Control and Cybernetics

vol. **34** (2005) No. 2

A discrete-time software reliability-growth model and its application for predicting the number of errors encountered during program testing

by

K. Worwa

Faculty of Cybernetics, Military Technical University
00-908 Warszawa, Kaliskiego 2, Poland
e-mail: kworwa@isi.wat.waw.pl

Abstract: An approach to evaluate the number of errors encountered during the program testing process is proposed in the paper. Considerations are based on some program reliability-growth model, constructed for an assumed scheme of the program testing process. In this model the program under testing is characterized by means of the so-called characteristic matrix and the program testing process is determined by means of so-called testing strategy. The formula for determining the mean value of the predicted number of errors encountered during the program testing is obtained. This formula can be used if the characteristic matrix and the testing strategy are known. Formulae for evaluating this value when the program characteristic matrix is not known are also proposed.

Keywords: software testing, software reliability, software reliability-growth model.

1. Introduction

An important step in the development of many software systems is some form of reliability assessment. In some cases, it is essential to have confidence that software achieves a required level of reliability before it is put into service. With the aim of achieving this goal software reliability-growth models are widely used to describe the process of software reliability growth as a direct consequence of detecting and removing software failures during the software testing phase of software development. The testing of a newly developed program, prior to its practical use, is a commonly followed practice. The program testing process involves the execution of the program with many sets of input data with the intention of finding errors. Testing is done to lower the chances of in-service failures which are defined as an unacceptable departure from program operation. A long period of testing results in the increased chances of detecting program errors and decreased chances of in-service failures, but it also results in the increased cost of the program testing process.

It is known that testing is the most significant money consuming stage of the program development. The cost of program testing process can account for even more than 50% of the total cost of the program development, especially for complex program systems (see e.g. Thayer, Lipov and Nelson, 1978, or Kit, 1995). Considering the essential impact of the testing cost on the whole program development cost, the testing process ought to be prudently planned and organized. Decisions relative to the testing process organization should be made on the basis of results of the testing efficiency analysis. In order to make such analysis easier it may be convenient to evaluate the number of program errors that could be encountered during the program testing process. The knowledge of this evaluation makes it possible to assess the duration and the cost of the program testing process, e.g. by means of formal, mathematical expressions. Such estimations can be very useful in practice, e.g. for comparing the effectiveness of different ways of program testing process organizations (i.e. in order to find an optimal organization).

The number of program errors encountered during the testing process depends on many factors, such as the testing process organization and technology (that define the manner of the testing process realization), duration of the testing, the testers' qualifications and professional experience and the reliability level of the program at the beginning of the testing process. The duration of the program testing process can be determined by the predicted time spent on testing activities or by the predicted cardinality of the set of input data that should be used for the testing.

For the purpose of the rational prediction of the program testing process duration the software reliability-growth models are recommended in this paper. These models can provide a first program reliability-estimate and support project management in planning further development processes. Therefore many software reliability-growth models have been proposed in the literature, see e.g. Basu and Ebrahimi (2003), Chen and Mathur and Rego (1995), Csenki (1990), Gaudoin (1999), Hayakawa and Telfar (2000), Jeske and Pham (2001), Sawada and Sandoh (2000), Tokuno and Yamada (2000), Whittaker, Rekab and Thomason (2000), Yamada and Fujiwara (2001), Zhang and Pham (2000), Musa and Iannino and Okumoto (1987), Cai (2000), Chen and Yu (1994, 1996), Yang and Chao (1995). The number of models that have actually been employed to assist the software development projects is, however, much smaller. The reason for this is that none of these models gives sufficiently accurate estimates of reliability. This seems to be chiefly due to the fact that the authors of the various models have paid little or no attention to the manner in which software is tested.

Software reliability modelling has become one of the most important aspects in software reliability engineering since models of Jelinski-Moranda (1972) and Schooman (1972) appeared. Various methodologies have been adopted to model software reliability behaviour. The most of existing work on software reliability modelling is focused on continuous-time base, which assumes that software reliability behaviour can be measured in terms of time. It may be a calendar time, a clock time or a CPU execution time, see e.g. Basu and Ebrahimi (2003), Chen, Mathur and Rego (1995), Csenki (1990), Gaudoin (1999), Hayakawa and Telfar (2000), Jeske and Pham (2001), Sawada and Sandoh (2000), Tokuno and Yamada (2000), Whittaker, Rekab and Thomason (2000), Yamada and Fujiwara (2001), Zhang and Pham (2000), Musa, Iannino and Okumoto (1987). Although this assumption is appropriate for a wide scope of software systems, there are many systems, which essentially diverge from this assumption. For example, reliability behaviour of a reservation software system should be measured in terms of how many reservations are successful, rather than how long the software operates without any failure. Similarly, reliability behaviour of a bank transaction processing software system should be assessed in terms of how many transactions are successful, etc. Obviously, for these systems, the time base of reliability measurement is essentially discrete rather than continuous. Models that are based on a discrete-time approach are called input-domain or run-domain models, see e.g. Cai (2000), Chen and Yu (1994, 1996), Worwa (1995a, b), Yang and Chao (1995). They usually express reliability as the probability that an execution of the software is successful. Cai (2000) has proposed the terms "a run" and "run reliability" and a conceptual framework of software run reliability modelling. A run is a minimum execution unit of software and run reliability means the probability that software successfully perform a run. The concrete sense of a run is subject to application context. For example, a run can correspond to execution of a test case, of a program path, etc.

This paper attempts to describe a new discrete-time software reliability-growth model for some scheme of program testing and to determine a formula to evaluate the predicted number of program errors encountered during the testing process.

2. Description of the program testing scheme

The organization of a program testing process depends on the program testing strategy, which was selected for the testing. In particular, this strategy defines the way of the testing process realization and the set of program input data which is used for the testing.

It is assumed that the program testing process consists of a number of organizational units of the program testing phase that are called the testing stages. Every stage of the program testing process consists of the two following steps of testing:

- testing of the program under consideration with a prepared set of program input data (tests),
- comparison of the results with the expected outputs for the data used and removing all errors encountered during testing.

It is noteworthy that all program faults discovered during the first step of some testing stage are only registered and removed after this step is finished. Such organization of the testing stage means that testing and debugging are performed in different steps (not simultaneously) and consequently some program fault can be observed more than once in the testing stage. Yang and Chao (1995) have underlined that the above definition of the testing scheme is used in the majority of mathematical models of software testing.

Let S mean the program testing strategy, which is defined as follows

$$S = (K, (L_1, L_2, \dots, L_k, \dots, L_K)), \tag{1}$$

where:
K — the number of stages of the program testing process,
L_k — the number of program input data used in the k-th program testing stage, $L_k > 0$, $k = \overline{1, K}$.
Execution of the program under the testing process with one input data set (test case) will be called a run in this paper. The run can be successful, if program execution did not lead to encountering of any program errors or not successful, if program execution was incorrect, i.e. some errors were encountered.

Let **S** denote the set of all strategies that have the form (1). The strategy (1) defines a program testing scheme. In accordance with this scheme, the process of removing program errors which were encountered during the k-th program testing stage can be started after the execution of the program on all L_k tests is finished.
According to the assumed the testing scheme a situation that a number of different tests of all L_k tests executed during the k-th stage encounter the same error in the program under the testing is possible. So, according to the note mentioned above, a situation that several runs will lead to encountering the same program error is possible.

Let p_{nm} define the probability of the event that n errors will be encountered during a single stage of the program testing if there are m tests that have incorrect execution in that stage. If we assume that every execution of the program with a single test can lead to encountering at most one program error, we will have

$$0 \le p_{nm} \le 1 \quad \text{if } n \le m, \quad n \ge 0, \tag{2}$$

where in particular

$$\begin{aligned} &p_{00} = p_{11} = 1 \\ &p_{nm} = 0 \quad \text{if } n > m \end{aligned} \tag{3}$$

and

$$\sum_{n=0}^{\infty} p_{nm} = 1, \quad m \in \{0, 1, 2, \dots\}. \tag{4}$$

Probabilities p_{nm}, $n \in \{0, 1, 2, \ldots, m\}$, $m \in \{0, 1, 2, \ldots, \}$, form an infinite matrix $P = [p_{nm}]$ as follows

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & \ldots \\ 0 & 1 & p_{12} & p_{13} & p_{14} & \ldots \\ 0 & 0 & p_{22} & p_{23} & p_{24} & \ldots \\ 0 & 0 & 0 & p_{33} & p_{34} & \ldots \\ 0 & 0 & 0 & 0 & p_{44} & \ldots \\ & & \ldots & & & \end{bmatrix}, \tag{5}$$

where values p_{nm} are defined by (2-4).
The matrix P contains the values 0 below the main diagonal because — in accordance with earlier assumption that every execution of the program with a single test can lead to encountering of at most one program error — it is not possible to encounter more different errors than the number of incorrect runs.
The values of probabilities p_{nm} for every program testing stage depend only on the number of tests which are used during that stage.
The values of probabilities p_{nm} that form the matrix P depend on a logical structure of the program. In particular, an important impact on these probabilities comes from:

— number of paths that have been identified in the program,
— degree of overlapping among individual paths, that can be measured by the number of program instructions that belong to two or more paths,
— length of individual paths, measured by the number of program instructions that are executed in case of path activation.

The matrix P will be called the characteristic matrix of the program under testing.
The following example illustrates the construction of the characteristic matrix P.
Let the characteristic matrix of the program under testing be of the form

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & \ldots \\ 0 & 1 & 0.6 & 0.5 & 0.2 & \ldots \\ 0 & 0 & 0.4 & 0.3 & 0.4 & \ldots \\ 0 & 0 & 0 & 0.2 & 0.3 & \ldots \\ 0 & 0 & 0 & 0 & 0.1 & \ldots \\ & & \ldots & & & \end{bmatrix}.$$

Then, the probabilities p_{nm} that form the matrix P have the following interpretation:

<u>for the first row of the matrix:</u>

- $p_{00} = 1$ — if there is no test that has an incorrect execution during the testing stage ($m = 0$), the probability that no program error will be encountered ($n = 0$) is equal 1;

- $p_{01} = 0$, $p_{02} = 0$, $\ldots$ — if more than one test has an incorrect execution ($m = 1, 2, \ldots$), the probability that no program error will be encountered ($n = 0$) is equal 0 (as far as the assumption that every execution of the program with a single test can lead to encountering of at most one program error is concerned);

<u>for the second row of the matrix:</u>

- $p_{10} = 0$ — if no test has an incorrect execution during the testing stage ($m = 0$), the probability that one program error will be encountered ($n = 1$) is equal 0;
- $p_{11} = 1$ — if one test has an incorrect execution ($m = 1$), the probability that one program error will be encountered ($n = 1$) is equal 1;
- $p_{12} = 0.6$ — if two tests have incorrect executions ($m = 2$), the probability that one program error will be encountered ($n = 1$) is equal 0.6;
- $p_{13} = 0.5$ — if three tests have incorrect executions ($m = 3$), the probability that one program error will be encountered ($n = 1$) is equal 0.5;
- $p_{14} = 0.2$ — if four tests have incorrect executions ($m = 4$), the probability that one program error will be encountered ($n = 1$) is equal 0.2;

<u>for the third row of the matrix:</u>

- $p_{20} = 0$, $p_{21} = 0$ — if not more than one test has an incorrect execution during the testing stage ($m = 0, 1$), the probability that two program errors will be encountered ($n = 2$) is equal 0;
- $p_{22} = 0.4$ — if two tests have incorrect executions ($m = 2$), the probability that two program errors will be encountered ($n = 2$) is equal 0.4;
- $p_{23} = 0.3$ — if three tests have incorrect executions ($m = 3$), the probability that two program errors will be encountered ($n = 2$) is equal 0.3
- $p_{24} = 0.4$ — if four tests have incorrect executions ($m = 4$), the probability that two program errors will be encountered ($n = 2$) is equal 0.4;

<u>for the fourth row of the matrix:</u>

- $p_{30} = 0$, $p_{31} = 0$, $p_{32} = 0$ — if not more than two tests have incorrect executions ($m = 0, 1, 2$), the probability that three program errors will be encountered ($n = 3$) is equal 0;
- $p_{33} = 0.2$ — if three tests have incorrect executions ($m = 3$), the probability that three program errors will be encountered ($n = 3$) is equal 0.2;
- $p_{34} = 0.3$ — if four tests have incorrect executions ($m = 4$), the probability that three program errors will be encountered ($n = 3$) is equal 0.3;

<u>for the fifth row of the matrix:</u>

- $p_{40} = 0$, $p_{41} = 0$, $p_{42} = 0$, $p_{43} = 0$ — if not more than three tests have incorrect executions ($m = 0, 1, 2, 3$), the probability that four program errors will be encountered ($n = 4$) is equal 0;
- $p_{44} = 0.1$ — if four tests have incorrect executions ($m = 4$), the probability that four program errors will be encountered ($n = 4$) is equal 0.1.

3. Program reliability coefficient

Let $M_k(S,P)$ denote the number of runs, which lead to incorrect execution of the program under testing, i.e. to encounter program errors, during the k-th stage of program testing process according to the testing strategy S and the characteristic matrix P.

Let $N_k(S,P)$ denote the total number of errors encountered during the k-th stage of program testing process in accordance with the testing strategy S and the characteristic matrix P.

While planning the program testing process it is reasonable to treat the values $M_k(S,P)$ and $N_k(S,P)$, $k=\overline{1,K}$, as random variables, $N_k(S,P) \leq$ $\leq M_k(S,P)$, $k=\overline{1,K}$. Joint distribution of the random variables $N_k(S,P)$, $M_k(S,P)$ can be determined as follows:

$$\begin{aligned} &Pr\{N_k(S,P)=n_k, M_k(S,P)=m_k\} = \\ &= Pr\{N_k(S,P)=n_k/M_k(S,P)=m_k\}Pr\{M_k(S,P)=m_k\}. \end{aligned} \tag{6}$$

Probability distribution of the random variable $N_k(S,P)$ can be determined as a marginal distribution in a distribution of two-dimensional random variable $(N_k(S,P), M_k(S,P))$:

$$\begin{aligned} &Pr\{N_k(S,P)=n_k\} = \\ &= \sum_{m_k=0}^{L_k} Pr\{N_k(S,P)=n_k/M_k(S,P)=m_k\}Pr\{M_k(S,P)=m_k\}. \end{aligned} \tag{7}$$

Let $N(S,P)$ denote the total number of errors encountered during the process of the program testing in accordance with the testing strategy S and the characteristic matrix P. The value $N(S,P)$ is a random variable and can be determined as follows:

$$N(S,P) = \sum_{k=1}^{K} N_k(S,P). \tag{8}$$

Taking into account the assumed testing scheme the probability distribution of the random variable $N(S,P)$ has the form:

$$\begin{aligned} &Pr\{N(S,P)=n\} = \\ &= \sum_{n_1+n_2+\ldots+n_K=n} Pr\{N_1(S,P)=n_1, N_2(S,P)=n_2, \ldots, N_K(S,P)=n_K\}, \\ &\qquad\qquad n=0,1,2,\ldots \end{aligned} \tag{9}$$

where probabilities $Pr\{N_1(S,P)=n_1, N_2(S,P)=n_2, \ldots, N_K(S,P)=n_K\}$ determine the joint distribution of the K-dimensional random variable $(N_1(S,P)$, $N_2(S,P), \ldots, N_K(S,P))$.

Probability distribution $Pr\{N_1(S,P) = n_1, N_2(S,P) = n_2, \ldots, N_K(S,P) = n_K\}$ can be determined as follows (see Worwa, 1995a):

$$\begin{aligned} &Pr\{N_1(S,P) = n_1, N_2(S,P) = n_2, \ldots, N_K(S,P) = n_K\} = \\ &= \prod_{k=1}^{K} \sum_{m_k=0}^{L_k} Pr\{N_k(S,P) = n_k / M_k(S,P) = m_k\} \cdot \\ &\cdot Pr\{M_k(S,P) = m_k / N_i(S,P) = n_i,\ i = \overline{1, k-1}\} \end{aligned} \tag{10}$$

where

$$\begin{aligned} &Pr\{M_k(S,P) = m_k / N_i(S,P) = n_i,\ i = \overline{1, k-1}\} = \\ &= Pr\{M_k(S,P) = m_k / N_1(S,P) = n_1, N_2(S,P) = n_2, \ldots, N_{k-1}(S,P) = n_{k-1})\} \end{aligned} \tag{11}$$

and $N_0(S,P) = 0$.
According to earlier notations, we have

$$p_{nm} = Pr\{N_k(S,P) = n\big|_{M_k(S,P)=m}\}, \quad k = \overline{1, K}, \tag{12}$$

and so formula (10) takes the form:

$$\begin{aligned} &Pr\{N_1(S,P) = n_1,\ N_2(S,P) = n_2, \ldots, N_K(S,P) = n_K\} = \\ &= \prod_{k=1}^{K} \sum_{m_k=0}^{L_k} p_{n_k m_k} Pr\{M_k(S,P) = m_k / N_i(S,P) = n_i,\ i = \overline{1, k-1}\}. \end{aligned} \tag{13}$$

By the substitution of this equation into (9), we get:

$$\begin{aligned} &Pr\{N(S,P) = n\} = \sum_{n_1+n_2+\ldots+n_K=n} Pr\{N_1(S,P) = n_1, \\ &N_2(S,P) = n_2, \ldots, N_K(S,P) = n_K\} = \\ &= \sum_{n_1+n_2+\ldots+n_K=n} \prod_{k=1}^{K} \sum_{m_k=0}^{L_k} p_{n_k m_k} \cdot Pr\{M_k(S,P) = m_k / N_i(S,P) = n_i,\ i = \overline{1, k-1}\}. \end{aligned} \tag{14}$$

As a result of the program testing process execution, in accordance with the assumed testing strategy S, $N(S,P)$ program errors will be encountered. Successful removal of these errors will increase the level of program reliability. The number of encountered errors $N(S,P)$ depends on both the assumed testing strategy and the initial level of program reliability, i.e. reliability of the program at the beginning of the program testing process.

It is very well known that in spite of the successful testing phase realization the program can still contain some errors. Bearing this fact in mind it is rational to describe a result $X(S,P)$ of one program execution, after finishing the testing process according to strategy S, as follows:

$$X(S,P) = \begin{cases} 1 & \text{if an execution of the program with the characteristic} \\ & \text{matrix } P \text{ that was tested in accordance with the strategy } S, \\ & \text{is correct;} \\ 0 & \text{otherwise.} \end{cases}$$

When planning the phase of the program testing process it is rational to treat the quantities $N(S,P)$ and $X(S,P)$ as dependent random variables, because the result $X(S,P)$ of program execution for some input data set, after finishing the testing according to strategy S, depends on the number of errors $N(S,P)$ encountered during this process.

Let $r(S,P)$ denote the probability of the correct execution of the program with the characteristic matrix P, after finishing the testing process according to the strategy S, i.e.

$$r(S,P) = Pr\{X(S,P) = 1\}. \tag{15}$$

The probability $r(S,P)$ will be treated as a program reliability measure in the paper.
We can write:

$$r(S,P) = \sum_{n=0}^{L(S)} r(S,P)|_{N(S,P)=n} \cdot Pr\{N(S,P) = n\}, \tag{16}$$

where:
$r(S,P)|_{N(S,P)=n}$ — conditional probability of a correct execution of the program with the characteristic matrix P if the testing process, according to the strategy S, led to encountering of $N(S,P) = n$ program errors;
$L(S)$ — the total number of runs that are performed during the testing of the program according to the strategy S, i.e.

$$L(S) = \sum_{k=1}^{K} L_k. \tag{17}$$

By substitution of expression (14) into (16), we get

$$r(S,P) = \sum_{n=0}^{L(S)} r(S,P)|_{N(S,P)=n} \cdot \tag{18}$$

$$\cdot \sum_{n_1+n_2+\ldots+n_K=n} \prod_{k=1}^{K} \sum_{m_k=0}^{L_k} p_{n_k m_k} Pr\{M_k(S,P)=m_k/N_i(S,P)=n_i,\ i=\overline{1,k-1}\}.$$

If a certain number of errors is encountered in the program and they are successfully corrected, the program reliability level will increase. We will describe it as follows:

$$r(S,P)|_{N(S,P)=n} = r + \Delta r(P,n), \tag{19}$$

where:
r — an initial value of the program reliability coefficient, i.e. the value of the program reliability coefficient at the beginning of the program testing process,
$\Delta r(P,n)$ — an increase of the reliability coefficient value of the program with the characteristic matrix P, as a result of encountering and removing n errors.

On the basis of facts described in literature, see e.g. Musa, Iannino and Okumoto (1987), Thayer, Lipov and Nelson (1978), Trachtenberg (1990), it is assumed that the increase $\Delta r(P,n)$ is of the form:

$$\Delta r(P,n) = (1-r)(1-e^{-\alpha n}), \tag{20}$$

where α is a parameter that characterizes both the internal structure of the program under testing and the impact of one error‘s removal on the increase of the program reliability.
It should be stated that evaluation of values of both parameters r and α relies in practice on the testing history of the software.
Thus, by substituting the last equation into (19), we get:

$$r(S,P)|_{N(S,P)=n} = 1-(1-r)e^{-\alpha n}. \tag{21}$$

It is easy to notice that the assumed testing scheme, such that the probability of successful performance of one run remains constant during the testing stage, leads to the so-called Bernoulli scheme. Consequently, conditional probabilities from (18) can be determined by means of the binominal distribution as follows:

$$\begin{aligned} &Pr\{M_k(S,P)=m_k/N_i(S,P)=n_i,\ i=\overline{1,k-1}\} = \\ &= \binom{L_K}{m_k}\left[e^{-\alpha\sum\limits_{i=1}^{k-1}n_i}(1-r)\right]^{m_k}\left[1-e^{-\alpha\sum\limits_{i=1}^{k-1}n_i}(1-r)\right]^{L_k-m_k}, \\ &m_k \in \{0,1,2,\ldots,L_k\}, \quad k=\overline{1,K}. \end{aligned} \tag{22}$$

By substituting (21) and (22) into (18), we get the following expression for the program reliability coefficient $r(S,P)$, after finishing the testing process according to the strategy S:

$$r(S,P) = 1-(1-r)A(S,P), \tag{23}$$

where

$$\begin{aligned} &A(S,P) = \\ &= \sum_{n_1=0}^{L_1}\sum_{n_2=0}^{L_2}\cdots\sum_{n_K=0}^{L_K} e^{-\alpha\sum\limits_{k=1}^{K}n_k}\prod_{k=1}^{K}\sum_{m_k=0}^{L_k} p_{n_k m_k} A_{m_k}\left(\sum_{i=1}^{k-1}n_i, L_k\right) \end{aligned} \tag{24}$$

and

$$
\begin{aligned}
&A_{m_k}\left(\sum_{i=1}^{k-1} n_i, L_k\right) = \\
&= \binom{L_k}{m_k}\left[e^{-\alpha\sum\limits_{i=1}^{k-1} n_i}(1-r)\right]^{m_k}\left[1-e^{-\alpha\sum\limits_{i=1}^{k-1} n_i}(1-r)\right]^{L_k-m_k}, \qquad (25)\\
&m_k \in \{0,1,2,\dots,L_k\}, \quad k=\overline{1,K}.
\end{aligned}
$$

The quantity $A_{m_k}\left(\sum\limits_{i=1}^{k-1} n_i, L_k\right)$, $m_k \in \{0,1,2,\dots,L_k\}$, $n_i \in \{0,1,2,\dots,L_k\}$, $k \in \{1,2,\dots,K\}$, means the probability that m_k runs of all L_k runs performed during the k-th testing stage will be incorrect, i.e. will lead to encountering of errors on condition that in the previous $k-1$ testing stages $\sum\limits_{i=1}^{k-1} n_i$ errors were encountered and removed.
It is easy to check that

$$
\sum_{m_k=0}^{L_k} A_{m_k}\left(\sum_{i=1}^{k-1} n_i, L_k\right) = [1-e^{-\alpha\sum\limits_{i=1}^{k-1} n_i}(1-r)+e^{-\alpha\sum\limits_{i=1}^{k-1} n_i}(1-r)]^{L_k} = 1
$$

and

$$
\begin{aligned}
&\sum_{n_k=0}^{L_k}\sum_{m_k=0}^{L_k} p_{n_k m_k} A_{m_k}\left(\sum_{i=1}^{k-1} n_i, L_k\right) = \sum_{m_k=0}^{L_k} A_{m_k}\left(\sum_{i=1}^{k-1} n_i, L_k\right)\sum_{n_k=0}^{L_k} p_{n_k m_k} = \\
&= \sum_{m_k=0}^{L_k} A_{m_k}\left(\sum_{i=1}^{k-1} n_i, L_k\right) = 1.
\end{aligned}
$$

The increase of the program reliability coefficient $\Delta r(S,P) = r(S,P) - r$, as a result of testing phase realization according to the strategy S, is of the form:

$$
\Delta r(S,P) = (1-r)(1-A(S,P)), \qquad (26)
$$

where the quantity $A(S,P)$ is determined by (24).

Let $S', S'' \in \mathbf{S}$ denote the following program testing strategies:

$$
S' = (L, \underbrace{(1,1,\dots,1)}_{L \text{ times}}), \qquad (27)
$$

$$
S'' = (1, L). \qquad (28)
$$

It could be said that the above strategies S' and S'' are the extreme forms of the testing strategy (1). In particular, if the testing strategy has the form S, then it means that in every testing stage the program under testing process is executed only once. Therefore, if the program execution encounters any fault it will be

removed immediately, i.e. before the next testing stage will be started. For the strategy S' it becomes impossible to observe the repeated appearances of faults in any testing stage. Thus, this strategy is very profitable from the point of view of potential increase of program reliability, but it is rather inefficient from the point of view of the economic aspect of the program testing process.
If the testing strategy has the form S'', it means that the program testing phase consists of only one testing stage containing all tests. Obviously, this strategy is very attractive because of the low cost of the program testing process, but probably it would not guarantee the sufficient increase of the program reliability.
If the strategies S' or S'' are used the formulae for the program reliability coefficient $r(S,P)$ will take the simplified forms:

$$\begin{aligned}
&r(S',P) = r(S') = \\
&= 1-(1-r)\sum_{n_1=0}^{1} e^{-\alpha n_1}(1-r)^{n_1} r^{1-n_1}\cdot \\
&\cdot \sum_{n_2=0}^{1} e^{-\alpha n_2}[e^{-\alpha n_1}(1-r)]^{n_2}[1-e^{-\alpha n_1}(1-r)]^{1-n_2}\ldots \\
&\ldots \sum_{n_K=0}^{1} e^{-\alpha n_K}\left[e^{-\alpha\sum_{m=1}^{K-1} n_m}(1-r)\right]^{n_K}\left[1-e^{-\alpha\sum_{m=1}^{K-1} n_m}(1-r)\right]^{1-n_K}
\end{aligned} \tag{29}$$

and

$$r(S'',P) = \sum_{n=0}^{L} e^{-\alpha n}\sum_{m=0}^{L} p_{nm}\binom{L}{m}(1-r)^m r^{L-m}. \tag{30}$$

The main advantage of using the strategy S' consists in avoiding the possibility of encountering the same program error by different tests during individual testing stages. On the one hand, it makes possible to increase the effectiveness of the testing process that could be measured, for example, by a total number of encountered faults in relation to the total number of tests, which were used in the testing process. On the other hand, the main disadvantage of this strategy is connected with its economic ineffectiveness because of both high cost and long duration of the testing process.
Practical application of the strategy S'' can be connected with the effect of recapturing the same program errors by different tests during individual testing stages. Obviously, this effect can significantly decrease the effectiveness of the testing process. In particular, if the program under the testing process has an incorrect instruction at the beginning of its source code then it is possible that all tests used in some testing stage will encounter the same program error, related to this incorrect instruction. Because of mentioned reasons, every practical testing strategy has character of some compromise between the strategies S' and S''.

Let $\mathbf{P}$ denote the set of all matrices that have the form P, i.e:

$$\mathbf{P} = \{P = [p_{nm}\}, \; n, m \in \{0, 1, 2, \ldots\} : \quad \text{probabilities } p_{nm} \text{ meet the constraints defined by (2)-(4)},$$

where every individual program is characterized only by one characteristic matrix $P \in \mathbf{P}$. Let P^*, P^{**} denote characteristic matrices of the program under the testing that have form:

$$P^* = \begin{bmatrix} 1 & 0 & 0 & 0 & \ldots \\ 0 & 1 & 0 & 0 & \ldots \\ 0 & 0 & 1 & 0 & \ldots \\ 0 & 0 & 0 & 1 & \ldots \\ & & \ldots & & \end{bmatrix}, \qquad P^{**} = \begin{bmatrix} 1 & 0 & 0 & 0 & \ldots \\ 0 & 1 & 1 & 1 & \ldots \\ 0 & 0 & 0 & 0 & \ldots \\ 0 & 0 & 0 & 0 & \ldots \\ & & \ldots & & \end{bmatrix}. \tag{31}$$

It can be proved (see Worwa, 2000) that if P^*, $P^{**} \in \mathbf{P}$ are the matrices of the form (31) then

$$\Delta r(S, P^*) = \max_{P \in \mathbf{P}} \Delta r(S, P) \tag{32}$$

and

$$\Delta r(S, P^{**}) = \min_{P \in \mathbf{P}} \Delta r(S, P). \tag{33}$$

The expressions (32) and (33) make it possible to obtain the following evaluation of the program reliability coefficient value for any program testing strategy $S \in \mathbf{S}$:

$$r(S, P^{**}) \le r(S, P) \le r(S, P^*). \tag{34}$$

The double inequality (34) is a direct conclusion from (32) and (33). This inequality allows for a two-sided rough estimate of the program reliability coefficient value after the finishing the program testing process, according to the testing strategy S. This estimate is better, i.e. more precise, than $0 \le r(S, P) \le 1$, and may be useful in a situation when the probabilities p_{nm}, that define the program characteristic matrix P, can not be determined in practice.

4. Evaluating the mean value of the number of predicted errors encountered during the program testing process

By substituting (22) into (14), we get the following expression for probability distribution of the total number of errors encountered during the program testing process, in accordance with the testing strategy S and the characteristic matrix P:

$$\begin{aligned} &Pr\{N(S,P)=n\} = \\ &= \sum_{n_1+n_2+\ldots+n_K=n} Pr\{N_1(S,P)=n_1, N_2(S,P)=n_2,\ldots,N_K(S,P)=n_K\} = \\ &= \sum_{n_1+n_2+\ldots+n_K=n} \prod_{l=1}^{K} \sum_{m_l=0}^{L_l} p_{n_l m_l} A_{m_l}\left(\sum_{i=1}^{l-1} n_i, L_l\right), \quad n=\overline{0,L(S)} \end{aligned} \tag{35}$$

where the quantities $A_{m_l}\left(\sum\limits_{i=1}^{l-1} n_i, L_l\right)$ and $L(S)$ are determined by (25) and (17), respectively.
The knowledge of the probability distribution function (35) makes it possible to obtain a formula for the mean value of the number of errors encountered during the program testing process according to the strategy S. We have

$$E[N(S,P)] = \sum_{n=0}^{L(S)} nPr\{N(S,P)=n\}, \tag{36}$$

and then, according to (35):

$$\begin{aligned} &E[N(S,P)] = \\ &= \sum_{n=0}^{L(S)} n \sum_{n_1+n_2+\ldots+n_K=n} \prod_{l=1}^{K} \sum_{m_l=0}^{L_l} p_{n_l m_l} A_{m_l}\left(\sum_{i=1}^{l-1} n_i, L_l\right) = \\ &= \sum_{k=1}^{K} \sum_{n_1=0}^{L_1} \sum_{n_2=0}^{L_2} \ldots \sum_{n_K=0}^{L_K} n_k \prod_{l=1}^{K} \sum_{m_l=0}^{n_l} p_{n_l m_l} A_{m_l}\left(\sum_{i=1}^{l-1} n_i, L_l\right). \end{aligned} \tag{37}$$

The formula (37) will be simplified if the program characteristic matrix P is of the form (31), i.e.:

$$\begin{aligned} &E[N(S,P^*] = \\ &= (1-r)\sum_{k=1}^{K} L_k \sum_{n_1=0}^{L_1} \sum_{n_2=0}^{L_2} \ldots \sum_{n_{k-1}=0}^{L_{k-1}} e^{-\alpha \sum\limits_{i=1}^{k-1} n_i} \prod_{l=1}^{k-1} A^*_{n_l}\left(\sum_{i=1}^{l-1} n_i, L_l\right), \end{aligned} \tag{38}$$

and

$$\begin{aligned} &E[N(S,P^{**}] = \sum_{k=1}^{K} \sum_{n_1=0}^{1} A^{**}_{n_1}(0,L_1)\ldots \\ &\ldots \sum_{n_{k-1}=0}^{1} A^{**}_{n_{k-1}}\left(\sum_{i=1}^{k-2} n_i, L_{k-1}\right) A^{**}_1\left(\sum_{i=1}^{k-1} n_i, L_k\right), \end{aligned} \tag{39}$$

where

$$A^{*}_{n_l}\left(\sum_{i=1}^{l-1} n_i, L_l\right) = \\ = \left[e^{-\alpha \sum\limits_{i=1}^{l-1} n_i}(1-r)\right]^{n_l}\left[1-e^{-\alpha \sum\limits_{i=1}^{l-1} n_i}(1-r)\right]^{L_l-n_l} \tag{40}$$

and

$$A^{**}_{n_k}\left(\sum_{i=1}^{k-1} n_i, L_k\right) = \\ = \left\{1-\left[1-e^{-\alpha \sum\limits_{i-1}^{k-1} n_i}(1-r)\right]^{L_k}\right\}^{n_k}\left\{1-e^{-\alpha \sum\limits_{i-1}^{k-1} n_i}(1-r)\right\}^{L_k(1-n_k)}. \tag{41}$$

For example, if $P = P^*$ and $S = (2,(1,1))$, we will have

$$E[N(S,P^*)] = (1-r)[1+r+e^{-\alpha}(1-r)].$$

In practice, the formula (37) for evaluating the mean value of the number of errors encountered during the program testing process can be used if the program characteristic matrix P is known. If the probabilities $p_{n_k m_k}, n_k, m_k \in \{0,1,2,\dots L_k\}$, $k=\overline{1,K}$, are unknown, it is possible to determine the boundary values of this evaluation.
Let P^*, P^{**} denote the characteristic matrices of the program under testing of forms (31). Then, as proved in Worwa (2000), for both any program testing strategy S and any characteristic matrix P there is:

$$E[N(S,P^{**})] \le E[N(S,P)] \le E[N(S,P^*)], \tag{42}$$

where the quantities $E[N(S,P^*)]$ and $E[N(S,P^{**})]$ are determined by (38) and (39), respectively.

5. Conclusions

The formula (37) for determining the mean value of the number of errors encountered during the program testing process, has been obtained under the assumption that both the program testing scheme and the program reliability growth model are of the forms presented in Section 2. It is noteworthy that both the assumed program testing scheme and the program testing strategy are very popular in software testing practice. The assumptions concerning the program reliability-growth model, including the formula (21), have been made according to some precepts known from literature, e.g. Csenki (1990), Musa, Iannino and Okumoto (1987), Thayer, Lipov and Nelson (1978), Trachtenberg (1990). The formula (21) can be obtained on the basis of software reliability models proposed by Shooman (1972) and Jelinski and Moranda (1972).

In order to apply the methodology proposed in this paper, software run reliability data must be available. Compared to the amount of software reliability data reported in the literature, the amount of discrete-time software reliability data is rather limited in relation to continuous-time software reliability data, although some authors have published their own data.

It is noteworthy that the parameters (including r, α and the probabilities that form the characteristic matrix P) of the program reliability-growth model presented in the paper can be evaluated by using a process known as life testing or statistical-usage testing, in which long-term behaviour of the software is observed and values of these parameters are estimated on the basis of the observations, see e.g. Cobb and Mills (1990) or Thayer, Lipov and Nelson (1978).

Sensible estimation of the values of such parameters as r, α and p_{nm} requires a suitable data base of facts that contains information about similar former programs under testing. As far as the data base of facts is concerned, it is an important component of software engineering. The maintenance of the data base of facts is especially useful if a software development process is stable in a domain sense. The fundamental information that constitutes this data base of facts concerns the backgrounds, the circumstances, the reasons and the time of the program errors encountering. Detailed analysis of data included in the data base of facts enables better understanding of mutual relations between the types of software and the number of program errors encountered during the development process. Such mathematical and statistical methods as regression analysis, variation analysis, least squares method, maximum likelihood method etc. are the most popular for estimating the values of parameters that are commonly used in software reliability models. The practical usefulness of the parameter estimation methods mentioned above can be confirmed by the results of studies described in several papers, see e.g. Cobb and Mills (1990), Musa, Iannino and Okumoto (1987), Thayer, Lipov and Nelson (1978).

The knowledge of the mean value of the predicted number of errors encountered during the program testing process is very useful from the practical point of view. In particular, it makes possible to reasonably estimate both the duration and the cost of the program testing process. These estimations can be very useful for the planning of the program testing process. The mean value of the number of errors encountered during the program testing can be treated as a measure of the increase of the program reliability that was caused by the testing.

References

Basu, S. and Ebrahimi, N. (2003) Bayesian software reliability models based on martingale processes. *Technometrics* **2**, 150–158.

Cai, K.Y. (2000) Towards a conceptual framework of software run reliability modeling. *Information Science* **126**, 137–163.

Chen, M., Mathur, A.P. and Rego, V. (1995) Effect of testing techniques on software reliability estimates obtained using a time-domain model. *IEEE Transactions on Reliability* **44** (1), 97–103.

Chen, T.Y. and Yu, Y.T. (1994) On the relationship between partition and random testing. *IEEE Transactions on Software Engineering* **20** (12), 977–980.

Chen, T.Y. and Yu, Y.T. (1996) On the expected number of failures detected by subdomain testing and random testing. *IEEE Transactions on Software Engineering* **22** (2), 109–119.

Cobb, R.H. and Mills, H.D. (1990) Engineering software under statistical quality control. *IEEE Software* **16**, 44–54.

Csenki, A. (1990) Bayes predictive analysis of a fundamental software reliability model. *IEEE Transactions on Software Engineering* **39** (2), 177–183.

Gaudoin, O. (1999) Software reliability models with two debugging rates. *International Journal of Reliability* **1**, 31–42.

Hayakawa, Y. and Telfar, G. (2000) Mixed poisson-type processes with application in software reliability. *Mathematical and Computer Modelling* **31**, 151–156.

Jelinski, Z. and Moranda, P.B. (1972) *Software Reliability Research. Statistical Computer Performance Evaluation.* Academic Press, New York.

Jeske, D.R. and Pham, H. (2001) On the maximum likelihood estimates for the Goel-Okumoto software reliability model. *The American Statistician* **3**, 219–222.

Kit, E. (1995) *Software testing in the real world.* ACM Press Books.

Musa, J.D., Iannino, A. and Okumoto, K. (1987) *Software Reliability. Measurement, Prediction, Application.* McGraw-Hill, Inc.

Sawada, K. and Sandoh, H. (2000) Continuous model for software reliability demonstration testing considering damage size of software failures. *Mathematical and Computer Modelling* **31**, 321–326.

Schick, G.J. and Wolverton, R.W. (1978) An Analysis of Competing Software Reliability Models. *IEEE Transactions on Software Engineering* **SE-4** (2), 104–120.

Shooman, M.L. (1972) *Probabilistic Models for Software Reliability Prediction. Statistical Computer Performance Evaluation.* Academic Press, New York.

Thayer, T.A., Lipov, M. and Nelson, E.C. (1978) *Software Reliability.* North-Holland Publishing Company. Amsterdam.

Tokuno, K. and Yamada, S. (2000) An imperfect debugging model with two types of hazard rates for software reliability measurement and assessment. *Mathematical and Computer Modelling* **31**, 343–352.

Trachtenberg, M. (1990) A general theory of software reliability modeling. *IEEE Transactions on Software Engineering* **39** (1), 92–96.

Whittaker, J.A., Rekab, K. and Thomason, M.G (2000) A Markov chain model for predicting the reliability of multi-build software. *Information and Software Technology* **42**, 889–894.

Worwa, K. (1995a) Estimation of the program testing strategy. Part 1 — The same errors can be encountered. *Cybernetics Research and Development* **3-4**, 155–173.

Worwa, K. (1995b) Estimation of the program testing strategy. Part 2 — The same errors can not be encountered. *Cybernetics Research and Development* **3-4**, 175–188.

Worwa, K. (2000) *Modelling and estimation of software reliability growth during the testing process.* Publishers of Warsaw Technical University, Warsaw (in Polish).

Yamada, S. and Fujiwara, T. (2001) Testing-domain dependent software reliability growth models and their comparisons of goodness-of-fit. *International Journal of Reliability* **3**, 205–218.

Yang, M.C. and Chao, A. (1995) Reliability-estimation & stopping-rules for software testing, based on repeated appearances of bugs. *IEEE Transactions on Reliability* **44** (2), 315–321.

Zhang, X. and Pham, H. (2000) Comparisons of nonhomogeneous Poisson process software reliability models and its applications. *International Journal of Systems Science* **9**, 1115–1123.