs of Statistics, University of Erlangen-Nuremberg.
- Piwowski, P., M. Ohba, and J. Caruso (1993). Coverage measurement experience during function test. In *Proc. Fifteenth International Conference on Software Engineering*, Baltimore, pp. 287–301.
- Rivers, A. T. and M. A. Vouk (1998). Resource-constrained non-operational testing of software. In *Proc. Ninth International Symposium on Software Reliability Engineering*, Paderborn, pp. 154–163.