ISSN: 0975-8585

Research Journal of Pharmaceutical, Biological and Chemical Sciences

Models for Estimating the Reliability of the Software of an Onboard Control System.

Igor N. Kartsan*.

Reshetnev Siberian State University of Science and Technology, 660037, 31 pr. imeni gazety "Krasnoiarskii rabochii", Krasnoyarsk, Russia.

ABSTRACT

Remote sensing systems have found a wide application in many fields of modern science. These include many disciples in Earth sciences such as biology. Currently, a large area of southern Siberia forests (the taiga) is affected by the Siberian silk moth (Dendrolimus superans). This insect feeds on such species as Larix, Picea, and Pinus causing serious issues of defoliation in the regions of northern Asia. Satellite remote sensing systems, which incorporate constellations of navigation spacecraft are a useful tool in determining the affected areas and monitoring the environment of a given area. However there are certain issues in the software that these systems use, including its reliability, that require immediate attention from specialists in the field of electronics and software. This paper discusses a number of selected models for estimating the reliability of software systems and analyzes existing software cost-estimating models. The authors suggest an approach for utilizing the reliability of function points and efforts required for its execution in large-scale complex software development projects.

Keywords: software, spacecraft, ground control complex, estimating reliability, remote sensing.

**Corresponding author*

INTRODUCTION AND PROBLEM STATEMENT

Computer software has found the widest application in many areas, including remote sensing. However there are certain issues concerning the software of these systems that need to be solved. The reliability of software systems is, thus, dictated by the area of their exploitation. For example, the probability of faultless operation of an onboard control system of a space probe must not be less than 0.9999 [1]. Naturally, such features of reliability as faultless operation of software, the intensity of software malfunctions, and others should be assessed and estimated.

Existing software reliability estimating models enable the assessment of the qualitative characteristics of the software using test data.

All reliability models are classified as either analytical or empirical.

Analytical models enable measuring and assessing the qualitative characteristics of reliability based on the performance of the program during testing.

Empirical models are based on the analysis of the structural features of software. These models demonstrate the dependence of the reliability features from the number of intermodal relations, module cycles, etc. Such models belong to a classical scheme as they show the relation between the complexity of computer-aided data processing systems and their reliability. Empirical models are hence employed during the stage of software design when the potential system is broken into modules and the general structure of the system is visible.

METHODS AND FINDINGS

Let us evaluate the immunity ratio of signal to interference, which enable radio receivers to operate in conditions of a deficit of interference susceptibility with a probability of less than 0.9.

The time and probability characteristics of the failure of the signal tracking process for a radio receiver it determined using nonlinear methods for analyzing the statistical dynamics of the tracking systems; these methods are based on the Markov chain [2, 3, 4]. The application of these methods allows us to determine the algorithm for calculating the average time and time dispersion to the tracking failure simultaneously for both PLL and DTS considering coupling. Simultaneous signal tracking failure in DTS and PLL is understood as the first exit of the trajectory of the phase φ and delay τ_p beyond the limits of the discrimination characteristics.

Analytical models of reliability

Analytical models for estimating reliability can be either dynamic or static. In dynamic models failures occurring in the system are considered during a time period. Static models consider failures outside of a specific time period, focusing on the dependence of the number of identified errors from the number of test runs or the dependence of the number of identified errors from the features of the output domain.

In order to use the dynamic modules, it is necessary to register data on failures in time during the testing process. If the time intervals are registered between failures, a relation between the failure rate and time may be observed. It is also possible to register the number of failures for a random time interval.

Dynamic models of reliability

In the reliability assess model La Padula a random testing in m stages is performed. After each testing stage has been completed, the software is supplied with data on the changes concerning the correction of the detected errors. The increasing function of the reliability is calculated from the number of errors, which are identified during each stage of the test. The reliability of the program during the i-th stage is equal to:

$$R(i) = R(\infty) - \frac{A}{i},$$

$$R(i) = \lim_{i\to\infty} R(i),$$

where *R(∞)* is the marginal reliability of the program; A is the reliability growth parameter. These unknown variables can be found using the following system of equations:

$$\begin{cases} \sum_{i=1}^{m}\left(\frac{S_i - m_i}{S_i} - R(\infty) + \frac{A}{i}\right) = 0, \\ \sum_{i=1}^{m}\left[\left(\frac{S_i - m_i}{S_i} - R(\infty) + \frac{A}{i}\right)\cdot\left(\frac{1}{i}\right)\right] = 0. \end{cases}$$

where S_i is the number of test runs; mi is the number of failures during the i-th state of the test.

The exponent R(i), found using this model, is the reliability of the program's operation during the i-th stage of testing. This model is forecasting and enables to compute the probability of the program's flawless performance during further stages.

In 1972 Z. Jelinski and P.B. Moranda proposed a software reliability model which employs initial data that had been obtained during the testing of the program; the time intervals between the failures are registered. The model is based on the following assumptions:

- the time to the next failure is distributed exponentially;
- all errors are equally probable and their identification is arbitrary;
- new errors are not introduced into the program when correcting errors;
- the intensity of the failures is constant during the interval between adjacent time moments of error detecting;
- the intensity of failures is proportional to the number of potential errors:

$$\lambda(t_i) = C(N - (i-1)),$$

where *C* is the coefficient of proportionality; *N* is the number of errors that were initially present in the program; ti is the time interval between the detection of the (*i*-1)-th and *i*-th error; *i* is the number of errors, detected by the moment of time t_i.

As the time to the next failure is determined exponentially, the probability of the program's flawless performance, i.e. the probability of default of the *i*-th failure is

$$P(t_i) = e^{-\lambda(t_i)\cdot t_i},$$

then the probability density function is equal to:

$$f(t_i) = \lambda(t_i)\cdot e^{-\lambda(t_i)\cdot t_i} = C(N-(i-1))\cdot e^{-C(N-(i-1))\cdot t_i},$$

Estimating the known parameters of models N and C is performed using the maximum likelihood method of data that has been obtained during the testing. It is necessary to register the time of the program's performance to the next failure t1, t2, …, tn.

As the detection of all failures is equally probable, the likelihood function is

$$L(t_1, t_2, \ldots, t_n) = \prod_{i=1}^{n} f(t_i) = \prod_{i=1}^{n} C(N-(i-1))\cdot e^{-C(N-(i-1))\cdot t_i}.$$

If we take a natural logarithm from this expression we shall get the following equation:

$$\ln\left(L(t_1,t_2,...,t_n)\right)=\sum_{i=1}^{n}\left(\ln\left(C(N-(i-1))\right)-C(N-(i-1))\cdot t_i\right).$$

Having found the partial derivatives $\frac{\partial \ln L}{\partial C}$, $\frac{\partial \ln L}{\partial N}$ and having set them to zero, we shall get a system of equations for determining the value of the maximum likelihood for parameters *N* and *C*:

$$\begin{cases}\frac{\partial \ln L}{\partial C}=\sum_{i=1}^{n}\left(\frac{1}{C}-C(N-(i-1))\cdot t_i\right)=0,\\ \frac{\partial \ln L}{\partial N}=\sum_{i=1}^{n}\left(\frac{1}{N-(i-1)}-C\cdot t_i\right)=0.\end{cases}$$

The system of equations is solved after the unknown parameters *N* and *C* are replaced by the values of their estimates $\hat{N}$ and $\hat{C}$. Further, having substituted *N* and *C* in the equation of the intensity of failures, the value of λ_i is determined.

The Schick-Wolverton reliability model is a modification of the model proposed by Jelinski and Moranda. This model is based on the assumption that the intensiveness of errors is proportional not only to the number of errors in the program, but to the testing time ti, i.e. the probability of detecting errors increases over the course of time:

$$\lambda(t_i)=C(N-(i-1))\cdot t_i,$$

where *C* is the coefficient of proportionality; *N* is the number of errors in the program at the beginning of the test; t_i is the time interval between detecting the (*i*-1)-th and *i*-th error; I is the number of errors detected by the moment of time t_i.

The probability of an errorless performance of the software is

$$P(t_i)=e^{\int_0^{t_i}\lambda(t)\cdot dt}=e^{-(C(N-(i-1)))\cdot\frac{t_i^2}{2}},$$

then the probability density function is equal to:

$$f(t_i)=C(N-(i-1))\cdot t_i\cdot e^{-C(N-(i-1))\cdot\frac{t_i^2}{2}},$$

Further, as in the previous model, it is necessary to use the maximum likelihood method. The likelihood function has the following view:

$$L(t_1,t_2,...,t_n)=\prod_{i=1}^{n}f(t_i)=\prod_{i=1}^{n}C(N-(i-1))\cdot t_i\cdot e^{-C(N-(i-1))\cdot\frac{t_i^2}{2}}$$

If a natural logarithm is taken from the expression, we shall get:

$$\ln\left(L(t_1,t_2,...,t_n)\right)=\sum_{i=1}^{n}\left(\ln\left(C(N-(i-1))\cdot t_i\right)-C(N-(i-1))\cdot\frac{t_i^2}{2}\right).$$

Having found the partial derivatives $\frac{\partial \ln L}{\partial C}$, $\frac{\partial \ln L}{\partial N}$ and having set them to zero, we shall acquire a system of equations for determining the value of the maximum likelihood for parameters *N* and *C*:

$$\begin{cases} \frac{\partial \ln L}{\partial C} = \sum_{i=1}^{n} \left(\frac{1}{C} - C(N-(i-1)) \cdot \frac{t_i^2}{2} \right) = 0, \\ \frac{\partial \ln L}{\partial N} = \sum_{i=1}^{n} \left(\frac{1}{N-(i-1)} - C \cdot \frac{t_i^2}{2} \right) = 0. \end{cases}$$

The system of equations is solved after the unknown parameters *N* and *C* are replaced by the values of the estimates $\hat{N}$ and $\hat{C}$. Further, having substituted *N* and *C* in the equation of the intensity of failures, the value of λ_i an be determined.

In the discussed model, the recorded event is the number of errors detected during a given time interval; this is different to the Jelinski-Moranda model where the time of detecting each error is recorded.

Static models of reliability

A principal difference of static models from dynamic models is the recording of the time of error occurrences during the testing process. Besides, static models are not based on assumptions on the behavior of the function of failure intensity.

Corcoran's model considers only the result of *N* test runs in which N_i errors of the *i*-th type have been detected. For this the model uses the probability of detecting errors of each type. According to Corcoran's model, the probability of errorless performance of a program is calculated using the following formula:

$$R = \frac{N_0}{N} + \sum_{i=1}^{K} Y_i \frac{N_i - 1}{N},$$

$$Y_i = \begin{cases} a_i, N_i > 0, \\ 0, N_i = 0. \end{cases}$$

where N_0 is the number of errorless performances of the program; *N* is the total number of test runs; *K* is the known in advance number of error types; ai is the probability of detecting the i-th type error.

According to the model the probabilities ai are assessed according to data, which is known in advance, or data on the performance of similar programs.

Nelson's model dictates an assumption that the area of the output data is known in advance. This data will be tested. The area is divided into k non-intersecting subareas Zi, *i* = 1, 2, ..., *k*. In this model the probability of selecting a concrete test dataset for each program performance is considered for determining the probability of the program's errorless performance.

Let us suppose that Pi is the probability of choosing the dataset from Zi. Then the probability of errorless performance of the program shall be equal to:

$$R = 1 - \sum_{i=1}^{k} \frac{n_i}{N_i} P_i,$$

where N_i is the number of test runs; n_i is the number of detected failures; *i* is number of subarea of the input data.

The probability of selecting a concrete dataset for testing the program is determined by the partition of the area of the input data into subareas and determining the probability of the selected database for the next test run belonging to a designated subarea. Such assumptions can be made on the basis of empirical assessment of the probability of data appearing in a realistic scenario of exploiting the program.

It should be understood that it is impossible to produce an absolutely reliable program; there is an enormous number of variants for the program to perform that cannot be fully checked for their correctness. However, it is necessary to be aware of the reliability of a concrete software system. The described models are largely theoretical, hence having limited application. Currently, there have been suggested no methods for assessing reliability, having a minimal number of limitations [2].

Practical software engineering assumes that the task of providing reliability is greater than its assessment. However, it is evident that to provide a high level of reliability for a system it is necessary to first estimate it. For this task it is necessary to have at one's disposal a suitable unit for measuring the reliability of the software and for performing necessary calculations [3].

Considering the complexity of modern software systems, extremely high requirements for assessing their reliability should not be applied when speaking about their accuracy. Despite this, the estimation of their reliably may be used to provide rationale for their development, benefiting the consumer in terms of his or her assurance of the quality of the final software product.

Software cost-estimating models

With the development of the IT sector, the number of expensive software design projects has been constantly growing. It has become paramount to evaluate all potential profits and losses from the project during its earliest stages. Inadequate assessments of the challenges arising during the process of designing the software and the capabilities of the developers themselves often results in failing to meet the set deadlines and, consequently, in financial losses for the IT company. According to statistics, only 25% of the initiated projects met the required deadlines fully fulfilling the declared software features, 25% of the projects are called off, and 50% of the projects are completed with cost overruns and missed deadlines.

According to an annual report of The Standish Group, the costs of projects on average increases to 178% for major IT companies, 182% for middle-sector businesses, and 214% for small companies [4]. Part of the projects which did not meet the necessary deadlines and had an increase from their initial costs did not fully realize the software features declared in the design specifications. This can result in the additional financial losses for the IT company, if otherwise had not been specified in the terms of the contract.

For the development of complex software systems, which are generally integrated into corporate information systems, it is necessary to reduce the quality of the final results from such subjective attributes as the qualifications of the executive personnel, their work experience, and the management of the development process. In this aspect it is essential to employ methods for evaluating the resources, which are necessary for the development of the software.

The main reasons for an inadequate evaluation of the development costs are:

- unclear requirements declared by the ordering party during the early stages of development:
- mismanagement of the development process;
- unclear or incomplete technical specifications;
- underestimated labor intensity for fulfilling the task.

Consequently, the project manager requires an effective tool which can accurately estimate the efforts and costs required for developing the product. If the number of such projects is significant, the process should be computerized.

Forecasts can be made using algorithms for models of evaluating the costs of software development. These are the situations when such models can be employed:

- making strategic financial decisions for the development of the software product;
- determining the number of employees necessary for each stage of the project's life cycle, the financial and labor expenditures required for the project;
- making decisions on cutting the labor intensity, time requirements, software features, and product quality if limited resources are provided;
- distributing assignments and tasks to designated staff;
- making decisions regarding the purchase of prefabricated components for the project.

There are various models for estimating the costs of developing a software system. It is necessary to determine the concrete criteria of the model for the company.

Existing models and methods for software cost-estimating are divided into non-algorithmic methods (Price to Win, Parkinson's law, analogy estimation, evaluation by experts) and algorithmic models (SLIM, COCOMO, COCOMO II).

Non-algorithmic models

Non-algorithmic methods for software cost-estimating consist in the application specific schemes and principals, not mathematical formula. Here are some examples:

1. The Price to Win [5] method states that regardless of the amount of intented project expenditures, the software cost-estimation is corrected in accoradance with the demands of the client. Thus, the Price to Win method is an example of a policy in which there is regular negotiating with the client. This makes it popular with companies that do not have the opportunity to conduct a competent software cost-estimation. Consequently, this results in the deficit of resources, necessary for executing the project, failures in meeting deadlines, and, ultimatelly, in the loss of the contract and eventual bankrupcy of the IT company.

2. The cost-estimation method, based on Parkinson's law [6] reflects the process of ineffective exploitation of resources. According to Parkinson's law all the projected time will be consumed in the process of fulfilling the task; this time may even be exceeded. Thus, to increase the productivity of the team, the time required for executing the process is reduced.

3. The method involving evaluating by experts [7] is employed for projects that encorporate new technologies or for innovative tasks. The method assumes the participation of specialists who evaluate their part of the project. Further, all the results are combined into a unified system. The estimates of the experts are recorded and publically discussed. The inquiry of the experts may include the Delphi approach or its extended version, leading the experts to a consensus. Further, a componentwise estimate is performed. After each iteration the accuracy of the estimate improves.

4. A version of the aforementioned method is the anologue estimate. Here, as during the usage of algorithmic models, empirical data on earlier similar projects is utilized. In algorithmic models, however, this data is used indirectly, e.g. for calibrating the model's parameters. In the analogue estimate method empirical data is applied to select projects, which are similar to the one being realized. The first stage includes an analysis of requirements and the design of the potential system. Thus, data is collected and, based on an assesssment by experts, features for comparing the projects are selected. These features depend on the type of software system, potential challenges, the environment in which the product is developed, etc. During the next stage, a search and analysis of similar projects (according to their features) is performed. Finally, a number of projects with the slightest differences in the numerical values of the features is selected.

Algorithmic models

Software cost-estimating models describe the relation between the features of the project and the costs required for accomplishing it. The models are classified as linear, multiplicative, and exponential according to their interaction. There are empirical and analytical models in reference to the usage of data from earlier projects. Among the more often used and effective models are Putman's SLIM model (exponential and analytical) and the COCOMO model (exponential and empirical).

The SLIM model was developed by Larry Putman Sr. of Quantitative Software Management Inc. [8]. It is based on the analysis of software lifecycles and supports contemporary methods for measuring code size and function points. This enables building a curve for estimating the project's graph and its defects. In order to alter the shape of the curve, the Manpower Buildup Index (MBI) and Productivity Factor (PF) index are used. It is possible to save and analyze the data on earlier projects during the program realization of SLIM. If such data does not exist, a specialist can answer the relevant questions to calculate the value of the MBI and the PF using a specialized database.

In the SLIM model the value of productivity is used to link Reilich's simple capacity distribution model with the features of the project's scale and utilized technologies.

The productivity is calculated using the following formula:

$$P = \frac{S}{E},$$

where *P* is the productivity in units of measurement for program codes times man-month (person-months); *S* is the size of the software system, calculated using one of the techniques for calculating the size of the code (number of rows, function points); *E* are the expenses of the project, measured in man-months.

Reilich's curve is used to determine the distribution of the expenses. It is simulated by the expression:

$$m(t) = 2 \cdot K \cdot a \cdot t \cdot e^{-at^2},$$

where *m(t)* is the coefficient of workforce demand during the moment of time *t*; *K* is the total labor cost during the project, measured in man-years; *a* is the acceleration factor.

The acceleration factor can be calculated using the following formula:

$$a = \frac{1}{2t_d^2},$$

where *td* is the time required for completing the project.

The model used the assumption that the personnel's peak level in Reilich's curve corresponds to the time required for completing the project td.

Afterwards, the line of the main trend is built and it is determined whether the project is within the accessibility range; if it is, the project can be continued, otherwise it will be considered impossible to complete.
The following equation is used for this model:

$$S = \frac{C}{K^{1/3} + t_d^{4/3}},$$

where *S* is the number of rows in the software code; *C* is the environmental factor; *K* is the total labor cost for completing the entire project, measured in man-years; *td* is the time limit of the project according to schedule.

The environmental factor value can be calculated using the following formula:

$$C = \frac{S}{K^{1/3} + t_d^{4/3}},$$

where coefficients *K* and td are determined using a database with earlier projects, having a size *S*. The value of *C* is subjected to a calibration and can be used for future estimations.

The Constructive Cost Model (COCOMO) was developed by Barry Boehm. This model was initially based on the analysis of 63 software projects with different characteristics. The analysis considered the actual code size of the software, labor costs, and the actual time required to complete the project.

The model utilized three modes, which determine the complexity of the system and the environment in which the project is being conducted: an organic mode, semi-detached mode, and embedded mode.

The organic mode is applied to small-scale software products; the semi-detached mode is applied to projects in the middle range, requiring some innovative additions. The embedded mode, in its turn, is applied to highly-complicated projects, requiring significant research and development, innovations, and a large team of professionals.

The estimating accuracy of this model depends on the specifications requirement; it can be Basic, Intermediate, or Detailed. The Basic level required only the value of the code size and data on the current level. The Intermediate level utilizes the value of 15 additional parameters, which are rated on a six-point scale. This level also requires additional variables.

To determine the values of these variables (cost drivers) rating tables are used. These contain a number of attributes related to the development process. To select the necessary value, the project manager should select those conditions which are relevant to the actual situation. The basic formula for estimating the effort applied is

$$E = a \cdot S^b,$$

where *a* and *b* are constants, determined during a regressive analysis (depending on the project); *E* are the effort applied (man-months); *S* is the size of the software code (thousands of rows).

COCOMO includes such development stages as design, coding, and testing. All other stages are excluded. The model allows for a calibrating of the coefficients on the base of a number of projects from a specific company; this contributes to the model's accuracy. A major disadvantage of this model is the fact that it is built on a complicated system, which is measured in thousands of rows, i.e. the size of the software package's own code.

As requirements to software systems increased, COCOMO became largely obsolete.

In 1999 COCOMO II was developed. The calibration of the parameters was performed on the basis of 161 software projects.

The model utilized a regression formula, the parameters of which are determined based on industrial data and the characteristics of the specific project. There are two stages in the cost-estimating of a project: a preliminary assessment, conducted during the initial phase and a detailed assessment after the architecture design of the potential system. Estimating the effort applied for developing the project in man-months is determined using the following formula:

$$PM = A \cdot SIZE^E \cdot \sum_{i=1}^{n} EM_i ,$$

where *SIZE* is the size of the code in thousands of rows; EM_i are the multipliers of the effort applied, *A* = 2.94, *B* = 0.91.

Seven multipliers are used during the preliminary estimating stage (n = 7); n = 17 for the detailed stage. The coefficient *E* is calculated using the following formula:

$$E = B + 0.01 \cdot \sum_{j=1}^{5} SF_j ,$$

where SF_j is the scale factor.

The following scale factors SF are considered:

PREC – Precedentedness;
FLEX – Development Flexibility;

RESL – Risk Resolution;
TEAM – Team Cohesion;
PMAT – Process Maturity.
For preliminary estimating the following multipliers of effort applied are used:
PERS – Personnel Capability;
RCPX – Product Reliability and Complexity;
RUSE – Required Reuse;
PDIF – Platform Difficulty;
PREX – Personnel Experience;
FCIL – Facilities;
SCED – Schedule.
For detailed estimating the following multiplier of effort applied are used:
RELY – Required Software Reliability;
DATA – Data Base Size;
CPLX – Product Complexity;
RUSE – Required Reusability;
DOCU – Documentation match to life-cycle needs;
TIME – Execution Time Constraint;
STOR – Main Storage Constraint;
PVOL – Platform Volatility;
ACAP – Analyst Capability;
PCAP – Programmer Capability;
AEXP – Applications Experience;
PEXP – Platform Experience;
LTEX – Language and Tool Experience;
PCON – Personnel Continuity;
TOOL – Use of Software Tools;
SITE – Multisite Development;
SCED – Required Development Schedule [9].

For the scale factors and the multipliers of effort applied table attributes are given. It is evident that this model does not allow to the user to set the required level of reliability; the RELY factor is given only as a table attribute and is not informative.

An advantage of this model is its ability to utilize factual data on earlier projects and the expert review on them.

COCOMO II can utilize a composition of applications through object indicators (function points). The composition of applications has replaced the Basic, Intermediate, and Detailed levels of COCOMO. It allows incorporating modern design methods, e.g. constructing a prototype. Since it is impossible to assess the size of the product visually using code rows, the function points can be a screen form, a printed form, a report, a module, an element of a database, etc. Each function point corresponds to a complexity level: simple, medium, and difficult.

The estimating of the effort applied and time required for the development of the project is determined using the following formula:

$$Effort = A \cdot Size ,$$

where *Size* is the estimation of the size of the forms, reports, modules of the potential system (each object indicator is evaluated from 1 to 10 depending on its complexity); *A* is the coefficient, taking into account a secondary application of part of the components and the productivity of the developing process, which depends on the experience of the personnel and applied tools (evaluated from 4 to 50).

CONCLUSION

The conducted analysis has demonstrated the following key positive aspects of COCOMO II:

- it allows estimating the effort applied with account for different levels of the technical specifications;
- it supports contemporary software design methods.

The main disadvantage of the discussed software cost-estimating models is the necessity to calibrate the models – a process which requires a large database of earlier projects; information on many projects may not be available. Another disadvantage is inability to specify a more or less appropriate level of performance reliability; this is especially critical for developing fault secure systems.

To adequately plan the project and estimate its cost it is essential to conduct a preliminary design of the software product. Decomposition of the project produces a number of components, which form the software product. In COCOMO II such components are referred to as function points.

ACKNOWLEDGMENTS

The work was supported by the Ministry of Education and Science of the Russian Federation (State assignment to higher education institutions and research organizations in the field of scientific activity; agreement No. 2.7711.2017/Б4).

REFERENCES

[1] Gagarina, L. G. (2008). *Tekhnologiia razrabotki programnogo obespecheniia [The technology of developing software.* Moscow: INFRA-M, 2008,.

[2] Bakhtizin, V. V. (2007). *Uchebnoe posobie po distsipline “Nadezhnost’ programnogo obespecheniia” [Textbook on the reliability of software.* Minsk.

[3] Serenkov, V. I., Kartsan, I. N., Dmitriev, D. D. (2013). Method for the synthesis of amplitude-phase distribution of hybrid-mirror download. *Vestnik of Reshetnev Siberian State Aerospace University*, 16(3), 664–670.

[4] *The Standish Group Report.* Retrieved from http://www.projectsmart.co.uk/docs/chaos-report.pdf.

[5] Boehm, B. W. (1981). *Software engineering economics*. Prentice-Hall.

[6] Parkinson, S. N. (1957). *Parkinson’s Law and Other Studies in Administration*. Houghton-Miffin.

[7] Coates, J. (1999). *Technological Forecasting and Social Change*. The Netherlands: Elsevier Science Inc.

[8] Putnam, L., Myers, W. (1992). *Measures for Excellence*. Youdon Press Computing Series.

[9] Abts, C., Clark., B., Devnani-Chulani, S. *COCOMO II Model Definition Manual*. Retrieved from http://sunset.usc.edu/research/COCOMOII/Docs/modelman.pdf.