DOI https://doi.org/10.36059/978-966-397-107-0/5-35

SOFTWARE TOOLS

Kyselov V. B.

1. Engineering interpretation of programming

Engineering (industrial) methods, although are not universal, are now widespread in programming. They require a revision of the traditional concepts and the development of new ones.

Let us consider, first of all, the evolution of the concept «Program». Any computer while solving a specific problem works on a specific program. In general, a computer program (P) is a record of an algorithm for solving a problem in the form of a sequence of commands or operators in a strictly formalized language accessible for the computer. This definition of the program is to some extent abstract and not sufficiently specific. It is used when we are not interested in the specific form of the program, the language of presentation, the degree of completion[1].

The program can be recorded on ordinary sheets of paper, on special forms, on data carriers. It may be debugged or undebugged, ready for its intended use, or may require some conversion before use. To make the idea of the program more specific, epithets are added to the name of the program, such as debugged (undebugged), object, controlling, etc. The abstractness of the term «Program» creates great difficulty in communicating of professionals, and in some cases is unacceptable. Therefore, there is a need for additional terms and definitions.

Under, *the software tool* (ST) one understands a program or a set of programs on data carriers with program documentation developed in accordance with standards and other regulatory documents and suitable for its intended purpose. The definition of the software emphasizes the completeness of the product (the availability of software documentation) and the readiness to use it directly for its intended purpose to solve a specific problem on the computer (recording on a data carrier). But the software may not be a commodity product, especially a product for production and technical purpose. It may only be intended for use by the

[1] Antipensky V.E., Bilousko V.S., Chujdan T.I. Computing Machines and Programming: Workshop. Kyiv: Higher School. Main issue, 1987. 245 p.

developer himself and is not used by others. In this regard, there is a need to allocate a subset of software tools which have features of products that are developed and manufactured to meet the needs of the national economy, population and programs on data carriers with program documentation developed and manufactured in accordance with standards and other regulatory documents, having undergone state, inter-departmental or departmental testing and technical control of the manufacturer, provided with guarantees. A software product is a unit of software production for technical purposes. It follows from the definition that not every software tool is a software product.

The relation between programs, software tools and software products can be established using the concept of sets. Let us suppose A = {P}; B = {ST}; C = {SP}. The following relations are established between the sets A, B, C: And ⊃ B ⊃ C. So, any software product is a software tool, but not the other way around. Accordingly, any software tool is a program, but the reverse statement is not valid. The B\C difference is a subset of software tools which are not software products; the A\B difference is a subset of programs that are not software tools.

To get a software tool from a written program, it is necessary to insert this program into the computer memory, compile, debug and compile program documentation for it. This requires some labor costs, generally exceeding the labor costs for initial writing the program text. In order to obtain a software product from the software tool, it is necessary to more fully anticipate possible application and requirements of potential users, as well as to ensure that the previously stated requirements for SP are fulfilled. Labor costs, which three times exceed labor costs to create ST, may be needed for all that.

2. Software production specificity

Software production has features that should be taken into account at all stages of the software tools' lifecycle as well as in quality management. The software production has a high scientific capacity and intellectual content, it is created on the basis of intensive use of scientific knowledge and promotes the dissemination and use of knowledge by creating banks of this knowledge, information and expert systems, etc. Software tools development almost always requires high mental stress, deep and accurate knowledge. In ST development, besides programming specialists, highly

qualified specialists in a wide variety of subject areas (chemistry, physics, control systems, technological processes, and the like) have to be involved The narrow specialization and qualification of these specialists makes them unique. The complications in the control of progress and quality of development emerge. These difficulties are often subjective. High scientific capacity causes the need for increased costs for research and development work in the process of creating programs. This feature complicates the use of engineering methods in the design and quality management of ST

Software production is not expended and does not consume its resources when used. It is known that industrial production is divided into two main classes. The first are products expended when used (fuel, raw materials, substances, etc.); the second are products that consume their resources when used (machines, appliances, worktables, etc.).

The software production cannot be assigned to any of these classes of industrial production on these grounds. It is not expended when used and does not consume its resource. Moreover, with a well-established support service, ST is improved by detecting and correcting errors, as well as upgrading of methods, structure, and parameters. During the period of storage and usage the data carrier, on which the program is recorded, loses its features and may eventually become unusable. But by removing the copies from the ST in advance, the influence of this factor can be eliminated. The question of whether a copy of ST may be compared with the original and whether a user has the legal right to make copies of ST slightly impacts the nature and character of the features of the ST because it does not cause major complications for a user.

This feature significantly affects the methods of assessing the reliability of the ST and the possibility of extending the traditional interpretation of the reliability of technical means to software products. This is especially true of reliability indicators such as durability and maintainability. The nature of the main indicator of reliability, infallibility, is also changing.

2.1 Easy to manufacture

In most cases, manufacturing a software product on magnetic or paper media involves a relatively simple operation of removing copies from the product's sample-standart (original). However, no qualitative changes

occur. The identity of the copy with the original is easily controlled. It is somewhat more complicated to make copies of a software product in a permanent storage device. But this way of storing information is, firstly, not widespread, and secondly, it is also easily controlled and automated. This feature significantly affects the organization of quality control of software production. The main difficulty of this control lies not in the process of manufacturing the product, but the processes of development and testing of the prototype. The high quality of the prototype with strict adherence to the technology of rewriting guarantees the quality of the copies made from it – new copies of software products. It should be noted that the ease of manufacturing SP complicates the control over their distribution.

2.2 Easy to make changes

Upgrading a software product requires knowledge of the structure of the product being changed, a thorough analysis of the impact of the changes. But the process itself is simple. All you need is a good editor program. This feature, when skillfully used, is a significant advantage of software. This advantage is vital in dynamic spheres of applications, such as in automated control systems, where a constant search for the most optimal control modes goes on, which leads to the need for continuous improvement of the software. But the same feature easily becomes a drawback if the flow of change becomes poorly managed and unbalanced.

The ease (sometimes it seems to exist) of modernization generates a large number of relevant proposals, wishes, and sometimes insufficiently justified orders.

Attempts to implement all these changes are often unbalanced (uncoordinated) with real needs and opportunities. Under these conditions, the text of the program and the interrelations between its elements are confused; the program loses its consistency and accessibility for review; the difficulties in maintaining program documentation emerge. After all, it can accelerate the degradation of the ST to complete ineligibility. The ST upgrade process should be subject to careful control and planning.

2.3 The abstract materiality of the software

By its formal content, any ST is an information object. But the information contained in the software is very specific. In general,

information reflects the object of knowledge. The information contained in the command (operator) part of the program itself is obtained on the basis of the study of a certain object of knowledge (such as knowledge of a controlled object or process) and contains an order for the sequence of transformation (processing) of data that reflects the state of the cognition, to the required result. Information (data) is the object of processing in human-machine systems. The application code is not subject to processing. It itself contains data processing rules. This is the fundamental difference between the program as an information object and information (data) in general.

As part of the computing system, the ST manages the data conversion process. Naturally, the question arises whether ST can be considered a material object. The materiality of the ST, its components, the internal mathematical support of computers or implemented in long-term storage devices, is not questioned as they are susceptible to organoleptic perception. The materiality of the ST implemented on magnetic data carriers is questionable, since these ST carriers are not susceptible to organoleptic perception. Physical embodiment, the materiality of the software in this case are somewhat abstract. *The absence of concepts of tolerances and landings.* Each element of the program has its size in bytes. With hardware, program elements are easily moved within memory, which greatly facilitates the build process of the programs. Therefore, in programming there are practically no restrictions on the maximum tolerances for the required dimensions when designing program elements.

3. Software production life cycle

3.1 The stage of research

Software production is science-intensive, so its life cycle begins with the stage of research, which is carried out in the framework of research work on this issue. The main result of the research is the draft Terms of Reference (TOR) for development. TOR is the document that should be guided by the team of developers when creating the software. It is developed by the customer organization and agreed with the developer organization. In some cases, on behalf of the customer, the project of TOR is developed by the developer organization.

In the development of complicated software complexes on separate components of the complex one creates private TOR[2].

*The TOR should include the following sections:*name and scope; basis for development; aim and purpose of development, scientific and technical requirements; economic indicators; phases and stages of development; the procedure of control and acceptance. It follows that the main attention in the study should be given to the conditions of use and purpose of SP, the justification of scientific and technical requirements and economic (socio-economic) efficiency, which determine the level of quality and performance characteristics of the product. In TOR for multifunctional ST, besides the general requirements for ST in general, requirements for the quality of implementation of each function, as well as the priority (significance) of functions and information interaction between them, should be defined. Such differentiation will allow more purposefully influence the quality of the software tool being developed. A software quality assurance plan should be attached to the TOR, which defines the measures to ensure the required quality of the ST being developed, the sequence of their implementation, the responsibility for carrying it out, the objects and methods of control, the forms of recording data on quality and reporting. The value of a thoroughly grounded and compiled TOR cannot be underestimated. Such underestimation, especially on the part of the customer and the contractor, leads to a delay in the development and release of defective products. When implementing complex software complexes, up to 70% of all emerging problems are directly related to the imperfections of the requirements in the TOR and only 30% are the result of errors in the development process.

The imperfection of the TOR for the development of SP is caused not only by the misunderstanding of the TOR value, but also by such objective difficulties as novelty of problems, lack of relevant experience, lack or insufficient reliability of the initial data for the design, etc. In these cases, the customer and the developer want to have good TOR, but cannot develop it. In such a situation, preliminary (at the stage of research) prototype of automated systems, data processing systems and processes give good results. The essence of the prototype is as follows. The developer, having received from the customer the most general information

[2] Antipensky V.E., Bilousko V.S., Chujdan T.I.. Computing Machines and Programming: Workshop. Kyiv: Higher School. Main issue, 1987. 245 p.

about the purpose of SP, creates a prototype (simplified preliminary sample) of SP on the basis of those computing and software tools that he possesses. In doing so, he makes extensive use of unified software modules-components and modules of software products-analogs.

At the same time, in the requirements implementation restrictions when using the hardware interface and the equipment itself should be specified. When conducting research it is necessary to predict the life cycle of SP, trying to properly account possible changes in the conditions of use, tasks performed, the direction of upgrades. Particular attention should be paid to substantiating the requirements of SP resistance to various distortions (failure of information sensors, operators' errors, errors in communication channels and computing devices).

Many guidelines for the development of Software Requirements Specifications (SRS) include the IEEE Guide to Software Requirements Specifications standard.

The *first section* of the standard provides information about the SRS environment, the characteristics of the «correct» SRS, and aspects regarding the evolution of the SRS. The characteristics of the «correct» SRS are of particular interest.

Such characteristics are: uniqueness of interpretations, completeness, verification, possibility of citation, consistency, modification, clarity (possibility of tracing), usefulness at the stages of operation and maintenance.

It is considered that the SRS has the property of uniqueness of interpretation only when each requirement contained therein permits a single interpretation. SRS is complete if it has the following properties: includes all the essential requirements related to the operation, method of display, restrictions, equipment, attributes and external interfaces; determines ST responses to various (correct and incorrect) types of input information in different situations; meets some standard (individual discrepancies must be specified); all pictures, charts and tables in it are accompanied by detailed signatures and definitions of all terms and scales of measurement. The SRS is true if its every claim is true. The SRS is considered consistent if none of the requirements contained therein are in conflict with each other. The SRS is modified if it is easy to make any necessary changes in it without making any contradictions. The SRS has a track record if the reasons for any requirements are obvious and if it

facilitates the process of justifying the requirements arising from the development or improvement of the documentation. The SRS will be useful at the operational and maintenance stage if it facilitates the use and upgrade of the ST at this stage.

The second section of the standard addresses the basic ways of expressing requirements (using I/O specifications, multiple examples, and model specification); annotation (explanation and ranking) of requirements, as well as the most common mistakes in the description of requirements. It should be noted that none of the methods considered in the standard is universal. The method of expressing requirements using I/O specifications is only suitable if possible inputs and expected results are available for review; the way of expressing using examples – if possible system situations are available for review; the method of expressing using model specification imposes restrictions on the construction of software that contradicts the purpose of the SRS.

The third section of the standard sets out the overall structure of the SRS. In particular, in the «General Questions» section, it is recommended to display the product purpose, functions, user characteristic, general constraints, assumptions, and dependencies (factors that influenced the choice of the SRS requirements). Obviously, these issues should be reflected in one way or another in the TOR for the development of the SP.

3.2 Development stage

This stage begins with the development (consistency and approval) of the TOR and ends with the test of the prototype SP. In the general case, the stage of development of SP consists of the following stages: the development of SP, the development of technical proposals, sketchy design, technical engineering, functional engineering, testing. The results of the works for each stage respectively are: TOR, technical suggestions, sketch project, technical project (algorithms for solving problems), functional project (text of the program), test prototype. Let us consider the content of the works at these stages.

3.3 Technical Suggestions and Developments

SP should include a justification for the feasibility of the proposed variant of the structure selected on the basis of the analysis of the TOR and the various options for possible solutions.

The technical proposals list all the fundamental issues to be addressed in the engineering process with a preliminary assessment of their feasibility. For example, technical proposals for the development of ACS software should reflect the following issues: the degree of system automation; the composition of the general algorithm; previous structure and scope of the algorithm; determination of the structure and scheme of information of flows between computers, information sources and managed objects; preliminary estimation of the temporal diagram of data exchange between the computer and the objects of the system being automated; development of quality assessment criteria and methodological bases for their verification (control) during development; identification of problems that require preliminary modeling; determining the scope, methods and tools of modeling; preliminary selection and evaluation of methods for solving major problems; preliminary elaboration of principles and methods of ensuring the stability and reliability of management; working out the issues of development organization, as well as providing developers with general-purpose hardware and software.

The technical suggestions are the starting point for the development of the sketch project. In justified cases, both of these stages can be combined.

The sketchy design should include fundamental solutions that give a general idea of the structure of the SP, the designation of its components, the organization of relationships between these parts, data exchange and dynamic distribution of computer resources, as well as programming technologies. As a result of sketchy design, a preliminary estimate of the computer system (CS) resources required for the development and operation of SP is given. While developing of sketchy design, the problem of choosing the optimal structure to be released, the manufacturability of design, debugging and testing programs and the construction of a common algorithm for solving the problem are solved.

If necessary, one develops structural diagrams of the general algorithm at the level of its components, including databases; pre-connect the components of SP on the time of execution, use of external computers and information; sets the acceptable range of characteristics of the input and output values for each component; make private TOR for the development of the main components; simulates the operation of kernel components in order to test the basic principles of data processing and control; establishes basic principles of quality management of SP; identify

the critical ways and paths of the SP calculation; solves organizational issues of work at the stage of technical design. At the stage of the sketch project all the fundamental issues of technology creation of software complex should be developed. The sketchy design is approved by the customer organization. It serves as a guidance document for the development of a technical project. In justified cases, it is decided to carry out the task of sketch design in the framework of a technical project. The reasons for this may be the experience of developing similar products or their simplicity.

The SP technical project is a set of design documents that give a complete picture of the algorithmic and information structure of the product under development and contain all the source data for programming. The language used to describe the algorithms for solving problems in a technical project depends on the set programming technology. In the traditional approach, the so-called linguistic-formula descriptions and graphical schemes are used.

When describing algorithms, the developer uses any terms, concepts, and designations that are understood by him (but not necessarily understood by other developers). Mutual understanding between developers of complex software systems is difficult, which leads to unproductive spending of time and other resources. Therefore, in modern programming technologies, much attention is paid to the strict regulation of both the linguistic means of description and the design procedures themselves. After the design process is completed, the functional design stage begins. Functional design (FD) consists of three main stages: the development of the program, the development of program documentation and the testing of the SP prototype. The main content of the works on the stage of FD of program complexes is programming and debugging of components of the program complex, autonomous component testing, assembling of the program complex, development of program documents, development (alignment and approval) of the program and testing methods, conducting of all types of tests, adjustment of programs and program documents according to the results.

Production of SP. Production is a set of works to ensure the production of the required amount of SP in a set period; it includes the following types of work: studying the demand for this type of SP; production planning and production management; organization of

technological preparation and maintenance of production, logistics; storage and delivery. The stage of production of products for one-time orders has a hidden (implicit) character. It lies in making the required number of copies of the SP, including the program documentation.

Thus, in a single production, the SP developer combines the functions of the SP manufacturer and supplier.

3.4 Maintenance of SP

This stage consists of collecting information about the quality of SP during operation, modifying the product and notifying users of changes made. Maintenance functions are usually performed by the SP vendor. Practice shows that the initial stage of SP operation the developer's involvement in SP maintenance is very useful, and sometimes necessary. As the user and the supplier master the SP, this need is gradually eliminated. The maintenance stage is conditioned by the need to perform such tasks as the inclusion of new features in the SP, change of functions, modification and replacement of equipment in data processing systems, error detection and correction. The stages of operation and maintenance proceed in time paralleled. At the same time, the production of new SP can be carried out.

The reasons for the end of the SP life cycle may be different: no need for further use; replacement by new, more advanced SP; incompatibility with new equipment; dissatisfaction with the results of usage, etc. Due to the mentioned specificity of software production, the problems of evaluating its non-compliance with its purpose and utilization differ significantly from the corresponding problems of evaluating technical production. These differences are basically the following:

1. The difficulty of identifying non-compliant products as a result of uncontrolled upgrading of product units by users. In these circumstances, individual non-compliant units of products can be brought to compliance by users and, vice versa, compliant units to non-compliant ones. In general, regarding the appearance of software production, the assessment of its conformity becomes ambiguous;
2. The complexity of identifying inappropriate types of products causes the complexity of their isolation, i.e. separation from products that meet the requirements;

3. Non-compliant software products are generally not suitable for any use without further refinement and processing, so they should not be disposed (used for any other purpose). Their use should be completely excluded.

Separation and clear delineation of the phases and stages of the SP lifecycle, defining the necessary relations between the stages contribute to a clearer organization of certain types of work – ways to create appropriate technologies and technological tools, including methods and means of quality control both within and after the stage. When forming stages, it is very important to define clear links between them, to identify control points and decision making, which should facilitate a more accurate transition of information from one stage to another and ultimately reduce the development timeframe and improve the quality of the developed SP. The SP life cycle is not strictly consistent. It is iterative. The terms of reference, the sketch and the technical projects of the SP after their approval shall not remain unchanged. In the development of sufficiently complex SP it is impossible to achieve the invariability of the life cycle in practice. Sometimes, some design decisions made at previous stages of the life cycle have to be modified or refined and revisited. The reasons for this are different. Basically, they exist because the customer at an early stage of development does not quite clearly imagine and formulate system requirements, and the developer does not always immediately find the best solutions.

4. Software Tools Classification

The penetration of computer technology in all spheres of human activity, the desire to solve with the help of this technique a set of completely different problems extremely diversify software products by purpose, application, nature of production and maintenance, level of complexity, etc. Each type of ST has its characteristics that can significantly affect the methods of their development and quality management. Ignoring these features leads to problems of interaction, different kinds of misunderstandings and contradictions. Requirements for quality indicators depend on the type of ST: high requirements for one or another indicator for one ST may not be necessary for another[3].

[3] Ivashchenko N. N. Automatic regulation. Theory and elements of systems. Textbook for universities. Ed. 4th, rework. and ext. Moscow: Mechanical Engineering, 1978. 236 p.

Depending on the purpose, five subclasses of software are identified: *system ST*, applications *for scientific research*, applications for *designing*, applications for *control of technical devices and technological processes*, applications for the solution of *economic tasks*.

The following breakdown into types of software can be considered universally accepted: application, system, and tool software. We give the following informal definition of these types. For a computer to do your job, you need to create application software. For the computer to cope effectively with many applications and to be well-adapted to the environment, you need to create system software. To make it easier to develop software, you need to create and use tool software. Application ST are developed by experts who are well versed in the processes they automate. System ST are usually more complex than application and tool ones. They are developed by experts who know all the intricacies of programming and operation of computer systems.

Examples of system software are operating systems, database management systems, and the like. Instrumental ST are used at the stages of program development and maintenance, including debugging and testing. Typical examples of software tools are compilers, text editors, data archivers, change analyzers, and the like. Instrumental ST are developed by experts who are knowledgeable in programming technology as a whole or in specific aspects (transmission, editing, debugging, testing, etc.).

All considered types of ST are classified by one attribute – purpose. But when planning development, development management, quality management of software products, it is necessary to consider not only the purpose of the software, but also their other characteristic features. Such features include, for example, the number of users. Of course, the ST that a specialist has developed for himself and which he will use for himself, do not have the requirements that apply to ST developed for thousands of users.

In the first case, the requirements for the ST are determined by the developer at his discretion, and in the second they must be determined by the customer, taking into account the possibility of using the ST in a wide range of conditions. Even when developing software for your own use, the frequency of use (one-time, daily, weekly, annually, etc.) is essential. Problems that are of great importance for specific (consisting of components) ST may be irrelevant to unspecified ST (that is, the components themselves). The requirements for real-time ST differ

significantly from the requirements for ST that implement, for example, basic computational tasks. The problem of comprehensive detailed classification of software is extremely complex and has not yet been resolved. Let us consider some classification methods that are important for understanding software quality management issues. By the nature of the manufacture one should distinguish between single and mass production. The developer himself performs the functions of the manufacturer and the supplier. He provides training for the customer support staff, assists the user with the commissioning of the ST into industrial operation, accompanies the ST.

The single nature of the manufacturing does not preclude the re-production of ST for implementation at another enterprise (re-introduction). In this case, the developer usually has to refine the software, taking into account the specifics of use in the new conditions. Depending on the scope of the revision, it may happen that the revised ST should be considered as new ST. The batch type is characterized by periodic production of batches of homogeneous software that is in high demand.

By nature of supply and use, software may be characterized by the autonomy of supply and use, or the supply and use of PCS or an automated technological complex (ATC). A distinctive feature of autonomous ST is that it can be developed, manufactured, tested and delivered (sold) autonomously An example of autonomous ST can be almost any ST related to system ST. Distinctive features of the ST supplied as part of PCS or ATC are the joint development, manufacture and testing of ST and system being automated in which it is delivered to the customer (user). Examples of such systems are the PCS. In terms of the number of functions performed, all specified SP except the software modules are multifunctional. For example, in the Logistics Supply Subsystem (LSS) of ACS considered as SP, the following functions (tasks) can be implemented: determining the need for materials, determining the need for equipment, determining the need for spare parts, developing a schedule for the supply of units, etc. Each of these features, with the exception of the latter, has a specific purpose and therefore has autonomy of use, i.e can be used independently. The LSS itself can be considered as a multi-purpose system, although it has a general (global) purpose. But multifunctional SP can be one and the same. These SP include, for example, those SP that bring two managed objects,

moving in space, together. The goal here is one (approximation for a given distance), but for approximation it is necessary to solve the following problems: determine the position of each controlled object in space at time (t+1); by extrapolation calculate the position of objects at time (t+1); calculate the optimal approximation trajectories for each object at time (t+1); to calculate the control actions corresponding to these trajectories. It is clear that the successful solution of the general convergence problem is possible with the correct solution of all partial problems. Depending on the nature of the implementation process, the following types of ST are distinguished: implemented as part of the developed ACS; implemented in existing PSC (ATK); autonomous implemented and self-relevant ones; software components that are built into the software system. To determine the level of unification, ST and their components belong to one of the following types: *standard; unified; original.*

To *standard* one includes components of a specific ST that meet the requirements of state, industry or national standards, which are referenced in the design specifications. *Unified ST* include components of a specific ST that can be used to solve the same problems in several software tools. Unified ST are purchased, borrowed ST, as well as developed according to the standards of given enterprise and used in various ST. Components of a specific ST that are not made in the organization but purchased are called acquired. Borrowed items include components of a specific ST that were previously designed as original to other ST, used in given ST, and which have developed program documentation.

Original ST include components of a specific ST developed for the first time for this ST and used only in this particular ST. In practice, original ST are often refined (unified) and thus transformed into unified ST. ST is a complex product. The average software package contains 40–50 thousand source text operators.

Taking into account the existing semantic relationship of operators, which needs to be known and recorded, the average ST is compared to the locomotive, which has about 25 thousand details. Many materials basing on ST quality and programming technology are difficult to use because it is unclear to which class the ST they belong, and unreasonable generalizations often make it difficult to create effective methods and tools for evaluating ST quality.

5. Quality features of the ST

While identifying the terms «Software tool» and «program», sometimes the measure of its relevance to the original algorithm is understood under the quality of ST. However, the following two mistakes are made: the concepts of «program» and «software tool» are not identical; errors in the original algorithm are no less likely than programming (encoding) errors. Errors may also be contained in the program documentation. Therefore, the full compliance of the program text with the algorithm text cannot guarantee the suitability of the ST for its intended purpose. The compliance of the ST algorithm is characterized not by the quality of the ST as a product, but the quality of one of the main technological processes in the creation of the ST- the quality of programming. It is more common to define the quality of the ST as a measure of compliance of the real characteristics of the ST with the characteristics given in the TOR. Such interpretation of the ST quality is acceptable only if the TOR has fully and uniquely defined all the consumer properties that the developed ST must have. But this condition is not yet feasible due to the lack of a common nomenclature of quality indicators, methods of setting scientific and technical requirements for ST and lack of experience in solving these problems.

Software quality one should understand as a set of ST features that determine their suitability to meet specific needs according to their intended purpose. It is based on three key concepts: ST feature, need, ability to meet needs. Let us consider these concepts[4].

The existence of ST as a product of labor is an objective reality. As a product, ST has many attributes – objective features that determine its difference or similarity with other objects and are manifested in its creation and operation. These attributes may be common to a given product class, and specific to a particular type of software or specific ST. According to the impact on the ST quality one should distinguish between essential attributes and insignificant. Only the essential attributes are of interest. But the measure of the impact of these attributes on the quality of specific ST is also different. These differences need to be taken into account. ST attributes are quantitatively and qualitatively characterized by quality indicators.

[4] Miroshnik I.V. Automatic control theory. Linear systems. St. Petersburg: Peter, 2005. 336 p.: pic. (Training Series).

The following main aspects of ST needs can be distinguished: scientific, technical, economic social. The need for ST is established at the beginning of its life cycle, usually at the stage of research (marketing). In doing so, the following basic questions should be answered: field of usage; solved tasks; expected effect from use (scientific, economic, social). In the narrower sense, the need for accuracy of data conversion results, trouble-free operation, ease of maintenance, ease of exploration, and the like are established. In order to meet the needs, ST as a product of labor must possess certain useful (consumer) attributes that collectively determine the public usefulness of ST. Usually, each product has several useful features, each of them satisfies one or more needs. ST quality is both a technical and a socio-economic category. On the one hand, it is closely linked to features that satisfy certain needs. But the attributes of the subject are not economic categories, so a technical approach is promising from these positions on the ST quality.

On the other hand, the ST quality is a concrete expression of the public consumption value, so an economic approach should be applied to it.

Two characteristic features should be noted in the formation of ST quality.

1. The quality of any industrial product on batch or mass production depends largely on the manufacturing process. It cannot exceed the technical level achieved in design. But it may be well below this level due to non-compliance with manufacturing technology requirements. Production of SP is often a simple technological operation of making a copy from the sample-standard by rewriting from one data carrier to another. However, no qualitative changes occur. The identity of the records is easily controlled. Therefore, it is necessary to control and manage the ST quality, not mainly while its production, but in the development, that is, when the ST quality is formed.

2. SP during operation and maintenance, as a rule, are constantly changing, modernized. Therefore, the maintenance process itself can be called, for example, an extended development process. But any change in the structure of the software production leads to a certain qualitative change, so the quality control and management of the ST must occur not only during their development, but also at the stage of operation (maintenance).

Factors affecting the quality of the ST can be constant and variable, direct, indirect and inverse. Quality factors should not be confused with quality indicators. Quality factors characterize the conditions and elements that influence the formation of quality.

Quality indicators, more precisely, denotation of quality indicators, directly characterize the quality itself. Any management is a purposeful action on a managed object to achieve specific, predetermined results. Quality, as a set of consumer attributes of products and services, is a specific management object and has significant features. The product quality management itself differs from the management of the product creation process in that not the organization of production, but the regulation of the properties of the products produced is the object of management here. These attributes are formed at the stages of the life cycle under the influence of various conditions and factors. Quality management (QM) is the process of acting on those conditions, factors and socio-economic relationships that influence the formation and change of consumer attributes of products.

To accomplish this process, a system of governance is created – a set of interacting bodies, tools and methods of management. Software quality management has organizational, methodological and socio-economic aspects. The organizational and methodological side of SQM is expressed in the development and application of advanced programming technologies, consolidation of scientific and technical achievements in the relevant standards and methodological documents, equipping the developers with advanced technology, etc.

The socio-economic side of SQM is expressed in the creation of such a system of socio-economic relations between all participants in the development, which will ensure the creation and production of PP of the required and guaranteed quality. This system shall cover: a) the relationship of the administration of the developer organization with the customer organizations; b) the relationship of the administration of the organization with the staff of its units; c) the relationship between the development units; d) the relationship of the managers of all units with its individual executors.

At the same time, personal interest and responsibility of each manager and contractor for the quality of the software under development should be ensured at all levels. Users are most interested in the quality of the

software, but they are little interested in the ways in which the developer achieves a certain level of ST quality. It is only important that this level meets the actual needs. Actions aimed at assuring the user (customer) with the confidence in the proper level of purchased products constitute the External Quality Assurance. Such actions include, in particular, marketing, drafting and mutual harmonization of specifications of requirements, testing and maintenance of software. Elements of internal and external quality assurance can generally overlap. Having a specification of requirements contributes, for example, to confidence in achieving the required level of quality both by the developer and the customer.

For a better understanding of the nature of software quality management in the process of its development, let us consider the conceptual model of management.

6. Conceptual model of software quality management

In general, *management* is an integral part of the functioning of systems of organization of various nature: biological, technical, socio-economic. In each of them there are objects that subordinate to others, and therefore, and control them, forcing them to move in a certain direction, perform the specified actions, organize their activities as a whole[5].

Management of the project (object-system), its components and processes, with the purpose of increasing the efficiency of the systems functioning occurs at the stage of system design, creation, formation, development, formation, functioning of the system. The effectiveness of the management is determined by the adequacy of the control actions to the object of the management.With regard to computer ST, the quality management scheme is as follows. The object of management (action) is the quality of the object of labor. The subject of labor, depending on the stage of development, respectively, is the TOR, technical design, functional design (program text), ST prototype. ST quality is mainly formed at these stages, so quality management should start from the very beginning of the software development process and be continuously implemented throughout the process. In general, management actions can affect not only object labor directly, but also labor and technological processes, if they do not contribute to the management goal, as well as factors affecting ST quality.

[5] Popovich M.G., Kovalchuk M.G. Automatic control theory: a textbook. 2nd edition, revised. and suppl. Kyiv: Libid, 2007. 656 p.

Programming labor tools include compilers, downloaders, program builders, documenters, automated debuggers, test data generators, automated programmer jobs, including computers, etc. Technological processes consist of certain technological operations for ST creation, performed by means of labor under the control of programmers (operators) or directly by programmers. In any case, the role of the person in the technological process, its impact on the quality of the product is crucial. The purpose of management is to provide the necessary level of ST quality, which guarantees the expected socio-economic effect of the use of this ST for its intended purpose. The main role in software quality management is performed by software development, operation and maintenance managers (depending on the life cycle stage), direct developers or ST maintenance specialists, together with management tools.

The category of specialists involved in software quality management will be referred to as *developers*. Developers have an effect on the state of the ST either directly or through appropriate technological means (TM) and technological processes (TP). These actions can be both positive (coinciding with the purpose of management) and negative (not predictable perturbations). Negative actions result in program errors. Sensors of information about the status of the managed process and the quality of the software depending on the stage of the software life cycle are either the developers themselves (at the design and debugging stage), or experts of the quality control groups (at the design stage), or testers (at the test stages), or users (at stage of operation). Naturally, certain categories (concepts) should be used to describe quality. First of all, it is necessary to determine the consumer properties of the ST being developed, that is, those attributes that the software must possess to be able to be used effectively for its intended purpose. Each attribute or group of attributes is quantitatively characterized by quality indicators. In order to manage quality, it is necessary to know the acceptable rates of the quality indicators as well as the criteria for quality assessment at each stage of the ST life cycle. This information should be contained in the terms of reference and specifications for specific ST or groups of homogeneous ST.

The actual rates of the Quality Score can only be set when the ST development is completed. In the process of development, you can only make predictions about the quality of the ST, controlling the presence or absence of certain features in the project documentation and programs

when debugging. Monitoring the current state of the quality of software being developed usually relies on special quality quality management groups (QMG) that are independent from the developers. In order for the QMG to perform its functions successfully, it must have a clear understanding of the quality of the controlled ST analyzed at different stages of the life cycle, of the methods for determining quality indicators and quality criteria.

The conclusion about the current state of quality of the controlled software is made by the ST QMG on the basis of examination of the project documentation (if the program has not yet been written or has not acquired the performance capability), or by analyzing the correctness of the initial data (results) by comparing the actual data with the expected ones (in the working program). In the first case, it should have a methodology for conducting the examination, and in the second case there should be clear signs of identification of correct (incorrect) results. All ST quality information should be submitted to developers or maintenance professionals who, basing on the analysis of the information, make decisions about how to influence the management object. Primary information about the quality of software in the development stages is often symptomatic. Only external signs of design errors, deviations of the data processing process (ST operation) from normal mode, or lack of the required attribute in the ST are recorded. In order to make a decision on the impact on a management object in order to improve its quality, it is necessary to establish the reason for deviation from the required quality level and the way to eliminate this cause. When designing an impact, developers should consider the requirements and capabilities of the programming technology used, the requirements for ST from customers (users), the structure of the ST, the available resources, as well as the relationship between the signs of errors, their causes and ways to eliminate them.

The essence of managers' influence is to change the structure of the program and program documentation (error correction, introduction of new functions and procedures, improvement of methods of solving problems, etc.) in the direction of its optimization according to the criteria specified in the TOR. If necessary, labor tools and technological processes change. Thus, ST quality management tasks are a variety of optimization tasks and have the following components: defining the goal of quality management

(QM); knowledge of ST quality assessment criteria; knowledge of the current position in relation to the purpose; knowledge of the microstructure of ST and factors that affect the quality of ST; knowledge of limiting conditions in terms of execution and resources; determining the best ways to reach your goal. Under the products quality management system (QMS) one understands a set of organizational, scientific, technical and economic interrelated measures to establish, provide and maintain the required level of ST quality in its development, production and operation. The ST QM system is multilevel.

Previously, a quality management scheme for a single ST was considered. But organizations that specialize in the development of complex software solutions can simultaneously develop or prepare for the development several ST. Taken together, these ST constitute *software* produced by this organization. The governing bodies in this scheme form the administrative and technological units of the organization. Direct management objects are not software, but teams, software developers, *technological lines* (TL) *development* and *technological processes* (TP). The ultimate goal of management is the required quality level of software. Information on the state of the software development process goes to the Software Quality Control Service (SQCS) along with the QCG of the ST being developed. To make decision about management actions it is necessary to have an annual and perspective plans of software development (thematic plan of research and development works); a list of requirements for the quality of software by potential users; data on the current state of quality of the developed software; data on available labor, material and time resources; a list of organizational and economic mechanisms for regulating the activities (OEMRA) of developers, including the rationing of labor and resource costs, the promotion of high productivity and quality of software; methods and means of technological preparation of development (TPD), including formation of technological lines and technological processes; data on the availability and characteristics of technological programming modules, etc.

Having this data at its disposal, the governing body influences the quality of the software created in the organization by beforehand and purposeful technological preparation of the development of specific ST, setting and correction of the ST QM goals for separate periods of time depending basing the state of the developed ST, regulating the team of

developers, stimulating the creation of high quality software. It is important to emphasize the special role of the latter factor, since an individual in the system of SQM certainly plays a decisive role.

The quality of software is formed at all stages of its creation, therefore, operational quality control is necessary for the operational impact on quality. During the operation of the SQM system, there is a need to collect, store and process large amounts of information. Naturally, the SQM system should be as automated as possible. Like any automated control system, it consists of the following elements: organizational, methodological, technical, software and information providing. The model under consideration contains the basic elements of the SQM system at the level of the developer organization and the relationship between them. On its basis, by further detailing, it is possible to determine the composition of the necessary regulatory, methodological, information and software tools for supporting the SQM systems, as well as the tasks and overall structure of the quality system of the developer organization.

Three quality objectives, that the organization faces, have been identified. These tasks can be interpreted as follows: the organization must achieve and, in the case of support, maintain the quality of the software at a level that ensures continuous satisfaction of the user set or offered requirements; the organization must assure its management that the required quality is achieved and maintained at the required level; the organization should provide the user with the assurance that the required quality of the delivered software is achieved or will be achieved. If necessary, the user may require appropriate evidence to be provided. Solving these tasks requires the introduction and definition of key terms and definitions. Quality policy is the main directions, goals and objectives of a quality organization, formally formulated by its senior management. Overall quality management is an aspect of the overall management function that defines and implements quality policy. General management includes quality planning, resource allocation, evaluation and other systematic quality actions.

A quality system is a set of organizational structure, responsibilities, procedures, processes and resources that ensure the overall quality management. As a condition of the contract, the customer may require clear evidence of the use of certain elements of the system. Methods and activities of an operational nature are used to meet the quality

requirements. In order to avoid confusion, it is advisable to add specific references to narrower, specific concepts, such as «quality management in the design process». Quality assurance is a set of planned and systematic activities needed to create confidence that a product meets certain quality requirements. The quality of software is formed at all stages and stages of its life cycle. Therefore, the quality system functions simultaneously with all other activities affecting quality. The quality loop of software has some differences from the quality loop of other industrial products. These differences are mainly due to the decisive role of the software prototype in shaping the quality of the development stages rather than the production stage, as is the case in industry.

7. Factors affecting the quality of software

The software quality depends on many factors. Let us consider the main of them.The responsibility of the management in quality assurance is determined by the presence in the organization of the quality system of the following elements: documented policy in the field of quality, goals and obligations; responsibility and interaction of the staff which affects quality; means of inspection and specially trained personnel; representative of management bearing personal responsibility for meeting product quality requirements; periodic analysis of the effectiveness of the quality system which runs in the organization, the quality of regulatory documents of the software being developed, in the part of optimality and completeness of the claims set in it.

Preparation of Terms of Reference (TOR) for ST development and defining the main list of requirements therein is the first stage of ST design. TOR must be composed both on software, supplies (software complexes), which are standalone objects, and on program components. When developing complex ST that have no analogues at the time of design, direct assembly of the TOR is usually preceded by research work, the purpose of which is to determine the purpose of the ST, areas and features of its application, as well as to analyze the requirements of potential users.

Efficiency of programming technologies. Technological preparation of software development. The process of creating a PP is costly and time consuming. Programming technology, management of the software creation process should provide the maximum beneficial effect at

certain costs. Naturally, such an effect can only be achieved by using the most advanced methods and tools to develop software. The technological preparation of software development should be complete and timely.

Regularity and effectiveness of quality control of development. The process of creating software should be under constant and careful control. The technology for detecting and eliminating errors, as well as temporary material resources for the implementation of this technology, should be installed in advance. Practice shows that for the production of high quality products, it is necessary to plan up to 60% of labor costs in advance to ensure proper control, debugging and testing of programs, to establish control procedures in advance, to create software and technical means for debugging and testing. Regular use of inspection methods prevents up to 60% errors in advance.

Developer Qualification. The quality of the created ST is determined by the following properties of developers: the level of knowledge (knowledge of problems, programming languages and computers, engineering techniques, data processing principles), the availability of practical skills (experience in creating similar programs and software systems);) level of initiative (understanding of the tasks being solved and their relationships, efficiency of working time use, the desire to bring each task to a complete completion, maintaining working contacts with the co-workers); level of responsibility (focus on the work being performed, constant desire for self-improvement, healthy self-esteem).

Content and quality of software (instrumental) tools used in development. The development of sophisticated software is associated with the need to use various computer hardware and system software. These tools serve as a kind of technological equipment for programmers. Naturally, the quality of ST created depends on the reliability of this equipment and the stability of technological operations. Also timely and fully meeting the developers' need for these tools is important.

Stimulating the creation of high quality software. Despite significant achievements in the field of programming industrialization, the nature of programmers' work is also individual and largely dependent on the personal abilities of the performers. The performance and quality of programmers working under the same conditions can vary several times. Therefore, an effective system of stimulating the creation of high quality software must be introduced when creating programs, which

involves the remuneration programmers depending on the quantity and quality of results. Development managers should always remember that, with stimulation for quality work, developers will find effective ways to achieve a set goal. Conversely, with the absence of stimulation, many useful start-ups will be unfulfilled. This is one of the manifestations of the human factor.

Formation and adherence to uniform principles of software development. Based on the results of the study and analysis of the factors affecting the quality of the software, taking into account the specificity and experience of creating these products in each development organization, the basic principles of software development should be formulated: development management with the help of a project plan broken down into stages, quality control throughout the development period, from the early stages, ensuring strict control of compliance of the features of the original software product with the requirements set out in the specifications; use of advanced methods and programming tools; supporting a high sense of responsibility for the quality of the programs being developed in each project partner; use of the minimum number of highly qualified employees; continuous improvement of methods, means and software development organization. Another quality system is based on the following principle: all stages of development are clearly distinguish At each stage, the outputs and quantitative and qualitative criteria for their evaluation are determined. Quality processes and output are standardized according to quality. Outputs are monitored according to previously established criteria; special attention is paid to the organization and quality control of the work of autonomously working groups of programmers. Various methods of software quality checking are considered, which are considered not an optional occupation, but one of the most important elements of design[6].

Marketing. The quality of a particular ST depends on the effectiveness of the system of market research measures and the consumer features of that ST (marketing effectiveness) throughout its life cycle under different conditions of application. Marketing units should work closely with the software support units, as the support team usually receives information not only about ST errors found during their operation, but also suggestions on ways to improve the software.

[6] Tsypkin Ya.Z. Fundamentals of automatic systems. Main Editing Physical and Mathematical Literature Publishing "Science", Moscow, 1977, 56 p.

The clarity of the results of the quality control. For each software at the earliest stages of development quite simple and clear criteria (signs) of high quality and lack of design should be set. Information on the progress of software development and the results of its monitoring should be clear and publicly available. Software developers should always be prepared not only to guarantee high quality ST, but also to demonstrate it convincingly.

Existence of a comprehensive quality assurance plan for the software developed. The plan includes a set of measures to ensure and maintain the required level of quality of software, distributed by contractors, in time and by material resources. It is based on the specification of the requirements for the software, the knowledge of the quality factors, the specifics of the ST being developed, and the necessary resources for implementation. The plan is developed at the same time with the development of the TOR as an appendix to it. The listed quality factors (first order factors) are common to all types of software products and to all the attributes of these products. In addition to these factors, it is possible to distinguish into separate groups such factors (second-order factors), which most significantly influence the formation of a specific attribute or group of software attributes. The specific attribute of the ST in this case can be considered as a consequence of the actions of the selected factors.

8. Errors in software and ways to prevent them

Errors in the programs of automated process control systems lead to the violation of technological regimes and the production of defective products. Errors in automated organizational management systems lead to irrational use of material resources and labor costs. In some cases, bugs in the programs can have catastrophic consequences. In addition, bugs in the software, poor quality, or lack of quality assurance for individual software are reasons for poor implementation rates. With the implementation of software containing gross errors, in tens or even hundreds of enterprises, the negative effect will increase an appropriate number of times. This effect is exacerbated by the need to involve in the search and eliminate the mistakes of many of the most qualified professionals who are doing the job with the detriment of their kernel business. It is an admitted pattern that the earlier a project error is detected, the easier it is to correct it. The dependence of the relative cost of bug fixing on the time of its detection is shown in table 1.

Table 1

The dependence of the relative cost of correcting the error on the time of detection

Stage of the life cycle	The relative cost of bug fixing
Development of the TOR	0,1…0,2
Sketch design	0,3
Technical engineering	0,5
Programming	1,0
Combined testing	2,0
Preliminary tests	3,0
Experimental operation	4,0
Acceptance Tests	5,0
Operation	20…30

Therefore, ST bug prevention measures in the early stages of design should take a special place in software quality management systems in development organizations (enterprises). In order to develop effective measures to prevent software bugs, it is first of all necessary to establish their nature, causes and symptoms. To understand the nature of bugs, it is needed to consider the following characteristics: *nature of the external manifestation, physical essence*, *stages of introduction*, *nature of bugs*, their *types* and *classification*. Any program, after all, is a set of instructions, the execution of which provides the conversion of the varied initial data to the desired result. An error (a set of errors) in the program leads to an incorrect result. This is the essence of the external manifestation of bugs in the program. The physical essence of the software product is a record of the program on a data carrier. Therefore, the physical expression of the error is the incorrect entry of any element of the program (commands, macros, elementary construction, operator, data set, etc.). The error correction process in this case is a replacement the incorrect entry with the correct one. Thus, an error in the software product from the end-user perspective is the entry of a program element on a data carrier or in the software documentation, which results in the wrong result being sought. Note that this definition allows the correct result to be obtained in the presence of errors in the program. This is possible indeed in cases where program elements containing bugs are not used in specific implementation conditions. The elements of the program can be not only prescriptions for the order of conversion of the initial data into the desired result, but also records of quantities, descriptions of variables, etc.

Therefore, the definition indicates the use, not execution of the program element. Bugs in ST can be made at different stages and phases of their life cycle. Accordingly, there are *errors in the statement of the task*, in *designing*, in *programming* and *in recording on the data carrier*.

Errors in formulating a ST development task. The formulating a ST development task is formulated in the form of terms of reference and technical conditions. These documents define the consumer attributes of the ST, which must take into account all requirements of potential users. In turn, user requirements should be based on knowledge of the purpose and conditions of use of the ST. Thus, to understand the tasks of development means, first of all, to set the aim and purpose of development, conditions of application, expected ranges of input data and results. Misunderstanding of the problem being solved, inaccurate knowledge of the initial data, conditions of operation and expected results lead to errors in the formulation of the task, resource planning, which may eventually make all further work of the designers unnecessary. The requirements for the quality (specifications of quality) of the ST should be an integral part of the general technical requirements. Moreover, they must be comprehensive and well-grounded. Otherwise, the ST will be disabled. There are situations in which the TOR for development did not have the requirements for the stability of ST ACS in the presence of distortive effects. Such ST had to be radically modified immediately after experimental operation. If the ST has the ability to be modified, then the problem of improving the stability of the software will be solved. Otherwise, the design process must be started from the beginning.

Design errors. Design errors include: errors in the choice of methods for solving problems and parameters; inconsistency in the use of data in time (in real time systems); neglect of correlation between individual components, etc. All these errors can be qualified as the inadequacy of mathematical models to real processes occurring in the system, to researched processes. Design errors are sometimes referred to as algorithmic errors because they are formally contained in problem solving algorithms. All the errors that are not detected at the stage of algorithm development are subsequently transformed into programming errors.

Programming errors. Modern programming languages and translators contain some set of tools for debugging and checking programs. However, these tools are not enough to guarantee error-free programming. Therefore, programming (encoding) is also a source of ST errors.

Software errors include errors in the choice of numerical methods of implementation of algorithms for solving problems, schemes and calculations; interpretation of algorithmic constructions (semantic errors); coding (syntax errors); in conjunction of program modules and programs; in the implementation of logical conditions; in the data description; in the documentation.

Errors while recording on data carrier. Compiled program text must be recorded to a specific data carrier before entering the computer. This work is mostly done manually and can cause new errors. The percentages of these errors are small because they are easily controlled and eliminated. ST errors can also be introduced during operation and maintenance. Such errors are the result of unqualified correction of predicted errors, unqualified ST modification, negligent treatment of data carriers, etc.

The classification of errors considered is a priori. It is based on the types of ST creation and operation work (at the stages of the ST life cycle). This classification is useful for forecasting errors at different stages, assessing the quality of work of teams specializing in the performance of particular types of work, and making the necessary decisions. For example, data input/output errors are symptomatic because they have external characteristics (symptoms), which, however, do not allow to explicitly identify the causes of these errors. Computational errors usually directly indicate the true cause (error in sign, index, etc.), but have no characteristic features.

Table 2 shows the distribution of errors by ordering the signs of causes by frequency of occurrence. An attempt to establish the interdependence of causes and signs of manifestation of errors was made. The general pattern was not established, but it was possible to identify the signs of errors that are most common in these projects. These include: -bit grid overflow – 30.4%; incorrect management transfer – 16.4%; incompatibility of programs with databases – 14.5%; incompatibility of programs by the types of data being forwarded – 9%; failure to perform additional functions by the program – 4,9%, incompatibility of programs – 7%.

Table 2

Error distribution by frequency of occurrence

Type of an error	Error distribution, % from total quantity	Type of an error	Error distribution, % from total quantity
Calculations	7	Of interface	10
Logical	22	Database initialization	6
I/O	10	In the documentation	8
Data manipulation	15	Other	22

Collecting, processing error data, classifying errors, establishing their causes and probabilities make it possible to do purposeful work on error prevention and thus affect the quality of ST.

REFERENCES

1. Antipensky V.E., Bilousko V.S., Chujdan T.I. Computing Machines and Programming: Workshop. Kyiv: Higher School. Main issue, 1987. 245 p.
2. Ivashchenko N.N. Automatic regulation. Theory and elements of systems. Textbook for universities. Ed. 4th, rework. and ext. Moscow: Mechanical Engineering, 1978. 236 p.
3. Miroshnik I.V. Automatic control theory. Linear systems. St. Petersburg: Peter, 2005. 336 p.: pic. (Training Series).
4. Popovich M.G., Kovalchuk M.G. Automatic control theory: a textbook. 2nd edition, revised. and suppl. Kyiv: Libid, 2007. 656 p.
5. Tsypkin Ya.Z. Fundamentals of automatic systems. Main Editing Physical and Mathematical Literature Publishing "Science", Moscow, 1977, 56 p.

Information about the author:
Kyselov V. B.
Doctor of Technical Sciences, Professor,
Director of the Institute of Municipal Administration
and Urban Economics
of the V. I. Vernadsky Taurida National University