EasyChair Preprint

№ 3424

Optimization of Test Case Allocation Scheme in Program Partition Testing

Kazimierz Worwa

May 16, 2020

Optimization of test case allocation scheme in program partition testing

Kazimierz Worwa[1]

[1] Military University of Technology, Kaliskiego 2 Str., 00-908 Warsaw, Poland
kazimierz.worwa@wat.edu.pl

Abstract. The issue of testing of the program in the aspect of comparing the effectiveness of random testing and partitioning testing was considered in this paper. However, unlike other works on this subject, the term partition does not refer directly to a specific subset of the program input domain, but to a fragment of the program, which is a specific set of paths, leading from the beginning of the program to its end, where each path belongs to exactly one of the subsets of paths. A formal model of the program under the testing is presented. This model is based on the notion of a control flow graph. The relationship between faults and failures is characterizes by means so called the characteristic matrix of the program under testing. The main result of the paper is obtaining formulae for the mean value of the number of faults encountered during the random testing and program partition testing . The formulation of the single-criterion optimization task which allows the determination of a testing strategy that maximizes the expected value of the number of program faults is also formulated. Considerations are illustrated with a simple numerical example.

Keywords: Random testing, Partition testing, Program reliability.

1. Introduction

It is well known that a software development process consists of the following four phases: specification (including user requirements definition and software requirements definition), design (structural design and detailed design), coding and testing. The practice of modern software engineering shows that the testing stage has the largest impact on the error detection process.

The method of generating the test data set, on the basis of which the software testing process will be carried out, depends on the adopted testing method, each of which is based on a specific criterion for the selection of test data set. Depending on the approach to the problem of designing the test data set, the testing methods used in practice can be divided into deterministic methods and random methods.

Deterministic testing methods assume the generation of a test data set in a exclusively deterministic way, i.e. without the participation of a random factor. Individual sets of input data, which are elements of the designed test set, are determined based on the analysis of the requirements specification or analysis of the source code of the tested program.

Unlike deterministic methods in random methods, part of the test data sets (or even all) used in the software testing process is created using a random factor, i.e. by randomly selection, whereby the most commonly used in practice of designing test cases is the sampling without returning. The method of creating a test data set based on a randomly selection is very widely used in testing practice. This is mainly due to its simplicity (it does not require e.g. a very labor-intensive analysis of the logical structure of the tested program), the direct consequence of which is the low cost of the testing process and high susceptibility to its automation. The second, which determines the wide practical use, feature of the random method is its high efficiency, measured by the number of errors detected in the testing process. Research conducted to compare the effectiveness of the most-used testing methods in practice has shown that for a wide class of programs, testing based on random generation of test data is more effective in terms of error detection capability than other methods [5, 7]. In practice, random testing is carried out as random testing or partition testing.

Random testing consists in using in the testing process a set of test cases, created by randomly selection (usually without returning) its subsequent elements from the entire set of program input domain, i.e. set of all test cases of the tested program. The main problem here is determining the probability distribution with which individual test cases are randomly drawn. In the absence of relevant premises as to the nature of this distribution, a uniform distribution is often assumed over the program domain. Very often used in practice variation of the random testing is a such testing in which the sampling of subsequent test cases is based on the probability distribution specified by the so-called the operational profile of the program, determining the probability of occurrence of individual test cases of the program in the conditions of its actual use.

Partition testing, also called subdomain testing, consists in dividing the set of a program domain into subsets called partitions and then drawing a specific number of tests from each subset. Consequently, the testing process

is carried out based on a number of test sets (corresponding to the number of separate partitions), with the sampling of individual data case within the extracted partitions is most often based on an uniform distribution.

A comparison analysis of partition testing and random testing has attracted significant attention in the literature. Specific contributions have been made in papers [1-8].

Various reliability metrics have been in use in the analysis of the effectiveness of subdomain testing and random testing, each based on a different intuition. Almost all of these measures are formulated in terms of failures rather than faults. Ideally, we would like to assess the effectiveness of testing in terms of the faults detected. Faults are the software defects caused by programmer errors, while a failure is an observed departure from the specified behavior of the software. Normally, only failures are revealed by testing, and the associated faults are usually identified during debugging.

In the article, as in the analytical works [1-4, 6, 7-8] the issue of random testing of the program in the aspect of comparing the effectiveness of random testing and partitioning testing is considered. However, unlike the works cited above, the term partition does not refer directly to a specific subset of the all program testing cases, but to a fragment of the program, which is a specific set of paths, leading from the beginning of the program to its end, where each path belongs to exactly one of the selected subsets of paths.

Unlike the software reliability measures used in these works formulated in terms of failures the expected number of faults detected to analyze subdomain and random testing will be used in this paper. A program fault may cause a number of failures, while a program failure may have more than one fault associated with it. There is no simple relationship between faults and failures. This makes it difficult to use faults in measuring the effectiveness of testing. In this paper relationship between faults and failures is characterizes by means so called the characteristic matrix of a program under testing.

The rest of this paper is organized as follows. Section 2 describes the formal model of the program under the testing. Section 3 discusses the use of the expected number of faults detected during the random testing of the program. Section 4 analyzes partition testing using the expected number of faults and compares it with random testing. Section 5 presents formulation of the single-criterion optimization task, which allows the determination of a testing strategy that maximizes the expected value of the number of program faults. Section 6 summarizes and concludes this paper.

2. Description of the program under the testing

The tested program will be characterized by means of a directed G graph, defined as follows:

$$G = \boldsymbol{(I, U)},$$

where:

$\boldsymbol{I}$ - set of graph vertices corresponding to the set of module numbers of the tested program:

$$\boldsymbol{I} = \{1, 2, ..., i, ..., I\},$$

$\boldsymbol{U} \subset \boldsymbol{I} \times \boldsymbol{I}$ - set of ordered pairs $(i, j) \in \boldsymbol{I} \times \boldsymbol{I}$, while a pair $(i, j) \in \boldsymbol{U}$, if after the i-th module is executed, the j-th module can be executed next.

Without loss of generality it can be assumed that the relevant program has one input module and one output module with numbers i_{IN}, i_{OUT} respectively, $i_{IN}, i_{OUT} \in \boldsymbol{I}$. In literature, the graph G is called the control flow graph.

Let $\boldsymbol{D}$ denote the set of numbers of all paths of the tested program, while the term path means the sequence of modules from the input module i_{IN} to the output module i_{OUT} for which there is at least one input data set that activates it:

$$\boldsymbol{D} = \{1, 2, ..., d, ..., D\}.$$

It is worth noting that the set $\boldsymbol{D}$ can be very numerous but it is always finished.

Let $\boldsymbol{I}_d$ denote a set of all module numbers of the d-th path:

$$\boldsymbol{I}_d = \left\{ i_{d,1}, i_{d,2}, ..., i_{d,k}, ..., i_{d,I_d} \right\}, \; d \in \boldsymbol{D},$$

where:

$i_{d,k}$ – a number of the k-th module of the d-th path (in the order of execution of the modules included in it), while $i_{d,1} = 1$ and $i_{d,I_d} = I$,

I_d – the number of modules forming the d-th program path.

Let us define the reliability of a module as the probability that the module performs its function correctly, i.e., the module produces the correct output and transfers control to the next module correctly. When a set of user input is supplied to the program, a sequence of modules will be executed. The reliability of the output will depend on the sequence of modules executed and the reliability of each individual module. We first assume that the

reliabilities of the modules are independent. This means that faults will not compensate each other, i.e., an incorrect output from a module will not be corrected later by subsequent modules. Since errors do not compensate each other, the result of the execution of the program is correct if and only if the proper sequence of modules is executed and in every instance of module execution, the module produces the correct result. The reliability of a module, in general, is a function of many factors, and the study of the reliability function of a module is beyond the scope of this paper. However, if no modification is made on the modules and the user environment does not change, the reliability function of a module should remain invariant. Let r_i stand for the *i*-th module reliability coefficient. The module reliability coefficients form a following vector $\boldsymbol{R}$, called program reliability vector

$$\boldsymbol{R} = (r_1, r_2, ..., r_i, ..., r_I) .$$

We next assume that the transfer of control among program modules is a Markov process. This implies that the next module to be executed will depend probabilistically on the present module only and is independent of the past history. It is noteworthy that this assumption may not be valid for all types of programs.

Let us represent the logical structure of the program by a directed graph where every node *i* represents a program module and a directed branch $(i, j) \in \boldsymbol{U}$ represents a possible transfer of control from the *i*-th module to the *j*-th module. To every directed branch $(i, j) \in \boldsymbol{U}$ we will attach a probability p_{ij} as the probability that the transition (*i, j*) will be taken when control is at *i*-th node. If $p_{ij} = 0$, the branch (*i, j*) does not exist. Without loss of generality, let us assume that the program graph has a single entry node *1* and a single exit node *M*.

Probabilities p_{ij}, $i, j = 1,2,..,I$, form the following square matrix $P = \left[p_{ij} \right]_{I \times I}$, called the transition matrix:

$$P = \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1i} & \cdots & p_{1I} \\ p_{21} & p_{22} & \cdots & p_{2i} & \cdots & p_{2M} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ p_{i1} & p_{i2} & \cdots & p_{ii} & \cdots & p_{iI} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ p_{I1} & p_{I2} & \cdots & p_{Ii} & \cdots & p_{II} \end{bmatrix} \tag{1}$$

where element p_{ij} means the probability of the event that after execution of the *i*-th module, the *j*-th module will be executed next, $i, j = 1,2,...,i,...,I$. The matrix *P* determines so called the operational profile of the program.

Let p_q, $p_q > 0$, mean probability of an activation by a single test case of the *d*-th path of the tested program, wherein

$$p_q = \prod_{i,j \in \boldsymbol{I}_d} p_{ij} \ , \ d \in \boldsymbol{D}. \tag{2}$$

The process of program execution for a given test case of program input domain consists in activating a certain sequence of modules corresponding to this case, forming one of the executable paths of the program. It is assumed that the measure of reliability of the considered program is the probability of its correct execution for a single, randomly selected, test case. This probability is equal to the product of the probabilities of the correct execution of the modules included in the path from the input module to the output module, which activates the given test case. Using the introduced assumptions, the program reliability coefficient with the reliability structure $\boldsymbol{R}$ and the transition matrix *P* can be determined as follows:

$$r = \sum_{d \in \boldsymbol{D}} p_d \prod_{i \in \boldsymbol{I}_d} r_i \ . \tag{3}$$

3. The random testing of the program

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 encounter any default or not successful, if program execution was incorrect, i.e. some default were encountered.

It is assumed that the testing process of the considered program consists in carrying out n test cases (tests), drawn from the set of all possible test cases, with the draw being carried out based on the probability distribution, determined by the operational profile of the program, determined by the transition matrix *P*, defined by (1).

According to assumed testing scheme a situation that a number of different test cases of all *n* tests encounter the same program fault is possible. So, according to the note mentioned above, a situation that several runs will lead to encounter the same program fault is possible.

Let q_{nm} define the probability of an event that *n* faults will be encountered as a result of the program testing if there are *m* tests that lead to reveal failures during the program testing process. If we assume that every run of the program with a single test can lead to reveal at most one program failure, and every failure makes it possible to detect exactly one fault we have

$$0 \le q_{nm} \le 1 \text{ if } n \le m,\ n \ge 0, \tag{4}$$

where in particular

$$\begin{aligned} q_{00} &= q_{11} = 1 \\ q_{nm} &= 0 \quad if\ n > m \end{aligned} \tag{5}$$

Probabilities q_{nm}, $n \in \{0,1,2,...,m\}$, $m \in \{0,1,2,...,\}$, that are defined by (4-5) form an infinite matrix $Q = [q_{nm}]$ that has a following form

$$Q = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & ... \\ 0 & 1 & q_{12} & q_{13} & q_{14} & ... \\ 0 & 0 & q_{22} & q_{23} & q_{24} & ... \\ 0 & 0 & 0 & q_{33} & q_{34} & .. \\ 0 & 0 & 0 & 0 & q_{44} & ... \\ & & ... & & & \end{bmatrix}, \tag{6}$$

while

$$\sum_{n=0}^{\infty} q_{nm} = 1, \quad m \in \{0,1,2,...\}. \tag{7}$$

The matrix *Q* contains the values *0* below the main diagonal because – in accordance with earlier assumption that every run of the program with a single test can lead to reveal at most one program failure and every failure makes it possible to detect exactly one fault - it is not possible to encounter more different faults than the number of failures.

The values of probabilities q_{nm} that form the matrix *Q* depend on a logical structure of the program under the testing. In particular, an important impact on these probabilities has: number of paths that have been identified in the program, degree of covering individual paths, that can be measured by number of program instructions that belong to two or more paths and length of individual paths, i.e. measured by number of program instructions that are executed in case of path activation. The matrix *Q* will be called the characteristic matrix of the program under the testing.

As a result of the program testing process in accordance with the assumed testing scheme a number of failures may be revealed. Because different tests can reveal the same failures, the real number of faults encounter as a result of running n tests may be less. of course, the faults associated with reveal failures are usually identified during debugging.

Let *M(P)* denote the number of tests that lead to incorrect execution of the program under the testing with the transition matrix *P*, i.e. to reveal failures, during the program testing process.

Let *N(P,Q)* mean the total number of faults encountered during the program testing process with the transition matrix *P* and the characteristic matrix *Q*. While planning the program testing process it is reasonable to consider the values *M(P,Q)* and *N(P,Q)*, as random variables, while *N(P,Q)* $\le$ *M(P)*.

Joint distribution of the random variables (*N(P,Q), M(P)*) can be determined as follows:

$$Pr\{N(P,Q) = n, M(P) = m\} = Pr\{N(P,Q) = n|_{M(P)=m}\} Pr\{M(P) = m\} \tag{8}$$

According to earlier denotations we have

$$p_{nm} = Pr\{N(P,Q) = n|_{M(P)=m}\}, \tag{9}$$

and

$$Pr\{M(P) = m) = \binom{L}{m} (1-r)^m r^{L-m}, \tag{10}$$

where *L* is the number of tests performed during program testing process.

So,

$$Pr\{N(P,Q) = n, M(P) = m\} = q_{nm} \binom{L}{m} (1-r)^m r^{L-m}. \tag{11}$$

Probability distribution of the random variable *N(P,Q)* can be determined as a marginal distribution in a distribution of two-dimensional random variable (*N*, *M*):

$$Pr\{ N(P,Q) = n \} = \sum_{m=0}^{L} Pr\{ N(P,Q) = n, M(P) = m \} =$$

$$\sum_{m=0}^{L} Pr\{ N(P,Q) = n \big|_{M(P)=m} \} Pr\{ M(P) = m \} = \sum_{m=0}^{L} q_{nm} \binom{L}{m} (1-r)^m r^{L-m}. \quad (12)$$

The knowledge of the probability distribution function (11) makes possible to obtain a formula for the mean value of the number of faults encountered during the program testing process. We have

$$E_r [N(P,Q)] = \sum_{n=0}^{L} n \, Pr\{ N(P,Q) = n \}, \quad (13)$$

and then, according to (12):

$$E_r [N(P,Q)] = \sum_{n=0}^{L} n \sum_{m=0}^{L} q_{nm} \binom{L}{m} (1-r)^m r^{L-m}. \quad (14)$$

Let Q^*, Q^{**} denote characteristic matrices that have form:

$$Q^* = \begin{bmatrix} 1 & 0 & 0 & 0 & \dots \\ 0 & 1 & 0 & 0 & \dots \\ 0 & 0 & 1 & 0 & \dots \\ 0 & 0 & 0 & 1 & \dots \\ & & \dots & & \end{bmatrix}, \qquad Q^{**} = \begin{bmatrix} 1 & 0 & 0 & 0 & \dots \\ 0 & 1 & 1 & 1 & \dots \\ 0 & 0 & 0 & 0 & \dots \\ 0 & 0 & 0 & 0 & \dots \\ & & \dots & & \end{bmatrix}. \quad (15)$$

The characteristic matrices Q^*, Q^{**} correspond with some specific, extreme forms of the program testing scheme. The matrix Q^* correspond with a such testing scheme in which different tests can encounter only new faults, i.e. not encounter by tests that were executed earlier. The matrix Q^{**} correspond with a such testing scheme in which all tests can encounter at most one and the same program fault.

The formula (14) will be simplified if the program characteristic matrix Q is of the form (15), i.e.:

$$E_r [N(P,Q^*)] = \sum_{n=0}^{L} n \binom{L}{n} (1-r)^n r^{L-n} = L(1-r) \quad (16)$$

and

$$E_r [N(P,Q^{**})] = \sum_{n=0}^{L} q_{nm} \binom{L}{n} (1-r)^n r^{L-n} = \sum_{m=1}^{L} 1 \binom{L}{m} (1-r)^m r^{L-m} = 1 - P\{ M(P) = 0 \} = 1 - r^L. \quad (17)$$

In practice, the formula (14) for evaluating the mean value of the number of faults encountered during the program testing process can be used if the program characteristic matrix Q is known. If the probabilities q_{nm}, $n,m \in \{0,1,2,...L\},$, are unknown, it is possible to determine the boundary values of this evaluation.

Let Q^*, Q^{**} denote characteristic matrices of the program under the testing of forms (15). Then, as proved in [9], for any characteristic matrix P is:

$$E_r [N(P,Q^{**})] \le E_r [N(P,Q)] \le E_r [N(P,Q^*)], \quad (18)$$

where the quantities $E_r [N(P,Q^*)]$ and $E_r [N(P,Q^{**})]$ are determined by (16) and (17), respectively.

4. The partition testing of the program

Testing a program with partitions consists in dividing the program into fragments, each of which is a specific set of program paths, leading from the input module i_{IN} to the output module i_{OUT}, with each path belonging to exactly one of the selected subsets of paths. In further considerations, these subsets will be called program partitions.

Let's consider the division of the graph $G = (I, U)$ of the tested program into the subgraphs $G_k = (I_k, U_k)$ such that $I_k \subseteq I$, $U_k \subseteq U$, while $i_{IN}, i_{OUT} \in I_k$, $k=1,2 \dots K$, and

$$\bigcup_{k=1}^{K} I_k = I, \quad \bigcup_{k=1}^{K} U_k = I. \quad (19)$$

Let's consider the division of the set of numbers of all paths of the tested program leading from the input module i_{IN} to the output module i_{OUT} into disjoint subsets $D_k \subseteq D$:

$$D_k = \left\{ d_{k,1}, d_{k,2}, \dots, d_{k,l}, \dots, d_{k,D_k} \right\}, \; k=1,2 \dots K,$$

wherein

$$\bigcup_{k=1}^{K} \boldsymbol{D}_k = \boldsymbol{D} \text{ and } \bigcap_{k=1}^{K} \boldsymbol{D}_k = \boldsymbol{\Phi} . \tag{20}$$

Each subgraph $G_k = (\boldsymbol{I}_k, \boldsymbol{U}_k)$ has a fragment of the transition matrix $P_k = \left[p_{ij}^k \right]_{I_k \times I_k}$. It takes place

$$\sum_{k=1}^{K} P_k = P , \tag{21}$$

where the summation operator is the matrix summation operator.
The division of the set of all program paths leading from the input module i_{IN} to the output module i_{OUT} into disjoint subsets $\boldsymbol{D}_k \subseteq \boldsymbol{D}$ means the division of the set of all possible test cases into disjoint subsets, with each test case activating exactly one path. Therefore, this division can be treated as one of the ways to divide the all program input domain into the partitions referred to in [1-3, 5-8]. In the presented considerations, the term "partition" refers, however, not so much to a specific subset of all possible test cases, but to a part of the program being tested, represented by the subgraph $G_k = (\boldsymbol{I}_k, \boldsymbol{U}_k)$, the probability of transitions between modules described by the matrix P_k, $k=1,2,\ldots, K$. In further considerations, the term "partition" will be understood as a part of the program being tested, which is a subset of the set of paths leading from the input module i_{IN} to the output module i_{OUT}.

Similarly to the formulae (3), the reliability coefficient of the k-th partition of the program with the transition matrix P_k, understood as the probability of correct execution of the program for a single test case activating the path with the number from the set $\boldsymbol{D}_k$, can be determined as follows:

$$r_k = \sum_{d \in \boldsymbol{D}_k} p_d \prod_{i \in \boldsymbol{I}_d} r_i , \; k=1,2,\ldots, K. \tag{22}$$

Let n_k be the number of test cases randomly selected from the subset of all test cases that activate the set of paths with the numbers from the set $\boldsymbol{D}_k$, $k=1,2,\ldots, K$.
As an alternative to the testing strategy described in Section 3, the testing strategy consisting in randomly selection from each set $\boldsymbol{D}_k$ $n_k > 0$ test cases, $k = 1,2, \ldots, K$, will be now considered.
Let $\boldsymbol{\mathcal{L}}$ denote the set of vectors as follows

$$\boldsymbol{\mathcal{L}} = \{ \boldsymbol{L} = (L_1, L_2, \ldots L_k, \ldots, L_K) : \; L_k > 0, \sum_{k=1}^{K} L_k = L \} \tag{23}$$

The vector $\boldsymbol{L} \in \boldsymbol{\mathcal{L}}$ will be called a testing strategy with program partitioning. To ensure the proper conditions for comparability of random testing strategy and testing strategy with program partitioning, it is assumed that this occurs

$$\sum_{k=1}^{K} L_k = L \text{ and } \sum_{k=1}^{K} r_k = r . \tag{24}$$

Let $M_k(\boldsymbol{L},P)$ denote the number of tests that lead to incorrect execution of the k-th program partition under the testing with the testing strategy $\boldsymbol{L}$ and transition matrix P_k, i.e. to reveal failures, during the program partition testing process.

Let $N_k(\boldsymbol{L},P,Q)$ mean the total number of faults encountered during the program partition testing process with the testing strategy $\boldsymbol{L}$ and the transition matrix P and the characteristic matrix Q. While planning the program testing process it is reasonable to consider the values $M_k(\boldsymbol{L},P)$ and $N_k(\boldsymbol{L},P,Q)$, as random variables, while $N_k(\boldsymbol{L},P,Q) \le M_k(\boldsymbol{L},P)$.

Conducting considerations analogous to those presented in Section 3, it can be shown that the expected value of a number of faults encountered during the testing the k-th partition determines the following relationship:

$$E_k [N(\boldsymbol{L},P_k,Q_k)] = \sum_{n=0}^{L_k} n \sum_{m=0}^{L_k} q_{nm} \binom{L_k}{m} (1-r_k)^m r_k^{L_k - m} , \quad k=1,2,\ldots, K. \tag{25}$$

Thus, the mean value of total number of faults encountered during the testing of all K partitions takes the form

$$E_p [N(\boldsymbol{L},P,Q)] = \sum_{k=1}^{K} E[N_k(\boldsymbol{L},P_k,Q_k)] = \sum_{k=1}^{K} \sum_{n=0}^{L_k} n \sum_{m=0}^{L_k} q_{nm} \binom{L_k}{m} (1-r_k)^m r_k^{L_k - m} , \tag{26}$$

while L_k is the number of tests performed during testing the k-th program partition.
The formula (26) will be simplified if the program characteristic matrix Q is of the form (15), i.e.:

$$E_p [N(\boldsymbol{L},P,Q^*)] = \sum_{k=1}^{K} \sum_{n=0}^{L_k} n \binom{L_k}{n} (1-r_k)^n r^{L_k - n} = \sum_{k=1}^{K} L_k (1-r_k) = L - \sum_{k=1}^{K} L_k r_k \tag{27}$$

and

$$\mathrm{E}_p[\mathrm{N}(\boldsymbol{L},P,Q^{**})] = \sum_{k=1}^{K}\sum_{n=0}^{L_k} q_{nm}\binom{L_k}{n}(1-r_k\sum_{n=0}^{L_k} q_{nm}\binom{L_k}{n}(1-r_k)^n r^{L_k-n})^n r_k^{L_k-n} =$$

$$= \sum_{k=1}^{K}\sum_{m=1}^{L_k}\binom{L_k}{m}(1-r_k)^m r_k^{L_k-m} = \sum_{k=1}^{K}(1-P\{M_k(\boldsymbol{L},P_k)=0\}) = \sum_{k=1}^{K}(1-r_k^{L_k}) = K - \sum_{k=1}^{K} r_k^{L_k}. \quad (28)$$

Taking into account the relationship (28), the following lower and upper estimations of the mean value of total number of faults encountered during the partition testing of the program with a given characteristic matrix Q can be determined in the form:

$$E_p[N(\boldsymbol{L},P,Q^{**})] \le E_p[N(\boldsymbol{L},P,Q)] \le E_p[N(\boldsymbol{L},P,Q^{*})]. \quad (29)$$

Taking into account the relationship (18), the following lower and upper estimations of the mean value of total number of faults encountered during the partition testing of the program with a given characteristic matrix Q can be determined in the form:

$$E_p[N(\boldsymbol{L},P,Q^{**})] \le E_p[N(\boldsymbol{L},P,Q)] \le E_p[N(\boldsymbol{L},P,Q^{*})], \quad (30)$$

and therefore

$$\sum_{k=1}^{K-1}(1-r_k^{L_k}) + 1 - r_K^{L-K+1} \le E_p[\boldsymbol{L},P,N(Q)] \le L(1-r_k) + Kr_k - r. \quad (31)$$

5. Optimization of the program partition testing

The basic problem to be solved at the planning and organization stage of the software testing process is to specify a subset of the set of all possible test cases that would maximize the probability of detecting all program faults made in earlier stages of the software development process. This problem arises because, in general, due to the duration and cost of the testing process, it is impossible to test the software based on the entire set of all possible test cases. The problem of defining the mentioned subset is the problem of designing the test cases (tests). Each test is an acceptable combination of values that can take the input variables of the software program being tested, required for its single run, and the input variables are those variables whose values are determined directly based on the input data, e.g. as a result of executing the data loading instructions.

As noted earlier, the partition program testing consists of dividing the program into fragments, each of which is a specific set of paths, leading from the input module i_{IN} to the output module i_{OUT}, with each path belonging to exactly one of the selected subsets of paths. In the process of planning the implementation of the program testing stage, it is necessary to decide how many test cases should be used for testing each of program partitions.
Mentioned problem can be solved based on the formulation and solution of a single-criterion optimization task, allowing the determination of a testing strategy that maximizes the expected value of the number of program faults.

As noted earlier, the program testing process with partitions involves splitting the program into fragments, each of which is a specific set of program paths, leading from the initial vertex i_{IN} to the final vertex i_{OUT}, with each path belonging to exactly one of the selected subsets of paths. In the process of planning the program testing process, we should decide how many test cases should be used for testing each of program partitions.

The aforementioned problem can be solved based on the formulation and solution of the single-criterion optimization task, which allows the determination of a testing strategy that maximizes the expected value of the number of program faults. The optimization problem that was mentioned above can be formulated as follows:
for specific transition matrices P and characteristic matrix Q determine the testing strategy $\overline{\boldsymbol{L}} \in \mathcal{L}$ such that:

$$E_p[N(\overline{\boldsymbol{L}},P,Q)] = \max_{\boldsymbol{L}\in\mathcal{L}} E_p[N(\boldsymbol{L},P,Q)], \quad (32)$$

where quantity $E_p[N(\boldsymbol{L},P,Q)]$ is determined by the formula (26) and $\mathcal{L}$ is a set of acceptable strategies defined as follows

$$\mathcal{L} = \{\boldsymbol{L} = (L_1, L_2, \ldots L_k, \ldots, L_K): \; K>1,\, L_k>0,\, \sum_{k=1}^{K} L_k = L\}. \quad (33)$$

6. Numerical example

The methodology of determination of the program reliability structure that has been presented can be illustrated by the following numerical example. Let Fig. 1 be the directed graph representing the control structure of a program with seven modules, where the module number *1* represents the input module and the module number *7* is the output module.
Let's assume that the control flow graph of the tested program has the form shown in Figure 1.

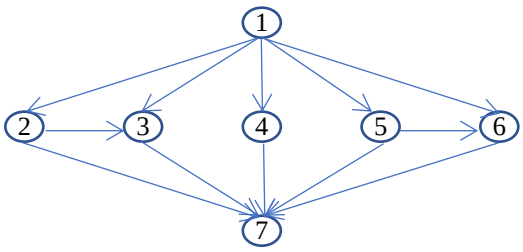

Figure 1. Graph of the sample program

According to the Figure 1 we have:

$\boldsymbol{I} = \{1, 2, 3, 4, 5, 6, 7\}$, $i_{IN} = 1$, $i_{OUT} = 7$,

$\boldsymbol{U} = \{(1, 2),(1, 3), (1, 4), (1, 5), (1, 6),(2, 3), (5, 6), (2, 7), (3, 7), (4, 7), (5, 7), (6, 7)\}$,

$\boldsymbol{D} = \{1, 2, 3, 4, 5, 6, 7\}$,

$\boldsymbol{I}_1 = \{1, 2, 7\}$, $\boldsymbol{I}_2 = \{1, 2, 3, 7\}$, $\boldsymbol{I}_3 = \{1, 3, 7\}$, $\boldsymbol{I}_4 = \{1, 4, 7\}$, $\boldsymbol{I}_5 = \{1, 5, 7\}$, $\boldsymbol{I}_6 = \{1, 5, 6, 7\}$, $\boldsymbol{I}_7 = \{1, 6, 7\}$.

Let the program characteristic matrix have the form

$$P = \begin{bmatrix} 0 & 0{,}4 & 0{,}1 & 0{,}3 & 0{,}1 & 0{,}1 & 0 \\ 0 & 0 & 0{,}7 & 0 & 0 & 0 & 0{,}3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0{,}6 & 0{,}4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Then, according to (2), the probabilities of activation by a single test case of paths have the following values:

$p_1 = p_{12} p_{27} = 0{,}096$, $p_2 = p_{12} p_{23} p_{37} = 0{,}2016$, $p_3 = p_{13} p_{37} = 0{,}09$, $p_4 = p_{14} p_{47} = 0{,}24$,

$p_5 = p_{15} p_{57} = 0{,}028$, $p_6 = p_{15} p_{56} p_{67} = 0{,}0294$, $p_7 = p_{16} p_{67} = 0{,}07$.

Let the values of the program reliability vector is as in Table 1.

Table 1. Module reliability coefficients.

i	1	2	3	4	5	6	7
r_i	1	0,8	0,9	0,8	0,7	0,7	1

Then, according to (3), the value of the program reliability coefficient is $r = 0.755$. Let's assume that the testing process of the investigated program consists in executing 10 randomly selected test cases, i.e. $L = 10$. Let the characteristic matrix Q has the form

$$Q = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0{,}8 & 0{,}3 & 0{,}1 & 0{,}05 & 0{,}05 & 0{,}05 & 0{,}05 & 0{,}05 \\ 0 & 0 & 0{,}2 & 0{,}5 & 0{,}6 & 0{,}2 & 0{,}1 & 0{,}05 & 0{,}05 & 0{,}05 \\ 0 & 0 & 0 & 0{,}2 & 0{,}2 & 0{,}5 & 0{,}3 & 0{,}3 & 0{,}1 & 0{,}05 \\ 0 & 0 & 0 & 0 & 0{,}1 & 0{,}2 & 0{,}4 & 0{,}3 & 0{,}3 & 0{,}15 \\ 0 & 0 & 0 & 0 & 0 & 0{,}05 & 0{,}1 & 0{,}2 & 0{,}2 & 0{,}3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0{,}05 & 0{,}05 & 0{,}1 & 0{,}2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0{,}05 & 0{,}1 & 0{,}05 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0{,}05 & 0{,}05 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0{,}05 \end{bmatrix}$$

According to (16), the value of mean value of the number of faults encountered during the program testing process is $E_r[N(P,Q)] = 1{,}557$. If the characteristic matrices have the form (15) we have respectively $E[N(P,Q^*)] = 2{,}45$ and $E[N(P,Q^{**})] = 0{,}9398$. That means, according to (18), that indeed it happens

$$E_r[N(P,Q^{**})] = 0{,}9398 \le E_r[N(P,Q)] \le E_r[N(P,Q^*)] = 2{,}45.$$

Let $K = 3$ and program partitions have the form presented on the Figure 2.

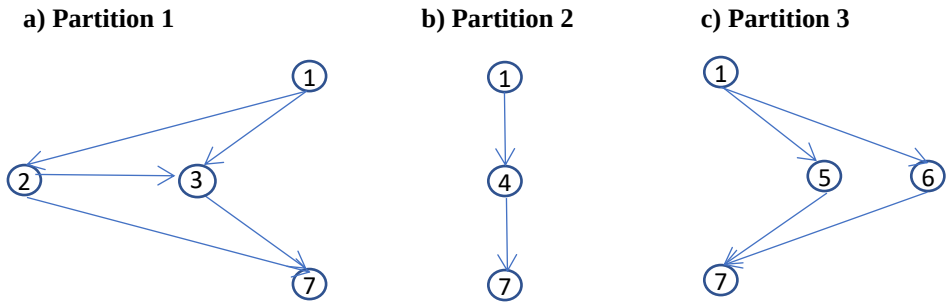

Figure 2. Sample program partitions

According to (22) we can calculate the values of the *k*-th program partition coefficients, *k=1, 2, 3*. The values of these coefficients are summarized in Table 2.

Table 2. Values of the program partitions coefficients.

k	*1*	*2*	*3*
r_k	*0,3876*	*0,24*	*0,1274*

Of course, according to (24) it occurs $\sum_{k=1}^{3} r_k = r = 0{,}775$.

Table 3. The mean values of the number of faults encountered during the program testing.

Strategy number	$\boldsymbol{L}$			$E_p[N(\boldsymbol{L},P,Q)]$	$E_r[N(P,Q)]$
	L_1	L_2	L_3		
1	*8*	*1*	*1*	*4,512690*	*1,557041*
2	*7*	*2*	*1*	*4,481045*	
3	*6*	*3*	*1*	*4,546941*	
4	*5*	*4*	*1*	*4,594280*	
5	*4*	*5*	*1*	*4,691253*	
6	*3*	*6*	*1*	*4,819420*	
7	*2*	*7*	*1*	*4,918613*	
8	*1*	*8*	*1*	*4,971773*	
9	*7*	*1*	*2*	*4,446580*	
10	*6*	*2*	*2*	*4,403888*	
11	*5*	*3*	*2*	*4,472574*	
12	*4*	*4*	*2*	*4,534005*	
13	*3*	*5*	*2*	*4,642040*	
14	*2*	*6*	*2*	*4,771958*	
15	*1*	*7*	*2*	*4,869695*	
16	*6*	*1*	*3*	*4,624030*	
17	*5*	*2*	*3*	*4,584128*	
18	*4*	*3*	*3*	*4,666907*	
19	*3*	*4*	*3*	*4,739398*	
20	*2*	*5*	*3*	*4,849184*	
21	*1*	*6*	*3*	*4,977647*	
22	*5*	*1*	*4*	*4,705099*	
23	*4*	*2*	*4*	*4,679288*	
24	*3*	*3*	*4*	*4,773128*	
25	*2*	*4*	*4*	*4,847371*	
26	*1*	*5*	*4*	*4,955702*	
27	*4*	*1*	*5*	*4,900472*	
28	*3*	*2*	*5*	*4,885723*	
29	*2*	*3*	*5*	*4,981314*	
30	*1*	*4*	*5*	*5,054102*	
31	*3*	*1*	*6*	*5,103681*	
32	*2*	*2*	*6*	*5,090683*	
33	*1*	*3*	*6*	*5,184819*	
34	*2*	*1*	*7*	*5,210870*	
35	*1*	*2*	*7*	*5,196417*	
36	*1*	*1*	*8*	*5,264051*	

The analysis of the data contained in Table 3 shows that for each strategy $\boldsymbol{L} \in \mathcal{L}$ the relationship (30) happens indeed. We can also notice that the best, i.e. the strategy that maximizes the expected value of the number of errors detected is the strategy $\overline{\boldsymbol{L}} = (1,1,8)$ for which $E_p[N(\boldsymbol{L},P,Q)] = 5{,}264051$. Successively, the worst, i.e. the

strategy that minimizes the expected value of the number of errors detected is the strategy $\overline{\overline{\boldsymbol{L}}} = (6,2,2)$ for which $E_p[N(\boldsymbol{L},P,Q)] = 4{,}403888$. From the calculations carried out in the example under consideration it follows that in the light of the program reliability coefficient, any program testing strategy with partitioning is better than a random testing strategy.

7. Conclusions

In this paper, as in the analytical works [1-4, 6, 8], the issue of random testing of the program in the aspect of comparing the effectiveness of random testing and partitioning testing was considered. However, unlike the works cited above, the term partition does not refer directly to a specific subset of the all program testing cases, but to a fragment of the program, which is a specific set of paths, leading from the beginning of the program to its end, where each path belongs to exactly one of the selected subsets of paths.

A formal model of the program under the testing was presented. This model is based on the notion of a control flow graph. It was assumed that the probabilities of transitions between individual program modules are determined by the transition matrix *P*, the element p_{ij} means the probability of the event that after execution of the *i*-th module, the *j*-th module will be executed next. The transition matrix defines so called operational profile of the program, determining the probability of occurrence of individual test cases of the program in the conditions of its actual use.

The use of the expected number of faults detected during the random testing of the program was discusses. Unlike the software reliability measures used in these works formulated in terms of the expected number of faults to analyze subdomain and random testing have been used. The relationship between faults and failures is characterizes by means so called the characteristic matrix *Q* of the program under testing, while the q_{nm} element of that matrix define the probability of an event that *n* faults will be encountered as a result of the program testing if there are *m* tests that lead to reveal failures during the program testing process. The values of probabilities q_{nm} that form the matrix Q depend on a logical structure of the program under the testing. The main result of this section is obtaining a formula for the mean value of the number of faults encountered during the program testing process. In practice, the formula for evaluating the mean value of the number of faults encountered during the program testing process can be used if the program characteristic matrix Q is known. In this section shown that if this matrix is unknown, it is possible to determine the both sides boundary values of the mean value of the number of faults encountered during the program testing.

Using the expected number of faults partition testing was analyzed and compares with random testing. The main result of this section is obtaining a formula for the total mean value of the number of faults encountered during the testing of all partitions.

A formulation of the single-criterion optimization task, which allows the determination of a testing strategy that maximizes the expected value of the number of program faults.

Presented considerations have been illustrated by a simple numerical example.

References

1. Boland P.J., Singh H., Cukic B., *Comparing partition and random testing via majorization and Schur functions*, IEEE Transactions on Software Engineering, Vol. 29, Jan. 2003.
2. Chen T.Y., Yu Y.T., *On the relationship between partition and random testing*. IEEE Transactions on Software Engineering, 20, No. 12, 1994.
3. Chen T.Y., Yu Y.T., *A more general sufficient condition for partition testing to be better than random testing*. Information Processing Letters, Vol. 57, No. 3, 1996.
4. Chen T., Y., Yu Y.,T., *On the Expected Number of Failures Detected by Subdomain Testing and Random Testing*, IEEE Transactions on Software Engineering, Vol. 22, No. 2, 1996.
5. Duran J. W., Ntafos S.C, *An evaluation of random testing.* IEEE Transactions on Software Engineering, Vol. 10, No. 4, 1984.
6. Gutjahr W.J., *Partition testing vs. random testing: the influence of uncertainty*, IEEE Transactions on Software Engineering, Vol. 25, No. 5, 1999.
7. Weyuker E. J., Jeng B. *Analyzing partition testing strategies*. IEEE Transactions on Software Engineering, Vol. 17, No. 7, 1991.
8. Worwa K.: *A comparison analysis of testing strategies based on subdomain testing and random testing*. Proceedings of the Fourth International Conference on Dependability of Computer Systems DepCos – Relcomex, Brunów, June 30 – July 2, 2009.
9. Worwa K.: *A discrete-time software reliability-growth model and its application for predicting the number of errors encountered during program testing*. Control and Cybernetics, No 2, 2005.