Method for determining the reliability parameters of software for complex decision support systems

Kazimierz Worwa[1,*], *Tadeusz* Nowicki[1], *Robert* Waszkowski[1], and *Maciej* Kiedrowicz[1]

[1]Military University of Technology, Kaliskiego 2 str., 00-908 Warsaw, Poland

Abstract. The testing stage, creating great opportunities to verify and shape software reliability, significantly increases the cost of its production. The effectiveness of the work related to testing, expressed by the interdependence of the level of program product reliability and the cost of testing it, strongly depends on the adopted testing strategy, specifying the organization and scope of the work performed. In this situation, therefore, there is a need to define the conditions for a compromise in terms of reliability and cost requirements set for the software. The practical finding of this compromise can be greatly facilitated if there are possibilities to formally assess the level of software quality and the cost of testing it using appropriate indicators. The paper attempts to describe a method of determining a program testing strategy as a result of solving a two-criteria optimization problem, with the program reliability coefficient and the cost of testing as component criteria. The paper consists of description of the program testing process and mathematical model of this process, formulation of the problem of two-criteria optimization of the program testing strategy, remarks on method of solving the problem that has been formulated. proposed. To illustrate the method of finding an optimal testing strategy that has been proposed a numerical example is considered.

1 Introduction

The work presents a formal description of the logical structure of the program under consideration, defined by a set of its component modules and interconnections existing between them. The logical structure of the program is reflected in the structure of the program reliability coefficient, the value of which depends on the value of reliability measures of its component modules. The problem of a two-criteria optimization of the program testing strategy is formulated, with the program reliability measure and the cost of testing as component criteria. The program testing strategy, determined as a result of this task, ensures simultaneous maximization of the value of the adopted reliability coefficient and minimization of the program testing cost. The presented method is illustrated by a numerical example.

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 predicted time spent on testing activities or by predicted cardinality of the set of input data that should be used for the testing.

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 program errors or not successful, if program execution was incorrect, i.e. some errors were encountered.

2 Description of the program testing process

The behavior of the computer program, depending on the manner, in which different program modules affect each other, is defined by the module structure of the program. The interactions between different modules in the form of mutual control transfer are present only during the execution of the program. The program structure may also contain some information on frequency of activation of the individual modules. When the program is working, some emergency situations related to software errors may occur. The software error may be connected with the implementation of any module or transfer of control between the two modules. A possibility of occurrence of some emergency situations during the use of the software is described by various software reliability coefficients. The knowledge of the component (module) structure of the examined software allows using the coefficients, whose values depend on such structure and the reliability coefficients of its particular components.

Depending on the method applied in that respect, it is possible to distinguish two approaches to the reliability

modeling process, on the basis of: the structure set by the source code instructions, structure based on logical paths [5].

The models based on the architecture set by the source code for modeling the software architecture use the so-called control transfer graph, illustrating a possibility of transferring control between the blocks of the source code. The nodes on the structure are indicated by the so-called program control guidelines. It is often assumed in the models of the discussed class that the control transfer between them may be described by the Markov chains. This assumption means that the control transfer after implementation of a specific module to other modules does not depend on previous activation of the program blocks. The software architecture is modeled by the Markov chain, e.g. discrete in states and in time. The representative examples of such models may be found in studies [2, 6, 10].

In the models based on the logical path concept, the software architecture is also created by separate blocks of the source code, e.g. by the modules, which are not - however - analyzed as independent components, but as sequences of components executed one by one in case of activation of a given path by an appropriate set of input data. Knowing the values of the reliability indicators of the components that create a given logical path, it is possible to determine the value of the reliability indicator of the whole software, using the knowledge of the architecture of the software. The representative examples of the models based on logical paths may be found in work [9, 16].

The program being tested will be characterized using the directed graph *G* defined as follows:

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

where

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

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

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

Without lost of generality it can be assumed that the graph of the considered program is an unigraph, with one input module and one output module with numbers i_{WE}, i_{WY} respectively, $i_{WE}, i_{WY} \in \boldsymbol{I}$.

In the program graph *G* we can distinguish a certain set of paths connecting the initial node with the final node, where the term “paths” is understood as in the graphs and networks theory [4]. Due to the fact that the analyzed graph *G* is a directed unigraph, each path connecting vertices $i_{WE}, i_{WY} \in \boldsymbol{I}$ can be unambiguously determined by giving the numbers of vertices through which it "passes". Any such path, for which, moreover, there is at least one set of program input data that activates it, will be called logical, whereby activating a specific logical path means the subsequent execution of the modules that make up it.

Let $\boldsymbol{Q}$ be a set of numbers of all logical paths of the tested program:

$$\boldsymbol{Q} = \{1,2,...,q,...,Q\}.$$

Let $\boldsymbol{I_q}$ mean a set of module numbers of the q-th logical path:

$$\boldsymbol{I_q} = \{i_{q,1}, i_{q,2}, ..., i_{q,k}, ..., i_{q,I_q}\}, \quad q \in \boldsymbol{Q},$$

where

$i_{q,k}$ the number of the k-th module of the q-th logical path (eg. in the order of executing the modules forming the given path),

I_q number of modules forming the q-th logical path.

It is assumed that the process of testing considered program consists in independent testing of its component modules, and that the testing of the i-th module, consists of the so-called testing cycles, each of which includes:

- executing the module with a number of previously prepared test data set;
- evaluation of the obtained results and location and removal of any errors found.

The adopted organization of the program testing process corresponds to the actual stage of the so-called autonomous testing of program components, followed by the so-called integration testing of the program [1, 8, 11-14].

Let *S* denote the testing strategy of the program under consideration, defined as follows

$$S = (S_1, S_2, ..., S_i, ..., S_I), \tag{1}$$

where:

S_i testing strategy of the i-th module, defined as follows:

$$S_i = (K_i, (L_{i,1}, L_{i,2}, ..., L_{i,k}, ..., L_{i,K_i})),$$

K_i number of i-th module testing cycles,

$L_{i,k}$ number of test cases (tests), based on which the i-th module is tested in the k-th cycle of its testing process, $k = \overline{1,K_i}$, $i \in \boldsymbol{I}$.

Let P_i, $i \in \boldsymbol{I}$ denotes the so-called characteristic matrix of the i-th module of the considered program i.e., $P_i = [p^i_{n,m}]$, $n,m \in \{0,1,2,...\}$, while $p^i_{n,m}$ means the probability of the event whereby *n* new bugs are detected in the i-th cycle of the testing process, provided that *m* tests give incorrect results in this cycle.

Assuming that in the case of each test that gives an incorrect result, a detection of error that it causes will occur

and that each test can detect at most one error, we can write that there is:

$$p_{00} = p_{11} = 1,$$
$$p_{nm} = 0 \text{ for } n=0 \text{ and } m>0 \text{ or for } n > m. \quad (2)$$

Taking into account the properties of elements $p^i_{n,m}$, described by the relations (2), the matrix P_i, $i \in \boldsymbol{I}$, can be presented in the following form

$$P_i = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & ... \\ 0 & 1 & p^i_{12} & p^i_{13} & p^i_{14} & ... \\ 0 & 0 & p^i_{22} & p^i_{23} & p^i_{24} & ... \\ 0 & 0 & 0 & p^i_{33} & p^i_{34} & .. \\ 0 & 0 & 0 & 0 & p^i_{44} & ... \\ & & ... & & & \end{bmatrix}. \quad (3)$$

The quantities $p^i_{n,m}$ are the values of probabilities and therefore:

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

Condition (4) means that the sum of elements in each column of the P_i matrix is equal to 1.

For the determined method of designing a set of test data sets, the values of probabilities $p^i_{n,m}$ that form the matrix P_i, $i \in \boldsymbol{I}$, depend on the logical structure of the source code of the tested program and on the level of its reliability. Assuming that the structure of program under the testing processs is a program control graph, it can be concluded that the probabilities $p^i_{n,m}$ depend primarily on:

- the number of logical paths connecting the input node with the output node;
- the length of these paths, measured for example by the number of program instructions, in a case of their activation;
- the level of mutual overlapping individual logic paths, the measure of which is the number of program instructions included in two or more paths.

For example, if in the tested program each instruction (except of instructions forming input and output nodes) belongs to only one logical path, i.e. that the program under testing consists of a set of disjoint paths, then it should be expected that values of probability $p^i_{n,m}$, $n \le m$ will be focused on the main diagonal of the P_i matrix or near it (above the main diagonal). In turn, if in the tested program individual roads overlap, i.e. many logical paths "pass" through the same instructions or their groups, then it is expected that values of probability $p^i_{n,m}$, $n \le m$, will be concentrated in the initial rows of the P_i matrix (subject to conditions (2) - (4)).

For the above reasons, in further considerations the P_i matrix will be called the characteristic matrix of the i-th module of the program being tested. In a such case the tested program will be characterized by a vector of characteristic matrices P, defined as follows:

$$P = (P_1, P_2, ..., P_i, ..., P_I),$$

where P_i, $i \in \boldsymbol{I}$, is the characteristic matrix of the i-th module of the tested program.

3 Formulation of the problem of two-criteria optimization of the program testing strategy

In further considerations, the reliability coefficient of the tested program, after finishing the testing process, implemented according to strategy S, will be defined as follows [15]:

$$r(S,P) = \sum_{q \in \boldsymbol{Q}} d_q \prod_{i \in \boldsymbol{I}_q} r_i(S_i, P_i), \quad (5)$$

where:

$r_i(S_i, P_i)$ reliability coefficient of the i-th module with a characteristic matrix P_i, after finishing the process of its testing, carried out according to the strategy S_i, defined as follows [17]:

$$r_i(S_i, P_i) = 1 - (1 - r_i) A_i(S_i, P_i), \quad (6)$$

where

$$A_i(S_i, P_i) = \sum_{n_1=0}^{L_{i,1}} \sum_{n_2=0}^{L_{i,2}} ... \sum_{n_K=0}^{L_{i,K}} e^{-\alpha_i \sum_{k=1}^{K_i} n_k} \prod_{k=1}^{K_i} \sum_{m_k=0}^{L_{i,k}} p^i_{n_k m_k} A^i_{m_k}\left(\sum_{i=1}^{k-1} n_i, L_{i,k}\right) \quad (7)$$

while

$$A^i_{m_k}\left(\sum_{j=1}^{k-1} n_i, L_{j,k}\right) = \binom{L_{i,k}}{m_k} \left[e^{-\alpha_i \sum_{j=1}^{k-1} n_j} (1 - r_i)\right]^{m_k} \cdot \left[1 - e^{-\alpha_i \sum_{j=1}^{k-1} n_j} (1 - r_i)\right]^{L_{i,k} - m_k} \quad (8)$$

r_i value of the i-th module reliability coefficient before the testing process begins,

d_q the probability of activating the q-th logic path by a single test case.

According to relation (5), the reliability coefficient $r(S, P)$ of the tested program is understood as the probability of its correct execution for a single test case. The values of probabilities d_q, $q \in \boldsymbol{Q}$, depend on the nature and frequency of specific data sets for which the program under consideration is executed. Determining the probabilities d_q, $q \in \boldsymbol{Q}$, is implemented in practice with the methods used for the purpose

of determining the operational profile of the program. A description of methods useful in this area can be found in [15]. The values of probabilities d_q, $q \in \boldsymbol{Q}$, are in presented consideration an objective nature, i.e. independent of the software developer, because they characterize so-called operational profile of the program. The developer of the program can influence the values of the reliability coefficient *r(S, P)* of the program by shaping the values of reliability coefficients $r_i(S_i, P)$, $i \in \boldsymbol{I}$, of the program component modules.

Let *C(S, P)* be the cost of the program testing process with the characteristic matrix vector *P*, implemented according to strategy *S*. Cost *C(S, P)*, which is the sum of the costs of testing the program modules, can be determined as follows:

$$C(S,P) = \sum_{i=1}^{I} [C_T^i(S_i) + C_E^i(S_i, P_i)] , \tag{9}$$

where:

$C_T^i(S_i)$ the average cost of preparing a test case set used in the testing process of the i-th module, implemented according to the testing strategy S_i:

$$C_T^i(S_i) = L_i(S_i)\, C_T^i , \tag{10}$$

where c_T^i is the average cost of preparing and running one test case in the i-th module testing process, while $L_i(S_i)$ it is the total number of tests used in the i-th module testing process, carried out according to the strategy S_i, i.e.:

$$L_i(S_i) = \sum_{k=1}^{K_i} L_{i,k} , \tag{11}$$

$C_E^i(S_i, P_i)$ the average cost of locating and removing errors detected in the i-th module testing process with the characteristic matrix P_i implemented according to the strategy S_i:

$$C_E^i(S_i, P_i) = C_E^i\, E[N_i(S_i, P_i)] , \tag{12}$$

where C_E^i is the average cost of locating and removing one error in the i-th module testing process, while $E[N_i(S_i, P_i)]$ means the expected value of the number of errors, the detection of which is expected in the i-th module testing process with the characteristic matrix P_i, implemented according to the strategy S_i.

Quantity $E[N_i(S_i, P_i)]$, according to the relationship (6), can be specified as follows:

$$E[N_i(S_i,P_i)] = \sum_{k=1}^{K_i} \sum_{n_1=0}^{L_{i,1}} \sum_{n_2=0}^{L_{i,2}} \cdots \sum_{n_K=0}^{L_{i,K}} n_k \prod_{l=1}^{K_i} \sum_{m_l=0}^{L_l} p_{n_l m_l}^i A_{m_l}^i \Big(\sum_{j=1}^{l-1} n_j, L_{j,l} \Big) \tag{13}$$

For practical reasons, the following limitations are imposed on the reliability coefficient *r(S, P)* and cost *C(S, P)*:

$$\begin{aligned} r(S,P) &\geq r_{MIN} , \\ C(S,P) &\leq C_{MAX} , \end{aligned} \tag{14}$$

where quantities r_{MIN}, C_{MAX} mean the minimum acceptable level of reliability of the tested program and the maximum acceptable level of costs incurred for testing, respectively.

The following two-criteria optimization problem of the program testing strategy can be formulated based on the introduced designations and the obtained relationships:

$$(\boldsymbol{S},\ F,\ \Phi) , \tag{15}$$

where:

$\boldsymbol{S}$ a set of acceptable solutions (strategies), defined as follows:

$\boldsymbol{S} = \{ S = (S_1, S_2, ..., S_i, ..., S_I) : S_i$ is defined by dependence (1) and constraints that are met (14)$\}$,

F quality of solution indicator:

$$F(S,P) = (\, r(S,P),\ C(S,P)\,) , \tag{16}$$

while quantities *r(S,P)*, *C(S,P)* are defined by (5) and (9) respectively,

Φ the dominance relationship defined as follows:

$$\Phi = \{\, y_1, y_2 \,) \in \boldsymbol{Y} \times \boldsymbol{Y} :\ y_1^1 \geq y_2^1,\ y_1^2 \leq y_2^2 \,\} , \tag{17}$$

$$y_1 = (y_1^1, y_1^2),\ y_2 = (y_2^1, y_2^2) ,$$

where $\boldsymbol{Y}$ is so-called criterion space defined as below:

$$\boldsymbol{Y} = \{\, y = (\, r(S,P),\ C(S,P)\,) :\ S \in \boldsymbol{S} \,\} . \tag{18}$$

Problem (15) is a non-linear integer programming two-criteria optimization problem. Its solution can be determined in accordance with the generally accepted methodology of solving polyoptimization problems [3]. According to this methodology, the solution of the problem (15) may consist in particular in determining:

- a set of dominated strategies,
- a set of non-dominated strategies,
- a set of compromise strategies.

Due to the nature of dependencies (5) and (9), which determine the components of the criterion functions of the problem in question, one should expect that the set of dominated solutions will be empty. In this situation, the practical significance is therefore the determining a set of non-

dominated solutions and its possible narrowing, e.g. by defining a set of compromise solutions.

4 Numerical example

To illustrate the considerations presented in previous sections, a simple numerical example will be considered.

Let the graph of the tested program, characterizing its modular structure, be determined as in Fig. 1.

Assuming that all paths connecting the beginning node $i_{WE}=1$ with the end node $i_{WY}=7$ are logical paths we obtain

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

where the sets of module numbers that make up particular logical paths are defined as follows:

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

The values of probabilities of activating individual logical paths by individual test case are specified in Table 1. The remaining numerical data for which the calculations will be carried out are contained in Table 2.

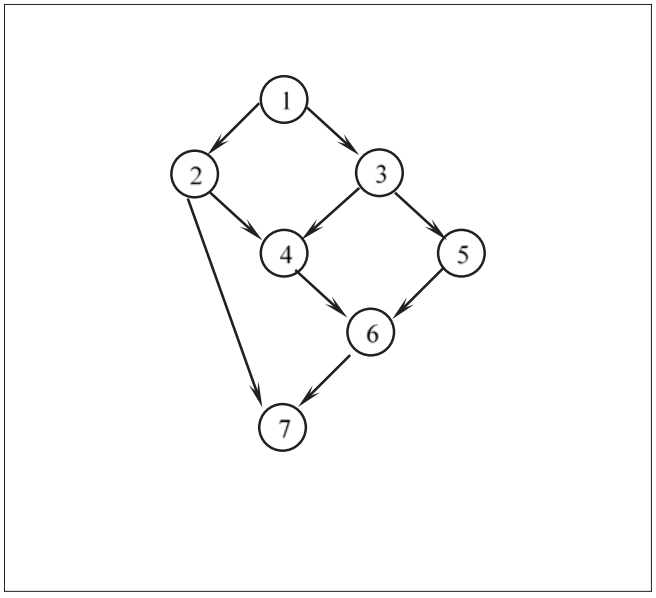

Fig. 1. Graph of the example program

The calculations will be carried out for the test scheme, in which each cycle of testing a single module involves performing only one test, followed by evaluation of test results and localization and removal of errors, if the test showed their occurrence. According to the determination of the testing strategy (1), this means that it occurs $L_{i,k}=1,\ k=\overline{1,L_{i,K_i}},\ i=\overline{1,7}.$

In the considerated case the testing strategy of the program can be presented in the following form

$$S = (S_1, S_2, S_3, S_4, S_5, S_6, S_7),$$

where

$$S_i = K_i,\ i=\overline{1,7},$$

while K_i means the number of cycles of the i-th module testing process, within each of which – in accordance with the previously assumed assumption – exactly one test is performed.

A direct consequence of the assumed testing scheme is the inability to detect in the subsequent cycles the testing process of individual modules of the so-called repeated errors, i.e. errors detected by different tests. According to the definition (3), the characteristic matrices of individual modules have the form:

$$P_i(S) = P_i = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad i=\overline{1,7}. \tag{19}$$

In the considerated case, expressions (5) - (6), defining the form of program reliability coefficient after the end of the test process, implemented according to strategy S, are simplified and take the form of

$$r(S) = \sum_{q\in\boldsymbol{Q}} d_q \prod_{i\in\boldsymbol{I}_q} r_i(S_i), \tag{20}$$

while quantities $r_i(S_i)$ are dedined as follows [17]:

$$r_i(S_i) = 1-(1-r_i)\sum_{n_1=0}^{1} e^{-\alpha_i n_1}(1-r_i)^{n_1} r_i^{1-n_1}$$
$$\sum_{n_2=0}^{1} e^{-\alpha_i n_2}[e^{-\alpha_i n_1}(1-r_i)]^{n_2}[1-e^{-\alpha_i n_1}(1-r_i)]^{1-n_2}\cdots$$
$$\cdots \sum_{n_{Ki}=0}^{1} e^{-\alpha_i n_{K_i}}[e^{-\alpha_i \sum_{m=1}^{K_i-1} n_m}(1-r_i)]^{n_{K_i}}[1-e^{-\alpha_i \sum_{m=1}^{K_i-1} n_m}(1-r_i)]^{1-n_{K_i}} \tag{21}$$

Table 1. Probability values for the activation of logic paths (set 1)

q	1	2	3	4	5	6
d_q	0.15	0.20	0.25	0.10	0.20	0.10

Table 2. Numeric data characterizing the modules of the program

i	1	2	3	4	5	6	7
r_i	0.995	0.99	0.99	0.985	0.97	0.99	0.995
α_i	1.5	10.0	2.0	3.5	15.0	6.0	2.5
C_T^i	35.0	27.0	30.0	27.5	25.0	26.0	35.0
C_E^i	50.0	43.0	50.0	45.0	35.0	42.0	55.0

The cost of the program testing process, according to dependence (9), is defined as follows:

$$C(S)=\sum_{i\in \boldsymbol{I}}[C_T^i(S_i)+C_E^i(S_i)]\,, \tag{22}$$

while

$$C_T^i(S_i)=K_i\cdot C_T^i\,, \tag{23}$$

$$C_E^i(S_i)=C_T^i\cdot E[N_i(S_i)]\,, \tag{24}$$

where quantity $E[N_i(S_i)]$, determining the expected value of number of errors, the detection of which is expected in the testing process of the i-th module, implemented according to the strategy S_i is determined as follows [16]:

$$E[N_i(S_i)]=(1-r_i)\sum_{k=1}^{K_i}\sum_{n_1=0}^{1}\sum_{n_2=0}^{1}\ldots\sum_{n_{k-1}=0}^{1}e^{-\alpha_i\sum_{i=1}^{k-1}n_i}\prod_{l=1}^{k-1}A_{n_l}(\sum_{i=1}^{l-1}n_i,1) \tag{25}$$

where

$$A_{n_l}(\sum_{i=1}^{l-1}n_i,1)=[e^{-\alpha_i\sum_{i=1}^{l-1}n_i}(1-r_i)]^{n_l}[1-e^{-\alpha_i\sum_{i=1}^{l-1}n_i}(1-r_i)]^{1-n_l}\,. \tag{26}$$

The values of coefficients (20) and (22), despite their relatively complex analytical form, can be very easily determined using computer technology.

The initial value of the reliability coefficient of the analyzed program, i.e. before the start of the testing process for data from Tables 1 and 2, is specified as follows:

$$r=\sum_{q\in \boldsymbol{Q}}d_q\prod_{i\in \boldsymbol{I}_q}r_i=0.9609.$$

In the considered example, it is assumed that the solution of the two-criteria optimization problem (15) consists in determining a set of non-dominated, i.e. strategies, solutions, i.e. the Pareto collection [3].

The polyoptimization problem (15) will be solved for a set of character constraints:

$$r_{MIN}\geq 0.9650,$$
$$C_{MAX}\leq 1500,$$
$$K_i\leq 13.$$

Last from above limits is technical one and serves only to reduce the size of the two-criteria optimization problem.
A solution of the task (15) (with the above restrictions) as the set of non-dominated strategies is presented in Table 3, wherein calculations were made for the data contained in Tables 1 and 2. This set was determined using the full review method, using computer technology. In the example under consideration, the set of non-dominated strategies contains 6 elements. According to dependence (1), each row in Table 3 defines the optimal testing strategy of the considered program, i.e. such strategy that maximizes the value of the reliability index and minimizes the cost of testing. The determined optimal values of these coefficients for individual non-dominated strategies are also presented in Table 3.

The value of the reliability coefficient of the program under consideration, determined by the dependence (20), obviously depends not only on the value of reliability coefficients of its component modules, but also on the so-called the operational profile of the program, i.e. from the probabilities of activating individual logical paths of the program by a single set of input data.

Table 3. Set of non-dominated solutions for $r_{MIN}=0.9650$ and $C_{MAX}=1500$

S_1	S_2	S_3	S_4	S_5	S_6	S_7	r(S)	C(S)
0	12	0	13	13	13	0	0,965026	1374,02
0	13	0	13	13	12	0	0,965032	1375,03
0	13	0	13	13	13	0	0,965078	1401,40
0	13	1	13	13	13	0	0,965112	1431,90
0	13	2	13	13	13	0	0,965145	1462,40
0	13	3	13	13	13	0	0,965178	1492,89

Table 4. Values of probabilities for activating logical paths (set 2)

q	1	2	3	4	5	6
d_q	0.30	0.05	0.20	0.30	0.10	0.05

Table 5. Set of non-dominated solutions for $r_{MIN}=0.9690$ and $C_{MAX}=1500$

S_1	S_2	S_3	S_4	S_5	S_6	S_7	r(S)	C(S)
0	13	8	13	12	7	0	0,969001	1461,26
0	13	9	13	13	5	0	0,969002	1464,66
0	13	9	13	12	6	0	0,969004	1465,33
0	13	3	13	13	12	0	0,969005	1466,52
0	13	10	13	12	5	0	0,969007	1469,40
0	13	10	13	11	6	0	0,969008	1470,04
0	13	4	13	13	11	0	0,969012	1470,63
0	13	5	13	13	10	0	0,969018	1474,73
0	13	6	13	13	9	0	0,969023	1478,82
0	13	6	13	12	10	0	0,969024	1479,48
0	13	7	13	13	8	0	0,969028	1482,91
0	13	8	13	13	7	0	0,969031	1486,99
0	13	8	13	12	8	0	0,969033	1487,65
0	13	9	13	13	6	0	0,969035	1491,06
0	13	9	13	12	7	0	0,969036	1491,73
0	13	10	13	13	5	0	0,969037	1495,12
0	13	10	13	11	7	0	0,969039	1496,44
0	13	4	13	13	12	0	0,969042	1497,00

By changing, for example, the values of probabilities d_q, $q=\overline{1,6}$, as shown in Table 4, with the remaining numerical data presented in Table 2 unchanged, it is obtained:

- initial value of the program reliability coefficient $r=0.9657$,
- a set of solutions for non-dominated tasks (15), with constraints:

$$r_{MIN} \geq 0.9690,$$
$$C_{MAX} \leq 1500,$$
$$K_i \leq 13,$$

presented in Table 5. In the present case the set of non-dominated strategies defined by particular rows of table 5 consists of 18 elements. Each strategy *S*, determined in accordance with (1), presented in this table has the property that for the corresponding reliability coefficient of the program *r(S)* is a strategy with a minimum cost *C(S)* (or respectively: for the corresponding cost of testing *C(S)* is a strategy with a maximum *r(S)* value). The values of constituent functions *r(S), C(S)* corresponding to individual non-dominated strategies, are also presented in Table 5. In the case when too high value of a cardinality number of non-dominated solutions set makes it difficult to decide on one of them, an additional criterion can be used, called in the polyoptimization theory the criterion of compromise (such a solution of the polyoptimisation problem is called a compromise solution) [3].

The analysis of the strategies oresented in Tables 3 and 5 shows that within the accepted limitations, including limitations on the total cost of the program testing process, optimal testing strategies prefer testing of these modules in the first place, the increase in the reliability coefficient which has the greatest impact on the value increase reliability coefficient of the entire program and those whose cost of testing is the lowest.

5 Summary

The final level of reliability of the created software product is formed during the implementation of all stages of the development process. Constantly increasing reliability requirements expected for modern software systems, especially systems with responsible software, enforce the need to improve the design and implementation methods used in all stages of the process. Despite the continuous development and improvement of these methods, their current level does not give full guarantee of creating a complex software product completely free of errors.

The software engineering practice shows that the biggest percentage of errors, among errors committed throughout the entire software production process, is made in its initial stages, i.e. as part of the requirements specification and design. For this reason, it is highly desirable to develop methods to verify the correctness of the results of the initial stages of the software development process, as a result of which there will be a significant increase in the share of verification methods in error detection, thus contributing to the reduction of errors detected at the testing stage. Such situation will create a real basis for a significant shortening of the software production cycle and a reduction of production costs.

The testing stage, creating great opportunities to verify and shape software reliability, significantly increases the cost of its production. The effectiveness of the work related to testing, expressed by the interdependence of the level of program product reliability and the cost of testing it, strongly depends on the adopted testing strategy, specifying the organization and scope of the work performed. In this situation, therefore, there is a need to define the conditions for a compromise in terms of reliability and cost requirements set for the software. The practical finding of this compromise can be greatly facilitated if there are possibilities to formally assess the level of software quality and the cost of testing it using appropriate indicators.

A method of determining a program testing strategy as a result of solving a two-criteria optimization problem, with the program reliability coefficient and the cost of testing as component criteria has been presented in the paper. The paper consists of description of the program testing process and mathematical model of this process, formulation of the problem of two-criteria optimization of the program testing strategy, remarks on method of solving the problem that has been formulated. proposed. To illustrate the method of finding an optimal testing strategy that has been proposed a numerical example has been considered.

References

1. P. Ammann, J. Offutt, *Introduction to Software Testing*, Cambridge University Press, 2017.
2. R.C. Cheung, *A user-oriented software reliability model*. IEEE Transactions on Software Engineering, Vol. SE-6, No 2 (1980)
3. H. Eschenauer, J. Koski, A. Osyczka, *Multicriteria design optimization: procedures and applications*. Springer-Verlag, Berlin (1990)
4. R. Gould, *Graph Theory*, Dover Publications, 2013.
5. K. Goseva-Popstojanova, K.S.Trivedi, *Architecture-based approach to reliability assessment of software systems*. Performance Evaluation, No 45 (2001)
6. M. Heusser, G. Kulkarni (editors), *How to Reduce the Cost of Software Testing*, CRC Press, 2016.
7. S.L. Ho, M. Xie, T.N Goh, *A Study of the Connectionist Models for Software Reliability Prediction*. Computer and Mathematics with Applications. Vol.46, No.10 (2003)
8. C.B. Jones, *Software Development: A Rigorous Approach*, Prentice Hall (1980)
9. S. Krishnamurthy, A.P. Mathur, *On the estimation of reliability of a software system using reliabilities of its components*, in: Proceedings of the Eighth International Symposium on Software Reliability Engineering (ISSRE'97) (1997)
10. P. Kubat, *Assessing reliability of modular software*, Operations Research Letter, No 8, 1989, 35–41

11. S. Mahsoodloo, D.B. Brown, C.J. Lin, *A Reliability & Cost Analysis of an Automatic Prototype Generator Test Paradigm*. IEEE Transactions on Reliability. Vol.41, No.4 (1992)
12. Y.K. Malaiya, M.N. Li, J.M. Bieman, R. Karcich, *Software Reliability Growth with Test Coverage*. IEEE Transactions on Reliability. Vol.51, No.4 (2002)
13. Y.K. Malaiya, N. Karunanithi, P. Verma, *Predictability of Software-Reliability Models*. IEEE Transactions on Reliability. Vol.41, No.4 (1992)
14. A. Mili, F. Tchier, *Software Testing: Concepts and Operations*, Wiley, 2015.
15. J.D. Pfefferman, B. Cernuschi-Frias, *A No-Parametric Non-Stationary Procedure for Failure Prediction*. IEEE Transactions on Reliability Vol.51, No.4 (2002)
16. R. Waszkowski, M. Kiedrowicz, T. Nowicki, Z. Wesołowski, K. Worwa, *Method for assessing software reliability of the document management system using the RFID technology*. MATEC Web of Conferences Vol. 76, No 04009 (2016)
17. K. Worwa, *A discrete-time software reliability-growth model and its application for predicting the number of errors encountered during program testing*. Control and Cybernetics, Vol. 34., No 2 (2005)