Symulacja w Badaniach i Rozwoju
Vol. 2, No. 2/2011

Kazimierz WORWA
Wojskowa Akademia Techniczna, 00-908 Warszawa, ul. Kaliskiego 2
E-mail: kworwa@wat.edu.pl

Using an optimization methods to selection the best software developer

1 Introduction

As software becomes increasingly important in systems that perform complex and critical functions, e.g. military defense, nuclear reactors, so too have the risks and impacts of software-caused failures. There is now general agreement on the need to increase software reliability and quality by eliminating errors created during software development. Industry and academic institutions have responded to this need by improving developmental methods in the technology known as software engineering and by employing systematic checks to detect software errors during and in parallel with the developmental process.

A large number of analytical models have been proposed and studied over the last two decades for assessing the quality of a software system. Each model must make some assumptions about the development process and test environment. The environment can change depending on the software application, the lifecycle development process as well as the capabilities of the engineering design team [5]. Therefore, it is important for software users and practitioners to be familiar with all the relevant models in order to make informed decisions about the quality of any software product.

Let us define the terms such as software error, fault and failure [1]. An error is a mental mistake made by the programmer or designer. A fault is the manifestation of that error in the code. A software failure is defined as the occurrence of an incorrect output as a result of an input value that is received with respect to the specification.

What precisely do we mean by the term failure? It is the departure of the external results of system operation from user needs. So failure is something dynamic. A system has to be operating for a failure to occur. The term failure relates to the behavior of the system. Note that a failure is not the same thing as a bug or, more properly, fault. The very general definition of failure is deliberate. It can include such things as deficiency in performance attributes and excessive response time when desired, although there can be disadvantages in defining failure too generally.

A fault in software is the defect in the program that, when executed under particular conditions, causes a failure. There can be different sets of conditions that cause failures. Hence a fault can be the source of more than one failure. A fault is a property of the program rather than a property of its execution or behavior. It is what we are really referring to in general when we use the term defect or bug.

Software reliability modelling has become one of the most important aspects in software reliability engineering since first software reliability models 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 [2, 4-8, 10]. Although this assumption is appropriate for a wide scope of software systems, there are many systems, which are essentially different 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. [4-8, 10]). They usually express reliability as the probability that an execution of the software is successful.

In spite of permanent improvement of software project and implementation methods which are used in a practice of a software development they still can not guarantee of developing a complicated software system entirely free of errors. These errors appeared during useful exploitation of software and they cause to arise some financial losses and other difficulties. In order to minimize of scale such difficulties software users demand from developers such reliable software as possible. Unfortunately, more reliable software is more expensive so there is a practical problem to determine some compromise on both software quality and cost requirements. In practice, determining such a compromise can be easier if there are possibilities to estimate both software reliability level and his development cost by means of appropriate measures.

The purpose of this chapter is to propose some formal way of determining software developer by formulating and solving the bicriterial optimization problem will both minimize the value of the number of software tasks which have incorrect realization during some time period and minimize the value of the software development cost. The method of determining the best software developer that is proposed will be illustrated by the simple numerical example.

2 Mathematical model of the software exploitation process

We will consider some useful software that services arriving tasks. Each input task involve a sequence of software operations, whereas an operation is a minimum execution unit of software. The concrete sense of an operation is subject to application context. For example, an operation can correspond to execution of a test case, of a program path, etc. Some input data set is connected with every operation and operation consists in executing of the software with that data set. An operation can be viewed as a transformation of an input state to an output state. One recognizes the possibility of a software failure by noting a discrepancy between the actual value of a variable occurring during an operation and the value of that variable expected by users.

We can view the execution of a program as a single entity, lasting for months or even years for real-time systems. However, it is easier to characterize the execution if you divide it into a set of operations. Then the environment is specified by the operational profile, where operational profile is a complete set of operations with their probabilities of occurrence. Probability of occurrence refers to probability among all invocations of all operations. In turn, there are many possible instances of operations, each called a run. During execution, the factors that cause a particular operation and a particular run within that operation to occur are very numerous and complex. Hence, we can view the operations required of the program as being selected randomly in accordance with the probabilities just mentioned. The runs within those operations also occur randomly with various probabilities.

A run is specified with respect to the operation of which it is a part by its input state or set of values for its input variables. Input variables are variables that exist external to an operation and influence its execution. The input state is not the same thing as the machine state, which is the much larger set of all variable values accessible to the computer. The machine state also includes variables that do not affect a run and are not set by it. Recurrent runs have the same input state. We judge the reliability of a program by the output states (sets of values of output variables created) of its runs. Note that a run represents a transformation between an input state and an output state. Multiple input states may map to the same output state, but a given input state can have only one output state. The input state uniquely determines the particular instructions that will be executed and the values of their operands. Thus, it establishes the path of control taken through the program. It also uniquely establishes the values of all intermediate variables. Whether a particular fault will cause a failure for a specific run type is predictable in theory. However, the analysis required to determine this is impractical to pursue.

It is noteworthy that such a way of software working is typical for greater part of software which are exploited in practice. As typical examples can be use reservations programs, inquiry programs, storage economy programs, registration and record programs and many others.

Both the human error process that introduces faults into software code and the run selection process that determines which code is being executed at any time and under what conditions, and hence which faults will be stimulated to produce failures, are dependent on an enormous number of time-varying variables. Hence, researchers have generally formulated software reliability models as random processes in time. The models are distinguished from each other in general terms by the probability distribution of failure times or number of failures experienced and by the nature of the variation of the random process with time. A software reliability model specifies the general form of the dependence of the failure process on the factors mentioned. You then particularize it by estimating its parameters.

It is assumed that tasks arrive to the software in a random way and time intervals τ_k k=1,2,…, between succeeding tasks are independent random values with the same distribution function $G(t)$, $t\geq 0$.

Let $L(t)$ mean a number of tasks which arrive to the software at time *[0, t]*. For $t\geq 0$ value $L(t)$ is a random variable, $L(t)\in\{1,2,3,\ldots\}$. The process $\{L(t),\ t\geq 0\}$ is a some stochastic process with continuous time parameter and countable infinite set of states.

A result of the growing maturity of software development is that it is no longer adequate that software simply works; it must now meet other customer-defined criteria. Software development is very competitive, and there are many competent software developers, spread throughout the world.

Surveys of users of software-based systems generally indicate users rate on the average the most important quality characteristics as: reliability, rapid delivery, and low cost (in that order). In a particular situation, any of these major quality characteristics may predominate. The quality of the software system usually depends on how much time development (especially testing) takes and what technologies are used. On the one hand, the more time people spend on development and testing, the more errors can be removed, which leads to more reliable software; however, the testing cost of the software will also increase. On the other hand, if the testing time is too short, the cost of the software could be reduced, but the customers may take a higher risk of buying unreliable software [7]. This will also increase the cost during the operational phase, since it is much more expensive to fix an error during the operational phase than during the testing phase.

When choosing a software developer the user is interested in a software quality (measured for example by a failure intensity), a development time and a development cost. For the purpose of this chapter a software developer will be characterized by a probability of correct realization by the software of single task, i.e. probability of correct execution of the software with single input data set. Let q mean this probability. Respectively, let p=1–q mean a probability of incorrect realization of single task, whereas realization of single task is incorrect if some errors appear during execution of the software with data set connected with that task. It is obvious that probabilities q and p=1–q characterize software developer. In general, value of q depends on the development methods and technologies, the lifecycle development process as well as the capabilities of the engineering design team.

Let $N(t)$ mean a number of tasks which have incorrect realization during time period $[0, t]$, i.e. during service of which some software errors appear. If we assume that none task which arrives to the software will be lost (system with an infinitive queue) value $N(t)$ can be determined as follows

$$N(t)=\sum_{n=0}^{L(t)} X_n\ , \tag{1}$$

where values X_n, $n=1,2,...,L(t)$ are random variables with the same zero-one distributions

$$P(X_n=1)=p,\quad P(X_n=0)=q\ .$$

For a some natural number m random variable $N(t)$ has following binomial distribution

$$P(N(t)=n_{|L(t)=m})=\begin{cases}\binom{m}{n}p^n q^{m-n} & for\ \ m\ge n\\ 0 & for\ m<n\end{cases}, \tag{2}$$

where n=0,1,2,... .

Taking into account (1) and (2) we can determine probability distribution function of random variable $N(t)$ as a border distribution of two-dimentional random variable ($N(t)$, $L(t)$)

$$P(N(t)=n)=\sum_{m=0}^{\infty}P(N(t)=n_{|L(t)=m})\cdot P(L(t)=m),\quad n=0,1,2,... \tag{3}$$

Considering prior assumptions probability $P(L(t)=m)$ can be determined as follows [3]

$$P(L(t)=m)=G_m(t)-G_{m+1}(t),\qquad m=0,1,2,... \tag{4}$$

where $G_m(t)$, $m \geq 1$, means distribution function of random variable t_m, that has the form

$$t_m = \sum_{k=1}^{m} \tau_k \tag{5}$$

It is assumed that $G_0(t) = 1$ and $G_1(t) = G(t)$.

Then, taking into account (2) and (4), we can determine probability distribution function of random variable *N(t)* as follows

$$P(N(t) = n) = \binom{m}{n} p^n q^{m-n} \, [\, G_m(t) - G_{m+1}(t)] \, , \, n=0,1,2,... \tag{6}$$

A mean value of number of incorrect serviced tasks at time period *[0, t]* has a following form

$$E[\, N(t)] = \sum_{n=0}^{m} n \cdot P(\, N(t) = n\,) \tag{7}$$

where probability $P(\, N(t) = n\,)$ is stated by (6).

For example, if $G(\, t\,) = 1 - e^{-\lambda t}$, i.e. when random variables τ_k, $k = 1,2,...$, have the same exponential distribution with λ parameter, we will obtain

$$P(N(t) = n) = \sum_{m=n}^{\infty} \binom{m}{n} p^n q^{m-n} \frac{(\lambda t)^m}{m!} e^{-\lambda t} \, , \quad n=0,\, 1,\, 2,\, ...$$

from where, after simple transformation, we have

$$P(N(t) = n) = (p\lambda t)^n \frac{e^{-\lambda p t}}{n!} \, , \quad n=0,1,2 \, . \tag{8}$$

Expression to calculate a mean value of random variable *N(t)* has in this case the form

$$E[\, N(t)] = p\lambda t \, . \tag{9}$$

Using (6) we can determine probability of event which consists in no errors will appear during useful exploitation of the software at time period *[0, t]*. This probability can be obtained from (6) by taking *n=0*. Both mentioned probability and mean value of number of incorrect tasks at time *[0, t]* in expression (7) can be used as reliability measures of the software under investigation. It is worth to add that these measures - unlike most reliability measures which can be found in subject literature, e.g. in [2, 6-7, 8] take into account not only reliability level of the software but circumstances of its exploitation as well.

3 Formulation of a bicriterial optimization problem of choice of the software developer

Mean value of number of incorrect serviced tasks during useful exploitation of the software at a given time period, appointed by (7), can be used as one of criteria (quality criterion) to determine a compromise which reconciles requirements of user of a software regarding its both reliability level maximization and production cost minimization.

A practical problem of choice of the best software developer by a potential user of that software will be considered. The user can choose one of some number possible developers of the software taking into account characterization of both software exploitation circumstances and technology of software development.

Let $\boldsymbol{I}$ mean set of numbers all possible software developers

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

From software user's viewpoint the i-th developer will be characterized by pair of numbers (q_i, k_i), where q_i means probability of a correct execution of the software with a single input data set if the software is developed by the i-th developer, and k_i means cost of the software development by the i-th developer.

Because individual developers use different project and implementation methods, have a different computer equipment, use different organization and management styles and so on, characteristics q_i, k_i can be different for every developer.

The software exploitation circumstances, i.e. circumstances in which it will be used, will be characterized by means of function $G(t)$ which is distribution function of time intervals between succeeding demands of use the software. We still assume that time intervals are independent random variables with the same distribution.

A choice which is made by the user will be characterized by means following zero-one vector

$$X = (x_1, x_2, ..., x_i, .., x_I)$$

where

$$x_i = \begin{cases} 1 & \text{if the software is developed by the i - th developer} \\ 0 & \text{if not} \end{cases} , \tag{10}$$

while

$$\sum_{i=1}^{I} x_i = 1 \ . \tag{11}$$

Expression (11) ensures that software is developed by exactly one developer .

Let $N_t(X)$ mean software reliability coefficient for choice X, that is interpreted as a mean value of incorrect serviced tasks arriving to the software at time period $[0, t]$.

According to (7) coefficient $N_t(X)$ can be determined as follows

$$N_t(X) = \sum_{i=1}^{I} x_i \cdot E[N(t)] \ , \tag{12}$$

where value $E[N(t)]$ is stated by (7) with substitutions p_i, q_i instead p and q respectively.

Let $K(X)$ denote a development cost of the software for choice X. This cost can be determined as follows

$$K(X) = \sum_{i=1}^{I} x_i \cdot k_i \ , \tag{13}$$

where k_i means cost of the software developed by the i-th developer.

On the base prior assumptions and expressions that were obtained we can formulate following bicriterial optimization problem of choice of software developer

$$(\boldsymbol{X}, F, R), \tag{8}$$

where:

X is a feasible solution set which is defined as follows

$\boldsymbol{X} = \{X = (x_1, x_2, ..., x_i, ..., x_I) : X \text{ complies with constraints } (10),(11)\}$;

F is a vector of quality coefficient which has form

$$F(X) = (N_t(X),\ K(X)),$$

where values $N_t(X)$, $K(X)$ are determined by expressions (12) and (13) respectively;

R is a so-called domination relation which is defined as follows

$R = \{(y_1, y_2) \in Y \times Y :\ y_1^1 \le y_2^1,\ y_1^2 \le y_2^2\},\ y_1 = (y_1^1, y_1^2),\ y_2 = (y_2^1, y_2^2)$,

whereas Y is so-called objective function space

$$Y = F(\boldsymbol{X}) = \{y = (N_t(X),\ K(X)) :\ X \in \boldsymbol{X}\}.$$

The problem (14) is a bicriteria optimization problem with linear objective functions and linear constraints. A solution of this problem can be obtained by using well known methodology of solving multiple optimization problems [9]. According to that methodology as a solution of the problem (14) we can determine:

- a dominate solution set,
- a nondominate solution set,
- a compromise solution set.

Taking into account that values of objective functions $N_t(X)$ and $K(X)$ are inverse (in sense that if a value $N_t(X)$ is decreased, the value $K(X)$ is increased) it is reasonable to expect that the dominate solution set will be empty. In a such situation practically recommended approach is to determine a nondominate solution set. If this set is very numerous we can narrow it down by determining a so-called compromise solution, i.e. such solution belongs to the nondominate solution set that is a nearest (in sense Euclidean distance) to the so-called an ideal point [9].

The best software developer that to be chosen after solving the optimization problem (14) will both minimize the value of the number of software tasks $N_t(X)$ which have incorrect realization during time period *[0, t]* and minimize the value of the software development cost $K(X)$.

4 Numerical example

In order to illustrate of considerations that were described some simple numerical example will be presented.

Let set of numbers of potential software developers have form $\boldsymbol{I}$={1,2,3,4,5} and quality-cost characterization of potential developers be as at the table 1.

Tab. 1. Quality-cost characterization of developers

i	*1*	2	*3*	*4*	*5*
q_i	0,50	0,80	0,85	0,90	0,95
$k_i \cdot 10^3$	5	6	6,5	9,5	12

For the quality-cost parameters that characterize of potential developers from Tab. 1 we will solve the bicriterial optimization problem (14).

If we assume that time intervals between task arrivals to the software have exponential distribution $G(t)=1-e^{-\lambda t}$, $t \geq 0$, we will obtain

$$N_t(X)=\lambda t\sum_{i=1}^{5} x_i p_i .$$

According to numerical values q_i, k_i, $i \in \boldsymbol{I}$, , that were assumed the feasible solution set has a form $\boldsymbol{X}$={(1,0,0,0,0), (0,1,0,0,0), (0,0,1,0,0), (0,0,0,1,0), (0,0,0,0,1)}.

Tab. 2. Values of the objective function space

i	*1*	*2*	*3*	*4*	*5*
$N_t(X)$	105,00	42,00	31,50	21,00	10,50
$K(X)$	5000,00	6000,00	6500,00	9500,00	12000,00

A feasible solution set for this problem is a five-elements set of zero-one vectors. We assume that λ=0,21 h^{-1} and $t=1000$ h. Objective function values for the feasible solution set are presented at Tab. 2. It is easy to check that the dominate solution set of

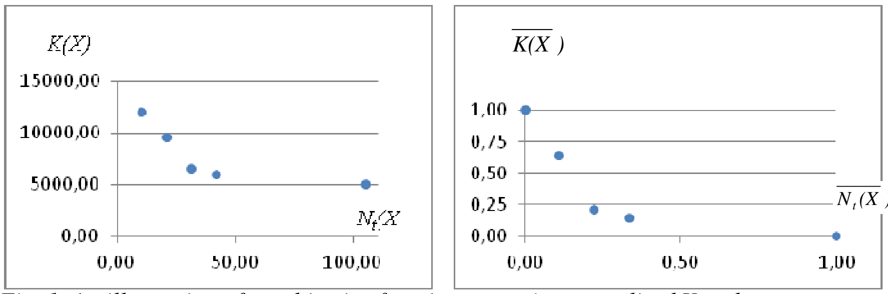

Fig. 1. An illustration of an objective function space (unnormalized Y and normalized Y)

the problem (14) is empty (it is because of objective functions $N_t(X)$ and $K(X)$ are inverse in sense their values). In that case practically recommended approach is to determine a nondominate solution set. It can be easy proved that above set $\boldsymbol{X}$ is a nondominated solution set, i.e. every vector from $\boldsymbol{X}$ is a nondominated solution of the bicriterial optimization problem (14). According to methodology of solving of multiple objective decision problems in such a case we can find a compromise solution with reference to some measure of a distance between so-called ideal point and particular points of the objective function space Y [9].

Co-ordinates of the ideal point $y^* = (y_1^*, y_2^*) \in Y$ are defined as follows:

$$y_1^* = \min_{X \in \boldsymbol{X}} N_t(X),$$

$$y_2^* = \min_{X \in \boldsymbol{X}} K(X).$$

In accordance with assumed values of quality-cost parameters we will have $y_1^* = 10{,}5$ and $y_2^* = 5000$.

In order to narrow down nondominate solution set we will determine a compromise solution of this problem, i.e. such solution belongs to the nondominate solution set that is a nearest (in sense Euclidean distance) to the so-called an ideal point [9]. For this reason both objective functions $N_t(X)$ and $K(X)$ determined by (12) and (13) respectively, will be normalized by means of the following formulae [9]:

$$\overline{N_t(X)} = \frac{N_t(X) - N_t^{min}(X)}{N_t^{max}(X) - N_t^{min}(X)}, \qquad \overline{K(X)} = \frac{K(X) - K_{min}(X)}{K_{max}(X) - K_{min}(X)}, \qquad (15)$$

where

$$N_t^{min}(X) = \min_{X \in \boldsymbol{X}} N_t(X), \qquad N_t^{max}(X) = \max_{X \in \boldsymbol{X}} N_t(X) \qquad (16)$$

$$K_{min}(X) = \min_{X \in \boldsymbol{X}} K(X), \quad , \; K_{max}(X) = \max_{X \in \boldsymbol{X}} K(X) \qquad (17)$$

where $\boldsymbol{X}$ is a feasible solution set.

Tab. 3. Values of the normalized objective function space

i	*1*	*2*	*3*	*4*	*5*
$\overline{N_t(X)}$	*1,00*	*0,33*	*0,22*	*0,11*	*0,00*
$\overline{K(X)}$	*0,00*	*0,14*	*0,21*	*0,64*	*1,00*

As a result of the normalization both normalized objective functions $\overline{N_t(X)}$ and $\overline{K(X)}$ have values belong to range *[0, 1]*.

It is easy to notice that in the example that is considered the ideal point $(\overline{N_t(X)}, \overline{K(X)})$ is of the form $(N_t(X), K(X)) = (0,0)$.

The searched compromise solution $X^\circ \in \boldsymbol{X}$ can be determined as follows

$$X^\circ = \overline{F}^{-1}(\overline{y}^\circ),$$

where point $\overline{y}^\circ = (\overline{y}_1^\circ, \overline{y}_2^\circ) \in \overline{Y}$ minimizes the above norm, i .e .

$$\left\| \overline{y}^* - \overline{y}^\circ \right\| = \min_{\overline{y} \in \overline{Y}} \left\| \overline{y}^* - \overline{y} \right\|.$$

According to numerical values which were assumed we have $y^\circ = (0.22, 0.21)$ and respectively $X^\circ = (0,0,1,0,0)$ and it means that an optimal variant of the software development is for *i=3*.

Table 2 presents the results of solving the optimization problem (14). The row (in bold) that corresponds to the vector is the compromise solution of this problem, i.e. this is a such vector that is a nearest to the ideal point *(0, 0)*, where a distance function $d[(\overline{N_t(X)}, \overline{K(X)}),(0,0)]$ is of the form

$$d[(\overline{N_t(X)}, \overline{K(X)}),(0,0)] = \sqrt{[\overline{N_t(X)}]^2 + [\overline{K(X)}]^2} \quad . \tag{18}$$

It is easy to notice that the vector $X^\circ = (0,0,1,0,0)$ complies following condition

$$d[(\overline{N_t(X^\circ)}, \overline{K(X^\circ)}),(0,0)] = \min_{X \in \mathbf{X}} d[(\overline{N_t(X)}, \overline{K(X)}),(0,0)] \tag{19}$$

Tab. 4. Compromise solution of the optimization problem (14)

i	X_1	X_2	X_3	X_4	X_5	$N_t(X)$	$\overline{N_t(X)}$	$K(X)$	$\overline{K(X)}$	$d[(\overline{N_t(X)},\overline{K(X)}),(0,0)]$
1	1	0	0	0	0	105,0	1,00	5000,0	0,00	1,00
2	0	1	0	0	0	42,0	0,33	6000,0	0,14	0,36
3	**0**	**0**	**1**	**0**	**0**	**31,5**	**0,22**	**6500,0**	**0,21**	**0,31**
4	0	0	0	1	0	21,0	0,11	9500,0	0,64	0,65
5	0	0	0	0	1	10,5	0,00	12000,0	1,00	1,00

5 Concluding remarks

The chapter proposes some formal way of choosing a software developer by formulating and solving the bicriterial optimization problem will both minimize the value of the number of software tasks which have incorrect realization during some time period and minimize the value of the software development cost.

An interpretation of terms "task" and "service" makes possibility to refer considerations that were presented to almost every useful software, including so-called real time software.

Practical usage of the method of determination of an optimal choice of a software developer which was presented is possible if values q_i, k_i and distribution functions $G_i(t)$, $i \in \mathbf{I}$ are known. These values can be estimated by means of methods that are developed on the base of software reliability theory (see e.g. [4, 6-8]).

Set of constraints which is used in bicriterial optimization problem (14) can be changed according to current needs. In particular, that set can be completed by such constraints which would guarantee that values of both software development cost and mean value of tasks that are serviced incorrect would keep within the feasible values.

The method of choosing a software developer that has been proposed was illustrated by the simple numerical example. For assumed values of quality-cost characteristics of developers the best producer has been obtained as a compromise solution of the bicriteria optimization problem (14).

References

1. *IEEE Standard Glossary of Software Engineering Terminology.* IEEE Standard 610.12, 1990.
2. Konopacki G., Worwa K.: *Uogólnienie modeli niezawodności oprogramowania Shoomana i Jelinskiego-Morandy.* Biuletyn WAT, Nr 12, 1984.

3. Koźniewska I., Włodarczyk M.: *Modele odnowy, niezawodności i masowej obsługi.* PWN, Warszawa, 1978.
4. Lyu M.R.: *Handbook of Software Reliability Engineering.* McGraw-Hill, New York, 1996.
5. Malaiya Y.K., Srimani P.K.: *Software Reliability Models: Theoretical Developments, Evaluation and Applications.* IEEE Computer Society Press, Los Angeles, 1990.
6. Musa J.: *Software reliability engineering: more reliable software, faster and cheaper*. AuthorHouse, 2004.
7. Pham H.: *System software reliability*. Springer-Verlag London Limited, 2006.
8. Rykov V.V.: *Mathematical and Statistical Models and Methods in Reliability: Applications to Medicine, Finance, and Quality Control*. Springer, 2010.
9. Stadnicki J.: *Teoria i praktyka rozwiązywania zadań polioptymalizacji.* WNT, Warszawa, 2006.
10. Trechtenberg M.: *A general theory of software reliability modeling.* IEEE Transactions on Software Engineering. Vo1.39, No. 1, 1990.

Summary

A practical problem of choosing a software developer is considered. This problem is investigated from a user's viewpoint, i.e. it is assumed that the software which is needed should be not only reliable but as cheap as possible too. The purpose of the paper is to propose some formal way of determining software developer by formulating and solving the bicriterial optimization problem will both minimize the value of the number of software tasks which have incorrect realization during some time period and minimize the value of the software development cost. Some numerical example is presented to illustrate of practical usefulness of the method which is proposed. The exemplary bicriterial optimization problem is solved on the base of the general methodology of solving multicriteria optimization problems.

Wykorzystanie metod optymalizacji do wyznaczania najlepszego producenta oprogramowania

Streszczenie

W artykule przedstawiono propozycję pewnego wskaźnika jakości programu, w konstrukcji którego uwzględniono warunki jego użytkowej eksploatacji. Dla zilustrowania przydatności skonstruowanego wskaźnika jakości programu w dalszej części artykułu sformułowano dwukryterialne zadanie wyboru wariantu produkcji programu, z kosztem produkcji i proponowanym wskaźnikiem jakości programu jako kryteriami składowymi.