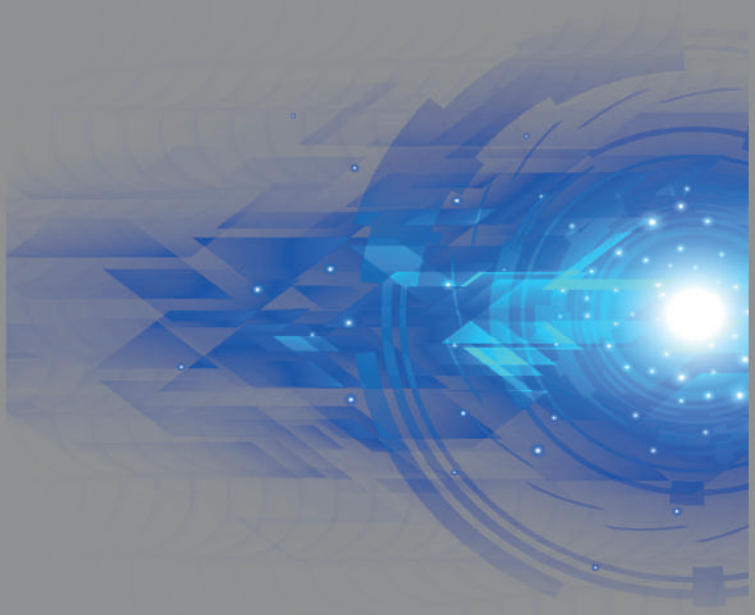
Rezanova V.G., Shcherban V.Yu., Demkivska T.I.

TECHNOLOGIES OF DEVELOPMENT SOFTWARE PRODUCTS



Tutorial for students of specialty 122 Computer Science

Ministry of education and science of Ukraine
Kyiv National University of technologies and design

**Rezanova V.G., Shcherban V.Yu.,
Demkivska T.I.**

TECHNOLOGIES OF DEVELOPMENT SOFTWARE PRODUCTS

Tutorial for students of specialty 122 Computer Science

Recommended by the Academic Council of the Kyiv National University of Technologies and Design (Protocol № 6 of Janyary, 26, 2022)

Kyiv-2022

УДК 004.42

Recommended by the Academic Council of the Kyiv National University of Technology and Design for students of computer science and related specialities (Protocol № 6 of January, 26, 2022)

Authors:

REZANOVA V.G. - Candidate of Technical Sciences, Associate Professor of the Department of Computer Science, Kyiv National University of Technologies and Design ;

SHCHERBAN V.Yu. – Laureate of the State Prize of Ukraine in the field of science and technology, professor, Doctor of Technical Sciences, Professor of the Department of Computer Science, Kyiv National University of Technologies and Design;

DEMKIVSKA T.I. - Candidate of Technical Sciences, Associate Professor of the Department of Computer Science, Kyiv National University of Technologies and Design

Reviewers:

OPANASENKO V, M. - Laureate of the State Prize of Ukraine in the field of science and technology, Doctor of Technical Sciences, Professor, Leading Researcher of the Institute of Cybernetics of the National Academy of Sciences of Ukraine;

KRASNITSKY S.M. - Doctor of Physical and Mathematical Sciences, Professor, Kyiv National University of Technology and Design

Rezanova V.G., Shcherban V.Yu., Demkivska T.I. Technologies of development software products. Tutorial for students of specialty 122 Computer science. – К.: Видавничий дім «Артек», 2022. – 234 с.

ISBN 978-617-8043-33-9

The tutorial outlines the approaches and methodology of designing complex objects and systems; principles of construction and operation of CAD; models of design objects; modern software development methodologies; software life cycle, methods, languages and standards of information support of software products at different stages of their life cycle; structural methodology of information systems development; methods of verification and testing of programs and systems; methods for assessing the quality of software products, software development technologies, CASE tools; features of modern large-scale information systems projects. The textbook is fully consistent with the program of the discipline " technologies of development software products " and in addition to theoretical information contains a guide to work in special software packages with examples of practical problems.

The tutorial is intended for for students of computer science and related specialities

ISBN 978-617-8043-33-9

УДК 004.42

CONTENT

INTRODUCTION

Humanity entered **the age of informatization,** as you can see in the following: information and information resources on the global market are becoming the most important high-tech product; firms, that develop automated information technology, stand at leading positions in the world economy, determine further development of competitive product development; it is impossible to create high technologies without informatization; information technologies (IT) open up new opportunities in increasing the efficiency of production process in the field of education and life, provide a management of group projects, Internet technologies, CALS-technologies, distance education etc.; informatization of society leads to internationalization of production.

An indicator of the scientific and technological power of the country is the foreign trade balance of professional knowledge, which is realized by the market of licenses of production processes, know-how and consultations on the application of science-intensive products. For example: USA handles about 80% of innovations to affiliated companies in other countries. As long as these companies master the proposed technology, the USA is preparing new ones, which means that the advanced technological cycle of a highly developed country is being implemented. Among the most important components of the power of information of the USA is the global leadership in the development, production and use of information technology.

This way the evolution of the world market benefits the country itself, which creates and transfers high-tech products to other countries. Other countries should include new technologies and modern professional knowledge. That is why in the information society information, knowledge, creativity become a strategic resource. Through distance learning,

computer games, computer video and other IT, computer technology has a tremendous impact on the formation of conditions and environments in which talent develops and thrives. It is assumed that the social impact of the information revolution will consist in the synthesis of Western and Oriental thought.

In connection with the above, there is a need for specialists who have the ability to apply the theoretical and practical foundations of methodology and modeling technology to study the characteristics and behavior of complex objects and systems; ability to apply methodologies, technologies and tools to manage the life cycle processes of information and software systems and information technology products in accordance with customer requirements.

PART I

I.1. BASIC CONCEPTS AND METHODOLOGY FOR DESIGNING COMPLEX OBJECTS AND SYSTEMS

Informational technologies - a new area of knowledge

Information technology plays a significant strategic role in the development of each country. Their significance is rapidly increasing due to the fact that IT:

- activates and increases the efficiency of the use of information resources, provides saving of raw materials, energy, minerals, materials and equipment, human resources, social time;

- realizes the most important important and intellectual functions of social processes; occupies a central place in the process of intellectualization of society, in the development of the educational system, culture, new forms of art (displayed on the screen), the popularization of world culture masterpieces and the history of human development;

- provides informational interaction among people, promotes the dissemination of mass information;

- quickly assimilated by the culture of society, eliminating many social, domestic and industrial problems, expanding domestic and international economic and cultural ties, affecting the migration of the population on the planet;

- optimizes and automates information processes during the formation of the information society;

- plays a key role in the processes of obtaining, accumulation, dissemination of new knowledge in three directions.

The first one is an information modeling that allows for a "computational experiment" even in conditions that are impossible for a natural experiment because of the danger, complexity and high cost. she second direction is based on methods of artificial intelligence. It allows you to find solutions to poorly formalized tasks, tasks with incomplete information

and incorrect outcomes by applying an analogy with the creation of metaprocesses used by the human brain. The third direction is based on the methods of cognitive graphics, that is, a set of techniques and methods of imaginary representation of the conditions of the problem, which allow you to immediately see the solution or get a clue for finding it. It opens up the possibilities for a person to learn about himself, the principles of functioning of his consciousness. In addition, in this case it becomes possible to implement the methods of information modeling of global processes, which provides the ability to predict many natural situations in regions of increased social and political tensions, environmental disasters, major man-made accidents.

General definitions

Terminology in the field of IT is annually with new concepts, abbreviations etc., that's why only the definitions of the general nature is given in this section.

The very term information comes from the Latin word, meaning "clarification, informing, presentation". The concept of "information" is widely used in the ordinary life of a modern person, so everyone has an intuitive idea of what it is. But when science begins to apply well-known concepts, it clarifies them, restricts the use of the term strictly within its scope in a particular scientific field. Thus, the concept of information in each of them is specified and enriched, becoming the subject of study of many sciences.

The concept of information is one of the primary concepts in modern science. The importance of information in the life of society is rapidly increasing, the methods of working with information are changing, the scope of application of new information technologies is expanding.

The complexity of the phenomenon of information, its versatility, the breadth of its scope and rapid development are reflected in the constant emergence of new interpretations of

the concepts of information and information technology. Therefore, there are various definitions of the concept of information, from the most general, philosophical - "Information is a reflection of the real world" - to the narrow, practical - "Information is all information that is the object of storage, transmission and transformation". Let's show some other definitions and characteristics for comparison. Information is one of the fundamental entities of the world around us. Information is the knowledge, transmitted by some people to other people in oral, written or some other way.

Information is one of the main universal properties of matter.

When talking about information, it is important to understand not the objects and processes themselves, but their reflection in the form of numbers, formulas, descriptions, drawings, symbols, images. The information itself can be attributed to the field of abstract categories, such as mathematical formulas although a working process is always associated with the use of any materials and energy costs. The information is stored as the rock paintings of ancient people in stone, in the texts of books on paper, in paintings on canvas, in musical tape recordings on a magnetic tape, in the data of the computer's memory, in the inherited DNA code of every living cell, in the human memory inside the brain etc. In order to record, store, process, distribute one of the materials is required (stone, paper, canvas, magnetic tape, electronic data carriers). In addition, energy is needed, for example, in order to operate printers, to create an artificial climate for the preservation of masterpieces of fine art, to nourish electricity circuitry, to support the work of transmitters on radio and television stations.

The term **informatization** can be deciphered as an effective use of information and computer technology in all spheres of activity as a set of measures aimed at ensuring the

full and timely use of reliable knowledge in all socially significant forms of human activity. by the public. The main purpose of informatization is providing a solution to the actual problems of society, satisfaction of demands for information products and services.

The term **technology** comes from the Greek teche + logos, meaning "skills + teachings". In the manufacturing process, technology is understood as the system of interconnected methods of processing materials and methods of manufacturing products. In general, technology is the rules of actions of using any means that are common to an entire set of tasks or task situations. The purpose of technology in industrial production is to improve the quality of products, reduce the timing of its production and reduce the cost.The production of information is aimed at the expedient use of information resources and their supply to all elements of the organizational structure and implemented through the creation of an information system.

Information technology is a set of forms, methods and means of automation of information activities in various spheres.

IT, as a study, includes methodical and methodological provisions, organizational settings, methods of using instrumental and technical means, etc. All that regulates and supports the information production and activities of people involved in this production.

The transformation of new scientific knowledge into a specific information technology is the main task of the IT as a study. Let's look at the subject of discussion and state some of the concepts:

IT is a set of scientific methods and techniques for the production of information products and services with the use of the entire variety of computing and communications;

IT is a bordering region that covers both computational technology and specific social information practices, rationalizes them at the expense of the widespread use of computer technology;

IT is a set of fundamentally new tools and methods that provide creation, processing, transmission, display and storage of information.

Information technology provides a transition from routine to industrial methods and working tools with information in various areas of human activity, allowing them to use it rationally and effectively.

There are three levels of **information technology review**:

the first level is theoretical. The main task is a creation of a complex of interrelated models of information processes, being parametric and criterion compatible;

second level is research. The main task is the development of methods that allow to automate the design of optimal concrete information technology;

third level is applicational.

Automated information technologies are aimed at increasing the degree of automation of all information operations and, consequently, to accelerate the scientific and technological progress of society.

Information systems

The term "information system" (IC) emerged alongside with the widespread use of new information technologies.

The information system carries out a collection, transmission and processing of information about the object; supplies employees of different levels of information for the implementation of the management function.

Information technology is based on the implementation of information processes. Their diversity requires the allocation of basic, typical of any information technology. The basic

technological process is based on the application of standard models and tools. Some of the basic technological processes are:

- receiving information;
- transportation of information;
- information processing;
- information storing;
- representation and use of information.

The process of obtaining information is related to the transition from the real representation of the subject domain to its description in a formal way and in the form of data representing this presentation.

In the process of transportation, the distance information is transmitted in order to make the exchange faster and organize a quick access to it, using different ways of transformation.

The process of information **processing** is used for obtaining some "information objects" from other "information objects" by performing some algorithms; it is one of the main operations while operating with information, and is a major factor in increasing its scope and diversity.

Information **storing** process involves the need for accumulation and long-term data storage, ensuring their relevance, value, security, availability.

The process of **representation** and **use** of information is aimed at solving the problem of access to information in a user friendly form.

Basic information technologies are built around the basis of basic technological operations, in addition including a number of specific models and tools. This type of technology is focused on solving a certain class of tasks and is used in specific technologies as a separate component. Among them are:

- multimedia technologies;

- geoinformation technologies;
- information security technologies;
- CASE-technologies;
- telecommunication technologies;
- CALS-technologies;
- artificial intelligence technologies .

The specifics of a particular subject area are reflected in **specialized information technologies**, for instance, organizational management, technology process management, automated design, studying, and others. Among them the most progressive are the following information technologies:

- organizational management (corporate information technologies);
- in industry and economics;
- in education;
- automated design.

CASE-technologies (Computer Aided Software Engineering) is a kind of "technological snap" that allows for automated design of information technology. Information tools provide an effective representation of the subject area; these include information models, classification and coding systems (all-Ukrainian, areal), etc. Mathematical tools include models for solving functional problems and the model of organization of information processes that provide effective decision-making. Mathematical tools are automatically converted into algorithms that ensure their implementation. Technical and software tools set the level of information technology implementation both during their creation and implementation.

CALS- technology is designed to unify and standardize the specifications of industrial products at all stages of its life cycle.

This way, specific information technology is determined by the compilation and synthesis of basic technological operations, "branch technologies" and means of

implementation. The implementation of information systems improves the efficiency of production and economic activity of the enterprise due to not only the processing and preservation of information, automation of routine work, but also fundamentally new methods of management. They are based on designing the actions of specialists during their decision-making process (methods of artificial intelligence, expert systems, etc.), using modern telecommunication facilities (e-mail, teleconferencing), global and local computer networks, etc.

Information systems are classified in the **area of application** as follows:

- C scientific research;
- IC automated design;
- IC organizational management.

Information technology of product design

The need of implementation of IT for product manufacturing can be explained by the requirements for reducing the design and preparation time for the production of new and upgraded products, the cost of design and production, the cost of long-term after-sales service . Besides, IT is required for the restructuring (re-engineering) of enterprises in accordance with modern requirements for improving the quality and competitiveness of products, restoring old markets and entering new markets. The following IT are widely used at the stages of the life cycle of electronic tools:

- CASE (Computer-Aided Software / System Engineering) - technologies,
- ERP (Enterprise Resource Planning), which provide a solution to a wide range of tasks of resource planning and management of enterprises.
- CRM (Customer Relationship Management) systems as a set of applications or as an add-on over ERP. In CRM-systems, the emphasis is on the relationship between the "client

company" and, above all, the maintenance of old customers by taking into account their individual needs and characteristics.

I.2. SYSTEM (STRUCTURAL) LEVEL OF COMPUTER DESIGN OBJECTS

The essence of the design process

The essence of the design process of products is to design constructions and technological processes for the production of new tools, which, with minimal cost and maximum performance efficiency of the functions assigned to them in the necessary conditions.

Designing any technological object means creating, transforming and presenting in an accepted form of the image of this not yet existing object.

The image of an object or its component parts can be created in the imagination of a person as a result of the creative process or generated in accordance with certain algorithms in the process of interaction between man and computer. In any case, engineering designing begins with the expressed need of society in some technical objects.

Designing includes the development of a technical proposal and/or a technical task (TT) that reflects these needs and the implementation of the TT in the form of project documentation. Typically TT is presented in the form of some documents, and it is the initial (primary) description of the object. Generally, The result of the design is a complete set of documentation, which contains sufficient information for the manufacture of the object in the specified conditions. This documentation is actually a project, or rather a definitive description of the object.

So, **designing** is the process of obtaining and transforming the initial description of an object into a final description based on the implementation of a complex of research, design and design works.

Designing of complex objects is based on the application of ideas and principles set out in a number of theories and approaches. The most general approach is a systematic approach, the ideas of which are imbued with different methods of designing of complex systems.

Projected products distinguish the following design tasks by the degree of novelty:

- partial modernization of the existing product (change of its parameters, structure and design), which provides a relatively small (several dozens of percents) improvement of one or more quality indicators for the optimal solution of the same or new tasks;

- significant upgrade, which provides significant improvement (by several times) of quality indicators;

- the creation of new products based on new principles of operation, design and production for a sharp increase (on a few levels) of quality indicators in solving the same or substantially new tasks.

Designing is a complex multi-stage process in which large teams of specialists, entire institutions and research and production associations, as well as the organization of customers who are to operate the developed equipment, can participate.

The design stages consist of separate **design procedures**, which end with a partial design decision. The **analysis** and **synthesis** of descriptions of different levels and aspects is typical for designing procedures.

The idea behind the **analysis** procedure is in determining the properties of a given (or selected) description. The analysis allows to assess the degree of satisfaction of the project solution with the given requirements and its suitability.

The idea behind the **synthesis** procedure is in creating a design solution (description) according to the given requirements, properties and restrictions.

Analysis and synthesis procedures in the design process are closely related, since both of them are aimed at creating an acceptable or optimal design solution.

A typical design procedure is **optimization**, which leads to an optimal (according to a certain criterion) design decision. Optimization provides a creation (synthesis) of the design decision, but includes a stage by stage evaluation of the characteristics (analysis).

Project procedures consist of separate **project operations**. Project procedures and operations are performed in a certain sequence, called the **design route**. Design paths can begin with lower hierarchical levels of descriptions (ascending design) or from the upper (downward design).

There is a deep interconnection between all stages of the design. Thus, the definition of the final design and development of the entire technical documentation can often not be completed before the end of the development of technology. A correction of the principal schemes, structure of the system and even output data may be required in the process of technology designing and development. Therefore, the design process is not only multi-stage, but is also repeatedly corrected as it performs, which means that the design is iterative.

Modern design is based on the systematic approach and the improvement of design processes with the use of mathematical methods andcomputers, complex automation of labor-intensive and routine design work, the replacement of layout and model simulation by mathematical modeling, the use of effective methods of multivariate design and optimization, as well as improving the quality of design management.

Methodology of a systematic approach to the task of the design of complex systems

The system approach allows finding the optimal, in the broadest sense, solution of the design task due to the comprehensive, holistic consideration of both the projected product and the design process itself, and can lead to genuinely creative innovative solutions, including large inventions and scientific discoveries.

The main means of automation of design are the ECD (electronic counting devices) and other technical devices, that create the necessary basis for the full realization of the potential opportunities of the system approach. The system approach is becoming more widespread in designing and managing.

The point of the system approach is that the object of design or management is seen as a **system**, that is, as a unity of interconnected elements that combine into a single thing and operate in the interests of realizing a single goal. The system approach requires examining each element of the system in the relationship and interdependency with other elements, revealing the patterns inherent in this particular system, identifying the optimal mode of its operation.

A systematic tool for implementing a systematic approach to the study, design or management of a complex process is called **system analysis**, which means a collection of techniques and methods of studying objects (processes) by presenting them in the form of systems and their further analysis.

The design process is a multi-layered hierarchical process for optimizing solutions in each layer from the point of view of the a system approach to design automation. The principle of hierarchy in design and management, as well as the principle of integrity, necessitate the construction of a system of criteria when partial criteria are used to solve problems of

the lower level of management (subsystems), logically match with the criteria used at a higher hierarchical level. The source values are compared in the process of designing and managing, that is, the result of the system's functioning with the criterion.

Thus, the **system** is a rather complex object, which can be disassembled into constituent elements or **subsystems**. Elements are informationally related to each other and to the environment surrounding the object. A set of relationships forms the **structure** of the system. The system has an algorithm for functioning, aimed at achieving a certain goal.

System approach to the task of automated design of a technological process

During analysis of complex processes, when it is not possible to find internal connections in the system, the principle known in the cybernetics a the "black box" is used. This idea lies in the fact that, without having information about the being, about the internal structure of the process, only the dependence of the output variables from the inputs are used for its mathematical description.

The concept of a "black box" refers to the basic concepts of cybernetics, helping in the study of the behavior of systems, that is, reactions to various external influences, abstract from their internal order. Many systems, especially large ones, are so complex that even with full information on the state of their elements, it is virtually impossible to link it with the behavior of the system as a whole. In these cases, the presentation of such a complex system in the form of some "black box", functioning in the same way, makes the construction of a simplified model easier.

Analyzing the behavior of the model and comparing it with the behavior of the system, a number of conclusions about the properties of the system itself can be made. When they coincide with the properties of the model,working hypothesis

about the predicted structure of the researched system can be made.

Let's assume, that the input of the system can be influenced by X, and the output indicators have a P quality. Looking at the behavior of this system for a long time and, if necessary, performing some active experiments on it, that is, changing the input in some way, a certain level of knowledge of the properties of system can be achieved, in order to be able to predict a change in its initial output values for any given change in the input. The method that uses the "black box" is widely used to solve the problems of modeling controlled systems (especially in the study of complex technical objects) in cases where the behavior of the system rather than its structure is of interest. Statistical methods of optimization are usually the most suitable in these situations, since neither the technologist, nor the ECD are capable of taking into account the total effect of many different factors, often associated with complex dependencies, in a number of cases.

The methodology of system analysis is quite universal and can be used both for the design process as a whole, and for individual phases and stages of designing. During the transition of general design, the content of goals, and decisions will change on certain stages, but the overall sequence of analysis stages will remain.

Traditionally, the design of complex technical systems is divided into following phases or stages of development:

- technical task for the designed object;
- scientific research work;
- sketch design;
- technical design;
- working design;
- technology of manufacturing and testing of the designed object (prototype or batch);
- making corrections (if necessary).

Technical task. At the stage of development of the technical task (TT) the following tasks are solved: the search and selection of the necessary scientific and technical information (about prototypes, patent data, etc.) from the corresponding database.

The new scheme (device) can either have or not have analogues. In the case if analogues exist, you can proceed to the design stage of the device (system). But, as a rule, there is no analogue or the developed system has to exceed the known analogue, therefore a SRW (scientific research work) is required; an analysis of the selected information and formulation of technical requirements for the projected object is formulated on its basis. At this stage, information search and document processing operations can be automated. Certain parts of the auxiliary actions of information analysis can be automated as well, for example, grouping them according to certain features, selecting the least or most compatible with each other options etc. In addition, some questions are solved and made up in specific documents, for example: the transfer of functions performed by the device; development of the structural scheme of the device; definition of the characteristics of individual nodes; development of algorithms of executed operations.

SR stage. This is a pr-design stage. This is one of the most important stages. In order to solve the tasks at this stage, a usage of ECD is required. These are the so-called automated systems of scientific research (ASSR). At the stage of SRW, the following tasks need to be solved: formulation of the criteria of quality and management; management of scientific experimentation; carrying out a passive or (and) active experiment with the processing of their results; development of mathematical models and their identification by experimental data; working out the technological processes of manufacturing of objects in order to find norms for parameters that provide

optimal output quality indicators; the formation of a generalized quality criterion, which includes all the partial quality indicators. A generalized criterion is than taken as the target function while solving the optimization problems; solving the optimization problems. The variation of the input and control parameters of the technological process in the framework of the established norms (accesses) is carried out in order to obtain the optimal quality criterion; a search for a fundamental possibility of building a system; a development of new technical means, including means of control and measurement.

A Technical Proposal (TP) is issued as a result of a SRW. CAD programs, methods and algorithms can b applied, even though the SRW is an independent stage.

Stage of sketch design. The following tasks are being solved at this stage: a sketch of a projected system (device) with a detailed development of its capabilities is developed, a search and selection of more detailed information is being carried out. Based on the analysis of the received information, the preliminary design decisions are taken and the first project documents are prepared. Various calculations are produced for the development of design documents, the content, the volume and complexity of which depend on the characteristics of the object being projected.

The works of this stage are mostly being automated, and their automation gives the greatest technical and economic effect by optimizing design decisions. The automation of these works is achieved through the application of optimization mathematical methods.

Stage of the **development of the object** of the technical project. At this stage, the decisions, made in the sketch design , are being detailed and clarified, and the new more precise project documents are created. A search, selection and the analysis of the output information (mainly technical and

techno-economical) is carried out once again. Numeric calculations are carried out as well, but from other, more accurate methods. These works can be largely automated. Most of the documents formed at the stages of sketch and technical design are used only for working design and are not included in the working and operational documentation. The information generated in the considered stages serves as the source for the working design. This means that it is expedient to create banks of temporary information based on the projected object in the conditions of automated design.

Working design. At the stage of working design, the main type of work to be done is the arrangement of design solutions in the form of drawings, specifications for them and operational documentation for the object. Modern means of computer technology allow to fully automate the execution of drawings and specifications, and to a certain extent - the formation of operational documentation. If the design automation system carries out not only the production project but also the design of the technology, then it is advisable not to make drawings and specifications in the traditional form, but to transfer information to the designers-technologists on the computer carriers as a database of the projected object.

Designing **technologies for manufacturing** of a designed object. At this stage, traditionally, a work in the process of technological preparation of manufacturing of a product or its components and parts is performed at the enterprise-manufacturer. When designing the technology, the following tasks are carried-out: search and selection of and output information (about the object to be manufactured; about the technological equipment of the enterprise; about the technological and labor standards); analysis and processing of data in order to determine the processing routes, the sequence of technological operations and their modes of operation, the needs of the instrument and measuring equipment, the creation

of a special tool; arrangement of the corresponding technological documentation.

The works referred to in the Technical Task and Technical Project are identical to many operations during the design of the product. The peculiar design of technologies requires original calculations and solutions for various types of technological operations. Nevertheless, the methods of formalizing most of these works exist, therefore, they can be automated.

Automation of information processing of operations and the management of processes of information usage at all reviewed stages of designing is the actual point of the functioning of modern CAD.

I.3. PRINCIPLES OF CONSTRUCTION AND FUNCTIONING OF CAD

Principles of functioning CAD

When creating and operating CAD using the following principles:

1. The **principle of system unity** is that, when creating, functioning and developing CAD, connections between subsystems must ensure the integrity of the system.

2. The **principle of inclusion** ensures the development of CAD on the basis of requirements that enable this CAD in a higher-level CAD.

3. The **principle of development** means that CAD should be created andfunction with the additions, upgrades and updates of subsystems and components.

4. The **principle of complexity** ensures the interconnection between the design of elements and the entire object at all stages and stages of design.

5. The **principle of informational unity** is the use in the subsystems, facilities and components of the CAD system of uniform conditional notations, terms, symbols, problem-

oriented languages, methods of making information conforming to accepted normative documents.

6. The **principle of compatibility** is that the simultaneous operation of all subsystems of CAD should be ensured while maintaining the openness of the system as a whole.

7. The **principle of standardization and inventory** is to unify, standardize the subsystems and components that are invariant to the industries and objects being designed.

8. The **principle of the dialogue** is that there is simultaneous use by the designer of manual, automated and automatic project operations, its active influence in the process of design solutions.

9. The **principle of accumulation of design experience** is the availability and replenishment of the archive of design procedures and design decisions, mathematical models (MM), algorithms, theoretical and experimental data, etc.

Composition and structure of CAD

Structural components of CAD, which are strictly related to the organizational structure of the project organization, are subsystems, in which, with the help of specialized complexes of the means, the functionally completed sequence of CAD tasks is solved.

CAD are devided to:

Designing subsystems that have object orientation and implement a separate stage (design stage) or group of directly related project tasks. Example: sketch design of products, design of body parts, etc.

Servicing subsystems have a common system use and provide support for the functioning of the designing subsystems, as well as the design, transmission and output of the results obtained therein. Example: automated data bank, documentation subsystem, and graphical input / output.

The **subsystem** consists of components of CAD, combined by a target function common to this subsystem, and which ensure the functioning of this system.

A **component** is an element of support that performs a separate function in the subsystem:

Methodical support - documents in which the composition, the rules for the selection and operation of design automation facilities are displayed.

Linguistic support - design languages, terminology;

Mathematical support - methods, mathematical models, algorithms;

Software - documents with text programs, programs on carriers and operational documents;

Technical support - means of computing and organizational engineering, data transmission, measuring and other devices;

Information support - documents describing standard project procedures, typical decisions, typical elements, etc .;

Organizational support - provisions, instructions, orders, staff schedules and other documents that regulate the organizational structure of CAD divisions.

General characteristics, definition of Technical support of CAD

Technical support of CAD is a complex of technical means , on the basis of which the entire automated design process is physically implemented: from input and preparation of output data to obtaining the finished project documentation.

In essence, technical support of CAD is the material basis of automated design and, together with the software (CAD software) creates the physical environment in which the other types of CAD support (mathematical, informational, linguistic, etc.) are implemented.

It should be noted that the problem of selecting technical support of CAD for any particular CAD is a very

important and responsible step in the design or operation of this CAD. This is due to the fact that the technical support of CAD with software CAD is the most expensive component of CAD and largely determines the effectiveness of the whole system as a whole.

Requirements for technical support of CAD

Requirements for technical support of CAD can be divided into four categories: system; functional; technical; organizational and operational.

System requirements determine the spectrum of properties, parameters and characteristics of the technical support of CAD as a technical system. System requirements for technical support are following: efficiency, universality, compatibility, flexibility and openness, reliability, accuracy (reliability), security, the possibility of simultaneous operation of a sufficiently wide range of users, low cost.

Functional requirements determine the properties of the technical support of CAD in terms of performance CAD functions. They are nominated for CAD and should provide: implementation of mathematical models; tasks of decision making and design procedures; archives, libraries of design decisions and typical elements; data retrieval system, providing visual information; work with graphic images and models; parallel development of individual nodes; interconnection of stages of designing; work of the user both in batch mode and in dialog mode with the ability to switch from one mode to another at any stage of designing; documentation of design results; delivery of results to technological equipment (recording of the program for equipment, etc.).

The **technical requirements** determine the parameters and characteristics of the technical support of CAD and individual technical support in the functioning of CAD and expressed in the form of quantitative, qualitative and nomenclature values of characteristics and parameters. The

main characteristics and parameters include the following: performance, speed of devices, information coding system; capacity of storage devices, types of data carriers; Types of interfaces for hardware connectivity.

Organizational-operational requirements include technical aesthetics, ergonomics, safety (labor protection), organization of operation and maintenance of the CAD system.

The most general requirements (mostly systemic and functional) lead to TP on CAD. More detailed and specified system and functional requirements, as well as technical and organizational-operational requirements, indicate the technical problems of complex facilities.

The main components of technical support of CAD

Currently, in the technical support of CAD, it was decided to allocate two groups of technical means (fig. I.3.1):

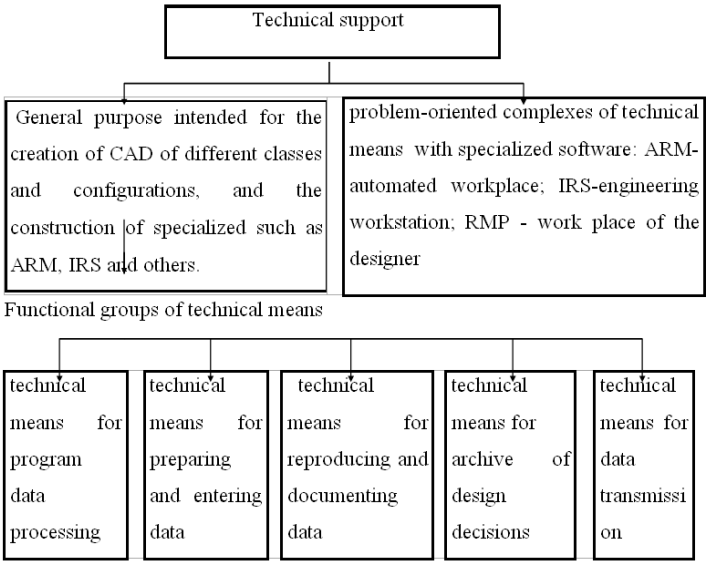


Fig. I.3.1.The main components of technical support of CAD

- technical means of general purpose, intended for creation CAD of different classes and configurations and the complexing of specialized technical means type of ARM, IRS and others;

- problem-oriented complexes of technical means with specialized software: ARM-automated workplace; IRS-engineering workstation; RMP - work place of the designer.

I.4. MATHEMATICAL SUPPORT FOR COMPUTER DESIGN

General characteristics

The mathematical support of automated design is called the combination of mathematical models of design objects, as well as methods and algorithms of operations and procedures.

The generalized structure of the MS CAD can be shown in the following form (See Fig.). As can be seen from this scheme, the whole set of mathematical models of objects, which are projected by the nature of their properties, are divided into functional and structural models.

Functional models are intended for the display of physical processes occurring in the object during its operation and establish connections between input, output, control and external parameters by means of functional dependencies, functionals, operators, probable dependencies, etc. Functional mathematical models (MM) together with some criteria for assessing the quality of the object's operation form the basis of a functional description of the design object (functional aspect).

Structural MM are designed to reflect the structural properties of the design object. Distinguish structural MM: topological and geometric.

Topological MM cover the composition and connection of elements of the design object. They are most often used to describe objects that consist of a large number of individual

elements when solving tasks of attachment of constructive elements to certain spatial positions (example of the problem: layout, tracing of connections), or to relative moments of time (for example - in the development of technological processes). Topological models can take the form of graphs, tables, lists, matrices, etc.

Geometric MM reflect spatial relationships and forms of the projected object and its component parts. Geometric MMs can be expressed by a set of equations of lines and surfaces, graphs and lists, etc. On the basis of topological and geometric MM are morphologicappointment of the design object.

he effectiveness of CAD, in many respects, is determined by the quality of mathematical support, since the choice of MS often determines the quality and design terms, as well as costs for it.

Generalized structure of mathematical support of CAD

Generalized structure of mathematical support of CAD is shown on the figure I.4.1.

The requirements for accuracy, reliability, economy, versatility and adequacy are put forward to the MM.

1. **Accuracy:** estimated by the degree of coincidence of real and estimated parameters of the object. The evaluation is carried out using data MM and algorithm. Let the quality of the design object displayed in MM are estimated by the vector of output parameters $Y = (y_1, y_2, \ldots, y_m)$. Then, knowing the actual and calculated using the MM value of the *j*-th output parameter through Y_{jv} and Y_{jmod} , respectively, define the relative error E_j of the calculation of the parameter Y_j as

$E_{j} = (Y_{j\,mod} - Y_{j\,v}) / Y_{j\,v}$.

The estimate $\varepsilon = (\varepsilon_1, \varepsilon_2, \ldots \varepsilon_m)$ is obtained. If necessary, the reduction of this estimate to the scalar use of any norm of the vector ε, for example

$\varepsilon_M = \| \varepsilon \| = \max \varepsilon_j$. (6.2) $j \in (1: m)$

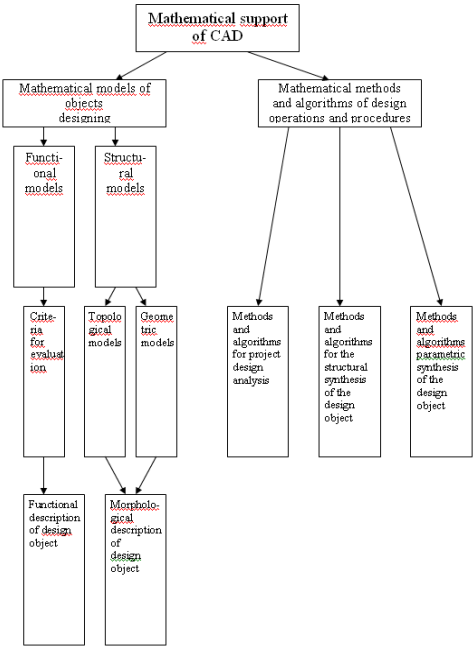


Fig. I.4.1.Generalized structure of mathematical support of CAD

2. **Reliability**: it is necessary to use such MM and algorithms which have strict justification of use.

3. E**fficiency** of MM is characterized by the cost of computing resources (by the cost of machine time T and memory M) for its implementation. The less T and M, the more economical the model. Instead of the values of T and M, which depend not only on the properties of the model but also on the characteristics of the computer, you can use other values: the average number of operations that are performed in one call to the model, the dimension of the control system, the number of internal parameters that are used in the model.

4. **Versatility:** involves the use of objects of the same type without significant alteration of MM and algorithms.

5**. Adequacy** of the MM is the ability of the MM to display properties with an error that would not be greater than the given one. Since the output parameters are functions of the vectors of the parameters of the external Q and internal X, the error ε_J depends on the values of Q and X. As a rule, the internal values of the MM are determined from the condition of minimizing the error ε_J in some point Q_n of the space of external variables, while using the model calculated vector X with various values of Q. Adequacy of a model, as a rule, takes place only in a limited field of change of external variables-area of adequacy (VAA) of a mathematical model:

$OA = \{Q \mid \varepsilon_M \le \delta\}$,

where $\delta > 0$ - the given constant is equal to the maximum permissible error of the model.

Functional description of design objects

Functional models of the design object or its elements are dependencies that connect the output characteristics with the input, internal (controllers) and external parameters. In the general case, functional models are written in the form of a ratio$Y = F(t, s, x, Q)$,

where $Y = (y_1, y_2, y_3, \ldots y_n)$ - vector of output parameters;

$X = (x_1, x_2, x_3, \ldots x_n)$ - vector of internal (controlled) parameters;

$Q = (q_1, q_2, q_3, \ldots q_n)$ - vector of external parameters;

t - time;

$S = (x, y, z)$ - vector of spatial coordinates.

The construction of a functional MM object is possible in the case when the morphological description of the design object is already executed, that is, the description of the composition of its elements and their interaction.

Classification of functional models

1. Depending on the method of construction: - theoretical;

- Experimental.

2. In the form of links between the parameters of the model: - analytical;

- algorithmic.

3. Depending on the consideration of random factors:

- deterministic;

- scholastic.

4. Depending on the type of given parameters of the model: - constant;

discrete

5. Depending on the features (type) of the equations included in the model: - linear; nonlinear

6. Depending oncounting or not taking into account time: - static; -dynamic

7. In relation to the hierarchical level: - micro-models; - macromodels; - metamodels.

Kinds of functional models

1. Mathematical models in the form of differential equations in partial
derivatives (distributed models). Such models reflect processes that
proceed in the general case in 3-dimensional space and in time they are
have the following look:

$\Phi(S, X, Y, Q, \partial Y/\partial S, \partial^2 Y/\partial S^2, \ldots, t) = 0,$

where Φ- communication operator between variables and their derivatives. Examples of Distributed Models:

- thermal equation for simulation of the thermal mode of the engine of internal combustion;

- diffusion equation for simulation of cooling processes;

- the equilibrium equation, when simulating tasks of statics and dynamics of machines.

2. Mathematical models in the form of ordinary differential equations (concentrated models).

$\psi(\partial Y/\partial t, X, Y, Q, t) = 0$.

Examples of concentrated models:

Differential equation of the curved axis of the beam on an elastic basis during the mo-division of the stressed-deformed state of the machine nodes, etc.

3. Mathematical models in the form of transcendental and algebraic equations:

$F(Y, X, Q, t) = 0$ - transcendental,

4. Mathematical models in the form of logical equations: used in automation systems, relays, etc.

5. Mathematical models of stochastic processes - mass maintenance systems (computers, databases, shops, gas stations, etc.).

Methods of constructing functional models

By their very nature, MM is divided into theoretical and experimental (empirical) MM. All other classifications are derivatives from the above. Let's consider the methods of constructing these MM.

Methods of constructing theoretical functional models:

In order to obtain theoretical distributed mathematical models, fundamental physical laws are used: laws of mass conservation, energy, amount of motion. Then they are supplemented by boundary conditions and MM -is ready

The basis of obtaining lumped models is also known laws, principles and hypotheses of a less general nature: the basic law of the dynamics of the translational and rotary motion, the principle of velocity generation, the Hooke's law, the hypothesis of plane cross sections, and so on.

Methods of constructing experimental functional models

To obtain static models, we use the mathematical apparatus of the theory of experiment planning, in which MM

is obtained in the form of an algebraic equation of the form Y = F (Q) - the response function.

I.5. EXPERIMENTAL MATHEMATICAL MODELS OF DESIGN OBJECTS

Planning an experiment

Theoretical studies play an important role in the process of knowing the objective reality, since they allow deeply plunge into the essence of natural phenomena, create a scientific picture of the world.

The solution of problems by mathematical methods is carried out by mathematical formulation of the problem, the choice of a method for studying a mathematical model, analysis of the result.

The mathematical formulation of the problem appears in the form of numbers, geometric images, functions, systems of equations, etc.

The mathematical model represents a system of mathematical relations - formulas, functions, equations that describe the object being studied.

Stages of Mathematical Modeling:

- statement of the problem, definition of the object and purpose of the study, task of signs of studying objects, etc .;
- choice of type of mathematical model (often several models are constructed and the best is chosen);
- description of the transformation of input signals into the output characteristics of the object (for example, using algebraic dependencies);
- studying the quality of the model.

Recently, there is a continuous expansion of the application of methods of mathematical planning of the experiment. These methods are successfully used to increase the efficiency of experimental studies, to find optimal technological regimes of production processes, to choose the

design parameters of a product, the composition of a multicomponent mixture, etc.

In experimental studies, they deal with the object of research. Objects of research can be devices, technological lines, various products, etc.

A rather common model of a research object is the cybernetic system depicted in the diagram (fig. I.5.1).

Fig. I.5.1. Model of a research

For such a system, inputs are distinguished - controlled factors x_1, x_2, ..., x_p, corresponding to the effects on the system, and outputs y_1, y_2, ..., y_n, (numerical characteristics of the research objectives) - parameters (criteria) of optimization

Each factor can take in the experiment one of several values, called levels. A fixed set of factor levels determines one of the possible states of a cybernetic system. At the same time, this set represents the conditions for conducting one of the possible experiments.

Each fixed set of factor levels corresponds to a certain point in the multidimensional space of factors, which is called factor space.

Experiments can not be implemented at all points of the factor space, but only at points belonging to the permissible domain of the factor space *G* (fig. I.5.2).

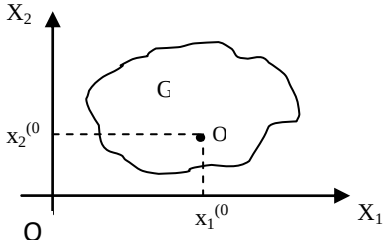


Fig. I.5.2. Factor space fo experiment

The system responds differently to different sets of factor levels. However, there is a certain correlation between the levels of factors and the response (feedback) of the system.

Function ψ that connects optimization parameter with factors is called the response function, and the geometric image corresponding to the response is the response surface (fig. I.5.3).

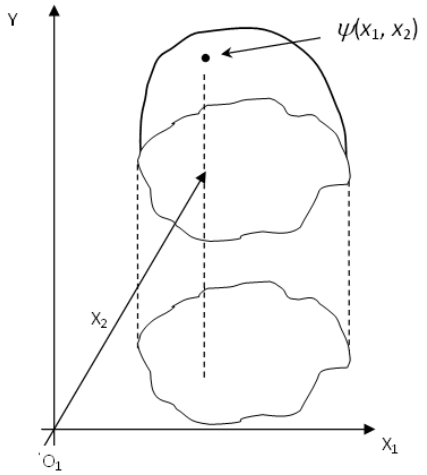


Fig. I.5.3. Response surface

The researcher is not aware in advance of the appearance of the dependences ψ_j. He has to get approximate equality: $\hat{y}_j = \hat{\psi}_j(x_1, x_2, ..., x_k)$

$j = 1, 2, ..., l.$

according to the experiment.

The experiment must be set so that with a minimum number of experiments, varying the value of independent variables in specially formulated rules, construct a mathematical model of the system and find the optimal values of the properties of the system.

The choice of factors, optimization parameters and models takes place taking into account the purpose of the study.

Distinguish quantitative and qualitative factors. Quantitative ones can be measured, weighed, etc .; qualitative - no, but for them it is possible to construct a conditional scale to distinguish factor levels.

On the other hand, factors can be controlled and uncontrolled.

Under the control are the following input variables (factors) whose values in the experiment are known at each time point. Thus, in the study of the technological process, all variables that determine the state of the process and the values of which are recorded using the appropriate measuring devices are controlled. Controlled variables, in turn, can be divided into managed and unmanaged. Managed are called variables (factors) for which a purposeful change of their values is possible during the experiment. Variables for which such a change is not possible are called unmanaged.

Uncontrolled factors include such input variables whose values can not be determined during an experiment or those that have an impact on the results of the experiment, but even the existence of which the experimenter does not have information.

The characteristic of the goal of an experiment, quantified, is called an optimization parameter (optimization criterion, target function).

To the optimization parameter put forward a number of requirements:

- efficiency in terms of achieving the goal (that is, the optimization parameter should evaluate the functioning of the system as a whole, rather than its individual subsystems);
- universality (the ability to comprehensively characterize the object of research);

- a quantitative expression in one number;
- the presence of physical sense;
- Simplicity and affordability of the calculation.

System properties can be described by different models. To select a model, we formulate requirements:
- adequacy (ie the ability of the model to predict the results of the experiment in some area with the required accuracy);
- content (ie the model should well explain the already known facts, identify new ones and predict the future behavior of the system);
- simplicity (this is a natural requirement: the model is simpler, therefore, in other equal conditions, it is better).

Depending on the task setting, different models can be used. Often, explicit functional dependences of the form are used:

$$y = f(x_1, x_2, ...x_p, \beta_1, \beta_2, ...\beta_m, \varepsilon) \qquad \text{(I.5.1)}$$

where f - some function, called regression function; $x_1, x_2, \ldots x_p$ - independent variables (factors); $\beta_1, \beta_2, ...\beta_m$ - unknown parameters; ε - is a random component. The latter is introduced into the model when the data show a noticeable variation of a random nature. It is often assumed that ε is an additive in model (I.5.1), that is, (I.5.1) takes the form:

$$y = f(x_1, x_2, ...x_p, \beta_1, \beta_2, ...\beta_m) + \varepsilon \qquad \text{(I.5.2)}$$

The relations (I.5.1), (I.5.2) are called regression models

To independent variables (factors) $x_1, x_2, \ldots x_p$ are given one or another value, while experimentally obtaining the corresponding values of y . Then (5.2) goes into the system of relations from which the parameters $\beta_1, \beta_2, ...\beta_m$ are determined. Due to the presence of a random component, the

parameters $\beta_1, \beta_2, ...\beta_m$ can only be estimated (and not precisely defined). In this case, the estimates $b_1, b_2, ...b_m$ of the corresponding parameters are obtained, and in reality, instead of the model (I.5.2), they operate with an approach $\hat{y}$ to it:

$$\hat{y} = f(x_1, x_2, ...x_p, b_1, b_2, ...b_m)$$

If a function f is a polynomial, then $b_1, b_2, ...b_m$ are called regression coefficients, and the function $\hat{y}$ takes the form:

$$\hat{y} = b_0 + \sum_i b_i x_i + \sum_{i,j} b_{ij} x_i x_j + ... \qquad \text{(I.5.3)}$$

If a model is chosen, that is, the type of the dependence y from x is chosen and the corresponding equation is written, then it is necessary to plan and carry out an experiment for the estimation of the numerical values of the coefficients of this equation in the research area of the factor space.

Evaluation of regression coefficients by the method of least squares

According to the results of experiments on the object of study the certain kind of mathematical model can be obtained. In particular, it can be a regression model with the required function as a polynom of a certain order - the so-called polynomial regression model.

The quality of the approximation regression model to the real object depends not only on experimental data, but also upon the method of model building. The **method of least** squares is often choosed for this purpose.

Found equation is exposed to statistical analysis (audited sustainability variance at different points of phase space, the significance of the coefficients, the adequacy of regression model). This is called a regression analysis.

The method of least squares

Let n experiments are performed, in each of which vector of independent variables (factors) $x = (x_1,\ldots x_p)$ has certain values, and thus some values of the dependent variable y are obtained. Let $x^i = (x^i_1,\ldots, x^i_p)$ set of values of the dependent variables that were given to them in the *i-th* experiment, y_i - the corresponding values of the dependent variable ($i = 1,2, \ldots, n$). According to the method of least squares (MLS) as an estimation of vector parameters $\beta = (\beta_1,\ldots,\beta_m)$ is taken vector $b = (b_1,\ldots, b_m)$ (another designation - $\hat{\beta} = (\hat{\beta}_1,\ldots,\hat{\beta}_m)$) where the sum

$$S(\beta) = \sum_{i=1}^{n} \left[y_i - f(x^i;\beta) \right]^2 \qquad \text{(I.5.4)}$$

takes a minimum value for $\beta \in R^m$, where R^m - m-dimensional Euclidean space.

If the regression function f is differentiated by the parameters $(\beta_1,\ldots,\beta_m)$, then the necessary condition for a minimum of $S(\beta)$ is the fulfillment of equalities

$$\frac{\partial S(\beta)}{\partial \beta_j} = 0,\ j = 1,2,\ldots,m\,. \qquad \text{(I.5.5)}$$

The system (I.5.5) consists of equations whose number equals the number of unknown of system – the coefficients $b_1, b_2, \ldots, b_m$. Such system is called the system of normal equations or the normal system.

Solution of the problem of minimizing the function $S(\beta)$ here will be given for the particular, but a very important case of model (I.5.5). Specifically, we assume that $p = 1$, so the vector of independent variables x is a scalar variable. Further will be considered that $m = 2$. Besides, instead of signs β_1, β_2 for dependance parameters will be used more widespread designation β_0, β_1. Will be also made a very important suggestion that f is a linear function of the parameters β_0, β_1.

We will consider in function f variable x is only in degree 1. Thus, focus will be on such view of regression function:

$$f(x) = \beta_0 + \beta_1 x, \qquad \text{(I.5.6)}$$

and thus we will study following particular case of model (I.5.5):

$$y = \beta_0 + \beta_1 x + \varepsilon, \qquad \text{(I.5.7)}$$

where, in accordance with the above, x and y - respectively, the independent and dependent variables, β_0, β_1 - model parameters, ε - random component of model.

Dependence (I.5.7) called the **simple linear regression**.

Let's go directly to the problem of estimation parameters β_0, β_1 by the experimental data. Let the independent variable x in experiments takes values $x_1,\ldots,x_n$ (last applied designation somewhat different from above - where x_1, x_2,... denote different independent variables, but now only one independent variable, with $x_1,\ldots,x_n$ is its values; in previous designations we had to write $x_1^1,\ldots,x_n^1$), and the dependent variable y - respectively, $y_1,\ldots,y_n$. In this case the problem of minimizing the function $S(\beta)$ becomes:

$$S(\beta)=S(\beta_0, \beta_1)= \sum_{i=1}^{n} \left[y_i -(\beta_0 + \beta_1 x_i)\right]^2 \to min, \qquad \text{(I.5.8)}$$

where the minimum is taken over all value β_0, β_1 for fixed $x_1,\ldots,x_n$ and $y_1,\ldots, y_n$. Let solution of the problem (8.5) be (b_0, b_1),, and appropriate assessment regression function (8.3) - $\hat{y}$, i.e.

$$\hat{y} = \hat{y}(x) = b_0 + b_1 x. \qquad \text{(I.5.9)}$$

The following figure shows schematically the linear regression and a set of experimental points (x_i, y_i). Also depicted vertical segments that connect these points and the line (fig. I.5.4).

It is easy to understand that in any way of drawing of a straight line in order to approximate relationship between x and y will occur deviation of experimental points from this line (if all these points not lie on a straight line, but the latter almost

never happens). It is convenient to measure these deviations as differences ordinates of corresponding experimental points and points on straight at values $x = x_1, \ldots, x_n$, i.e. algebraic values of vertical segments shown on fig. 1. If the approximation is performed according to the MLS, then the sum of the squares of the lengths of these segments will be the lowest possible.

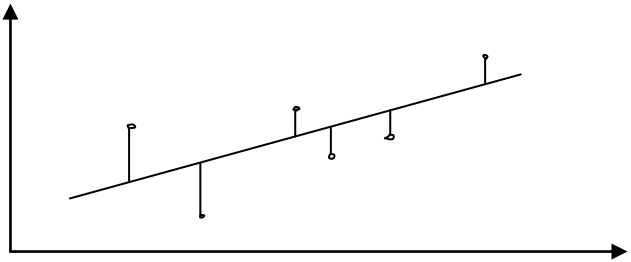


Fig.I.5.4. Geometric interpretation of the method of least

Clearly, that β_1 and β_0 are, respectively, the angular coefficient and constant term (a segment on the vertical axis at $x = 0$) line, and *b1* and *b0* - their estimated values, obtained from experimental data. This last is, accordingly, the angular coefficient and constant term in line equation (I.5.9).

To solve the problem (I.5.8) let's calculate partial derivatives of function $S = S(\beta_0, \beta_1)$ by β_0, β_1. We have

$$\partial S / \partial \beta_0 = -2 \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_i),$$

$$\partial S / \partial \beta_1 = -2 \sum_{i=1}^{n} x_i (y_i - \beta_0 - \beta_1 x_i).$$

Equating the found derivatives to zero and performing appropriate simplification we arrive at a system of two equations, where are unknown parameters β_0, β_1:

$$\beta_0 n + \beta_1 \Sigma x_i = \Sigma y_i,$$
$$\beta_0 \Sigma x_i + \beta_1 \Sigma x_i^2 = \Sigma x_i y_i, \qquad \text{(I.5.10)}$$

where to facilitate entry summation index is omitted (hereinafter in similar situations mark Σ means summing over all possible values of the summation index, in this case from 1 to *n*). This system is partial, the system of normal equations (I.5.5) and has the same name. You can strictly prove that normal system solution (I.5.10) and indeed the solution of the problem of minimizing (I.5.8).

The normal system of equations (I.5.10) is always compatible, regardless of whether it's determinant is 0 or not. Equality to zero of determinant can happen when, and only when all the observations are conducted only in one point x. In this case, this system has many solutions, each of which can be found from the equation

$$\beta_0 n + \beta_1 n x = \Sigma y_i, \qquad \text{(I.5.11)}$$

The main case is one in which the determinant of the system (5.10) is not equal to 0. In this case, the system has a unique solution for which (for some further goals) is convenient to introduce the following notation:

$S_{xy} = \Sigma(x_i - \bar{x})(y_i - \bar{y})$, $S_{xx} = \Sigma(x_i - \bar{x})^2$, $S_{yy} = \Sigma(y_i - \bar{y})^2$,

where, as above, the summation index is omitted. Let's also indicate $\bar{x}$ and $\bar{y}$ the average of the independent and dependent variable respectively:

$$\bar{y} = (y_1 + \ldots + y_n)/n\,, \quad \bar{x} = (x_1 + \ldots + x_n)/n.$$

Then we have the following expression for the solution of the system (I.5.10):

$$b_1 = S_{xy} / S_{xx}\,, \qquad \text{(I.5.12)}$$

$$b_0 = \bar{y} - b_1 \bar{x}\,. \qquad \text{(I.5.13)}$$

Finally, in the case of simple linear regression model of relation between the target function *y* and independent variable (factor) *x* is given by equation (I.5.9) in which the coefficients b0, b1 determined by equalities (I.5.12), (I.5.13).

Complete and fractional factor experiments

Let some experimental research be carried out. Each of the different values that accepts a variable X_i in the experiment is called the level of this variable. The number of different levels of the factor X_i is denoted by S_i.

An experiment, in which the levels of each factor are combined with all levels of other factors, is called a complete factor experiment (CFE).

A complete factorial experiment is written in the form: $S_1 \times S_2 \times ... \times S_k$, since the number of different points or different experiments $N_1 = S_1 \times S_2 \times ... \times S_k$.

An experiment plan is called an incomplete or fractional factor plan, if the number of different points $N_1 < S_1 \times S_2 \times ... \times S_k$.

Consider the response function

$$\eta = f(X_1, X_2, ..., X_k) \qquad \text{(I.5.14)}$$

Let the number of different values that can take a variable X_i $(i = 1,2,...,k)$ in all experiments is equal to two, that is, $S_i = 2$. In other words, the variable X_i in each experiment accepts one of two possible values, or they say, varies on two levels. Let's denote them X_{i1} and X_{i2}. We will assume $X_{i1} < X_{i2}$, then X_{i2} is called the upper level of the factor, and X_{i1} - the lower one.

We introduce coded variables: $x_i = \frac{X_i - X_i^0}{S_i}$, $i = 1,2,...,k$

where $X_i^0 = \frac{X_{i1} + X_{i2}}{2}$ $i = 1,2,...,k$;

$S_i = \frac{X_{i2} - X_{i1}}{2}$ $i = 1,2,...,k$.

Obviously, the coded variable x_i $(i = 1,2,...,k)$ in each experiment may have a value of 1 or -1. We call these values upper and lower levels. Without limitation of generality we can assume that in the expression (9.1) the variables $X_1, X_2,..., X_k$ are given in coded form:

$$\eta = f(x_1, x_2,..., x_k) \qquad \text{(I.5.15)}$$

Consider the case when in the expression (I.5.15) the number of independent variables $k = 2$, i.e. $\eta = f(x_1, x_2)$.

All possible combinations of levels of variables x_1 and x_2 in CFE 2^2 are presented in Table I.5.1:

Table I.5.1

Number of experiment	Matrix of independent variables				Research option	Observation
	x_0	x_1	x_2	$x_1 x_2$		
1	1	-1	-1	1	(1)	Y_1
2	1	1	-1	-1	a	Y_2
3	1	-1	1	-1	b	Y_3
4	1	1	1	1	ab	Y_4

Here the symbol (1) means that both factors are in the lower level; a - x_1 in the upper; b - x_2 in the upper; ab - both at the upper level. This is CFE 2^2. Often, it is believed that the response function has the form:

$$\eta = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_{12} x_1 x_2 \qquad \text{(I.5.16)}$$

Scheme CFE 2^2 can be depicted in the form:

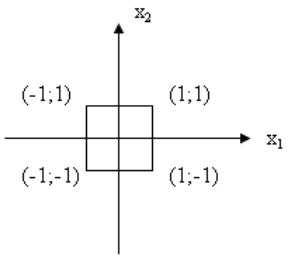


Fig.I.5.5. Scheme CFE 2^2

It is easy to see that observations y_1, y_2, y_3, y_4 are performed at the vertices of the square. The coefficients (I.5.16) are the least squares method

Let's consider the case of CFE 2^3. In this case: $\eta = f(x_1, x_2, x_3)$. All different combinations of levels of variables x_1, x_2, x_3 are presented in Table I.5.2.

Table I.5.2

Matrix of independent variables								Research option	Observation
x_0	x_1	x_2	x_3	$x_1 x_2$	$x_1 x_3$	$x_2 x_3$	$x_1 x_2 x_3$		
1	-1	-1	-1	1	1	1	-1	(1)	y_1
1	1	-1	-1	-1	-1	1	1	A	y_2
1	-1	1	-1	-1	1	-1	1	B	y_3
1	1	1	-1	1	-1	-1	-1	ab	y_4
1	-1	-1	1	1	-1	-1	1	C	y_5
1	1	-1	1	-1	1	-1	-1	ac	y_6
1	-1	1	1	-1	-1	1	-1	bc	y_7
1	1	1	1	1	1	1	1	abc	y_8

The function of the response is sought in the form:

$$\eta = \beta_0 + \sum_{1 \le i \le 3} \beta_i x_i + \sum_{1 \le i < j \le 3} \beta_{ij} x_i x_j + \beta_{123} x_1 x_2 x_3 \qquad \text{(I.5.17)}$$

The coefficients of (I.5.17) are obtained by the least squares method.

In a complete factor experiment 2^k the number of experiments $N = 2^k$. As the number of variables k increases, the number of experiments N is increasing rapidly. Therefore, when large-scale k sales of CFE 2^k becomes practically impossible. For PFE 2^k experiments the dependence, similar to (I.5.17), has the form:

$$\eta = \beta_0 + \sum_{1 \le i \le k} \beta_i x_i + \sum_{1 \le i < j \le k} \beta_{ij} x_i x_j + \ldots + \beta_{12\ldots k} x_1 x_2 \ldots x_k \qquad \text{(I.5.18)}$$

With growth N there is an increase in the number of interactions and their order in (I.5.18). But often in the equation (I.5.18), the effects of high-order interaction can be neglected, or it is known a priori that some of them are absent. The number of experiments to find estimates of unknown

coefficients of such an equation can be significantly reduced. This is achieved by applying fractional factor experiments. If in CFE 2^k observations are carried out at all vertices k - dimensional hypercube, then using fractional monitoring plans are carried out in some of them.

Consider an example of constructing a fractional replica. Let the feedback function look like:

$$\eta = \beta_0 + \sum_{1 \le i \le 3} \beta_i x_i \qquad \text{(I.5.19)}$$

In this expression the effects of pair and triple interactions are absent: $\beta_{12} = \beta_{13} = \beta_{23} = \beta_{123} = 0$

If you use CFE 2^3 to estimate unknown coefficients, then $N =$

However, the number of experiments can be reduced because in (I.5.19) the interaction effects are absent. To this end, we will build a plan whose matrix has the form:

$$D = \begin{array}{c} \begin{matrix} x_1 & x_2 & x_3 \end{matrix} \\ \begin{pmatrix} -1 & -1 & 1 \\ 1 & -1 & -1 \\ -1 & 1 & -1 \\ 1 & 1 & 1 \end{pmatrix} \end{array} \qquad \text{(I.5.20)}$$

Matrix of CFE 2^2

The matrix D is derived from the matrix CFE 2^3 by removing next lines from it: (1; -1; 1), (-1; 1; 1), (-1; -1; -1), (1; 1; -1). Constructed fractional factor experiment (FFE) (5.20) is a semi-recipe of CFE 2^3. For its record, the notation is used: 2^{3-1}, where 2 is the number of levels; 3 - the number of variables; $N = 2^{3-1}$ - number of experiments. The code mark of the semicircular: c; a; в; авс.

Consider the features of building a plan. As can be seen from (I.5.19), the variable x_3 at the points of the plan satisfies the ratio:

$$x_3 = x_1 x_2 \qquad \text{(I.5.21)}$$

(I.5.21) – this is the so-called generating ratio. It is easy to construct after it (I.5.20) - initially CFE 2^2, and then - vector-column x_3 corresponding to (I.5.21).
The coefficients of the model in the FFE are also based on the least squares method.

Planning an experiment on the "composition-property" diagrams

Let's consider the planning of an experiment for systems that represent mixtures of q different components. Variables $x_i \quad (i = 1,2,\ldots,q)$ of such systems are proportions (relative to the content) of the components of the mixture and satisfy the condition:

$$\sum_{1 \le i \le q} x_i = 1 \quad (x_i \ge 0) \qquad \text{(I.5.22)}$$

The geometric place of the points satisfying condition (I.5.22) is (q-1) - a measurable regular simplex (triangle for q = 3, tetrahedron for q = 4, etc.). Each
point of such a simplex corresponds to a mixture of a certain composition and vice versa, any point of a simplex corresponds to any combination of relative contents of q components. To a special simplex system (fig. I.5.6) in which the relative contents of each component are deposited along the corresponding faces of the simplex. On the vertices of a simplex each x_i =1, then - determined by the lines (or surfaces) of a level, which are parallel to the opposite side (or faces) of a

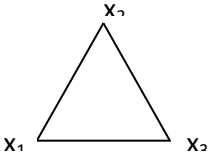


Fig. I.5.6. Simplex coordinate system

simplex. For example, for a three-component mixture, we have a simplex in the form of a triangle $x_1x_2x_3$ on the plane (Fig. I.5.6). The value of x_1 at the vertex x_1 is equal to one, and on the x_2x_3 side it is zero.

The problem of constructing a mathematical model "composition-property" can be solved by writing the desired function in the form of a polynomial of degree *n* in the canonical form:

$$\hat{y} = \sum_{1\le i\le q}\beta_i x_i + \sum_{m=2}^{n}\left\{\sum_{1\le i\le j\le q}\beta_{ij}^{(m)} x_i x_j \left(x_i - x_j\right)^{m-2}\right\} + \sum_{m=3}^{n}\left\{\sum_{1\le i_1\le i_2\le \ldots\le i_m\le q}\beta_s x_{i_1}^{s_1} x_{i_2}^{s_2}\ldots x_{i_m}^{s_m}\right\} \quad \text{(I.5.23)}$$

where $s_1 + s_2 + \ldots + s_m = n$.

A polynomial of this kind (so-called reducedpolynomials) is obtained from ordinary polynomials of the corresponding degree taking into account the relation: $\sum_{1\le i\le q} x_i = 1$ and contain C_{q+n-1}^{n} coefficients.

For example, a second degree polynomial, which in general has the form:

$$\hat{y} = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + b_{12}x_1x_2 + b_{13}x_1x_3 + b_{23}x_2x_3 + b_{11}x_1^{\,2} + b_{22}x_1^{\,2} + b_{33}x_1^{\,2}$$

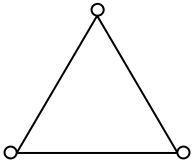
taking into account the correlation $x_1 + x_2 + x_3 = 1$ will take the form:

$$\hat{y} = \beta_1x_1 + \beta_2x_2 + \beta_3x_3 + \beta_{12}x_1x_2 + \beta_{13}x_1x_3 + \beta_{23}x_2x_3.$$

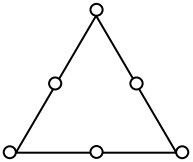
To estimate the coefficients of the reduced polynomial (I.5.23), plans were proposed that ensure a uniform spread of experimental points in (q-1) - dimensional simplex. The points of such plans are knots {q, n} -of simplex grid. In {q, n} - grid for each factor (component) uses (n + 1) evenly spaced levels in the range from 0 to 1 $\left(x_i = 0, 1/n, 2/n, \ldots, 1\right)$ and take a variety of their combinations. Thus, the number of such combinations C_{q+n-1}^{n} is equal to the number of coefficients in the given

polynomial (I.5.23). A set of points $(x_{1u}, x_{2u},..., x_{qu})$, $u = 1,2,..., N = C^n_{q+n-1}$, where $x_{iu} = 0, 1/n, 2/n,...,1$, $\sum_{1 \le i \le q} x_{iu} = 1$ forms a saturated simplex grid plan {q, n}.

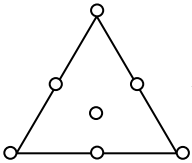
Examples of {q, n} grids:



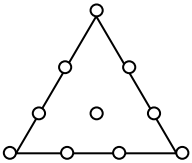
- linear



- quadratic



imcomplete cubic



- cubic

Planning with a pre-transformation of simplex area

When solving q - component mixed problems, it is often necessary to investigate only (q-1) -dimensional simplex suboblusion of a complete (q-1) -dimensional domain. The sub region can be specified by the restrictions on the region of change of all components, for example $x_i \ge q_i$ (i = 1, 2, ..., q). In this case, the direct application of the above methods is impossible because the condition $0 \le x_i \le 1$ is violated. Therefore, pre-transformation of the sub-region is made by moving to a new coordinate system $(z_1, z_2,..., z_q)$

For the sub-region we have:

$0 \le z_i \le 1$, i=1, 2, …, q; $z_1^{(u)} + z_2^{(u)} + ... + z_q^{(u)} = 1$, (I.5.24)

where u – any point of sub-region.

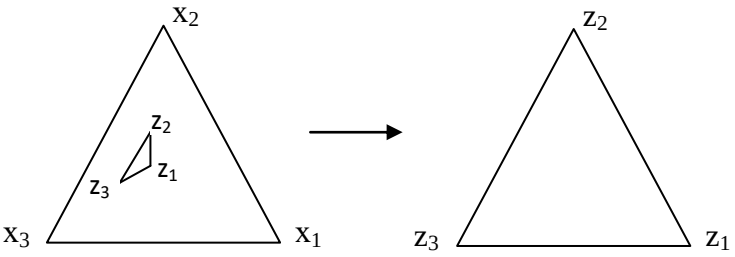


Fig.5.7.Transformation of the sub-region

Transforming relationship between coordinate systems $(x_1, x_2, ..., x_q)$ and $(z_1, z_2, ..., z_q)$ and providing (I.5.24) is given by the matrix equation X = AZ, in expanded form

$$\begin{vmatrix} x_1^{(u)} \\ x_2^{(u)} \\ \dots \\ x_q^{(u)} \end{vmatrix} = \begin{vmatrix} x_1^{(1)} & x_1^{(2)} & \dots & x_1^{(q)} \\ x_2^{(1)} & x_2^{(2)} & \dots & x_2^{(q)} \\ \dots & \dots & \dots & \dots \\ x_q^{(1)} & x_q^{(2)} & & x_q^{(q)} \end{vmatrix} \cdot \begin{vmatrix} z_1^{(u)} \\ z_2^{(u)} \\ \dots \\ z_q^{(u)} \end{vmatrix} \qquad \text{(I.5.25)}$$

Here the elements of the matrix A are the coordinates of the vertices of the simplex; $x_i^{(u)}$ i $z_i^{(u)}$ - input and new coordinates of the u-th transformed point.

For all new *z* variables, all plans that were used for a complete simplex can be constructed. However, realization of the experiment in such conditional plans is impossible. For the experiment, it is necessary to represent the experimental compositions in the *x*-coordinates (transition for (5.24)).

I.6. THEORETICALAL MATHEMATICAL MODELS OF DESIGN OBJECTS

Basic concepts of the theory of differential equations

Differential are called equations containing derivatives of an unknown function of one or more independent variables.

Equations containing derivatives of several independent variables are called partial derivatives.

Equations containing derivatives of only one of the independent variables are called ordinary differential equations.

The general form of the differential equation of the *n*-th order is as follows:

$$F\left(x, y, y', y'', ..., y^{(n)}\right) = 0. \qquad \text{(I.6.1)}$$

This is an implicit form of differential equation. An explicit form of the equation of the nth order is an equation that is solved with respect to the older derivative:

$$y^{(n)} = f\left(x, y, y', y'', ..., y^{(n-1)}\right). \qquad \text{(I.6.2)}$$

Let the variable x take values in the interval $I \subset R = (-\infty, \infty)$. The solution of the differential equation on the interval I is the differentiated function $y = \varphi(x)$ in I, after substituting it into an equation, it turns into equality for all $x \in I$ (the identity on the set I). The graph of the solution of the differential equation is called the integral curve.

The general solution of the equation usually contains one free numerical parameter and has the form

$$y = \varphi(x, C) \qquad \text{(I.6.3)}$$

where C is the mentioned parameter, φ- *i*s a certain function. Equation (I.6.3) determines the family of functions that depend on the parameter C. The isolation of a single solution of the family of solutions (I.6.3) can be fulfilled if the known initial value $y(x_0) = y_0$ for some $x_0 \in I$.

The general solution of equation (I.6.1) or (I.6.2) is the family of functions of the form

$$y = \varphi(x, C_1, \ldots, C_n), \qquad \text{(I.6.4)}$$

where $C_1, \ldots, C_n$ are numerical parameters, which are also called arbitrary constants, and each function of this family is the solution of this equation (at one or another numerical interval).

The parameters $C_1, \ldots, C_n$ can be determined by the initial conditions of the form $y(x_0) = y_{10}, \ldots, y^{(n-1)}(x_0) = y_{n0}$.

There are situations when solutions of differential equations in explicit form (6.3), (6.4) can not be obtained, but

we can find so-called general integrals, otherwise general solutions of these equations. In this case, the general integral of the differential equation (I.6.1) ((I.6.2)) is called so different from the identity of the equation

$$\psi\left(x, y, C_1, C_2, ..., C_n\right) = 0, \qquad \text{(I.6.3a)}$$

that the solutions (I.6.1) ((I.6.2)) are differentiated functions $y = \varphi(x)$, which are obtained as solutions to equation (I.6.3a) with values of stable $C_1, \ldots C_n$ from certain certain domains. The function ψ is also called the general integral of the equation.

Methods of solving differential equations

The problem of solving the ordinary differential equation in the general case is much more complicated than the problem of calculating single-time integrals, and therefore the fate of cases of explicit integration is much lower here.

Numerical methods for solving differential equations can be divided into two classes. One of them includes methods that use one starting value of the solution at each step, and the second forms methods that use several values at each step (multi-step methods). The latter are characterized by the fact that based on the previously obtained several values of the function, new ones are constructed, which then are refined with the help of the differential equations themselves.

The first class includes the methods of Runge-Kutti, in particular, the methods of Euler-Cauchy and trapezius. The second is, for example, the Adams method, the Krylov-Adams method.

Let's consider first the Euler-Cauchy method.

Let a differential equation is given

$$\frac{dy}{dx} = f(x, y), \qquad \text{(I.6.4)}$$

where (x, y) belongs to the domain G with the initial condition

$$x = x_0, y_0 = y(x_0) \qquad \text{(I.6.4′)}$$

The method of constructing an approximate solution of the Cauchy problem (I.6.4), (I.6.4′) Is based on the concept of so-called broken Euler. Laman Euler is a graph of a piecewise linear function, which is constructed by the following rule. Let *h* be a small positive number (step method). Let's consider in the Cartesian plane a point with coordinates (x_1, y_1), where

$$x_1 = x_0 + h,\ y_1 = y_0 + hf(x_0, y_0).$$

Note that according to the Taylor formula, due to the equations (I.6.4), (I.6.4′), The value of $y_1$can be considered as an approximation to the value of the solution $y(x_1)$ of the Cauchy problem under consideration. If the point (x_1, y_1) belongs to the set *G*, then we continue the construction by the inductive rule

$$y_{i+1} = y_i + hf(x_i, y_i),\ i = 0, 1, 2, \ldots .$$

Each value y is considered as an approximation to the value of the desired solution *y* at x_i. Thus we obtain a sequence of points(x_i, y_i) , $i = 0,1,2,\ldots$, where all x_i are located to the right of the point x_0. Similar construction, if necessary, is performed to the left of point x_0. On the received sequence we construct a piecewise linear function

$$y(x) = y_i + f(x_i, y_i)(x - x_i),\ x \in [x_i, x_{i+1}],\ i = 0, 1, 2, \ldots ,$$

which (or whose schedule) is called broken Euler. There are several theorems that guarantee that, under certain conditions, the Euler Layer directs to the solution of the Cauchy problem (3.10), (3.10′), When the step of the method *h* goes to 0.

Let a differential equation be given

$$\frac{dy}{dx} = f(x, y). \qquad \text{(I.6.4)}$$

It is necessary to find an approximate solution (I.6.4) at points with coordinates $x_1 = x_O + h,\ x_2 = x_O + 2h, \ldots x_n = x_O + nh$, where h - is a constant step; x_O - the coordinate of the beginning of the segment.

Initial condition: $x = x_O$, $y_O = y(x_O)$. The approximate value of the first derivative has the form

$$\frac{dy_k}{dx_k} \cong \frac{\Delta y_k}{\Delta x_k} = \frac{y_{k+1} - y_k}{h}, \qquad \text{(I.6.5)}$$

where $k = 0,1...n-1$.

Equating (I.6.4) and (I.6.5) we obtain:

$$\frac{y_{k+1} - y_k}{h} = f(x_k, y_k),$$

where:

$$y_{k+1} = y_k + hf(x_k, y_k). \qquad \text{(I.6.6)}$$

Using the recurrence formula (I.6.6) for the points $k = 0,1...n-1$, we construct the Euler lamina 2, which approximately replaces the integral curve 1 (see fig. I.6.1). The essence of the Euler-Cauchy method lies in the fact that, due to the beginning of each segment $[x_k, x_{k+1}]$, the tangent to the integral curve 1 (fig. I.6.1) Is carried out.

The accuracy of the Euler-Cauchy method is small. Method error is proportional to h^2.

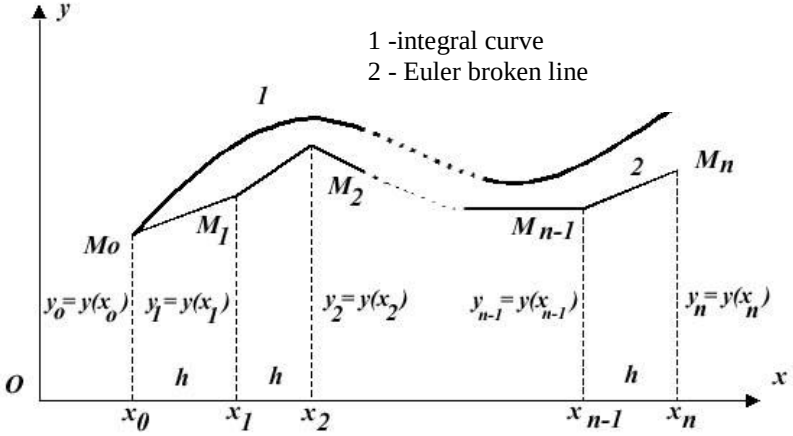


Fig. I.6.1. The calculation scheme of the Euler-Cauchy method

A variant of the Euler-Cauchy method is a trapezoid method. It is implemented by applying at each step a recurent formula

$$y_{k+1} = y_k + \frac{h}{2}\left\{ f(x_k, y_k) + f\left[x_k + h \ , y_k + hf(x_k, y_k) \right] \right\}. \qquad \text{(I.6.7)}$$

The error of the trapezium method is proportional to h^3 and is also attributed to the general methods of Runge-Kutta.

Multi-steps solving of differential equations (finite difference methods) are based on the resolution of these previous steps. This allows you to increase the computational speed. To realize the finite-difference methods for the numerical integration of differential equations need to know the function and its derivatives at several points close to the original. Here you can note Picard's method and the method of decomposition

Systems of differential equations

Set of relationships

$$\begin{cases} F_1(x, y_1, ..., y_1, y_1', ..., y_n') = 0 \\ F_2(x, y_1, ..., y_1, y_1', ..., y_n') = 0 \\ \quad ..... \\ F_n(x, y_1, ..., y_1, y_1', ..., y_n') = 0 \end{cases} \qquad \text{(I.6.8)}$$

where x - is an independent variable,

$y_1, y_2, ..., y_n$ - unknown functions depending of x,

$F_1, F_2, ..., F_n$ - are known functions,

is called the system of differential equations of the first order.

The solution of this system are functions $y_1(x), y_2(x), ..., y_n(x)$ that, when substituting in (I.6.8), transform the system into identity.

If the system of differential equations (I.6.8) allows for the solution of the derivatives, then we obtain the system

$$\begin{cases} \dfrac{dy_1}{dx} = f_1(x, y_1, y_2, ..., y_n) \\ \dfrac{dy_2}{dx} = f_2(x, y_1, y_2, ..., y_n) \\ \cdots\cdots\cdots\cdots\cdots\cdots \\ \dfrac{dy_n}{dx} = f_n(x, y_1, y_2, ..., y_n) \end{cases} \qquad \text{(I.6.9)}$$

which is called normal.

An example of one normal first-order equation is

$\dfrac{dy}{dx} = f(x, y)$.

This equation gives a field of directions in a plane x, y. The solution of the equation is one parametric family of curves located in one plane. If in this plane a point (x_0, y_0) and $f(x, y)$, $\dfrac{\partial f}{\partial y}$ - are continuous, then the equation has a unique solution that satisfies the initial conditions $y(x_0) = y_0$.

Let's now take two equations

$$\begin{cases} \dfrac{dy_1}{dx} = f_1(x, y_1, y_2) \\ \dfrac{dy_2}{dx} = f_2(x, y_1, y_2) \end{cases} \quad \text{or} \quad \begin{cases} \dfrac{dy}{dx} = f_1(x, y, z) \\ \dfrac{dz}{dx} = f_2(x, y, z) \end{cases}.$$

Under certain conditions we get a solution

$y_1 = y = \varphi_1(x); \quad y_2 = z = \varphi_2(x)$.

These solutions can be considered as parametric equations of the spatial curve in the coordinate system x, y, z.

Thus, the solution of one equation can be represented by a curve of two-dimensional space. The solution of two first-order equations can be represented by a curve in a three-dimensional space. The solution of n equations of the first order forms a curve in $(n+1)$-dimensional space. These curves are called integral.

Numerical solution of systems of differential equations is carried out in the same way as solving a single differential equation

I.7. INTEGRATED AUTOMATED DESIGN SYSTEMS. DESIGN AND PRODUCT LIFE CYCLE MANAGEMENT SYSTEMS

Introductory provisions

To understand the meaning of the CAD / CAM / CAE systems (all these systems are collectively referred to as automated design systems), it is necessary to examine the various tasks and operations that have to be solved and performed in the process of product development and production. All these tasks, taken together, are called product cycle (product cycle).

The development process begins with customer inquiries served by the marketing department and ends with a complete description of the product (executed in the form of a drawing). The production process begins with technical requirements and ends with the delivery of finished products.

The production process begins with the planning, which is executed on the basis of the drawings received at the stage of designing and ends with the finished product.

As a result of production preparation, a production plan, inventories of materials and software for the equipment are made. The last phase of the development process is the preparation of the project documentation. At this stage, the use of systems for preparing work drawings becomes useful. The ability of such systems to work with files allows you to systematize the storage and ensure the convenience of finding documents.

Computer technologies are also used at the production stage. The production process includes planning the release, designing and acquiring new tools, ordering materials,

programming of machines with numerical control (CNC), quality control and packaging.

Basic concepts of CAD/CAM/CAE systems

Computer-aided design (**CAD**) – is a technology that involves the use of computer systems to facilitate the creation, modification, analysis, and optimization of projects. Thus, any program that works with computer graphics, as well as any application used in engineering calculations, relates to automated design systems. In other words, most CAD tools can range from geometric forms for working with forms to specialized applications for analysis and optimization.

Computer-aided manufacturing (**CAM**) - is a technology that involves the use of computer systems for the planning, management, and control of production operations through a direct or indirect interface with the enterprise's productive resources. One of the most common approaches to automation of production is numerical control (NC, numerical control - NC).

CNC is to use programmed commands to control the machine, which can be grinding, cutting, milling, punching, bending and other ways to turn the workpieces into finished parts.

Another important function of automated production systems is the programming of robots that can work on flexible automated areas by choosing and installing tools and parts that are machined on CNC machines. The works can also perform their own tasks, for example, to weld, assemble and transport the equipment and parts of the shop.

Computer-aided engineering (**CAE**) - is a technology that uses computer systems to analyze CAD geometry, modeling and studying product behavior to improve and optimize its design. CAE tools can carry out many different analysis options.

Hardware and software CAD

To implement a computer-oriented approach to design and production requires special hardware and software.

The key aspect is interactive form control, so hardware and software for interactive manipulation of forms are one of the major components of the CAD / CAM / CAE system. Graphics devices and I / O peripherals together with the usual computing module make CAD / CAM / CAE system hardware.

CALS- technologies. Substantive provisions

Modern conditions are characterized by increasingly tight competition in the international market, increasing complexity and knowledge-intensive production, which puts new problems for industrialists and businessmen in the country. Among them are:

- the critical time needed to create a product and organize its sale;
- reduction of all types of costs associated with the creation and maintenance of the product;
- improving the quality of design and production processes;
- providing flexible and reliable maintenance services.

An effective means of solving these problems in the last decade are the new information CALS-technologies of cross-cutting support of complex science-intensive products at all stages of its life cycle (LC) from marketing to recycling. They are based on a standardized single electronic data view and collective access to them, these technologies make it possible to substantially simplify the implementation of the stages of the LC product and increase productivity, according to western experience, by about 30%, to automatically provide a given product quality.

For the first time, elements of CALS-technologies began to be used in the mid-80 with the interaction of the US Department of Defense with its suppliers, when asked to

translate all operations with them in electronic form. Subsequently, the scope of CALS-technologies expanded to the entire life cycle of the product and went beyond the military departments. Nevertheless, the most advanced users of CALS technology are still military developers.

In the field of the civilian introduction of CALS-technologies in the world, the leading aerospace, and nuclear industry, automobile, and shipbuilding. In Europe, CALS has also been widespread. The European Industrial Group in the field of CALS has been created, national CALS programs are created and created, as well as individual CALS projects.

Lack of introduction of CALS-technologies will make it impossible for enterprises to participate in international cooperation, will negatively affect the competitiveness and attractiveness of manufactured products, will cause loss of certain segments of the market.

At the moment, **CALS** is understood as a **global strategy for increasing the efficiency of business processes performed during the life cycle of a product** due to information integration and continuity of information generated at all stages of the lifecycle.

The means of realization of this strategy are CALS-technologies, which is based on a set of integrated information models: the life cycle itself and carried out in its course business processes, product, and production and operating environment.

The possibility of sharing information is ensured by the use of computer networks and the standardization of data formats, which provides a correct interpretation of information.

CALS (Continuous Acquisition and Life Cycle Support) - is a United States Department of Defense initiative for electronically capturing military documentation and linking related information**.** This is a strategy for increasing the efficiency, productivity, and profitability of the processes of

economic activity of enterprises due to the introduction of modern methods of information interaction of participants of the LC product.

The life cycle of a product is a set of processes performed from the moment of identifying the needs of society in certain products until the time these needs are met and the product disposed of.

The LC product is characterized by a large variety of processes. The most famous are production process, design process, procurement process. Each of these processes, in turn, consists of technological processes and organizational and business processes. For the general description of these processes, the term "business process" is used.

Business process - a set of technological and organizational and business processes, carried out purposefully within the framework of a predefined organizational structure.

Consider the definition of CALS in more detail. In the literal translation, the abbreviation CALS means "continuity of supply of products and maintenance of its life cycle". The first part of the definition - "continuity of supply of products" requires and provides optimization of the processes of interaction between the customer and the supplier in the development, design, and production of complex products, whose lifespan, taking into account various modernizations, is made for decades. To ensure efficiency, as well as reduce costs and time, the customer-supplier interaction process must be truly continuous. The second part of the definition of CALS - "lifecycle support" - is to optimize the maintenance, repair, spare parts supply and upgrades. Since the costs of maintaining a complex hi-tech product in an able-bodied state often equal or exceed the cost of its acquisition, a fundamental reduction in the "cost of ownership" is provided by investments in the establishment of a system of support for JCs.

The purpose of using CALS-technologies, as an instrument of organization and information support of all participants in the creation, production and use of the product, is to increase the efficiency of their activities by accelerating the research and development of products, adding products new properties, reducing costs in the processes of production and operation of products, increase the level of service in the processes of its operation and maintenance.

The subject of CALS is the technology of information integration, that is, the sharing and sharing of information about the product, environment, and processes that occur during the product lifecycle. The basis of CALS is the use of a complex of unified information models, standardization of ways to access information and its correct interpretation, information security, legal issues of sharing information (including intellectual property), use at various stages of JCs of automated software systems (CAD / CAM / CAE , ERP, etc.), allowing to produce and share information in the format of CALS.

Tasks that are solved with help of the CALS-technologies

Modeling the product lifecycle and executable business processes. This is the first and very significant step towards improving the effectiveness of an organizational structure that supports one or more stages of the product's JC, i.e. modeling and analyzing its operation.

The purpose of the business analysis is to identify existing interactions between component parts and evaluate its rationality and efficiency. To do this, using functional CALS-technologies, **functional models** are developed that contain a detailed description of the processes performed in their interconnection. The resulting functional model not only provides a detailed description of executable processes but also allows solving a number of tasks related to optimization,

estimation, and allocation of costs, estimation of functional productivity, loading and balancing of components, that is, questions of analysis and re-engineering of business processes. Functional modeling techniques, for example, can be successfully used in creating systems for product quality assurance.

Design and production of the product. Joint, co-operative design and product manufacturing can be effective if it is based on a single product information model (electronic product model). Once created, the product model is used repeatedly. It is supplemented and modified; it serves as a starting point for product upgrades. A product model in accordance with this standard includes geometric data, product configuration information, changes, approvals, and approvals. The standard way of presenting design and technological data allows you to solve the problem of information exchange between different divisions of the enterprise, as well as participants of the cooperation, equipped with heterogeneous design systems. The use of international standards provides a correct interpretation of stored information, the ability to quickly transfer functions of one contractor to another, which, in turn, can take advantage of the results of work already carried out.

Operation of the product. It is known that the volume of developer documentation for a complex science-intensive product is very large. Therefore, traditional paper documentation of complex products requires huge costs for supporting archives, adjusting documentation, and also reduces the operational attractiveness and competitiveness of the product. The solution to the problem is to translate the operational documentation for the product that comes to the consumer, in electronic form. Electronic documentation can be delivered on electronic media (for example, on CDs) or placed on the Internet. Standardization guarantees the applicability of

such electronic documentation on any computer platform. It is important to note that in the electronic form the operational documentation that was created earlier without the use of computer systems can be converted. For products that are already in operation for a long period and designed by traditional methods, the task of supporting documentation is equally relevant. As an example, you can cite the experience of projects in the Navy and US Air Force to massively transfer millions of manual pages and drawings to a standardized electronic view. The received electronic documentation is placed in special repositories on the Navy and Air Force bases or directly from producers and is accessible through computer networks. It uses modern scanning technologies, text recognition, vectorization of drawings and circuits, creating electronic directories for entire products and individual systems.

What CALS-technology does

CALS is considered as a comprehensive system strategy for improving the efficiency of all processes of industrial products, directly affects its competitiveness. The application of the CALS strategy is a condition for the survival of enterprises in a context of growing competition and allows:

- to expand the scope of enterprises (sales markets) through cooperation with other enterprises, provided by standardization of the presentation of information at different stages and stages of the life cycle;

- at the expense of information integration and reduction of expenses for paper document circulation, re-input, and processing of information ensure the continuity of the results of work in integrated projects and the possibility of changing the composition of participants without losing the results already achieved;

- to increase the "transparency" and manageability of business processes through their re-engineering, based on

integrated models of JCs and running business processes, reduce costs in business processes at the expense of better balance of links;

- to increase the attractiveness and competitiveness of products designed and manufactured in an integrated environment using modern computer technologies and have the means of informational support during the exploitation phase;

- to provide the given quality of production in the integrated system of support of the LC by electronic documentation of all processes and procedures;

- reduce production costs and reduce the cost of production;

- reduce the time of product creation, its modernization and increase its real lifetime, functioning in a workable state at the expense of high quality and electronic support during operation.

Life cycle of the software

The life cycle of software – is a period of time that begins with the decision on the need to create a software product and ends when it is completely decommissioned. This cycle – is the process of construction and development of software. The standard provides for the following stages and stages of the creation of an automated system (AS):

- formation of requirements to the AS;
- development of the concept of the AS;
- study of the object;
- conducting the necessary research work;
- technical task;
- sketch project;
- technical project;
- working documentation;
- development of working documentation on the AS and its parts;
- development and adaptation of programs;

- putting into operation;
- preparation of the object of automation;
- conducting preliminary tests;
- conducting experimental exploitation;
- support of the AS;
- performance of work in accordance with warranty obligations
- post-warranty service.

The model of the life cycle of the software is a structure that determines the sequence of execution and the interconnection of processes, actions, and tasks throughout the life cycle. The life cycle model depends on the specifics, scale, and complexity of the project and the specific conditions in which the system is created and functioning. The standard does not offer a specific life cycle model. Its provisions are common to any life-cycle models, methods, and technologies for creating an IP. It describes the structure of the processes of the life cycle, without specifying how to implement or execute the actions and tasks included in these processes.

The model of software includes stages; performance results at each stage; key events - points of completion and decision making.

Stage - part of the process of creating software, limited by certain time frames and ends with the release of a specific product (models, software components, documentation), determined by the requirements specified for this stage.

Software life cycle models

Waterfall (cascade, sequential) model

The waterfall model of the life cycle was proposed in 1970 by Winston Royce. It involves the consistent implementation of all stages of the project in a strictly fixed order. Moving to the next step means complete completion of the work in the previous step. Requirements defined at the stage of forming requirements are strictly documented in the

form of a technical specification and fixed for the entire time of project development. Each stage ends with the release of a complete set of documentation that is sufficient to allow the development to be continued by another team of developers.

Stages of the project according to the cascade model:

- formation of requirements;
- designing;
- realization;
- testing;
- implementation;
- operation and maintenance.

Advantages of cascade model:

- complete and consistent documentation at each stage;
- It is easy to determine the terms and costs of the project.

Disadvantages: in the fall-off model, the transition from one phase of the project to another implies the complete correctness of the result (output) of the previous phase. However, the inaccuracy of a requirement or the incorrect interpretation of it results in the fact that it is necessary to "roll back" to the early phase of the project and the required processing does not simply knock out the design team from the graph, but often leads to a qualitative increase in costs and, it is possible, to terminate the project in the form in which he initially thought. A foolproof model for large projects is realistic and can only be effectively used to create small systems.

Iterative model

An alternative to the successive model is the so-called model of iterative and incremental development, which also received the name of the evolutionary model from T. Gilba in the 70's. Also, this model is called an iterative model.

The model involves dividing the life cycle of the project into a sequence of iterations, each of which resembles a "mini-

project", including all development processes applied to the creation of smaller fragments of functionality, as compared to the project as a whole. The purpose of each iteration is to obtain a working version of the software system, which includes the functionality defined by the integrated content of all previous and current iterations. The result of the final iteration contains all the necessary functionality of the product.

The approach has its own negative sides, which, in essence, are the reverse side of merit. Firstly, there is no comprehensive understanding of the possibilities and limitations of the project for a long time. Second, iterations have to reject part of the work done before. Thirdly, the integrity of specialists in the performance of work is still reduced, which is psychologically understandable, because over them constantly hangs the feeling that "everything can still be reworked and improved later".

Version Control System

Version Control System – is software for facilitating work with changing information. The version control system allows you to save several versions of the same document, if necessary, to go back to earlier versions, to determine who and when they made one or another change, and much more. Such systems are most widely used in developing software for storing source code of the program, which is being developed. However, they can be successfully applied in other areas where work is underway with a large number of continuously changing electronic documents. In particular, version control systems are used in CAD, usually in the PDM system.

General information. The situation in which an electronic document undergoes a number of changes during its existence is quite typical. It is often important to have not only the latest version but several previous ones. In the simplest case, you can simply save multiple document variants by numbering them accordingly. This method is ineffective (it is

necessary to store several almost identical copies), requires increased attention and discipline and often leads to errors; therefore, tools were developed to automate this work.

Traditional version management systems use a centralized model when there is a single document repository driven by a special server that performs most of the functions of version management. A document user must first obtain the version of the document from the repository he needs; usually, a local copy of the document is created, i.e. "Working copy". The latest version or any of the previous ones may be obtained, which may be selected by version number or date of creation, and sometimes by other features. Once the necessary changes are made to the document, the new version is placed in the repository. Unlike a simple file save, the previous version is not erased, but also remains in the repository and can be obtained from there at any time. The server can use the so-called. Delta compression is a way of storing documents that store only changes between successive versions, which reduces the amount of stored data. Since the most recent version of the file is most in demand, the system can save it completely while saving the new version, replacing the last previously saved version in the repository with the difference between this and the latest version. Some systems support the preservation of versions of both types: most versions are stored in the form of deltas, but periodically (by a special administrative command) all versions of all files are kept in full; Such an approach provides the maximum full recovery of history in the event of damage to the repository. It often happens that several people work simultaneously on one project. If two people change the same file, one of them may accidentally cancel the changes made by others. Version control systems track such conflicts and offer solutions to them. Most systems can automatically merge (merge) changes made by different developers. However, such an automatic combining of changes, as a rule, is

possible only for text files and provided different (non-overlapping) parts of this file have been changed. This limitation is due to the fact that most version control systems are oriented to support the software development process and the source code of the programs is stored in text files. If the automatic merger fails, the system may suggest solving the problem manually. Often it is impossible to perform a merger either in automatic mode or in manual mode, for example, if the file format is unknown or too complicated. Some version control systems allow you to lock the file in the repository. The lock prevents others from obtaining a working copy or preventing a change in the working copy of the file (for example, by means of the file system) and thus provides exclusive access to only the user who is working with the document. Many version control systems provide a number of other features: Allows you to create different versions of a single document, i.e. branches, with the general history of changes to the point of branching and with different - after it. Allows you to know who and when added or modified a particular set of lines in a file. Conducts a log of changes in which users can write an explanation of what and why they changed in this version. Controls user access rights by resolving or prohibiting reading or modifying data, depending on who is asking for this action.

Typical working order with the system

Each version management system has its own specific features in the set of commands, user order, and administration. Nevertheless, the general working order for most VCS is absolutely stereotypes. It is assumed that the project, whatever it may be, already exists and the server hosts its repository, to which the developer has access.

The first action required by the developer is to extract a working copy of the project or the part that it will have to work with. This action is performed using the standard version

extractor command. The developer sets the version to be copied, by default, the last one (or chosen by the administrator as the main) version is usually copied. The removal command establishes a connection to the server and the project (or part of it - one of the directories with subdirectories) in the form of a tree of directories and files are copied to the developer's computer.

Modification. The developer modifies the project by modifying the files in it in a working copy in accordance with the design task. This work is carried out locally and does not require calls to the VCS server.

Changes fixation. Having finished the next stage of work on the task, the developer fixes his changes, passing them to the server.

Versions merge. Changes within a single text file made in different versions can be merged if they are located in different places of this file and do not overlap. In this case, all the changes made are made into the combined version. Changes within one file, if it is not text, are always conflicting and cannot be combined automatically. The overwhelming majority of modern version control systems are focused, first of all, on software development projects, in which the main kind of file content is text. Accordingly, mechanisms for automatic merging of changes are guided by the processing of text files, that is, files that contain text consisting of strings of alphanumeric characters, spaces, and tabs separated by lines of the line's characters. When determining the admissibility of a merger of changes within the same text file, the typical mechanism of line comparison works.

Conflicts and their solution. The situation when the merging of several versions of the changes made in them intersects each other is called a conflict. In case of conflict, the version control system cannot automatically create a merged project and is forced to contact the developer.

As already mentioned above, conflicts may occur at the stages of fixing changes, updating or merging branches. In all cases, when a conflict is detected, the operation is terminated until its permission. In order to solve the conflict, the system, in general, offers the developer three variants of conflicting files: basic, local and server. Conflicting changes or displayed to the developer in a special program modulus of the combination of changes (in this case there are shown merging options and dynamically changes depending on the user's team combined file option), or simply marked with a special markup directly in the text of the merged file (then the developer must formulate the desired text in controversial places and save it).

Locking. The lock mechanism allows one developer to seize a monopoly file or group of files to make changes to them. As long as the file is locked, it remains accessible to all other readers only, and any attempt to make changes to it is rejected by the server. In some cases, the use of blocking is entirely justified. An obvious example is the organization of work with binary files, for which there are no tools for merging changes, or such a merger is fundamentally impossible (as, for example, for image files). If automatic merging is not possible, then in the normal operation of any parallel change of such files will lead to conflict. In this case, it is much more convenient to make such a file blocked to ensure that any changes to it will be introduced only consistently.

I.8. CASE – TECHNOLOGIES OF COMPUTER DESIGN

The concept of the CASE - technology. Implementation methods

Trends in the development of modern information technologies lead to a continuous increase in the complexity of information systems (ICs) that are created in various areas of

the economy. **Modern large IP projects** are characterized, as a rule, by the following features:

· the complexity of the description (a considerable number of functions, processes, data elements and complex interrelationships between them), which requires careful modeling and analysis of data and processes;

· presence of a set of closely interacting components (subsystems) having their own local tasks and goals of operation (for example, traditional applications related to the processing of transactions and solution of regulatory tasks, and analytical processing applications (decision support) using unregulated data queries large volume);

· the absence of direct analogs, limiting the possibility of using any typical design solutions and application systems;

· the need to integrate existing and newly developed applications;

· functioning in a heterogeneous environment on several hardware platforms;

· disunity and heterogeneity of individual groups of developers according to the level of qualification and the traditions of the use of certain or other tools developed;

· the substantial temporal extent of the project is due, on the one hand, to the limited capabilities of the team of developers, and, on the other hand, the scale of the organization-customer and the degree of readiness of its individual units prior to the introduction of IP.

For the successful implementation of the project, the design object (IC) must be first and foremost adequately described; full and consistent functional and information models of the IC must be built. The accumulated experience of IP designing so far shows that this is a logically complicated, time-consuming and time-consuming work requiring a high qualification of the specialists involved in it. However, until recently, the design of IP was carried out mainly on an intuitive

level with the use of non-formalized methods based on art, practical experience, expert assessments, and expensive experimental verification of the quality of IP operation. In addition, in the process of creating and operating IP, user information needs may change or refine, which further complicates the design and maintenance of such systems.

In the 70's and 80's in the development of IPs widely used methodology, this provides developers with strict formalized methods for describing IPs and technical decisions that are adopted. It is based on visual graphics techniques: diagrams and diagrams are used to describe different types of IP models. The visibility and rigor of the analysis tools allowed developers and future users of the system to informally participate in its creation from the outset, discuss and consolidate understanding of the main technical solutions. However, the widespread use of this methodology and its compliance with its recommendations in the development of specific IPs was quite rare, since it is virtually impossible to develop in non-automated (manual) development. Indeed, it is very difficult to manually design and graphically present the strict formal specifications of the system manually, check them for completeness and consistency, and even more so. If you still manage to create a rigorous system of project documents, then it's processing in the event of serious changes is practically impossible. Manual development usually caused the following problems:

· Inadequate specification of requirements;
· Failure to detect errors in design decisions;
· Low quality of documentation, which reduces operational quality;
· Latency cycle and unsatisfactory testing results.

On the other hand, IP developers historically have always been the last in a number of those who used computer

technology to improve quality, reliability, and productivity in their own work (the phenomenon of "a shoe without a boot").

The listed factors contributed to the emergence of software-technological tools of the special class - CASE-tools, implementing CASE-technology for creating and maintaining IP. The term CASE (Computer Aided Software Engineering) is currently used in a very broad sense. The primary meaning of the CASE term, limited by the automation of software development (software), has now acquired a new meaning, which encompasses the process of developing complex ICs in general. Now under the term, CASE-tools are software tools that support the creation and maintenance of IPs, including analysis and requirements formulation, application software and databases, code generation, testing, documentation, quality assurance, configuration management and project management, as well as other processes. CASE-tools together with system software and hardware form a complete environment for the development of IP.

The emergence of CASE-technologies and CASE-funds was preceded by research in the field of programming methodology. In addition, the emergence of CASE-technologies contributed to such factors as:

· Preparation of analysts and programmers susceptible to the concepts of modular and structural programming;

· Widespread introduction and constant growth of the productivity of computers, which allowed the use of effective graphics tools and automate most of the stages of designing;

· Introduction of network technology, which enabled the joint efforts of individual performers into a single design process by using a database containing the necessary information about the project.

CASE technology is an IP design methodology, as well as a set of tools that allow you to visualize the subject area, analyze this model at all stages of the development and

maintenance of IP and develop an application in accordance with the information needs of users. Most existing CASE-based tools are based on structured (mostly) or object-oriented analysis and design methodologies that use charts or text-based specifications for describing external requirements, relationships between system models, system behavior and software architectures.

According to the Survey of Advanced Technology, compiled by Systems Development Inc. According to the results of the survey of more than 1,000 American firms, CASE technology is currently ranked among the most stable information technologies (half of all respondents used it in more than a third of their projects, 85% of them completed successfully). However, despite all the potential features of CASE-tools, there are many examples of their failure, resulting in CASE-tools becoming "shelfware". In this regard, it is necessary to note the following:

· CASE-tools do not necessarily give immediate effect; it can only be obtained after some time;

· Real costs for the implementation of CASE-funds are usually much greater than the cost of their acquisition;

· CASE-tools provide opportunities for significant benefits only after the successful completion of their implementation.

To successfully implement CASE-tools the organization must have the following qualities:

· Technology. Understanding the limitations of existing capabilities and the ability to adopt new technology;

· Culture. Willingness to implement new processes and relationships between developers and users;

· Management. Clear management and organization in relation to the most important stages and implementation processes.

If the organization does not possess at least one of the listed qualities, the introduction of CASE-funds may fail, regardless of the degree of diligence to comply with various recommendations for implementation.

"Pitfalls" use CASE-tools:

· The introduction of CASE-tools can be a rather lengthy process and may not bring immediate returns. Perhaps even a short-term decline in productivity as a result of efforts spent on implementation. As a result, the management of the user organization may lose interest in CASE-tools and cease support for their implementation;

· The lack of full compliance between those processes and methods supported by CASE-tools and those used in this organization may lead to additional difficulties.;

· Some CASE tools require a lot of effort to justify their use in a small project;

· The negative attitude of personnel towards the introduction of the new CASE-technology may be the main reason for the failure of the project.

Users of CASE-tools should be prepared for the need for long-term operating costs, the frequent emergence of new versions and the possibility of rapid moral aging, as well as constant costs for training and advanced training of staff.

Despite all the reservations expressed and some pessimism, a competent and wise approach to using CASE-tools can overcome all of these difficulties. Successful implementation of CASE-tools should provide such benefits as:

· High level of technological support for software development and support;

· Positive effect on some or all of the listed factors: productivity, quality of products, compliance with standards, documentation;

· Acceptable level of return on investment in CASE-tools.

An example of an object-oriented CASE-tool – is Rational Rose.

Analysis, verification, and optimization of design solutions by means of CAD

The decision to reduce the time for technological preparation of production and release of new products, especially small batches, ensures their competitiveness and enables the prompt response to changes in consumer demand. This, in turn, reduces both the cost of manufacturing new products and the time from the appearance of new design developments to their introduction into industrial designs. To solve this problem it is necessary to determine the set of necessary methods and means of education of design routes - sequences of design operations and procedures leading to the achievement of the goal. At the same time, methods of constructing design sequences are determined by the type of design tasks.

The basis for the implementation of multi-objective technological design are existing working production systems (WPS), focused on the production of their production tasks (PT) and have a free time fund of their technological equipment. Technology equipment with a free time fund is the resources of production systems (PS) necessary for the operation of virtual production systems (VPS). On the basis of the information on the resources of the VPS operatively formed the configuration (possibly changing in time), maximally meets the requirements of the executed PT. The peculiarity of such an approach is the use of elements of intellectual control, which allows you to make decisions about changing the configuration of the VPS and the formation of control information in real time with the minimum participation of the operator-operator.

Multifocal technological design with intelligent control in the WPS includes: techniques for designing technological processes, a method for ensuring the purposeful generation of possible variants of the WPS configuration, the method for verifying generated options and selecting the best, as well as a decision-making technique, on the basis of which the control of the configuration process is carried out WPS in time.

Due to the fact that the decision-making and the formation of its management influence is based on complex creative processes, management must be built as an intellectual. Thus, the conceptual idea of constructing a virtual production system lies in the mobile organization of a temporarily functioning object-oriented PS for the implementation of current technological processes based on the WPS. In other words, in the presence of some PT the strategy of their implementation in the WPS is necessary, have a free time fund and, in turn, oriented to the release of other, different in their parameters of products. In this case, the implementation of designed technological processes should not negatively affect the timing and cost of production of the main for these WPS products.

Implementation of the idea is achieved by the formation of the air force rational configuration, which allows you to carry out the PT in terms not exceeding the predetermined, but close to them, with a minimum cost. This approach ensures that there is no material rebuilding in the formation of the Armed Forces for the implementation of the UA, the minimum storage costs of finished products and the minimum amount of resources used by operational production systems (OPS).

The use of free technological equipment WPS, focused on the implementation of its planned technological processes, provides a significant reduction in the time and complexity of technological preparation of production. By the values of attributes calculated on the basis of the information received

from knowledge bases, with these operations, the selection of the necessary information from the database is carried out.

At the stage of decision-making, in accordance with the requirements and limitations of a higher level, a decision is made to execute a certain amount of PT.

At the design stage, the analysis of the selected PT, the development of the technological process in the form of a set of routes, descriptions, equipment selection, equipment, etc., is carried out.

At the planning stage, a plan is made for the manufacture of products with the corresponding technology on the technological equipment, which is the air force.

At the acquisition stage, the actual purchase of raw materials, semi-finished products, components, information necessary for the production of the product under the appropriate technology is carried out.

At the stage of production, a plan for manufacturing products is realized, which results in the implementation of the PT.

At the stage of quality control of the finished product, a comparison of the product with its specification and notification of non-compliance, if any, are made.

At the stage of delivery, the finished product, which passed the quality control, is sent to the consumer.

The functions performed on the listed stages are interrelated and can use data specific to one or another function, which are split between several functions, or common to all functions.

In solving the formation of the WPS, the formation and application of databases containing information obtained on the basis of the basic scientific provisions of the design technology is required; methods of mathematical modeling, system-structural analysis; theory of information, sets, mathematical logic, control, automated design and programming technology.

The model of the system of multi-objective technological design allows not only to provide functions and activities in the automated production, but is the basis for its system design.

The model is based on the concept of "controlled dynamic production", which performs the following successive stages: decision-making, design assessment, technological design, verification, control over the passage of the aircraft through the VPS.

Implementation of the mathematical models of the VPS operation takes into account that modern flexible automated production is based on the massive application of computer technology - starting from the Sun, which, as a rule, have built-in microprocessors, and ending with automated workplaces of designers, technologists, dispatchers, etc. By virtue of the physical distribution of these components objectively arises the task of creating an appropriate distributed computing system, computer, covering areas, workshops, factories, industries, etc.Ефективність управління

The effectiveness of management of the PS depends on the sequence and values of the decisions taken, as well as on the efficiency of the information received. In order to make the necessary decisions, it is necessary to obtain relevant information on the aircraft in real time, as well as about the past or the future. Since the time for processing information is limited, the analysis of the production situation and the formation of the appropriate command team requires the automation of the implementation of these actions. This leads to the need to use models simulating the main actions of the operator in the management of the aircraft. Such a system must have the elements of intellectual control. Generation of variants is based on an evolutionary method that uses genetic algorithms. To implement the generation of variants, a known method of combining heuristics. This method reduces the

required computing power of the entire genetic algorithm as a whole. Upon completion of the formation of the next versions of the configuration of the VPS, the process of their verification is carried out. The purpose of verification of the results is to evaluate the options and choose the best among them. In case, if at some stage of generation the generated version turns out to be able to work (it corresponds to the conditions of the target functions), such variant is considered as a worker, and on the basis of it the team is formed for practical realization in the VPS. Verification is a complex procedure, based on simulation simulation of processes occurring in the VPS. At individual stages of simulation, local optimization is carried out using such methods as linear programming, dynamic programming, etc. The choice of a particular method depends on the type of the current task. Simulation modeling allows to separate from the general task of simulation separate local, for solution of which these methods can be applied. The purpose of this phase is the attempt to find within the framework of the current airborne configuration of the best variant in terms of the volumes of production resources used in it under the provision of specified conditions. If the best configuration option that you receive does not meet the specified conditions, the ranking of the generated population occurs. On the basis of a ranked population, a new population is formed, and then the process is repeated until the working variant is obtained. After receiving arrays of data on technological operations, the execution of which in one or another composition and sequence provides execution of the WPS, it is necessary to formulate the final routes for their implementation and the sequence of launch in the Air Force. This task is complicated by the high computational power due to the high dimensionality. As the research shows, in solving such problems it is necessary to apply methods of evolutionary search for rational solutions.

Structural synthesis when designing technological processes

At the heart of the solution of the problems of structural synthesis of various complexity is the overcoming of options of the invoice set. When checking each sample includes:

- creation (search) of the next variant;
- the decision to replace the previously selected version with new ones;
- Continue or stop searching for new variations.

The tasks of structural synthesis in automated technological design depend on the level of complexity. In the simplest problems of synthesis (the first level of complexity), the structure of the technological process or its elements (operations, transitions) is determined. Then for a given class (group, subgroup, or type) of details the so-called generalized route (generalized structure) of processing is established. It includes a number of processing operations that are specific to a particular class, subclass or group of parts. The list is orderly and represents a multitude of existing individual routes. Routes have a typical sequence and content, and they reflect the advanced production experience of the enterprise or industry. At the third level of complexity of structural synthesis, the problem of choosing a variant of a structure in a plural with a large but finite number of known variants is solved. For solving such problems, algorithms of targeted selection are used (for example, algorithms of discrete linear programming); algorithms sequential, iterative and others; The task is to complete the search by restricting the search field to the stage of the formation of the output data.

The optimal strategy has the property that, whatever the way to achieve some state (technological transition), the following solutions should belong to the optimal strategy for the part of the surface treatment plan starting from this state (technological transition). A technology engineer working in a

dialog with a computer chooses such a variant of the structure, which represents the optimal compromise between the performance of the machine and the probability of providing a given quality of the workpiece. The computer helps the technologist to make a decision to change the structure, calculating the program modes and the performance of the machine.

The overall complexity of design can be reduced by switching from dialog mode to batch. Similar tasks are solved by applying training procedures (procedures for the formation of concepts). Recognition and classification programs are used as training procedures. At the same time there is a redistribution of routine and creative work using a batch mode of a higher level, the technologist is engaged in the preparation of output data and checks the final result.

The most complex level of structural synthesis is aimed at creating fundamentally new technological processes and is solved by so-called search engineering (artificial intelligence). One of the ways of search engine construction is to use the method of heuristic techniques:

- Explanation or formulation of technical task.
- Choice of one or several analogs (prototypes) of the technological process.
- Analysis of prototypes, identifying their drawbacks and formulating the problem in the form of answers to the question: what are the quality indicators in the prototype of the synthesis process and how much is it desirable to improve them? what new parts quality parameters should ensure the production process to be created and which quality parameters should lose the considered prototype?

The great difficulties encountered in search engine design and heuristic programming have led to the emergence of expert systems. The basis of expert systems is the database used by the expert (user technologist) in the dialogue mode. The

disadvantage of such systems is the dependence of the quality of design technological solutions (in particular, the design of route and operational technologies) from the level of expert preparation. Another drawback is to limit the range of tasks to be solved and their dimensionality. The need to increase the level of intellectualization of the automated process of synthesis of technological solutions at high dimensions of the solved problems requires the development of fundamentally new solutions, one of which was the creation and use of new methods and algorithms for the implementation of this work, and for them - the future.

I.9. SOFTWARE DEVELOPMENT METHODOLOGIES (RUP, XP, MSF, DSDM, RAD)

Rational Unified Process (RUP)

Rational Unified Process (**RUP**) is an iterative software creating process, created with Rational Software.

RUP Blocks

Main blocks are:

- Roles(who). The role determines a set of skills, attribution and responsibility.
- Operating products(what). An operating product is something received from an errand, including all the documents and models, produced during the operation on the process.
- Task(how). The task describes a unit of work, assigned for the role, that guarantees a significant result.

The tasks are divided into nine disciplines in every iteration: six "engineering discipline" (business-modeling, requirements, analysis and planning, realization, testing, deployment) and three supporting disciplines (configurations and changes of variables, projects management, environment).

Main RUP components

Six engineering disciplines

Business-modeling disciplines

Business-modeling explains, how to describe the view of the organization, in which the system is going to be engrained and how to utilize this view for assigning the process, roles and responsibilities.

Organizations are becoming more and more dependent on IT systems, which requires information systems engineers to know, in addition, that whatever they develop is inserted into the establishment. The first goal for the business-modeling is establishing a deeper understanding and communicational channel between business and software engineering. Understanding business means, that programmers must understand the structure and dynamics of a targeted organization(client), current problems in the organization and possible improvements. They also have to provide a general knowledge of a targeted organization among the clients, conclusive users and developers.

Disciplines of requirement

Requirements show exactly how to reveal requests of interested individuals and turn them into a set of requirements for the operating products.

Analysis and projecting discipline

The point of analysis and projecting discipline is to show, how the system is going to be realized.

Model design consists of class projecting, structured into packets and subsystems with clearly defined interfaces, that will represent, what will become the components in the final realization. It also contains a description of how the operands of these constructed classes cooperate for completing the precedent.

Realization discipline

The main goal for the realization is:

- To determine code organization from the point of view of a subsystem realization.
- Classes and objects realization in terms of the (output files, executed files, etc.).

Objective: results integration, received by individual operands (or groups) into the executed system.

Testing discipline

Testing objectives:

* To examine cooperation between operands.
* Check a due integration of all the software components.

Rational Unified Process offers an interactive approach, which means, that all the testing is done during the whole project. This allows to reveal defects as soon as possible, that drastically lowers the cost of fixing a defect. The tests are carried out by four quality surveys: reliability, functioning, supplement productivity and system productivity. For every one of these quality criterion surveys, the process describes how to pass the planning life cycle, designing, executing and test rating.

Dissemination discipline

The goal for the dissemination is to successfully make product versions and to supply software for conclusive users. It covers a wide field of measures, including manufacturing of external software versions, software and business-supplements packing, software installation and further support and assistance for users.

Project's life cycle phases

RUP Phases and Disciplines

RUP defines project's life cycle, which consists of four phases.

Initial phase

The initial goal is an adequate system estimate as a base for calculating primary valuations and budget. Business-cases are established at this stage, that include business-context,

success factors (expected income, market recognition etc.), and financial predictions.

If the project fails at this stage, which is called a life cycle's milestone, it can be canceled as well as iterated after being reconstructed with the purpose of satisfying the criteria.

Specification phase

The main goal is to make the key risks, discovered based on the analysis, more acceptable, until the very end of this phase. Specification phase is a phase where the project starts to get it's colors. The subject's province is being analyzed at this stage, the architecture of the project also starts to take its shape.

Constructing phase

The main goal for this phase is to create software system. All of the attention falls onto developing components and other characteristics of the system. All the main coding is being done at this stage. Larger projects can have several constructing phases. This stage creates the first software release.

Plantation phase

The main objective is transferring the system from developing into product, making it clear and comprehensible for the conclusive user. In terms of this phase the activity includes educating conclusive users and attendants, system testing for checking the users' expectation. The product is also being tested for quality standard, set in the initial phase. If all of the requirements are met, product release landmark is achieved and at this point the developing cycle is over.

Extreme Programming (XP)

Extreme Programming (*XP*) — one of the most flexible software developing methodology. The authors of this methodology are Kent Beck, Ward Cunningham, Martin Fowler and others.

The name comes from the idea to apply useful traditional methods and software development practice, lifting them to a new "extreme" level. This way, for example, code revision implementation practice, that relies on one programmer reviewing the code, written by another programmer, in the "extreme" version means "pair programming", meaning that the first programmer does the coding, while his partner continuously checks the code at the same time.

Basic XP techniques

All twelve basic extreme programming techniques can be combined into four group:

- Fine-scale feedback
  - Test-driven development
  - Planning game
  - Whole team, Onsite customer
  - Pair programming
- Continuous, not batch process
  - Continuous integration
  - Design improvement, Refactoring
  - Frequent small releases
- Understanding, shared by everyone
  - Simple design
  - System metaphor
  - Collective code ownership or Collective patterns ownership
  - Coding standard or Coding conventions
- Programmer's welfare

Testing

XP provides the writing of automated tests (software code, written specifically for purposes of testing other software codes). Special attention is payed to two types of testing:

- Module unit testing;
- Functional testing.

The developer can't be sure whether his code is written correctly, until absolutely all modules tests of the developed by him system work. Module tests (unit tests) allow the developers to make sure, that every one of them works correctly separately. They also help other developers understand the purpose of different parts of the code and how they work — the logic behind the tested code becomes clear during the examination of the test code, since it is obvious how it should be used. Module tests also allow the developer to refactor without any warning.

Functional tests are designed to test the functioning of the logic generated by the interaction of several parts. They are less detailed than unit tests, but when implemented, they relate to a large amount of code, so the chance to detect some improper behavior is obviously larger. Because of this, the writing of functional tests in industrial programming often takes higher priorities over writing unit-tests. An approach called TTD (test-driven development) is more important for XP. According to this approach, firstly, the test, that does not initially pass (as the logic it has to check does not exist yet) is written, than the logic that is needed to pass the test is implemented.

Game of planning

The main goal of the game of planning is to quickly form an approximate work plan and constantly update it as the terms of the task become more and more clear.

Game of planning has its own participants and its purpose. He himself reports on the need of one or another functionality. Programmers give an approximate valuation for each functionality. The customer and programmers have the same goal, but everyone uses a different way to achieve it: the customer chooses the most important tasks in accordance with the budget and the programmer evaluates the tasks in accordance with his capabilities for their implementation.

Ideally the scheduling game of customers and programmers involvement should be conducted systematically, before the start of the next iteration of development. This makes it simple to make adjustments to the project accordingly to the successes and failures of the previous iteration.

The customer is always nearby

An XP "customer" — is not the one paying the bills, but the conclusive user of the software product. XP states that the customer should always be available for phone calls and questions.

Pair programming

Pair programming assumes that the entire code is created by pairs of programmers working at the same computer. One of them works directly with the text of the program, while the other one looks through his work and looks at the general picture of what is happening. During the work on the project, the pairs are not fixed: they change so that each programmer in the team had a good idea of the whole system. Thus, dual programming enhances interaction within the team.

Continuous integration

In traditional techniques, integration, generally, is performed at the very end of the work on the product, when all components of the developing system are fully prepared. In XP, the integration of the code of the entire system is performed several times a day, after the developers are convinced that all module tests are working correctly.

The main task of continuous integration is to quickly find and fix errors, to improve the software quality and to reduce time spent on software verification and updates.

Refactoring

Refactoring is a method of code improving without changing its functionality. XP assumes that the code written in the process of working on the project, will almost certainly be

repeatedly reworked. XP developers recycle the previously written code to improve it.

Refactoring process is step-by-step changes accompanied by frequent start of the tests. The refactoring result is a pure code and a simple design.

Frequent small releases

Product releases should be put into operation as often as possible. The amount of work on each version should take as little time as possible. At the same time, each version should be sufficiently meaningful in terms of usefulness for business.

The sooner the first working version of the product is released, the better. The earlier the customer begins operating the product, the earlier the developers receive information from him about whether the product meets the customer's requirements. This information can come out rather useful in planning the next issue.

Simplicity of designing

XP proceeds from the fact that conditions of the problem can be repeatedly changed in course of the work, meaning that the developing product should not be designed at once completely in advance. An attempt to design the system in detail at the very beginning of the work is a waste of time. XP assumes that designing is an important process than needs to be carried out continuously throughout the whole project. Designing should be carried out in small stages taking constantly changing requirements into account.

System metaphor

Architecture is an idea about components of the system and their interconnections. Developers have to analyze software architecture in order to understand where the new functionality should be added in the system and what will interact with the new component.

System metaphor is an analog for what most techniques call architecture. The metaphor of the system gives the team an

idea of how the system currently works, where the new components are added and what form should they take.

Coding standards

All team members must comply with the requirements of the general coding standards during work.

If the team does not use common coding standards, it becomes harder for developers to refactor; there may be complications during the change of the partner in pair programming; project progress is slowing down.

It is necessary to ensure that it is difficult to understand who is the actual author of certain parts of the code in the framework of XP — the whole team works unified as a single person. The team must come up with a set of rules, then each member of the team must follow these rules in the process of writing the code.

Collective ownership

Collective ownership means each team member is responsible for the entire source code. This way everyone can make changes to any part of the program. Pair programming supports this practice: all programmers become acquainted with all the parts of the system code, working in different pairs. The important advantage of the collective ownership of the code lies in the fact that it accelerates the development process, because when the error is spotted, it can be corrected by any programmer.

Microsoft Solutions Framework (MSF)

Microsoft Solutions Framework (**MSF**) — software creation methodology, proposed by Microsoft. MSF bases itself on Microsoft practical experience and describes people and working processes management in solution development process.

MSF methodology consists of principals, models and personnel management, processes and technology elements disciplines, typical for most projects.

MSF consists of two models and three disciplines. MSF contains:

- **models**:
- project group model
- processes model
- **disciplines**:
- projects managing discipline
- risks managing discipline
- preparation managing discipline

Project group model

MSF (MSF Team Model) describes Microsoft approaches for personnel project worker's organization and his activity with a purpose of maximizing project's success. This model determines role clusters, their competence spheres and responsibility zones. It also determines recommendations for members of the project group, which allows them to successfully complete their mission of making the project a reality.

According to the MSF model, project groups are built as relatively not big multi-profile teams. Members of these teams divide responsibilities and supplement each other's cognizance area.

The MSF project group consists of six role clusters. Each one of them is responsible **for**:

- program management (program manager) — solution architecture developing, administrative service;
- development (developer) — application and infrastructure development, technological consultation;
- testing (QAE) — planning, tests developing and accounting according to tests;
- release management (release manager) — infrastructure, accompaniment, business-processes, finished product emission;

- customer satisfaction (user experience) — teaching, ergonomics. graphical design, technical support;
- production management (product manager) — business-priorities, marketing, representation of customer's interests.

The MSF project group offers braking big teams (more than 10 people) apart into small multi-profile direction groups (feature teams). These small groups work in parallel, synchronizing their efforts regularly.

MSF process model is a general development methodology and IT solutions introduction. The unique point about this model lies in its application during development of wide spread of IT projects due to its flexibility and absence of strictly intrusive procedures. This model combines the abilities of two standard models of life cycle: waterfall and spiral.

MSF process is oriented on milestones — key points of the project, that define achievements within its essential (transitional or conclusive) result. This result may be evaluated and analyzed.

MSF process model accounts for constant project requirements changes. It states that solution development should consist of short cycles, which create a progressive motion ranging from the earliest solution versions to its final look.

Processes model includes following main phases of process development:

- Conception developing (Envisioning)
- Planning
- Developing
- Stabilizing
- Deploying

Projects management — a specific area, during which certain goals are being set, balancing the amount of work, resources (such as money, labor, materials, energy, space etc.),

sometimes quality and risks. The key success factor for project management is an existence of a specific plan, set in advance, minimizing the amount of risks and plan deviations, an effective shift management.

Risks management

Risk management is one of Microsoft Solutions Framework (MSF) key disciplines. This process includes risk detection and analysis; prophylaxis strategy planning and realization and alleviating the potential consequences; tracking the state of the risks and learning one's lessons based on the received experience. **MSF motto — we do not fight the risks - we control them**.

Preparation management

Preparation management is also one of the key Microsoft Solutions Framework (MSF) disciplines. It is dedicated to knowledge management, professional skills and abilities, required for planning, creating and successful decision accompaniment.

Dynamic Systems Development Method (DSDM)

Dynamic Systems Development Method (DSDM) - is a software development technique based on the rapid development concept (Rapid Application Development, RAD).

RAD (Rapid application development) - is a concept of creating software development tools, that pays special attention to the programming speed and convenience, the creation of a technological process that allows the programmer to create computer programs as fast as possible. RAD has been widespread and approved since the end of the XX century.

Basic RAD

· The toolkit should be aimed at development time minimizing.
· Creating a prototype for customer requirements clarification.
· Development cycle: each new product is based on an evaluation of the previous version of the customer.
· Version development time minimizing by transferring the

already completed modules and adding functionality to the new version.

· The team of developers should cooperate closely and each participant must be prepared to perform several obligations.

· Project management has to minimize the development cycle duration.

DSDM - is and iterative approach, that gives a special meaning to the prolonged participation of the customer (user) in the process of working on the project. The goal of the method is to hand over a finished project on time and to invest in the budget, but at the same time adjusting requirements changes of the project during its development.

DSDM is a flexible software development methodology, as well as developments that are not part of the IT area.

DSMD consists of three stages: pre-project, project life cycle stage and a post-project stage. The project life stage consists of 5 phases:

- research of the possibility to be implemented;
- economic density research;
- creating a functional model;
- modeling and development;
- implementation phase.

It is possible to include parts of other techniques, such as the Rational Unified Process (RUP) and extreme programming (XP), into DSDM under certain conditions.

The DSDM method was developed in the UK at the end of the 19th - beginning of the 20th centuries by the DSDM consortium.

Principles of DSDM

- Involving the user (customer) is the basis for conducting an effective project, where the most accurate decisions are made.

- Frequent delivery of product versions. The analysis of versions from previous iterations is taken into consideration in the next one.
- Methodology of development is iterative. It is based on the customer's feedback in order to achieve the optimal solution from and economic standpoint.
- Testing is integrated into the development lifecycle.
- Interaction and cooperation between all participants is necessary for its effectiveness.

Preconditions for DSDM use

- it is necessary to organize interaction between the project team, future users and management team;

- a possibility to split the project into smaller parts, which will allow the usage of an iterative approach

Life cycle of the project and DSDM stages

DSDM consists of three successive stages: the pre-project stage, the life cycle stage of the project and the post-project stage. The main stage is the life cycle stage of the project: it consists of five phases, which form an iterative approach to the development of information systems.

Stage 1 - pre-project

At this stage, the probable parts of the project are identified, the fund allocation and the assignment of the project team is carried out. Solving tasks at this stage will help prevent problems at later stages of the project.

Stage 2 - life cycle of the project

Stages of the life cycle of the project:

- research:
- research of the opportunity to be implemented;
- economic density research;
- creation of a functional model;
- modeling and development;
- implementation.

Stage 3 - post-project

Efficient operation of the system is provided at this stage. The support of the project is held as a continuation of the development, based on the iterative nature of DSDM. Instead of completing the project in one cycle, it is common to go back to previous stages or phases, in order to improve the product.

Let's look at the life cycle stage of the project

Phase 1: research

- 1a: research of the possibility to be implemented

The possibility of implementing a project within the framework of DSDM is tested during this phase.

The final result of this phase is a report on the DSDM applicability to the project implementation, as well as an approximate global plan of the project and a protocol of possible project risks.

- 1b: economic density research

After recognizing the possibility of project implementation within DSDM, business processes are checked at this stage, groups of users are being involved, and their needs and wishes are analyzed. The use of working groups is the most requested method at this stage. Participants of the project discuss the planned system within the working groups; the received information is collected into the list of requirements, which are distributed according to priorities. Based on the received priorities, a development plan is created that will serve as a benchmark for the whole project.

In order to create this plan, a very important project methodic is used, called time-boxing. This methodic is required to achieve the goals of DSDM: to put up with time and budget, while maintaining the necessary quality of the product.

The result of this stage is a description of the sphere of commercial activity, a description of the architecture of the system and a development plan, which indicates the most important steps in the development process.

Phase 2: creating a functional model

The requirements that were defined at the previous stage are transformed into a functional model. It consists of a prototype and models. Prototyping is the key project method at this phase, which allows organizing an involvement of users into the project. The developed prototype is analyzed by different groups of users. Each iteration is tested in order to achieve the required quality.

Creation of a functioning model can be divided into the following sub-steps:

- Defining a functional prototype: determining a functional that will be laid in the prototype of the current phase.
- Plans adjustment: an agreement on how and when the prototype functionality should be developed is taking place.
- Creating a functional prototype. The prototype created during previous iterations is studied and refined.
- An analysis of a functional prototype is to check whether the designed system is in a good condition. Testing and reviewing is applied.

The outcome of this phase is a functional model and a functional prototype, which together represent the functionality obtained in this iteration, ready for testing by the user. The list of requirements is then updated and the already accomplished items are removed from it. The protocol for potential risks is also updated.

Phase 3: Design and development

The main task for this iteration is to merge the functional components from the previous stage into a single system, that meets the requirements of users. Testing is underway again at this stage.

The summary for the stage is the creation of a constructive prototype for testing by users. The tested system than moves to the next phase. The appearance and functionality

of the system are generally ready at this stage. Another result the creation of custom documentation.

Phase 4: implementation

At the implementation stage, the tested system, along with the user documentation, are delivered to future users and their training with the system is carried out. The system is analyzed to meet the requirements set at the early stages of the project.

Implementation can be divided into the following sub-phases:

• System approval by the user: conclusive users approve the tested system for further implementation.

• User training: training the future user to operate the system.

• Implementation: implementation of the tested system among users.

• Market of the system analysis: analysis of the influence of the released system on the market. The main question is whether the goal, set for designing the system, has been achieved.

Based on this, the project either moves on to the next stage or returns to the previous one further refinements.

Stage summary: a finished system, suitable for usage by conclusive users, and a detailed project analysis document.

Basic DSDM methodics

- ***Time-boxing***

Time boxing is one of the main DSDM methodics. It is used to achieve the main objectives of DSDM: to develop the information system in time, to commit to the budget while maintaining quality. The main idea of time-boxing is to divide the whole project into parts, each one with its own budget and time limits.

- ***MoSCoW***

MoSCoW method opens up a way to divide objects by priority. In the context of DSDM, the MoSCoW method is used to prioritize requirements. The abbreviation stands for:

- MUST - the requirements MUST meet the economic needs.

- SHOULD – SHOULD this requirement be met, if the project does not depend on its success.

- COULD – whether you COULD leave this requirement if it is not applicable to the business need of the project.

- WON'T - whether you WON'T be able to postpone the fulfillment of the requirement if you still have time to spare.

- ***Prototyping***

This methodic relates to the creation of the prototypes of the system during its development in the early stages. It allows you to identify deficiencies in the system and allows future users to test it. Thus user engagement in work is implemented, being one of the success key factors of the DSDM method.

- ***Testing***

The third and important part of achieving the DSDM goal is to create a high-quality information system. In order to achieve this, the DSDM system insists on conducting the testing of each iteration. The project team is free to choose the way to manage the testing.

- ***Workgroup***

This is one of the DSDM methodics, which aims to bring together different participants of the project in order to discuss the requirements, functionality and adjust mutual understanding.

- ***Modeling***

This methodic is mandatory and is used to visualize the individual side of the system or sphere of activity in the form of diagrams. Simulation provides a better understanding of the project's business activity for the entire team.

- ***Configuration management***

A good implementation of the configuration management technique is important, because of the dynamic nature of DSDM. A strict control over the quality of the products is required, since they are released quite often.

I.10. METHODS OF TESTING AND ADJUSTING PROGRAMS AND SYSTEMS

The fundamental concept of SS design includes basic conditions, strategies and methods, that are used in LC processes and provide testing (verification) on a multitude of trial data sets. The SS designing methods include structural, object-oriented and other methods. They are based on theoretical, instrumental and applied tools that influence the testing process.

Testing is a process of identifying errors in software by executing the source code of the SS on the test data, collecting performance data in the performance dynamics in a specific operating environment, detecting various errors, defects, failures and bugs, caused by irregular and abnormal situations or accidental software termination. Organizational aspects play an important role in the conduct of verification and testing: the activity of a group of specialists who plan these processes, preparation of tested data and monitoring of tests.

Processes of LC (life cycle) of verification and validation of programs

Verification and validation as methods ensure, respectively, the verification and analysis of the correct implementation of the specified functions and software compliance with the requirements of the customer, as well as assigned specifications. In standards, they are presented as independent LC processes and are used starting from the stage of the analysis of requirements and finishing with a verification of the correctness of the code functioning at the final stage, namely, testing.

For these processes, the goals, tasks and actions for verifying the correctness of the created product (working, intermediate products) are defined at the stages of the LC. Let's look at their interpretation:

1.1. Verification process

The goal of the project - is to make sure that each software product of the project reflects the agreed requirements for their implementation. The process is based on:

• strategy and verification criteria for all working software products;

• implementation of standard actions of verification;

• eliminating the shortcomings found in software products;

• on agreed results of verification with the customer.

The verification process can be carried out by the program executor or another employee of the same organization or an employee of another organization, such as the customer. This process includes actions for its implementation and execution.

The implementation of the process is to identify critical elements (processes and software products) that have to be verified, the choice of the verifier, the tools supporting the verification process, the preparation of the verification plan and its approval. When verifying according to the plan and requirements of the customer, the correctness of the system functions, interfaces and interconnections of the components, as well as access to the data and to the means of protection are checked.

1.2. Validation process

The goal of the process is to make sure that the specific requirements for the software product are met, and this is accomplished by:

• developed strategy and validation criteria for all work products;

• conditioned validation actions;

• demonstration of the developed software products accordance to the requirements of the customer and rules of their use;

• coordination of the results of validation with the customer.

Validation process can be carried out by the executor or another person, for example, the customer, who is taking actions on the implementation and execution of this process according to the plan, which displays the elements and tasks of verification. It uses methods, tools and procedures to perform the tasks of the process to establish compliance test requirements and features of the use of project software products.

Additional actions are taken at other stages of LC:

• checking and controlling design decisions with the help of revision of the working process techniques and methods;

• access to the CASE-system, which contain procedures for checking product requirements;

• reviews and inspections of the interim results for compliance with their requirements in order to confirm that the software meets the requirements and satisfies the conditions of the system execution.

Thus, the main task of the verification and validation processes lies in verification and confirmation, that the conclusive product meets its purpose and satisfies the requirements of the customer. These processes are interconnected and are usually defined by a common term "verification and validation" (V & V).

V & V is based on process planning and verification for the most crucial elements of the project: a component, interfaces (software, technical and informational), objects

interactions (protocols and messages) and data transfers between components and their protection.

After checking individual components of the system, they integrate, re-verify and validate the integrated system; a set of documentation is created that reflects the accuracy of the verification of requirements formation, inspection results and testing.

Programs testing

Testing can be interpreted as a process of semantic debugging (verification) of the program, which consists in executing the sequence of different sets of final tests for which the result is known beforehand. That is, the testing involves the implementation of the program and obtaining concrete results of the tests.

Tests are gathered up so that they cover as many types of situations as possible in the program algorithm. The less strict requirement is for every brunch to be executed at least once.

Historically, the first kind of testing was debugging.

Debugging is a verification of a description of a software object in order to detect errors and their further elimination. Compilers detect errors with their syntactic control. After that, a verification is carried out to verify the correct code and validation to verify the compliance of the product with the requirements.

The purpose of the test is to check the performance of the functions implemented in accordance with their specifications. Functional tests are created based on the basis of external specifications of functions and design information on the processes of the LC. Due to those tests, the testing is conducted taking into account the requirements, formulated at the stage of domain analysis.

Methods of functional testing are divided into static and dynamic.

Static testing methods

Static methods are used during inspections and consideration of component specifications without their execution. The technique of static analysis lies in methodological review and analysis of the structure of the programs, as well as in proving their correctness. Static analysis is aimed at analyzing the documents developed at all stages of the LC and exists in order to inspect the source code and cross-check of the program. Software Inspection is a static verification of the program's compliance with the given specifications. It is carried out by analyzing various representations of the results of the projection (documentations, requirements, specifications, schemes or source code of programs) in the process of LC. Reviews and inspections of the projecting results and their compliance with the customer's requirements provide a higher quality of the created LC. Documents of the operational design of LC are reviewed during the inspection of the program alongside with independent experts and participants of the development of the LC. At the initial designing stage, the inspection involves verification of completeness, integrity, singularity, consistency and compatibility of documents with the initial requirements to the software system. At the stage of implementation of the system, inspection refers to the analysis of the texts of programs to comply with the requirements of standards and adopted guidance documents of programming technology.The effectiveness of this examination is that the involved experts try to look at the problem "from a different angle" and subject it to a comprehensive critical analysis.

Dynamic testing methods

Dynamic testing methods are used in the implementation of programs. They are based on a graph that binds the causes of errors with the expected responses to these errors. In the process of testing, the information, accumulated

on the errors, is used in assessing the reliability and quality of the SS.

Dynamic testing is oriented at verifying the correctness of the SS based on a number of tests with purpose of checking and collecting data at the stages of LC and conducting the measurements of individual indicators (the number of failures, malfunctions) of testing to assess the quality characteristics specified in the requirements. Testing is based on systematic, static (probabilistic) and simulation methods.

Let's give them a short reference.

Systematic testing methods are divided into methods in which programs are viewed as a "black box" (using information about a solvable problem), and methods, in which the program is viewed as a "white box (using structure of the program). This type is called data management testing or input / output management. Its purpose is to clarify the circumstances in which the behavior of the program does not meet its specifications. In this case, the number of errors found in the program is a criterion for the quality of the input testing.

The purpose of the dynamic testing of programs relies on the principle of the "black box" - the detection of the maximum number of errors with a single test using a small subset of possible input data.

Functional testing

The purpose of functional testing is to identify inconsistencies between the actual behavior of the implemented functions and the expected behavior in accordance with the specification and output requirements. Functional tests should cover all implemented functions, taking into account the most common types of errors.

The tasks of functional testing include:

• Identification of a set of functional requirements;

• identification of external functions and the construction of a sequence of functions in accordance with their use in the SS;

• identification of the set amount of input data of each function and determination of areas of their alteration;

• construction of test kits and function testing scripts;

• Identification and presentation of all functional requirements through test kits and testing for errors in the program and in conjunction with the environment.

Tests created by project information are related to data structures, algorithms, interfaces between individual components, and are used to test components and their interfaces. The main objective is to ensure the completeness and consistency of the implemented functions and the interfaces between them.

The combined method of "black box" and "white box" is based on the division of the input field of a function on the subfield of error detection. The subfield contains homogeneous elements that are processed correctly or incorrectly. To test the subfield, the execution of the program is performed on one of the elements of this field.

Infrastructure of the testing process of the SS

When talking about *infrastructure of the testing process*, it means:

•allocating of testing subjects;

•carrying out the error classification for the regarded class of tested programs;

• preparation of tests, their execution and search for various kinds of bugs and errors in components and in the system as a whole;

• service of conduction and management of the testing process;

• analysis of test results.

Test objects are components, group of components, subsystems and systems. A testing strategy is being developed for each one of them. If the testing object refers to "white box" or "black box" while the compound of components is unknown, then the testing is done by inputting the input test data to obtain the source data. The strategic goal of testing is to make sure that each considered input data set corresponds to the expected output data. With this testing approach the internal structure knowledge and knowledge of logic of the tested object is unnecessary.

The test designer has to look inside the "black box" and explore the details of the data processing, protection and data recovery issues, as well as interfaces with other programs and systems.

For some types of objects, the testing team can not generate a representative set of test kits that demonstrate the functional integrity of the component at all possible test kits. Therefore the "white box" method is advisable, which allows you to use the structure of the object to organize testing on different branches. For example, you can run test kits that pass through all operators or all checkpoints of the component in order to make sure in to make sure they work properly.

Methods of searching for errors in programs

The international ANSI/IEEE standard separates all errors in the development of programs into the following types:

• *Error* - a state of a program, in which the wrong results are given due to defects in the program operators or in the technological process of the development, which leads to an incorrect interpretation of the source information, and therefore to the wrong decision.

• A *fault* is a consequence of the developer's errors at any of the stages of development. They can be contained in the initial or project specifications, program code texts, operational

documentation, etc. A defect or a failure can be detected in the process of execution of the program.

• Failure is a refusal of the program to function or an inability to perform the functions defined by the requirements and limitations. It is considered as an event to be the cause of the transition of the program into a non-operating state due to errors, hidden somewhere in the program or due to failures in the functioning environment.

All of the potential mistakes in the program are divided into the following *classes*:

• logical and functional errors;
• calculation and runtime errors;
• input/output and data manipulation errors;
• interface errors;
• errors of data capacity, etc.

Logical errors are the reason for the violation of the logic of the algorithm, the internal inconsistency of variables and operators, as well as the rules of programming. Functional errors are the result of improperly defined functions, violation of the order of their application or a lack of completeness of their implementation, etc.

Calculation errors usually come up as a result of inaccuracy of the initial data and implemented formulas, mistakes in the methods, misuse of operations of calculations or operands. Runtime errors are caused by a failure to provide the required processing speed or recovery time.

Input/output and data manipulation errors are the result of poor data preparation for execution of the program, failures in entering or sampling them in the database.

Interface errors relate to the mistakes of the relationship of individual elements with each other, which manifests itself in the transmission of data among them as well as when interacting with the environment of functioning.

C*apacity* errors relate to data and are a consequence of the fact that implemented methods of access and database capacities do not meet the actual capacities of information of the system or the intensity of their processing.

These basic classes of errors are inherent in different types of software components and they appear in programs in different forms. Thus, when working with DB errors of data management and manipulation, logical errors in the task of application procedures of data processing appear. Computational errors prevail n computing programs, while there are logical and functional errors in the management and processing programs. Software that consists of many multiple programs that implement different functions may contain different types of errors. Interface errors and capacity violations are typical for any type of system.

Analysis of the types of errors in the programs is a prerequisite for creating test plans and testing methods to ensure software correctness. At the present stage of the software development support tools (CASE technologies, object-oriented methods and tools for models and programs designing) a specific projecting is carried out: the software is protected from the most typical errors and therefore prevents the occurrence of software defects.

Classification of errors and tests

Errors

Based on long-term activity in the field of software development, different firms have created their classification of errors, grounded on identifying the reasons for their appearance in the development process, in the functions and in the field of functional activities of the software.

We know many different approaches to the classification of errors, let's take a look at some of them.

IBM has developed an approach to the classification of errors, which involves the breakdown of errors into categories with developers being responsible for them (table I.10.1).

Table I.10.1

Classification of errors (developed by IBM)

Context errors	**Classification of errors**
Function	Interface errors of conclusive users caused by hardware or related to global data structures.
Interface	Errors in interaction with other components, in calls, macros, control blocks or in the list of settings.
Logic	Errors in program logic, as well as in the use of variables.
Assignment	Errors in the data structure or in initializing the variables of individual parts of the program.
Looping	Errors caused by time resource, real time or time distribution.
Environment	Errors in the repository, in the management of changes or in conclusive versions of the project.
Algorithm	Errors associated with ensuring the efficiency, correctness of algorithms or data structures of the system.
Documentation	Errors in maintenance documents records or in publications.

Hewlett-Packard used the Butch classification by setting the percentages of errors found in the software at different stages of development (Fig. I.10.1)

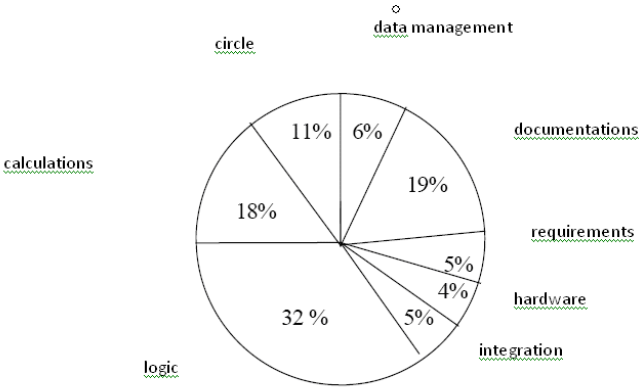


Fig. I.10.1. Hewlett-Packard errors classification

Tests

Let's look at a classification of the verification tests on tested objects at the main stages of development:

✓ Testing of specifications

- checking the completeness and consistency of functions;
- checking of interfaces consistency.

✓ Program testing

- verifications of the structure of the program;
- verifications of calculations and data transformations.

✓ Testing of the complex

- checking the structure of the complex;
- checking the component interface;
- checking the memory limit;
- checking the execution duration;
- checking the completeness of the complex problems solving.

✓ Tests during trial

- compliance verification;

- checking the convenience of the installation of the working version;
- checking the working process of the complex on the equipment;
- user interface verification;
- checking the comfort of the escort.

Software testing service

Developers and customers are both responsible for testing.

In order to achieve the testing objectives, a software verification service is usually created - a team of testers that does not depend on the state of the developers of the software. This team includes analysts, programmers, test engineers.

Testers compile test plans, test data and scripts, as well as test schedules from the very beginning. Professional testers work in conjunction with the configuration management team in order to provide their documentation and other mechanisms for linking each other with the requirements of the project, configuration, and code.

Many experts compare the testing system with the creation of a new system, in which analysts reflect the needs and objectives of the customer, working together with designers and seeking to implement the ideas and principles of the system. The errors found in the program and changes in the system are reflected in the documentation, requirements, projects, as well as in the descriptions of input and output data or in other developed artifacts. The changes made during the development process lead to the modification of the test scenarios or, to a large extent, to the change of the testing plans. Configuration management specialists take these changes into account and coordinate compilation of tests.

The test team also includes users. They evaluate the results, utilization simplicity, and also express their opinion about the principles of the system.

Testing process management

All software testing methods are merged into a database, which contains the results of the system testing. It contains all the components, test data, test results, and information about documenting the testing process.

The project database is supported by special tools, like CASE, which provide analysis, object data collection, data flows, etc. The project database also stores the initial and referenced data, used to compare the data accumulated in the database to the data obtained during the system testing.

Various types of calculations of the characteristics of this process and the methods of planning and management are carried out during the testing process:

- calculation of the duration of the performance of functions by collecting average performance of operators index without executing the program on the machine. Some components appear, which require a huge amount of time in the real environment;

- the management of the implementation of the test by selecting the tests of verification, their execution, conducting a comparison with the reference values. The results of this process are displayed on the screen, for example, in a graphical form (path through the graph of the program), in the form of UML diagrams, data on the failures and errors, or specific values of the output arguments of the program. This data is analyzed by the developers to formulate the conclusions about directions of further verification of the correctness of the program or their completion.

- Testing planning is intended for the distribution of the terms of testing work, the distribution of testers for certain types of work and their ability to compile tests for system testing. Certain tests, criteria an input values are created during the process of coming up with the plan for execution paths.

I.11. MODELS OF QUALITY AND RELIABILITY IN SOFTWARE ENGINEERING

The development of the PS reached such a level of development that it became necessary to use engineering methods, including for evaluating the results of design at the LC stages, monitoring the achievement of quality indicators and metric analysis thereof, assessing the risk and the degree of use of the finished components to reduce the cost of developing a new project. The basis of engineering methods in programming is a quality improvement, for the achievement of which methods for determining quality requirements, approaches to selecting and improving models of metric analysis of quality indicators, methods for quantitative measurement of quality indicators at the LC stages were formed.

Software quality is the subject of standardization. Standard GOST 2844-94 defines the quality of software as a set of properties (quality indicators) software that provides its ability to meet the needs of the customer in accordance with the purpose. This standard regulates the basic model of quality and indicators, the main one among them is reliability. The ISO / IEC 12207 standard defined not only the main processes of the PC's development of the PS but also the organizational and additional processes that govern the engineering, planning and quality management of the PS.

According to the standard at the stages of the LC, software quality *control* should be carried out:

- verification of compliance with the requirements of the product and the criteria for achieving them;
- verification and certification (validation) of the intermediate results of the software in the phases of the LC and measuring the degree of satisfaction of the individual indicators achieved; testing of the finished PS, a collection of data on failures, defects and other errors detected in the system;

- selection of reliability models for assessing reliability based on the results of testing (defects, failures, etc.);
- assessment of quality indicators specified in the requirements for the development of the PS.

The following describes models of quality and reliability, as well as ways to use them.

Software quality model

Software quality is a relative concept that makes sense only when real conditions of its application are taken into account, therefore, the requirements for quality are set in accordance with the conditions and the specific area of their application. It is characterized by three aspects: the quality of the software product, the quality of the LC processes and quality of support or implementation (fig. I.11.1)

Fig. I.11.1. The main aspects of software quality

The quality of the product is achieved by the procedures for controlling intermediate products on the LC processes, by checking them for achieving the required quality, and by the methods of accompanying the product. The effect of implementing the PS is largely dependent on the knowledge of the staff of the product functions and the rules for their implementation.

The software quality model has the following four levels of presentation.

The first level corresponds to the definition of characteristics (indicators) of software quality, each of which reflects a separate point of view of the user on the quality. According to the standard, the quality model includes six characteristics or six quality indicators:

1. functionality;
2. reliability;
3. usability;
4. efficiency;
5. maintainability;
6. portability.

The attributes for each quality characteristic corresponds **to the second level** that detail the different aspects of a particular characteristic. A set of attributes of quality characteristics is used in assessing quality.

The third level is designed to measure quality with the help of metrics, each of them according to the standard is defined as a combination of the method of measuring the attribute and the scale of measuring the values of attributes. To assess the quality attributes at the LC stages (when reviewing documentation, programs and test results of programs), metrics with a given estimated weight are used to level the results of the metric analysis of the aggregate attributes of a particular indicator and the quality as a whole. The quality attribute is determined using one or more evaluation methods at the LC stages and at the final stage of software development.

The fourth level is the estimated element of the metric (weight), which is used to estimate the quantitative or qualitative value of a separate attribute of the software indicator. Depending on the purpose, features, and conditions of software maintenance, the most important quality characteristics and their attributes are selected.

The selected attributes and their priorities are reflected in the requirements for system development or the corresponding

priorities of the software class standard to which this software applies.

Characteristics of quality indicators

The characteristics of the software quality indicators are shown in Fig.I.11.2

1. Functionality is a set of properties that determine the ability of a software to perform a list of functions in a given environment and in accordance with the processing requirements and system-wide means. A *function* is understood as an ordered sequence of actions to satisfy consumer properties. Functions are a target (basic) and auxiliary.

The attributes of functionality include:

functional completeness is a component property that shows the degree of sufficiency of the main functions for solving problems in accordance with the purpose of the software;

- *correctness* (accuracy) is an attribute that indicates the degree to which the correct results are achieved;
- *Interoperability* is an attribute that shows the ability to interact with software by special systems and environments (OS, network);
- security - an attribute that indicates the ability of the software to prevent unauthorized access (accidental or willful) to programs and data.

2. Reliability is a collection of attributes that determine the software's ability to convert raw data into results under conditions that depend on the lifetime period (wear and aging are not taken into account). The decrease in software reliability occurs due to errors in requirements, designing, and implementation. Refusals and errors in programs appear for a specified period of time.

he sub-characteristics of software reliability include:

- *Fault-freeness* is an attribute that determines the ability of a software to function without failures (both programs and equipment);

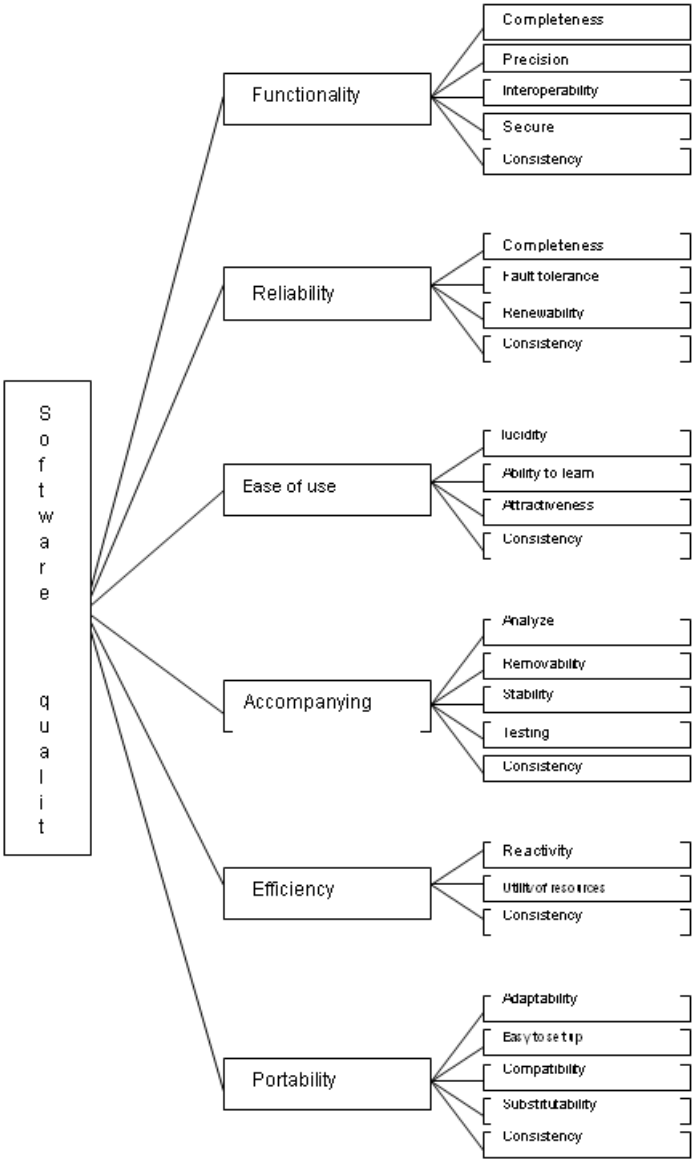


Fig. I.11.2. Model of software quality characteristics

- *resistance to errors* is an attribute that indicates the software's ability to perform functions under abnormal conditions (hardware failure, data and interface errors, violation of operator actions, etc.);
- *restore ability* is an attribute that indicates the ability of the program to restart for re-execution and recovery of data after failures.

3. The ease of use is characterized by a multitude of attributes that show the necessary and suitable conditions for use (dialog or non-dialog) of the software by a specified range of users to obtain relevant results. In the standard, the usability is defined as a specific set of attributes of the software product, characterizing its ergonomics.

The sub-characteristics of ease of use include:

- understandability - an attribute that determines the effort spent on recognizing the logical concepts and conditions for using the software;
- study ability (ease of learning) is an attribute that determines the user's efforts to determine the applicability of the software by using operational control, diagnostics, and procedures, rules, and documentation;
- efficiency - an attribute that indicates the response of the system when performing operations and operational control;
- consistency is an attribute that shows the compliance of a development with the requirements of standards, agreements, rules, laws, and regulations.

4. Efficiency is a set of attributes that determine the relationship between the levels of software execution, the use of resources (tools, equipment, materials - paper for the printer, etc.) and services performed by the staff, etc.

To the sub-characteristics of the effectiveness of the software are:

- reactivity is an attribute that shows the response time, processing, and execution of functions;
- resource efficiency - an attribute that shows the amount and duration of resources used when executing software functions;
- consistency is an attribute that shows the correspondence of this attribute with the specified standards, rules and regulations.

5. **The accompaniment** is a set of properties that show the efforts that need to be spent on making modifications that include updating, improving and adapting the software when the environment, requirements, or functional specifications change.

Accompaniment includes the sub-characteristics:

- analysability is an attribute that determines the necessary effort to diagnose failures or identify parts that will be modified;
- variability is an attribute that determines the removal of errors in the software or the introduction of changes to eliminate them, as well as the introduction of new features in the software or in the operating environment;
- stability - an attribute indicating the constancy of the structure and the risk of its modification;
- testability is an attribute that demonstrates the efforts to perform validation and verification in order to detect inconsistencies in requirements, as well as the need for software modification and certification;
- consistency is an attribute that shows the correspondence of this attribute to conventions, rules, and regulations of the standard.

6. Portability - the set of indicators that point to the ability of the software to adapt to the new environment of the runtime. The medium may be organizational, hardware and software. Therefore, the transfer of software into a new runtime environment can be associated with a set of actions aimed at ensuring its functioning in an environment different from the

environment in which it was created taking into account new programmatic, organizational and technical capabilities.

Portability includes sub-characteristics:

- adaptivity - an attribute that determines the effort spent on adapting to different environments;

- customizability (ease of installation) - an attribute that determines the necessary effort to run this software in a special environment;

- coexistence - an attribute that determines the possibility of using special software in the environment of the current system;

- replaceability - an attribute that provides the possibility of interoperability when working together with other programs with the necessary installation or adaptation of software;

- consistency - an attribute that indicates compliance with standards or software transfer agreements.

Software quality metrics

At present, the system of metrics has not yet been fully developed in software engineering. There are different approaches to determining their set and methods of measurement.

The measurement system includes metrics and measurement models that are used to quantify software quality.

When determining the software requirements, the external characteristics and their attributes (sub-characteristics) corresponding to them are defined, which determine the different aspects of product management in a given environment. For a set of quality characteristics of software, given in the requirements, the relevant metrics, models for their evaluation and the range of values of measures for measuring individual quality attributes.

According to the standard, metrics are defined by the software attribute measurement model at all stages of the LC

(intermediate, internal metric) and especially during the testing or operational phase (external metrics) of the product.

Let us dwell on the classification of software metrics, the rules for conducting metric analysis and the process of measuring them.

Types of Metrics

There are three types of metrics:

- software product metrics that are used to measure its characteristics - properties;
- process metrics that are used to measure the process property of the product creation process.
- usage metrics.

The metrics of the software product include:

- external metrics that indicate product properties visible to the user;
- internal metrics that denote properties visible only to the development team.

External product metrics are metrics:

- product reliability, which serves to determine the number of defects;
- functionality, through which the presence and correctness of the implementation of functions in the product;
- tracking, by which the resources of the product are measured (speed, memory, environment), the applicability of the product, which help determine the degree of availability for study and use;
- cost, which determines the cost of the created product.

Internal product metrics include:

- size metrics needed to measure a product using its internal characteristics;
- complexity metrics required to determine the complexity of the product;

- style metrics that serve to define approaches and technologies for creating individual components of a product and its documents.

Internal metrics allow you to determine product performance and are relevant to external metrics.

External and internal metrics are specified at the stage of the formation of software requirements and are the subject of planning and managing the achievement of the quality of the final software product.

The development time, the number of errors found during the testing phase, etc. can be as the **process metrics**. Practically the following process metrics are used:

- total development time and separately time for each stage;
- modification time;
- time of work on the process;
- number of errors found during inspection;
- cost of quality control;
- the cost of the development process.

Usage Metrics serve to measure the degree of satisfaction of the user's needs when solving his tasks. They help to evaluate not the properties of the program itself, but the results of its operation - the operational quality. An example can serve - the accuracy and completeness of the implementation of user tasks, as well as the spent resources (labor, productivity, etc.) to effectively solve the tasks of the user. The user's requirements are assessed using external metrics.

Standard assessment values of quality indicators

Evaluation of software quality according to the four-level model of quality begins with the lower level of the hierarchy, i.e. from the most elementary property of the evaluated attribute of the quality index according to the established measures. At the design stage, the values of the

evaluation elements for each attribute of the indicator of the analyzed software included in the requirements.

Quality metrics are used in assessing the degree of testability with the help of data (failure-free work, the feasibility of functions, the usability of user interfaces, databases, etc.) after testing software on a variety of tests.

MTBF as an attribute of reliability determines the average time between the emergence of security threats and provides a hard-to-measure estimate of damage that is caused by appropriate threats. Very often the evaluation of the program is based on the number of lines. When comparing two programs that implement one application task, a short program is preferred, as it is created by more qualified personnel and has fewer hidden errors and is easier to modify. At the cost, it is more expensive, although the time for debugging and modification takes more. Those. the length of the program can be used as an auxiliary property for comparing programs, taking into account the same developer skills, a single style of development and a common environment.

Based on the measurement of quantitative characteristics and the examination of qualitative indicators using weighting factors that neutralize different indicators, the final evaluation of product quality is calculated by summing the results by individual indicators and comparing them with the reference software indicators (cost, time, resources, etc.).

Ultimately, the result of the quality assessment is a criterion for the effectiveness and appropriateness of applying design methods, tools, and techniques for evaluating the results of creating a software product at the stages of the LC.

To express an assessment of the values of quality indicators, a standard is used in which the following methods are presented: measuring, registration, calculation, and expert (and combinations of these methods).

The measuring method is based on the use of measuring and special software to obtain information about the characteristics of software, for example, determining the volume, the number of lines of code, operators, the number of branches in the program, the number of entry points (output), reactivity, etc.

The registration method is used to calculate the time, the number of failures or failures, the beginning and the end of the software during its execution.

The calculation method is based on statistical data collected during testing, operation and maintenance of the software. Calculation methods assess the reliability, accuracy, stability, reactivity, etc.

The expert method is carried out by a group of expert-specialists who are competent in solving this task or the type of software. Their evaluation is based on experience and intuition, and not on immediate results of calculations or experiments. This method is carried out by viewing programs, codes, accompanying documents and contributes to a qualitative assessment of the created product. For this purpose, controlled characteristics are established that are correlated with one or more quality indicators and are included in expert questionnaires. The method is used in assessing such indicators as the analyticity, documentation, structuring of software, etc.

PS quality management

Quality management refers to the totality of the organizational structure and responsible persons, as well as the procedures, processes, and resources for planning and managing the achievement of the quality of the PS. Quality Management - SQM (*Software Quality Management*) is based on the application of standard provisions for quality assurance - SQA (*Software Quality Assurance*).

The objective of the SQA process is to ensure that products and processes are consistent with the requirements, consistent with the plans and include the following activities:

- implementation of standards and relevant procedures for the development of the PC at the stages of the LC;
- assessment of compliance with these standards and procedures. The quality guarantee is as follows:
- check consistency and feasibility of plans;
- harmonization of intermediate work products with planned indicators;
- verification of manufactured products to specified requirements;
- analysis of applied processes for compliance with the contract and plans;
- agreement with the customer environment and product development methods;
- check the accepted metrics of products, processes, and methods of measuring them in accordance with the approved standard and measurement procedures.

The purpose of the SQM management process is monitoring (systematic control) of quality to ensure that the product will satisfy.

Quality engineering includes a set of methods and activities through which software products are tested to meet quality requirements and are supplied with the characteristics required by the software requirements.

Quality system (QS) is a set of organizational structures, methods, activities, processes, and resources for implementing quality management. Two approaches are used to ensure the required level of software quality. One of them is focused on the final software product, and the second - in the process of creating a product.

Planning for quality is an activity aimed at defining goals and requirements for quality. It encompasses

identification, setting goals, quality requirements, classification and quality assessment. A calendar plan is drawn up for the analysis of the state of development and the subsequent measurement of the planned indicators and criteria at the stages of the LC.

Models of reliability assessment

Of all the areas of software engineering, the reliability of the PS is the most explored area. It was preceded by the development of the theory of reliability of technical means, which had an impact on the development of reliability of the substation. PS software developers dealt with PS reliability, trying to provide reliability that satisfies the customer by various system means, as well as theorists who, studying the nature of PS functioning, created mathematical reliability models that take into account different aspects of PS operation (errors, failures, failures, etc.) and allowing to estimate real reliability. As a result, the reliability of the PS was formed as an independent theoretical and applied science.

The reliability of complex PCs differs significantly from the reliability of the equipment. Data carriers (files, server, etc.) have high reliability, records on them can be stored for a long time without destruction since they are not subject to physical destruction.

From the point of view of applied science, *reliability* is the ability of the PS to retain its properties (*reliability, stability*, etc.), Convert raw data to results for a certain period of time under certain operating conditions. The decrease in reliability of the substation occurs due to errors in requirements, design, and implementation. Failures and errors, depending on their performance and time in the programs when they are executed for a certain period of time.

For many systems (programs and data), reliability is the main target function of implementation. To some types of systems (real-time, radar systems, security systems, medical

equipment with built-in programs, etc.) high-reliability requirements are imposed, such as the absence of errors, reliability, safety, etc.

Thus, the evaluation of the reliability of the PS depends on the number of remaining and not eliminated errors in the programs. During the operation of the MS, errors are detected and eliminated. If no new or at least new errors are introduced in the correction of errors, then it eliminates the reliability of the PS continuously during operation. The more intensive the operation, the more errors are detected and the reliability of the system grows faster and, accordingly, its quality.

Reliability is a function of the errors remaining in the MS after commissioning it. PS without errors is absolutely reliable. But for large programs, absolute reliability is almost unattainable. The remaining undetected errors manifest themselves from time to time under certain conditions (for example, with a certain set of initial data) to maintain and operate the system.

To assess the reliability of the PS, statistical indicators such as *probability* and *time of trouble-free operation*, the possibility of failure and frequency (*intensity*) of failures are used. Since only errors in the program that can not be eliminated are considered as the causes of failures, the PS should be classified as a class of non-renewable systems.

Reliability assurance factors include:

- risk as a combination of threats leading to adverse consequences and damage to the system or environment;
- threat as a manifestation of instability that violates the security of the system;
- risk analysis - the study of the threat or risk, their frequency, and consequences;
- integrity - the ability of the system to maintain the stability of the work and not have a risk.

Risk converts and reduces reliability properties, since the detected errors can lead to a threat if the failures are of a frequency nature.

Basic concepts in reliability issues of PS

Formally, the reliability assessment models for PS are based on reliability theory and mathematical apparatus with the assumption of some constraints affecting this estimate. The main source of information used in reliability models is the testing, operation of the PS and various kinds of situations that arise in them. Situations are generated by the occurrence of errors in the MS and require their elimination to continue testing.

Classification of models of reliability

As is known, at the given time a large number of reliability models for PS and their modifications have been developed. Each of these models defines a reliability function that can be calculated by assigning to it the relevant data collected during the operation of the MS. The main data are failures and time. Other additional parameters are associated with the type of MS, environmental conditions and data (fig. 11.3).

In view of the wide variety of reliability models, several approaches to the classification of these models have been developed. Such approaches are generally based on a history of errors in the tested and tested MS at the LC stages. One classification of software reliability models is the Hatch classification. It proposes the division of models into predictive, measurement and evaluation.

Predictive reliability models are based on measuring the technical characteristics of the created program: length, complexity, the number of cycles and the degree of their nesting, the number of errors on the page of program operators, etc.

For example, the Motley-Brooks model is based on the length and complexity of the program structure (number of branches, cycles, nesting cycles), the number and types of variables, and interfaces. In these models, the length of the program serves to predict the number of errors, for example, for 100 program operators, you can simulate the failure rate.

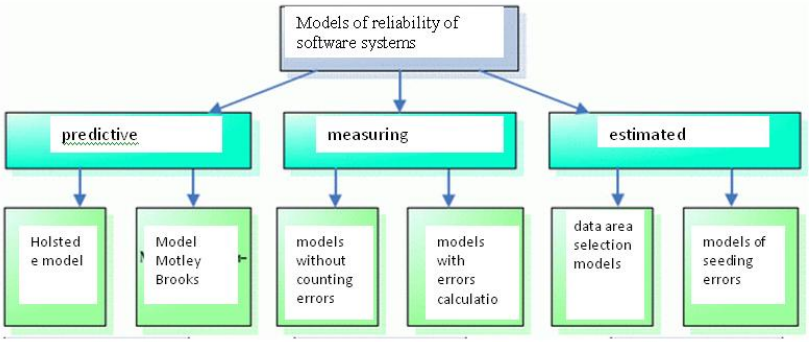


Fig. I.11.3. Models of reliability

Measuring models are designed to measure the reliability of software that works with a given external environment. They have the following limitations:

- the software is not modified during the reliability properties measurement period;
- the detected errors are not corrected;
- reliability measurement is performed for a fixed software configuration.

A typical example of such models are the Nelson and Ramamurti Bastani and others models. The Nelson reliability assessment model is based on the fulfillment of the k-runs of the program in testing and allows determining reliability. Thus, this model considers the quantitative data on the runs carried out.

Evaluation models are based on a series of test runs and are conducted during the testing phases of the PC. The test

environment is determined by the probability of failure of the program when it is executed or testing.

These types of models can be used in the LC stages. In addition, the results of predictive models can be used as inputs to the evaluation model.

Another kind of classification of models suggested by Goel, according to which the reliability models are based on failures and are divided into four classes of models:

- without counting of errors;
- with calculation of failures;
- with overseeding of errors;
- models with a choice of input ranges.

Models without error counting are based on measuring the time interval between failures and allow to predict the number of errors remaining in the program. After each failure, reliability is evaluated and the average time until the next failure is determined. Such models include the models of Jelinsky and Moranda, Shick Woolverton, and Linwood-Verrall.

Models with calculation of failures are based on the number of errors detected at specified intervals of time. The occurrence of failures as a function of time is a stochastic process with a continuous intensity, and the number of failures is a random value. The detected errors are usually eliminated and therefore the number of errors per unit time is reduced. This class of models includes Schumann, Schick-Woolverton, Poisson model, and others.

Models with overseeding error are based on the number of eliminated errors and overseeding made into an artificial error program, the type and number of which are known in advance. Then the ratio of the number of remaining predicted errors to the number of artificial errors is determined, which is compared with the ratio of the number of detected real errors to the number of artificial errors detected. The result of the

comparison is used to assess the reliability and quality of the program. When making changes to the program, repeated testing and reliability assessment are carried out. This approach to the organization of testing is cumbersome and is rarely used due to the additional amount of work involved in the selection, implementation, and removal of artificial errors.

Models with a choice of the input value range are based on generating a plurality of test samples from the input distribution, and reliability estimation are performed based on the received failures based on the test samples from the input area. This type of model includes the Nelson model and others.

I.12. ASSEMBLY, DOCUMENTATION AND MAINTENANCE OF SOFTWARE

Software documentation

In the course of working on a project to create any complex software system, a large number of *project documents* are created. Its main purpose is to coordinate the joint actions of a large number of developers for more or less long periods of time - during the initial development of the system, in the process of performing work on its modification, during the escort. The structure of project documentation in most projects is almost the same - these are the requirements for a system of different levels (system, functional and structural), a description of its architecture, program code, tests and documents accompanying the implementation process (installation guides, customization, user manuals).

Since the verification of the software system (in the optimal case) is performed throughout the development life cycle by a sufficiently large team of developers, test documentation is created during testing. Its main purpose, in addition to synchronizing the actions of testers at different levels, is to ensure that testing is performed in accordance with the selected criteria for assessing the quality and that all aspects

of the system's behavior are tested. Also, the test documentation is used when making changes to the system to verify that both the old and new functionality is working correctly (Figure 1).

Before the verification manager begins testing, a document is created, called a verification plan (or test plan, but this is not the same as a test-plan). The test plan is an organizational document that contains requirements for how testing should be performed in this particular project. It defines common approaches to the harmonization of development and verification processes, defines verification methods, the composition of the test documentation and its relationship with the developer documentation, the timing of the various verification stages, the various roles and qualifications of the testers required to perform all testing work, the requirements for testing tools and test stands, as well as assess risks and provide ways to overcome them.

This document also defines requirements for the test documentation itself - test requirements, test plans, test reports.

According to these requirements for the system and functional requirements, test developers create test requirements-documents that detail what aspects of the system's behavior should be tested. Based on the description of the architecture, low-level test requirements are created, which describe aspects of the behavior of a particular software implementation of the system that needs to be tested (fig. I.12.1).

Based on test requirements, test designers create test plans - documents that contain a detailed step-by-step description of how test requirements should be tested.

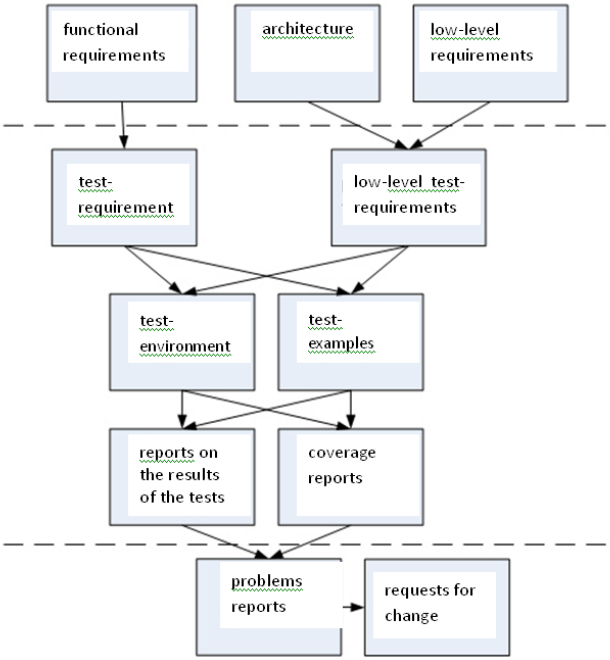


Fig. I.12.1. Documentation accompanying the verification process

Based on the test requirements and design documentation of the developers, a test environment is also created, which is necessary for correct execution of tests on test stands - drivers, stubs, setup files, etc.

As a result of the tests, testers create test reports (they can be created either automatically or manually) that contain information about what inconsistencies in the requirements were identified as a result of testing, as well as coverage reports, containing information about what percentage of the system's software code was involved as a result of testing.

By using nonconformities generate reports on problems - documents that are sent for analysis to the development team in order to determine the cause of the inconsistency.

Changes to the system are made only after a comprehensive study of these reports and the localization of problems that have caused a non-compliance with the requirements. To ensure that the process of change does not get out of hand and any change is recorded (and associated with the tests that detected the problem), a request is made to change the system. After all the work on the change request has been completed, the testing process is repeated until an acceptable level of software system quality is achieved.

The formats of various test documents are described in the IEEE 1012 and IEEE 829.

All documents must have unique identifiers and be stored in a single database of project documents. This will keep the controllability of the testing process and maintain the required quality of the system being developed.

Software maintenance

Software maintenance – is a set of actions to ensure the operation of the software, as well as to make changes in the event of errors in the process of operation, to adapt the software to the new operating environment, as well as to increase productivity or improve other characteristics of the software. Maintenance (in accordance with ISO / IEC 12207 and ISO / IEC 14764) is considered a modification of the software product during operation provided that the integrity of the product is maintained.

The "*Software maintenance*" knowledge area consists of the following sections:

- Basic Concepts,
- Process Maintenance,
- Issue in Software Maintenance,
- Techniques for Maintenance.

Maintenance is considered from the point of view of satisfying the requirements to the created software, the

correctness of its implementation, the learning processes and the operational tracking of the support process.

The basic concepts describe the basic definitions and terminology, approaches to the evolution and maintenance of software, as well as to assess the cost of maintenance, etc..

The main concepts can be referred to the JV PO (ISO / IEC 12207 standard) and documentation. The main purpose of this area of knowledge is to implement a ready-made software system, fix the errors that occur during the execution, investigate the causes of errors, analyze the need to modify the system in order to eliminate errors, estimate the cost of work to carry out changes in functions and the system as a whole. The problems associated with increasing the complexity of the product with a large number of changes and methods for overcoming it.

Software maintenance includes: models of the maintenance process and planning of the activities of people who are running the software, checking the correctness of its implementation and making changes to it. The accompanying process according to ISO / IEC 14764 is carried out by:

- adjustments, i. product changes to eliminate detected errors or unrealized tasks;
- adaptation, i.e. product settings in changed operating conditions or in a new environment for running this software;
- Improvements, i. evolutionary changes in the product to improve performance or maintenance level;
- software checks to find and fix errors found during system operation.

Key issues in software maintenance. The main of these issues are

- managerial,
- measuring,
- cost.

The essence of management issues is the control of software in the process of modification, improvement of functions and avoidance of system performance degradation. Measuring issues are related to the evaluation of the characteristics of the system after its modification, as well as re-testing and evaluation of quality indicators. Cost issues are related to the evaluation of software maintenance costs, depending on its type, staff qualifications, platform, etc. Knowing these factors often allows you to reduce costs.

Software evolution. A well-known software expert J. Lehman (1970) suggested that support should be considered as an evolutionary development of software systems since the system commissioned is not always complete; it needs to be changed during the life of the system. As a result, the software system becomes more complex and poorly managed; the problem of reducing its complexity arises. Technologies for software evolution include re-engineering, reverse engineering, and refactoring.

Reengineering is the improvement of legacy software through its reorganization or restructuring, as well as by reprogramming individual elements or adjusting parameters to another platform or execution environment while maintaining the convenience of its maintenance.

Reverse engineering consists of restoring the specification (call graphs, data streams, etc.) from the received system code to monitor it at a higher level. Identification of software components and connections between them is restored to ensure reprogramming of the system to a new form.

Most often, reverse engineering is used after many changes have been made to the software code and it has become unmanageable.

Refactoring is the reorganization of code to improve the characteristics and quality indicators of object-oriented and component programs without changing their behavior. This

process is realized by gradually changing individual operations on texts, interfaces, the programming and executing environment, and setting up or making changes to the software support tools. If the form of the existing system is preserved when changing, then refactoring is one of the variants of reverse engineering.

Software configuration management

Software Configuration Management (SCM) consists in the identification of the system components, determination of the functional and physical characteristics of the hardware and software for controlling the introduction of changes and tracing the configuration throughout the LC. This control corresponds to one of the auxiliary processes of the LC (ISO / IEC 12207), carried out by the technical and administrative management of the project; reports on the changes made to the configuration and the degree of their implementation are compiled, and the compliance of the changes made with the specified requirements.

System configuration - the composition of the functions, software, and hardware of the system, possible combinations of them, depending on the availability of equipment, system-wide tools identified in the technical documentation of the system, and product requirements.

The software configuration includes a set of functional and technical characteristics of the software, specified in the technical documentation and implemented in the finished product. This is a combination of different elements of the product together with the specified assembly and adjustment procedures for the environment in accordance with the purpose of the system. Examples of configuration items include the development schedule, project documentation, source and executable code, a component library, installation and deployment instructions.

The knowledge area "Software Configuration Management" consists of the following sections:

- Management of SCM Process,
- Software Configuration Identification,
- Software Configuration Control,
- Software Configuration Status Accounting,
- Software Configuration Auditing,
- Software Release Management and Delivery.

Configuration management. This is an activity to control the evolution and integrity of the product when identifying, monitoring changes and providing reporting information regarding the configuration.

Include:

- systematic tracking of changes to individual configuration components, performing an audit of changes and automated control over making changes to the configuration of the system or software;
- support for the integrity of the configuration, its audit and the provision of changes to configuration items;
- a configuration audit to verify the availability of the software or hardware developed and to reconcile the configuration version with the specified requirements;
- tracing changes to the configuration during the maintenance and operation phases of the software.

Identification of software configuration consists in documenting the functional and physical characteristics of software configuration elements, as well as in the design of technical documentation for software configuration items.

Software configuration monitoring consists of coordinating, approving or discarding implemented changes in configuration elements after formal identification, as well as in analyzing incoming components in the configuration and matching their identification.

Accounting of the status or software configuration status is carried out with the help of a set of measures to determine the degree of configuration change received from the developer, as well as the correctness of the changes made to the configuration of the software when it is accompanied. Information and quantitative indicators are accumulated in the relevant database and are used in configuration management, reporting, quality assessment and other processes.

Configuration audit is an activity that is performed to evaluate a product and processes for compliance with standards, instructions, plans, and procedures. Audit determines the degree to which the configuration element meets the specified functional and physical (hardware) characteristics of the system. Based on the functional and physical audit of the configuration, the baseline of the manufactured product.

Software version control is the tracking of an existing version of the configuration item; assembly of components; creating new versions of the system based on existing ones by making changes to the configuration; the coordination of the product version with the requirements and the changes made at the LC stages; providing quick access to information about the configuration elements and the system to which they relate. Release management covers the identification, packaging and transfer of product elements and documentation to the customer. The following basic concepts are used.

Baseline - formally designated set of software elements, fixed in the stages of the software center.

The software library is a controlled collection of software, and documentation objects designed to facilitate the development, use, and maintenance of software.

Software assembly – is the integration of the correct software elements and configuration data into a single executable program.

PART II
II.1. PRACTICAL METHODS OF WORK IN MICROSOFT PROJECT ENVIRONMENT
II.1.1. Creating and planning a project in Microsoft Project
Theoretical information

The Microsoft Office Project window consists of the following elements:

1. the line of the menu;
2. toolbars;
3. input line;
4. representations panel;
5. working area;
6. status line.

The line of the menu, **toolbars**, and status bars are standard for all Windows applications, and the way of working with them is the same as in Microsoft Office.

The input line is intended for input and editing of data into table cells (like the formula line in Excel).

The representations panel is used to switch between the views of the window's working area. All data about the project is stored in a single database consisting of a large number of fields. **A view** is a way to display a portion of the related data from a common project database. The system implemented a large number of views - Gantt chart, network graph, calendar, resource schedule, etc. If you want, you can change the standard views by adding or removing the data fields displayed on their tables. The first time you launch the program, the representations panel may be absent. To display it, select the View/Representations panel. Switch between views by clicking on the icon of the desired view.

The workspace is intended for displaying images. It can contain tables, charts, graphs, forms and used for both: viewing and editing project data. To create a new project,

select File / New. A blank project with a blank database will be created.

Features of Task Scheduling in the Microsoft Project System

There are several types of project work:

1. ordinary work (hereinafter referred as the word work or task);
2. milestone;
3. phase;
4. the total project task.

The work indicates some actions aimed at the execution of some part of the project. The milestone is a zero-length work. **Milestones** are designed to fix the control points in the project plan, in which important event management events take place. For example, the completion of one stage of work and the beginning of another. Usually, milestones are used to indicate the beginning and end of the project, as well as to indicate the end of each phase.

The Phase is a composite work consisting of several works and completing by a milestone. The phase describes a certain logically completed stage of the project and may consist of both: works and other phases.

The following rule is adopted in order to delimit work and phases in the system. All works are divided into levels that specify their hierarchy. Any work with subordinate work of the lower level is a phase. All other works are not phases.

The overall task of the project is an artificially created system of work, the duration of which is equal to the duration of the whole project. This work is used to compute, display and analyze aggregated project data.

Связь между задачами определяет, каким образом время начала или окончания одной задачи влияет на время окончания или начала другой. **The connection** between tasks

determines how the start or end time of one task affects the end time or the start of another.

There are four types of connections in Microsoft Project:

1. end-to-start;
2. start-to-start;
3. end-to-end;
4. start-to-end.

An **end-to-start** connection is the most common case of a link between works. With such a connection, work B can't start earlier than the work A.

A **start-to-start** connection means that work B can't begin until work A begins. With this connection, tasks that can be run in parallel are combined. For example, training the personnel to work with the program and entering data into the program can take place simultaneously, but data entry can't begin until staff training begins.

The **end-to-end** connection denotes the dependency at which task B can't end until task A is completed. Usually, this work combines work that is performed simultaneously, but one can’t end before the other. For example, commissioning a program and testing it and debugging can be done in parallel. During the commissioning process, staff training, preparation, and data entry take a place. However, commissioning can't be completed until the testing and correction of errors found in the program is completed.

A **start-to-end** connection denotes a relationship in which work B can't end until work A begins. For example, A is the commencement of a program in commercial operation, the beginning of which is scheduled for a strictly defined date. B - pilot operation of the program, which can't be completed until the program is put into commercial operation. Moreover, an increase in the duration of Problem A does not entail an increase in the duration of Problem B.

The list of tasks begins with the separation of the stages of the project. Each phase will correspond to a phase. If necessary, especially for large projects, the stages
can be divided into smaller stages. In this case, the phase will consist of smaller phases. When the list of stages is ready, a list of tasks to be performed at each stage is drawn up. As the last work of the stage, the problem of zero length, which corresponds to the milestone, is used.

Entering a list of project tasks is performed in any of the views that have a table for data entry. Best for this is the Gantt Chart, which, in addition to the table, shows the project's calendar schedule.

To enter a task, simply enter the name of the task in the column *Name of the task* in the empty row of the table. By default, the duration of a new task is taken equal to one day, and the start date of the task is the start date of the project. A question mark is displayed next to the duration value, which indicates that this duration value is preliminary and is set by the system. After the appointment of the duration of the user, the question mark disappears.

Creation of connections between tasks is carried out both: directly in the calendar schedule, and in the data entry table.

On the calendar chart, you should point the mouse on the task icon, click the left mouse button and, without releasing it, move the pointer to the icon of another task, and then release the mouse. A link will be established between them.

Binding tasks in the data entry table is performed using the *Predecessor* column, which contains the numbers of the immediately preceding tasks, separated by a semicolon.

The task duration can be assigned in two ways::

1. change the value in the *Duration* column of the data entry table;

2. double-click on the task line to open the *Task Information window* and on the *General tab*, set the duration value..

By default, the duration is set in days. However, you can change the unit of measure by specifying it next to the numeric value. For example, 10d means 10 days, 10h - 10 hours, 10m - 10 minutes, 10month - 10 months.

Performance of work

1. Let's start drawing up a plan for our project to add the first task, which will be a milestone, for this, we set the task duration of 0 days. Such a task is by default a milestone. You can also assign any task to a milestone through the details menu in the *Advanced tab*.

2. Then add stages that will summarize the amount of work.

Fig II.1.1. Stages of work

3. We will add the work that will be included in these stages: using the insert key add new tasks after the training phase, then we select all the added tasks and make them subtasks of the preparation stage with the appropriate button in the *Task tab*.

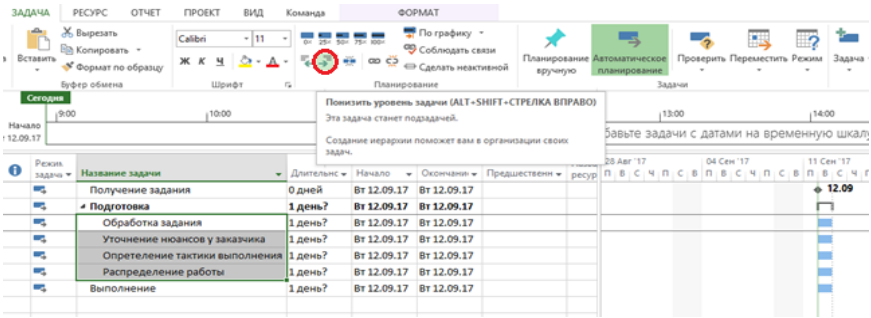


Fig II.1.2. Tasks and subtasks

4. We also add subtask to the stage of execution, in it we will mark the milestone of the beginning of work, and after this the milestone stage of the ordering

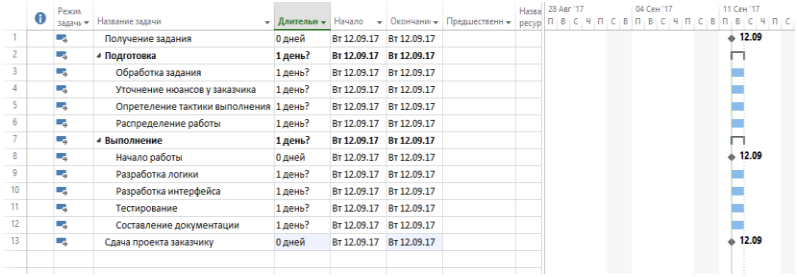


Fig II.1.3. Stages, tasks and subtasks

5. Now we can associate completion of tasks with the beginning of the following

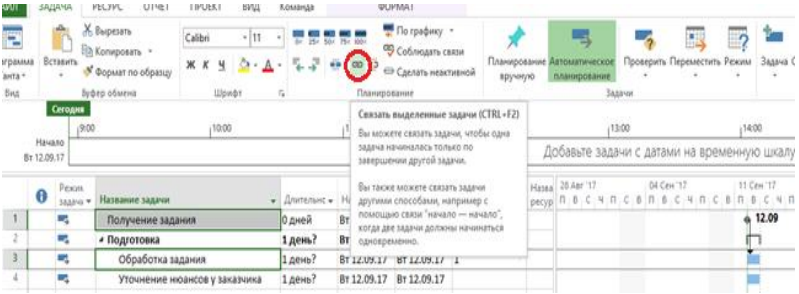


Fig. II.1.4 Binding tasks

6. It is clear that in the preparation stage all subtasks go one by one, so they can be highlighted and click on the same button, and they will contact each other

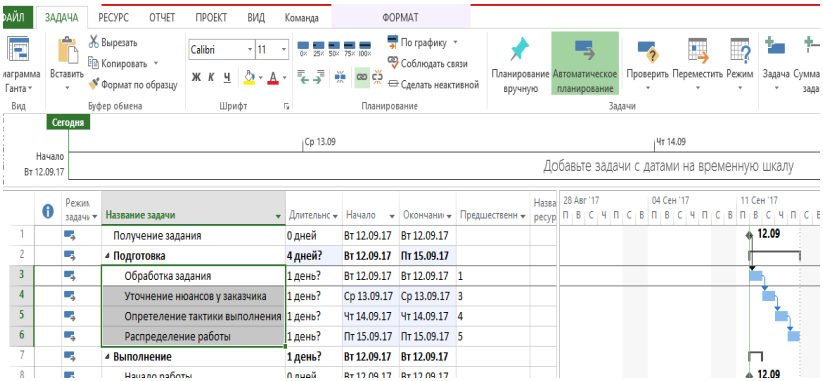


Fig. II.1.5 Binding tasks (continued)

7. Let's do the same with the execution of the work, but there we can combine the development of logic and interface, for this, with a double click, click on the development of the interface, and in the tabs of predecessors, we will change the type of the previous logic from "end-to-start" to "start-to-start", and in testing will add in the predecessors the development of logic with the type of "end-start"

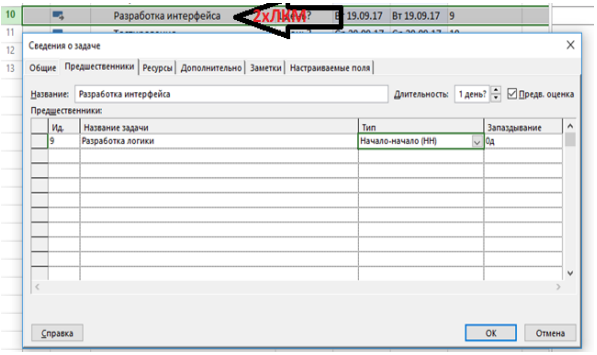


Fig. II.1.6 Parallel tasks

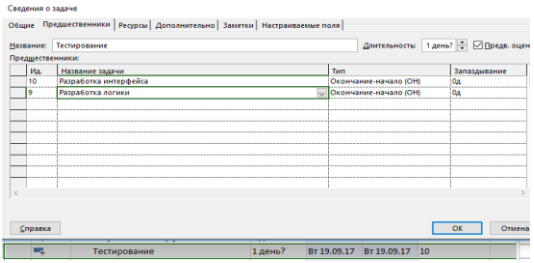


Fig. II.1.6a. Parallel tasks (continued)

8. After these operations planning will look like this:

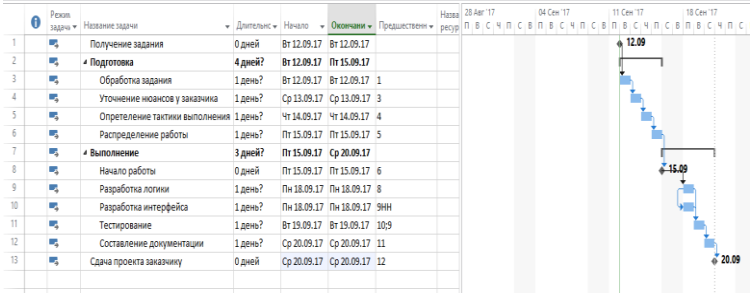


Fig. II.1.7 The result of planning

9. We set the time for each task and include the total project task in a tab format that will show the total cost of our entire project.

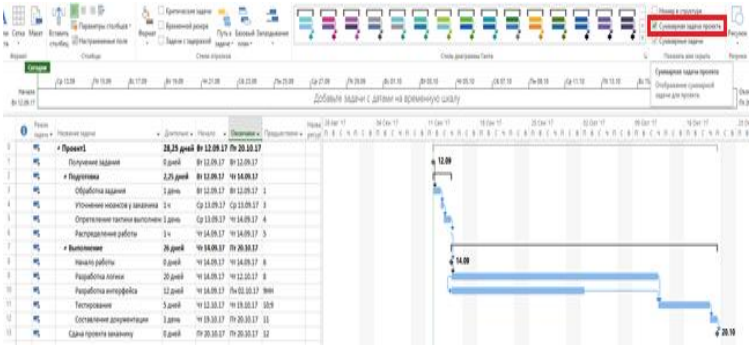


Fig. II.1.8. Determining the time of the tasks

10. Let's turn to Gantt charts with tracking, in which we can see the critical path of our project.

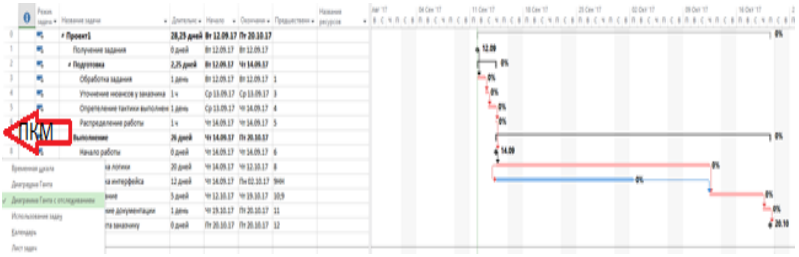
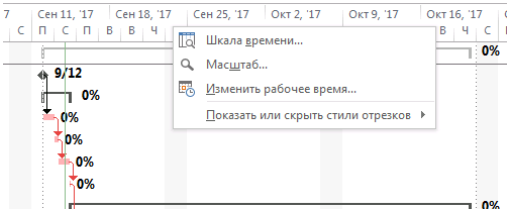


Fig. II.1.9. Gantt Chart with Tracking

11. Change the scale on the timeline, click on the calendar, select "Zoom" and select "All project" in it.



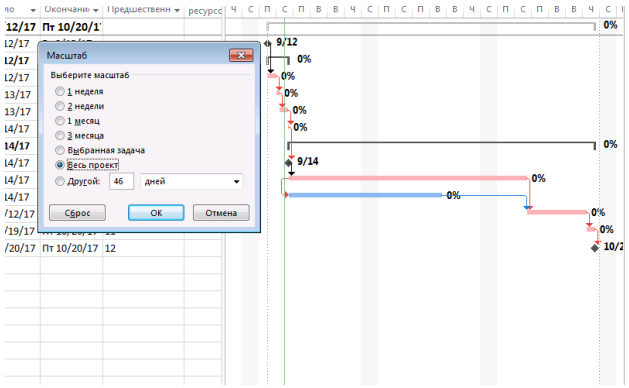


Fig. II.1.10. Calendar on Gantt chart with tracking

12. Add the executors (employees) the necessary resources (material) and expenses (finance), we will go to the tab resources:

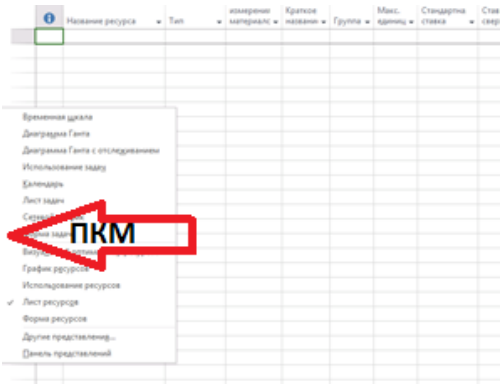


Fig. II.1.11. Add resources to the project

We can choose one of the three types of resources for each item:

- Labor - executors (people who work and receive salary);
- Material - resources required to perform work (materials, etc.);
- Costs - cash expenses.

	Название ресурса	Тип	Краткое названи	Группа	Макс. единиц	Стандартна ставка	Ставка сверхурочн	Начисление	Базовый календарь
1	Фронт-энд	Трудовой	Ф		100%	200.00 ₴/ч	0.00 ₴/ч	Пропорциональ	Стандартный
2	Бэк-энд	Трудовой	Б		100%	400.00 ₴/ч	0.00 ₴/ч	Пропорциональ	Стандартный
3	Дизайнер	Трудовой	Д		100%	200.00 ₴/ч	0.00 ₴/ч	Пропорциональ	Стандартный
4	Руководитель	Трудовой	Р		100%	500.00 ₴/ч	0.00 ₴/ч	Пропорциональ	Стандартный
5	Тестировщик	Трудовой	Т		100%	150.00 ₴/ч	0.00 ₴/ч	Пропорциональ	Стандартный
6	бумага	Материальн	лист	б		0.10 ₴		Пропорциональ	
7	кофе	Материальн	чашка	к		10.00 ₴		Пропорциональ	

Fig. II.1.12. Project resources

We also specified units of measurement for mat.resources, their cost and salary of employees.

13. Next, we distribute resources by assignments. Returning to the Gantt chart, select the Resource tab, then Assign Resources.

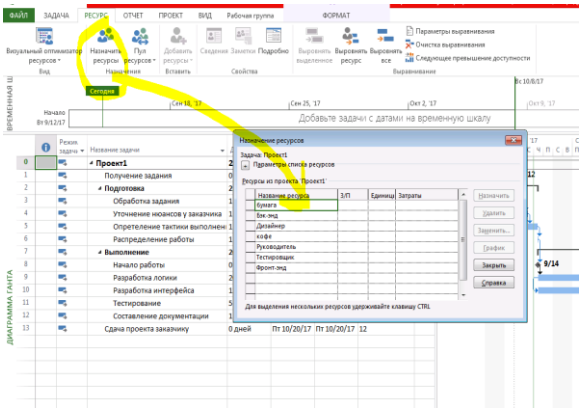


Fig. II.1.13. Distribution of resources by assignments

In milestones, we appoint only (!) workers who start work.

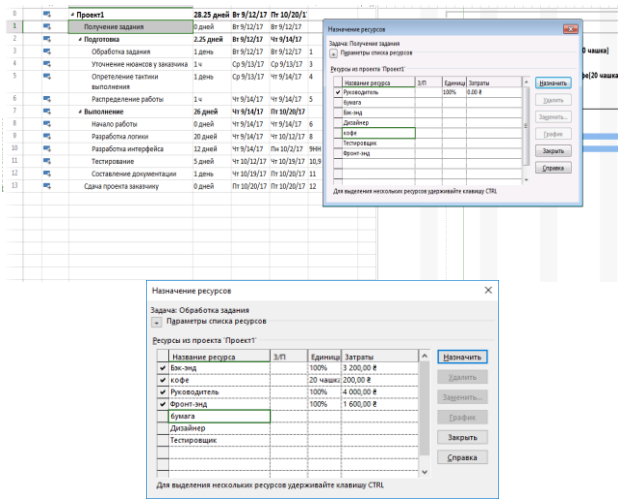
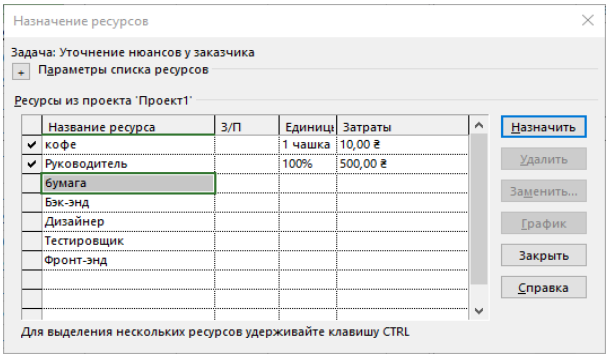
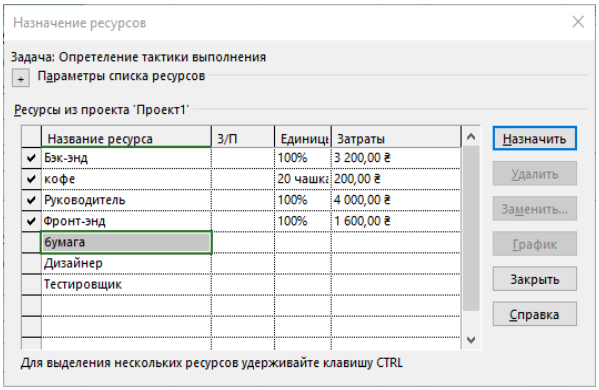


Fig. II.1.14. Distribution of resources for tasks and subtasks





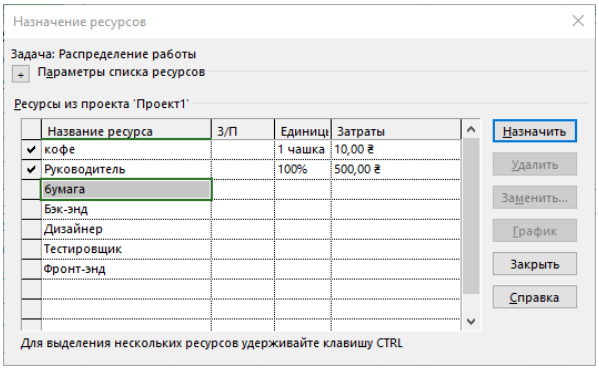


Fig. II.1.14a. Distribution of resources for tasks and subtasks (continued)

II.1.2. Risk management in Microsoft Project

Performance of work

Part 1: Introduction and identification of risks

1. Open the outline of your project and go to the **Gantt Chart** view.

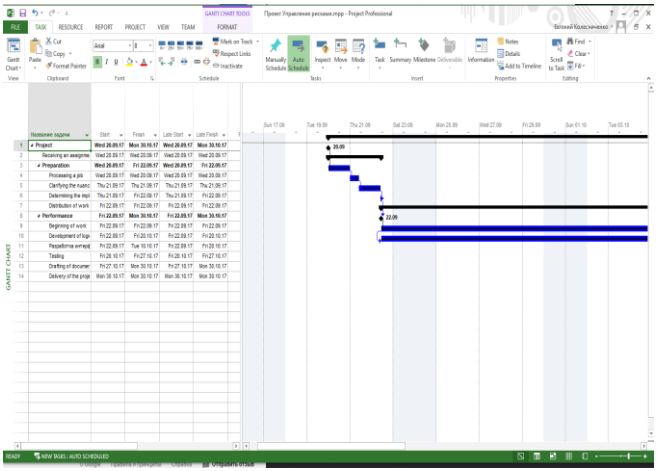


Fig. II.1.15. Gantt Chart view

2. Next, we need to create a table for analysis and risk management. Right-click on the intersection of the row and column header and select **More Tables** from the shortcut menu.

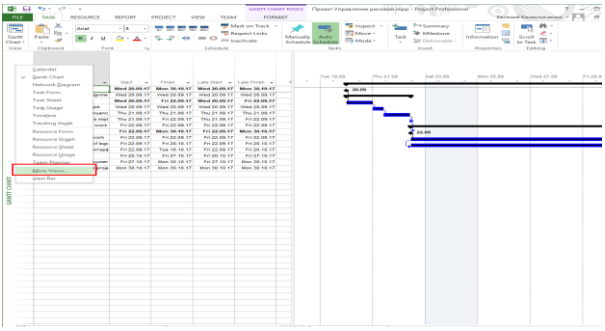


Fig. II.1.16. Gantt Chart. More Tables

3. We will not use the template, create a new table, click the **New** button.

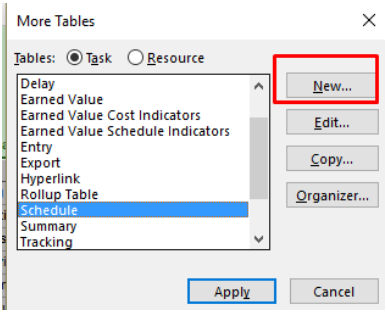


Fig. II.1.17. Creation a new table

4. Name the **Risk Analysis** table and add the following fields: *ID*. (task sequence number), *Name* (to understand which task the risks are), and the *Text1* field (to identify opportunities) and the *Text2* field (for threat identification). Do not forget to include the **Show in menu** option to make it easier to navigate to this table.

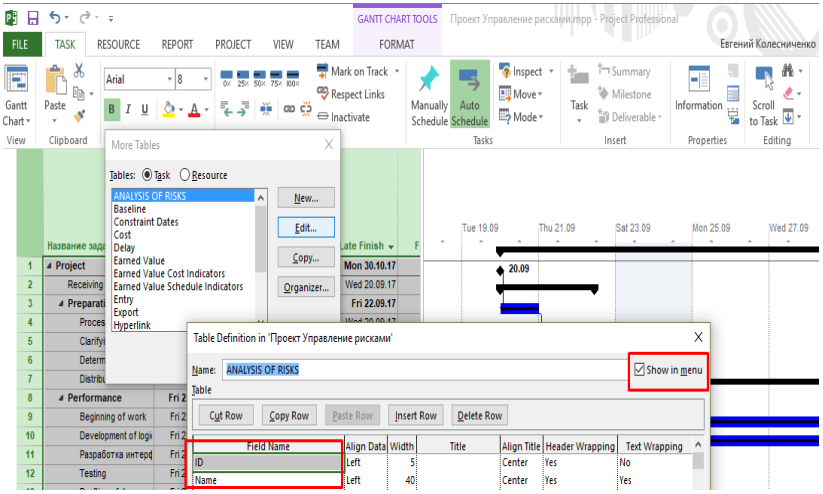


Fig. II.1.18. Risk Analysis table

5. Click **OK** and click **Apply** again to go directly to our created table. Adjust the width of the columns so that you can work with it. In this table you can already perform the identification of Opportunities and Threats.

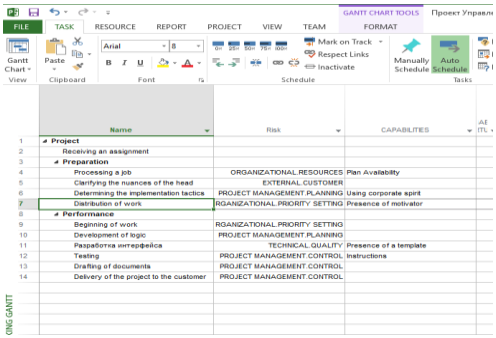


Fig. II.1.19. Risk Analysis table

6. In order not to introduce new opportunities and threats each time, these two fields will be supplemented with certain parameters that will allow us to re-use the already identified opportunities and risks. To do this, move the mouse cursor over the header of the **Features** column, right-click and select **Custom Fields** in the context menu.

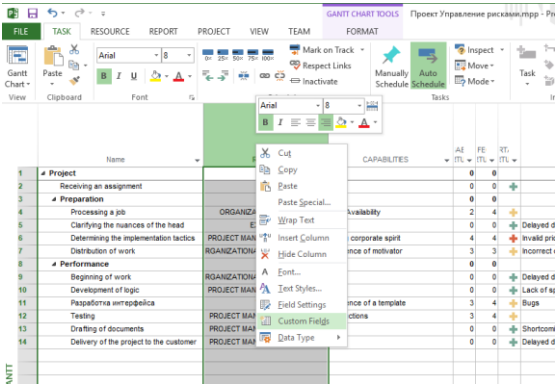


Fig. II.1.20. Custom Fields

7. Before you open the **Custom Fields** dialog box. Rename the ***Text1*** field to *CAPPABILITIES*, and the Text2 field in *THREATS*. Then, for the *CAPABILITY* field, enable the Substitution option, do the same for the *THREATS* field.

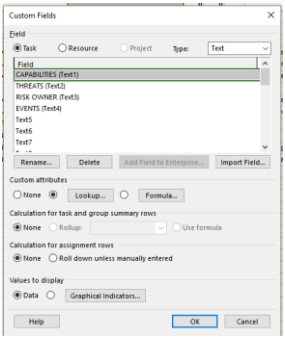


Fig. II.1.21. Threats Fields

8. Then, in succession, click the **Lookup** button for the *CAPABILITY* field and for the *THREATS* field. In the dialog that appears, **expand Display order for the lookup table** and switch to **Ascending**. That is, all the input values will automatically be sorted in ascending order.

9. Then expand the **Input Options** and enable the option to allow the addition of additional elements to the fields. That is, all the opportunities or threats that we enter into these fields will be automatically added to the list, sorted in ascending order and will be available for reuse.

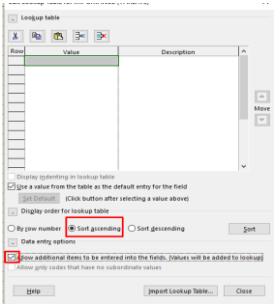


Fig. II.1.22. Threats Fields

10. Close the **Custom Fields** dialog box and return to the project. Now we can identify the opportunities and threats that will be automatically added to the list. For example, for task *processing task*, we identify the possibility - Having **a job processing plan** will shorten preparation time. And for the task of **Developing a project interface** as a threat identify the **lack of a specialist with the proper qualifications** can lead to incorrect evaluation of the project.

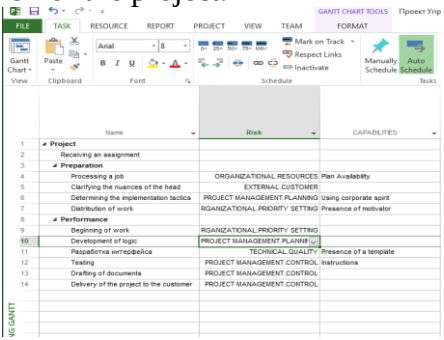


Fig. II.1.23. Identification of the opportunities

Part 2: Classification of risks

1. In our practical work, I will use the standard risk structure. So, move the cursor to the intersection of the row and column header and select **More tables** from the context menu.

2. Add the *Code field of the directory1. (Structure code 1 in older versions of Office)* and click **Ok** to apply and add a new column.

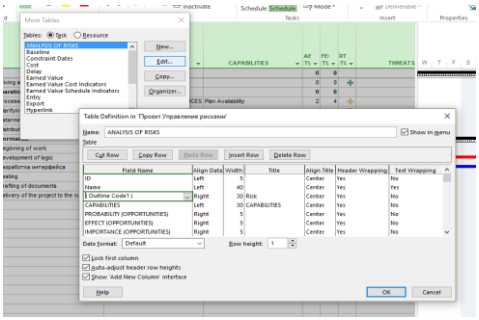


Fig. II.1.24. Identification of the opportunities

3. On the heading of the *Outline field1* box, right-click and select **Custom Fields**. In the dialog that opens, first of all, rename the Outline Encoding field1 to the *SDRISK* (structural risk decomposition). And then click the **Lookup** button.

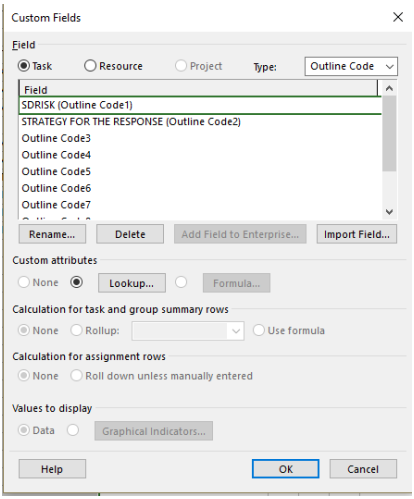


Fig. II.1.25. Custom Fields

4. In the **Lookup** dialog, first open the + (plus) icon for the Code **Mask** value and click the **Edit Mask** button. This is necessary to add another layer of the structure

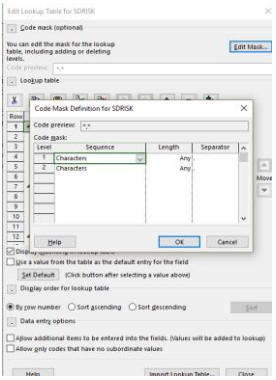


Fig. II.1.26. Edit Mask

5. Close this dialog box and enter our structural decomposition of the risks. Do not forget to use the structure buttons, in this dialog they are called, respectively, the **Lead** and **Indent**.

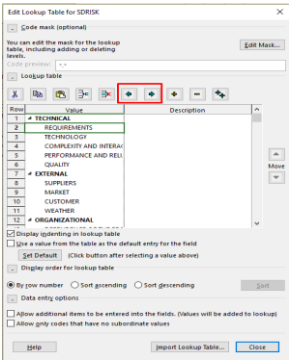


Fig. II.1.27. Lead and Indent

6. You can close this dialog box and the **Custom Fields** dialog box. Now we are ready to classify all of our identified risks (OPPORTUNITIES and THREATS).

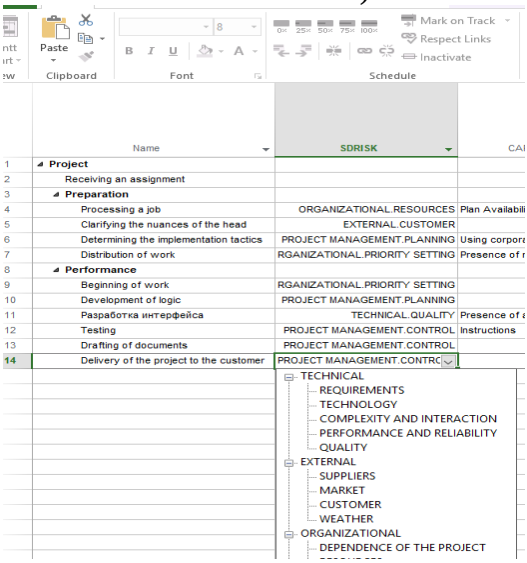


Fig. II.1.28. Opportunities and Threats

7. In my example, two risks were identified. Opportunity - Having a plan ... and a threat Lack of a specialist ... Both risks are related to organizational risks associated with resources.

8. Further in this table we need to add one more column for the RISK OWNER. To do this, move the cursor to the intersection of the row and table header, right-click and select **Other tables**.

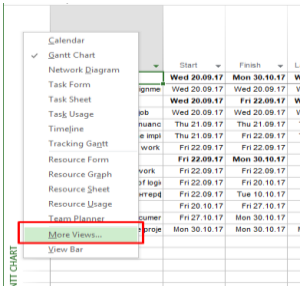


Fig. II.1.29. Other tables

9. In the dialog box, select the RISK ANALYSIS table and click the **Change** button. Next, in the dialog box that opens, immediately after the **THREATS** line, add the Text3 field and define the title RISK OWNER.

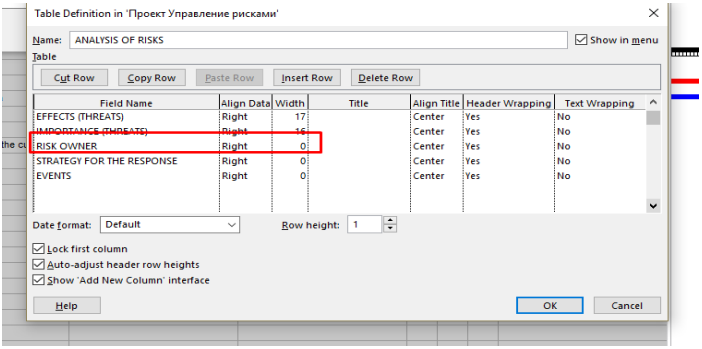


Fig. II.1.30. Risk owner

10. Click **OK** and immediately apply this table. In this field you can manually enter the risk owners, but we will make a reference with universal roles.

11. In order to make a directory with a list of risk owners, simply move the mouse cursor over the column header of the RISK OWNER and select the **Custom Fields** command. In the opened window, replace the name of this field with the RISK OWNER and click the **Lookup** button and enter the following values: Corporate management, Organization management, Senior user, Senior supplier, Project manager, Internal project control and Project support.

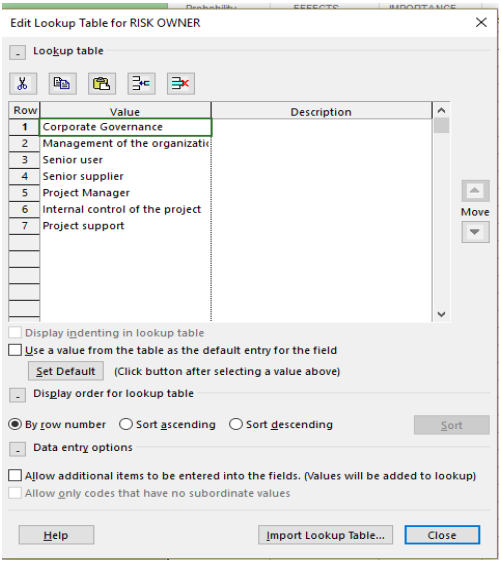


Fig. II.1.31. Custom Fields

12. After that, you can close this window and click **OK** twice to return to the view. For the task of processing the project task by the risk owner, I will assign the Project Support, and for the task Developing an Interface ... Management of the organization.

Part 3: Qualitative Risk Analysis

1. Use the Risk Analysis table. Then move the cursor to the header of any column, right-click and select the Custom Fields command in the context menu. Select the type of the Number field. And then sequentially rename the fields: Number1 - PROBABILITY (POSSIBILITIES), Number 2 - INFLUENCE (POSSIBILITIES), Number3 - IMPORTANCE (CAPABILITIES), Number4 - PROBABILITY (THREATS), Number5 - EFFECTS (THREATS) and Number6 IMPORTANCE (THREATS).

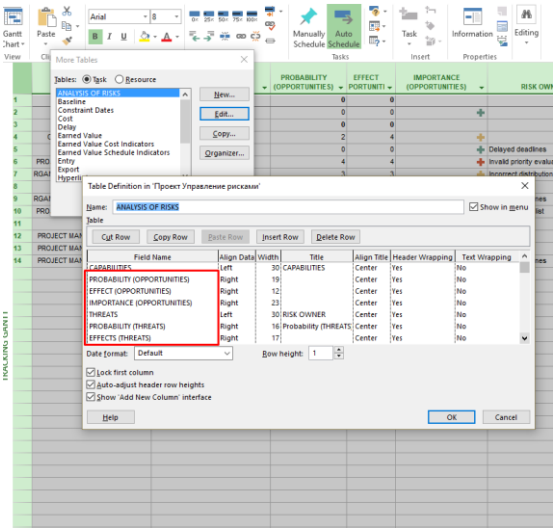


Fig. II.1.32. Rename Custom Fields

2. Now select the **PROBABILITY (OPPORTUNITY)** box, click the **Lookup** button and enter the following values: 1 - VERY LOW; 2 - LOW; 3 - AVERAGE; 4 - HIGH and 5 - VERY HIGH.

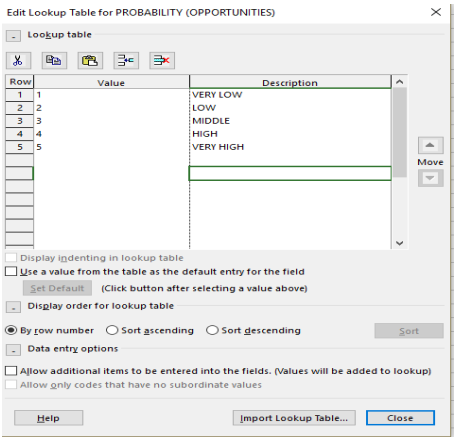


Fig. II.1.33. Probability (Opportunity) box

3. Do the same for the EFFECT (OPPORTUNITY) field.

4. Go to the IMPORTANCE (OPPORTUNITIES) box and click the **Formula** button. In the window that opens, enter the formula [PROBABILITY (OPPORTUNITY)] * [EFFECT (OPPORTUNITY)].

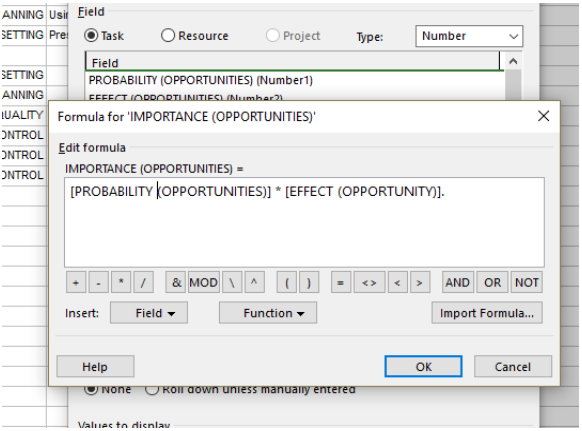


Fig. II.1.34. Importance (Opportunity) box

5. Now, for the IMPORTANCE (OPPORTUNITIES) field, configure the graphic indicators. To do this, click the Graphic Indicators button and define it as follows:

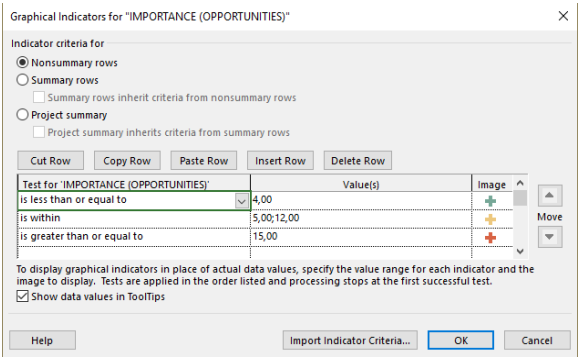


Fig. II.1.35. Graphic indicators

6. Note that if you choose to check the field from the drop-down list, then the checks "within" are not, nevertheless, we use this check, since it works, as opposed to checking "inside".

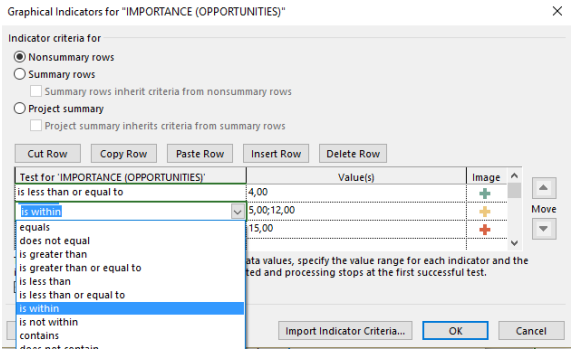


Fig. II.1.36. Graphic indicators (continued)

7. Now, in order not to enter values for fields PROBABILITY (THREATS), EFFECT (THREATS), I simply copy them from the corresponding fields of possibilities. To do

this, highlight the PROBABILITY (THREATS) field and click the Import Field button. Then select the appropriate field. Repeat for the EFFECT (THREATS) field.

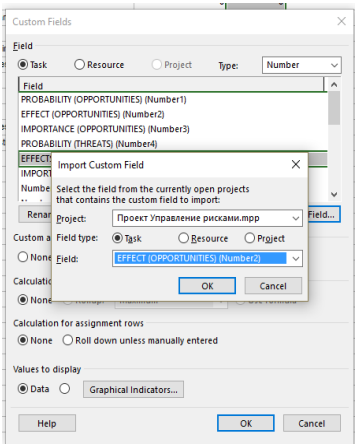


Fig. II.1.37. Effect (Threats) field

8. Now select the IMPORTANCE (THREATS) field and enter the formula [PROBABILITY (THREATS)] * [EFFECT (THREATS)].

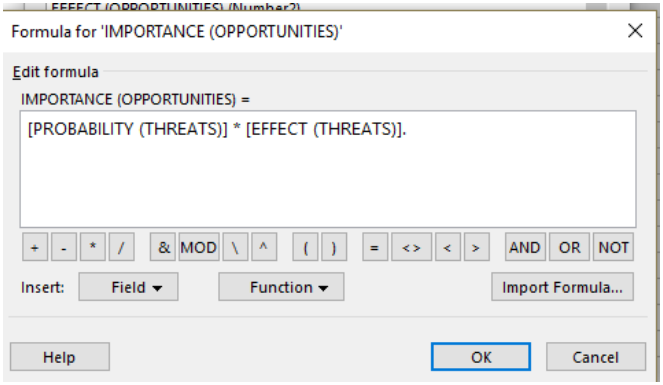


Fig. II.1.38. Formula

9. Next, configure the graphic indicators for this field. We also use import conditions from the field IMPORTANCE (OPPORTUNITIES).

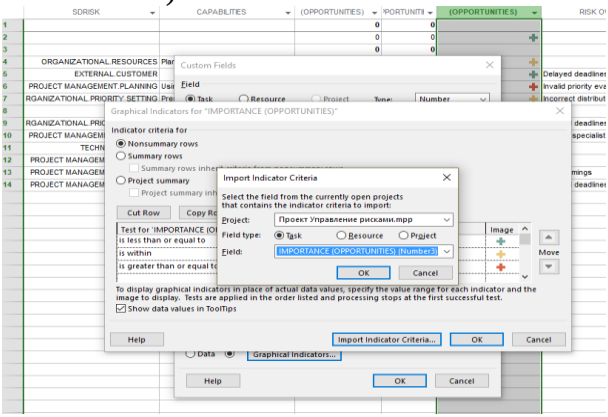


Fig. II.1.39. Use import conditions

10. But if they are threats, then the graphical indicators will be a minus sign

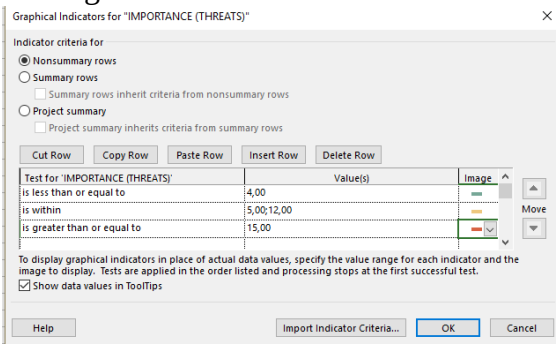


Fig. II.1.40. Minus as a graphical indicators

11. We have to adjust the table, and we will be ready to perform a qualitative risk assessment. Close the **Custom Fields** dialog box. Then move the mouse cursor to the upper right corner of the table, right-click and select **More tables** in the context menu.

12. In the window that opens, select RISK ANALYSIS and click the Change button.

13. In the Definition of Table ... dialog box, highlight the THREATS line and click. Add a row ... and add the PROBABILITY (OPPORTUNITY), EFFECT (OPPORTUNITIES), and IMPORTANCE (OPPORTUNITIES) fields sequentially.

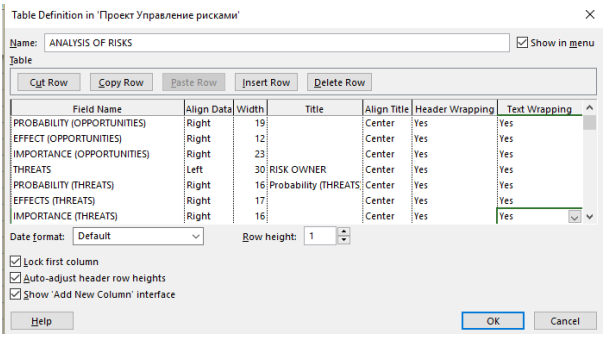


Fig. II.1.41. Probability (Opportunity), Effect (Opportunities), and Importance (Opportunities)

14. Now select the row RISK OWNER and add the PROBABILITY (THREATS), EFFECT (THREATS) and IMPORTANCE (THREATS) fields.

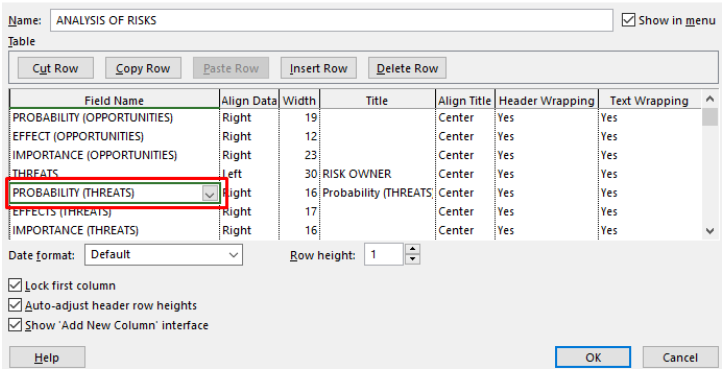


Fig. II.1.42. Probability (Threats), Effect (Threats) and Importance (Threats)

15. Click **OK** to close this dialog and apply the table. We are ready to carry out a qualitative risk analysis of our project. Note that we have not done a risk analysis, however, the plus sign or minus sign is already displayed in the IMPORTANCE column for opportunities and threats. The fact is that when we determined the conditions for the graphic indicators, we indicated "less than or equal to 4".

16. Do an assessment of the likelihood and impact for those opportunities and threats that we identified with you. Probability we will evaluate as "average", but in real life you must clearly understand the context of your company, so the probability estimate will be more accurate. Influence for the possibility we will point out as "high", and for the threat of "very high".

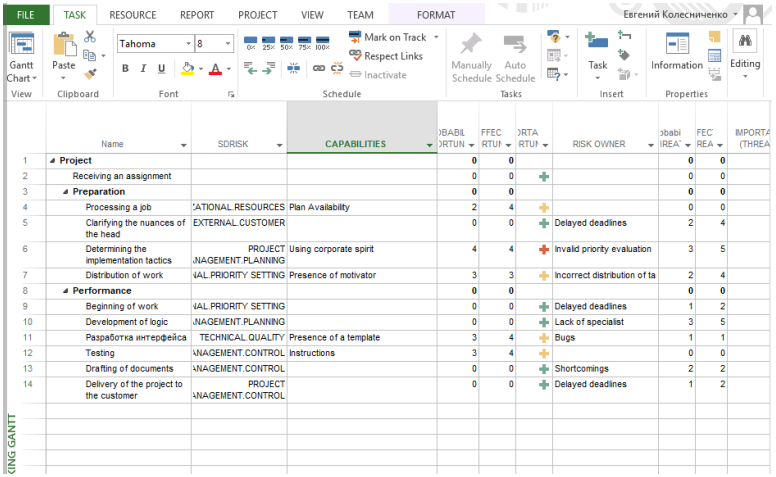


Fig. II.1.43. Evaluation of Probability and Importance

17. In conclusion of the qualitative risk analysis, you can identify the root causes of the risks and perform the ranking of risks by priorities.

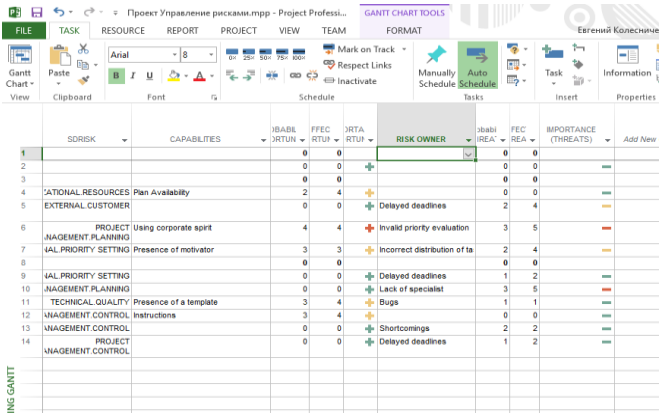


Fig. II.1.44. Ranking of risks by priorities

Part 4: Planning responses to risks

1. Open the **More Tables** dialog box.

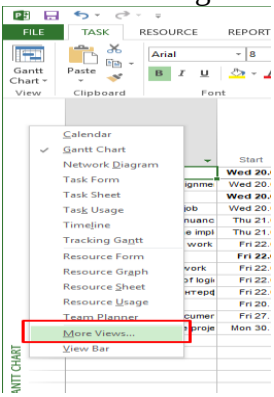


Fig. II.1.45. More Tables

2. In the Other Tables dialog box that appears, click Edit.

3. In the Define a table in the project ... dialog box, go to the very last line and add two fields: Directory2 encoding and Text4. These fields are still free and we can add the necessary information to it.

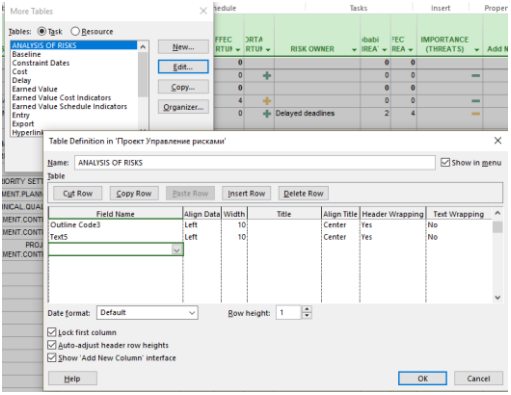


Fig. II.1.46. Add the necessary information

4. Click OK to close this dialog box and click. Apply to return to this table. You see that we have two additional fields that do not yet contain any information. In addition, the field headers are impersonal in nature - Directory encoding2 and Text4. To configure these fields, right-click the heading for the Directory Encoding field2 and select the Custom Fields command.

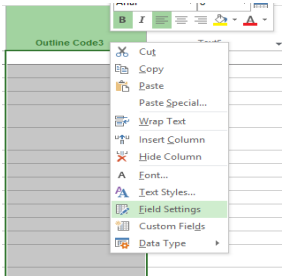


Fig. II.1.47. Custom Fields

5. In **the Custom** Fields window that opens, select the Directory Encoding *field2* and click **Rename**. Enter the name of the STRATEGY OF THE RESPONCE. Then click the Substitution button and in the window that opens, enter the following data. When entering data, do not forget to add a

second level in the Code Mask. After entering the data, click Close.

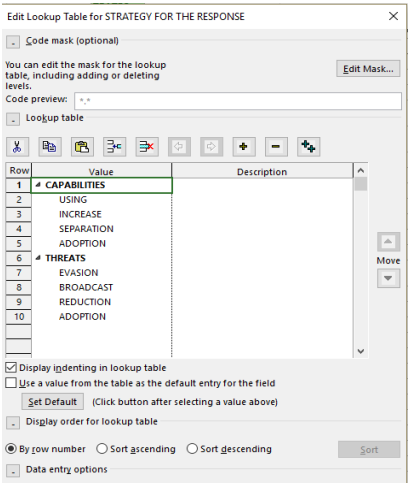


Fig. II.1.48. Strategy of the Responce

6. In the Custom Fields dialog box, go to the Text field, and then click Rename and type the name of the activity. In this field, we will keep a brief description of the risk management activities.

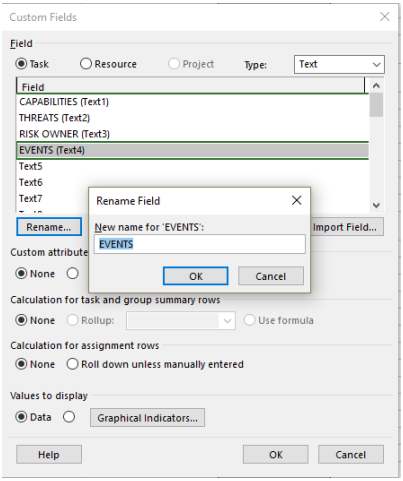


Fig. II.1.49. Name of the Activity

7. Click **OK** two times to complete the risk analysis. In our table, choose a strategy for responding to risks (opportunities and threats) and make a brief description of the measures to manage these risks

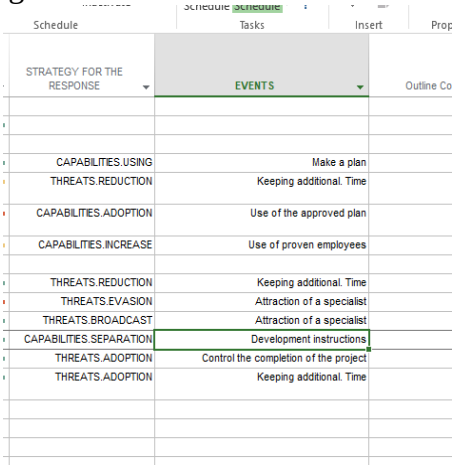


Fig. II.1.50. Brief description of the measures

II.1.3. Visual reports in MS Project

Performance of work

Reports in MS Project are for creating reports in the form of graphs, charts, tables, etc. in the Project within the familiar Office infrastructure of your project data to analyze your project and then share the results with others. There are many ready-made reports, but you can also edit and customize them.

1. Let's create the first report, for this we will use the standard template. Select the tab reports, dashboards, BURNDOWN.

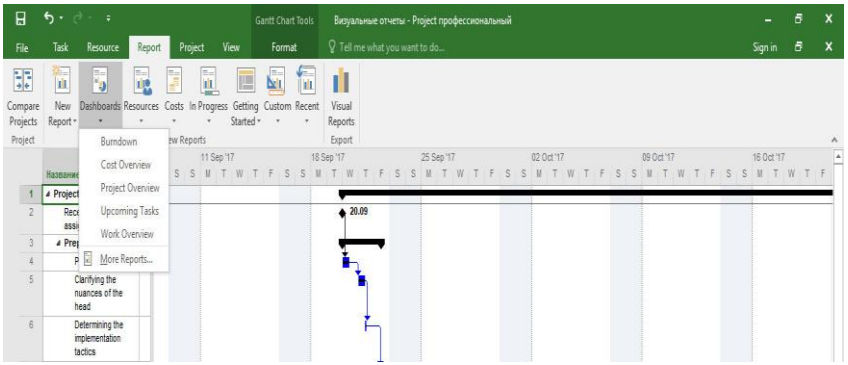


Fig. II.1.51. Reports, Dashboards, Burndown

2. Burndown shows which part of the work is completed and how much is left. If the line of the remaining total labor is steeper, it is possible that the project will not be completed on time. That is, the orange line shows how it should be in an idyllic scenario, but as we see on the blue line, which shows the current state of our remaining labor, our resources are rapidly ending and the project may not be completed on time.

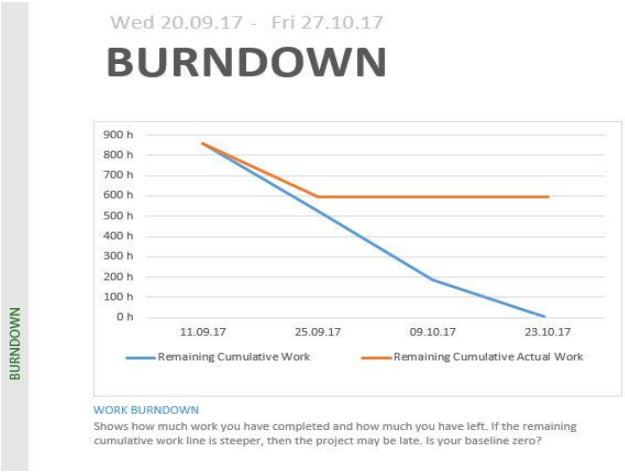


Fig. II.1.52. Report, Burndown

3. Further we choose the following offered standard report COST OVERVIEW

In the left part we see the amount: how much money was allocated for the project, how much has been spent at this stage of implementation in 40%, the funds that remained and the costs in excess of the norm for one reason or another.

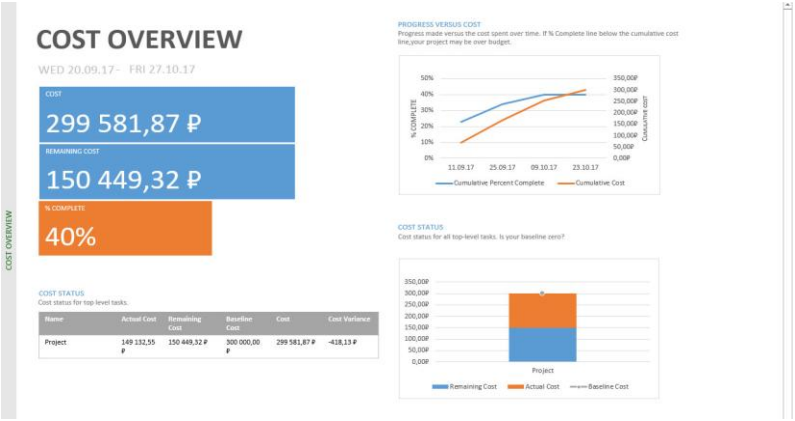


Fig. II.1.53. Cost overview

Also, pay attention to the table in our project, which corresponds to these charts, it is obvious that in the first variant everything is much simpler and more understandable.

ИСПОЛЬЗОВАНИЕ ЗАДАЧ	3	**Preparation**	**0,00 ₽**	**Пропорциональное**	**1 000,00 ₽**	**0,00 ₽**	**1 000,00 ₽**	**1 000,00 ₽**	**0,00 ₽**
	4	Processing a job	1 000,00 ₽	Пропорциональное	1 000,00 ₽	700,00 ₽	300,00 ₽	1 000,00 ₽	0,00 ₽
	5	Clarifying the nuanc	0,00 ₽	Пропорциональное	0,00 ₽	0,00 ₽	0,00 ₽	0,00 ₽	0,00 ₽
	6	Determining the impl	0,00 ₽	Пропорциональное	0,00 ₽	0,00 ₽	0,00 ₽	0,00 ₽	0,00 ₽
	7	Distribution of work	0,00 ₽	Пропорциональное	0,00 ₽	0,00 ₽	0,00 ₽	0,00 ₽	0,00 ₽
	8	**Performance**	**0,00 ₽**	**Пропорциональное**	**66 010,00 ₽**	**0,00 ₽**	**66 010,00 ₽**	**60 601,00 ₽**	**5 409,00 ₽**
	9	Beginning of work	0,00 ₽	Пропорциональное	0,00 ₽	0,00 ₽	0,00 ₽	0,00 ₽	0,00 ₽
	10	Development of logi	10 000,00 ₽	Пропорциональное	10 000,00 ₽	6 000,00 ₽	4 000,00 ₽	10 000,00 ₽	0,00 ₽
	11	Разработка интерф	50 000,00 ₽	Пропорциональное	50 000,00 ₽	45 000,00 ₽	5 000,00 ₽	50 000,00 ₽	0,00 ₽
	12	Testing	0,00 ₽	Пропорциональное	6 010,00 ₽	5 000,00 ₽	1 010,00 ₽	601,00 ₽	5 409,00 ₽
		Tester			*6 000,00 ₽*	*0,00 ₽*	*6 000,00 ₽*	*600,00 ₽*	*5 400,00 ₽*
		coffee			*10,00 ₽*	*0,00 ₽*	*10,00 ₽*	*1,00 ₽*	*9,00 ₽*
	13	Drafting of documer	0,00 ₽	Пропорциональное	0,00 ₽	0,00 ₽	0,00 ₽	0,00 ₽	0,00 ₽
	14	Delivery of the proje	0,00 ₽	Пропорциональное	0,00 ₽	0,00 ₽	0,00 ₽	0,00 ₽	0,00 ₽

Fig. II.1.54. Cost overview table

In the lower right corner is also a graph that graphically shows the above amounts.

4. But on the schedule, the PROGRESS VERSUS COST should be paid attention.

Progress made versus the cost spent over time. If % Complete line below the cumulative cost line, you project may be over budget.

This is easy to understand, since we see that the project is completed at 40% and our cost line exceeds this number, and already at around 42-43%.

Fig. II.1.55. Progress versus Cost

5. The next report will show us the PROJECT OVERVIEW

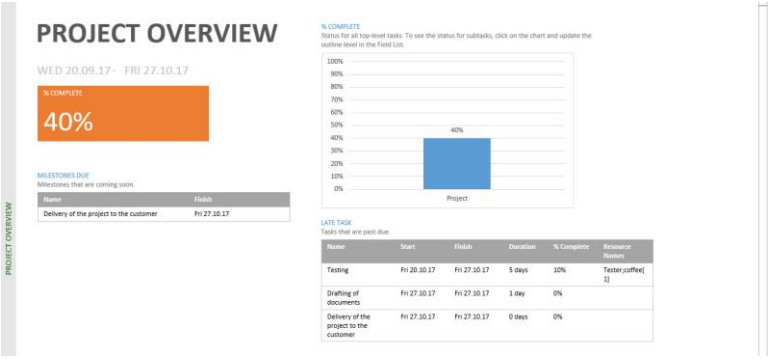


Fig. II.1.56. Project overview

Here we can see the overall percentage of completion of the project, and the time when the project should be completed.

Here we can include a report on UPCOMING TASK

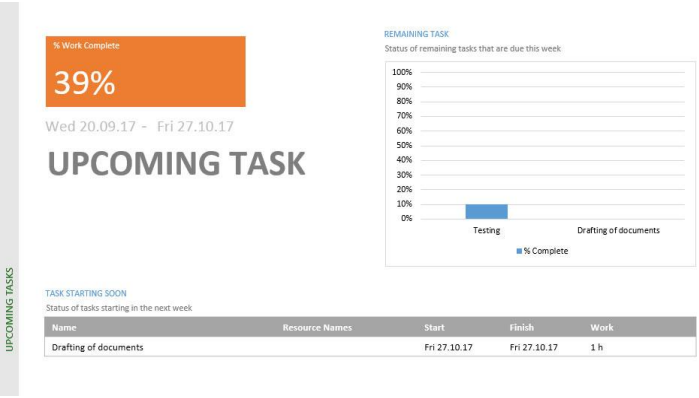


Fig. II.1.57. Upcoming Task

This report shows the current task, the percentage of its implementation and the forthcoming next task, in this case, after the completion of testing, it will be necessary to draw up documents for delivery of the project to the customer.

5. Now go to the section Reports, Costs, select the CASH FLOW

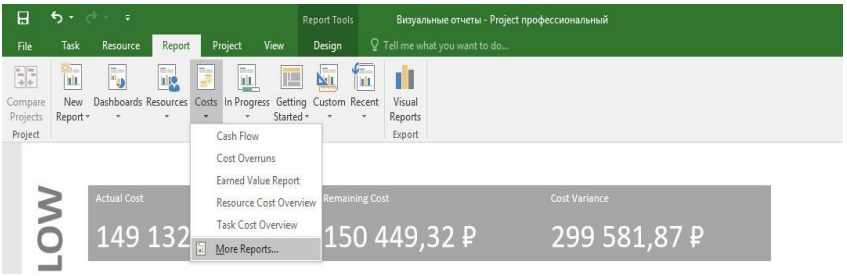


Fig. II.1.58. Cash Flow

Let's look at the example of this report and see how we can edit the finished schedule.

Click 2 times on the graph that you want to change, on the right appears the editing menu.

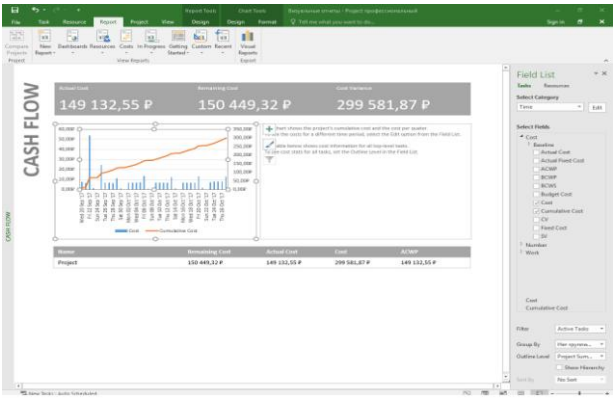


Fig. II.1.59. Editing Menu

Let's change the time period, click Edit, change the units from days to weeks.

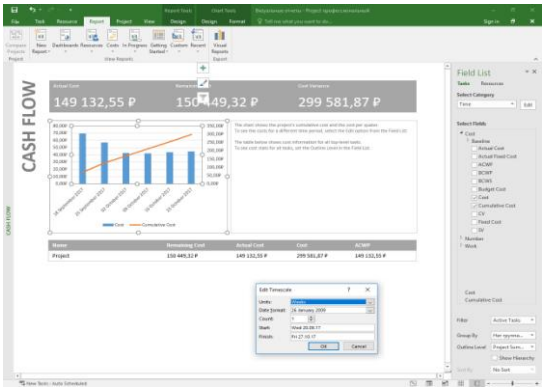


Fig. II.1.60. Changing the time period

This graph shows the total costs (all project costs for this period), and quarterly costs (in this case, selected for weeks from September 18 to October 23), it turns out that a huge amount of information is placed on one compact schedule.

6. Further we can look at the TASK COST OVERVIEW

On the left chart, we can see the actual and remaining labor resources (that is, we see how much has already gone to pay for the work of one or another specialist, and how much is still available), as well as the exact amounts shown in the table and the staff rate per hour of work.

Along the labor force, we still have material, such as printer paper, or coffee for employees, although the costs are not so great, but these resources are also very important.

The cost ratio for each category can be viewed as a pie chart on the right.

Fig. II.1.61. Task Cost Overview

These charts correspond to the following table in the project, agree that on the charts you can find the information you need much quicker.

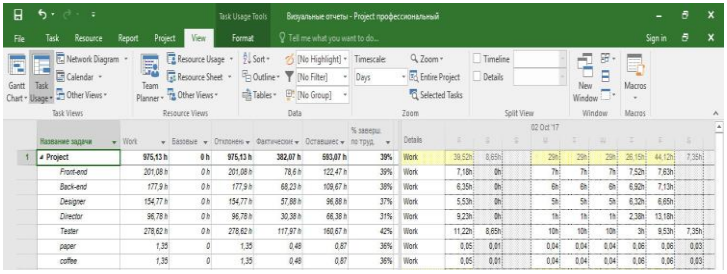


Fig. II.1.62. Task Cost Overview - Table

This category also includes the report WORK OVERVIEW

Fig. II.1.63. Work Overview

On the top schedule WORK, we can immediately conclude that: If the line of the remaining total labor costs falls down, the project implementation may be delayed. (it is very convenient that this is written in the tooltip immediately near the graph)

Pay attention to the graph at the bottom right, it displays the remaining availability of labor resources by the example of all project specialists, the resource ends when the job completes, but as soon as the employee performs his part of the work and the project moves to another person, the labor resource of the first starts to recover.

Fig. II.1.64. Work Overview - more

7. We will finish our study of this new function of visual reports with reports about TASK PROGRESS
Here we can see a general pie chart of the state of the tasks, as well as tasks that need to be completed on time, and can not be shifted or delayed.

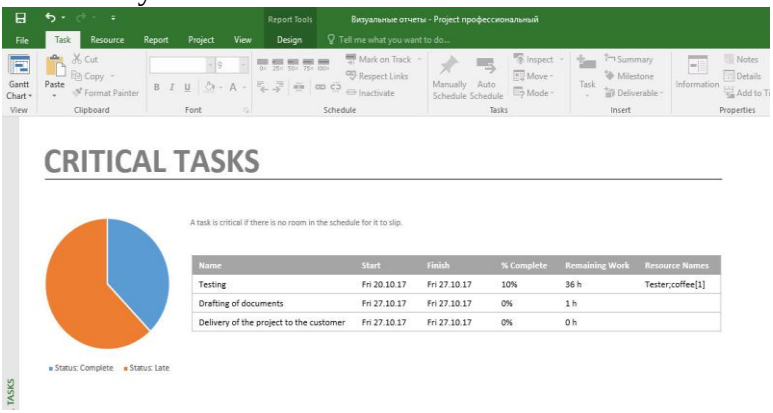


Fig. II.1.65. Critical Tasks

The second graph shows milestones (key project dates), and a schedule with the exact number of completed and remaining tasks.

8. Actually, you can create your own report, edit it as you like, specify stylistics, types of graphs, colors, etc.

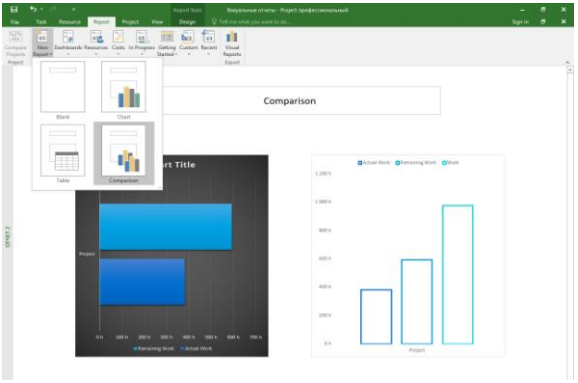


Fig. II.1.66. Your own report

Control questions and tasks for part II.1

1) Name the stages of the life cycle of software development.
2) How to configure MS Project?
3) How data is stored and displayed in MS Project (what are "internal" and "external" tables)?
4) What standard tables are part of MS Project?
5) What is a Gantt chart?
6) How is the group formatting of Gantt chart elements?
7) Name the built-in versions of Gantt charts in MS Project and describe their purpose.
8) How the skeletal plan of the project is made?
9) What is a milestone? How to create milestones in MS Project?
10) What types of connections can be defined between tasks in MS Project?
11) What is a phase and how phases are created in MS Project?
12) Create a new project in the Microsoft Project.
13) Build a Gantt chart tracking the critical path of the project.
14) Plan resources (labour, material, financial) for tasks and subtasks.
15) What are risks in the Microsoft Project environment?
16) Identification the risks. Classification of risks
15) In the Microsoft Project environment create and enter risks in the previously created project.
16) Identify the risks. Carry out the classification of risks
17) Carry out a qualitative risk analysis
18) Study and use the main features of MS Project for creating reports
19) Upcoming Task report
20) Cash Flow report
21) Work Overview report
22) Task Cost Overview report
23) Create your own report

II.2. PRACTICAL METHODS OF WORK IN RATIONAL ROSE ENVIRONMENT

II.2.1. Visual modeling of information systems. use case and actions diagrams in the design system Rational Rose

Theoretical information

Actors and Use cases

Algorithm of creation actors in the program **Rational Rose:**

1. Click the right mouse button on the section **Use Case View** in the browser window.

2. In the context menu that appears, select **New → Actor** . The actor called **New Class** will be added to the list of the browser window.

3. Select a new item list and enter an **Actor** name. Browser window with a list of actors for the courses registration system is shown on Fig. II.2.1.

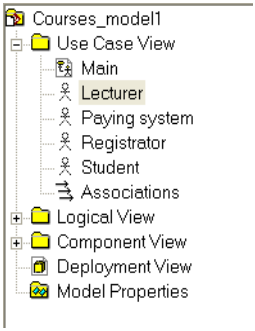


Fig. II.2.1 Item list with Actor names

To create **Use Cases** in program **Rational Rose** do the following:

1. Click the right mouse button on the section **Use Case View** in the browser window.

2. In the context menu that appears, select **New Use Case**. In the list of the browser will be a new precedent.

3. Enter the desired name for it.

Browser window with a list of **Use Cases** for courses registration system is shown on Fig. II.2.2.

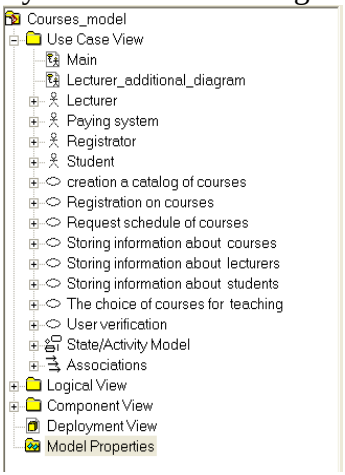


Fig. II.2.2. List of Use cases

Use Case diagrams

Use case diagram is a graphical representation of all or part of the actors, precedents and their interactions in the system. Each system has usually the main **Use case diagram**, which shows the boundaries of the system and basic functional behavior of the system. Other **Use case diagrams** can be created if necessary. Some examples:

- a diagram that shows all the precedents for certain actor;
- a diagram that shows all the precedents implemented in this iteration;
- a diagram showing a precedent and all his relations.

To create the main **Use case diagram** in program **Rational Rose** do the following:

1. Double-click on the item **Main** in the **Use Case View** in the list of the browser to open the diagram.
2. Select the **actor** in the browser and drag it to the diagram using the mouse.

3. Similarly place other necessary actors on the diagram.

4. Select **precedent** in the browser and drag it to the diagram using the mouse.

5. Similarly place other necessary precedents on the diagram.

Also **Actors** and **Use Cases** can be obtained directly in the diagram using the toolbar.

To create a **communicative associations** in program **Rational Rose**:

1. On the toolbar, click the button **Association** or button **Unidirectional Association**. If desired button is missing, click the right mouse button on the toolbar, in the context menu that appears, select **Customize** to add a button.

2. Click on the **actor** - the initiator of communication and drag the communication line to the desired precedent.

To create a relation **"include"** in program **Rational Rose** you need:

1. On the toolbar click on the button **Unidirectional Association**.

2. Click on using precedent and drag a **line of communication** to the used precedent.

3. Double-click on the **communication line** to open the **Specification**.

4. In the **Stereotype** select **"include"**.

Creating of relationship **"complementary"** in the program **Rational Rose** involves the following steps:

1. Click on the **Unidirectional Association** on the toolbar,.

2. Click on the precedent with additional features and drag an **association line** on the basic precedent.

3. Double-click on the line to open the **Specification**.

4. In the Stereotype select **"extend"**.

The main use case diagram for the system of

registration of training courses is shown on fig. II.2.3.

The procedure to create of **additional use case diagrams** in program **Rational Rose**:

1. Click the right mouse button on the section **Use Case View** listed browser.

2. In the context menu that appears, select **New → Use Case Diagram** .

3. Enter the name of the diagram.

4. Click the diagram and place on it the appropriate actors, precedents and communications.

Additional use case diagram is shown on fig. II.2.4.

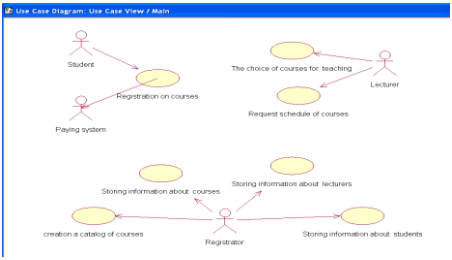


Fig. II.2.3. Main Use Case Diagram

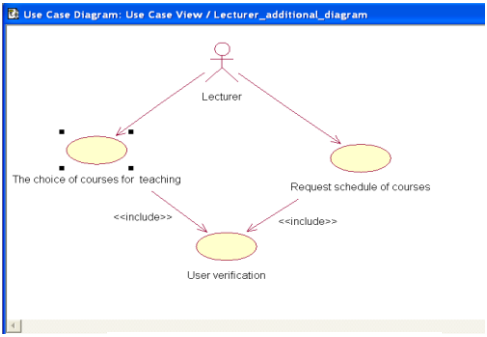


Fig. II.2.4. Additional Use Case Diagram

Activities

Activity diagrams reflect the dynamics of the project

and are the flow management schemes in system from action to action and also parallel actions and alternative flows.

In particular point of lifecycle the action diagrams may represent flows between functions or within a single function. At different stages of the life cycle they are created to reflect the sequence of operations.

Activity diagrams illustrate actions, transitions between them, elements of choice and synchronization lines.

To construct the **activity diagram** in program **Rational Rose** do the following:

1. Click the right mouse button on the section **Use Case View** listed browser.

2. In the context menu that appears select **New → Activity Diagram**. The new diagram will be added to the list.

3. Enter name of the diagram.

4. To open the diagram, double-click on it in the browser.

Browser window with the action diagram is shown on fig. II.2.5.

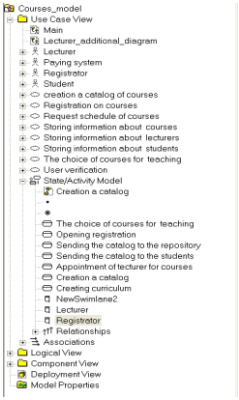


Fig. II.2.5. Activity Diagram

Activities

The activity is the performance of certain behavior in

flow of system control (see fig. II.2.6).

To create an **activity** in program **Rational Rose**:

1. Click the button **Activity** on the toolbar.

2. Click on the **activity diagram** to place the element that represents the activity.

3. Enter a name for the new **activity**.

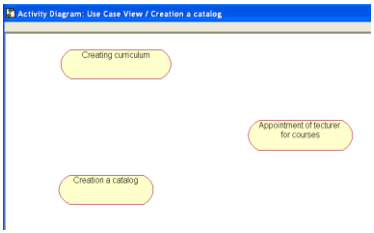


Fig. II.2.6. Activities

Transitions

Transitions are used to show the path of control flow from action to action (see. fig. II.2.7). They are usually made at the end of certain action.

To build **transitions** in the program **Rational Rose**:

1. Click button **Transition** on the toolbar.

2. Click on the initial action on the diagram and move the arrow to the next action.

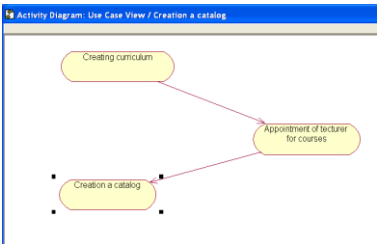


Fig. II.2.7. Activities and Transitions

Elements of choice

In the simulation of control flows in systems often need to show place of their separation on ground of conditional Choice. Transitions from the choice element include restrictive conditions that determine which direction of transition will be selected. The elements of choice and conditions allow to set an alternative way of control flow.

To create an element of choice in the program **Rational Rose** do the following:

1. Click on the button **Decision** on the toolbar.
2. Click on the activity diagram to put an element of choice in it.
3. Enter a name for the new element.
4. Click on the **State Transition** toolbar.
5. Click on the action in the activity diagram and move the arrow to the element of choice.

The element of choice is shown in Fig. II.2.8.

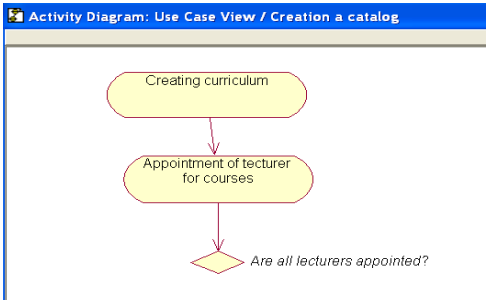


Fig. II.2.8. Element of choice

Consistency of creation **conditional branches** in the program **Rational Rose**:

1. Click **State Transition** button on the toolbar.
2. Click the element of choice in the activity diagram and move the arrow to the further action.
3. Double-click on the arrow to go to open the **Specification**.

4. Click on the tab **Detail**.

5. In the box **Guard Condition** input condition of transition .

Conditional branches are depicted in Fig. II.2.9.

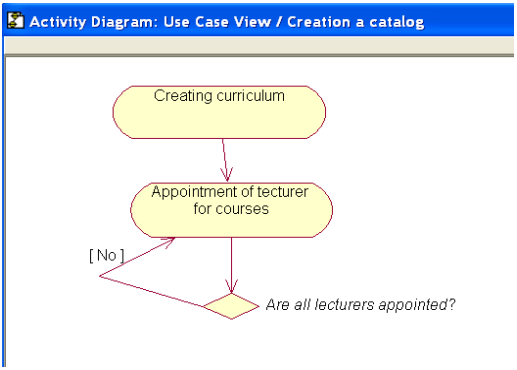


Fig. II.2.9. Conditional branches

Synchronization lines

Typically in the stream there are actions that are performed in parallel. **Synchronization line** allows you to specify the need of simultaneous execution, and also provides the whole performance of actions in a stream (that indicates the need to complete certain actions to move to the next. Thus, the synchronization lines can have multiple lines of input transitions and a single output in the flow or one input and a few outgoing.

To create a **synchronization line** in program **Rational Rose**:

1. Click on the button **Horizontal Synchronization** or **Vertical Synchronization** on the toolbar.

2. Click on the activity diagram to put on it a synchronization line .

3. Click on the button **State Transition** on the toolbar and add the necessary input and output transitions to the

synchronization line.

Synchronization lines are shown on fig. II.2.10.

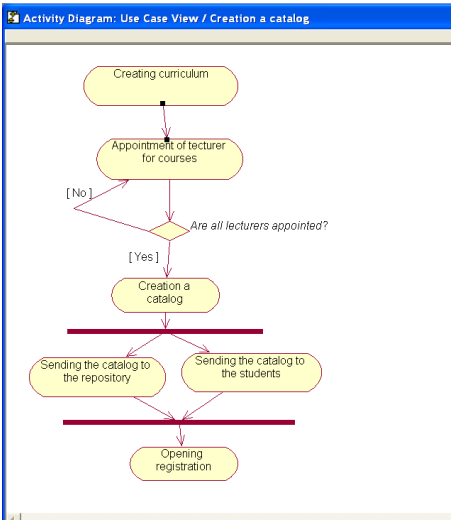


Fig. II.2.10. Synchronization lines

Swimlanes

Swimlanes divide the activity diagram in several areas. This is necessary in order to show who is responsible for the implementation of activities at each site.

Algorithm of creation of **swimlanes** in the program **Rational Rose**:

1. Click on the button **Swimlane** on the toolbar.
2. Click on the activity diagram to create a new section called **New Swimlane**.
3. Double-click on the name of the new section to open the **Specification** (Options).
4. Enter the desired name in the field Name.
5. To resize, move the section of the border.
6. Move all the necessary activities and transitions on

the diagram into the new section.

Activity diagram with swimlanes is shown on fig. II.2.11.

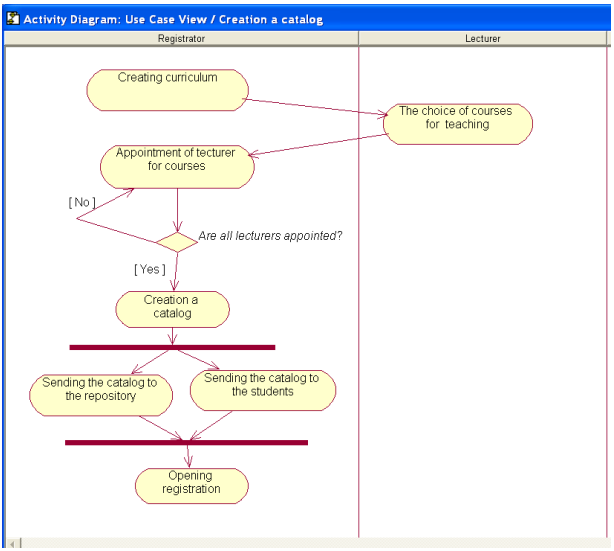


Fig. II.2.11. Swimlanes

The start and the end states

To indicate the initial and final states in the flow of system control the special characters are used. The **start state** is represented by solid circle, and the **end** - by solid circle, circled by additional circle. Usually in the flow there is one start state and several end states - for each alternative direction.

The sequence of creating the **start** and the **end states** in program **Rational Rose**:

1. Click on the button **Start State** or **End State** on the toolbar.

2. Click on the activity diagram to put the symbol of the start or the end state on it.

3. If you added the initial state, click on the button State Transition on the toolbar, and then on the symbol and the initial state transitioned to the first steps in the flow. Similarly include the final state to the diagram.

The activity diagram with start and end states is shown on fig. II.2.12.

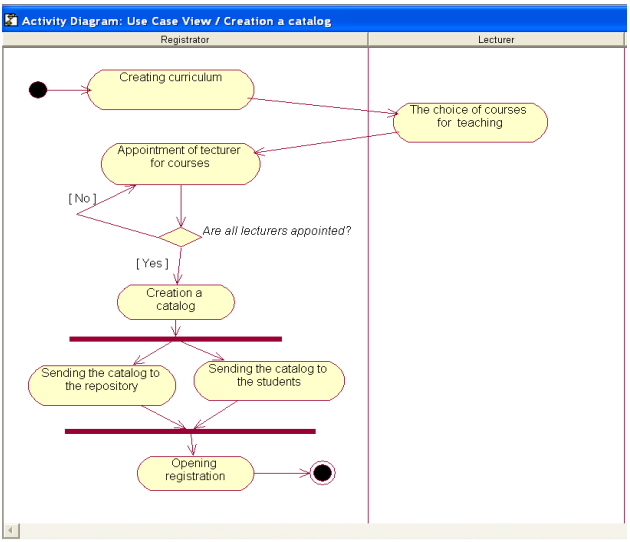


Fig. II.2.12. Start and end states

II.2.2. Development of class diagram as a model of real object

Theoretical information

Classes

An object is some kind of real world essence or conceptual essence. The object may be something specific, such as a truck or computer, or conceptual, such as a chemical process, a banking transaction, a trade order, a credit history, or a rate of return.

An object is a concept, abstraction, or thing with clearly defined boundaries and meaning to the system. Each object in the system has three characteristics: state, behavior and individuality.

The condition of an object is called one of the conditions in which it can be. The state of the system usually changes over time and is determined by a set of properties called attributes, property values, and relationships between objects. For example, a training course object in a course registration system may be in one of two states: open to enrollment or closed to enrollment. If the number of students enrolled in the course is less than ten, the course enrollment continues. After registration of the tenth student, it is terminated.

Behavior determines how an object responds to requests from other objects and what the object itself can do. The behavior is implemented using a set of operations for the object. In the course registration system, the object of the training course may have operations to add a student and remove a student.

Individuality means that each object is unique, even if its state is identical to that of another object.

Class is a description of a group of objects with common properties (attributes), behavior (operations), relationships with other objects, and semantics. So the class is a template for creating an object.

Each object is an instance of a particular class and cannot be an instance of several classes. For example, a class training course may be defined by the following characteristics:

- attributes - place of employment, time of employment;
- operations - to get a class, get class time, add a student to the course.

Class represents one and only one abstraction, that is, it must reflect one basic essence. For example, a class that can

store student information and course data that a student has attended for several years is not a "good" class because it does not represent one entity. This class should be divided into two related classes: student and student history.

Class names are selected according to the concepts of the visual area. In UML, classes are represented as separated rectangles. The upper section specifies the name of the class, the middle section contains its structure - attributes, and the lower section describes its behavior - operations. The class is shown in fig. II.2.13.

a) **b)**

Fig. II.2.13. a) A class created in a browser window
b)Notation of the UML language for the class

Each object is an instance of a particular class and cannot be an instance of several classes. For example, a class training course may be determined by the following characteristics:

- attributes - place of employment, time of employment;
- operations - to get a class, get class time, add a student to the course.

The procedure for the construction of **classes** in the program **Rational Rose**:

1. Click the right mouse button on the section **Logical View** in a browser window.

2. In the context menu that appears , select **New → Class**. A new class called NewClass will be added to the

browser list .

3. Enter class name.

Consistency create **class diagram** to show the attributes and operations in program **Rational Rose**:

1. Click the right mouse button on a package in the browser window.

2. In the context menu that appears, select **New → Class Diagram**. A new diagram will be added to the browser list.

3. Enter the name of the new diagram.

To display all **attributes** and **operations** in program **Rational Rose** do the following:

1. Click the right mouse button on a class in the diagram.

2. In the context menu that appears select **Options → Show All Attributes**.

3. Call the class context menu and select **Options → Show All Operations**.

Class diagram for the package Items of University is shown on Fig. II.2.14.

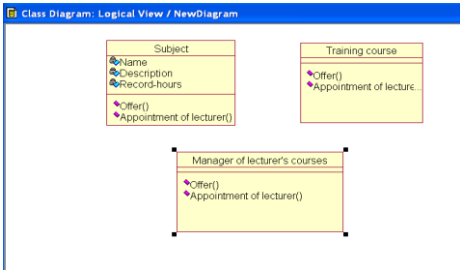


Fig. II.2.14. Class diagram for the package Items

Sequence of creation relation of **inheritance** in program **Rational Rose**:

1. Open the class diagram, which shows the hierarchy of classes.

2. Click on the button **Class** on the toolbar, and then on the diagram to place here class.

3. Enter the name of the class. The class can also be created in browser and moved to the diagram.

4. Click on the button **Generalization** in the toolbar.

5. Click on the subclass and draw a **line of communication** to the superclass.

Relations of inheritance are shown on fig. II.2.15.

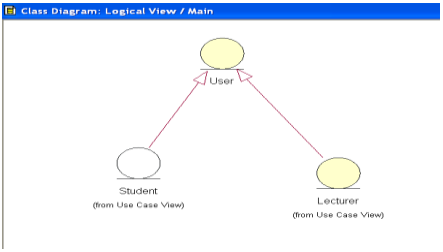


Fig. II.2.15. Relations of inheritance

The class in the browser window is shown in fig. II.2.16.

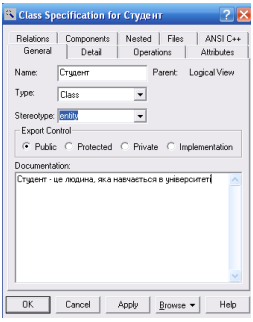


Fig. II.2.16. Class Student

An *entity* class is used to model data and behavior with a long lifecycle. This type of class can represent the essence of the real world or the internal elements of the system. Such classes are usually independent of the environment, that is, they are insensitive to the interaction of the environment with the system. Therefore, they are application-independent and can be used in a variety of applications.

The first step is to study the responsibilities outlined in the event stream to identify precedents (what the system should do). Essential classes are usually the classes that the system needs to perform certain responsibilities

The *entity* classes are usually defined at the processing stage. They are often called visual area classes because they are abstractions of real-world objects.

Boundary classes provide interaction between the environment and the internal elements of the system. These classes provide an interface for the user or other system (that is, for the actor). They form an externally dependent part of the system and are used to model the system interfaces.

Actor / script pairs are studied to identify boundary classes. Such classes defined in the processing phase are usually top-level classes. For example, you can model a window but not model its dialog elements and buttons. In this case, you will describe the requirements of the user interface but do not implement it.

Boundary classes are also used to communicate with other systems. At the design stage, these classes are being refined and discussed for the implementation of interaction protocols.

Control classes are used to model the sequential behavior of one or more precedents and to coordinate events that implement the behavior they contain.

Governing classes can be represented as classes,

"executing" precedent and determining its dynamics. They are usually application dependent.

In the early stages of processing, management classes are added for each actor / precedent pair. Such classes determine the flow of events in precedents.

The management class for each actor / precedent pair is created initially. In further analysis and design, management classes can be excluded, separated, or combined.

Stages of creating stereotypes for classes in the Rational Rose program (fig. 83):

1. Right-click on the class name in the browser list.

2. In the context menu that appears, select **Open Specification**.

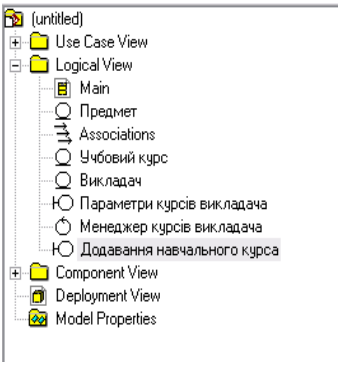


Fig. II.2.17. Classes for the script for adding a training course

3. Click the **General** tab.

4. In the list that opens, - **Stereotype** - select the desired stereotype. To create a new stereotype, enter its name in the **Stereotype** list box.

If there are not many classes in the system, it is quite easy to manage them. But many systems consist of a large number of classes, so a mechanism is needed to break them down into groups and make it easier to manage and reuse. The concept of packages is useful here.

A *package* in a logical representation of a model is a

collection of classes and other related packages. By combining classes into packages, we can get a higher-level representation of the model. By examining the contents of the package, on the contrary, we get a more detailed view.

Each package contains an interface implemented by a set of its public classes, that is, with which classes from other packages can communicate. Other package classes are implementation classes that do not interact with classes in other packages.

In a sophisticated system, packets can be created at the processing stage to facilitate perception. In a simpler system, the classes highlighted in the analysis phase can be grouped into one package representing the system itself. In the course of further analysis and design, packages are needed to group the classes used in the system architecture.

In UML, packets are represented as folders.

To create packages in **Rational Rose**:

1. Right-click **Logical view** in the browser window.

2. In the popup menu that appears, select **New → Package**.

3. Enter the desired package name.

The package created in the browser list is shown in fig. 84. After creating the package, you can place the required classes in it.

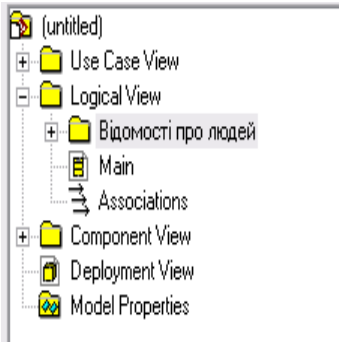


Fig. II.2.18. Package in browser

Objects and classes in the course registration system

Consider the scenario of adding a course (which is an internal stream of precedent for selecting subjects to teach. This scenario allows the teacher to select a course for a particular semester.

Although we look at this process step by step, in practice most steps can be done at the same time.

This precedent only interacts with the actor teacher. This scenario is only one of the opportunities provided by precedent (it also determines that the teacher can modify, delete, view, and print courses). This means that there must be a mechanism in the system that allows the teacher to choose the desired action. To provide the needs of the teacher, a special class is created - the course parameters of the teacher.

Additionally, we can specify a class that is used to add new courses available to the teacher - to add a training course.

This scenario consists of subjects, training courses, and assignments of teachers. We can distinguish three class-entities: subject, training course and teacher.

We will add one management class to handle the event stream for precedent - the teacher's course manager.

Now classes (with selected stereotypes) can be added to the model (see Fig. II.2.19).

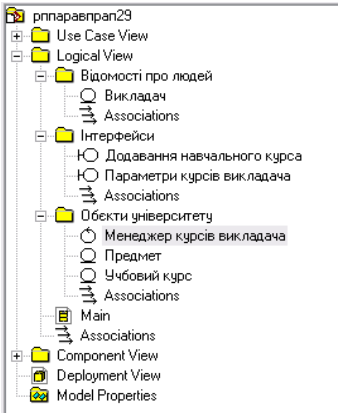


Fig. II.2.19. Packages

The next step is to merge the classes into packages. At this stage, we distinguish six classes: subject, training course, teacher, teacher course parameters, adding training course and teacher course manager. They can be divided into three logical groups: university-specific objects; objects containing information about people; interfaces for actors. So we can create the following packages: Interfaces, University Objects and People Info. The classes are then placed into appropriate packages (see Figure 22).

Class diagrams

As new classes are added to the system, their text rendering becomes uncomfortable. Class diagrams help graphically represent some or all of the classes in the model.

The main class diagram in the logical representation of the model usually displays the system packages. Each package also has its main class diagram, which usually contains public classes of the package. Other charts are created as needed.

Here are some typical examples of using class diagrams:

- viewing all classes of implementation in the package;
- reviewing the structure and behavior of one or more classes;
- View the class hierarchy.

Rational Rose automatically creates a master class diagram in the logical representation of the model.

To add packages to the main class diagram, you must do the following:

1. Double-click the **Main diagram** list in the browser to open the diagram.
2. Select the desired package from the list.
3. Drag the package to the chart.

The main class diagram for the registration system is shown in fig. II.2.20.

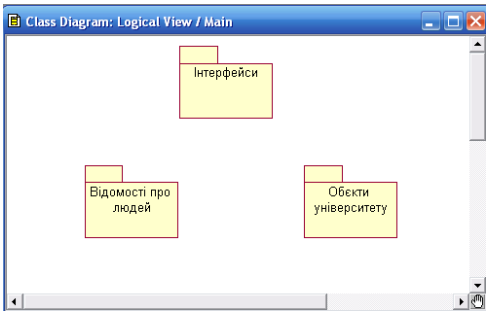


Fig. II.2.20. Main class diagram

Stages of creating a master class diagram of a package in **Rational Rose**:

1. Double-click the package image in the **class diagram**. The package will open and the main class diagram will appear.

2. Select the class you want from the browser list and drag it to the chart with the mouse. You can use the **Format → Stereotype** display menu command to display the class stereotype in the diagram.

Conclusions

Objects - a computer representation of entities (objects of the real world or concepts invented by man). An object is a concept, abstraction, or thing with clearly defined boundaries and meaning for the system. Each object in the system has three characteristics: status, behavior and personality. The state of the object is one of the conditions in which it can be. Behavior characterizes an object and shows how it responds to requests from other objects. Individuality means that each object is unique, even if its abundance is identical to that of another object.

A class is a description of a group of objects with common properties (attributes), behavior (operations), relationships with other objects (associative or aggregation), and semantics.

A package in the logical representation of a model is a

collection of classes and other related packages. By combining classes into packages, we can get a higher-level representation of the model.

Class diagrams help graphically depict some or all of the system classes. Class diagrams can also be created in the precedent model view. They are usually attached to a precedent and contain representations of the classes involved in their execution.

Defining relationships

The system consists of a large number of classes and objects. Its behavior is ensured by the interaction of objects. For example, a student is added to a course when a message is added to the course to add a student. In this case, it is said that the object sends a message to another object. Relationships serve as conductors between objects. Two types of relationships that can be distinguished in the analysis phase are association and aggregation.

An *association* is a bidirectional semantic link between. classes. This is not a data flow defined in structural analysis and design - data can flow in both directions of associative communication. The presence of association between classes indicates that the objects of these classes are interconnected. For example, associative relations between subject classes and the course manager mean that the objects of the subject class are related to the objects of the course manager class. The number of related objects depends on the strength of the associative relationship.

In UML, *associative relationships* are represented as a line connecting connected objects.

The sequence of associative relationships in **Rational Rose**:

1. In the toolbar, click the **Association** button. If it is not, right-click in the toolbar and select **Customize** from the context-sensitive menu that appears.

2. Click one of the classes in the class diagram.

3. Drag the associative link to the second class.

The associative relationship between classes is shown in Fig. II.2.21.

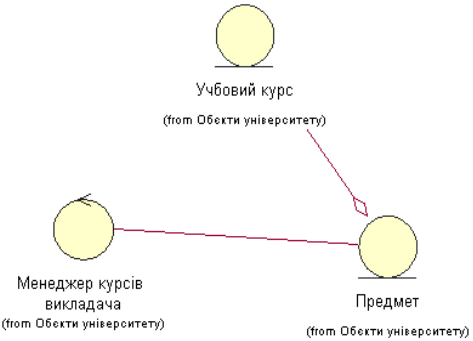


Fig. II.2.21. Associative

Aggregation is a special form of association between the whole and its part or parts. Aggregation is known as a "part of" or "containing" type relationship.

To create aggregation relationships in **Rational Rose**:

1. In the toolbar, click the **Aggregation** button. If it does not, right-click on the toolbar and select **Customize** from the context menu that appears.

2. In the class diagram, click on the class that appears as a whole and drag the aggregation link to the class that is part of it.

If two classes are rigidly bound by the whole-to-part relationship, then this is a typical aggregation relationship

Whether the relationship is an association or an aggregation often depends on the subject area. To determine the relationship between the two classes, study the scenarios. The transmission of messages between objects indicates that the latter interact with each other. Associations and aggregations provide an opportunity for interaction.

The diagram of classes with the indicated relations is shown in fig. II.2.22

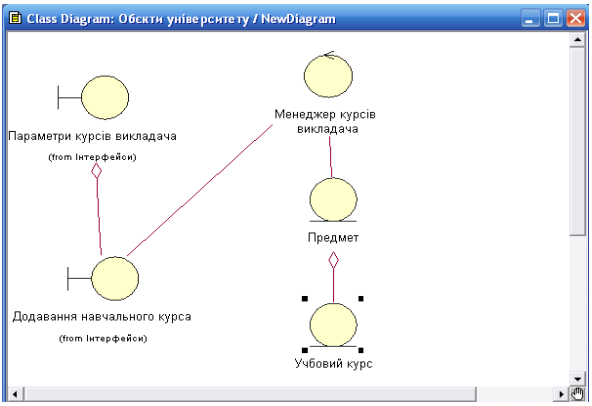


Fig. II.2.22. Diagram of classes with the indicated relations

Relationship between packages

Relationships between packages are also included in the model. This type of communication is a dependency relation and is represented as a dotted arrow pointing to the dependent packet. If packet A depends on packet B, then one or more classes in packet A initiate communication with one or more public classes in packet B. Package A in this case is called packet-client, and packet B is packet-supplier.

Relationships between packages are revealed by examining scenarios and relationships between classes of the system. Because this is an iterative process, relationships can change during analysis and design.

In a script to add a training course, the Add a Training Course class sends a class manager message to the teacher. This indicates that there is a connection between the Interfaces package and the University Objects. At this stage, we cannot isolate any relationship with the People Information package.

To create relationships between packages in **Rational**

Rose:

1. Click the **Dependency Relationship** button in the toolbar.

2. Click on the client package and drag the line to the provider package.

Conclusions

Relationships act as a guide between objects. The two types of object relations that can be distinguished in the analysis phase are associations and aggregations. The association is called bidirectional semantic communication between classes. Aggregation is a special form of association between the whole and its part or parts.

Scenarios are being explored to find relationships between the two classes. Packages can be linked by a dependency relation. If packet A depends on packet B, then one or more classes in packet A initiate communication with one or more public classes in packet B.

Attributes

Most class *attributes* are found when analyzing the subject area, system requirements, and event flow descriptions, and when compiling a class description. In addition, the visual area itself is a good source for defining attributes. For example, system requirements state that the subject name information, its description and number of teaching hours are contained in the catalog of training courses per semester. From this it follows that the title, description and number of lessons are the attributes of the subject class.

The **Rational Rose** attribute creation sequence:

1. Right-click on a class in the browser window.

2. In the popup menu that appears, select **New** → **Attribute**.

3. Enter a name for the new attribute.

The *attribute* definitions in the documentation should be concise and clear and contain information about the attribute assignment, not its structure. Here is an unsuccessful example of the attribute description of the class name of the subject:

"Character string up to 15 characters long". The following variant will be correct: "Name of academic subject used in university publications".

To describe attributes in **Rational Rose**:

1. In the browser window, click the "+" icon to the left of the class name to open a list of its properties.

2. Select an attribute by clicking on it.

3. Place the cursor in the description box and enter a description for the class attribute.

The class implements a number of responsibilities that determine the behavior of its objects. Responsibilities are fulfilled through class-specific operations.

To create an operation in **Rational Rose**:

1. Right-click on a class in the browser window.

2. In the popup menu that appears, select **New → Operation**.

3. Enter a name for the new operation.

Attributes and operations can be shown in the class diagram. Most often, it is created to reflect the structure and behavior of package classes. Relationships to this chart are usually not tolerated.

The sequence of creating a class diagram to display the attributes and operations of a package in **Rational Rose**:

1. Right-click on the package in the browser window.

2. In the popup menu that appears, select **New → Class Diagram**. A new chart will be added to your browser list.

3. Enter a name for the new chart.

To display all attributes and operations in Rational Rose, follow these steps:

1. Right-click on a class in the chart window.

2. In the popup menu that appears, select Options → Show All Attributes.

3. Recall the context menu for the class and select Options → Show All Operations.

The class diagram for the University Objects package is shown in fig. II.2.23.

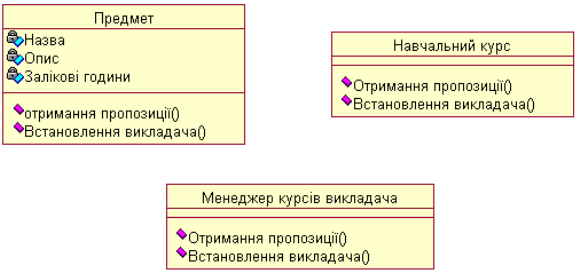


Fig. II.2.23 Class diagram for the University Objects package

A relationship can also have structure and behavior. This occurs when the information is linked to objects, not to the object itself.

Consider the following example. A student can attend up to four training courses, and a training course can be read by several students - from three to ten. Each student receives a grade for the course. Where should the score be stored? It does not belong to the student because he / she will probably receive different grades in different subjects. The grade does not belong to the course, because students will receive different grades for this course. Assessment information relates to the relationship between the student and the course. They are modeled using an associative class that behaves like any other class and can also be relevant. In our example, the student receives an assessment report that includes the associated assessment objects.

To create associative classes in a program in **Rational Rose**:

1. Click the Class button on the toolbar.
2. Click on the diagram to place a class on it.
3. Enter a name for the class.

4. Add the necessary attributes and operations for the class.

5. Click the Association Class button in the toolbar.

6. Click on the associative class and draw a line to connect the classes that associate the associative class.

7. If necessary, add additional relations to the associative class.

The associative grade is shown in Fig. II.2.24.

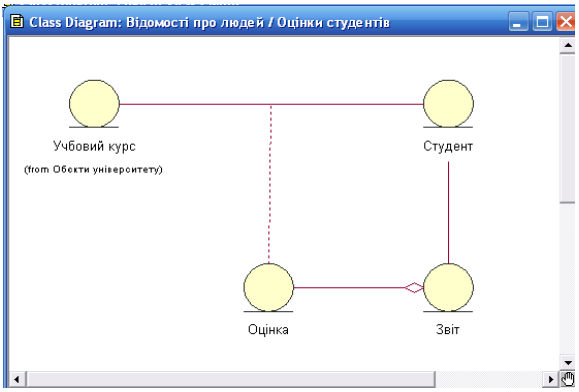


Fig. II.2.24. Associative class

Conclusions

The class performs a number of responsibilities that determine the behavior of its objects. Duties are performed through specific operations. The structure of the object is described by the attributes of the class.

Each attribute is a data field contained in a class object. An object derived from a class endowed with the values of all class attributes. Attributes and operations defined for the class are the main significant and functional elements in the application being developed.

Many class attributes are found when analyzing the domain, system requirements, and description of event streams, as well as when compiling a class description. In addition, the domain itself is a good source for defining attributes.

A relationship can also have structure and behavior. This occurs

when the information is linked to objects, not to the object itself. Relationship structure and behavior are modeled using associative classes.

Inheritance

Inheritance is called the relationship between classes when one class uses part of the structure and behavior of another or more classes. Imitation creates a hierarchy of abstractions in which a subclass is inherited from one or more superclasses. Inheritance is also called a hierarchy of the same type or appearance. The subclass inherits all the attributes, operations, and relationships defined in each of its superclasses. Therefore, all attributes and operations defined at the top level of the hierarchy will be inherited by classes at the lower levels. Additional attributes and operations applied only at this level of the hierarchy can be added to the subclass. The subclass may contain its own implementation of the inherited operation.

There are no restrictions on the number of classes in the hierarchy.

Inheritance allows you to reuse classes. You can create a class for one application, and then generate a subclass with advanced functionality for use in another application.

There are two ways to define inheritance - generalization and specialization. In any system under development, both methods are commonly used.

The sequence of creating an Fig. 103. The relation of imitation inheritance relationship in **Rational Rose**:

1. Open the **class diagram** showing the inheritance hierarchy.
2. Click the **Class button** on the toolbar and then the diagram to place a class on it.
3. Enter a name for the class. The class can also be created in a browser and moved to a chart.
4. Click the **Generalization button** in the toolbar.
5. Click on the subclass and draw the link to the

superclass.

The inheritance ratio is shown in Fig. II.2.25.

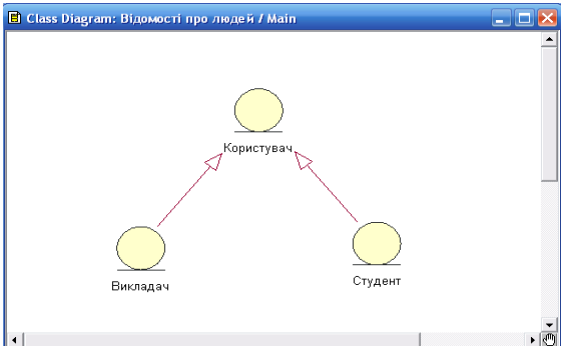


Fig. II.2.25. The relation of inheritance

Inheritance tree

The basis for specialization (that is, the purpose of creating a subclass) in relation to imitation is called a discriminator.

For example, one of the discriminators for the class is the subject of study. The classes in the subject and the extramural subject may become subclasses for the class of objects created on the basis of this discriminator. Inheritance ratio for all subclasses received from one discriminator, are represented in the form of a tree. Another subclass of a class may be a compulsory class. This subclass will not be part of the imitation tree because it belongs to another discriminator - the object type. The issue of identifying multiple discriminators for one class should be carefully considered. For example, what happens if the required item is also full-time? Is this an example of multiple inheritance? Do you need to apply aggregation here? In the course of analyzing and designing the answers to these questions, we will gradually get a complete model structure.

To create an inheritance tree in **Rational Rose**:

1. Open the **class diagram** showing the inheritance hierarchy.

2. Click the **Class button** on the toolbar and then the chart

to place a class on it.

3. Enter a name for the class. The class can also be created in a browser and moved to a chart.

4. Click the **Generalization** button on the toolbar.

5. Click on the subclass and draw the link to the superclass.

6. For each subclass that is part of the inheritance tree: click the **Generalization button** in the toolbar, click on the subclass, and draw a generic link to the inheritance icon (as a triangle).

The tree-like relationship of inheritance is shown in Fig. II.2.26.

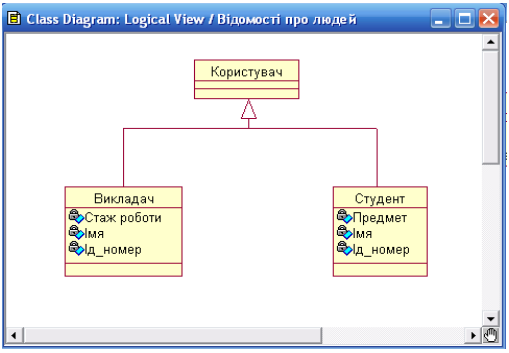


Fig. II.2.26. Inheritance hierarchy for the class

After creating a superclass, attributes, operations, and relationships are placed at the highest level of the hierarchy whenever possible. What properties should be transferred? Consider a hierarchy with a base user class. The attributes, operations, and relationships for the subclasses are shown in Figs. 31. Since the attributes of the name and id are the same format, they can be confidently transferred to the superclass of the user.

To move attributes and operations in **Rational Rose**:

1. In the browser window, click the "+" icon to the left of the subclass to open a list of its properties.

2. Select the **attribute** or **operation** you want to move.

3. Drag an attribute or operation to a superclass.

4. Remove this attribute or operation from other subclasses.

The inheritance hierarchy after the attribute transfer is shown in Fig. II.2.27.

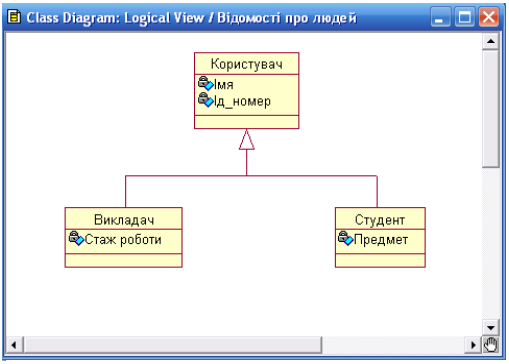


Fig. II.2.27. Attributes Moved to Superclass

Single and multiple inheritance

In single inheritance, the class contains a single set of descendants, that is, one chain of superclasses (for example, a car is a car and a car is a vehicle). Multiple imitation involves more than one chain of superclasses (an amphibian is a car, a car is a vehicle, at the same time an amphibian is a boat and a boat is a vehicle). Multiple inheritance raises a number of problems, including name conflict and multiple copies of inherited properties. The way to solve such problems is chosen depending on the programming language, in particular, virtual base classes in C ++. Multiple inheritance can cause confusing and hard-to-follow code - the more superclasses it is, the harder it is to determine where it came from and what would happen when making changes. Conclusion: Use multiple inheritance only when needed and with extreme caution.

Inheritance and aggregation

Inheritance is often misused. There is an opinion that "the more it is used, the better the code will become". This is a

mistake. In fact, misuse of imitation can lead to problems. For example, a student may study full-time or part-time. Let's create a superclass student and two subclasses - full-time student and part-time student. There are some problems with this structure. What happens if:

- the student of the full-time department decides to go to correspondence? This means that the object will have to change its class ?;

- Will another dimension be added (for example, a student receiving a scholarship and not receiving a scholarship)? New subclasses will be needed here to present scholarship information, as well as multiple inheritance for support for all combinations (full-time student, scholarship recipient, part-time student, scholarship recipient, and so on).

Inheritance should serve to separate the community from the specific. Aggregation - to reflect combined relationships. Often, both types of relationships are used together. The student class has a classification (aggregation), which, in turn, is divided into full-time and part-time (imitation) classes - see. Fig. II.2.28.

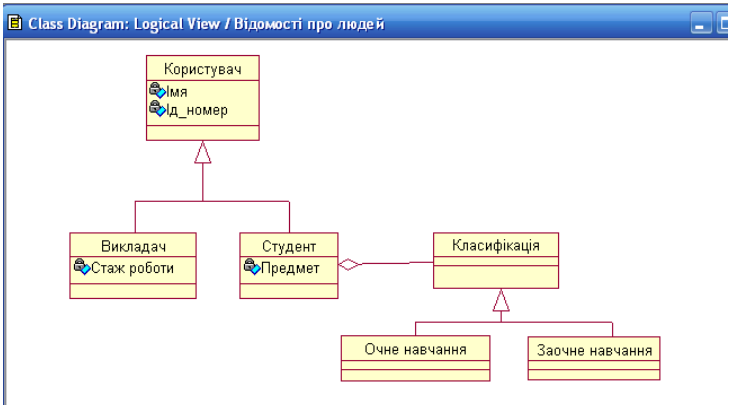


Fig. II.2.28. Inheritance and aggregation

Conclusions

Inheritance allows you to create a hierarchy of classes when shared structure and behavior are shared between them. The term "superclass" refers to a class containing common information. Descendant classes are called subclasses. The subclass inherits all the attributes, operations, and relationships defined in all its superclasses.

II.1.3. Software for class diagram implementation

Theoretical information

A class is a special design of an object-oriented programming language used to group related variables and functions. In this case, according to OOP terminology, global class variables (member variables) are called data fields (also properties or attributes), and function members are called class methods. A created and initialized
class instance is called a class object. Based on a single class, you can create multiple objects that will differ in their state (field values).

Classes can be used to create subclasses that inherit the properties and behavior of parent classes. This allows you to create an entire class hierarchy.

The methods implement the behavior of objects. Practically all work with objects occurs through methods. They can change the state of the object or simply give access to the data encapsulated in the object. There are several types of methods that have some differences in different programming languages. Different access rights can be assigned to methods and data fields, which will depend on them from different parts of the code. Access rights and type of methods are specified by modifiers when describing methods. The method that creates and initialises an instance of a class is called a class constructor. The method that implements an object is called a class destructor.

Basic principles of object-oriented programming:

- encapsulation;
- inheritance;
- polymorphism.

Encapsulation is a mechanism that combines data and methods that process this data and protects both against external influences or misuse.

In the middle of the object, data and methods can have different degrees of openness. Typically, open class members are used to provide an interface controlled by its closed part.

Inheritance is the process by which one object can acquire the properties of another object and add features specific to itself.

Polymorphism is a programming concept that uses a common interface to process data of various specialized types.

A method in object-oriented programming is a subroutine (procedure, function) that is used exclusively with a class or with an object.

Similar to a procedure in procedural programming languages, a method typically consists of a sequence of operations to perform an action, a set of input parameters to configure this action, and possibly an output value of some type (value to be returned).

The purpose of the methods is to take some action on the class fields (member variables) and provide a mechanism for accessing those data fields that are encapsulated in the object or class.

Operator overloading is one way of realizing a polymorphism that consists in the possibility of several different applications of the operator having the same name but differing in the types of parameters to which they are applied simultaneously.

Work execution

In the **Rational Rose** environment, we create a matrix class (Fig. II.2.29).

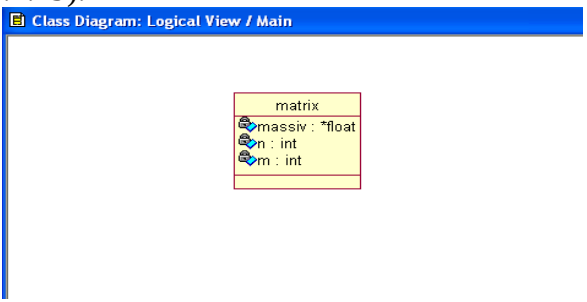


Fig. II.2.29. *Matrix class* in the environment Rational Rose

Let's describe our class in the **Documentation** section (Fig. II.2.30).

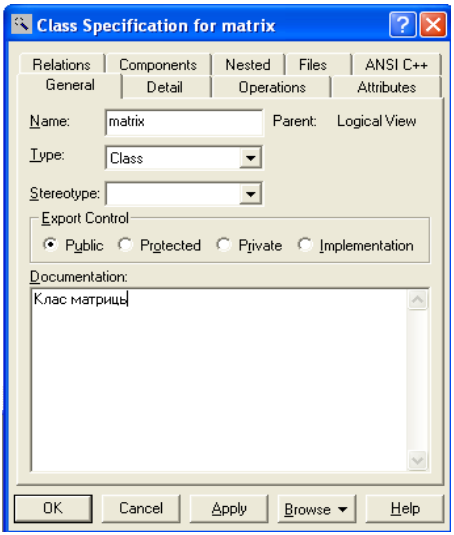


Fig. II.2.30. Class description

Let's create class fields - two-dimensional array and its dimension (number of rows and columns) - see. Fig. II.2.31, II.2.32, II.2.33.

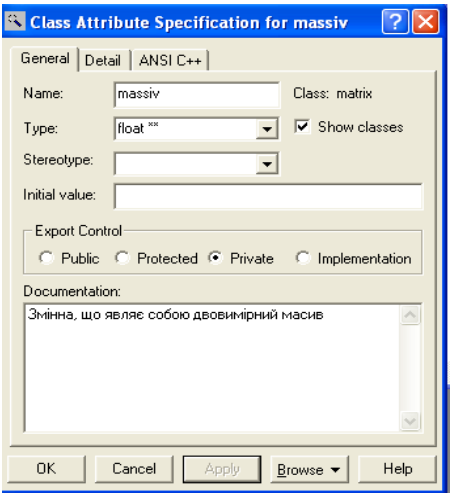


Fig. II.2.31. The *massiv* variable and its description

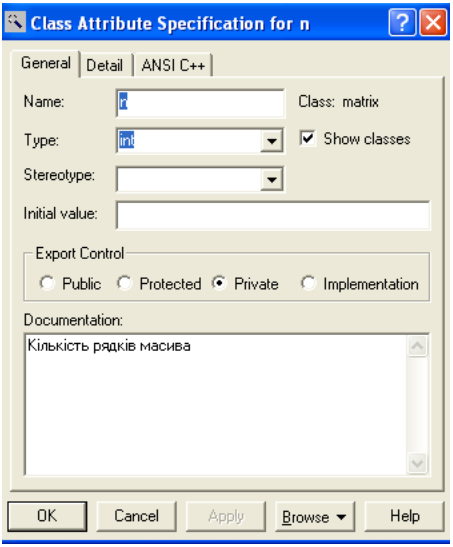


Fig. II.2.32. The *n* variable and its description

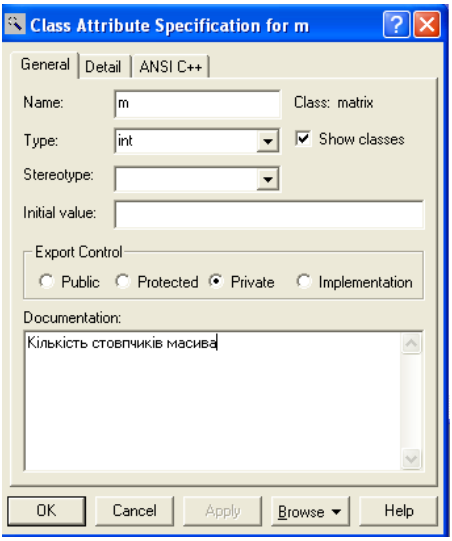


Fig. II.2.33. The *m* variable and its description

Now let's create class methods, describe them, specify the output type and input variables - see. Fig. II.2.34 - II.2.40.

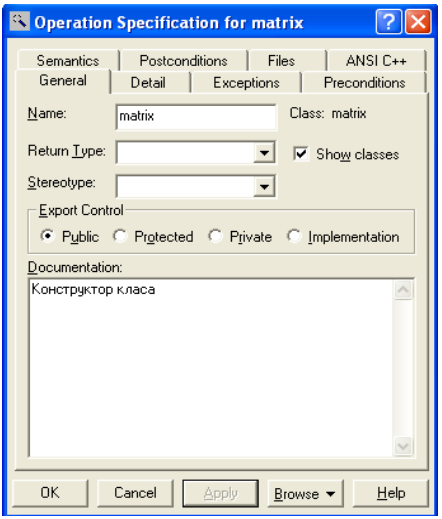


Fig. II.2.34. *Class constructor* and its description

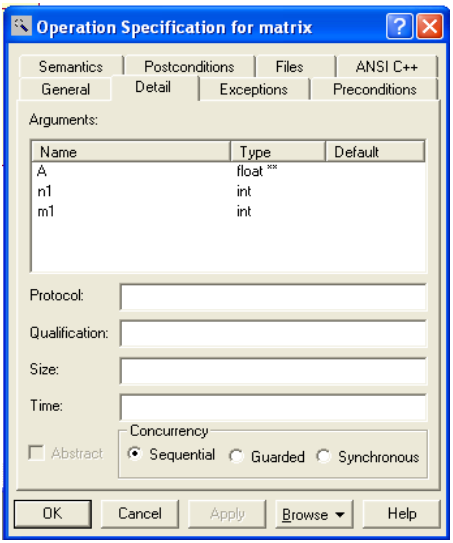


Fig. II.2.35. Input parameters of the overloaded constructor

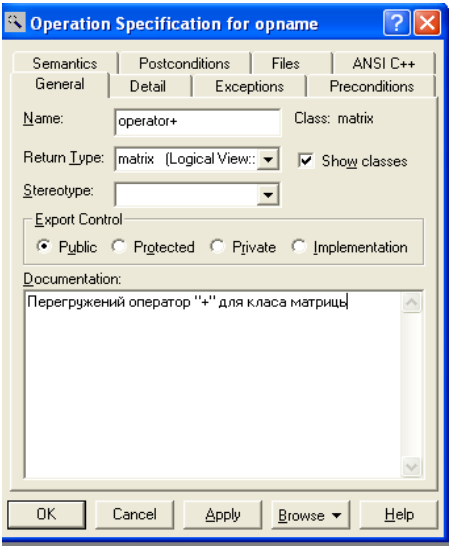


Fig. II.2.36. Overloaded operator "+" and its description

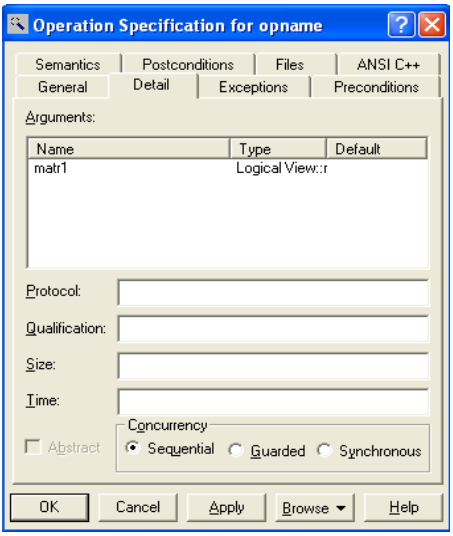


Fig. II.2.37. Input parameter of overloaded operator "+"

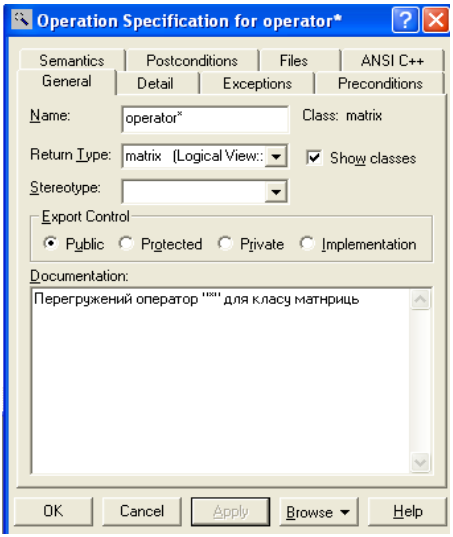


Fig. II.2.38. Overloaded operator "*" and its description

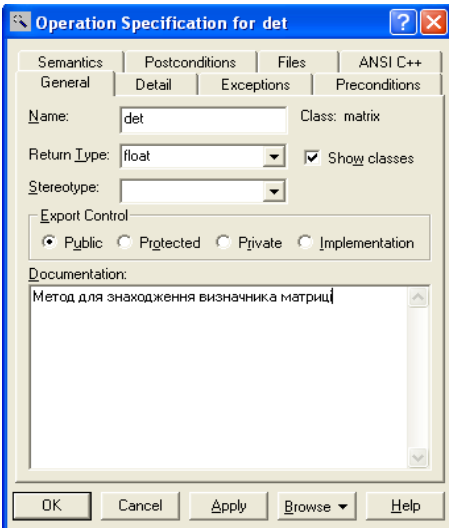


Fig. II.2.39. The method of finding the determinant of the matrix and its description

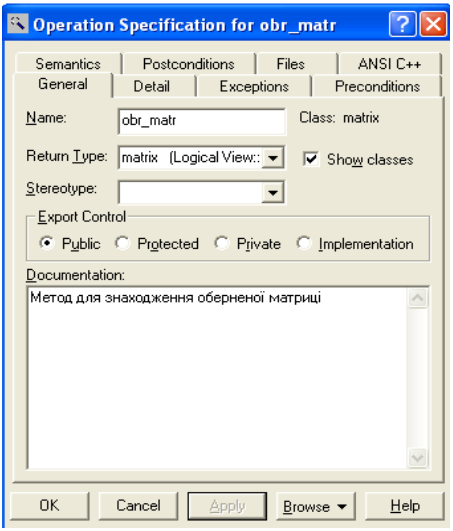


Fig. II.2.40. Method of finding the inverted matrix and its description

We have this class with all the necessary fields and methods - fig. II.2.41.

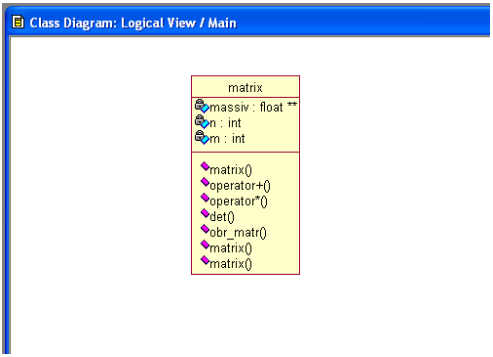


Fig. II.2.41. *Matrix* class with all the required fields and methods

The next step is to generate the code. To do this, first convert the model - see. Fig. II.2.42.

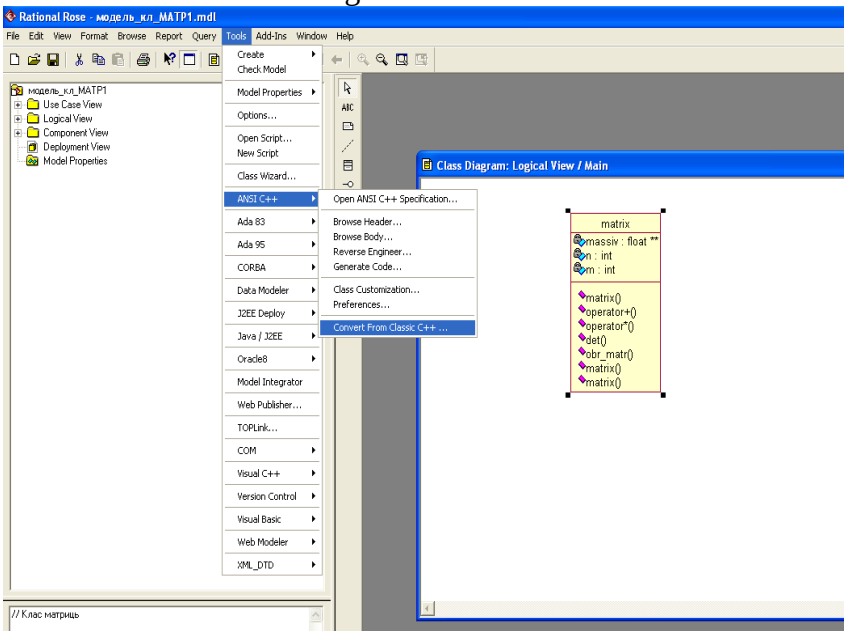


Fig. II.2.42. Model conversion for code generation

And now, actually, we generate the code - fig. II.2.43.

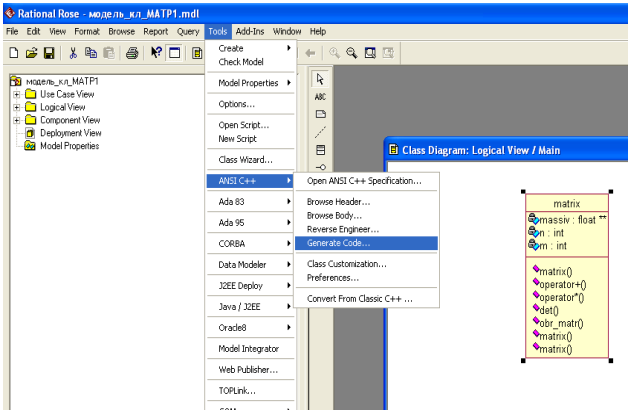


Fig. II.2.43. Code Generation

The generated code is shown in fig. II.2.44 and II.2.45.

Fig. II.2.44. *Matrix* class header file

Fig. II.2.45. The *matrix* class implementation file

Control questions and tasks for part II.2

1) Actors and Use cases in Rational Rose
2) Use Case diagrams
3) Activities. Activity diagrams
4) Transitions. Elements of choice
5) Synchronization lines. Swimlanes
6) The start and the end states
7) Identify the actors, use cases and actions.
8) Construct use case and actions diagrams.
9) Classes. Class diagrams
10) Associative and aggregation relations
11) Attributes of classes
12) Define classes according to the variant.
13) Define the relations of inheritance, aggregation and association.
14) Build a class diagram .
15) Generate code in language C ++ for the diagram.
16) In the Rational Rose environment, create a class of matrices and in it the necessary fields and methods. It is mandatory to have two overloaded constructors, as well as methods for adding, multiplying, finding the determinant and inverting the matrix; in addition, matrix input and output methods (from a file or from visual components on a form). Add and multiply methods using overloaded + and * statements.
17) Generate code in C ++
18) In C ++ Builder, create software that implements all the classes described, and performs arbitrary actions on matrices.

LITERATURE

1. Васильев А. Программирование на С++ в примерах и задачах. М.: Эксмо, 2021. – 368 с.
2. Грекул В.И., Коровкина Н.Л., Левочкина Г.А. Проектирование информационных систем. М.: Юрайт, 2017, 386 с.
3. Кватрани Т. Rational Rose 2000 и UML. Визуальное моделирование. – М.: ДМК Пресс, 2001. – 176 с.
4. Кондаков А.И. САПР технологических процессов. – М.: Издательский центр «Академия», 2007. – 272 с.
5. Левинсон Дж. Тестирование ПО с помощью Visual Studio 2010. - М.: ЭКОМ-Паблишерз, 2012.
6. Липпман С., Лажойе Ж., Му Б. Язык программирования С++. Базовый курс. - М.: Вильямс, 2016. - 1120 с.
7. Литвиненко Н. А. Технология программирования на С++. – С.-Пб.: БХВ-Петербург, 2010. – 288с.
8. Мейер Дж. Д. Командная разработка с использованием Visual Studio Team Foundation Server / Дж. Д.Мейер, Дж. Тейлор, А. Макман, П. Бансод, К. Джонс - Изд. Корпорация Microsoft, 2007.
9. Мейерс С. Эффективный и современный С++. М.: Вильямс, 2016. - 304 с.
10. Орлов С.А. Программная инженерия. Технологии разработки програмного обеспечения. – С.-Пб.: Питер, 2016, 640 с.
11. Прата С. Язык программирования С++ (С++11). Лекции и упражнения, 6-е издание — М.: Вильямс, 2012. — 1248 с.
12. Резанова В.Г., Резанова Н.М. Програмне забезпечення для дослідження полімерних систем // К.: АртЕк. - 2020. 358 с.
13. Страуструп Б. Язык программирования С++. Краткий курс. 2-е издание. К.: Диалектика, 2019. – 369 с.

14. Страуструп Б. Язык программирования C++. Специальное издание М.: Бином, 2011. – 1136с
15. Токмаков Г.П. CASE-технологии проектирования информационных систем. Ульяновск: УлГТУ, 2018. - 224 с
16. Чистов Д.В. Проектирование информационніх систем. М.: Юрайт, 2016, 260 с.
17. Шалумов А.С., Никишкин С.И., Носков В.Н. Введение в CALS-технологии. Ковров: КГТА, 2002. – 137с.
18. Шилдт Г. C++. Базовый курс. – М.: Диалектика-Вильямс, 2018. – 624 с.
19. Шлее М. Qt 5.10. Профессиональное программирование на C++. С.-Пб.: БХВ-Петербург, 2018. – 1074 с.
20. Щербань В.Ю., Краснитський С.М., Резанова В.Г. Математичні моделі в САПР. Обрані розділи та приклади застосування. – К.:КНУТД, 2011. – 219 с.
21. V. Yu. Shcherban', V.G. Rezanova, T.I.Demkivska Programming of numerical methods and examples of practical application // К.: Education of Ukraine, 2021. – 150 p.
22. Stroustrup B. Programming: Principles and Practice Using C++ (2nd Edition). Addison-Wesley Professional, 2014. – 1312 p.

Резанова В.Г., Щербань В.Ю., Демківська Т.І.

ТЕХНОЛОГІЇ РОЗРОБКИ ПРОГРАМНИХ ПРОДУКТІВ

(англійською мовою)

Редактор Резанова В.Г.

Дизайн та верстка авторські

Формат 60*84/16
Папір офсетний 80гр/м2. Друк цифровий. Гарнітура Times New Roman
Умовн.-друк. арк. 14.75 Обл.- вид. арк. 7.80
Замовлення № 0202-0037
Підписано до друку 02.02.2022 р.

ТОВ «Видавничий дім «АртЕк»
04050, м. Київ, вул. Юрія Ільєнко, буд. 63
Тел.. 067 440 11 37 ph-artek@ukr.net
www.book-on-demand.com.ua
Свідоцтво про внесення суб'єкта видавничої прави
ДК №4779 від 15.10.14р.