Metrics Applicable for Evaluating Software at The Design Stage

Iryna Gruzdo[a], Iryna Kyrychenko[a], Glib Tereshchenko[a] and Nadiya Shanidze[b]

[a] *Kharkiv National University of Radioelectronics, Nauky Ave. 14, Kharkiv, 61166, Ukraine*
[b] *National Technical University "KhPI", Kyrpychova str. 2, Kharkiv, 61002, Ukraine*

Abstract

The paper reviewed and analyzed the existing metrics used in the software evaluation process at the design stage. Discussed issues related to the selection and accounting of indicators affecting the evaluation of software at the design stage in the management of software development. The study of the metrics used to perform the evaluation of software at the design stage. Find conclusions about the use of the metrics examined in management practice. Proved the need to study the metrics described in the Rada standards governing the process of developing software.

Keywords 1

Software, software evaluatıon, metrıcs, software qualıty, project type, software development management, complexıty, accountıng

1. Introduction

At the present stage of development of information technologies, both throughout the world and in Ukraine, the tasks related to software evaluation throughout the life cycle acquire particular relevance, and in particular, special attention is given to software quality assessment at the design stage, since it doesn't depend on only the quality, but also the risks of the project, as well as its cost.

The quality of a software product consists of a set of features and characteristics of software that can satisfy the needs and requests of interested parties.

It should be noted that the importance of each quality characteristic varies depending on the restrictions adopted in the project, 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. The quality of software is influenced by human, material, hardware, time resources, as well as restrictions are adopted within a specific type of project.

In view of the above, it can be concluded that a balance between a number of variable characteristics that affect the overall quality plays a significant role. Therefore, it is necessary to pay special attention to the accumulation of information and analysis of the interrelations of factors in the previously obtained results.

Also noteworthy is the number of problems associated with the success of the project as a whole; in most cases, success depends on the evaluation of the software at the design stage from those methods and metrics that were initially chosen and which gave them the approximate assessment to the realities of the developed project itself [4, 11].

The classical software evaluation process consists of three stages: the establishment (definition) of quality requirements, the preparation for assessment, and the assessment procedure. This process can be applied both at any phase of the life cycle and during the development of individual software components in order to understand how the development takes place.

COLINS-2021: 5th International Conference on Computational Linguistics and Intelligent Systems, April 22–23, 2021, Kharkiv, Ukraine
EMAIL: tigralwovna@gmail.com (I. Gruzdo); iryna.kyrychenko@nure.ua (I. Kyrychenko); hlib.tereshchenko@nure.ua (G. Tereshchenko); nashanidze@ukr.net (N. Shanidze)
ORCID: 0000-0002-4399-2367 (I. Gruzdo); 0000-0002-7686-6439 (I. Kyrychenko); 0000-0001-8731-2135 (G. Tereshchenko); 0000-0002-9613-186X (N. Shanidze)

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 [2] is given below.

- Algorithmic modeling.
- Expert estimates.
- Evaluation by analogy.
- Parkinson's law.
- Score to win the contract.

When using these methods in practice, you should consider the strengths and weaknesses of each of them. It is necessary to determine the appropriateness of their use depending on which of the methodologies is used for software development - Waterfall, Scrum, Kanban, Rational Unified Process, etc. It is also necessary to take into account the features that are inherent in one or another methodology chosen for project management.

Before proceeding to the selection or analysis of the project, it should be understood that although there are a number of methodologies, models and methods, they are in most cases divided into theoretical, practical, and biased in the process of classifications.

Depending on the size of the programmer of the company, it depends on what criteria they choose to evaluate the software at the design stage. But it should be noted that in their practice they do not use expensive and unconfirmed methods of software evaluation, and in a number of cases they use accumulated historical data to compare the complexity of the project with the complexity of previous projects of a similar type, size, orientation and human composition.

Therefore, by virtue of the above, this article will discuss the practical aspects that allow the evaluation of software at the design stage. Those metrics will be considered that allow to evaluate software not only for large firms but also for a group of people when working on a startup.

The purpose of the article is to analyze the existing practical software evaluation solutions at the design stage, review the most commonly used metrics aimed at increasing the quality of the software, and also justify the choice of using the metrics discussed in management practice.

As a result of solving the problem, the current state of the software evaluation problem will be reviewed at the design stage, as well as during the analysis, the most used metrics that have practical experience in development and are aimed at improving the quality of software throughout the life cycle will be considered.

2. Research problem statement

Currently, there are many metrics for evaluating software at the design stage, suitable for assessing the necessary resources that affect the quality of all software. In addition, classes of metrics are known that are aimed at increasing the quality of software and facilitating the software development process, therefore, having formulated a number of characteristic criteria, it is necessary to establish the suitability of each of the known software evaluation metrics to improve software quality at the design stage.

The solution to this problem involves the implementation of the following sequence of steps:

- Used metric classes that are aimed at increasing the quality of software.
- Review and analysis of existing metrics used in the evaluation process of software at the design stage.
- Analysis of the advantages and disadvantages, the provision of sound conclusions about the appropriateness of using the considered metrics in management practice.

3. Software evaluation metrics at the design stage

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.

Therefore, the most appropriate scheme is the one when the project manager, along with the architecture, the team lead, develops and tests [3]. In the process of analysis, perform several iterations in assessing the necessary resources affecting the quality of the entire software.

For each project, it is customary to calculate and take into account the following indicators:

- The number of people required to implement the software;
- Total labor costs (in person-months, person-hours);
- Program size (in thousands of source code lines);
- Development cost;
- Volume of documentation;
- Errors that will be detected during the year of operation;
- Development time.

To measure the above indicators and quality criteria in various literary sources, metrics are used.

The following metrics classes are used most often at the design stage for software evaluation:

- Statistical;
- Graphic;
- Estimated;
- Predictable;
- Next, we will consider some of the most commonly used metrics.

Halstead metrics [13, 14] are characteristics of programs, which are identified based on the static structure of the program in a specific programming language. It is based on counting the number of operators and operands used in the program. These metrics allow you to calculate

- program length;
- program size;
- evaluation of its software implementation;
- difficulty understanding software;
- coding complexity;
- level of language of expression;
- informational content;
- optimal modularity in software;
- forecast of system resources;
- prediction of the number of errors;
- overall complexity;
- connectivity;
- hybrid.

This metric is based on measurable program characteristics: the number of unique program statements (n_1); the number of unique operands (n_2); total number of operators (N_1); total number of operands (N_2).

After highlighting the main characteristics of the software, you can calculate such indicators as:

- alphabet (n) = n1+n2 ;
- experimental program length (Ne) = N1+N2 ;
- theoretical program length (N_T): = n1·log2(n1) + n2·log2(n2);
- program size (V) =Ne·log2(n);
- potential volume (V*) = (N1*+N2*) ·log2(n1* + n2*);
- program level (L) =V* / V (from 0 till 1);
- program complexity (S)=L-1;
- expected program level (L^): =(2/n1)·(n2/N2);
- program intelligence (I)=L^ · V;
- programming work (E)=V·S ≡ V/L;
- coding time (T) =E/St (St – Stroud number from 5 to 20);
- expected coding time (T^) =n1·N2 · N·log2(n) / (2·St·n2);
- Programming language level (Lam): =(V*) ·(V*)/V;
- Mistakes level (B)=V / 3000.

With the help of this metric, you can describe the potential volume of the program, the corresponding most compactly implementing the algorithm, you can calculate the programming time, the level of the program (structure and quality). It can also be used to assess the complexity of intermediate development products.

Cyclomatic complexity of McCabe (McCabe's cyclomatic number) [8, 13] – characterizes the complexity of the testing program. This metric is based on the calculation of the cyclomatic number and the cyclomatic complexity of the software. The complexity of the program control flow is calculated on the basis of the control flow graph of the program. This graph is constructed as a directed graph, in which computational operators or expressions are represented as nodes, and the transfer of control between nodes is represented as arcs.

To calculate the cyclomatic McCabe number Z(G), apply the formula

$$Z(G) = l - v + 2p, \tag{1}$$

where l is the number of arcs of a directed graph G;
v is the number of vertices;
p is the number of connected components of the graph.

Moreover, the number of connected components of a graph is considered as the number of arcs that must be added to convert the graph into a strongly connected one.

The advantages of this metric are in the simplicity of calculations and the ability to repeat the results, as well as visibility. It also allows not only to make an assessment of the complexity of the implementation of individual elements of a software project and adjust the overall indicators for assessing the duration and cost of the project, but also to assess the associated risks and make the necessary management decisions. The disadvantages include: insensitivity to the size of the program, insensitivity to changes in the structure of the program, no correlation with the structure of the program, no difference between the structures.

Metric T. Jilba [12] allows you to determine the logical complexity of the program using the expressions IF_THEN_ELSE, for this purpose, the absolute complexity of the program (CL), characterized by the number of condition operators; cl is the relative complexity of the program, characterized by the saturation of the program with conditional operators, i.e. cl is defined as the ratio of CL to the total number of operators.

Allows you to evaluate:

- the number of operators L1oop cycle;
- the number of conditional operators LIF;
- the number of modules or subsystems L mod;
- the ratio of the number of links between modules to the number of modules f = N4SV / L mod;
- the ratio of the number of abnormal outputs from the set of operators to the total number of operator’s f * = N*SV / L.

With the help of the Jilba metric, you can perform an analysis of cyclic constructions You can also analyze the relationship between the number of variables in the program and the number of calls to them with the complexity of the program itself. This metric also allows you to improve such quality indicators as: internal elasticity; openness (adaptability); tolerance to changes in the system; universality; convenience of transfer; comparability.

The complexity of the program by the method of boundary values [12, 14]. This metric is based on the calculation of values using an oriented graph of a program with a single initial and only final vertices G = (V, E). In the graph under consideration, the number of arcs entering a vertex is called the negative degree of the vertex, and the number of arcs emanating from the vertex is called the positive degree of the vertex. The set of vertices of the graph consists of vertices with a positive degree <= 1 – receiving vertices and vertices with a positive degree >= 2 – selection vertices.

The graph G is divided into the maximum number of subgraphs G 'that satisfy a number of conditions: the entrance to the subgraph is made only through the selection peak; each subgraph includes a vertex (called the lower boundary of the subgraph) that can be reached from any other vertex of the subgraph.

Each receiving vertex has a corrected complexity equal to 1, except for the final vertex, the corrected complexity of which is 0. The corrected difficulties of all the vertices of the graph G are summed to form the absolute boundary complexity of the program. Further, the relative boundary complexity of the program is determined:

$$S_0 = 1 - (v - 1)/ S_a , \tag{2}$$

where S_0 – relative boundary complexity of the program;
S_a – absolute boundary complexity of the program;
v – the total number of vertices of the program graph.

This metric allows you to determine the boundary values of these functions, the complexity of the program and to plan actions in case of going beyond the permissible boundaries.

Metrics of data flow complexity [12]. The main calculations are based on the data "module – global variable (p, r)", where p is the module that has access to the global variable r. Depending on the presence in the software of the actual access to the variable r, two types of "module-global variable" pairs are formed: actual and possible. A possible reference to r with p shows that the region of existence of r includes p.

The ratio of the number of actual references to possible is determined by the formula

$$R_{up} = A_{up}/P_{up} , \tag{3}$$

where A_{up} value – how many times the U_p modules actually accessed global variables, and P_{up} - how many times they could get access.

This metric demonstrates the approximate probability of a link of an arbitrary module to an arbitrary global variable. In this case, the higher the probability, the higher the probability of an "unauthorized" change in any variable, which can significantly complicate the work associated with the modification of the program.

Metrics of Pivovarsky [12] – aimed at assessing the complexity of the program differences not only between sequential and nested control structures, but also between structured and unstructured programs. The metric can be calculated by the formula

$$N(G) = n * (G) + S\,P_i , \tag{4}$$

where n *(G) – is the modified cyclomatic complexity is taken into account in that the CASE operator with n-outputs is treated as one logical operator, and not as n - 1 operators. P_i is the nesting depth of the i - that predicate vertex.

To calculate the depth of nesting of predicate vertices, the number of spheres of influence is required. It should be borne in mind that the depth of nesting increases due to the nesting not of the predicates themselves, but of the spheres of influence. This metric is well used in the case of transition from sequential to nested programs and further to unstructured ones.

Metrics of the Chepin program complexity [12, 14]. The calculation of the metrics is based on an assessment of the informational strength of a single program module using the analysis of the nature of using variables from the input-output list.

In the course of the analysis, the entire set of variables that make up the I / O list is divided into functional groups: P - input variables for calculations and for providing output. M - variables modified or created within the program. C - variables involved in the management of the program module (control variables). T – "parasitic" variables not used in the program. Such variables do not directly participate in the implementation of the information processing process for which the analyzed program is written, however they are stated in the program module.

The initial expression for determining the Chepin metric is as follows:

$$Q = a_1P + a_2M + a_3C + a_4T , \tag{5}$$

where a_1, a_2, a_3, a_4 – weight coefficients.

The weight coefficients are used to reflect the different effects on the complexity of the program of each functional group.

The Chepin metric is based on an analysis of the source code of the programs, which provides a unified approach to the automation of the calculation and can be calculated using specially developed software. You can also get an estimate of reliability using weights. The main problem is to obtain these factors, which are calculated based on the experience of previous projects. With the rapid change of technology, methods and design tools, the experience of previous projects is inappropriate to use and as a result it will give incorrect estimates.

MacClure metric [9, 12]. Allows you to assess the complexity of the software, using data on the number of possible ways to run the program, the number of control structures and variables.

First, the complexity of the function C (i) is calculated by the control variable i:

$$C(i) = (D(i) * J(i))/n \ , \tag{6}$$

where D (i) is a value that measures the scope of the variable i. J (i) is a measure of the complexity of the interaction of modules in terms of the variable i;
n is the number of individual modules in the partitioning scheme.

Next, determine the value of the complexity of the functions M (P) for all modules

$$M(P) = fp * X(P) + gp * Y(P) \ , \tag{7}$$

where fp and gp are the number of modules immediately preceding and immediately following the module P;
X (P) is the difficulty of accessing the module P, Y (P) is the complexity of controlling the call from the module P of other modules.

After that, the total complexity MP of the hierarchical scheme of program partitioning into modules is calculated according to all possible values of the P - program modules

$$MP = SUMM(M(P)) \, . \tag{8}$$

When calculating, it is necessary to take into account that in each module there is one entry point and one exit point, the module performs exactly one function, and the modules are called according to a hierarchical control system that defines the call relationship on multiple program modules.

The MacClure metric is designed to manage the complexity of structured programs in the design process. In most cases, this metric is well applied to hierarchical schemes for dividing programs into modules; this allows you to choose a dividing scheme with less complexity long before writing a program. The metric demonstrates the dependence of the program's complexity on the number of possible execution paths, the number of control structures and the number of variables (on which the choice of the path depends).

The Berlinger metric [13] is a measure of the complexity of information theory based on the frequency of program symbols in a program. The measure of complexity is calculated as

$$I(R) = m\,(F * (R) * F - (R))2\, . \tag{9}$$

The disadvantage of this metric is that a program containing many unique characters, but in small quantities, will have the same complexity as a program containing a small number of unique characters, but in large quantities. Another of the problems of applying this metric is to determine the values of the characteristics and determine the probability of each value

The Cocolt metric [16] takes one metric as a basis, and also takes into account other metrics that should influence the final result. It is defined as follows:

$$H\_M = (M + R_1 * M(M_1) + \ldots + R_n * M(M_n)/(1 + R_1 + \ldots + R_n) \ , \tag{10}$$

where M is the base metric;
M_i are other interesting measures;
R_i are correctly chosen coefficients;
M (M_i) are functions.

The functions M (Mi) and the coefficients Ri are calculated using regression analysis or task analysis for a specific program.

The metric of Cocolt allows to receive the general numerical value for a set of metrics taking into account the weighed coefficients. It also allows in a numerical form to get the normalized characteristics of the groups of characteristics that reflect the quality of the program code.

ABC-metric (Fitzpatrick) [6] inventory used to classify and prioritize resource allocation. This metric is based on the counting of assignments of variables (Assignment), explicit control transfers beyond the scope, i.e. function calls (Branch), and logical checks (Condition).

It is determined based on three different indicators ABC =. The first indicator n_a (from the English Assignment) is highlighted under the lines of code that correspond to the assignment of variables to a certain value, for example, int number = 1. The indicator n_b (from the English Branch) is responsible for the use of functions or procedures, that is, operands that work outside software visibility. The indicator n_c (from the English Condition) counts the number of logical operands such as conditions and loops. The metric value is calculated as the square root of the sum of the squares of the values n_a, n_b, and n_c.

$$F = \sqrt{n_a^2 + n_b^2 + n_c^2} \ . \tag{11}$$

The metric is easily calculated for different code fragments and is visual. The ABC metric can be used to estimate the size and complexity of the fragments of the analyzed application, as well as for automatic search.

Myers interval metrics (Myers) [5] calculate the complexity of the program based on the strength of the individual modules of the program and the coupling between each pair of modules:

$$\mathrm{D}_{\mathrm{ij}} = \begin{cases} 0.15 \cdot (S_i + S_j) + 0.7 \cdot C_{ij}, \text{ if } C_{ij} \neq 0 \\ 0, \text{ if } C_{ij} = 0 \\ 1, \text{ if } \mathrm{i} = \mathrm{j} \end{cases} , \qquad (12)$$

where D_{ij} is the probability that module j will have to change when module i changes, if we consider modules i and j outside the context of the entire program (the relationship is considered symmetrical);
S_i and S_j are the strengths of these modules i and j;
C_{ij} is the coupling of modules i and j.

This metric allows you to distinguish between different software complexity or functions, but it is very rarely used in Ukraine, and it has proven itself well in America. As applied to code analysis, the Myers metric does not have any noticeable advantages over simple cyclomatic complexity.

Hanson Metric [12]. Allows you to estimate the complexity of a program or function using betting values - cyclomatic complexity and the number of operators (A, B), where A is the McCabe metric, B is the number of executable operators. This increases the sensitivity of the metrics to the structure of the program.

Sometimes in practice, this metric is used to assess the complexity of analyzing a binary code in a situation where the analyst knows the approximate size of the entire software and the number of instructions in its component functions.

The Schneidewind metric [10] is expressed in terms of the number of possible paths in the control flow graph. This metric is based on the approach that the later the errors occur, the more important they are for the process of predicting errors in the program.

Assumptions are based on the fact that there are m testing intervals, and presumably fi errors are found on the i-th interval.

This metric is appropriate to use in cases where data from all testing intervals are needed to predict the future state of the software. Or in the case when there was a certain change in the process of error detection, and only the data of the last m - (s-1) intervals makes sense to take into account in the forecasts for the future. It should be borne in mind that this metric does not take into account the complexity of the unfulfilled branches of the program.

The Kafura metric (Henry & Kafura) [15] is based on the concept of information flows (also found under the name “Fan in / out complexity”). This metric allows the evaluation of local and global information flows, an assessment of the information complexity of the procedure and module.

A local information flow from A to B exists if: module A calls module B (direct local flow); module B calls module A and A returns B the value used in B (indirect local flow); module C calls modules A, B and transfers the result of module A to B.

The global information flow from A to B through the global data structure D exists if module A places information in D, and module B uses information from D. On the basis of these concepts, the quantity I is entered - the information complexity of the procedure:

$$I = length * (fan\_in * fan\_out)\text{^}2 \ , \qquad (13)$$

where length is the complexity of the procedure text (measured using the Halstead, McCabe, LOC, etc. metrics);
fan_in - the number of local streams inside the procedure plus the number of data structures from which the procedure takes information;
fan_out is the number of local streams from the procedure plus the number of data structures that are updated by the procedure.

You can define the informational complexity of a module as the sum of the informational complexities of its member procedures. Next, the informational complexity of the module is calculated relative to some data structure:

$$J = W * R + W * WrRd + WrRdxR + WrRd * (WrRd - 1) \ , \qquad (14)$$

where W is the number of procedures that only update the data structure;

R – only read information from the data structure;
WrRd – both read and update information in the data structure.

Kafur metric allows you to take into account the complexity of the text. Allows you to determine the information complexity of the module given the number of elements and data structures from which the module takes information and which are updated by the module, respectively.

The Oviedo metric [7]. In this case, the program is divided into linear non-intersecting segments – rays of operators that form a graph of the control flow of the program. The complexity metric of each ray is defined as the sum of the quantities of the determining occurrences for each variable used in the ray.

There is also an interpretation of the Oviedo metric: $C = aCF + bDF$, where CF is the complexity of the control flow, taking into account only the number of arcs of the graph; DF – the complexity of the data stream, calculated as the sum of the complexity of the data streams of the basic blocks of the program, and the complexity of the data stream of a block is the number of variables that are used but not defined in the block; a, b – weights that can be taken equal to 1.

The author of the metric assumes that it is easier to find the relationship between definitions and uses of a variable inside a ray than between rays, and that the number of different defining occurrences in each ray is more important than the total number of occurrences of the variable in each ray. The Oviedo measure can be considered as a transition from “pure” metrics to a combined data stream because it takes into account the flow of control and increases with the number of “inter-block links” when a variable is defined and used in different basic blocks of the program.

As can be seen from the general analysis of metrics, not all of them allow an even and unambiguous assessment: labor-intensive; labor costs; total time to create a program; the number of people needed to work on the software; development period; cost and costs of the project stages; project risks; functional fitness; the complexity of the execution of the software; depth of the inheritance tree; software size; total number of objects; average labor productivity; scope of testing and cost of testing; popularity of the technologies used. Therefore, it is advisable to continue the study in order to find answers to questions related to software evaluation which should facilitate the software evaluation stage at the design stage.

4. Analysis software evaluation metrics at the design stage

Based on the foregoing, it is further advisable to perform a more complete analysis of software assessment metrics at the design stage. The result of the analysis is shown in Table 1, which shows the calculated criteria of the considered metrics and implicit criteria that allow you to select the necessary criteria for evaluating the quality of software within the metric. Table 2 shows the advantages and disadvantages of each considered metric used to evaluate software at the design stage.

Table 1
Explicit and implicit characteristics of metrics that allow software quality assessment

№	Metrics	Explicit	Implicit
1	Halstead Metrics	- program length; - program volume - assessment of its implementation; - the difficulty of understanding it; - the complexity of coding; - level of language of expression; - informational content;- optimal modularity.	- the time necessary for a person to perform the elementary difference of objects; - program level - characterizes the efficiency of the implementation of the algorithm relative to memory costs; - programming work; - determination of the number of modules in the program; - to predict probable errors in the program for necessary work; - average number of differences between possible programming errors; - assessment of intellectual efforts;

			- level of programming quality; - the complexity of understanding the program; - the complexity of coding software; - language level - this characteristic allows you to determine the mental costs of creating software; - required solutions when writing software.
2	McCabe Cyclomatic Complexity (McCabe Cyclomatic Complexity)	- the complexity of the program control flow; - graph of the control logic of the program; - cyclomatic complexity.	- the complexity of the program; - logical complexity of the program; - criterion for covering all branches of the program; - the number of tests sufficient for testing; - assessment of the complexity; - the feasibility of individual software elements; - to predict probable errors in the program for necessary work; - the duration and cost of the project; - assess the associated risks; - determination of the most complex, high risks, on the basis of which take measures to eliminate risks by making adjustments.
3	Metric T. Jilba	- the number of loop operators; - the number of condition operators; - the number of modules or subsystems; - the ratio of the number of connections between modules to the number of modules; - the ratio of the number of abnormal exits from the set of operators to the total number of operators; - logical complexity of the program; - the absolute complexity of the program; - the relative complexity of the program.	- assessment of the cost of software at the initial stages of design; - the complexity of development; - the complexity of understanding the program; - the difficulty of writing a program; - general software complexity; - assessment of the complexity; - software reliability; - the maximum level of nesting of conditional and cyclic operators; - to predict probable errors in the program for necessary work; - internal elasticity; - openness (adaptability); - tolerance to changes in the system; - universality; - ease of transfer; - comparability; - determination of the most complex, high risks, on the basis of which take measures to eliminate risks by making adjustments.

4	The complexity of the program according to the method of boundary values (boundary value)	- relative marginal complexity of the program; - absolute boundary complexity of the program; - the boundary complexity of the program; - the total number of vertices of the program graph.	- plan actions in case of exceeding permissible boundaries; - evaluate implemented functionality; - the complexity of the processes; - to predict probable errors in the program for necessary work; - openness (adaptability); - tolerance to changes in the system; - ease of transfer; - determination of the most complex, high risks, on the basis of which take measures to eliminate risks by making adjustments.
5	Data Flow Complexity Metric	- informational complexity of the module relative to some data structure; - analysis of static data flow (Static Data Flow Analysis); - Cross Reference Analysis; - analysis of information flow (Information Flow Analysis); - cohesion of the data structure; - complicate the work associated with the modification of the program.	- perform analysis of data streams; - the duration of the program and the level of data nesting; - evaluate the integrity of software modules; - the complexity of understanding the program; - the difficulty of writing a program; - general software complexity; - determination of the complexity of the work associated with the modification of the program; - evaluate implemented functionality; - identification of hidden dependencies in the software and convert them into an explicit form, thereby simplifying the program logic.
6	Pivovarsky Metrics	- a modified cyclomatic measure of complexity; - general assessment of software complexity; - nesting depth.	- the complexity of the program; - criterion for covering all branches of the program; - the number of tests sufficient for testing; - assessment of the complexity; - the feasibility of individual software elements; - to predict probable errors in the program for necessary work; - internal elasticity; - openness (adaptability); - tolerance to changes in the system; - universality; - ease of transfer; - comparability.
7	Chepin's complexity metrics	- a measure of the difficulty of understanding programs based on input and output data; - assessment of the information strength of a single software module;	- the complexity of the program; - criterion for covering all branches of the program; - the number of tests sufficient for testing; - assessment of the complexity; - the feasibility of individual software elements;

		- the complexity of the program of each functional group.	- to predict probable errors in the program for necessary work; - tolerance to changes in the system; - ease of transfer; - reliability assessment.
8	McClure Metric	- the complexity of the interaction of the modules; - complexity of functions; - software complexity; - the number of possible ways to execute programs; - the number of control structures and variables.	-the degree of standardization of interfaces (communication commonality); - functional completeness (completeness); - the feasibility of individual software elements; - to predict probable errors in the program for necessary work; - the uniformity of the used design rules and documentation (consistency); - error tolerance; - performance efficiency; - expandability (expandability); - independence from the hardware platform (hardware independence); - completeness of logging errors and other events (instrumentation); - modularity; - convenience of work (operability); - security (security); - simplicity of work (simplicity); - ease of learning (training).
9	Burlinger Metric	- the complexity of the program.	- assessment of the possibility of software to transition from sequential to embedded programs; - assessment of the relationship of elements within the program; - predict hidden dependencies and convert them into an explicit form; - the number of tests sufficient for testing; - assessment of the complexity; - the feasibility of individual software elements; - to predict probable errors in the program for necessary work; - the ability to simplify the logic of the program.
10	Bell Metric	- program length; - program volume - assessment of its implementation; - the difficulty of understanding it; - the complexity of coding; - level of language of	- classification of applications according to complexity of analysis; - assessment of labor costs; - determination of the number of tests sufficient for testing; - assessment of the logical complexity of the program; - identification of the most complex, high

		expression; - informational content; - optimal modularity; - cyclomatic complexity.	risks and take measures to eliminate risks by making adjustments; - the time necessary for a person to perform the elementary distinguishing of objects; - the effectiveness of the implementation of the algorithm relative to memory costs; - evaluation of programming work; - errors in the program; - the average number of elementary differences between possible programming errors; - assessment of intellectual efforts - level of programming quality; - the complexity of understanding the program; - the complexity of coding the program; - level of language of expression; informational content of the program (this characteristic allows you to determine the mental costs of creating the program); - assessment of intellectual efforts in software development.
11	ABC Metric (Fitzpatrick) [6]	- software size; - assessment of software complexity; - accumulation of technical debt of the application; - analysis of duplicate code fragments; - finding errors; - distribution of resources; - distribution of indirect costs.	- assessment of the size and complexity of software fragments; - the complexity of development; - the complexity of understanding the program; - the difficulty of writing a program; - general software complexity; - to predict probable errors in the program for necessary work; - tolerance to changes in the system; - universality; - determination of the most complex, high risks, on the basis of which take measures to eliminate risks by making adjustments.
12	Myers Interval Metric (Myers)	- cyclomatic measure, - the number of individual conditions.	- the complexity of the program; - logical complexity of the program; - the number of tests sufficient for testing by the criterion of coverage of all branches of the program; - the number of tests sufficient for testing; - assessment of the complexity; - the feasibility of individual software elements; - duration and cost of the project; - assess the associated risks; - determination of the most complex, high risks, on the basis of which take measures to eliminate risks by making adjustments.

13	Hanson Metric	- a pair (cyclomatic number, number of operators); - there are errors; - prediction of errors in the program; - testing intervals.	- the complexity of the program; - the logical complexity of the program; - the number of tests sufficient for testing by the criterion of coverage of all branches of the program; - the number of tests sufficient for testing; - assessment of the complexity; - the feasibility of individual software elements; - assess the associated risks; - determination of the most complex, high risks, on the basis of which take measures to eliminate risks by making adjustments.
14	Kafura Metric (Henry & Kafura)	- informational complexity of the procedure; - the complexity of the text of the procedure; informational complexity of the module; - level of commenting on the program; - the overall complexity of the structure.	- evaluate implemented functionality; - the complexity of the processes; - assessment of the relationship of elements within the program; - Predict hidden dependencies and convert them into an explicit form; - the number of tests sufficient for testing; - assessment of the complexity; - the feasibility of individual software elements; - to predict probable errors in the program for necessary work; - the ability to simplify the logic of the program.
15	Oviedo Metric	- complexity of the control flow; - the complexity of the data stream; - base blocks of the program; - weighting factors; - program complexity.	- tests to achieve an acceptable level of code coverage; - assessment of the relationships between subtasks; - evaluate the integrity of software modules; - the complexity of understanding the program; - the difficulty of writing a program; - general software complexity; - determination of the complexity of the work associated with the modification of the program; - evaluate implemented functionality; - identification of hidden dependencies in the software and convert them into an explicit form, thereby simplifying the program logic; - determination of the most complex, high risks, on the basis of which take measures to eliminate risks by making adjustments.

Table 2
Advantages and disadvantages of software evaluation metrics at the design stage

№	Metrics	Advantages	Disadvantages
1	Halstead Metrics	- Allows, in numerical form, to evaluate the potential volume of the program, corresponding to the most compactly implementing algorithms. - Allows you to calculate the approximate programming time, program level. - It can be used to assess the complexity of intermediate development elements. - You can evaluate the quality of the development process based on the degree of expansion of the text relative to the potential volume. - Allows you to determine the mental costs of creating a program and evaluate the necessary intellectual effort. - Evaluation does not depend on the language of software implementation. - There is no need to completely recount all indicators when translating a program from one language to another.	- The need for the availability of the source code of the software. - The characteristics of reliability, functionality is not taken into account. - There is a limitation that the length of a correctly compiled program should not deviate from the theoretical program length by more than 10%. - When determining the mental costs of creating a program, debugging work, which also requires intellectual costs, is not taken into account.
2	McCabe Cyclomatic Complexity (McCabe Cyclomatic Complexity)	- The simplicity of the calculations. - Opportunities for repeatability of results. - Visibility. - Allows you to evaluate the complexity of the implementation of individual software elements. - Ability to adjust the overall indicators for assessing the duration and cost of the project. - Allows you to assess the associated risks and make the necessary management decisions. - The indicator of cyclomatic complexity can be calculated for different structural units of software (module, method, etc.).	- Insensitive to program size. - Lack of correlation with the structure of the program. - Lack of distinction between designs. - When calculating cyclomatic complexity, logical operators are not taken into account - Insensitivity to changes in program structure. - Representation of the same graphs may have predicates of completely different complexity - When calculating the logical complexity of software, the choice of data structures, algorithms, variables or comments is not taken into account. - It is intended only for evaluating programs developed in accordance with certain

			requirements for a programming style. - Not suitable for building a software complexity profile.
3	Metric T. Jilba	- Ability to perform analysis of cyclic structures. - You can analyze the relationship between the number of variables in the program and the number of calls to them with the complexity of the program itself. - Allows you to increase such quality indicators as: internal elasticity; openness (adaptability); tolerance to changes in the system; universality; ease of transfer; comparability. - Allows you to evaluate the relative complexity to build a profile of complexity. - Allows you to control the program development process from TK to trial operation.	- The need for the availability of the source code of the software. - The characteristics of reliability, functionality is not taken into account. - Not suitable for building a software complexity profile.
4	The complexity of the program according to the method of boundary values (boundary value)	- It makes it possible to determine the boundary values of the complexity of the program. - Allows you to plan actions in case of exceeding the permissible limits of complexity. - Allows you to evaluate in different ways those implementing the same functionality.	- Requires a certain degree of creativity and specialization in the task at hand. - Determining the boundaries for the task is a time-consuming process. - Lack of verification of the combination of input values.
5	Data Flow Complexity Metric	- Ability to perform data flow analysis. - It takes into account the duration of the program and the level of data nesting. - Allows you to evaluate the integrity of software modules. - It makes it possible to determine the complexity of the work associated with the modification of the program. - Allows you to identify hidden dependencies in the program and convert them into an explicit form, thereby simplifying the program logic.	- Sometimes a metric demonstrates the approximate probability of an arbitrary module referencing an arbitrary global variable. - The subjectivity of some criteria for choosing routes. - The need for templates for control structures. - It must be taken into account that the higher the probability of linking an arbitrary module, the higher the likelihood of an “unauthorized" change in any variable, which in turn can significantly complicate the work associated with modifying a program.

6	Pivovarsky Metrics	- Allows you to evaluate software in the event of a transition from sequential to embedded programs. - There is a possibility of transition from embedded software to unstructured.	- A reliable number of spheres of influence is necessary in order to calculate the nesting depth of predicate vertices. - It is necessary to monitor that the nesting depth increases due to the nesting of predicates themselves, but of spheres of influence.
7	Chepin's complexity metrics	- Provides a unified approach to calculation automation. - Facilitates the process of software automation. - Allow you to calculate reliability using weights.	- Based on an analysis of the source code of programs. - Definition of weights, which are calculated based on the experience of previous projects. - It is inexpedient to use when rapidly changing technologies, methods and design tools (demonstrates incorrect estimates).
8	McClure Metric	- Allows you to manage the complexity of structured programs in the design process. - Allows you to evaluate hierarchical schemes based on the division of programs into modules, which in turn allows you to choose a partition scheme with less complexity long before writing a program.	- It is necessary to take into account that in the calculations this metric is based on the fact that in each module there is one entry point and one exit point, the module performs exactly one function, and the modules are called in accordance with the hierarchical control system, which sets the call ratio on many program modules. That is, for complex elements, its use is not advisable. - Demonstrates the dependence of program complexity on the number of possible execution paths, the number of control structures and the number of variables (on which the choice of path depends). - Each metric affects the assessment of several quality factors. - The numerical expression of the calculated factor is distributed differently for different organizations, development teams, types of software, processes used, etc. - Designed for programs that are well structured and composed of

			hierarchical modules that define the functional specification and management structure.
9	Burlinger Metric	- Allows you to calculate the complexity of the program. - Allows you to evaluate software in the event of a transition from sequential to embedded programs. - Allows you to evaluate the relationship of elements within the program. - Allows you to identify hidden dependencies in the program and convert them into an explicit form, thereby simplifying the program logic.	- The need for the availability of the source code of the software. - The need to consider that a program containing many unique characters, but in small numbers, will have the same complexity as a program containing a small number of unique characters, but in large numbers. - The choice of values of the characteristics and determination of the probability of each value is not unique, which in turn affects the indicators. - Depends on the probability of occurrence of the value.
10	Bell Metric	- Allows you to get the total numerical value for a set of - Empirical data on the relationship of the elementary measures used in previous projects should be taken into account. - The characteristics of reliability, functionality is not taken into account. - It is intended only for evaluating programs developed in accordance with certain requirements for a programming style. - It takes into account the experience of employees and their other qualities. - Insensitive to program size. - Dependence on the expert who makes the calculation.	- The need for the availability of the source code of the software. - Empirical data on the relationship of the elementary measures used in previous projects should be taken into account. - The characteristics of reliability, functionality is not taken into account. - It is intended only for evaluating programs developed in accordance with certain requirements for a programming style. - It takes into account the experience of employees and their other qualities. - Insensitive to program size. - Dependence on the expert who makes the calculation.
11	ABC Metric (Fitzpatrick) [6]	- Allows classification and prioritization of resource allocation. - Possibilities for repeatability of results. - Visibility. - Allows you to calculate the assignment of variable values, i.e. explicit transfers of control out of scope.	- May take a value of zero for some non-empty program units. - It does not depend on the selected period. - Sensitive to a small number of observations. - The data are very different from the results of other metrics when calculating.

		- Easy to calculate, can be calculated for different pieces of code. - It can be used to assess the size and complexity of fragments of the analyzed application, as well as for automatic search. - Allows you to take into account more cases of adding operators to the script text. - Approach management issues in terms of cost, quality and productivity of the actions performed, as well as assess the risks associated with them.	- High complexity and significant costs for the implementation of ABC in the enterprise.
12	Myers Interval Metric (Myers)	- Allows you to distinguish between different software or functions in complexity. - The simplicity of the calculations. - Opportunities for repeatability of results. - Visibility. - Allows you to evaluate the complexity of the implementation of individual software elements. - Ability to adjust the overall indicators for assessing the duration and cost of the project. - Allows you to assess the associated risks and make the necessary management decisions.	- With regard to code analysis, the Myers metric has no noticeable advantages over simple cyclomatic complexity. - Insensitive to program size. - Lack of correlation with the structure of the program. - Lack of distinction between designs. - When calculating cyclomatic complexity, logical operators are not taken into account. - Requires additional analysis of each predicate to determine the number of variables on which it depends.
13	Hanson Metric	- Allows you to evaluate the complexity of a program or function using a bet of values - cyclomatic complexity and the number of operators. - Allow you to perform a structured program. - Sensitivity to software structuring. - Allows you to evaluate the complexity of binary code analysis in a situation where the analyst knows the approximate size of the entire software and the number of instructions in its constituent functions.	- Insensitive to program size. - Lack of correlation with the structure of the program. - Lack of distinction between designs. - When calculating cyclomatic complexity, logical. operators are not taken into account. - Insensitivity to changes in program structure. - It is intended only for evaluating programs developed in accordance with certain requirements for a programming style. Not suitable for building a software complexity profile

14	Kafura Metric (Henry & Kafura)	- Considers the complexity of the software text. - Allows you to determine the information complexity of the module given the number of elements and data structures from which the module takes information and which are updated by the module, respectively.	- The need to introduce the concepts of local and global flows in software, on which the assessment of information complexity depends. - Based on an analysis of the source code of programs - Definition of weights, which are calculated based on the experience of previous projects. It is inappropriate to use with a rapid change in technology, methods and design tools (demonstrates incorrect estimates).
15	Oviedo Metric	- Allows you to consider the control flow in different base blocks of the program. - Allows you to assess the associated risks and make the necessary management decisions - Allows you to evaluate the relationship of elements within the program. - Allows you to identify hidden dependencies in the program and convert them into an explicit form, thereby simplifying the program logic.	- The subjectivity of some criteria for choosing routes. - The need for templates for control structures. - Distortion of the control flow, which leads to an increase in the complexity of the vertices of the control flow graph, which does not allow to use the metric efficiently. - Control variables can also affect program control flow.

Thus, the solution of the same problem by different authors can lead to significant variations in the values of the metrics; therefore, it is advisable to use several assessment metrics for their subsequent comparison. Since the metrics complement each other and answer different questions in the software evaluation process, this allows you to achieve the required quality, taking into account the required and necessary criteria.

The results obtained will allow us to further develop a generalized classification model of metrics used to evaluate software, taking into account the type of software, selected project management functions based on a hierarchical quality model and highlighted advantages and disadvantages of software assessment metrics at different stages of software development. In turn, the introduction and use of such a classification of metrics will not only improve control over the software development process, but also more consciously carry out their selection and use at various stages of the LC software, thereby improving the quality of the final product and facilitate the process of choosing the necessary metrics.

5. Conclusions

In the course of this work:

- the analysis of the current state of the problem of the applied software assessment metrics was performed at the design stage, which are currently used in development management;
- the importance of performing a preliminary assessment of the software was determined before embarking on its implementation;
- review of the most used software evaluation metrics at the design stage was made.

As a conclusion throughout the work, it can be said that software evaluation at the design stage has a high practical application, since allows you to perform a software assessment 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 needed and what the future cost of the project. It should be noted that practically on all software development platforms there are tools for evaluating most of the reviewed software evaluation metrics at the design stage.

During the analysis, it was found that not a single universal metric exists. Any controlled metric characteristics of the program should be controlled either depending on each other, or depending on the specific task. It should be noted that any metric is only an indicator that depends heavily on the language and programming style; therefore, no measure can be raised to an absolute and any decisions can be made based only on it.

It should be remembered that there exist and apply in practice standards that support the metric assessment of the quality and reliability of software, including the regulatory documents of the IEEE 982 series, ISO / IEC 9126, DSTU 28195, RUP.

When evaluating software at the design stage, it is advisable to use several assessment metrics for their subsequent comparison to improve the quality. Since the metrics complement each other and answer different questions in the software evaluation process, this allows you to achieve the required quality with the necessary criteria. 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, it is necessary to use additional information, or choose more indicative criteria, after which it is necessary to 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 the analysis of the used statistical metrics of software evaluation used in the development of software projects throughout the life cycle of software development.

6. References

[1] A. A. Platonov, V. I. Timofeev, Integrity Monitoring of Dynamic Objects of Computing Systems Using Metric Standards, volume 38, 2015, pp. 136–160. doi: 10.15622/sp.38.8.

[2] P. E. Efimova, Ensuring the quality of management decisions when designing instrumentation on the basis of a comprehensive mathematical model of the process, Ph.D. thesis, Rybinsk, 2011.

[3] A. Wasif, Metrics in Software Test Planning and Test Design Processes, Ronneby, 2018.
G. M. Muketha, Metrics and Models for Evaluating the Quality and Effectiveness of ERP Software, in: G. M. Muketha (Eds.), Advances in Systems Analysis, Software Engineering and High Performance Computing, 1st ed., 2019.

[4] O. V. Kazarin, Reliability and security of software: a textbook for undergraduate and graduate studies, Publishing House Yurite, Moscow, 2019. ISBN 978-5-534-05142-1. URL: https://urait.ru/bcode/441287 (case date: 15.02.2021).

[5] K. E. Serdyukov, Study of methods for assessing the complexity of program code when generating input test data, Proceedings of the Information technologies and nanotechnologies (ITNT-2020), volume 4. Data sciences, Publishing House Samara, 2020, pp. 662–671.

[6] I. Ledovskikh, Code complexity metrics: Technical Report 2012-12, p. 22, 2012. URL: http://www.ispras.ru/preprints/docs/prep_25_2013.pdf .

[7] K. Smelyakov, A. Datsenko, V. Skrypka, A. Akhundov, Efficiency of Image Reduction Algorithms with Small-Sized and Linear Details, in: Proceedings of the 2019 IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology, PIC S&T'2019, Kyiv Ukraine, 2019, pp. 745-750.

[8] K. Smelyakov, O. Ponomarenko, A. Chupryna, D. Tovchyrechko, I. Ruban, Local Feature Detectors Performance Analysis on Digital Image, in: Proceedings of the 2019 IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology, PIC S&T '2019, Kyiv Ukraine, 2019, pp. 644-648.

[9] D. A. Mayevsky, E. Yu. Mayevskaya, A. A. Orekhova, A. Yu. Krivtsov, V. S. Kharchenko, Green software, in: Proceedings Workshop, National Aerospace University named after N.E. Zhukovsky "KHAI.", Kharkov, 2015.
[10] M. Frappier, Software Metrics for Predicting Maintainability, 2014.
[11] Models and metrics of software quality assessment, 2020. URL: http://www.met-rix.narod.ru/page2.htm.
[12] A. V. Averyanov, I. N. Koshel, V. V. Kuznetsov, Application of Halstead metrics for quantitative estimation of computer program characteristics, Journal of Instrument Engineering, volume 62, N 11, 2019, pp. 970-975 (in Russian).
[13] N. Sharonova, O. Kanishcheva, Image and video tag aggregation, in: CEUR Workshop Proceedings, 2017, pp. 161-172.
[14] Y. I. Hrytsiuk, O. T. Andrushchakevych, Means for determining software quality by metric analysis methods, Scientific Bulletin of UNFU, 28(6), 2018, pp.159-171. doi:10.15421/40280631.
[15] A.V. Tsarevsky, A priori estimation of information processing algorithms by topological metrics of complexity, Automation of control processes. 1, 2012, pp. 95–101.
[16] S. K. Chernonozhkin, Measures of program complexity (review). System Informatics, Novosibirsk: Science, Issue 5: Architectural, formal and software models (2019) 188-227. URL: https://www.elibrary.ru/item.asp?id=26865926.