Failure Risk Estimation via Markov Software Usage Models

Walter J. Gutjahr
Department of Statistics, Operations Research and Computer Science
University of Vienna, Austria

Abstract

A software usage models describes the prospective use of a program in its intended environment and allows the generation of random test cases leading to unbiased estimates of the failure risk, i.e., the expected loss by program failure. We concentrate on usage models of Markov type and show that by suitable changes of the probabilities of state transitions during test, the precision of the risk estimate can be optimized. An algorithm for the computation of optimal transition probabilities is presented, and experimental results based on a C++ implementation of this algorithm are reported.

1 Introduction

Recently, software usage models of Markov type have found considerable interest (see [10, 11, 12, 9]). The purpose of a software usage model is to give a formal description of the expected operational use of a software system, i.e., its use in its intended application environment. Such a model is an essential prerequisite for statistical testing, a special variant of random testing allowing predictions on the operational behavior of the software ([8, 9]).

Markov software usage models aim at representing the (estimated) distribution of possible uses by means of a Markov chain. This probabilistic concept combines the ability of picturing relative complex dynamic use structures with the advantage of still being mathematically tractable. A Markov usage model is based on a directed graph $G = (V, A)$, where

- V is a set of *nodes*, representing usage states (e.g., program invocation, program termination, input/output screens), and
- A is a set of *arcs*, representing state transitions which always correspond to specific operations of the program. An arc from state i to state j can be denoted by the ordered pair (i, j).

Furthermore, each arc (i, j) is labelled by a *transition probability* $p(i, j)$. This value indicates the relative frequency of a transition to state j, given that the current state is state i, during the operational use of the program.

Fig. 1 shows a (very simple) example of a Markov usage model: A program has a main menu, from which the user may select two special functions. The probabilities for the selection of function 1 resp. function 2 are estimated to be 0.6 resp. 0.3. After the execution of the selected function, the program returns to the main menu. In about 10 percent of all cases, the user decides not to make a (new) function call, but to terminate the program.

A Markov usage model of more realistic, but still moderate complexity is shown in Fig. 2. It is models a (small) part of a train schedule program of the Austrian Federal Railways.

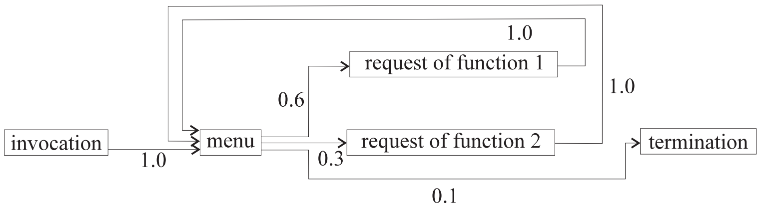

Fig. 1. A simple Markov usage model. The values assigned to the arcs are transition probabilities; the arcs themselves are operations of the program.

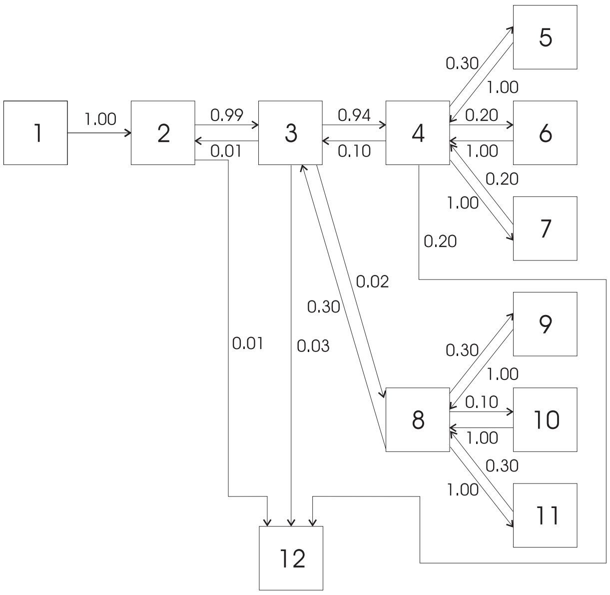

Fig. 2. A Markov usage model with $n = 12$ nodes.

We always assume that the states (nodes) of G are numbered by $1, \ldots, n$. State 1 is the *initial state* corresponding to program invocation, state n is the *final state*, corresponding to program termination. In order to obtain a complete Markov chain, we add an arc with assigned probability one from node n to node n; by this self–loop, n gets an *absorbing state* which cannot be left anymore.

Since program failures cannot be completely excluded, each program has a *failure risk*, defined as the expected loss by program failure during a single program execution in the operational environment (see [1, 6]). A straightforward method for getting an unbiased estimate of the risk consists in statistical testing with an input distribution corresponding to the operational use, and to take the average loss occured during test as a risk estimate. On the base of a Markov usage model, a statistical test case is obtained as a state sequence or *path* $(X_0, \ldots, X_q)$ of the usage chain, beginning at the initial state and ending at the final state, where each transition (X_{k-1}, X_k) is selected according to the operational transition probabilities $p(i, j)$.

Walton, Poore and Trammell [9] have outlined a serious drawback of this method: There can be critical operations (connected with high failure probability and/or high loss in case of failure) that are nevertheless invocated infrequently during the operational use of the program. Then, in a Markov chain model, the corresponding arcs get only small probabilities of being activated, which may exclude these operations from test. Walton et al. suggest an adjustment of the usage distribution to ensure that such operations are tested more frequently, but they admit that this approach limits statistical inferences on the program behavior in its operational environment.

The intention of this paper is to develop a technique of changing the transition probabilities *and* compensating the effect of this change in such a way that

- critical operations are tested with sufficiently high frequency, and
- the risk estimate derived from the test remains unbiased.

To optimize the benefit of our technique, we try to find *that* probability change that allows *the most precise risk estimation.* (A similar approach for the more special case of a program with n alternative program functions has been developed in [2].)

Let us emphasize that the drawback of conventional Markov usage models mentioned in [9] is especially prohibitive in the case of *safety–critical software*, where there are always operations with low use frequency and high criticality. So, without a technique overcoming this drawback, the applicability of Markov usage models to this type of software would be very limited. The technique outlined here provides a satisfactory solution to this problem.

2 Change of transition probabilities

Our method relies on the *importance sampling* technique of Rare Event Simulation, which prescribes that a probility change in a simulation has to be compensated by adjusting suitable weights to the outcomes of the simulation run. For a Markov usage model, one obtains the following compensation for a shift of the transition probabilities from $p(i,j)$ to $t(i,j)$: Let loss (X) denote the loss occuring in a single test execution along path $X = (X_0, \ldots, X_q)$, resp. loss $(X) := 0$ if no failure occurs along X. Then (cf., e.g., [4]),

$$S := \prod_{k \geq 1} \frac{p(X_{k-1}, X_k)}{t(X_{k-1}, X_k)} \cdot \text{loss}\,(X) \tag{1}$$

is an unbiased estimate of the risk. This can be seen as follows. Let P resp. T stand for the operational distribution, resp. the test distribution obtained by the probability change. By E_P and E_T we denote the mathematical expectation under distribution P and T, respectively. Obviously,

$$p(X) := \prod_k p(X_{k-1}, X_k) \quad \text{resp.} \quad t(X) := \prod_k t(X_{k-1}, X_k)$$

is the probability that path X occurs, given that the states are selected according to the probabilities $p(i,j)$ resp. $t(i,j)$. So for the expected value of S under distribution T,

$$E_T(S) = E_T\left(\frac{p(X)}{t(X)} \cdot \text{loss}\,(X)\right) = \sum_x t(x) \cdot \frac{p(x)}{t(x)}\,\text{loss}\,(x)$$

$$= \sum_x p(x)\,\text{loss}\,(x) = E_P(\text{loss}\,(X))$$

holds. The rightmost expression, however, is just the risk. Let us mention that in importance sampling, the weight $p(X)/t(X)$ is usually called the *likelihood ratio*.

Taking the average value of S for all random test cases (selected according to the changed transition probabilities) yields the overall risk estimate.

For example, the transition probabilities in Fig. 1 could be changed to, say,

$$t(\text{menu, req\_function\_1}) := 0.2, \quad t(\text{menu, req\_function\_2}) := 0.8$$

(other probabilities as before). Assume that, selecting states according to these new probabilities, we obtain the path

$$X = (\text{invoc, menu, req\_function\_2, menu, req\_function\_1, menu, term}),$$

and that an execution of the program with inputs driving it along X reveals a program failure of a severity estimated by 10 cost units. Then this loss has to be weighted by

$$\frac{0.3}{0.8} \cdot \frac{0.6}{0.2} = 1.125,$$

such that we derive a risk estimate of $S = 1.125 \cdot 10 = 11.25$ from this test case. Of course, the risk estimate corresponding to a test case is zero, if the test case does not reveal a failure. If N test cases are selected, executed, and evaluated in the way described above, yielding risk estimates $S(1), \ldots, S(N)$, then the overall risk estimate is given by the arithmetic mean $\bar{S}$ of the values $S(1), \ldots, S(N)$.

3 Optimizing the precision of the risk estimate

Now let us turn to the question how to choose the transition probabilities in order to get a risk estimate with maximum precision. As a measure for the unprecision of an (unbiased) estimator we take its *variance*, as customary in software reliability (cf. [7]). The variance $var_T(S)$ of S for test cases selected according to test distribution T would satisfy our requirements on a criterion; however, $var_T(S)$ cannot be determined in advance, since S depends on loss (X), and loss (X) is unknown before the test.

Therefore, a Bayesian approach is applied: For each fixed x, loss (x) is conceived as a random variable, whose distribution reflects the *prior information* of the tester on failure probabilities and amount of loss in case of failure. To be more explicit: It is assumed that the operation corresponding to arc (i, j) has a prior probability $f(i, j)$ of failing, and causes a loss of $l(i, j)$ in case of the failure event. Failures of different arcs (i, j) occur independently from each other. The values $f(i, j)$ and $l(i, j)$ have to be estimated for each arc. Although this model is only an approximation to reality, it already allows the test designer to take account of critical operations, which is not possible in the conventional Markov usage model framework.

Denoting the expectation with respect to the (estimated) prior loss distribution by E_Q, the aim is to minimize $E_Q(var_T(S))$ by an appropriate choice of the test distribution T. (Notice that E_Q acts, for *fixed* path x, on the loss, while E_P and E_T act on the path.)

In the sequel, we assume that the loss occuring at a specific program execution mainly depends on the path x triggered by the input data. Then, for fixed x, loss (x) can be considered (in a first approximation) as independent of the actual input causing the execution of x. Of course, this is a simplification: For example, it may happen that input_1 and input_2 lead to the same path x, but input_1 is processed correctly, while input_2 is not. Nevertheless, a Markov usage model with a sufficiently high granularity (cf. [9]) has the property that inputs triggering the same path are processed in a very similar way, such that they can be considered as "near–to–homogenous" in their failure behavior. (For the notion of homogenous sets of inputs, see [3]). We emphasize that this assumption is not required for the unbiasedness of the estimator (1).

By $A(x)$, the set of transitions (i, j) contained in path x is denoted. According to the model described above, we obtain now

$$\text{loss}\,(x) = \sum_{(i,j)\in A(x)} \text{fail}\,(i, j)\, l(i, j), \tag{2}$$

where

$$\text{fail}\,(i, j) = \begin{cases} 1, & \text{if transition } (i, j) \text{ fails}, \\ 0, & \text{else}, \end{cases}$$

such that $E_Q(\text{fail}\,(i, j)) = f(i, j)$. Since

$$var_T(S) = E_T(S^2) - [E_T(S)]^2 = E_T(S^2) - (\text{risk})^2,$$

minimization of $E_Q(var_T(S))$ with respect to T is equivalent to minimization of $E_Q(E_T(S^2))$. By a short calculation, one finds

$$E_Q(E_T(S^2)) \; = \; E_P\left(\frac{p(X)}{t(X)} \cdot E_Q(\,[\,\mathrm{loss}\,(X)]^2)\right). \qquad (3)$$

Furthermore, because of (2), $E_Q(\,[\,\mathrm{loss}\,(X)]^2)$ is equal to

$$E_Q\left(\sum_{(i,j)} \mathrm{fail}\,(i,j)\,[l(i,j)]^2 \; + \sum_{(i,j)\neq(k,m)} \mathrm{fail}\,(i,j)\,\mathrm{fail}\,(k,m)\,l(i,j)\,l(k,m)\right)$$

$$= \sum_{(i,j)} f(i,j)\,[l(i,j)]^2 + \sum_{(i,j)\neq(k,m)} f(i,j)\,f(k,m)\,l(i,j)\,l(k,m), \qquad (4)$$

all sums being only over arcs contained in $A(X)$. On the usual assumption that prior failure probabilities are small quantities (only programs that are already relatively stable are worthwile to be exposed to tests for risk estimation!), the second sum on the rightmost side of (4) is negligible, compared to the first sum. Omission of this second sum, insertion into (3) and re–insertion of the expressions for $p(X)$ and $t(X)$ leads to the following stochastic optimization problem:

$$\text{Minimize } G(T) := E_P\left(\prod_{k\geq 1}\frac{p(X_{k-1},X_k)}{t(X_{k-1},X_k)} \cdot \sum_{(i,j)\in A(X)} f(i,j)\,(l(i,j))^2\right). \quad (5)$$

Therein, the variables $t(i,j)$ $(1 \leq i,j \leq n)$ to be optimized have to satisfy the constraint of being transition probabilities of a Markov chain with absorbing state n. Additionally, the constraint $t(i,j) = 0$ if and only if $p(i,j) = 0$ must be satisfied, otherwise the likelihood ratio could become infinite.

It can be shown that the function $G(T)$ in (5) is *convex* in the variables $t(i,j)$. This fact allows an efficient solution of (5) by well–known stochastic optimization techniques.

4 Numerical computation of the optimal transition probabilities

For the numerical solution of (5), we use, in our implementation, an optimization algorithm of Frank–Wolfe–type (see [5], section 10.10.3). Basically, we proceed as follows: A sample of R independent paths according to distribution P is drawn, and the minimization of the expected value of the product in (5) is replaced by the minimization of the average value of the product over the paths in the sample. This is done iteratively. We start with the $[n \times n]$–matrix T of the variables $t(i,j) := p(i,j)$ as the initial solution. The solution is improved successively: For each fixed line i of the current matrix T, consider the partial derivatives $D(i,j)$ of $G(T)$ to the variables $t(i,j)$ $(j = 1,\ldots,n)$. Take that $j = j^*$ for which $D(i,j)$ is minimal, and modify the current solution T by augmenting the value of $t(i,j^*)$ by a certain stepsize. The partial derivatives $D(i,j)$ can be computed explicitly. We use the result of this computation in the following, more detailed description of the algorithm:

procedure optimize_test $(P,\, F,\, L)$
$/*\ P = (p(i,j)),\ F = (f(i,j)),\ L = (l(i,j))\ */$
{ **for** $r := 1$ to R
  { draw path X according to P;
    compute $Y(r) := \prod_{k\geq 1} p(X_{k-1}, X_k) \sum_{(i,j)\in A(X)} f(i,j)\,(l(i,j))^2$;
    for each (i,j)
      compute $M(r,i,j) :=$ number of transitions $(X_{k-1}, X_k) = (i,j)$; }
  set $T := P$;
  for $s := 1$ to S
    for $i := 1$ to n
    { **for** each j with $p(i,j) > 0$
        compute $D(i,j) :=$ arithmetic mean over $r = 1,\ldots,R$ of
        $-\,Y(r)\ \prod_{k,m}(t(k,m))^{-M(r,k,m)}\ M(r,i,j)\ /\ t(i,j)$;
      determine that $j = j^*$ for which $D(i,j)$ is minimal;
      for each j with $p(i,j) > 0$
      { $t(i,j) := (1 - 1/(2s))\, t(i,j)$;
        if $(j = j^*)$ $t(i,j) := t(i,j) + 1/(2s)$; } }
  return T; }

By some minor modifications, both the runtime efficency and the solution quality of this algorithm can still be improved.

5 Experimental results

For judging the solution quality of the approximation algorithm optimize_test described in the previous section, one may start with the small example of a Markov usage model in [9], p. 102, since it is still possible to calculate the solutions of (5) *exactly* for this case. The Markov chain in the example has four states (invocation, main_menu, display, and termination). The operational transition probabilities, the failure probabilities and the failure severities (losses in case of failure) are estimated as follows:

$$P = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0.5 & 0.5 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},\quad F = \begin{bmatrix} 0 & 0.1 & 0 & 0 \\ 0 & 0 & 0.4 & 0.1 \\ 0 & 0.1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix},\quad L = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 5 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

Because of the condition $t(i,j) = 0 \Leftrightarrow p(i,j) = 0$, the test transition probabilities are identical to the values $p(i,j)$, except $t(2,3) =: t$ and $t(2,4) = 1 - t$. Explicit solution of (5) yields $t^* = 0.620$ as the optimal value for t.

Our C++ implementation of optimize_test finds for $R = 100$ and $S = 100$ an approximation to the exact value of t^* that is is sufficiently good for

R	S	K	t^*	Runtime (in sec)
100	100	1	0.635	6
1000	100	1	0.617	59
100	100	10	0.621	56

Table 1. Application of optimize_test to the 4–state Markov usage model: results and runtimes on a PC with Intel 80486 DX processor.

practical purposes (see line 1 of Table 1). Of course, increasing the sample size R improves the solution quality (see line 2 of Table 1). Note, however, that storing the entries $M(r, i, j)$ requires space of order $O(R \cdot n^2)$, which can be prohibitive for large Markov usage models. So we have investigated an alternative: Instead of generating a very large sample of paths and storing the quantities $Y(r)$ and $M(r, i, j)$ for all these paths, we generate only a more limited sample, perform the approximation steps of optimize_test, and *repeat* this process K times, such that newly drawn samples can influence the current approximate solution. In order to obtain convergence of the algorithm, the stepsize for the modification of the $t(i, j)$ has to be decreased in the successive iterations $k = 1, \ldots, K$. We have chosen a stepsize of $1/(2sk)$ in the kth iteration. The effect of this procedure is here even more convincing than that of working with a large sample as a whole (see line 3 of Table 1). However, the last observation cannot be generalized: for other values R and S, we have also found cases where augmenting K by a certain factor produced slightly worse results than augmenting R by the same factor.

Another situation where exact solution values can be determined is the case of Markov chains consisting of edge–disjoint paths from node 1 to node n. Here, Theorem 2 in [2] yields an explicit formula for the optimal test probabilities. We have tested some examples of this type and found a good agreement between theoretical and experimental values.

Finally, we present the outcomes of optimize_test for the Markov usage model of Fig. 2. The values $f(i, j)$ and $l(i, j)$ have been estimated as follows:

$F =$

	1	2	3	4	5	6	7	8	9	10	11	12
1	0	l	0	0	0	0	0	0	0	0	0	0
2	0	0	m	0	0	0	0	0	0	0	0	l
3	0	l	0	m	0	0	0	l	0	0	0	l
4	0	0	l	0	m	m	l	0	0	0	0	l
5	0	0	0	l	0	0	0	0	0	0	0	0
6	0	0	0	l	0	0	0	0	0	0	0	0
7	0	0	0	l	0	0	0	0	0	0	0	0
8	0	0	l	0	0	0	0	0	l	h	l	0
9	0	0	0	0	0	0	0	l	0	0	0	0
10	0	0	0	0	0	0	0	l	0	0	0	0
11	0	0	0	0	0	0	0	l	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0

$L =$

	1	2	3	4	5	6	7	8	9	10	11	12
1	0	m	0	0	0	0	0	0	0	0	0	0
2	0	0	m	0	0	0	0	0	0	0	0	l
3	0	m	0	m	0	0	0	m	0	0	0	l
4	0	0	m	0	m	m	m	0	0	0	0	l
5	0	0	0	m	0	0	0	0	0	0	0	0
6	0	0	0	m	0	0	0	0	0	0	0	0
7	0	0	0	m	0	0	0	0	0	0	0	0
8	0	0	m	0	0	0	0	0	l	h	m	0
9	0	0	0	0	0	0	0	m	0	0	0	0
10	0	0	0	0	0	0	0	m	0	0	0	0
11	0	0	0	0	0	0	0	m	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0

Therein, the constants l (low), m (medium) and h (high) have the following values: In F, $l = 0.001$, $m = 0.01$, and $h = 0.1$. In L, $l = 1$, $m = 10$, and $h = 100$. We have assumed that arc (8,10) corresponds to a newly implemented (i.e., error–prone) function with high criticality. Here, explicit solutions of (5) are not available anymore, so we cannot compare the output of optimize_test with the true solution values. For $R = 100$, $S = 100$ and $K = 10$, we obtained, after 196 seconds of computation time, the following output matrix T:

$T =$

	1	2	3	4	5	6	7	8	9	10	11	12
1	.00	1.0	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00
2	.00	.00	.99	.00	.00	.00	.00	.00	.00	.00	.00	.01
3	.00	.01	.00	.83	.00	.00	.00	.11	.00	.00	.00	.05
4	.00	.00	.11	.00	.31	.20	.20	.00	.00	.00	.00	.18
5	.00	.00	.00	1.0	.00	.00	.00	.00	.00	.00	.00	.00
6	.00	.00	.00	1.0	.00	.00	.00	.00	.00	.00	.00	.00
7	.00	.00	.00	1.0	.00	.00	.00	.00	.00	.00	.00	.00
8	.00	.00	.29	.00	.00	.00	.00	.00	.15	.28	.28	.00
9	.00	.00	.00	.00	.00	.00	.00	1.0	.00	.00	.00	.00
10	.00	.00	.00	.00	.00	.00	.00	1.0	.00	.00	.00	.00
11	.00	.00	.00	.00	.00	.00	.00	1.0	.00	.00	.00	.00
12	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	1.0

Comparing the values $t(i,j)$ with the values $p(i,j)$ unveils that the intended effect of favoring the test of critical operations has been achieved: While for, say, 100 test cases in total, the generation of test sequences according to P gives the critical function (8,10) only a small probability of being tested (note that already (3,8) has a small probability!), transition (8,10) obtains a fair chance in a test sequence generation according to T.

Since in a careful implementation, the runtime of optimize_test grows only linearly in the size (number of entries) of the input matrices, even large Markov usage models can be treated in reasonable time. The bottleneck is the storage requirement of order $O(Rn^2)$: For large n, R has to be chosen relatively small, which reduces the solution quality. This can partially be compensated by increasing the value K. In any case, the number $R \cdot K$ of generated test sequences should be *essentially* larger than the number of test cases to be executed. Otherwise, it may happen that infrequently used, but error–prone paths do not occur in the sample and get therefore no opportunity to shift the numbers $t(i, j)$ to more favorable values.

An approach for the solution of the last–mentioned problem is to re–formulate (5) in such a way that instead of E_P, the expected value $E_{\tilde{P}}$ with respect to another distribution $\tilde{P}$ giving *all* possible state transitions fair chances is used. For the sake of brevity, we omit the details of this more sophisticated technique.

References

[1] Ehrenberger, W. D., "Combining probabilistic and deterministic verification efforts", *Proc. SAFECOMP '92* (ed. H. Frey), 299 – 304, Oxford: Pergamon Press (1992).

[2] Gutjahr, W. J., "Optimal test distributions for software failure cost estimation", *IEEE Trans. Software Eng.*, vol. 21 (3), 219 – 228 (1995).

[3] Hamlet, R., Taylor, R., "Partition testing does not inspire confidence", in: *Proc. 2nd Workshop an Software Testing, Verification, and Analysis*, July 1988, 206 – 215 (1988).

[4] Heidelberger, P., "Fast simulation of rare events in queuing and reliability models", *ACM Trans. on Modeling and Computer Simulation*, vol 5 (1), 43 – 85 (1995).

[5] Murty, K. G., *Linear Complementary, Linear and Nonlinear Programming*, Heldermann (1988).

[6] Sherer, S. A., *Software Failure Risk*, New York, London: Plenum Press (1992).

[7] Thayer, Th. A., Lipow, M., and Nelson, E. C., *Software Reliability*, Amsterdam: North–Holland (1978).

[8] Thevenod–Fosse, P., "From random testing of hardware to statistical testing of software", *Proc. IEEE COMPEURO '91*, 200 – 207 (1991).

[9] Walton, G. H., Poore, J. H., Trammell, J., "Statistical testing of software based on a usage model", *Software – Practice and Experience*, vol. 25 (1), 97 – 108 (1995).

[10] Whittaker, J. A., "Markov chain techniques for software testing and reliability analysis", Ph. D. dissertation, University of Tennessee (1992).

[11] Whittaker, J. A., Poore, J. H., "Markov analysis of software specifications", *ACM Trans. Software Eng. and Method.*, vol. 2 (1), 93 – 106 (1993).

[12] Woit, D. M., "Operational profile specification, test case generation, and reliability estimation for modules", Thesis, Queen's University, Kingston, Ontario, Canada (1994).