UDC 004.413.5

OVERVIEW AND ANALYSIS OF METHODS FOR EVALUATING SOFTWARE AT THE DESIGN STAGE

GRUZDO I.V., KYRYCHENKO I.V., TERESHCHENKO G.Y.

Analyzes the existing solutions used in the evaluation of software (software) at the design stage, which can reduce a number of problems encountered during development. The issues and problems of software evaluation at the design stage, which are used in the development management process, are considered. Methods for evaluating software at the design stage are discussed and a reasonable assessment is made of the appropriateness of their use.

Key words: software, software, quality, software evaluation, standards, method, algorithm, type of project.

1. Introduction

To reduce the risks of not completing a software project, special attention is paid to the quality of both the developed software and its development process. Because of this, the tasks related to software evaluation throughout its entire life cycle (LC) are of particular relevance. Therefore, special attention should be paid to the quality assessment of the software at the design stage, since not only the quality but also the project risks depend on it same and cost.

Software quality (software quality) – the entire volume of features and characteristics of software, which refers to the ability to meet established or perceived needs.

It should be borne in mind that the importance of each quality characteristic varies depending on the software class, web, desktop, mobile application or cross-platform. It should also be remembered that often on a project, the importance of each quality characteristic varies depending on the constraints adopted in the project, and depending on the decisions of the project manager or team in relation to the project and the team involved in the design and development of the project.

Therefore, taking into account the above, a significant role is played by the balance between a number of variable characteristics that affect the quality as a whole. It should be noted that the following resources affect the quality of software: human, material, hardware and time resources. You should also not forget about the importance of the adopted design constraints for various classes of software. Therefore, when organizing a software development management process, special attention should be paid to the accumulation of information and analysis of the interrelationships of factors and the results obtained, as well as the influence of decisions taken at the software design stage on the success of the entire project and its quality, completeness and popularity among users.

The above circumstances determine the relevance of the task of studying existing software evaluation methods at the design stage, namely, statistical evaluation methods. All this will allow a more conscious approach to the choice of methods, relying not only on accepted standards for developing projects, but also on a particular class of software using statistical prerequisites to improve its quality.

The purpose of the article is to analyze existing software evaluation solutions at the design stage, analyze the most frequently used statistical software evaluation methods, as well as substantiate the choice of a solution depending on the specific software class and the subtasks to be solved.

2. Problems of Software Evaluation at the Design Stage

The software evaluation process consists of three stages: the establishment (definition) of quality requirements, the preparation for the assessment and the assessment procedure, as well as the selection of the main characteristics for use in subsequent projects as statistical prediction prerequisites. This process can be applied in any suitable life cycle phase for each software component.

Software evaluation problems that may arise at the design stage: 1) the subject area may not be well understood by the developers and / or customers due to the fact that some facts are missed or distorted; 2) insufficient or complete lack of statistical data or experience of the project manager in the development of a project of a specific type, which makes it impossible to create a basis for software evaluations in the future; 3) ignorance of the standards with which you can perform the evaluation process or their complete disregard in the process of software design and development, resulting in a decrease in software quality and a lack of knowledge in the conduct of the evaluation process; 4) poorly documented and described requirements at the beginning of the project, lack of specification; 5) a complete discrepancy in understanding the requirements between the project manager, the team lead, the people involved in the project and the customer; 6) a complete discrepancy in understanding the purpose of software development between the project manager, team lead, the people involved in the project and the customer; 7) at the design stage of the software, errors are either hidden or overlooked, as a result of which a false impression is created about important characteristics affecting the quality of the software; 8) the quality of assessment is highly dependent on the subjects and their experience involved in the assessment process; 9) the project manager, team leads, developers, analysts, testers, and those who embed the product may have different ideas about the assessment processes and software improvement opportunities; 10) the choice of the main quality characteristics for use as statistical prerequisites of the forecast in subsequent projects is not sufficiently substantiated.

Most of the problems that arise during the software design phase reduce the use of historical data accumulated, namely, the analysis of the main characteristics that make it possible to compare the complexity of a project with the complexity of previous projects of a similar type, size, orientation and human composition.

However, it is not always possible to apply for a new project, only if a number of conditions are met: 0) the real results of previous projects are accurately documented in the company; 1) the selected characteristics of software quality for documentation remain unchanged for different projects and within the adopted restrictions on the project, as well as most fully and briefly describe the decisions made and their impact on the project; 2) at least one of the previous projects, and preferably some belong to the corresponding software class (web, desktop or mobile application) or cross-platform software that was previously developed; 3) at least one of the previous projects, and preferably several have a similar substantive focus and size; 4) Life cycle, the used methodology, methods and development tools, qualifications and experience of the project team of the new project are also similar to those that occurred in previous projects; 5) are developed using the same programming language and are based on the same design patterns as similar, archival projects; 6) have in their composition the same or similar functions that in the complex perform similar calculations.

Independent software assessments at the design stage in most cases are performed by people who do not always take into account the relationship between software quality and the development team and resources that are on the project, which in turn leads to erroneous results and is impractical.

Given the above, the most appropriate practice is that when the project manager, together with the architect, the team lead on the development and testing, in the analysis process, perform several iterations in assessing the necessary resources affecting the quality of the entire software. It is also necessary to pay attention to the processes of identifying and describing problems associated with the evaluation of software that may arise at the design stage.

3. Software Evaluation Methods at the Design Stage and Their Classification

A special place in the process of evaluating software quality is represented by software assessment methods at the design stage. The classification of methods according to Vendrov [1] is given in Table 1.

Table 1

№	Title	Short description
1	Algorithmic modeling	based on the analysis of statistical data on previously completed projects, often determined by the dependence of the complexity of the project from a quantitative measure of software. At first, the selected quantitative indicator for software is evaluated, and then future costs are predicted using the model.
2	Expert ratings	firstly, a survey is conducted of several experts on software development technology who know the scope of the software being developed. Each of the experts gives his assessment of this project. All ratings are compared and discussed. The process is repeated until agreement is reached on the final decision-making and estimates.
3	Evaluation by analogy	is used if such software has already been implemented previously and there is complete information about it. The planned software is compared with previous projects with similar characteristics. Experts give a likely assessment of the complexity, based on the differences between the new and previous projects.
4	Parkinson's Law	according to this “law”, the efforts spent on work are distributed evenly over the time allotted for the project. Here, the criteria for estimating project costs are human resources, and not the target assessment of the software product itself.
5	Evaluation in order to win the contract	project costs are determined by the availability of those funds that are available to the customer. Therefore, the complexity of the project depends on the budget of the customer, and not on the functional characteristics of the product being created. Requirements have to be changed so as not to go beyond the adopted budget.

Each of the above software evaluation methods has weaknesses and strengths; therefore, to improve the quality of software, it is advisable to apply several assessment methods simultaneously for their subsequent comparison and estimation of the motion vector and achieving the required quality. In the case when in the course of the analysis completely different results are obtained, it is possible to judge that there is not enough information to obtain a more accurate assessment, or the wrong characteristics were selected within the framework of the designed software. In this case, you need to use additional information, or choose more indicative criteria, after which you should perform a reassessment, and so on, until the results of different methods become close enough.

Given the above, we can conclude that the project was successful, you must most accurately perform an assessment of the complexity of the project. Therefore, it is necessary to take into account the main characteristics affecting the assessment of the complexity of software development at the design stage. However, they must meet the following characteristics [1]:

1) characteristics should be created and maintained by the project manager and the teams of architects, developers and testers responsible for carrying out the work;

2) is perceived by all performers equally and as an ambitious, but achievable task;

3) based on a detailed and well-founded assessment model;

4) based on data from similar projects, which include similar processes, technologies, environment, quality requirements and qualifications of employees;

5) has the same or similar data for the calculated / forecast models;

6) be described in detail so that all key risk characteristics are visible, and the probability of success is objectively evaluated.

There are both theoretical and statistical models for calculating the complexity of software development. All of them are based on the general classification of software methods according to Vendrov.

The most well-known and applied in practice: a method for analyzing the complexity of a project based on the complexity of a known sample [4]; analysis of labor intensity based on expert assessments [1]; assessment of the cost and size of software depending on the stage of the project [4, 5 6]; algorithmic modeling of the complexity of software development based on the class of models COCOMO [1, 5, 7-10]; methodology for assessing the complexity of software development based on use cases [11]. Consider them in more detail in paragraph 2 of this article.

Most of the existing models for determining the complexity of software development can be reduced to the relationship of five parameters:

1) the size of the final product, which is usually measured by the number of lines of source code or the number of functional points necessary to implement this functionality;

2) features of the process used to obtain the final product, in particular its ability to avoid unproductive activities;

3) the capabilities of the staff involved in software development, in particular his professional experience and knowledge of the project domain;

4) an environment that consists of the tools and methods used to effectively perform software development and automate the process;

5) the required quality of the product, including its functionality, performance, reliability and adaptability.

The relationship between the parameters can be summarized as a formula:

Workload = (Personnel) • (Environment) • (Quality) • Code Size / Software).

Among all the parameters in the classical literature, the most significant factor for evaluating the complexity is the size of the software.

The procedure for assessing the complexity of software development in general consists of the following actions [2]:

1) an estimate of the size of the product being developed;

2) an assessment of labor input in man-months or man-hours;

3) an estimate of the duration of the project in calendar months;

4) project cost estimate.

Back in the 1970s, Lawrence Putnam [8], using statistical analysis of projects, found that the relationship between the three main project parameters (size, time, and labor-intensiveness) resembles the Norden-Rayleigh function, reflecting the distribution of project labor resources over time. Rayleigh function [8]:

$$\frac{dy}{dt} = 2 * K * a * t * \exp(-at^2).$$

,

where $\frac{dy}{dt}$ – growth rate of project staff; t is the time elapsed from the start of the project to withdrawal of the product from operation; K - the area under the curve – represents the full complexity during the entire life cycle, expressed in man-years; a is the acceleration factor (constant), defined as [8]:

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

where t_d – development time.

Taking a number of assumptions, Putnam derived the equation [8]:

$$E = 0,4 * (\frac{S}{C})^3 * \frac{1}{(t_d)^4},$$

where E is the complexity of software development, S is the size of the software in LOC, t_d-planned development time, C - technological factor, taking into account various hardware limitations, staff experience and characteristics of the programming environment.

The second directions in the use of statistical models are those that use accumulated historical data to obtain values for the coefficients of the model. There are two directions: linear and nonlinear.

Linear statistical models are expressed in the form:

$$\text{Laboriousness} = b_0 + \sum_{i=1}^{n} b_i * x_i ,$$

where Xi are factors that influence labor intensity, Bj are model coefficients.

In practice, linear models do not work too well, because the relationship between labor-intensiveness and software size is non-linear. As the size of the software grows, an exponential negative scale effect occurs.

Nonlinear, statistical models are as follows:

$$\text{laboriousness} = A * (\text{Softwaresize})^b ,$$

where A is a combination of factors affecting the complexity; b is the exponential scale factor.

Estimation of product size is based on knowledge of system requirements. For such an assessment, there are two main ways: by analogy and by calculating the size of certain algorithms based on the original data.

The base among all indicators is the labor input indicator [2]:

$$Q = q * c ,$$

where Q is the conditional number of teams, q is a coefficient that takes into account the conditional number of commands, depending on the type of task, is calculated in a table, c is a coefficient that takes into account the novelty and complexity of the program, and is determined in the same way.

Then the time to create software is calculated. The total time to create software is made up of various components. The structure of the total time to create software is calculated [3] depending on the time needed for a particular development stage.

The time spent at each stage of software creation is calculated according to the following algorithm:

1) T_{PO} – time for preparing the description of the task, varies from 3 to 5 days, 8 hours each: person / hour.

2) T_O – the time for the description of the task is determined by the formula:

$$To = Q * B / (50 * K) \text{ men/hour} ,$$

where B is the coefficient for accounting for changes in the task, depends on the complexity of the task and the number of changes; K - coefficient taking into account the qualifications of the programmer, depending on the length of service.

3) Ta is the time for the development of the algorithm; we count by the formula:

$$Ta = Q / (50 * K) \text{ men/hour}$$

4) Tbs - the time to develop a flowchart is defined as the same as Ta.

5) Tn - the time of writing a program in a programminglanguage is determined by the formula:

$$Tn = Q * 1,5 / (50 * K) \text{ men/hour}$$

6) TP - time "set" of the program is determined by the formula:

$$Tp = Q / 50$$
$$\text{men/hour}$$

7) Tom - the time for debugging and testing the program is determined by the formula:

$$Tот = Q * 4,2 / (50 * K) \text{ men/hour}$$

8) Td - time for registration of documentation, is taken after the fact and is from 3 to 5 days for 8 hours:

$$Td = \text{men/hour.}$$

Total labor costs are calculated as the sum of the composite labor costs according to the formula:

$$T = Tpo + To + Ta + Tbc + Tn + Tp + Tот + Td$$

It can be concluded that statistical models are easy to understand, but have the following disadvantage: the results are valid mainly for a specific situation. Also, as the number of input parameters increases, the amount of data needed to calibrate the model also increases.

4. Review and Analysis of the Most Used Software Evaluation Methods at the Design Stage

4.1 Method analysis of the complexity of the project on the basis of the complexity of a known sample can be represented as an algorithm:

1) As the value of the complexity of the main work choose data characterizing the complexity of the same software.

2) Regarding similar software, a coefficient of complexity of a new development or part of it is introduced.

3) Calculate the complexity of the program-analogue or its separate part.

4) Determine the qualification ratio of the employee (programmer, tester, etc.), which reflects the degree of his preparedness to perform the work assigned to him.

5) Calculates the complexity of manufacturing new software or its individual modules according to the formula [4]:

$$q_i^{new} = \frac{q_i^a \cdot n_{skl}}{n_{kv}}$$

6) Determine the time of execution of all work, or work within individual stages. In the classic management is divided into time intervals: the development of a general scheme of the software, writing software, testing and making corrections, as well as writing the supporting documentation.

7) Calculate the labor costs for a specific stage [4]:

$$q_i = q_i^{prog} + q_i^{al} + q_i^{test} + q_i^{doc},$$

where q_i^{prog} - labor costs and software manufacturing, q_i^{al} - algorithmization costs, q_i^{test} - labor costs for testing and making corrections are determined by the amount of labor costs for the implementation of each component of this work, q_i^{doc} - the cost of writing documentation reflect the ratio of labor costs for the creation of supporting documentation in relation to the labor costs of software development.

8) Definition of labor costs for the stages of a project or a project as a whole:

$$q_i = q_i^{prog}(1 + n_i^{al} + n_i^t + n_i^{cor} + q_i^{doc})$$

9) Definition of labor costs for software design:

$$q_i^{prog} = \frac{q_i}{n_a + 1 + n_t + n_c + n_{To}}$$

Labor costs for the introduction of new software q_0 depends on the time for the implementation of trial operation, which is agreed with the customer and, often by Scrum is equal to the 1st sprint or one month or 22 man-days. At the same time it is necessary to take into account the risks associated with personnel, namely, illness, unplanned meetings, etc. etc. as they take productive staff time.

10) Calculate the total value of labor for the project:

$$Q_p = Q_{pr} + q_0$$

A complete list of works with their separation by project implementation stages is sometimes drawn up in the form of a design complexity calculation table, which reflects the content of the project work depending on the specific development stage.

This method of calculating the complexity of the design, gives a more complete picture of the attitude to the stages of project development and will take into account the possible risks, because more approximate gives adequate estimates. Also, the software correction factor allows you to operate with values when the amount of work increases. During the determination of labor input for each stage of work, it allows to correlate the labor intensity of the main work with the laboriousness of other types of work, which in turn makes it possible to calculate the costs corresponding to real conditions.

4.2 Analysis of the complexity on the basis of expert estimates

Carrying out the analysis of labor intensity on the basis of expert assessments, it is necessary to select experts at the beginning within the solvable domain of the software. Algorithmically, the method is as follows:

1) Survey a few experts.

2) Assignment of weights to each of the experts, depending on the position held and work experience.

3) The results recorded in the table (see table 2.)

Table 2

Specifications	Weight	T_{min}	T_{avr}	T_{max}
	0,25			
	0,05			
	…			
General				

4) On the basis of the received expert estimates, the definition of the duration of each work (stage of work) for the project (minimum, average and maximum duration of work).

5) The definition of the integral assessment and development of a generalized table.

6) The definition of the expected duration of work q_i, calculated as the expectation for β - distribution by the formula [1]:

$$q_i = \frac{3T_{min}^i + 2T_{avr}^i + T_{max}^i}{5},$$

where T_{min} и T_{max}- min. and max. work duration, and T_{avr} - the average duration of work, according to expert estimates.

This method is very good in the case when experts have experience in developing such projects, and when the level of expert in the team corresponds to its professionalism, although there are cases when the middle knows more than the leader. The weakness of the method lies in the fact that the degree of similarity of the new project and the previous ones, as a rule, is not too great if the company develops different classes and orientation of the software and the company has frequent staff turnover.

4.3 Estimation of cost and software size depending on the project stage.

The accuracy of estimating the cost and size of software depending on the stage of the project is determined using the schedule [4]. This method uses software size measurements using the number of lines of code and function points metrics.

4.3.1 Number of lines of code (LOC - Lines of Code) is one of the most common units of measurement. However, it contains a number of subjective

assessments that affect the result, since different values can be obtained on the same data.
Algorithm LOC.
1) Calculation of the expected value of the assessment:

$$LOC_{expected} = \frac{LOC_{best} + LOC_{worst} + 4LOC_{possible}}{6}.$$

2) Calculate performance values:

$$Pr = Pr_{average} * (LOC_{average} / LOC_{expected}).$$

3) Determination of total software costs:

$$3 = (LOC_{expected}) / Pr_{average}.$$

4) Project cost estimate:

$$Cm = (LOC_{expected}) / UdS\tau_{average}.$$

The advantages of using LOC are as follows:
easy adaptability; the ability to compare methods of measuring size and performance in different groups of developers; ease of evaluation before project completion; Estimation of software size based on the developer's point of view.
Although in practice this metric is often used, it has several drawbacks: it is difficult to estimate the size of software in the early stages of development; not regulated by standards; Possible distortion of LOC indicators by a programmer to get more salary. Similarly, the relationship between LOC and the effort expended is not linear.
Despite all the shortcomings of this method, it is advisable to use it in combination with other indicators, which allows you to get a rough estimate that will display values close to real results.
4.3.2 Function Point Calculation (FP - Function Points), used to estimate the resources required for software development and maintenance.
The counting of functional points can be represented as a sequence of steps [6]:
1) Determination of the type of assessment performed: development project, development project, product.
2) Determination of the scope and boundaries of the product: all developed functions or all added, modified and deleted functions; only functions actually used, or all functions.
3) Determining the number and complexity of functional types by data. It is determined based on "entity-relationship" diagrams or class diagrams. For each identified functional type, its complexity is determined. It depends on the number of elementary data (data element types, DET) and elementary records (record element types, RET) associated with this functional type.
The dependence of the complexity of functional types on the number of DET and RET is determined by the table "The complexity of ILF and EIF" [6].
4) Calculation of function points associated with transactions. The number of transactional functional types is determined on the basis of identifying input and output documents, screen forms, reports, and also by class diagrams.
It should be noted that there are a number of rules that must be followed when calculating DET for EI and calculating DET for EO.
5) The determination of the total number of non-aligned functional points (UFP) is determined by summing over all information objects (ILF, EIF) and elementary operations (EI, EO, EQ transactions).

$$UFP = \sum_{ILF} UFPi + \sum_{EIF} UFPi + \sum_{EI} UFPi + \sum_{EO} UFPi + \sum_{EQ} UFPi$$

6) The definition of the equalization factor (VAF) value is applied when system-wide requirements are imposed on software that limit developers to choose a solution and increase the complexity of development. The value of the factor VAF depends on the 14 parameters that determine the system characteristics of the product.
The calculation of the equalization factor is made according to the formula [6]:

$$VAF = (TDI * 0.01) + 0.65.$$

For each functional type, the number of functional points included in its composition is calculated. The calculation is performed in accordance with the values from the table showing the dependence of the amount of FP on the complexity.
7) Calculate the number of aligned functional points (AFP). The initial estimate of the number of aligned functional points for software is determined by the following formula [6]:

$$AFP = UFP * VAF.$$

This assessment takes into account only the new functionality that is implemented in the software. A software development project is evaluated at a DFP (development functional point) using the formula:

$$DFP = (UFP + CFP) * VAF,$$

where CFP (conversion functional point) - functional points calculated for the additional functionality that will be required when installing the software.
The project of improvement and improvement of the product is estimated at the EFP (enhancement functional point) by the formula:

$$EFP = (ADD + CHGA + CFP) * VAFA + (DEL * VAFB),$$

where ADD - functional points for added functionality; CHGA - functional points for modified functions; VAFA - the value of the equalization factor calculated after the completion of the project; DEL is the amount of remote functionality; VAFB - the value of the equalization factor calculated before the start of the project.
The expediency of using functional points is due to the fact that the measurements do not depend on the

technological platform on which the product will be developed. At the same time, a significant advantage is the fact that a uniform approach to the evaluation of all projects in the company is ensured. It is also necessary to store statistical data on labor costs for the implementation of functional points for previously implemented projects.

4.4 Algorithmic modeling of software development based on the class of models COCOMO

Constructive COst Model - a constructive cost model developed by Barry Boehm is one of the most well-known and well-documented models for estimating the complexity of software development. It includes different approaches for different classes of software and development methodologies.

COCOMO estimated equations [8]:

$$W = C_1 * EAF * (Siz)^{P_1};\ T = C_2 * (W)^{P_2},$$

where W is the number of work person-months C_1 - the scaling factor. EAF is a clarifying factor characterizing the subject area, personnel, environment and tools used to create software. Siz - the size of the final product. P_1 is an exponent characterizing economies of scale. T is the total number of months. C_2 - scaling factor for the timing of execution. P_2 is an exponent that characterizes inertia and parallelization inherent in software development management.

The basic equation of COCOMO:

$$E = a_b * (KLoC)^{b_b};\quad T = c_b(E)^{d_b};\quad Nd = E/T$$

where E - the complexity of software development in person-months; KLoC - estimated program size in thousands of lines of source code; T - development time / duration, in months; Nd - the number of developers in people. The coefficients ab and exponent bb are taken from the table.

Formula COCOMO for the average level [8]:

$$E = a_b * (KLoC)^{b_b} * Rf,$$

where Rf is the regulatory factor.

The detailed level (Advanced COCOMO) is aimed at improving the accuracy of assessment due to the hierarchical decomposition of the software being created and taking into account the cost factors at each level of the hierarchy and in phases of work [8]. Allows you to perform software evaluation by introducing additional factors.

COCOMO for software production by assembling reusable components for:

- simple project - for small development teams

$$E = 2.4 * (KLoC)1.05 * M$$

The multiplier M consists of: reliability and software complexity level, reusable components, development platform complexity, staff capabilities, staff experience, work schedules and support tools. The advantages of this approach are that it is possible to calculate, by combining the values, more detailed indicators that are used at the post-architectural level.

- medium complexity - in the development of which team members may feel a lack of experience and knowledge of the relevant systems

$$E = 3 * (KLoC)1.12 * M$$

- an embedded system project, where the software is part of a complex of hardware and software, other technical mechanisms and devices

$$E = 3.6 * (KLoC)1.20 * M$$

COCOMO II is a more advanced metric for calculating the complexity of a project used in multi-component development [9].

Labor input (in person-months):

$$E_{NS} = a \times KLoC^{E} \times \prod_{i=1}^{n} EM_t,$$

where $$E = B + 0{,}01 \times \prod_{j=1}^{5} SF_j,$$

Calendar time:

$$TDEV_{NS} = C \times (E_{NS})^{F},$$

where

$$F = D + 0{,}2 \times 0{,}01 \times \sum_{j=1} SF_j = D + 0{,}2 \times (E - B),$$

EMt - multiplicative coefficients of labor; SFj - exponential scale factors; KLoC - software size expressed in thousands of lines of source code or the number of function points without taking into account correction factors (UFP). coefficients EMt reflect the combined effect of parameters.

The estimated process maturity level (EPML) is calculated as follows:

$$EMPL = 5 \times (\sum_{i=1}^{n} \frac{KPA\%_i}{100}) \times \frac{1}{n},$$

where the value of KPA% is determined tabularly.

COCOMO II for multicomponent development [8, 9].

1) The total size of the product is calculated as the sum of the sizes of its components:

$$KLoC^{a} = \sum_{k=1}^{N} KLoC_k\ .$$

2) The basic complexity of the project is calculated by the formula:

$$E^{b} = a * (KLoC^{a})^{E} * SCED,$$

where SCED is the schedule compression.

3) Then the basic labor intensity of each component is calculated:

$$E_k{}^{b} = E^{E} * \frac{KLoC_k}{KLoC^{a}}.$$

4) In the next step, an estimate of the complexity of the components is calculated taking into account all the factors of labor intensity, except for the SCED factor.

$$E_k' = E_k^b * \prod_{i=1}^{6} EM_i .$$

5) The total complexity of the project is determined by the formula:

$$E = \sum_{k=1}^{N} EM_k' .$$

The project duration in the COCOMO II method is calculated by the formula:

$$TDEV = C*(E_{NS})^{D+0.2*0.01*\sum_{j=10}^{B} SP_j} * \frac{SCED}{100},$$

where C = 3.67; D = 0,28; - the complexity of the project without taking into account the SCED multiplier, which determines the schedule compression.

SOSOMO II is actively used in the Rational Unified Process technology and is constantly evolving.

SOSOMO Agile is a light version of SOSOMO II adapted for software development according to the Agile methodology. The technique is as follows [10]:

1) set the complexity of the previous completed project to a metric or as the final cost of the project;

2) the characteristic of SOSOMO of the previous completed project is set;

3) the characteristics of the SOSOMO new project are assumed.

4) the labor intensity and cost of the new project are calculated as deviations from the values of the previous one.

This technique works well with Agile projects.

As advantages of SOSOMO, it can be noted that actual data are used. The method is repeatable and fairly universal, well suited for projects that are not very different in size, complexity, and is quite simple. The disadvantages include the fact that the variability of requirements is poorly taken into account, the skills and knowledge of the customer, as well as levels of staff interaction are ignored.

4.5. Methodology for assessing the complexity of software development based on use cases

This technique is based on the identification of actors and use cases. It consists of the following basic steps [11]:

1) Determination of the weights of the actors. The calculated number of actors of each type ni is multiplied by the corresponding weighting factor kai, after which the total weight indicator A is calculated

$$A = \sum n_i \times k_{ai}.$$

2) Determination of the weights of use cases. The generic weight indicator UUCP (unadjusted use case points) is calculated as:

$$UUCP = A + UCP.$$

3) Determination of the technical complexity of the project (TCF - technical complexity factor) - is calculated according to the table, taking into account indicators of technical complexity. The TCF value is calculated by the formula:

$$TCF = 0,6 + (0,01*(\sum T_i Weight_i)).$$

4) Determination of the level of qualifications of developers (EF - environmental factor) is calculated taking into account the weights of the table.

The value of EF is calculated by the following formula:

$$EF = 1,4 + (-0,03*(\sum F_i Weight_i)).$$

5) Evaluation of the complexity of the project. UCP end value (use case points):

$$UCP = UUCP * TCF * EF$$

Using this method in the work of an IT company, you can achieve the desired results, regardless of the complexity of the software being developed, since all the essential characteristics are embedded in this model.

4. Conclusions

1) The analysis of the current state of the software evaluation problem at the design stage, which is used today in the management of the development, was performed. During the analysis, the importance of performing a preliminary assessment of the software was determined before embarking on its implementation, since This allows you to evaluate the software before it starts, which in turn allows you to take into account most of the possible risks of the project and the development stages. In turn, this allows you to compare what costs are necessary and calculate the cost of the project.

2) Issues and problems of software evaluation at the design stage, which are used in the development management process, are considered. The conclusion was made that the use of accumulated historical data allows to reduce the problems arising at the software design stage and thereby improve the quality of the final software.

3) Methods for evaluating software at the design stage are discussed and a reasonable assessment is made of the appropriateness of their use. When evaluating software at the design stage for improving the quality, it is advisable to use several evaluation methods for their subsequent comparison, since this will allow to achieve the required quality. If the result is completely different results, it means that there is not enough information to obtain a more accurate assessment, or the wrong characteristics were selected within the framework of the designed

software. In this case, you need to use additional information, or choose more indicative criteria and then repeat the assessment, and so on until the results of the various methods become close enough.

The obtained results will allow to continue the work on solving the problem of choosing the metrics for software evaluation used in the development of software projects on the entire life cycle.

Further studies are related to the analysis of existing software development standards that support the assessment of software quality and reliability, among which special attention should be paid to the documents of the IEEE 982 series, ISO / IEC 9126, DSTU 28195, RUP.

Transliterated bibliography:

1. *Vendrov A .M.* Designing software of economic information systems: Textbook. M.: Finance and Statistics, 2006. 544 p.

2. *Burlak G. N. Blagodatskikh V. A.* Economic aspects of the development and use of software. M.: MESI, 1990. 102 p.

3. It can be used to make it easier to change the cost of the product. Http://www.doklad.ru/view/WDusdgI-tCE.html

4. *Kuldin S. P.* Genetic development with the quality requirements // Applied Informatics. 2010. №5. 30-42 p. URL: https://cyberleninka.ru/ article / n / geneticheskiy-podhod-k-probleme-otsenki-sro kov-i-trudoemkosti-razrabotki-programmnogo-obespe cheniya-s-zadannymi-trebovaniyami-k.

5. *Sidorov N. A., Batsenko D. V., Vasilenko Yu. N., Shchebetin Yu. V., Ivanova L. N.* Methods and tools for estimating the cost of software. Collection of scientific works "problems of system benefits in economics". NAU 2004. №7. 113-118 p.

6. *Arkhipenkov S.* Lectures on software project management [Electronic resource] access mode - http://citforum.ru/SE/project/arkhipenkov_lectures /

7. *Bitkovsky D. I., Motorko A. V.* Application of the COCOMO model in the economics of software engineering // Economics and Business: Theory and Practice, no. 4-2, 2017, pp. 11-14.

8. *Fatrell R., Schafer D., Schafer L.* At the minimum cost: Trans. from English. M.: Williams, Moscow - St. Petersburg-Kiev, 2004, 1136 p.

9. COCOMO Assessment Tool [Electronic resource] - Access mode http://www.softstarsystems.com/index .html

10. *Sharman G.* Agile COCOMOII [Electronic resource] / *G. Sharman.* CSE Annual Research Review. 2003. March 17-21. access mode - http://sunset.usc.edu/events / 2003 / March_2003 / Agile_COCOMOII_ARR.pdf

11. Use Case Points [Electronic resource] http://www.bfpug.com.br/Artigos/UCP/Banerjee- UCP_ An_Estimation_Approach.pdf.

Поступила в редколлегию 21.06.2019

Рецензент: д-р техн. наук, проф. Шаронова Н.В.

Gruzdo Iryna, Candidate of Technical Sciences, Senior Lecturer of the Department of Software Engineering, Kharkov National University of Radio Electronics. Scientific Interests: aerospace engineering, cybernetic linguistics, models and methods of risk management, Soft-skils. Address: Ukraine, 61166, Kharkiv, Nauka Ave., 14, Phone/fax: +380577021446,
e-mail: irina.gruzdo@nure.ua.

Kyrychenko Iryna, Candidate of Technical Sciences, Assistant of the Department of Software Engineering, Kharkov National University of Radio Electronics. Scientific Interests: ERP systems, models and methods of risk management. Address: Ukraine, 61166, Kharkiv, Nauka Ave., 14, Phone/fax: +380503110317,
e-mail: iryna.kyrychenko@nure.ua.

Tereshchenko Glib, Graduate student of the Department of Software Engineering, Kharkov National University of Radio Electronics. Scientific Interests: blokchain technology, ERP systems, models and methods of risk management. Address: Ukraine, 61166, Kharkiv, Nauka Ave., 14, Phone/fax: +380675707102,
e-mail: hlib.tereshchenko@nure.ua.