SOFTWARE PROJECT MANAGEMENT

Lesson 1

INTRODUCTION TO SOFTWARE PROJECT MANAGEMENT

CONTENTS

1. Introduction to Subject

1.1. Area. Causes of the Emergence of the Software Project Management Discipline

Analysis of software development showed that a huge number of projects are being developed with deviations from the requirements document for the project, the timing of the project, and it always goes beyond the budget. There are many reasons for this.

Let's list some of them together with consequences.

- **The customer doesn't provide opportunities for development and application**, as a consequence, during the development process the customer has completely new views on what he/she would like to receive.
- **The customer doesn't understand the complexity** of the development, as a consequence, the set budget and deadlines are impossible.
- **The executor doesn't know the subject area and is not able to assess the complexity of the task**, as a consequence, there is a delay in timing, budget overrun, violation of both budget and timing, wrong decision or its complete absence.
- **Change of the existing state of things**, beginning with the emergence of new methods and technologies of development or replacement of the customer's technologies and ending with the collapse of the customer's company or the executor's company.

There are various combinations of these and other reasons that lead to a huge variety of explanations of the collapse of most projects (estimated to be up to 30 percent of failed projects and only about 30 percent of completed projects).

This led to the need of insurance of both the customer and the executor against unsuccessful completion, which is provided by computer engineering. The Software Project Management discipline is one of the components of com-

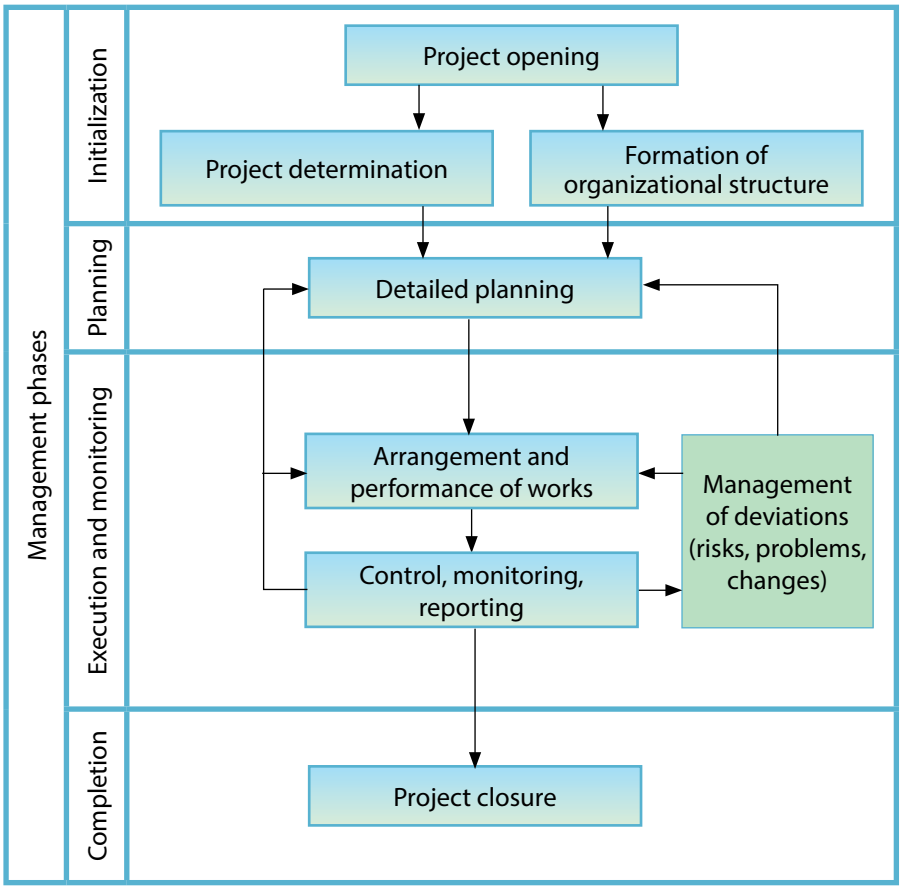

Figure 1

puter engineering. It deals with the arrangement of the software development process.

The software development process can be divided into several components called project lifecycle (Fig. 1).

The picture shows that phases of the lifecycle of the program are divided into non-conditional phases; each one has its own purpose. So, the phase of the project opening is often called the phase of the intentions of parties. The planning phase used to be called the development of the requirements document, which was immutable until the end of the project.

The figure shows that now the impact of the execution phase on the planning phase is foreseen, which in this case is already called replanning.

1.2. Gantt Charts

The development process can be divided into components, for each one we can determine the duration and sequence of the execution. In this case, we consider actions, the sequence of which is unalterable relative to each other. For example, it is impossible to develop a form for entering object properties unless we have determined what properties this object has.

Table 1

	Sat	Sun	Mon	Tue	Wed	Thu	Fri
Study the lesson material							
Write a program							
Debug the program							
Process the result and repeat the material							

Or let's consider a simpler example: you can't debug a part of the program that has not yet been written. To display successive components of any process, it is convenient to use Gantt charts (a researcher of the production processes of the early 20th century, the first chart was proposed in 1910 — 100 years ago!); they are horizontal bar charts, in which each new type of activities is displayed on a new line. For example, let's write a plan for doing home task (*Table 1*).

Gantt charts are used for planning. Each activity is represented by a bar. The project tasks are placed vertically. The beginning, end, and length of the timeline column correspond to the start, end, and duration of each task. You can specify the dependency between activities. The chart can be used to represent the current schedule status: a part of the rectangle that corresponds to the performed task is shaded. The vertical line marks the current moment. A table with a list of activities is attached to the chart. In this table, the rows correspond to a particular activity from the chart, and the columns contain additional information on the planned work.

1.3. What is a Project? What is a Software Project?

Software is a program or a group of programs, which is a final product of the software development project. Software development is called software engineering.

Software Engineering Institute, SEI gave the following definition of software: "Software is the programs, routines, and symbolic languages that control the functioning of the hardware." A **project** is a pre-planned sequence of actions.

Project Management Institute, PMI, gave the following definition of the project:

"A project is:

- a specific plan or project development;
- Scheme;
- Planned action:
  (a) part of the investigation formulated in a certain way;
  (b) a large action, usually supported by subsidies of some organizations or government;
  (c) challenge or problem that is usually put before a group of students and then is used in the classroom activities."

Another definition is related to the development management. This is a management definition:

Management is a practice of project implementation and project handling.

1.4. Software Engineering Definition

According to Barry Boehm: "**Software Engineering** is the practical application of scientific knowledge in the design and construction of computer programs and the associated documentation required to develop, operate, and maintain them." *(Boehm B. "Software Engineering Economics", Englewood Cliffs, NJ: Prentice Hall, 1976. — p.16).*

The IEEE Institute gives this definition of Engineering: "**Software Engineering** is the application of a systematic, disciplined, quantifiable approach to the development, operation, support, and maintenance of software." *(Institute of Electrical and Electronics Engineers. IEEE Std 610.12.1990 Standard Glossary of Software Engineering Terminology. "Software Engineering Collection". NY: Institute of Electrical and Electronics Engineers, 1983. — p.76).*

According to Stephen Schach: "Software engineering is a discipline whose aim is the production of fault-free software, which is delivered on time and within budget, and that satisfies the user's needs." *(Schach Stephen R. "Classical and Object-Oriented Software Engineering" 4th ed. Boston,MA: McGrawHill, 1999. — p.4).*

If you combine these definitions, you will get a definition that most fully reflects the essence:

Software engineering is a regulated, systematic methodology for the development, use, operation, support, and maintenance of software based on the practical application of scientific knowledge.

1.5. Project Definition

The definition given by James Lewis: "A **project** is a one-time job that has definite starting and ending points, clearly defined objectives, scope and a budget; differentiated from repetitive activities such as production, order processing and so on; a special activity with very tactical goals." *(Lewis James P. "Project Planning, Schedulling and Control: A Hands-On Guide to Bringing Projects in on Time and on Budget", rev ed. Chicago, IL: Irwin, 1995. — pp. 2–3).*

According to Harold Kerzner: "A **project** is a series of activities or tasks that have a specific objective to be completed within certain specifications, have defined start and end dates, have funding limits (if applicable), and consume resources (i.e., money, people, equipment)." *(Kerzner H. "Project Management: A System Approach to Planning, Schedulling and Controlling" 6th ed. NY: John Wiley & Sons, 1998 — p.2).*

These definitions come down to the resulting triangle "Time-Quality-Money" (common people like to say "Fast-Good-Cheap". There is a well-known joke on this topic, which is essentially reduced to: "Leave only two items.") During the project execution, the problem of keeping the cost within specified limits, maintaining deadlines at a certain level of quality is solved. The project management task is to balance these three components: cost, schedule, and qualitative result. But this usually doesn't happen. The schedule, budget, and quality don't stand at the required level. Therefore, project managers have to choose only one or two parameters of the triangle as a target. As it was already mentioned above, in professional slang this action is known as "Fast-Good-Cheap. Pick two."

PMI gives the following definition of the project: "A **project** is a temporary endeavor undertaken to create a unique product or service with defined start and end dates that differs from continuing and repetitive actions and requires progressive improvement of characteristics."

These project definitions have common features:

- **Objective.** The objective or a set of objectives of the project have to be clearly defined. After the project is completed, a certain result should be obtained. If the project involves achieving several objectives, they should be interrelated and consistent.
- **The moment of the beginning and the end of the project**. Project has duration. It has a well-defined beginning and end of the action associated with the timeline of any dates. Software support is not a project and is usually an action prolonged beyond the project, but it can be included in the project (for example, as separate versions).

- **Uniqueness.** A project is a one-time entity that doesn't repeat itself, when a similar project is repeated. But repetitive work can also be a project. Building a house is usually defined as a project, despite the fact that the contractors have already built dozens of buildings. Although the sample and the process basically match and are executed according to a template, there are some differences in each house (it is a collective, placement, materials, etc.). Otherwise, it is a flow line, where identical parts are performed in a similar way. The same is true for software development professionals: they never create an identical software system, although they can copy it, compose from existing solutions, and transfer in an arbitrary manner.
- **Limitations.** The project has limitations on the cost, development schedule, and execution quality. These limitations form a triangle, which should be balanced and managed to achieve success." *(Project Management Institute. "A Guide to the Project Management Body of Knowledge" Sylva, NC: PMI Publication Division. 1996. — p.167).*

So, a project in terms of software is basically the following: A **project** is a unique temporary action with certain start and end dates aimed at achieving one or more goals under cost, schedule, and quality limitations.

1.6. What is a Project Management?

The Project Management Institute (PMI) defines project management as "a set of proven principles, methods and techniques for the effective planning, scheduling, controlling

and tracking of deliverable-oriented work (results) that help to establish a sound historical basis for future planning of projects."

Kerzner defines project management as "the planning, organizing, directing and controlling of company resources for a relatively short-time objective that has been established to complete specific goals and objectives."

Note general thoughts in these definitions:

- **Management.** Skills in project management are a part of the overall management skills.
- **Skills.** Skills in the field of project management are the use of management competence to achieve the planned result. They include planning and arrangement of the production process, scheduling, control, management, and analysis.

So, by summarizing, we get the software project **management definition**: Software project management is a specialization of general management that determines the application of guiding standard skills of planning, staffing, arrangement, as well as management and control to achieve a pre-determined project goal.

1.7. What is a Team Development?

There was a time when a programmer defined, designed, wrote, and checked his/her work him/herself. The environment had contributed to that: simple single-user nature of tools used, small size of applications being developed. This made it possible to consider software development as an individual activity. But, at some point, the times have changed: "Software development has become a team sport" (*G. Buch, 1998*).

What are team development, team, and group? A group can consist of any number of people interacting with each other, psychologically accepting other members of the group, and treating themselves as a part of the group. A **team** is a group of members who influence each other to achieve a common goal. The main difference between a team and a group is the result of effective interaction between people on the basis of common aspirations and values, as well as complementary skills that leads to the fact that the team's total effort far exceeds the sum of the efforts of its individual members. As in sports, teamwork is crucial for solving problems of competitiveness in the common market, where an individual skill is less important than a high level of collective actions. In the modern world of information technologies, team is the only survival condition, a rule, not an exception. A characteristic feature of the team is a wide range of powers or decision-making powers of each team member.

The software development team should have the following skills:

- Skill to correctly understand the problem, the solution of which is intended to solve the software being developed;
- Ability to identify requirements imposed on the software being developed through the communication with the system user and other stakeholders;
- Skill to transform understanding of a problem and needs of clients into an initial system definition that will meet these needs;
- Skill to manage the scale of the project;
- Skill to refine the definition of the system to the level of detail suitable for design and implementation;

- Ability to assess the correctness of the software being developed, to conduct its verification, and manage changes.

Team development is a process of software development that implements:

- Team management;
- Managing tasks of an individual employee and a team as a whole;
- Guidance on which components to implement;
- Provides criteria for tracking and measuring products and project functioning.

An important fact in team development is that team members have differences in professional skills and abilities. A balance and a variety of skills are the two most important components of an excellent team. This makes a team the team. Some *team members* are able to work effectively with customers, others have programming skills, the third ones are able to test programs, and the forth ones are specialists in system design and architecture.

1.8. Analysis of Problems of Solo and Team Software Development

Currently, both solo and team development of software are successfully used. But today, the vast majority of large projects are the result of collective work. At the same time, there are many freelancers and programmers who combine work for a company and solo development. For example, Ethan Nichols, the engineer of the Sun Microsystems company, because of dire financial constraints (recently became a father) had to look for sources of income, and was inspired by a success story

of Steve Demeter who is an independent developer of iPhone applications. Ethan Nichols developed the Trism puzzle, which brought him about 600,000 dollars in revenue.

Another example of one person developer is a creator of Bit Torrent, American programmer, Bram Cohen. Let's try to analyze the features of solo and team software development:

Table 2

Solo Software Development	Team Software Development
You should find customers yourself and negotiate with them on the volume, quality, deadline, and cost of work	Specially trained employees of the company are engaged professionally in search for customers and development maintenance
All decisions are made alone (programming language, architecture, application design, etc.) based on the previous experience of a particular developer	All decisions (programming language, architecture, application design, etc.) are made taking into account the opinion of several developers based on previous experience
The customer communicates directly with the developer, which eliminates any misinterpretations	The customer not always communicates directly with the developer, which can lead to some misinterpretations
The developer is free to choose the customer, time, and place of work	The developer is limited to working in the company's projects according to the company's staff schedule
The developer is more prone to risks of non-payment (incomplete payment) of the work performed	The developer's income for the work done, as a rule, is guaranteed by the company

1.9. Analysis of Subject Area Terms

- Process;
- Project;
- Staff;
- Product;
- Quality.

The software development process is a way of how we create software, what sequence of actions we perform, how we interact with other team members, what norms and rules we adhere to in our work, how our project interacts with the outside world. The basis of the success of software development is the methodologically correct alignment of the development process, its understanding, and improvement.

The process of software creation is a lot of different activities, methods, techniques, such as developing a requirements document, application architecture, coding, testing, writing documentation, and so on.

To date, there is no universal software development process, following which we can guarantee to get a quality product within the agreed deadlines. Each specific development that is carried out by some development team has a large number of individual features. But before starting the project, it's necessary to plan the work process by defining roles and responsibilities in the team, plans, and deadlines for performing intermediate and final versions of the product.

The result of the software development **process** and the outcome of the project is a **product** ready for delivery (**final product**). It includes the body, that is, the source code in the form of components that can be compiled and executed, the reference guide, and additional component parts of the delivery. Before the product becomes ready for delivery, it passes through the stage of the work product. **A work product** is an intermediate result of the software development process that helps to identify, plan, and evaluate different parts of the result. Intermediate results help managers of different levels track the project implementa-

tion process, and the customer gets an opportunity to get acquainted with the results long before the end of the project. In the daily work, project developers get a simple and effective way of exchanging working information, that is, the exchange of the results.

Software development refers to **project activities**. Projects are divided into industrials and creative, according to different management principals. So, software development refers to creative projects.

Industrial projects often combine a large number of different organizations with low uniqueness of the works themselves, for example, constructing a many-storeyed house. They include various international projects and not only industrial ones, but educational, cultural, and so on. The main task in managing such projects is to foresee everything, control everyone, forget nothing, pull everything together, and achieve coordinated interaction.

Creative projects are characterized by an absolute novelty of the idea: a new type of service, completely new software product that has no analogues in the market, it also includes projects in the field of art and science. Every beginning business usually becomes such a creative project. The novelty in such projects is not absolute, because this may already has taken place, but for the project team, it happens for the first time. Here, it is said about a huge volume of novelty for the people who embody this project. Software development projects absorb parts of industrial projects, and, undoubtedly, parts of creative projects by being between these poles. They are often complex because they are voluminous and are at the border between different disciplines, for example, target business,

where the software product should be applied, and complex, non-trivial programming. The development of unique equipment is often added to software projects. On the other hand, since programming actively penetrates into different areas of human activity, absolutely new and unique software products are created, and their development and promotion have all the features of creative projects.

At the development stage, regular measurements of the developed product are made to determine compliance with requirements. Any inconsistencies should be considered as flaws, that is, lack of quality.

The product **quality** is defined in the **ISO 9000:2005** standard as the degree of compliance of its characteristics with the requirements. A quality software product can't be created without a quality development process.

Methods of software quality assurance are:

- Creation and improvement of the quality software development process;
- Quality assurance of the code by fulfilling the standards of the code design in the project and controlling the compliance with these standards. This includes rules for creating variable identifiers, methods and class names, for commenting, etc. The code quality is also ensured by refactoring, that is, rewriting the code not in order to add a new functionality, but to improve its structure;
- Testing is the most common method today, without which no product is produced today.

If software is created by team efforts, then high quality and productivity appear when **the staff**, that is team members,

are effectively involved in the common cause and keep being motivated, and the entire team is effective. To manage a team, it is necessary to move to the human relations field. Typically, the hierarchical structure of a team is used.

The team is headed by the project manager, who reports to the work-stream leader (chief of department). A typical team consists of developers, database administrators, software testers, and possibly other professionals. A large project can include module leaders, each of them having several developers subordinated to them. The goal of the project manager is to get a self-confident team. The following human factors should be taken into account:

- Qualification, training, and experience of team members;
- Personal aspirations and career development of team members;
- Needs for mentoring and development.

Team communication is very important. The team, which members will work together for several months in order to achieve a common goal, should be a single whole, the full mutual understanding should be between its members. To support the team members' awareness of project progress and its problems, the project managers can use the following methods:

- Arrangement of bulletin boards for messages, project reports;
- Electronic mailing on the project;
- Internet (intranet) site for publishing documents associated with the project;
- Project meetings.

In summary, let's define the terms:

- **Project** is an organizational entity that is used to manage software development. As a result of the project, a software product appears.
- **Staff** includes developers, architects, testers, managers, users, customers, and all other stakeholders; this is the main driving force of the software project.
- **Quality** is a software characteristic as a degree of its compliance with requirements. Compliance with requirements assumes that requirements should be clearly defined so that they can't be understood and interpreted incorrectly.
- **Product** is artifacts (from the Latin artefactum meaning something artificially made), created in the process of project activity. These include models, program texts, executable files, and documentation.
- Software creation **process** is a definition of the full set of activities required to convert the user's requirements into a product.

1.10. Project Characteristics

- Project type;
- Project objective;
- Quality requirements;
- Budget requirements;
- Deadline requirements.

The software development project has four characteristic features:

1. Focus on achieving specific goals;
2. Coordinated implementation of interrelated actions;

3. Limited duration with a certain beginning and end and limitations in resources used (qualitatively-quickly-cheap);
4. Originality and uniqueness (at least partial).

These are features that distinguish projects from other types of human activity.

Projects are Aimed at Achieving Specific Goals

All efforts to plan and implement the project are undertaken in order to achieve its goals and to obtain specific results. Specifically goals are the driving force of the project. Therefore, in the project management, it is very important to accurately identify and formulate goals, from the top level to the smallest details of aims and objectives.

Coordinated Implementation of Interrelated Actions

Projects always have components and staff that implements specific tasks. Some parts of the project are dependent; staff actions are not isolated. It is necessary to arrange their interaction in order to properly combine them and get the final result. The synchronization violation of execution of different parts jeopardizes the implementation of the project. A project is a dynamic system that consists of interrelated parts and requires special approaches to management.

Limited Funding

Money is always not enough. Time is always limited. In addition, the project is limited to the framework of state laws. It is necessary to remember about staff who have moral principles that can also be a restriction. And, in the end, the project is limited by the laws of nature. Here are the main limitations of the project.

Originality and Uniqueness

Because of constant change in the conditions of the project implementation, planning should be so flexible that even the original goals and methods of achieving them can change. It is impossible to predict all the changes that will accompany the project execution.

Lerman's law says: "**Any technical problem can be overcome given enough time and money**."

The law has an effect (Lerman): "**You are never given enough time or money.**"

The methodology of managing project based activities was developed precisely to overcome the problem formulated in the effect of the Lerman's law. Typically, the project manager understands his/her main task in the project implementation as ensuring the works execution. A more experienced project manager will more accurately formulate the definition of the main task of the project manager: "Ensure the works execution on time, within the allocated funds and according to the requirements document." Specifically these three components: time, budget, and quality of work are under the constant attention of the project manager. Therefore, we will expand the definition of the project: A project is an activity aimed at achieving certain goals with the highest possible efficiency under the specified limitations of time, money, and the quality of the final product.

In order to cope with time constraints, methods for constructing and monitoring work schedules are used. To manage money constraints, methods of forming a financial plan (budget) of the project are used, and, as the work is carried

out, the budget compliance is monitored so to keep costs from getting out of control. For example:

Table 3

# of the stage of work	Description of the stage of work	Complexity of the stage of work	Labor intensiveness of the stage of work (based on data from previous projects), plan	Labor intensiveness of the stage of work, fact	Cost of the stage of work, plan	Cost of the stage of work, fact
1	Update personal data	high	8 man-hours			
2	Add address	average	6 man-hours			
3		average	6 man-hours			
4		low	4 man-hours			
5		average	6 man-hours			
	...					
	Total:					

It is very difficult to develop software, and it is even more difficult to develop high-quality software and complete the work on time. According to Jet Info Online, in the late 60s of the last century, in the USA, there was a software crisis. This included a backlog of schedules; cost overruns in large projects. In addition, the developed software didn't have contractual functionality, had low productivity, and its quality didn't satisfy consumers.

According to the results of investigations conducted in 1995 by Standish Group, which analyzed the work of 364 American companies and the results of more than 23 thousand projects associated with software development, they looked like this:

- only **16.2%** of software projects were completed on time, they implemented all required functions and capabilities, and didn't exceed the planned budget;
- **52.7%** of projects were completed late, the required functions were not fully implemented, costs exceeded the planned budget;
- **31.1%** of projects were cancelled before completion;
- For the last two categories of projects, the average project duration was exceeded by **122%**, and the budget was exceeded by **89%**.

In 1998, the percentage of the three listed categories of projects only slightly changed for the better (26%, 46%, and 28% respectively).

According to leading analysts, in recent years, the percentage of the three listed categories of projects was slightly changed for the better mainly due to reduction of the scope of projects, but not due to increase of controllability and design quality.

Among reasons for these failures, according to experts, we can name the following:

- insufficiently clear and complete wording of the software requirements;
- there is no constant contact with the customer;
- lack of required resources;
- disgusting planning and incompetent project management;
- frequent changes in requirements, specifications, conditions;
- imperfection, insufficient knowledge or novelty of the technology used;
- lack of support from the top management;
- low qualification of developers, lack of necessary experience.

1.11. Costs Associated with the Project

Naturally, capital investments are required to develop software. What are they used for?

- Salary for developers;
- Leasing of premises;
- Amortization of development tools (computers, networks, software products);
- Payment for energy resources;
- Payment of management staff.

According to the existing terminology, costs are divided into direct and indirect.

Direct costs include costs that are directly transferred to the cost of a particular product or service. They include material, salary to major producers (programmers, testers, chiefs of departments and parts of the project, researchers and consultants, and so on), social investments (pension fund, social insurance fund, employment fund, and compulsory health insurance fund), electricity, costs for the previous period charged to the cost of production of the reporting period, and other direct costs. These costs and indirect costs characterize the group of production costs by the way they are included in the cost of production: they are put directly into the production account.

Indirect costs are costs that are difficult to relate to any particular activity or project, but nevertheless they are required for normal functioning of the organization and successful performance of its tasks: administrative salaries, advertising, general utilities (telephone, gas, electricity, elevator, antenna, and so on).

2. Models and Methodology of the Development Process

2.1. Overview of the Development Process Models and Methodologies

Historically, there are several models of developing software systems. The first, which is called classic, came from engineering practice, and corresponds to processes that came from the design of systems. All the works that make up the classic model are executed sequentially, one by one in a predetermined order. Unfortunately, such a well-understood model almost never works; new requirements and clarifications constantly arise during the development process, the rejection of them leads to a conflict with the customer. Separation of the processes of coding and testing leads to hard-to-redesign code. Later new, iteration types appeared which allowed making corrections and refinements many times, and including testing in the development process from its very beginning.

2.2. Phases of Process

In any case, regardless of the model, the development process can be divided into several separate components, which are called phases of the design process. The sequence of their implementation alone determines this or that model (Fig. 2).

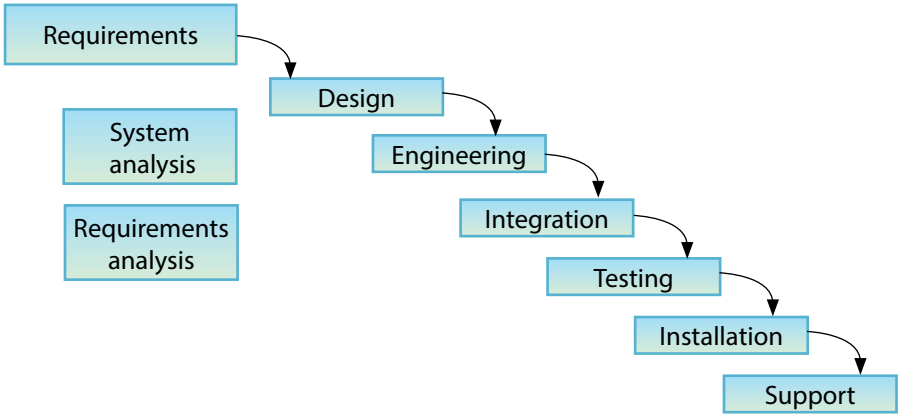

Figure 2. **Development process phases**

Requirements

This phase is required to find out what the customer wants and what to do to make it work. The system being developed is considered as a single whole and is determined as system functionality, characteristics of the equipment, on which the system will operate, the physical location of the system, the first approximation according to development costs definition. This phase is usually divided into two components: system analysis and requirements analysis.

- **System analysis.** This part of the requirements definition phase defines the role of each element in the computer system and describes the interaction of elements with each other. It begins by defining the requirements for each element of the system and combining these requirements into the program part of the system. When a software interface is formed with other parts of a system, that is, equipment, people, databases, etc., the need for a system approach is most clearly manifested. At the same time, the task of planning a software project is

being solved, during which the volume and risk of each of design works are determined, the required labor costs are calculated, work tasks and a work schedule are shaped.

- **Requirements analysis.** This part of the requirements definition phase refers to the program component, that is, software. When analyzing requirements, software functions, its characteristics and interface are refined and detailed.

All the generated definitions are saved in a document called *analysis specification*. This part of the requirements definition phase completes the project planning.

Design

The design phase determines what will become the basis of development, and will allow displaying:

- software architecture;
- software modular composition;
- software algorithmic structure;
- data structure;
- input and output interfaces (input forms and reports).

The initial design data is recorded as an **analysis specification.** During design, software requirements are converted into a number of design views (for example, a set of UML-diagrams). When designing software, the focus is on the quality of the software product being created. It is important to correctly identify software subsystems and the ways they interact.

Engineering (Implementation, Coding)

This phase is usually called programming, although it solves only the role of translating what we *already* know into any programming language. All subsystems defined at the design

stage are developed in parallel (if there are enough resources: a company of one person can't develop subsystems in parallel).

Integration

If subsystem interfaces are developed in accordance with specifications, the subsystems don't have a side effect on each other, and when developing the specification, all requirements are taken into account, therefore, the process of assembling or combining the subsystems doesn't cause any difficulties. It never happens. First, not all subsystems are developed simultaneously. Secondly, errors in one system often lead to a malfunctioning of the group of dependent systems. Therefore, when assembling, two approaches are used: the "big bang" method (now, everything will work!) and the method of serial connection. None of them allows you to determine with absolute accuracy the subsystem leading to errors. When using the "big bang" method, all subsystems are turned on simultaneously, the number of errors is off scale, and one usually tries to solve this problem by the method of serial connection. However, there are subsystems that depend on still unconnected systems so that when switching on, they stop performing their functions, and it is difficult to determine the root case. Therefore, the next phase begins.

Testing and Debugging (Verification)

At the testing phase, flaws are searched for. To facilitate the search for flaws, test examples are developed. Successful or non-successful execution of an example is called a test case. In theory, it is impossible to check all branches of the program code, therefore, in practice, number of test cases and time for testing are limited. Testing is divided into alpha-testing (the developer him/herself) and beta-testing (the customer

or third parties who are neither a customer nor a developer). The flaws found are sent for revision.

Installation

This phase reveals the shortcomings and inconsistencies associated with the work of the software being developed on the equipment and in the customer's environment. A variety of inconsistencies are possible. Let's list some of them:

- In the program, the path that is absent on the client's computer (or host) is hardcoded.
- The program uses a service that is absent on the customer's network (for example, you bring the program written in C#, and the customer doesn't have it or he/she has an obsolete .Net Framework version).
- The customer's computer and operating system meet the requirements of the specification, but because of the computer's overloading with various background processes, your software can't support the required characteristics of interactive work (a case from the author's practice).

Support

When the customer uses the developed software, different situations can arise that can be solved only together with the developer:

- The equipment is changed (printer, monitor, mouse, any adapter, hard drive, processor, and so on).
- The requirements for the generated reports (new tax forms) or interface (remove the squeaker) are changed.
- The need to update the basic directories that can be adjusted only by the developer.

All these issues not resolved in time harm the prestige of the company and the number of possible customers (negative publicity). Therefore, the performer introduces a support phase into the development plan. Support can be carried out by the confidant, but not by the performer (for example, the well-known 1C system).

In addition to the described phases, there may be others, but these seven phases are present in almost all development models. Let's list the phases once again:

- Requirements;
- Design;
- Engineering;
- Integration;
- Testing;
- Installation;
- Support.

The presence and mutual arrangement of these phases, as well as the addition of some new phases or functions attached to the phase, the interaction of the team during the execution of each phase: all this determines the so-called *development model.*

Let's try to evaluate the advantages and disadvantages of the most common models.

2.3. Waterfall model

It is also called cascade or classical model. Winston Royce is an author of this model that represents a rigorous and understandable construction. There is no return to the previous phase. The customer sees the finished project only at the end

of the design, when the customer can change the idea of nature of the software being purchased. The figure shows that it differs from the previous ones by the phase names, some phases are assembled into one, and some of them are divided into components.

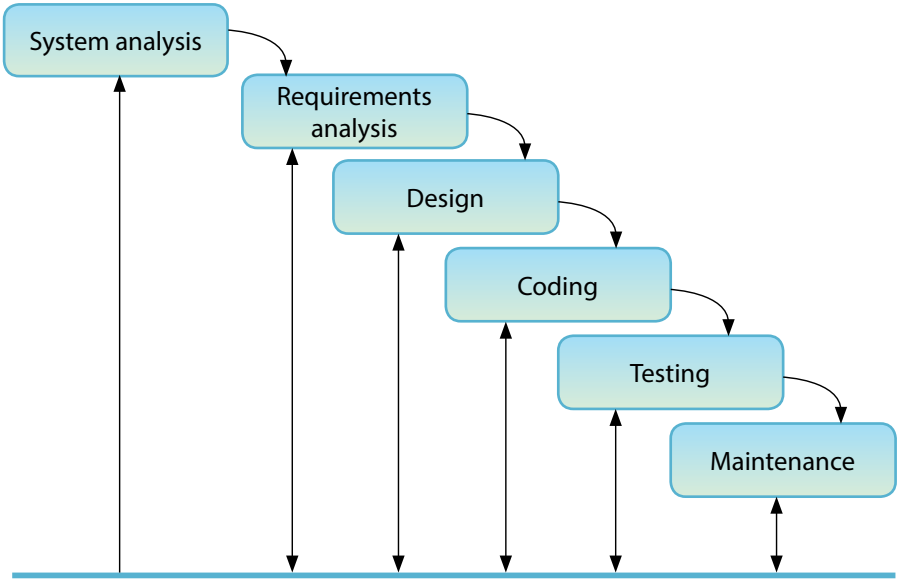

Figure 3. **Cascade model of Winston Royce**

This is an outdated, but still often used in practice model. It provides a plan and a timeline for all phases of the project, streamlines the design process. It should be emphasized that real projects often require a change in the standard sequence of steps. Nevertheless, due to the fact that the development life cycle is based on the exact formulation of the initial software requirements, most customers recognize this model, although in practice, at the beginning of the project, the customer can define his/her requirements only partially. Such a model has the largest number of unfinished projects.

2.4. Prototyping

It is a methodology of working with the customer, which is successfully used in a wide variety of project models, but not a model.

If the customer can't immediately formulate the requirements for the software being developed or the developer is not sure of the normal software functioning in the real customer's environment, it is customary to use **prototyping** through the implementation of the dialogue with the user or for the effectiveness of the algorithm being implemented.

The main goal of prototyping is to get rid of uncertainties in customer requirements.

Prototyping is the use of a simplified model instead of the required software product.

This model takes one of three forms:

1. **Paper draft or PC-based draft** (a human-machine dialogue is depicted or drawn);
2. **Working draft** (some of the required functions are performed);
3. **Existing program** (the characteristics of which must be changed after).

Prototyping is based on repeated iterations of creating an increasingly accurate draft, in which both the customer and the developer participate.

In principle, prototyping can be used in any software development models, as a part of the requirements definition phase.

Advantages of prototyping: prototyping provides the most complete definition of software requirements.

Disadvantages:

- the customer can mistake the draft for a system being developed;
- the developer can mistake the draft for a system being developed.

The figure shows the generalized model of working with a draft.

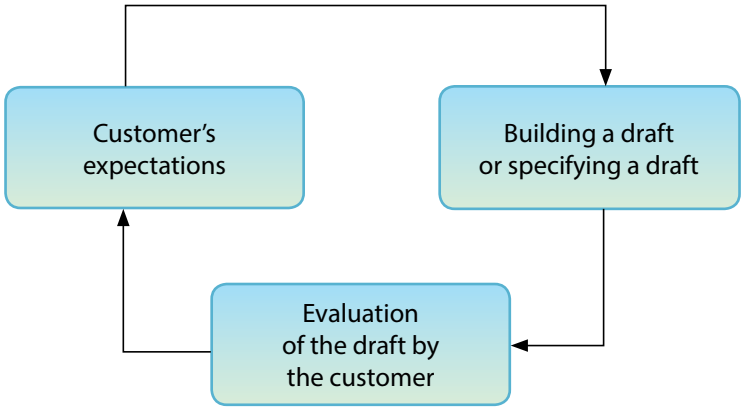

Figure 4

2.5. Spiral model

The spiral model was proposed in 1988 by Barry Boehm. The spiral model is based on:

- prototyping;
- a strongly modified waterfall model, twisted into an evolutionary spiral;
- new component is a risks analysis.

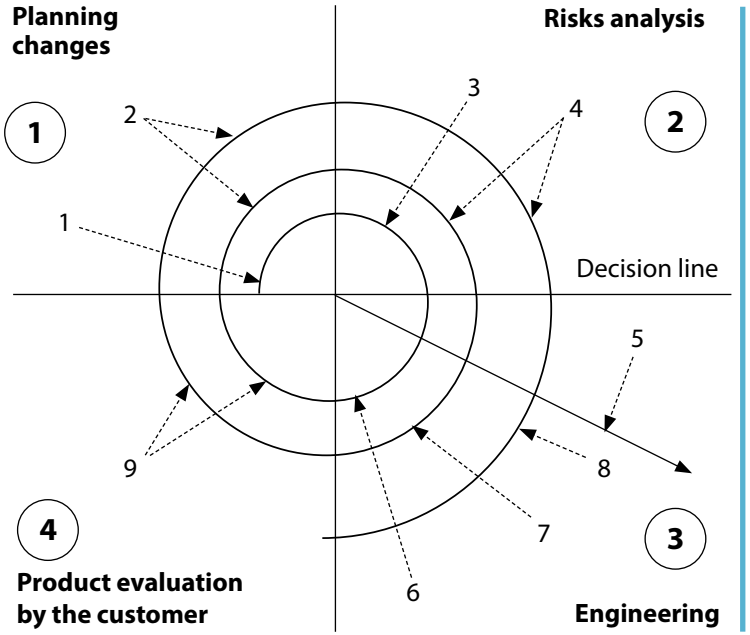

Figure 5. **Spiral model***: 1 — initial collection of requirements and project planning; 2 — taking into account the recommendations of the customer on the introduction of changes; 3 — risks analysis based on primary requirements; 4 — risks analysis based on the customer's reaction to the previous stage; 5 — transition to the complex system; 6 — system initial draft; 7 — next level of the system draft; 8 — next implementation; 9 — customer evaluation of the implementation.*

The spiral model includes four stages, arranged in four quadrants in a spiral:

1. **Planning.** At this stage, the project objectives or the next *volution* of development, possible solutions, and all kinds of limitations are determined.
2. **Risk analysis.** Options analysis and the choice: to carry out further development or the risk exceeds the possible positive effect.

3. **Design.** Transition to the next level of development.
4. **Evaluation.** This stage includes testing and customer evaluation of the current development results.

The spiral model refers to evolutionary models, the characteristic feature of which is the multiple *mini* life cycle with the output to a new level when ending, a new *spiral volution* with the increase in the number of functions available to the customer and the product quality. The customer is often involved in the development process, and after each cycle, he/she can see and evaluate the system state.

Advantages:

1. Close to the reality, it displays the process of software development in an evolutionary form;
2. Allows you to explicitly take into account risks at each evolutional *volution* of the development;
3. Includes a system approach (waterfall model) into the iterative development model;
4. Uses prototyping to reduce the risk and improve the software product.

Disadvantages:

1. There is no sufficient statistics on the effectiveness of the model;
2. Requirements to the customer are increased;
3. There are problems with the control and management of the development time (the customer doesn't see when the final product will be ready; the developer doesn't know whether he/she will have a job in the future).

2.6. Component-Based Model

Component-based model is a spiral model development. It is also based on the evolutionary design strategy. Here, the content of the design quadrant is clarified: the fact that under modern conditions a new development should be based on the reuse of the existing software components is reflected.

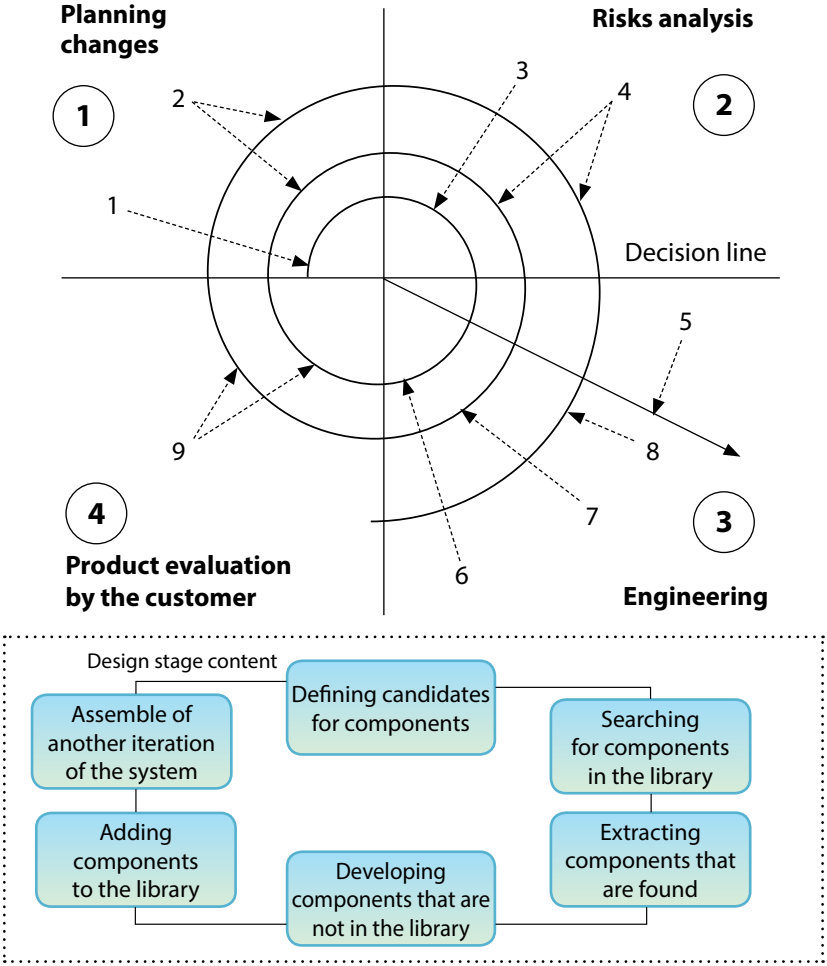

Figure 6. **Component-based model**

2.7. Iterative Model

In principal, the spiral model and all evolutionary models are a kind of iterative model. The iterative model points to the repetition of a group of actions and introduces risk analysis into the development process after each iteration. In addition to spiral model, there are many techniques, the essence of which comes down to minimizing risks through performing in the form of a set of short cycles.

Let's list some of them.

Agile

Agile is a family of development methods in software development, which began with the spiral model (1985), that is, a many times repeated, short, but full development cycle. The next stage in the development of Agile was the Scrum methodology based on 30-day cycles, self-organization of development teams, and daily meetings (1990-1999). Another type of self-adapting iterative development process was proposed by the authors of RAD (Rapid Application Development) in 1994. The description of the XP (eXtreme Programming) methodology by Kent Beck is referred to the end of 1996; it is a variant of iterative development process with hypertrophied testing and code transfer between team members (shared code ownership). And, finally, the last of the iterative methods included in the Agile is Feature Driven Development (FDD) proposed in 1997 and immediately saved a hopeless project. The authors of the listed methodologies have published their generalized features in **Agile Manifesto**, a main document for the agile processes, allowing changes in

requirements and even development aims. **Agile Manifesto** defines values and principles that are applied by the most successful teams.

This approach includes values, the importance of which is reduced to zero in unsuccessful teams:

- Processes and tools are not as important as individuals and their interactions;
- Running software is more important than full documentation;
- Responding to change is more important than rigorous execution of the plan. Cooperation with the customer is more important than obligations under the contract.

Here are **principles that are included in Agile Manifesto:**

- Daily communication between the customer and the developer during the project implementation;
- Changes are encouraged even at the end of the development, which increases the competitiveness of the obtained product;
- Frequent delivery of working software (maybe every month or week);
- Better customer satisfaction due to the earliest possible and uninterrupted supply of valuable software;
- All project stakeholders, sponsors, developers, and users, should be able to maintain a sustainable pace for a long period of time;
- Simplicity of development, that is, it is necessary to develop the really required parts only;

- The project should be done by motivated individuals who need to be provided with the required working conditions, support, and trust;
- The most recommended way of communication is a personal conversation (face to face);
- Working software is the best measure of progress;
- Continuous attention is paid to technical excellence and good design;
- Frequent or permanent adaptation (improvement of efficiency) to changing conditions;
- The best architecture, requirements, and design emerge from self-organizing teams.

Next, let's consider in detail some Agile-methodologies.

Scrum

Scrum is a methodology of project management for agile software development. The authors are Jeff Sutherland and Ken Schwaber. This methodology is based on the analysis of practices of Japanese industrial companies and daily meetings of the Borland company.

The development process is divided into 30-day cycles (sprints). At the beginning of the development process, all requirements and objectives are determined, and the ones of higher priority are chosen for the first cycle.

At the end of each cycle, at the meeting (*review*), the team provides the management and customer with a ready-to-use product which has only the functionality implemented that was planned for this sprint rather than all customer requirements. The customer, together with the project manager, examines

the current state of the product and evaluates it. Based on the results, the receiving party draws conclusions concerning the development of the software, how it can be improved, and what to do in the next sprint.

The management does not interfere in the work of the team throughout the cycle (*complete trust*) and does not add additional tasks to the team. To reconcile the team's interaction and to strictly comply with the scrum rules, a scrum master is chosen. Meetings-reports of each team member are held daily by the team in order to decide how to redistribute efforts to timely solve the set tasks, which is a deep feedback and allows controlling the development process qualitatively.

From the point of view of a methodology, Scrum includes a set of methods, predefined roles, and artifacts (in this case, control points that determine the development direction, for example, a proven inapplicability of one approach can be an artifact). The obtained artifacts are discussed at the meetings.

At the end of the sprint, a *demonstration* and a *retrospective meeting* (a joint meeting of the team, management, and customer) are held.

You will find out more about Scrum in the following lessons.

Scrum terminology. Roles:

- Product Owner;
- Scrum Master;
- Scrum Team;
- Users;
- Stakeholders;

- Consulting Experts.

Methods:

- Planning Meeting;
- Sprint Abnormal Termination;
- Daily Scrum;
- Demo Meeting;
- Retrospective Meeting.

Artifacts:

- Product backlog;
- Sprint Backlog.

XP

Do you recognize? This is an outdated version of Windows!:-) Of course, not. The abbreviation originated from **Extreme Programming.** The whole team of programmers sits down in a hammock or gets on skis or skateboards, and ... No. The idea of extreme programming is to bring all the components of Agile programming to the point of absurdity. If the development cycle should be short, then we select as short a cycle as possible. If you need to meet the customer often, let's include the representative of the customer in the team.

If testing needs to be done as often as possible, let's develop the product function only after developing a test for it. If it is unnecessary to overload the product with extra functionality, let's define through the customer only those functions that are extremely necessary. If the customer doubts whether further development is needed, let's immediately stop it. If it is difficult for a programmer to master another's code, we will make so that the code is developed by two programmers (pair program-

ming, change of programmers in pairs during the project), but not by one programmer. If a person is not a horse, and we want maximum productivity and concentration throughout the development process, there will be a 40-hour work week.

RUP

Rational Unified Process (RUP) is a software development method developed by Rational Software, which provided it with a full set of tools. First of all, RUP is an iterative development model. At the end of each iteration (from 2 to 6 weeks), developers should achieve the goals for this iteration, create or complete project artifacts, and get an intermediate functional software version. This development method allows you to quickly respond to changed requirements, detect and eliminate risks at the early stages of the project, as well as get a highly effective control over the quality of product being created. The development iteration is divided into four phases:

1. Inception;
2. Elaboration;
3. Construction;
4. Transition.

Each iteration ends with a checkpoint that allows you to evaluate the success of this iteration.

During the development process, a lot of processes are controlled (there is an appropriate tool for each):

- Business Modeling;
- Requirements;
- Analysis and design;
- Implementation;

- Testing;
- Deployment;
- Configuration and Change Management;
- Project management;
- Environment.

MSF

Microsoft Solutions Framework (**MSF**) is a Microsoft software development methodology, in fact, a variant of the spiral model.

The spiral is divided into phases by milestones:

- **1 phase** — Envisioning
  1 milestone — Vision approved.
- **2 phase** — Planning
  2 milestone — Project Plans approved.
- **3 phase** — Developing
  3 milestone — Project Scope Approved.
- **4 phase** — Stabilizing (testing and preparation for deployment)
  4 milestone — Release Readiness Approved.
- **5 phase** — Deploying
  5 milestone — Deployment Complete.

Microsoft's extensive hands-on experience is used. Includes a description of principles of people and workflow management.

MSF uses two types of models:

- Team Model;
- Process Model.

MSF includes three disciplines:

- Project Management;
- Risk Management;
- Readiness Management.

Microsoft does not use the MSF method in the *pure* version in its IT projects.

2.8. Analysis of Existing Models and Methods

In recent years, the success of the project has become increasingly important both for the creators and for the project customers. Practice shows that it is impossible to plan and implement a long-term project without making changes throughout the project. Therefore, new methodologies arise, some of them are suitable for small teams of developers and small customers because risks are not so great, there are no restrictions when developing, the development goals are not clearly defined (spiral model and XP, perhaps Scrum). In practice, most iterative methods compete with each other, but some of them (RUP, MSF) allows the development of large projects with specific goals, development time, and budget, which is applicable to large teams of developers, since the amount of work is predetermined, and, accordingly, employment is provided. Accordingly, the first iterative development methods were called **lightweight**, and the second ones that require the full determination of all characteristic of the project at the initial stage were called **heavyweight**.

3. Quality Control

3.1. Quality Management

For any enterprise, the main source of existence is the successful implementation of a quality product. When we talk about the quality of a product, we always mean the consumer, who determines the acceptable properties of the product. In the market economy conditions, the quality of a product is a task number one, since the competitiveness of the company is determined by:

- Product price level;
- Product quality level.

Moreover, the quality of a product is gradually coming to the forefront. There is a large number of parameters that determine the quality of goods, therefore, in management, there was a need for development of such a direction as quality management.

Quality management is an activity of managing all stages of the product life cycle, as well as interaction with the external environment.

At the moment, many (several hundreds) quality control systems have been developed all over the world. The overall objective of these systems is to ensure the compliance of products with the requirements of consumers.

The most famous and widely used standard for the arrangement of quality control processes is a series of the ISO 9000 standards.

For the software development, the ISO 9000-3 manual is used, which describes the development process to achieve the required level of quality.

Currently, it is recommended to use the standard of version of the year 2014. The main focus is now on process management. The standard still doesn't contain parts for software developers.

The implementation of ISO 9000 standards creates a basis for independent certification of products, which is aimed at confirming the level of product quality that determines its competitive capabilities.

The disadvantage of the ISO 9000 standard is a fact that it is impossible to adequately evaluate the quality level of the software development process for the proposed model of quality assessment.

Alternative quality model (USA): **CMM — SEI (Software Engineering Institute)**. This quality model was developed in the Institute of Software Engineering for use by state, in particular, military organizations when placing orders for software development.

Now the CMM-SEI model is used for analysis and certification of software development processes by the companies that produce the most complex programming developments.

3.2. History of the Development of Quality Systems

In the history of the development of quality systems, the following five stages can be distinguished:

- **1995, the Taylor system** is a system of requirements for the quality of products in the form of tolerances (maximum and minimum limit value). Contains templates configured

for the upper and low tolerance limits. This is a quality management system for each individual product.

- In **1924,** the *Bell Telephone Laboratories* company laid the foundations of ***statistical quality control.*** Later, statistical methods of quality control became widespread. For Japan, statistical methods of control became the basis of the economic revolution. With statistical methods of quality control, instead of checking and detecting flaws, the main focus is on the prevention of flaws by eliminating their causes.
- **In 1950s**, the idea of **total (universal) quality control — TQC** *(Total Quality Control)* appeared. The author is the American scientist A. Feigenbaum (1957, article *Total Quality Control*).

The main tasks of TQC are:

1. Predictable correction of possible product inconsistencies even at the design stage.
2. Quality control of the supplied products, components and materials.
3. Production management.
4. Applying service.
5. Monitoring compliance with the requirements.

At this stage, documented quality systems emerged that established responsibility, authority, and interaction in the field of quality of enterprise management and quality service specialists.

In Europe, there were auditors (an independent third party performing system registration and certification), and

much attention was paid to documenting the quality assurance systems. Japan met with enthusiasm the TQC ideas and took steps to further develop them.

- **1980s.** There is a transition from the total quality control (TQC) to total quality management (TQM). A lot of new international standards for quality management systems appeared, for example, ISO 9000 standards (1987). TQM does not just manage quality, it also provides management of goals and requirements. In TQM, quality assurance is a system of measures designed to cause the customer of a software product to have confidence in its quality. The basic principle of TQM is that there is no limit to improvement. The quality management system does not solve all the challenges necessary to ensure competitiveness, but it becomes more popular, and today it occupies a firm place in the market mechanism.
- **1990s.** The influence of society on enterprises grows, and enterprises take into account the interests of society more and more.

This leads to the appearance of standards of the ISO 14000 series, which establish requirements for management systems in terms of environmental protection and product safety.

The appearing standards complement each other, expand, and lead to a multifaceted consideration of the issue of product quality management. Now, quality management is not just a control of quality parameters and reasons for their deviations, it is a management activity that covers the life cycle of products, systematically provides strategic and operational processes of improving quality of products and functioning of the management quality system itself.

3.3. What is Quality? Quality Factors

Quality is a measure of utility, expediency, and efficiency of work, felt by every person. Improving quality, in turn, leads to reduction of losses at all stages of the product life cycle (marketing -> development -> production -> consumption -> utilization). As a result, the prime cost and price of the product are reduced, and this leads to the increase in the people's standard of living.

Now, there are several definitions of quality that are generally compatible with each other. The definition of "software quality" in international standards is: *[ISO 8402:1994 Quality management and quality assurance]*.

Quality is the totality of characteristics of an entity that bear on its ability to satisfy stated and implied needs. *[1061-1998 IEEE Standard for Software Quality Metrics Methodology]*

Software quality is the degree to which software possesses a desired combination of attributes.

"A **software quality factor** is a non-functional requirement for a software program which is not called up by the customer's contract, but nevertheless is a desirable requirement which enhances the quality of the software program." (wikipedia.org)

The table below shows various quality factors:

Understandability	The purpose of the software should be understandable, that is, the program interface and documentation.
Completeness	The program should be fully implemented.
Conciseness	There is no duplicate information. There is no duplicate code (replacement with procedure calls). There is no duplication of documentation.
Portability	Easy transfer of software into a new environment.

Consistency	The same formats and symbols should be used throughout the program and documentation.
Maintainability	The requirement for documentation and spare capacity when resources change (memory, processor). How easy it is to change the program with the changed customer requirements.
Testability	The program should run test cases and tools for measuring performance.
Usability	Simplicity and convenience of the user interface.
Reliability	There is no failures or malfunctions in programs, there is an ease of restoring the operating state
Structure	Parts of the program and links between them are clearly identified
Efficiency	Rational use of resources
Security	Storage and restriction of access to personal information, transaction fixing

Management and quality assurance is a concept that covers all stages of the development, release, and operation of software.

3.4. Documentation

Software documentation is documents that describe the rules for installing and operating the software.

Documenting software is such an important stage of its development that often the success of the distribution and operation of a software product depends largely on the quality of its documentation. The more complex and confusing the project, the higher the role of accompanying documents. It should be noted that the types of documents vary depending on the nature of the information they contain.

There are five objectives for the development of documents:

- Documents establish correlation between all those involved in the development process.
- Documents describe the responsibilities of the development team.
- Documents allow managers to evaluate the development process.
- Documents form the basis of the documentation for software maintenance.
- Documents describe the history of software development.

The software quality, along with other factors, is determined by the completeness and quality of a set of documents accompanying software. The set of documents accompanying software includes documents containing information necessary for the development, production, maintenance, and operation of the program.

There are four main types of documentation on the software:

- **Architecture/Design.** General description of software, working environment, and basic ideas for software development.
- **Technical documentation.** Detailed documentation for the program code, algorithms used, inter-module interfaces.
- **End user.** Manuals for end users, system administrators, installers, etc.
- **Marketing.** Principles of implementation, bonus and partnership additions, etc.

Any company imposes unique requirements on the composition and content of documents for its products. Most of

these requirements are based on the provisions and recommendations of national standards and/or standards common in world practice.

The principles of managing software documentation are the same for any project size.

Although for small projects, a significant part of provisions can be not applied, but the principles remain the same.

Most projects are developed on the basis of 2-3 documents:

- requirements document;
- user guide;
- administrator's guide.

Upon request of the customer, the developer should provide a complete list of documents that comply with national or international standards or contract requirements.

The documentation can be developed both for the entire software package and for its individual components.

Let's list the required and sufficient documents to provide the software system with the necessary level of quality.

Herewith we should note:

- **The need of the document**, that is, a document without which it is impossible to create a quality product.
- **Sufficiency of the document.** If the work is carried out in accordance with the document, the product will be guaranteed to be produced.

The process of developing a software product is similar to many other engineering tasks.

The Institute of Electrical and Electronics Engineers (IEEE, www.ieee.org) has developed standards for documentation of the software development process.

Below, there is a composition of the project documentation according to these standards by groups of documents in the following order:

1. **Software Verification and Validation Plan.** This document describes how, and in what order, the stages of the product and the product itself should be checked for compliance with requirements.
   Simply put, this document specifies how the customer and the developer will be able to make sure that they did what they wanted to do and that they really did it. It does not describe what and how they will do.
2. **Software Quality Assurance Plan.** This document describes how the project should achieve the compliance with the established level of quality.
   That is, in a separate document, the developer describes that he/she will try to ensure that there are no errors in the software product. And if, in spite of all the "organizational measures", they arise all the same, the customer will know that the developer tried very hard to avoid them.
3. **Software Configuration Management Plan.** This document describes where and how the developer will store versions of documentation, program code, and how he/she will determine how the code is associated with the documentation.
   For example, if a programmer should make changes to a program module, it can be difficult to find a description what exactly this module should do, what it did before, and so on, among a huge heap of different versions of documents. To help in the search, you need another document

that describes what it should do to find the information you need.

4. **Software Project Management Plan.** This document describes how the developer will manage the project of creating software to bring the development to the quality result. It includes risk management, time schedule, staffing issues, and so on.
5. **Software Requirements Specification.** It is the most important document that includes two parts: customer requirements and implementation sketches, that is, what can be implemented.
   Therefore, the customer is provided with two parts for verification: whether all his/her requirements are described correctly in both parts of the document. After the customer signs this document (verdict), it will be his/her fault if he/she missed something or changed his/her mind, if the developer made a mistake when writing requirements or the product doesn't meet the customer requirements.
6. **Software Design Document.** This document is the software architecture and application design details. It is not provided to the customer. It is often a secret of the enterprise.
7. **Software Test Documentation.** This document describes how, by whom, and when testing should be conducted.
   This approach to the development of documentation was considered to be the only correct one in the 80's and 90's of the XX century. In the late 90's, developers came to the understanding that the software is constantly being changed because of changing requirements, and there is no need to try to create the most durable program.

As a result, an approach called extreme programming emerged.

Extreme Programming allows the Customer:

1. Not to determine absolutely all requirements in advance;
2. Make changes in the development process;
3. Manage the need for further software development;
4. Not to pay for unnecessary documentation.

Extreme programming makes quite hard demands on programmers and the program code created by them because this code is the main documentation of the project, and it is enough to successfully maintain the product. Extreme programming requires several components to document the code:

- Selection of understandable names of functions, variables, and classes;
- Modular and intuitively understandable test cases;
- Expanded comments when describing classes and their components.

3.5. Documentation Standardization

The main function of technical documentation is a storage and receipt of all kinds of technical information.

Therefore, technical documentation should be as understandable as possible, informative, convenient, etc.

Everyone reads this kind of documentation, we suppose? It is very difficult to write clear and precise documentation, but there are standards and methods that simplify the process.

The technical documentation has one more important function — normative — it documents mutual obligations of the development participants.

After the technical documentation is approved, the unilateral modification of the content is legally impossible neither by the customer nor by the developer.

Example 1

If the approved requirements document for an automated system says that the site visitor should see the slogan: "Glory to work", the customer does not have the right to suddenly demand a demonstration of a commercial on this topic. It is possible that showing a video is a good idea, but the customer should have said it earlier when the developer agreed with him/her on the requirements document. On the other hand, if instead of the required text the system displays a night club advertising, the customer will have every reason to require the developer to make necessary changes to the system, and not to pay him/her money until the system is brought into compliance with the requirements document.

Example 2

If in the approved technical project it is documented that the PHP script is used to implement the automated system, the developer has no right to baffle the customer with the offer to purchase Lotos Domino instead of the free interpreter of this language. But, on the other hand, the customer too can't require the developer to use the computer BK 0010, lying around in the pantry instead of a normal server.

The operating documentation is a part of the product since it is impossible or very difficult to use the software without it. The operating documentation Standardization allows you to describe in the contract or in the requirements document

detailed requirements for it. Instead, references to relevant standards or parts thereof are used.

Therefore, the development of a set of documentation must be carried out by a highly qualified specialist who understands both the content of the project and the psychology of the future user, it is desirable to know a literary style, etc.

Lesson 1

INTRODUCTION TO SOFTWARE PROJECT MANAGEMENT