Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Уфимский государственный авиационный технический университет»

И. В. КИРСАНОВА

# PROFESSIONAL ENGLISH FOR SOFTWARE DEVELOPERS

Уфа 2022

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Уфимский государственный авиационный технический университет»

И. В. КИРСАНОВА

# PROFESSIONAL ENGLISH FOR SOFTWARE DEVELOPERS

*Допущено Редакционно-издательским советом УГАТУ
в качестве учебного пособия для студентов очной и заочной форм обучения,
обучающихся по направлению подготовки бакалавров и магистрантов
09.03.04, 09.04.04 Программная инженерия*

Учебное электронное издание сетевого доступа


**ISBN 978-5-4221-1597-6**

Уфа 2022

*Рецензенты:*
*декан факультета математики и информационных технологий БашГУ*
*д-р физ.-мат. наук, профессор З. Ю. Фазуллин;*
*доцент кафедры русского языка и иностранных языков*
*ГБОУ ВО «БАГСУ» при Главе Республики Башкортостан*
*канд. филол. наук В. Р. Габдуллина*

**Кирсанова И. В.**
Professional english for software developers : учебное пособие [Электронный ресурс] / Уфимск. гос. авиац. техн. ун-т. – Уфа : УГАТУ, 2022. – URL: https://www.ugatu.su/media/uploads/MainSite/Ob%20universitete/Izdateli/ El_izd /2022-117.pdf

Предназначено для совершенствования речевых навыков и развития умений профессионально-ориентированного иноязычного общения в устной и письменной формах, чтения и перевода оригинальных английских текстов, соответствующих направлению подготовки IT-специалистов. Тексты снабжены упражнениями с использованием элементов функционально-коммуникативного обучения английскому языку, а также ссылками на видеоматериалы и заданиями для самостоятельной работы студентов.

Предназначено для студентов и магистрантов, изучающих дисциплину «Иностранный язык в профессиональной деятельности», а также для студентов, связанных с информационными технологиями.

При подготовке электронного издания использовались следующие программные средства:

- Adobe Acrobat – текстовый редактор;
- Microsoft Word – текстовый редактор.

Автор: *Кирсанова Инна Вячеславовна*

Редактирование и верстка: *О. А. Соколова*
Программирование и компьютерный дизайн: *О.М. Толкачёва*



Подписано к использованию: 30.06.2022

Объем: 2,7 Мб.

ФГБОУ ВО «Уфимский государственный авиационный технический университет»
450008, Уфа, ул. К. Маркса, 12.
Тел.: +7-908-35-05-007
e-mail: rik@ugatu.su

# PREFACE

The textbook ‘Professional English for Software Developers’ is intended as a manual for the students who have chosen software engineering as the sphere of their specialization. They will be able to acquire and master communication skills in English and use them effectively in their professional field. Thus, the main objective of the textbook is to develop learners’ ability to use the English language for a variety of communicative purposes.

The manual also allows you to organize students’ independent work to master the English language and form intercultural communication of future specialists. From the very beginning of the unit the students are given the opportunity to work independently with the topical vocabulary.

The book contains 12 units and texts for additional reading. These texts can be recommended for testing and controlling text comprehension and translation skills. Each unit contains a sufficient number of lexical exercises, different tasks for discussions, suggested topics for writing reports and making presentations. Lexical and grammar tasks ‘English for Software Development’ have been developed according to modern principles of learning and teaching foreign languages. Grammar Revision tasks are presented in tables and a wide range of grammar tasks on transformation, translation and sentence completion tasks by using grammar patterns is also very useful for learning and using the English language in practice.

We hope that these tasks can be a good way to let students and Master students practice a wide variety of language skills. In turn teachers may engage students in authentic language practice experiences, supporting their learning strategies and critical thinking development.

In conclusion, we wish you success in your learning English for professional purposes!

# UNIT 1. Introduction to Software Engineering

---

*Learning objectives*

- to acquire basic knowledge about software engineering and its types
- to understand if all software requires software engineering

*Key words and phrases. Give Russian equivalents and remember the meanings of the key words and phrases used in the text*

Concept; to be complicated; application; definition; tools; security risks; vulnerability; operational software engineering; transitional software engineering; recurrent; lifecycle; implementation; maintenance; retirement; to be congruent; stand-alone applications; interactive transaction-based applications; embedded control systems; batch processing systems

*Read the following text and do the exercises given after it*

Software engineering is a concept in and of itself, but to better understand it, you need to know what each part of the term means before you can fully understand how they operate together. It can be difficult to understand, even though it does seem straightforward. That is because the pieces are more complicated than many believe - and working with software engineering for an application is difficult and time-consuming. Software engineering has two parts: software and engineering.

**Software** is a collection of codes, documents, and triggers that does a specific job and fills a specific requirement.

**Engineering** is the development of products using best practices, principles, and methods.

## What is Software Engineering?

- It is a branch of engineering that deals with the development of software products. It operates within a set of principles, best practices, and methods that have been carefully honed throughout the years, changing as software and technology change.

Software engineering leads to a product that is reliable, efficient, and effective at what it does. While software engineering can lead to products

that do not do this, the product will almost always go back into the production stage. So, what is the complete definition of software engineering?

The IEEE fully defines software engineering as:
1. The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.

What the software engineering meaning doesn't explain is that everything that has been software engineered needs to work on real machines in real situations, not within.

Software engineering starts when there is a demand for a specific result or output for a company, from an application. From somewhere on the IT team, typically the CIO, there is a request put into the developer to create some sort of software. The software development team breaks down the project into the requirements and steps. Sometimes, this work will be farmed out to independent contractors, vendors, and freelancers. When this is the case, software engineering tools help to ensure that all of the work done is congruent and follows best practices.

How do developers know what to put into their software? They break it down into specific needs after conducting interviews, collecting information, looking into the existing application portfolio, and talking to IT leaders. Then, they will build a roadmap of how to build the software. This is one of the most important parts because much of the "work" is completed during this stage - which also means that any problems typically occur here as well.

The true starting point is when developers begin to write code for the software. This is the longest part of the process in many cases as the code needs to be congruent with current systems and the language used in them. Unfortunately, these problems often aren't noticed until much later on in the project and then rework needs to be completed.

The code should be tested as it is written and once it has been completed – at all parts of the life cycle. With software engineering tools, you will be able to continuously test and monitor.

## Software Engineering Basics

The true work of software engineering begins before the product has even been designed – and the software engineering basics dictate that it continues long after the "work" has been completed. It all begins with a

thorough and complete understanding of what your software needs to have – this includes what the software needs to do, the system in which it needs to operate, and all of the security that it entails. Security is one of the software engineering basics because it is so essential to all aspects of development. Without tools to help you better understand how your code is being built and where any security problems may fall, your team can easily become lost in the development stage.

Software engineering design basics require creating the instructions for the computer and the systems. Much of this will take place at the coding level by professionals who have comprehensive training. Still, it is important to understanding that software engineering isn't always a linear process, which means that it requires thorough vetting once it has been completed.

Not all software requires software engineering. Simplistic games or programs that are used by consumers may not need engineering, depending on the risks associated with them. Almost all companies do require software engineering because of the high-risk information that they store and security risks that they pose.

Software engineering helps to create customized, personalized software that should look into vulnerabilities and risks before they even emerge. Even when the software engineering principles of safety aren't required, it can also help to reduce costs and improve customer experience.

## Types of Software Engineering

Software engineering studies the design, development, and maintenance of software as an umbrella definition. Still, there are different types of software engineering that a company or product may need. Problems tend to emerge when software is low-quality or isn't properly vetted before deployment.

There has been a lot of demand for software engineers because of the rate of change in user requirements, statutes, and the platforms we use. Software engineering works on a few different levels: *Operational Software Engineering:* Software engineering on the operational level focuses on how the software interacts with the system, whether or not it is on a budget, the usability, the functionality, the dependability, and the security.

*Transitional Software Engineering:* This type focuses on how software will react when it is changed from one environment to another. It typically

takes some scalability or flexibility in the development. *Software Engineering Maintenance:* Recurrent software engineering focuses on how the software functions within the existing system, as all parts of it change.

Software engineering functions at all parts of the software development lifecycle, including analysis, design, development, testing, integration, implementation, maintenance, and even retirement.

It is important to understand that software engineering isn't a new practice, but it is constantly changing and can feel new on a regular basis. Software is used in everything around us, so it is important to ensure that all software is working properly. If it does not, it can result in loss of money, loss of reputation, and even in some cases, loss of life.

[https://www.castsoftware.com/glossary/what-is-software-engineering-definition-types-of-basics-introduction]

## 1. Text-based Assignments

*1.1. Make nouns from the following verbs according to the model and translate them*

*Verb+-tion (-ation)*

Inform, create, connect, integrate, explore, prepare, destine, realize, associate, implement, operate.

*1.2. Give English equivalents of the following words and word combinations:*

Казаться простым; кропотливый, занимающий много времени; конкретное требование; оттачиваться на протяжении многих лет; полное определение; проблемы безопасности; тщательная проверка; ассоциироваться с чем-л.; сократить расходы; появляться; всеохватывающее определение; потребность; взаимодействовать с системой; операционное ПО; переходная *(переход с одной платформы на другую)* разработка ПО; техническое обслуживание; количественный подход; отправная точка.

*1.3. Match the following words with their definitions:*

| | |
|---|---|
| 1. High technology e | a) the range of operations that can be run on a computer or other electronic system |
| 2. ngineering f | b) come into existence or greater prominence |
| 3. functionality a | c) take or use another instead of |
| 4. change c | d) instructions for a computer in some programming language, often machine language |
| 5. emerge b | e) advanced technological development, especially in electronics |
| 6. code d | f) a field of study or activity concerned with modification or development in a particular area |
| 7. application developer h | g) a position or stage on a scale of quantity, extent, rank, or quality |
| 8. level g | h) a person who writes computer programs to meet specific requirements |

*1.4. Answer the following questions on the text*

1) What does software engineering deal with?
2) What is the starting point for developers?
3) Does all software require software engineering?
4) How can you explain the importance of software engineering?
5) What are the types of software engineering?
6) Software engineering functions at all parts of the software development, doesn't it?

*1.5. Read the text again and decide if the following statements are true or false.*

1) Software engineering is a branch of engineering that deals with the development of software products and security problems are the most important for developers.
2) Software engineering leads to a product that is reliable, efficient, and effective at what it does.
3) All software requires software engineering.
4) Problems tend to emerge even if software is high-quality and is properly vetted before deployment.

5) Software engineering functions at four parts of the software development lifecycle, including analysis, design, development, testing.
6) Software engineering isn't a new practice, but it is constantly changing and can feel new on a regular basis.
7) Software engineering isn't always a linear process, which means that it requires thorough vetting once it has been completed.

## 2. Focus on Grammar

*2.1. Choose one of the verbs in brackets. Put them into the necessary form to complete the following sentences*

1) A lot of humans (be, have, do) dependent on technology today, which will (be, have, do) bad to them.
2) I (be, have, do) had my iPad for years now and I (be, have, do) very happy with it.
3) Because I (be, have, do) not have the chance to speak to my boss yesterday I (be, have, do) to text her in Viber.
4) The 21st century (be, have, do) the age of cutting-edge technologies.
5) They (do, be, have) doing research work on the latest applications for mobile devices.
6) All books can (be, do, have) read online.
7) We (be, have, do) not see any downsides in using personal computers at all.
8) They (be, have, do) surfing the Internet all day yesterday.
9) Every day Linda (be, do, have) a lot of exercise to keep fit.
10) Safari browser online tutorial (be, do, have) provided the user with help and support in using it.

*2.2. Identify passive structures and translate the sentences*

Все времена в страдательном залоге образуются из вспомогательного глагола to be в соответствующем лице числе и времени и смыслового глагола в форме причастия прошедшего времени /Participle II/.

1) This method has been referred to in an earlier paper.
2) I do not think this instrument can be relied upon.

3) In operational categories, the factors that decide the software performance in operations. It can be measured on: budget, usability, efficiency and correctness.
4) If the job is entered without errors it will be chosen for execution.
5) This electronic equipment has been designed for speeding up production.
6) Business variables have been and are being expressed as mathematical functions and are being statistically analyzed.
7) Even when the software engineering principles of safety aren't required, it can also help to reduce costs and improve customer experience.
8) This interview was also recorded as a video podcast. Check out the video on the Software Daily YouTube channel.
9) The code should be tested as it is written and once it has been completed – at all parts of the life cycle.

## 3. Discussion

*3.1. Possible topics for discussion*

1) What are the attributes of good software? What is your idea of it?
2) What is the difference between software engineering and computer science?
3) Do you agree? Why/why not?
4) What is interesting for you in this branch of engineering?

*3.2. Write a short summary of the text about software engineering*

*3.3. Make up and dramatize a dialogue using the top interview questions for software engineers*

Hiring a software engineer is a process that should be approached carefully and with deliberation. A good software engineer will help your company grow, but one that does not have the right skills or a good work ethic can slow down and hinder your growth.

Therefore, you should know the best questions to ask during the hiring process to successfully recruit software engineers.

1) Why did you decide to become a software engineer?
2) What programming languages do you prefer?
3) What's important when checking a team member's code?
4) What project management tools have you used?
5) Talk about a project you completed successfully.
6) What are you looking for in this job?
7) Why should we hire you?
8) How did you solve a problem you faced?
9) What are you working on right now?
10) How do you assure software quality?
11) Do you enjoy working with a team or alone?
12) What are your career goals?
13) How do you keep your skills sharp and up to date?
14) What questions do you have for us?

*3.4. Write your comments on the following topic* How to Become an Expert Software Engineer (and Get Any Job You Want)

Future Selection Ideas > How to Become an Expert Software Engineer (and Get Any Job You Want)

 Hi all! A few years ago, working for Canonical Ltd. on the world's most popular Linux distribution: Ubuntu, seemed like just a dream to me… For over 2 years now, I've been living that dream!

Achieving my dream job inspired me to write a book to help others make their dreams a reality too. The book is called: "How to Become an Expert Software Engineer (and Get Any Job You Want)". It helps readers build impressive résumés by introducing them to the world of free and open source software development as a means of acquiring new skills and gaining focused, real world work experience.

I'd really love to hear what you guys thought of it! [11].

## 4. Additional reading

*4.1. Read and translate the following text and suggest the title*

Software engineering is a systematic approach to the production of software that takes into account practical cost, schedule, and dependability issues, as well as the needs of software customers and producers. How this systematic approach is actually implemented varies dramatically depending on the organization developing the software, the type of software, and the people involved in the development process. There are no universal software engineering methods and techniques that are suitable for all systems and all companies. Rather, a diverse set of software engineering methods and tools has evolved over the past 50 years. Perhaps the most significant factor in determining which software engineering methods and techniques are most important is the type of application that is being developed.

There are many different types of application including:

1. *Stand-alone applications* are application systems that run on a local computer, such as a PC. They include all necessary functionality and do not need to be connected to a network. Examples of such applications are office applications on a PC, CAD programs, photo manipulation software, etc.

2. *Interactive transaction-based applications* are applications that execute on a remote computer and that are accessed by users from their own PCs or terminals. Obviously, these include web applications such as e-commerce applications where you can interact with a remote system to buy goods and services. This class of application also includes business systems, where a business provides access to its systems through a web browser or special-purpose client program and cloud-based services, such as mail and photo sharing. Interactive applications often incorporate a large data store that is accessed and updated in each transaction.

3. *Embedded control systems* are software control systems that control and manage hardware devices. Numerically, there are probably more embedded systems than any other type of system. Examples of embedded systems include the software in a mobile (cell) phone, software that controls anti-lock braking in a car, and software in a microwave oven to control the cooking process.

4. *Batch processing systems* are business systems that are designed to process data in large batches. They process large numbers of individual inputs to create corresponding outputs. Examples of batch systems include periodic billing systems, such as phone billing systems, and salary payment systems.

5. *Entertainment systems* are systems that are primarily for personal use and which are intended to entertain the user. Most of these systems are games of one kind or another. The quality of the user interaction offered is the most important distinguishing characteristic of entertainment systems.

6. *Systems for modeling and simulation* are systems that are developed by scientists and engineers to model physical processes or situations, which include many, separate, interacting objects. These are often computationally intensive and require high-performance parallel systems for execution.

7. *Data collection systems* are systems that collect data from their environment using a set of sensors and send that data to other systems for processing. The software has to interact with sensors and often is installed in a hostile environment such as inside an engine or in a remote location. 8. *Systems of systems* are systems that are composed of a number of other software systems. Some of these may be generic software products, such as a spreadsheet program. Other systems in the assembly may be specially written for that environment.

Of course, the boundaries between these system types are blurred. If you develop a game for a mobile (cell) phone, you have to take into account the same constraints (power, hardware interaction) as the developers of the phone software. Batch processing systems are often used in conjunction with web-based systems. For example, in a company, travel expense claims may be submitted through a web application but processed in a batch application for monthly payment. You use different software engineering techniques for each type of system because the software has quite different characteristics. For example, an embedded control system in an automobile is safety-critical and is burned into ROM when installed in the vehicle. It is therefore very expensive to change. Such a system needs very extensive verification and validation so that the chances of having to recall cars after sale to fix software problems are minimized.

User interaction is minimal (or perhaps nonexistent) so there is no need to use a development process that relies on user interface prototyping .[10]

*4.2. Explain each type of applications in your own words*

→ boot: (bilgisayar) sistemi yeniden başlatma
→ generic: (program kodu) tüm veri tiplerinde çalışacak şekilde yazılmış

# UNIT 2. What is Software? Types of Software

---

*Learning objectives*

- to understand what software is and how it works
- to acquire basic knowledge about different types of software
- to understand the difference between system software and application software

*Key words and phrases. Give Russian equivalents and remember the meanings of the key words and phrases.*

Application software; system software; to execute; scripts; language processor; middleware; assemblers; compilers; debuggers; interpreters; source code; computer's hard drive; high-level application software; to boot up; to run on a device; image editors; object code; to launch

*Before reading the text B watch the video from https://www. https://searchapparchitecture.techtarget.com/definition/software to get some information on the topic*

*Read the following text and do the exercises given after it*

Software is a set of instructions, data or programs used to operate computers and execute specific tasks. It is the opposite of hardware, which describes the physical aspects of a computer. Software is a generic term used to refer to applications, scripts and programs that run on a device. It can be thought of as the variable part of a computer, while hardware is the invariable part.

The two main categories of software are application software and system software. **Application software** is a computer software package that performs a specific function for a user, or in some cases, for another application. An application can be self-contained, or it can be a group of programs that run the application for the user. Examples of modern applications include office suites, graphics software, databases and database management programs, web browsers, word processors, software development tools, image editors and communication platforms.

et al. → gruplarda isimleri söylerken üçten sonra yazılır
diğerleri gibi bir anlam

spreadsheet → excel genel ad

beneath → altında yatan

**System software** is designed to run a computer's hardware and provides a platform for applications to run on top of. System software coordinates the activities and functions of the hardware and software. In addition, it controls the operations of the computer hardware and provides an environment or platform for all the other types of software to work in. The OS is the best example of system software; it manages all the other computer programs. Other examples of system software include the firmware, computer language translators and system utilities.

Other types of software include programming software, which provides the programming tools software developers need; middleware, which sits between system software and applications; and driver software, which operates computer devices and peripherals.

Language Processor: As we know that system software converts the human-readable language into a machine language and vice versa. So, the conversion is done by the language processor. It converts programs written in high-level programming languages like Java, C, C++, Python, etc.(known as source code), into sets of instructions that are easily readable by machines (known as object code or machine code).

**Driver software.** Also known as device drivers, this software is often considered a type of system software. Device drivers control the devices and peripherals connected to a computer, enabling them to perform their specific tasks. Every device that is connected to a computer needs at least one device driver to function. Examples include software that comes with any nonstandard hardware, including special game controllers, as well as the software that enables standard hardware, such as USB storage devices, keyboards, headphones and printers.

**Middleware.** The term *middleware* describes software that mediates between application and system software or between two different kinds of application software. For example, middleware enables Microsoft Windows to talk to Excel and Word. It is also used to send a remote work request from an application in a computer that has one kind of OS, to an application in a computer with a different OS. It also enables newer applications to work with legacy ones.

**Programming software.** Computer programmers use programming software to write code. Programming software and programming tools enable developers to develop, write, test and debug other software programs. Examples of programming software include assemblers, compilers, debuggers and interpreters.

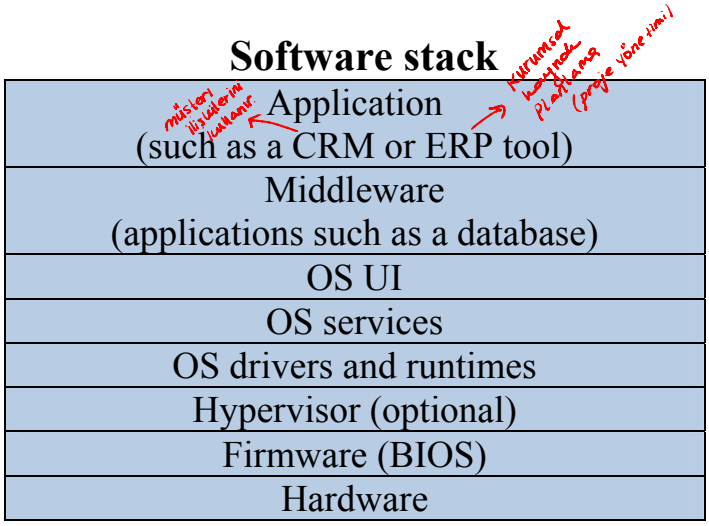



Fig.1. Here is a complete picture of the full software stack

**How does software work?**

Application software consists of many programs that perform specific functions for end users, such as writing reports and navigating websites. Applications can also perform tasks for other applications. Applications on a computer cannot run on their own; they require a computer's OS, along with other supporting system software programs, to work.

These desktop applications are installed on a user's computer and use the computer memory to carry out tasks. They take up space on the computer's hard drive and do not need an internet connection to work. However, desktop applications must adhere to the requirements of the hardware devices they run on.

Web applications, on the other hand, only require internet access to work; they do not rely on the hardware and system software to run. Consequently, users can launch web applications from devices that have a web browser. Since the components responsible for the application functionality are on the server, users can launch the app from Windows, Mac, Linux or any other OS.

System software sits between the computer hardware and the application software. Users do not interact directly with system software as it runs in the background, handling the basic functions of the computer. This software coordinates a system's hardware and software so users can run high-level application software to perform specific actions. System software executes when a computer system boots up and continues running as long as the system is on.

**System software vs. application software**

| System software | Application software |
|---|---|
| General-purpose software that manages basic system resources and processes | System software that performs specific tasks to meet user needs |
| Written in low-level assembly language or machine code | Written in high-level languages, such as Python or JavaScript |
| Must meet specific hardware needs; interacts closely with hardware | Does not take hardware into account and doesn't interact directly with hardware |
| Installed at the same time as the OS, usually by the manufacturer | User or admin installs software when needed |
| Runs any time the computer is on | User triggers and stops the program |
| Works in the background and users do not usually access it | Runs in the foreground and users work directly with the software to perform specific tasks |
| Runs independently | Needs system software to run |
| Is necessary for the system to function | Isn't needed for the system to function |

Fig. 2. The key differences between system and application software

Early software was written for specific computers and sold with the hardware it ran on. In the 1980s, software began to be sold on floppy disks, and later on CDs and DVDs. Today, most software is purchased and directly downloaded over the internet. Software can be found on vendor websites or application service provider websites.

[https://searchapparchitecture.techtarget.com/definition/software]

## 1. Text-based Assignments

*1.1. Give English equivalents of the following words and word combinations:*

Управление базами данных; неизменяемая часть; самостоятельный (автономный); текстовый редактор с расширенными возможностями форматирования; предоставлять платформу; микропроцессорное программное обеспечение; периферийные устройства; набор команд; межплатформное ПО, обеспечивающее прозрачную работу

приложений в неоднородной сетевой среде; наушники; посылать запрос; выполнять задачи; работать непосредственно с системным ПО

*1.2. Write a sentence with **each** word to **illustrate** its meaning*

a) to provide – provider
b) to vary – invariable – variation
c) specific – specification – to specify
d) to require – requirement
e) to execute – execution – executable

*1.3. Match the following words from the text with their meanings*

***Self-contained, vendor, machine language, functionality, application, platform, to perform***

a) the range of operations that can be run on a computer or other electronic system;
b) not depending on or influenced by others;
c) to work, to function, or do something to a specified standard;
d) machine code;
e) a standard for the hardware of a computer system, which determines the kinds of software it can run;
f) a seller, particularly of real property;
g) a program or piece of software designed to fulfill a particular purpose.

*1.4. Answer the following questions on the text*

1) What is system software designed for?
2) What types or categories does software include?
3) What is the other name of driver software?
4) Are assemblers and compilers the examples of programming software or middleware?
5) What can you say about desktop applications?
6) What more would you like your software to do?

*1.5. Read the text again and decide if the following statements are true or false.*

1) Software is the opposite of hardware, which describes the physical aspects of a computer and it can be thought of as the variable part of a computer, while hardware is the invariable part.
2) An application can never be a group of programs that run the application for the user.
3) Communication platform is one of the examples of modern applications.
4) System software coordinates the activities and functions of software.
5) C++ is one of the low level programming languages.
6) Every device that is connected to a computer doesn't need any device driver to function.
7) Since the components responsible for the application functionality are on the server, users can launch the app from Windows, Mac, Linux or any other OS.
8) In the 1990s, software began to be sold on floppy disks, and later on CDs and DVDs.

## 2. Focus on Grammar

*2.1. Complete the following sentences and translate them (mind the modal verb)*

1) Our manager had to solve many complicated practical problems last month *(had to, is to, can, must)*.
2) Now he may study this phenomenon *(may, is to, could)*.
3) He might have seen a jet engine in action many years ago *(can't see, couldn't have seen, might have seen)*.
4) Users can launch the app from Windows, Mac, Linux or any other OS *(must have, can, couldn't)*.
5) Desktop applications must adhere to the requirements of the hardware devices they run on *(are allowed to, must, mustn't)*.
6) Software can be found on vendor websites or application service provider websites *(can't, have to, can, must)*.

7) For example, without your Internet browser software, you could not surf the Internet and read this article *(mustn't, could not, may not)*.
8) The business growth comes together with a rise in the amount of data that should be administered, which results in the need to manage all that information that is continuously growing successfully *(can, can't be, should be)*.

*2.2. Study the table and make up your own sentences to demonstrate the following modal meanings*

| | | |
|---|---|---|
| a) *must, may u might + Perfect Infinitive* | Выражают возможность или вероятность действия, относящегося к прошлому и обычно переводятся словами «должно быть, возможно». | He **must have lost** his book somewhere. Он, должно быть, потерял свою книгу где-то. |
| b) *can / could +not + Perfect Infinitive* | Выражают сомнение в возможности совершения действия в прошлом и обычно переводятся при помощи слов «не может быть». | He **cannot have made** such a serious mistake. Не может быть, чтобы он допустил такую серьезную ошибку. |
| c) *ought (to), should, could, might + Perfect Infinitive* | Указывают на то, что действие, которое могло или должно было бы совершиться, не совершилось. | You **should have changed** the current strength at all points of the circuit. Вам следовало бы изменить силу тока во всех точках цепи. |

*2.3. Translate the following sentences with modal verbs in combination with Perfect Infinitives*

1) They must have lost their way, as they appeared in the village only at night.
2) He cannot have entrusted this scientific work to a man he has known for such a short period of time.
3) She rested her eyes on him thinking of all things he must have done since she saw him last.

4) You could not have seen him there because he left the place two months ago.
5) There are so many mistakes in your exercises. You should have been more attentive.
6) In the morning I did not find him in his room, he must have gone leaving no note for us.
7) She might have overlooked something that may turn out to be important in proving his innocence.
8) In the fewest words he told them that a fatal accident must have happened to her.

## 3. Discussion

*3.1. Answer the questions concerning software development*

1) How much do you know about computer software?
2) What is your favorite piece of software?
3) Do you ever have software problems?
4) Do you keep up to date with the latest software?
5) Are you surprised that a lot of software is free?
6) Have you ever bought pirated software? Would you?
7) What do you think of speech recognition software?
8) What more would you like your software to do?
9) What questions would you like to ask Bill Gates or Steve Jobs about software?

*3.2. Possible topics for discussion*

1) Bill Gates said: "A solid working knowledge of productivity software and other IT tools has become a basic foundation for success in virtually any career." Do you agree with him?
2) What is the greatest piece of software ever created?
3) Why do you think Microsoft has stuck to software and never went into producing computers, like Apple?
4) Do you think software is good value for money?
5) What do you think of people who design software?

## 4. Additional reading

Bak

*4.1. Read and translate the following text*

### Design and implementation

The software development lifecycle is a framework that project managers use to describe the stages and tasks associated with designing software. The first steps in the design lifecycle are planning the effort and then analyzing the needs of the individuals who will use the software and creating detailed requirements. After the initial requirements analysis, the design phase aims to specify how to fulfill those user requirements.

The next is step is implementation, where development work is completed, and then software testing happens. The maintenance phase involves any tasks required to keep the system running.

The software design includes a description of the structure of the software that will be implemented, data models, interfaces between system components and potentially the algorithms the software engineer will use.

The software design process transforms user requirements into a form that computer programmers can use to do the software coding and implementation. The software engineers develop the software design iteratively, adding detail and correcting the design as they develop it.

The different types of software design include the following:

*Architectural design.* This is the foundational design, which identifies the overall structure of the system, its main components and their relationships with one another using architectural design tools.

*High-level design.* This is the second layer of design that focuses on how the system, along with all its components, can be implemented in forms of modules supported by a software stack. A high-level design describes the relationships between data flow and the various modules and functions of the system.

*Detailed design.* This third layer of design focuses on all the implementation details necessary for the specified architecture.

Yazılım geliştirme yaşam döngüsü, proje yöneticilerinin yazılım tasarlamaya ilişkin aşamaları ve görevleri tanımlamak için kullandıkları bir çerçevedir. Tasarım yaşam döngüsünün ilk adımları, yazılımı kullanacak ve ayrıntılı gereksinimler oluşturacak kişilerin çabalarını planlayıp ihtiyaçlarını analiz etmektir. İlk gereksinimler analizinden sonra, tasarım aşaması bu kullanıcı gereksinimlerinin nasıl karşılanacağını belirlemeyi amaçlar. Bir sonraki adım, uygulama, yükleme işi tamamlanır ve yazılım testigerçekleşir. Bakım aşaması, sistemi çalışır durumda tutmak için gereken tüm görevleri içerir. Yazılım tasarımı uygulanacak yazılımın yapısının, veri modellerinin, sistem bileşenleri arasındaki arayüzlerin ve yazılım mühendisinin kullanacağı algoritmaların bir açıklamasını içerir. Yazılım tasarımı işlemcilerikullanıcı gereksinimlerini, bilgisayar programcılarının yazılım kodlama ve uygulama için kullanabileceği bir biçimde biçimler. Yazılım mühendisleri yazılım tasarımını yinelemeli olarak geliştirerek, ayrıntı ekler ve tasarım geliştirildikçe düzeltebilirler. Farklı yazılım tasarımı türleri şunlardır:

Mimari tasarım. Bu temel tasarım, mimari tasarım araçlarını kullanarak sistemin genel yapısını, ana bileşenlerini ve birbirleriyle olan ilişkilerini tanımlar.
Üst düzey tasarım. Bu, tüm bileşenleriyle birlikte sistemin bir yazılım yığını tarafından desteklenen modü l biçimlerinde nasıl uygulanacağına odaklanan ikinci tasarım katmanıdır. Üst düzey bir tasarım, veri akışı ile sistemin çeşitli modülleri ve işlevleri arasındaki ilişkileri açıklar.
Ayrıntılı tasarım. Bu üçüncü tasarım katmanı, belirtilen mimari için gerekli tüm uygulama ayrıntılarına odaklanır.

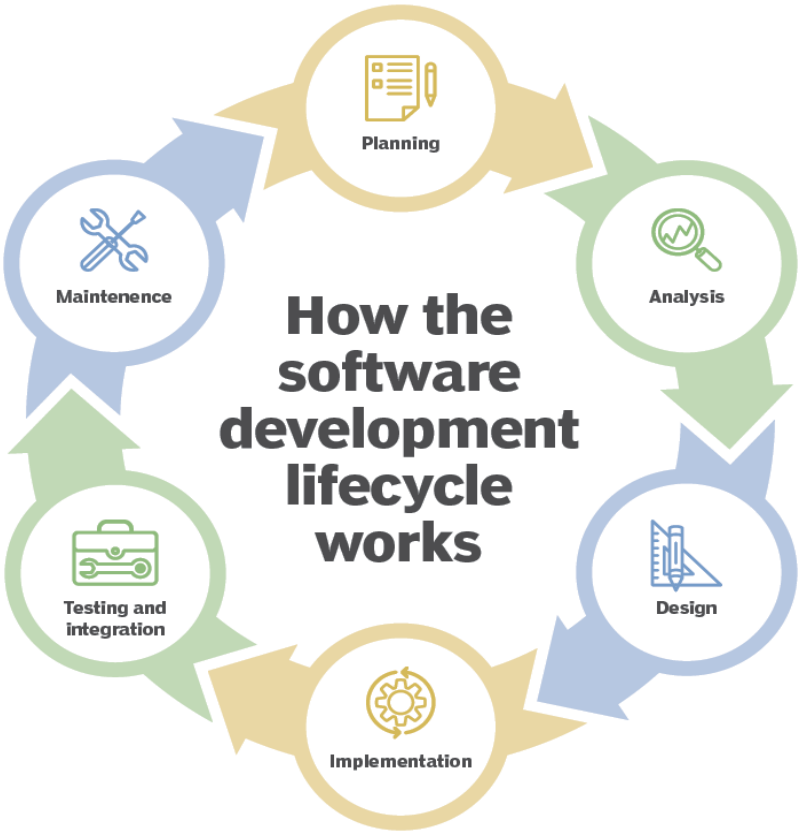



Fig. 3. The main steps involved in developing software

## How to maintain software quality

Software quality measures if the software meets both its functional and nonfunctional requirements.

Functional requirements identify what the software should do. They include technical details, data manipulation and processing, calculations or any other specific function that specifies what an application aims to accomplish.

Nonfunctional requirements -- also known as quality attributes -- determine how the system should work. Nonfunctional requirements include portability, disaster recovery, security, privacy and usability.

Software testing detects and solves technical issues in the software source code and assesses the overall usability, performance, security and compatibility of the product to ensure it meets its requirements.

The dimensions of software quality include the following characteristics:

Accessibility. The degree to which a diverse group of people, including individuals who require adaptive technologies such as voice recognition and screen magnifiers, can comfortably use the software.

Compatibility. The suitability of the software for use in a variety of environments, such as with different OSes, devices and browsers.

Efficiency. The ability of the software to perform well without wasting energy, resources, effort, time or money.

Functionality. Software's ability to carry out its specified functions.

Installability. The ability of the software to be installed in a specified environment.

Localization. The various languages, time zones and other such features a software can function in.

Maintainability. How easily the software can be modified to add and improve features, fix bugs, etc.

Performance. How fast the software performs under a specific load.

Portability. The ability of the software to be easily transferred from one location to another.

Reliability. The software's ability to perform a required function under specific conditions for a defined period of time without any errors.

Scalability. The measure of the software's ability to increase or decrease performance in response to changes in its processing demands.

Security. The software's ability to protect against unauthorized access, invasion of privacy, theft, data loss, malicious software, etc.

To maintain software quality once it is deployed, developers must constantly adapt it to meet new customer requirements and handle problems customers identify. This includes improving functionality, fixing bugs and adjusting software code to prevent issues. How long a product lasts on the market depends on developers' ability to keep up with these maintenance requirements.

When it comes to performing maintenance, there are four types of changes developers can make, including:

Corrective. Users often identify and report bugs that developers must fix, including coding errors and other problems that keep the software from meeting its requirements.

Adaptive. Developers must regularly make changes to their software to ensure it is compatible with changing hardware and software environments, such as when a new version of the OS comes out.

**Perfective**. These are changes that improve system functionality, such as improving the user interface or adjusting software code to enhance performance.

**Preventive**. These changes are done to keep software from failing and include tasks such as restructuring and optimizing code.

## Software licensing and patents

A software license is a legally binding document that restricts the use and distribution of software.

Typically, software licenses provide users with the right to one or more copies of the software without violating copyright. The license outlines the responsibilities of the parties that enter into the agreement and may place restrictions on how the software can be used.

Software licensing terms and conditions generally include fair use of the software, the limitations of liability, warranties, disclaimers and protections if the software or its use infringes on the intellectual property rights of others.

Licenses typically are for proprietary software, which remains the property of the organization, group or individual that created it; or for free software, where users can run, study, change and distribute the software. Open source is a type of software where the software is developed collaboratively, and the source code is freely available. With open source software licenses, users can run, copy, share and change the software similar to free software.

Over the last two decades, software vendors have moved away from selling software licenses on a one-time basis to a software-as-a-service subscription model. Software vendors host the software in the cloud and make it available to customers, who pay a subscription fee and access the software over the internet.

Although copyright can prevent others from copying a developer's code, a copyright cannot stop them from developing the same software independently without copying. A patent, on the other hand, enables a developer to prevent another person from using the functional aspects of the software a developer claims in a patent, even if that other person developed the software independently.

In general, the more technical software is, the more likely it can be patented. For example, a software product could be granted a patent if it

creates a new kind of database structure or enhances the overall performance and function of a computer.

[https://searchapparchitecture.techtarget.com/definition/software]

*4.2. Make a report on history of software. This brief timeline of the history of software will help you to prepare for your report*

The term *software* was not used until the late 1950s. During this time, although different types of programming software were being created, they were typically not commercially available. Consequently, users -- mostly scientists and large enterprises -- often had to write their own software.

June 21, 1948. Tom Kilburn, a computer scientist, writes the world's first piece of software for the Manchester Baby computer at the University of Manchester in England.

Early 1950s. General Motors creates the first OS, for the IBM 701 Electronic Data Processing Machine. It is called General Motors Operating System, or GM OS.

1958. Statistician John Tukey coins the word *software* in an article about computer programming.

Late 1960s. Floppy disks are introduced and are used in the 1980s and 1990s to distribute software.

Nov. 3, 1971. AT&T releases the first edition of the Unix OS.

1977. Apple releases the Apple II and consumer software takes off.

1979. VisiCorp releases VisiCalc for the Apple II, the first spreadsheet software for personal computers.

1981. Microsoft releases MS-DOS, the OS on which many of the early IBM computers ran. IBM begins selling software, and commercial software becomes available to the average consumer.

1980s. Hard drives become standard on PCs, and manufacturers start bundling software in computers.

1983. The free software movement is launched with Richard Stallman's GNU (GNU is not Unix) Linux project to create a Unix-like OS with source code that can be freely copied, modified and distributed.

1984. Mac OS is released to run Apple's Macintosh line.

Mid-1980s. Key software applications, including AutoDesk AutoCAD, Microsoft Word and Microsoft Excel, are released.

1985. Microsoft Windows 1.0 is released.

1989. CD-ROMs become standard and hold much more data than floppy

disks. Large software programs can be distributed quickly, easily and relatively inexpensively.

1991. The Linux kernel, the basis for the open source Linux OS, is released.

1997. DVDs are introduced and able to hold more data than CDs, making it possible to put bundles of programs, such as the Microsoft Office Suite, onto one disk.

1999. Salesforce.com uses cloud computing to pioneer software delivery over the internet.

2000. The term software as a service (SaaS) comes into vogue.

2007. IPhone is launched and mobile applications begin to take hold.

2010 to the present. DVDs are becoming obsolete as users buy and download software from the internet and the cloud. Vendors move to subscription-based models and SaaS has become common.

# UNIT 3. Operating System

---

*Learning objectives*

- to understand what operating system is and how it works
- to acquire basic knowledge about different functions and characteristics of OS and to use a chart to organize the information
- to understand the role of OS

*Key words and phrases. Give Russian equivalents and remember the meanings of the key words and phrases.*

Core software program; to run; a consistent environment; to interact; command-line interface; graphical user interface; to utilize; to coordinate; device drivers; files and folders; to rename; to monitor system health; to retrieve data; software compatibility; open source; to be owned by smb.; to charge money; incompatible; edition; random access memory.

*Before reading the text B watch the video from https://edu.gcfglobal.org/en/computerbasics/understanding-operating-systems/1/ to get some information on the operating systems. Render it into Russian*

*Read the following text and do the exercises given after it*

Operating System (OS) is one of the core software programs that runs on the hardware and makes it usable for the user to interact with the hardware so that they can send commands (input) and receive results (output). It provides a consistent environment for other software to execute commands. So we can say that the OS acts at the center through which the system hardware, other software, and the user communicate. The following figure shows the basic working of the operating system and how it utilizes different hardware or resources.

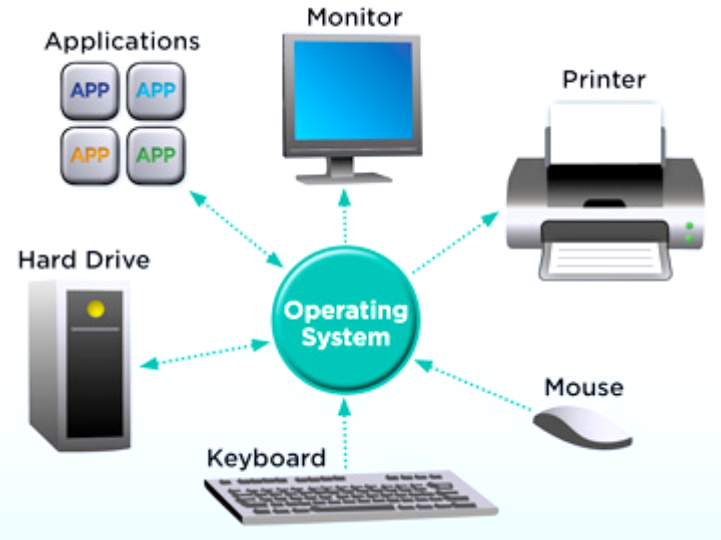



Fig. 4. Operating system working as a core part

## Basic Functions of the Operating system

The key five basic functions of any operating system are as following:

1. **Interface between the user and the hardware:** An OS provides an interface between user and machine. This interface can be a graphical user interface (GUI) in which users click onscreen elements to interact with the OS or a command-line interface (CLI) in which users type commands at the command-line interface (CLI) to tell the OS to do things.

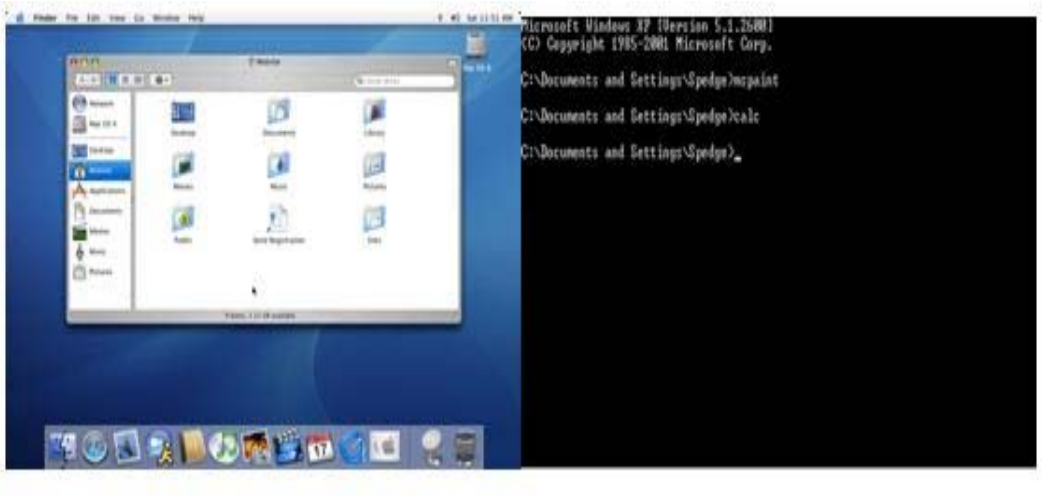



Fig. 5. GUI vs CLI

2. **Coordinate hardware components:** An OS enables coordination of hardware components. Each hardware device speaks a different language, but the operating system can talk to them through the specific translational software called device drivers. Every hardware component has different drivers for Operating systems. These drivers make the communication successful between the other software and the hardware.

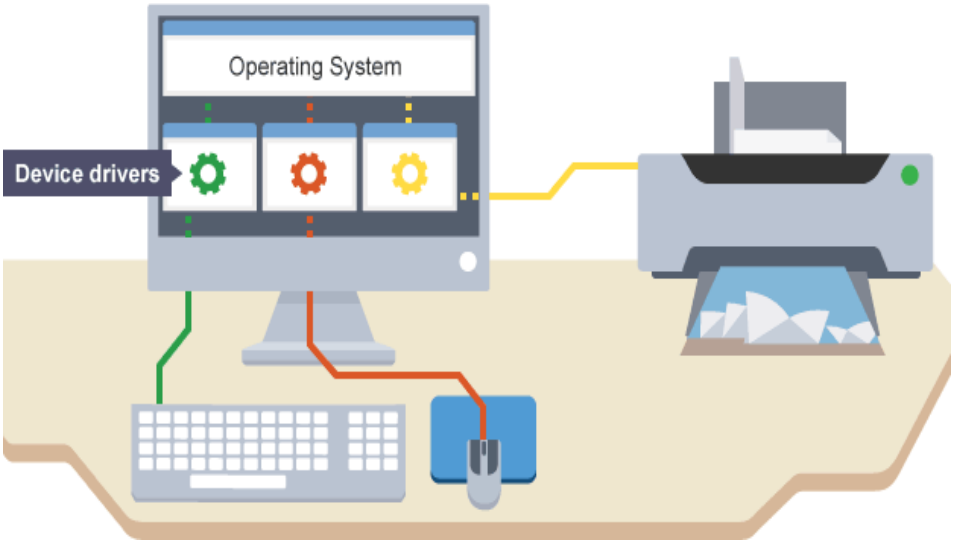



Fig. 6. Device Drivers in between OS and Hardware devices

3. **Provide environment for software to function:** An OS provides an environment for software applications to function. An application software is a specific software which is used to perform specific task. In GUI operating systems such as Windows and macOS, applications run within a consistent, graphical desktop environment.
4. **Provide structure for data management:** An OS displays structure/directories for data management. We can view file and folder listings and manipulate on those files and folders like (move, copy, rename, delete, and many others).
5. **Monitor system health and functionality:** OS monitors the health of our system's hardware, giving us an idea of how well (or not) it's performing. We can see how busy our CPU is, or how quickly our hard drives retrieve data, or how much data our network card is sending etc. and it also monitors system activity for malware.

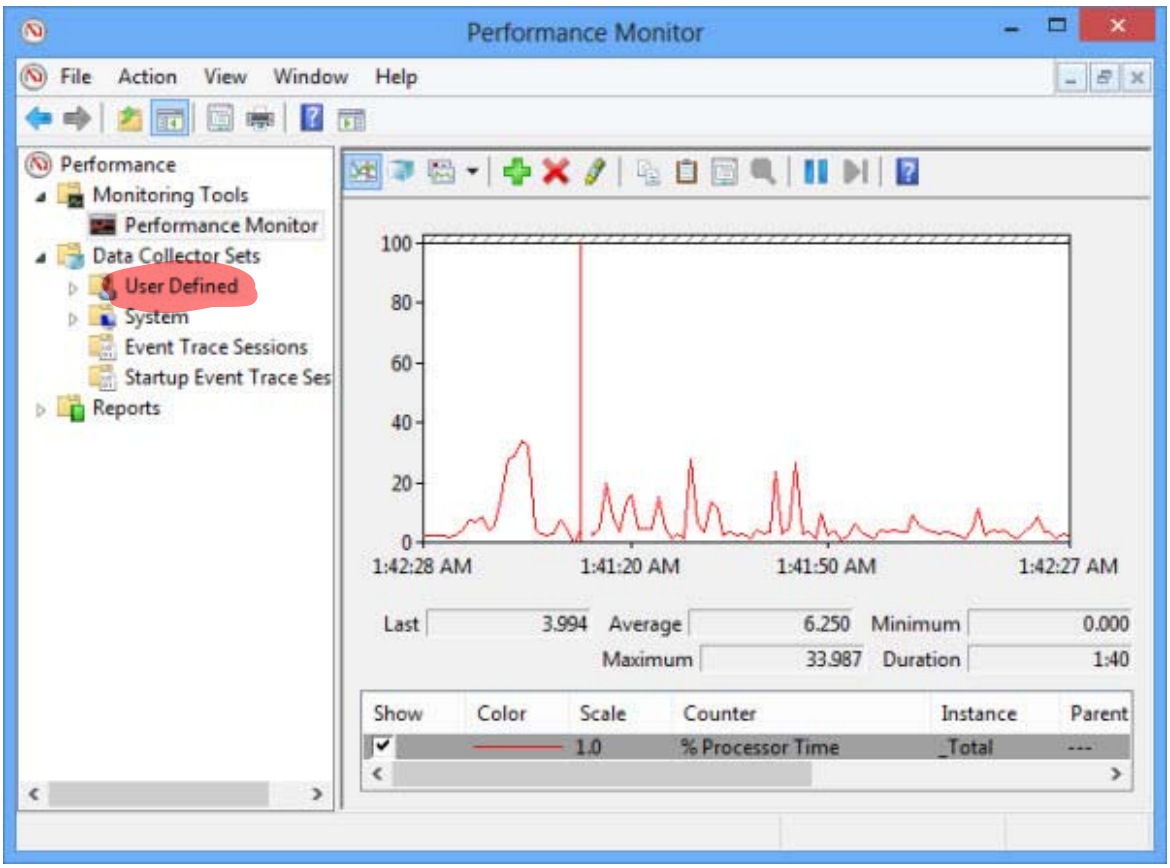



Fig. 7. Performance Monitor in windows

## Operating System Characteristics

The Operating systems are different according to the three primary characteristics which are licensing, software compatibility, and complexity.

***Licensing***

There are basically three kinds of Operating systems. One is Open Source OS, another is Free OS and the third is Commercial OS.

Linux is an ***open source*** operating system which means that anyone can download and modify it for example Ubuntu etc.

A ***free OS*** doesn't have to be open source. They are free to download and use but cannot modify them. For example, Google owns Chrome OS and makes it free to use.

***Commercial operating systems*** are privately owned by companies that charge money for them. Examples include Microsoft Windows and Apple macOS. These require to pay for the right (or *license*) to use their Operating systems.

***Software Compatibility***

The developers make the software which may be compatible or incompatible in different versions within the same operating system's type

but they can't be compatible with the other OS types. Every OS type has its own software compatibility.

**Complexity**

Operating systems come in basically two editions one is 32-bit and other is 64-bit editions. The 64-bit edition of an operating system best utilizes random access memory (RAM). A computer with a 64-bit CPU can run either a 32-bit or a 64-bit OS, but a computer with a 32-bit CPU can run only a 32-bit OS.


Fig. 8. 32-bit vs 64-bit Windows OS

[https://medium.com/computing-technology-with-it-fundamentals/operating-system-its-functions-and-characteristics-c0946e4215c6]

## 1. Text-based Assignments

*1.1. Give English equivalents of the following words and word combinations:*

Отправлять инструкции; обеспечивать среду; кликать на элементы на экране; программное обеспечение для перевода; выполнять определенную задачу; управление данными; удалять файлы; перемещать; жесткий диск; печатать команды; вредоносное ПО; главные характеристики; сложность; бесплатная операционная система; право пользования; совместимость; использовать наилучшим образом.

*1.2. Match the following words from the text with their meanings*

| | |
|---|---|
| 1. Device driver g | a) the smallest piece of information used by a computer. |
| 2. Bit a | b) system software that manages hardware, software, and resources, and provides services for other software. It will also usually provide an interface for the end user. |
| 3. Command line interface f | c) any software designed to do something that the user would not wish it to do, hasn't asked it to do, and often has no knowledge of until it's too late. |
| 4. GUI e | d) the level in difficulty in solving mathematically posed problems as measured by the time, number of steps or arithmetic operations, or memory space required. |
| 5. Malware c | e) a type of interface that allows the user to interact with a computer system. It usually involves clicking on icons or selecting options from a menu. |
| 6. Complexity d | f) a means of communication between a program and its user, based solely on textual input and output. Commands are input with the help of a keyboard or similar device and are interpreted and executed by the program. Results are output as text or graphics to the terminal. |
| 7. Operating system b | g) smth. that often forms part of the lowest level of the operating system kernel, with which they are linked when the kernel is built. Some more recent systems have loadable device drivers which can be installed from files. |

*1.3. Answer the following questions on the text*

1) What is the relationship between operating systems and computer hardware?
2) What is the main purpose of an operating system?
3) How many basic functions does an operating system have?
4) What devices make the communication successful between the other software and the hardware?

5) What does an OS display?
6) Is there any difference between Open Source OS and Free OS?
7) What does device driver software do?

*1.4. Read the text again and decide if the following statements are true or false.*

1) An OS provides a consistent environment for other software to execute commands. T
2) There are basically three kinds of Operating systems. One is Open Source OS, another is Free OS and the third is Network Operating System. F
3) The 64-bit edition of an operating system best utilizes read only memory (ROM). T
4) Every OS type has its own software compatibility. T
5) In GUI operating systems such as Windows and macOS, applications cannot run within a graphical desktop environment. F
6) Every hardware component has different drivers for Operating systems. T
7) Linux is one of the examples of commercial operating systems. F
8) OS monitors the health of our system's hardware. T

## 2. Focus on Grammar

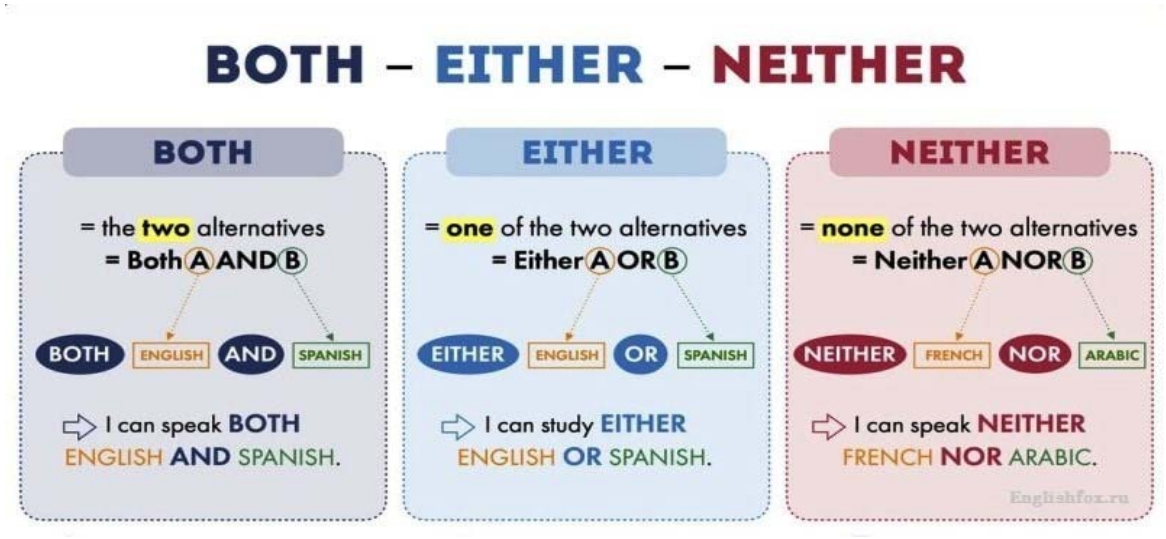
*2.1. Define tense and voice of the predicates in the following sentences and translate them into Russian*

1) Actually, the term "computer" is fast being replaced by the more appropriate term "electronic data processing machine".
2) Recently certain binary machines have been announced which will be capable of utilizing magnetic disc file memories.
3) This kind of computers will be equipped with a disc file of extremely high capacity and access speed.
4) In the past few years several designs have been advanced and some have actually been built.
5) Business variables have been and are being expressed as mathematical functions and are being statistically analyzed.
6) Eight distinguished speakers have been asked to consider the

potentialities and limitations of the computer in activities related to management.

7) The problem of designing a non-mechanical printer has already been studied in the central research laboratory.
8) The computer is given position, data and velocity vectors of the satellite for a given time.

*2.2. Insert either, neither or both. Translate the sentences into Russian*



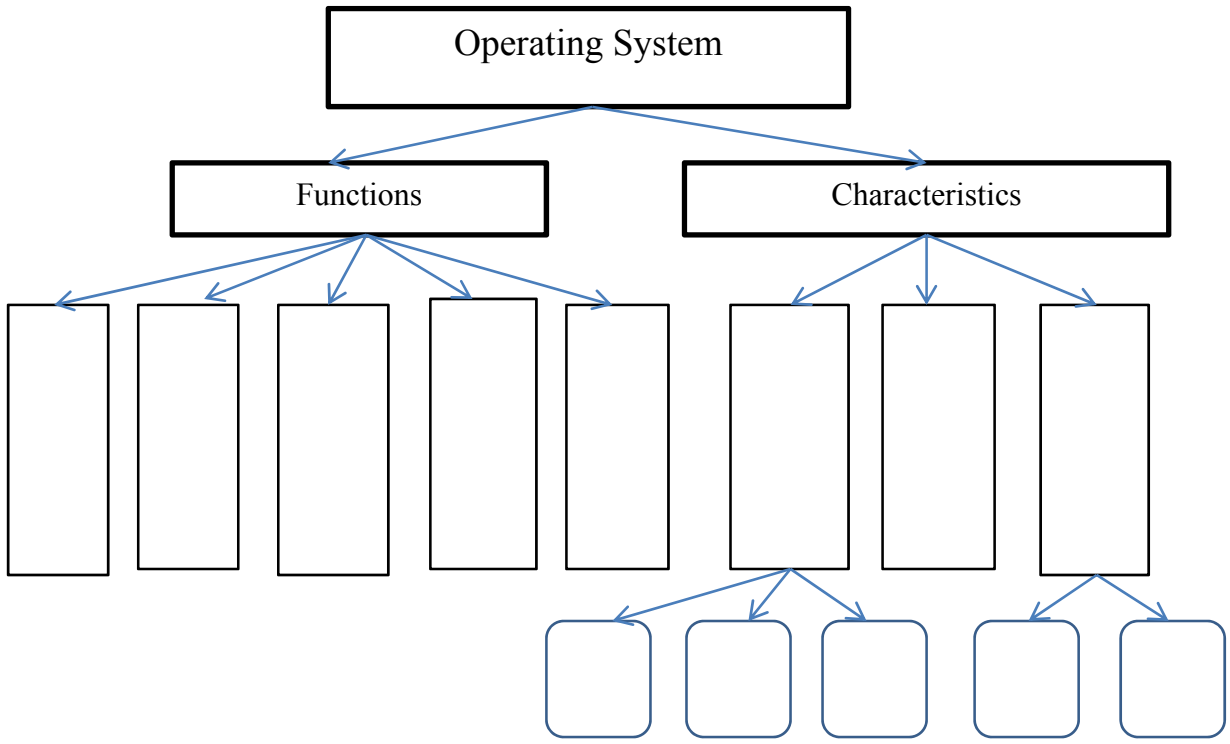

1) …operating systems (Linux and UBUNTU) are very good.
2) I don't want to install … Windows nor macOS.
3) They … laughed and one of the programmers looked down at his desk.
4) We were … in the office, but … of us spoke for some time.
5) **…**you apologize, or I'll never speak to you again.
6) Can **…** of you prepare a presentation?
7) … students passed the test.
8) Whatever **…** of you is thinking, you're wrong.
9) …he … I went to speak to the manager of our project.
10) I enjoy … the report and the presentation.

## 3. Discussion

*3.1. Discuss the following questions with your groupmates*

1) How would using a computer be different if it had no operating system? How would programming be different?
2) How would using a computer be different if it had no operating system? How would programming be different?
3) List several mental tasks that people do better than computers. List several mental tasks that computers do better than people. Can you find any general characteristics that distinguish the items on the two lists?
4) Some popular Operating Systems include Linux Operating System, Windows Operating System, VMS, OS/400, AIX, z/OS, etc. What OS does your home computer have? Are you satisfied with it? Have you had any troubles?

*3.2. According to the information in the text complete the graph and speak about functions and characteristics of OSs*



## 4. Additional reading

*4.1. Read and translate the following text about some types of Operating Systems*

### Batch operating system

The users of a batch operating system do not interact with the computer directly. Each user prepares his job on an off-line device like punch cards and submits it to the computer operator. To speed up processing, jobs with similar needs are batched together and run as a group. The programmers leave their programs with the operator and the operator then sorts the programs with similar requirements into batches.

### Time-sharing operating systems

Time-sharing is a technique which enables many people, located at various terminals, to use a particular computer system at the same time. Time-sharing or multitasking is a logical extension of multiprogramming. Processor's time which is shared among multiple users simultaneously is termed as time-sharing.

The main difference between Multiprogrammed Batch Systems and Time-Sharing Systems is that in case of Multiprogrammed batch systems, the objective is to maximize processor use, whereas in Time-Sharing Systems, the objective is to minimize response time.

Multiple jobs are executed by the CPU by switching between them, but the switches occur so frequently. Thus, the user can receive an immediate response. For example, in a transaction processing, the processor executes each user program in a short burst or quantum of computation. That is, if **n** users are present, then each user can get a time quantum. When the user submits the command, the response time is in few seconds at most.

The operating system uses CPU scheduling and multiprogramming to provide each user with a small portion of a time. Computer systems that were designed primarily as batch systems have been modified to time-sharing systems.

## Distributed operating System

Distributed systems use multiple central processors to serve multiple real-time applications and multiple users. Data processing jobs are distributed among the processors accordingly.

The processors communicate with one another through various communication lines (such as high-speed buses or telephone lines). These are referred as **loosely coupled systems** or distributed systems. Processors in a distributed system may vary in size and function. These processors are referred as sites, nodes, computers, and so on.

## Network operating System

A Network Operating System runs on a server and provides the server the capability to manage data, users, groups, security, applications, and other networking functions. The primary purpose of the network operating system is to allow shared file and printer access among multiple computers in a network, typically a local area network (LAN), a private network or to other networks.

Examples of network operating systems include Microsoft Windows Server 2003, Microsoft Windows Server 2008, UNIX, Linux, Mac OS X, Novell NetWare, and BSD.

## Real Time operating System

A real-time system is defined as a data processing system in which the time interval required to process and respond to inputs is so small that it controls the environment. The time taken by the system to respond to an input and display of required updated information is termed as the **response time**. So in this method, the response time is very less as compared to online processing.

Real-time systems are used when there are rigid time requirements on the operation of a processor or the flow of data and real-time systems can be used as a control device in a dedicated application. A real-time operating system must have well-defined, fixed time constraints, otherwise the system will fail. For example, Scientific experiments, medical imaging systems, industrial control systems, weapon systems, robots, air traffic control systems, etc.

There are two types of real-time operating systems.

***Hard real-time systems*** guarantee that critical tasks complete on time. In hard real-time systems, secondary storage is limited or missing

and the data is stored in ROM. In these systems, virtual memory is almost never found.

***Soft real-time systems*** are less restrictive. A critical real-time task gets priority over other tasks and retains the priority until it completes. Soft real-time systems have limited utility than hard real-time systems. For example, multimedia, virtual reality, Advanced Scientific Projects like undersea exploration and planetary rovers, etc.

[https://www.tutorialspoint.com/operating_system/os_types.htm]

*4.2. Fill in the table with information about advantages and disadvantages of OS types. You can use some Internet resources if necessary*

| Types | Advantages | Disadvantages |
|---|---|---|
| Batch operating system | … | Lack of interaction between the user and the job.<br>… |
| Distributed operating System | Better service to the customers.<br>… | … |
| Time-sharing operating systems | Provides the advantage of quick response.<br>… | Problem of reliability.<br>… |
| Network operating System | Security is server managed.<br>… | High cost of buying and running a server.<br>… |

*4.3. Answer the following questions and try to explain your answer*

## Test yourself

Question 1. Which one of the following is not software?
(A) MS-Word
(B) MS-Excel
(C) Keyboard
(D) Microsoft windows
Solution: _____

Question 2. Which one of the following is acts as an interface between the user and the computer hardware?
(A) Monitor
(B) Operating system
(C) User thread
(D) Application program
Solution: _____

Question 3. The only language that the computer can process or execute is called _____ ?
(A) Machine language
(B) Normal language
(C) Computer language
(D) High-level language
Solution:_____

Question 4. Which of the following software is used to control the operations of a computer?
(A) Application Software
(B) System Software
(C) Utility Software
(D) Language Processor
Solution: _____

Question 5. Which one of the following software is designed to solve a specific problem or to do a specific task?
(A) Language Processor
(B) Application Software
(C) System Software
(D) Utility Software
Solution: _____

Question 6. Which one of the following is not an example of an operating system?
(A) Linux
(B) Apple macOS
(C) Microsoft Windows,
(D) None of the above
Solution: _____

Question 7. Which of the following is a language processor?
(A) C++ programming language
(B) Compiler
(C) Linux
(D) All of the above
Solution: _____
[https://www.geeksforgeeks.org/software-and-its-types/]

*4.4. Translate the following sentences*

1. Операционная система является основным программным обеспечением, которое управляет всем аппаратным и другим программным обеспечением на компьютере.
2. Операционная система, также известная как «ОС», взаимодействует с аппаратным обеспечением компьютера и предоставляет службы, которые могут использовать приложения.
3. Операционные системы также включают в себя множество программных продуктов, таких как общие системные службы, библиотеки и интерфейсы прикладного программирования (API), которые разработчики могут использовать для написания программ, работающих в операционной системе.
4. Большинство программных приложений написано для операционных систем, что позволяет операционной системе делать много работы.
5. Например, при запуске Minecraft Вы запускаете его в операционной системе.
6. Minecraft не должен точно знать, как работает каждый отдельный аппаратный компонент.
7. Minecraft использует различные функции операционной системы, а операционная система переводит их в низкоуровневые аппаратные инструкции.
8. По количеству одновременно работающих программ операционные системы делят на однозадачные и многозадачные.

*Key answers to the test 4.3.*

1. The correct option is C, i.e., Keyboard. Because a keyboard is not software, as it is a hardware device (input device).
2. The correct option is B, i.e., Operating System. Because an operating system provides an interface to the user, which helps the user to interact with the computer system.
3. The correct option is A, i.e., Machine language. The only language that the computer can process or execute is called machine language as this language is capable of telling the computer explicitly what to do.
4. The correct option is B, i.e., System Software. There are two types of software: system software and application software. System Software is used to control the operations and also controls a computer's internal functioning and hardware devices.
5. The correct option is B, i.e., Application Software. Because a software that performs special functions or provides function which are much more than basic operation of the computer are application software.
6. The correct option is D, i.e., None of the above. Because Linux, Apple macOS, Microsoft Windows are the examples of operating systems.
7. The correct option is B, i.e., Compiler. Because a language processor is designed or used to convert program code to machine code. So, a compiler is a language processor and used in C/C++ programming language.

# UNIT 4. What is Computer Programming?
# Coding vs. Programming

---

*Learning objectives*

- to acquire basic knowledge about computer programming
- to understand the main purpose of programming
- to understand the difference between coding and programming

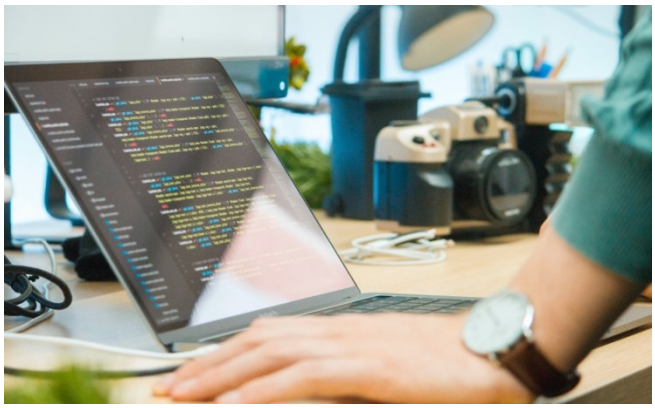
*Key words and phrases. Give Russian equivalents and remember the meanings of the key words and phrases.*

Programming language; to accomplish; intelligible; a sequence of instructions; expertise; source code maintenance; machine code; to process; reverse engineering; to assign; syntax; tools; outcome; proficient; application domain; central processing unit; coding basics; a piece of code; to have responsibility

*Before reading the text watch the video from https://www.youtube.com/watch?v=C9FlbhRVYdc&ab_channel=LiveLessons to get some information on the topic*

1. *Read the following text and do the exercises given after it*

**Programming** is the process of taking an algorithm and encoding it into a notation, a programming language, so that it can be executed by a computer. Although many programming languages and many different types of computers exist, the important first step is the need to have the solution. Without an algorithm there can be no program.



Computer science is not the study of programming. Programming, however, is an important part of what a computer scientist does. Programming is often the way that we create a representation for our solutions. Therefore, this language representation and the process of creating it becomes a fundamental part of the discipline.

What is computer programming?

- Computer programming is the process of designing and building an executable computer program to accomplish a specific computing result or to perform a specific task.

Programming involves tasks such as: analysis, generating algorithms, profiling algorithms' accuracy and resource consumption, and the implementation of algorithms in a chosen programming language (commonly referred to as **coding**).

The source code of a program is written in one or more languages that are intelligible to programmers, rather than machine code, which is directly executed by the central processing unit. The purpose of programming is to find a sequence of instructions that will automate the performance of a task (which can be as complex as an operating system) on a computer, often for solving a given problem. Proficient programming thus often requires expertise in several different subjects, including knowledge of the application domain, specialized algorithms, and formal logic.

Tasks accompanying and related to programming include: testing, debugging, source code maintenance, implementation of build systems, and management of derived artifacts, such as the machine code of computer programs. These might be considered part of the programming process, but often the term *software development* is used for this larger process with the term *programming*, *implementation*, or *coding* reserved for the actual writing of code. *Software engineering* combines engineering techniques with software development practices. *Reverse engineering* is a related process used by designers, analysts and programmers to understand and re-create/re-implement.

What is coding?

- Code is the language used by computers to understand and process our requests.

Coding is vital to the way our modern world operates, yet many people don't realize it. If you're on any webpage, just right click your mouse pad and click on 'View Page Source,' scroll through, and see if you can understand anything. There's a lot of information about the webpage there, which you probably don't know, but this is your first experience with coding basics (HTML, CSS, etc.).

Techopedia defines coding as "assigning a code or classification to something." This is essentially the way that humans communicate with machines. And, it all started in the 1950s, when the invention and development of coding languages was in full swing. Many of the coding languages created then are still used today, like FORTRAN, LISP and COBOL.

## Key differences between coding and programming

How do coding and programming fit into the software development industries? There are a few key differences to know between the two:

*Skills needed:* Coders don't necessarily have to be skilled in programming, but programmers should be knowledgeable with the bigger picture and should understand coding. Programmers not only have to write code, they also have to understand algorithms to ensure that the code they write is optimized in the best possible way.

*Difficulty:* If you want to be a coder, you need to learn all about programming language. As a programmer, you'll need to know this as well as how various algorithms work.

*Work scope:* When putting together software, as a coder, you're responsible for putting together a specific piece of code for part of the program. As a programmer, you have more responsibility as you need to look at the bigger picture and ensure that the software runs smoothly on all fronts.

The information in this chart from Hackr.io provides a great summary of the key differences between coding and programming.

| Key points | Coding | Programming |
|---|---|---|
| **Skills** | It is a process to convert a set of instructions into a language that the computer can understand | Has a wider scope so apart from coding it also involves defining requirements, writing pseudo code, testing and building executables |

deploy

| | | |
|---|---|---|
| **Scope** | As a coder, you need to know the syntax of the programming language | As a programmer, you need high-level thinking and analytical skills apart from coding skills |
| **Tools** | Eclipse, Bootstrap, Delphi, ATOM and many more | To add on to the coding tools other tools such as Git and Github, Database Tools, Analytical tools such as Apache Spark, Presentation tools, Cloud tools are also essential. |
| **Outcome** | A working piece of code | The whole application, a software product or a website |
| **Support** | Extensive developer community support is available | Extensive community support is available |

[https://www.lighthouselabs.ca/en/blog/coding-vs-programming]

## 1. Text-based Assignments

*1.1. Give English equivalents of the following words and word combinations:*

Кодирование алгоритма в нотацию; выполняться компьютером; языковое представление; исходный код программы; доступный для понимания; опытный; обслуживание исходного кода; реверсивное (обратное) проектирование; последовательность команд; просматривать путем прокрутки; используются до сих пор; изобретение; отвечать за что-л.; создание исполняемых кодов; помимо навыков кодирования; поддержка сообщества.

*1.2. Using suffixes **-al, -able, -ory, ive,** etc., give adjectives which are related to the following verbs. Use them in your own sentences*

to change
to program
to create
to regulate
to value
to structure
to expense
to control

*1.3. Match the terms with their definitions*

| | |
|---|---|
| 1. API h | a) a set of rules for grammar and spelling. In other words, it means using character structures that a computer can interpret. |
| 2. Software c | b) a collection of binary digits or bits that the computer reads and interprets. |
| 3. Syntax a | c) a collection of instructions that enable the user to interact with a computer, its hardware, or perform tasks. |
| 4. Machine code b | d) a process of analyzing a computer program and removing its logical or syntactical errors. |
| 5. Implement g | e) achieve or complete successfully |
| 6. Accomplish e | f) put into effect |
| 7. Debugging d | g) a document with information required by an individual or organization to apply for a position, financial grant, or another limited resource. |
| 8. Application f | h) a set of routines, protocols, and tools for building software applications. APIs allow programmers easier entry into another company's program or service. |

*1.4. Answer the following questions:*

1) What are the key differences between coding and programming?
2) What does proficient programming require?
3) What are the tasks accompanying and related to programming?
4) Do people realize the importance of programming in our modern world? Why?
5) How is the code program written?
6) What do you mean by extensive community support?

*1.5. Complete one of the profiles for the following positions. You can use some phrases, for example:*

I graduated in … My first job was in …
It involved monitoring and …
I am responsible for … operations. The job involves management too.
My background is… I was attracted by… I have always been fascinated by the…
Working abroad is a big priority for me too and that's one of the main reasons I chose …
I'm moving job this year to… for … years. I'll be based in an office in …
Another great thing about my job is that…
Professionally, when you're a woman you have to prove you're as good as a man.
… because so many aspects are technology dependent.
After my training period in… over, I plan to …
Another …years, I returned to Paris and spent … years working on designing….

| *Computer and IS Manager* | *System Administrator* | *Computer programmer* | *Database administrator* |
|---|---|---|---|
| | | | |

## 2. Focus on Grammar

*2.1. Complete the sentences with the correct tense form of the verbs in brackets*

**a. Active voice**

1. They (to test) the program and (to detect) the bugs by 3 p.m. tomorrow. 2. This company (to play) an important role in multimedia development since its very inception. 3. She never (to be able) to fix the problem. 4. They (not to install) the updates yet. 5. You ever (to watch) TV on the Internet? 6. He (to study) some high-level computer languages by next year.

**b. Passive Voice**

1. After the program (to be improved) it (to be published) as an updated version. 2. All the articles on programming languages (to be translated) by

next Friday. 3. Five networks for large companies (to be set up) recently. 4. A flowchart (to be designed) by 3 p.m. yesterday. 5. The program already (to be translated) into machine language.

*2.2. Transform the following sentences with the verbs in the Active Voice into Passive*

1) Once you've made a program, you can save it and share it with your friends and family.
2) Finally, after you do the calculation, you want to display the result on the screen.
3) Computer programmers use specialized languages to communicate with computers, applications and other systems to get computers and computer networks to perform a set of specific tasks.
4) George noted that students build software in online labs.
5) Programmers design Assembly language to be easily translated into machine language.
6) Dutch programmer Guido van Rossum developed the open-source language Python in 1991.
7) Another approach is to use a language designed for Web scripts for the browser execute it.

*2.3. Put in the to-infinitive form. (Some may be continuous, perfect or in Passive). Comment on the form of the infinitive.*
*Use these verbs:* ***take, talk, interfere, design, do, weigh, speak.***

1) It is very easy _____ about functionalities, because they are tangible.
2) It will enable us _____ the reports on our own.
3) He seemed _____ something in his mind.
4) There's nothing _____ with this new program.
5) You know how I hate _____in other people's business.
6) She was too timid _____.
7) He seemed _____his defeat quietly.

## 3. Discussion

### *3.1. Discuss the following questions*

1) What does the phrase “a good programmer” mean to you?
2) What abilities/skills should a good programmer have?
3) Api or library? Which is more important? https://www.youtube.com/watch?v=OVvTv9Hy91Q&ab_channel=SimplyExplained
4) Give some examples of the main programming areas.
5) Which programming languages are important now?

*3.2. Discuss with your partner some amazing* ***Great Programming Quotes*** *about software development (by Jeremy Morgan #programming)*

- "Programming isn't about what you know; it's about what you can figure out.” - *Chris Pine*

Especially important for beginners. At first, we're so anxious about knowing everything, especially language syntax. Problem-solving is the skill we end up using most.

- "The only way to learn a new programming language is by writing programs in it." - *Dennis Ritchie*

Programmers are mostly "learn by doing" types. No amount of academic study or watching other people code can compare to breaking open an editor and start making mistakes.

- "Sometimes it's better to leave something alone, to pause, and that's very true of programming." - *Joyce Wheeler*

When managing developers I would always encourage getting up and walking away from the computer when you have a problem. Some of your best solutions will come to you when you're not at the machine.

- "In some ways, programming is like painting. You start with a blank canvas and certain basic raw materials. You use a combination of science, art, and craft to determine what to do with them." - *Andrew Hunt*

Outsiders question whether programming is art. Programmers don't.

- "Testing leads to failure, and failure leads to understanding." - *Burt Rutan*

Some developers hate testing. However, shifting your attitude and embracing it makes you a better developer.

- "The best error message is the one that never shows up." - *Thomas Fuchs*

Remember this the next time you decide to focus on some great error reporting.

- “The most damaging phrase in the language is.. it's always been done this way” - *Grace Hopper*

To be a programmer long term, you have to love change. You can't just tolerate it, you have to love it.

## 4. Additional reading

*4.1. Read the text to yourself and answer the questions below to see how well you understand the topic*

Quantum computing has been seeing increased attention over the last decade, since these computers, which function according to the principles of quantum physics, have enormous potential. Today, most researchers believe that these computers will one day be able to solve certain problems faster than classical computers, since to perform their calculations they use entangled quantum states in which various bits of information overlap at a certain point in time. This means that in the future, quantum computers will be able to efficiently solve problems which classical computers cannot solve within a reasonable timeframe.

This quantum supremacy has still to be proven conclusively. However, some significant technical advances have been achieved recently. In late summer 2019, a quantum computer succeeded in solving a problem -- albeit a very specific one -- more quickly than the fastest classical computer.

For certain "quantum algorithms," i.e. computational strategies, it is also known that they are faster than classical algorithms, which do not exploit the potential of quantum computers. To date, however, these algorithms still cannot be calculated on existing quantum hardware because quantum computers are currently still too error-prone.

Utilizing the potential of quantum computation not only requires the latest technology, but also a quantum programming language to describe quantum algorithms. In principle, an algorithm is a "recipe" for solving a problem; a programming language describes the algorithm so that a computer can perform the necessary calculations.

Today, quantum programming languages are tied closely to specific hardware; in other words, they describe precisely the behaviour of the underlying circuits. For programmers, these "hardware description languages" are cumbersome and error-prone, since the individual programming instructions must be extremely detailed and thus explicitly describe the minutiae needed to implement quantum algorithms.

Computer scientists refer to computer languages that abstract from the technical details of the specific type of computer as high-level programming languages. Silq is the very first high-level programming language for quantum computers. High-level programming languages are more expressive, meaning that they can describe even complex tasks and algorithms with less code. This makes them more comprehensible and easier to use for programmers. They can also be used with different computer architectures.

[Abridged from
https://www.sciencedaily.com/releases/2020/06/200615115820.htm ]

1) Utilizing the potential of quantum computation requires _____.
a) the latest technology
b) a quantum programming language and the latest technology
c) a quantum programming language

2) High-level programming languages can describe _____.
a) complex tasks and algorithms with less code.
b) complex tasks and algorithms with much code
c) complex algorithms

3) Today, _____are tied closely to specific hardware.
a) software programs
b) quantum programming languages
c) high-level languages

4) For programmers, these "hardware description languages" are _____.
a) easily managed and error-prone
b) not easily managed and error
c) cumbersome and error-prone

5) Computer scientists refer to computer languages that abstract from the technical details of the specific type of computer as _____.
   a) high-level programming languages
   b) low-level programming languages
   c) high-level and low-level programming languages

*4.2. Read and translate the following text*

**How the brain is programmed for computer programming?**

Countries around the world are seeing a surge in the number of computer science students. Enrolment in related university programs in the U.S. and Canada tripled between 2006-2016 and Europe too has seen rising numbers. At the same time, the age to start coding is becoming younger and younger because governments in many different countries are pushing K-12 computer science education. Despite the increasing popularity of computer programming, little is known about how our brains adapt to this relatively new activity. A new study by researchers in Japan has examined the brain activity of thirty programmers of diverse levels of expertise, finding that seven regions of the frontal, parietal and temporal cortices in expert programmer's brain are fine-tuned for programming. The finding suggests that higher programming skills are built upon fine-tuned brain activities on a network of multiple distributed brain regions.

"Many studies have reported differences between expert and novice programmers in behavioural performance, knowledge structure and selective attention. What we don't know is where in the brain these differences emerge," says Takatomi Kubo, an associate professor at Nara Institute of Science and Technology, Japan, and one of the lead authors of the study.

To answer this question, the researchers observed groups of novices, experienced, and expert programmers. The programmers were shown 72 different code snippets while under the observation of functional MRI (fMRI) and asked to place each snippet into one of four functional categories. As expected, programmers with higher skills were better at correctly categorizing the snippets. A subsequent searchlight analysis revealed that the amount of information in seven brain regions strengthened with the skill level of the programmer: the bilateral inferior frontal gyrus pars triangularis (IFG Tri), left inferior parietal lobule (IPL), left supramarginal gyrus (SMG), left middle and inferior temporal gyri (MTG/IT), and right middle frontal gyrus (MFG).

"Identifying these characteristics in expert programmers' brains offers a good starting point for understanding the cognitive mechanisms behind programming expertise. Our findings illuminate the potential set of cognitive functions constituting programming expertise," Kubo says.

More specifically, the left IFG Tri and MTG are known to be associated with natural language processing and, in particular, semantic knowledge retrieval in a goal-oriented way. The left IPL and SMG are associated with episodic memory retrieval. The right MFG and IFG Tri are functionally related to stimulus-driven attention control.

"Programming is a relatively new activity in human history and the mechanism is largely unknown. Connecting the activity to other well-known human cognitive functions will improve our understanding of programming expertise. If we get more comprehensive theory about programming expertise, it will lead to better methods for learning and teaching computer programming," Kubo says.

*4.3. Write a short summary of the text (Task 4.2) in English*

*4.4. Translate the following sentences*

1. Техники-программисты работают в вычислительных центрах, IT-компаниях, банках, образовательных учреждениях. Они занимаются разработкой программного обеспечения, устранением неполадок в работе вычислительной техники, наладкой оборудования, обучением пользователей.
2. Программирование основывается на использовании языков программирования, на которых записываются исходные тексты программ.
3. После того, как было принято решение о возможности программной реализации поставленной задачи, необходимо построить алгоритм её решения.
4. Выбирая профессию программиста, следует быть готовым к тому, что учеба не кончится ни после университета, ни после получения высокой должности. Эта специфика, в первую очередь, появляется из-за того, что сфера информационных технологий достаточно молода и постоянно развивается.
5. Программирование – это объяснение машине что, в каком виде и как нужно получить пользователю. То есть это своеобразное искусство перевода пожеланий человека на язык машины.

shell (kabuk), bilgisayardaki programları çalıştırmak için bir komut satırı arabirimidir.

# UNIT 5. Programming Languages

---

*Learning objectives*

- to acquire (al / elde etmek) basic knowledge (bilgi) about programming languages as an engineering tool
- to understand the main difference between high-level and low-level (düşük seviyeli) languages
- to understand the role of assembly language in computer programming

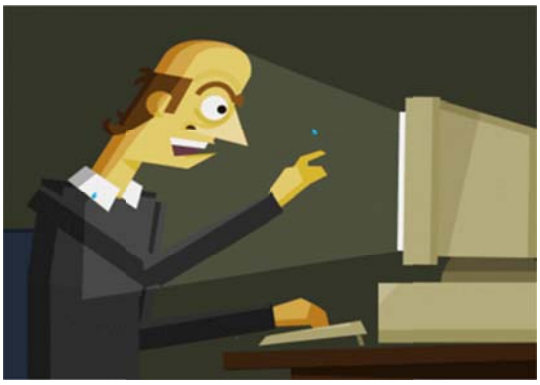
*Key words and phrases. Give Russian equivalents and remember the meanings of the key words and phrases.*

Talimatların toplanması

Collection of instructions; low-level languages; high-level languages; sequences (sıralama); assembly language; sophisticated means (gelişmiş yöntemler); peripheral devices (çevresel aygıtlar); syntax (söz dizimi); client-server application (istemci-sunucu); shell scripting languages (çok yönlü shell (kabuk) komutları); markup language (biçimleme); to compile (derlemek); to interpret (yorumlamak); statements (açıklama, komut); expertise (bilir kişi); imperative form (zorunlu); declarative form (bildirim deyimi); to split into (böl); extension (uzantı); to be treated as (işlem görmüş); dominant implementation (son cihaz / tamamlanmış cihaz / uygulama sonucu / uygulama); uzantı dosyaları adı

*Before reading the text watch the video from https://www.youtube.com/watch?v=EGQh5SZctaE&ab_channel=Codecademy to get some information on the topic. What do you already know about the topic?*

*Read the following text and do the exercises given after it*



Programming is an important engineering tool. It is a process of writing a computer program using computer language. Computer programs are collections of instructions that tell a computer how to interact with the user and the computer hardware and how to process data. Our work would have been very demanding and time consuming without programming.

Programming languages can be classified as either low-level languages or high-level languages. Low-level programming languages or machine languages are the most basic type of programming languages and can be understood directly by a computer. It is extremely tedious to program directly in machine language because instructions are written as sequences of 1s and 0s called bits. Assembly languages are used to make machine language programs easier to write. For example, assembly languages use abbreviations such as ADD, SUB, MPY to represent instructions. The program is then translated into machine language by software called an assembler.

Assembly language is designed to be easily translated into machine language. Although blocks of data may be referred to by name instead of by their machine addresses, assembly language does not provide more sophisticated means of organizing complex information. Like machine language, assembly language requires detailed knowledge of internal computer architecture. It is useful when such details are important, as in programming a computer to interact with peripheral devices (printers, scanners, storage devices, and so forth).

High-level languages are relatively sophisticated sets of statements utilizing words and syntax from human language and therefore easier to read, write, and maintain. Examples of high-level languages are Pascal (widely used as a beginner or as a teaching language), C (used to write system software, graphics and commercial programs), C++ (primarily utilized with system / application software, drivers, client-server applications), Cobol (popular for business applications), Fortran (used for scientific and mathematical applications), Java (designed to run on the Web), Visual Basic (used to create Windows applications) and shell scripting languages such as those found in the UNIX, Linux and Mac OS X environment. The languages used to create Web documents are called markup languages, they use instructions (markups) to format and link text files, for example, HTML (Hypertext Markup Language).

Regardless of what language you use you need to translate it into machine language so that a computer can understand and process it. There are two ways to do this: to compile the program and interpret the program.
In a compiled language, the programmer writes more general instructions and a compiler (a special piece of software) automatically translates these high level instructions into machine language. The machine language is then executed by the computer. A large portion of software in use today is

programmed in this way. In an interpreted programming language, the statements that the programmer writes are interpreted as the program is running. This means they are translated into machine language on the fly and then are executed as the program is running.

People communicate instructions to the computer in programming languages and the choice of the language depends on the type of computer, the sort of program, the expertise of the programmer, etc. [10].

Thousands of different programming languages have been created, and more are being created every year. Many programming languages are written in an imperative form (i.e., as a sequence of operations to perform) while other languages use the declarative form (i.e. the desired result is specified, not how to achieve it).

The description of a programming language is usually split into the two components of syntax (form) and semantics (meaning). Some languages are defined by a specification document (for example, the C programming language is specified by an ISO Standard) while other languages (such as Perl) have a dominant implementation that is treated as a reference. Some languages have both, with the basic language defined by a standard and extensions taken from the dominant implementation being common.

Programming language theory is a subfield of computer science that deals with the design, implementation, analysis, characterization, and classification of programming languages.

[https://en.wikipedia.org/wiki/Programming_language]

## 1. Text-based Assignments

*1.1. Give English equivalents of the following words and word combinations:*

Взаимодействие с пользователем; непосредственно; аббревиатура; называемых битами; перевести на машинный язык; сложная информация; внутренняя архитектура компьютера; выражения; в первую очередь; язык разметки гипертекстов; компилируемый язык; интерпретируемый язык; большая доля ПО; немедленно, «на лету»; опыт программиста; нормативный документ; справочное описание, указатель.

*1.2. Match the following words with their definitions*

| | |
|---|---|
| 1. Syntax<br>c | a) the creation of an executable program from code written in a compiled programming language. |
| 2. High-level Language<br>e | b) a program used to convert or translate programs written in assembly code by humans to machine code (binary) that can be understood by the computer. |
| 3. Compile<br>a | c) a set of rules for grammar and spelling. In other words, it means using character structures that a computer can interpret. |
| 4. A script or scripting language<br>f | d) that consists of easily understood keywords, names, or tags that help format the overall view of a page and the data it contains. E.g. BBC, HTML, SGML, and XML. |
| 5. Assembler<br>b | e) is a computer programming language that isn't limited by the computer, designed for a specific job, and is easier to understand. It is more like human language and less like machine language. |
| 6. Markup language<br>d | f) is a computer language with a series of commands within a file capable of being executed without being compiled. E.g. Perl, PHP, and Python, JavaScript. |

*1.3. Make up English-Russian pairs of the words equivalent in meaning:*

1. Time-consuming, 2. sophisticated, 3. to run, 4. client-server application, 5. environment, 6. tedious, 7. to treat.

1) Высокой сложности, 2) интерпретировать, 3) трудоемкий, 4) утомительный, 5) запускать на выполнение, 6) внешние устройства системы, 7) клиент-серверное приложение.

*1.4. Read the text again and decide if the following statements are true or false*

1) Computer programs are collections of instructions that tell a user how to interact with a computer. T
2) Our work would have been very demanding and easy without programming. F
3) High-level programming languages or machine languages are the most basic type of programming languages and can be understood directly by a computer. F
4) Assembly language is designed to be easily translated into machine language. T
5) High-level languages are relatively sophisticated sets of statements utilizing words and syntax from human language and therefore easier to read, write, and maintain. T
6) Visual Basic is used to write system software, graphics and commercial programs. T F
7) In a compiled language, the programmer writes more general instructions and a compiler (a special piece of software) automatically translates these high level instructions into machine language. F?
8) The description of a programming language is usually split into the three components of syntax (form) and semantics (meaning) and links between them. T

*1.5. Answer the following questions on the text*

1) Which programming languages do you use? Why do you use them?
2) What are the examples of high-level languages?
3) Assembly language doesn't require detailed knowledge of internal computer architecture, does it?
4) What does "to compile the program" mean?
5) What does programming language theory deal with?
6) What would happen if you forgot to include the correct punctuation in a statement?

## 2. Focus on Grammar

*2.1. Choose the right variant*

### Algorithmic languages

1) Algorithmic languages _____to express mathematical or symbolic computations.
   A) are designed B) designed C) was designed
2) The C programming language _____ in 1972 by Dennis Ritchie and Brian Kernighan at the AT&T Corporation for programming computer operating systems.
   A) has been developed B) was developed C) developed
3) C _____with assembly language the power to exploit all the features of a computer's internal architecture.
   A) share B) share C) is shared
4) Its capacity to structure data and programs through the composition of smaller units is _____to that of ALGOL.
   A) comparable B) compare C) comparability
5) ALGOL had recursive subprograms—procedures that _____invoke themselves to solve a problem by reducing it to a smaller problem of the same kind.
   A) can B) might C) could
6) ALGOL contributed a notation for _____the structure of a programming language, Backus–Naur Form, which in some variation became the standard tool for stating the syntax (grammar) of programming languages.
   A) describing B) description C) being described
7) ALGOL introduced block structure, in which ____ program is composed of blocks that might contain both data and instructions and have the same structure as an entire program.
   A) the B) ---- C) a
8) C uses a compact notation and provides the programmer _____the ability to operate with the addresses of data as well as with their values.
   A) with B) --- C) for

9) The first important algorithmic language was FORTRAN (*for*mula *tran*slation), designed in 1957 by an IBM team _____ John Backus.
A) leading B) lead by C) led by

10) FORTRAN _____for scientific computations with real numbers and collections of them organized as one- or multidimensional arrays.
A) was taken B) was intended C) was composed

## 3. Discussion

*3.1. Discuss the following questions in small or large groups. You can use the Internet resources if necessary*

1) What challenges did you run into when choosing these languages?
2) Your friend says: "I just started learning to code, and at times, I feel like I'm simply not smart enough to understand concepts right away. How do I prevent getting stuck and what's the best way to keep learning and moving through material?" What would you suggest?
3) How do you identify a problem in programming?

*3.2. Give a short explanation of the computer term 'programming languages'*

*3.3. Suggested topics for Presentations and Reports*

1) Types of programming languages.
2) Business-oriented languages – SQL (structured query language) and COBOL (*co*mmon *b*usiness *o*riented *l*anguage).
3) The main problems faced by programmers.

## 4. Additional Reading

*4.1. Read and translate the following text 'Programming Paradigms'*

Programming languages mimic the operations of the computer they are running on. Therefore the computer they are designed for has a significant effect of how the programming language is created and which

characteristics are attributed to the language. Various attributes of a programming language will determine the computational paradigm of the language. The following are different paradigms.

**Imperative Paradigm:** Instructions are executed sequentially, variables are used to represent memory locations, and assignments are used change the values of variables. Imperative languages are also referred to as procedural languages, due to the sequence of statements that represent the commands. Most programming languages currently used are imperative.

**Functional Paradigm:** Based on mathematics and the abstract notion of a function in lambda calculus. This paradigm bases the description of computation on the evaluation of functions or the application of functions to known values. Languages incorporating the functional paradigm are sometimes called *applicative languages*. The functional paradigm uses a functional call, where the program evaluates a function, transfers values as parameters to certain functions, and returns values from functions. LISP is an example of a functional programming language.

**Logic paradigm:** Logic programming is based on the symbolic logic. These languages are based on a set of statements that describe the truth of a statement, rather than giving a sequence of sentences that are restricted to be executed in a particular manner. These languages have no need for loops, and the only necessity is the statement of properties of the computation. Since all properties are declared and there is no sequence of execution, logic programming is referred to as declarative programming. The only widely used logic based language is Prolog.

**Object-oriented Paradigm:** This paradigm is based on the idea of an object. Objects can be described as a collection of locations together with all the operations that can change the values of these memory locations. An example of an object is a variable. In many object-oriented languages, objects are put into classes that represent all of the objects with the same characteristics. These classes define four things. First, a constructor allocates memory and provides an initial value for the data of an object. Second, a way to access the value from the first part of the class is determined. Then the procedures are executed and a value is defined. Object-oriented programming is found in numerous new languages and seems to be a staple for the future of programming.

```
// Purpose:   This program find the absolute value of an
integer
//          without using a function
//
#include <iostream>
using namespace std;
int main()
{
     int number;
     int abs_number;
     // Ask for input
     cout << "This program finds the absolute value of an
integer." << endl;
     cout << "Enter an integer (positive or negative): ";
     cin >> number;
          // Find the absolute value
     if(number >= 0)
     {
          abs_number = number;
     }
     else
          abs_number = -number;
     // Print out output
     cout << "The absolute value of " << number << " is "
<< abs_number;
     cout << endl;
     return 0;
}
```

[http://www.csun.edu/~vgc30838/Projecth.html]

*4.2. Write a short summary of the text given above (Task 4.1.)*

*4.3. Answer the following questions*

## Test yourself

### Introduction to programming test questions

1) What is a programming language?
   A) Written English
   B) An artificial language used to program a computer
   C) A language used in pseudocode
2) What is machine code?
   A) The serial number of a computer
   B) A programming language that a computer understands
   C) The make and model of a computer
3) What is a program?
   A) A series of step-by-step instructions that tell a computer how to solve a task
   B) A video that is watched on a computer

C) A flowchart written on a computer

4) What is a statement**?**
   A) A box in a flowchart
   B) A keyword in a programming language
   C) A calculation performed in a programming language
5) What is an instruction?
   A) A box in a flowchart
   B) A calculation performed in a programming language
   C) One or more statements grouped together to instruct the computer to perform a task
6) What does the statement 'print' do?
   A) Output a hard copy of a program to a printer
   B) Output a message on the screen
   C) Print a hard copy of a flowchart to a printer
7) What does the statement 'while' do?
   A) Tell the computer to wait for a while before continuing with the program
   B) Implement selection
   C) Implement a loop
8) What does the statement 'def' do?
   A) Creates a function or a procedure
   B) Implement a loop
   C) Implement selection
9) What do the statements 'if' and 'else' do?
   A) Implement selection
   B) Implement a loop
   C) Tell the computer to wait for a while before continuing with the program
10) How many statements are there in this line of code: print ("If I am 17, I can drive a car")?
   A) There are two statements - 'print' and 'if'
   B) There are no statements
   C) There is one statement - 'print'

[https://www.bbc.co.uk/bitesize/guides/zts8d2p/test]

*4.4. Translate the following sentences*

1) Язык программирования – это набор лексических, синтаксических и семантических правил, которые придумали люди, чтобы создавать программы.
2) Изучить язык до начального уровня можно за 6–10 месяцев, но если ошибиться с выбором, язык может устареть, а вы потеряете время и деньги.
3) Чтобы отслеживать востребованность языков программирования, компании составляют специальные рейтинги.
4) C – один из самых старых и популярных языков программирования. Он «легкий» и быстрый, поэтому его используют там, где нужна высокая производительность. Например, для создания драйверов, операционных систем или ПО для микроконтроллеров.
5) Java – кроссплатформенный (cross platform) язык с большим количеством библиотек и большим сообществом разработчиков.
6) Кроссплатформенность – это возможность написать программу один раз и сразу пользоваться ей на нескольких операционных системах: Windows, Linux и MacOS.

Благодаря библиотекам Java подойдет практически для всего: работы с графикой, звуком, создания небольших игр.

*Key answers to the task 4.3.*
1. B); 2. B); 3. A); 4. B); 5. C); 6. B); 7.C); 8. A); 9. A); 10. C).

Paradigm

course → yönelim
Major → uzmanlık alanı

# UNIT 6. Object-oriented Programming (OOP)

---

*Learning objectives*

- to acquire basic knowledge about object-oriented programming and languages types in it
- to understand the main features of object-oriented programming
- to consider the difference between OOP and traditional programming

*Key words and phrases. Give Russian equivalents and remember the meanings of the key words and phrases used in the text*

Self-contained entities; formal set of rules; relational database; object database; encapsulation; inheritance; code modularity; routines; at the bottom of a hierarchy; to be inherited; polymorphism; to be integrated; memory allocation; to dispense; "event-driven" programming; to succeed; portable; data addresses; feature; to extend; interrelated applications; to be similar to smth.

*Read the following text and do the exercises given after it*

inwich = wherein

A programming language structure wherein the data and their associated processing ("methods") are defined as self-contained entities called "objects." The norm today, object-oriented programming (OOP) languages, such as C++ and Java, provide a formal set of rules for creating and managing objects. The data are stored in a traditional relational database or in an object database if the data have a complex structure.

depolama → conventional

There are three major features in object-oriented programming that makes them different than non-OOP languages: encapsulation, inheritance and polymorphism.

özellik, öz nitelik

**Encapsulation Enforces Modularity**

parçalı olmak

Encapsulation refers to the creation of self-contained modules that bind processing functions to the data. These user-defined data types are

işaret eder, refere eder

blueforce attack

called "classes," and one instance of a class is an "object." For example, in a payroll system, a class could be Manager, and Pat and Jan could be two instances (two objects) of the Manager class. Encapsulation ensures good code modularity, which keeps routines separate and less prone to conflict with each other.

**Inheritance Passes "Knowledge" Down**

Classes are created in hierarchies, and inheritance allows the structure and methods in one class to be passed down the hierarchy. That means less programming is required when adding functions to complex systems. If a step is added at the bottom of a hierarchy, only the processing and data associated with that unique step needs to be added. Everything else is inherited. The ability to reuse existing objects is considered a major advantage of object technology.

**Polymorphism Takes any Shape**

Object-oriented programming allows procedures about objects to be created whose exact type is not known until runtime. For example, a screen cursor may change its shape from an arrow to a line depending on the program mode. The routine to move the cursor on screen in response to mouse movement would be written for "cursor," and polymorphism allows that cursor to take on whatever shape is required at runtime. It also allows new shapes to be easily integrated.

The following compares basic OOP terms with traditional programming.

| ***OOP*** | ***Traditional Programming*** |
|---|---|
| | |
| class | define data + processing |
| object | data + processing |
| attribute | data (a field) |
| method | function |
| message | function call |
| instantiate | allocate a structure |

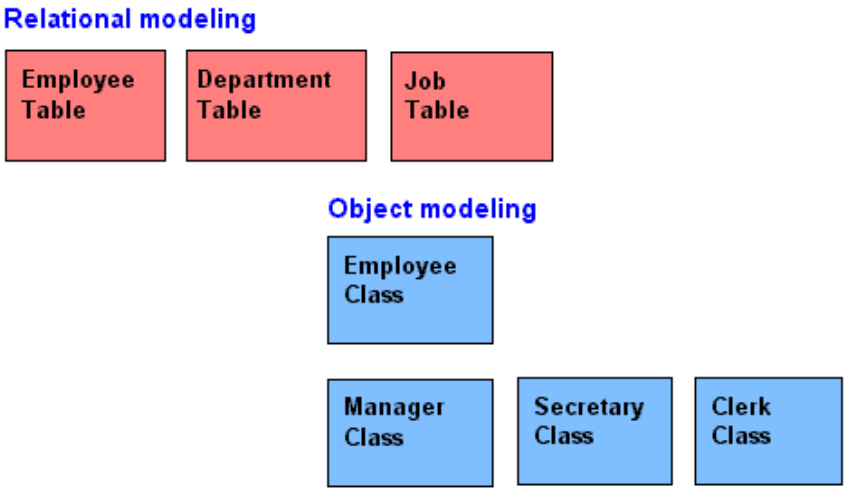



Fig. 9. Relational vs. Object Modeling

Instead of separate employee, department and job tables, an employee class contains the data and processing for all employees. Each subclass (manager, secretary, etc.) has its own data and processing but also inherits everything from the employee class. Changes made to the employee class affect every subclass.

## OOP Languages

Today, C++, C#, Java, JavaScript, Visual Basic.NET and Python are popular object-oriented languages.

The ***C++ language***, developed in the mid-1980s, extended C by adding objects to it while preserving the efficiency of C programs. It has been one of the most important languages for both education and industrial programming. Large parts of many operating systems were written in C++. C++, along with Java, has become popular for developing commercial software packages that incorporate multiple interrelated applications. C++ is considered one of the fastest languages and is very close to low-level languages, thus allowing complete control over memory allocation and management. This very feature and its many other capabilities also make it one of the most difficult languages to learn and handle on a large scale.

***C#*** (pronounced *C sharp* like the musical note) was developed in 2000. C# has syntax similar to that of C and C++ and is often used for developing games and applications for the Microsoft Windows operating system.

In the early 1990s ***Java*** was designed by Sun Microsystems, Inc., as a programming language for the World Wide Web (WWW). Although it resembled C++ in appearance, it was object-oriented. In particular, Java dispensed with lower-level features, including the ability to manipulate

data addresses, a capability that is neither desirable nor useful in programs for distributed systems. In order to be portable, Java programs are translated by a Java Virtual Machine specific to each computer platform, which then executes the Java program. In addition to adding interactive capabilities to the Internet through Web "applets," Java has been widely used for programming small and portable devices, such as mobile telephones.

***Visual Basic*** was developed by Microsoft to extend the capabilities of BASIC by adding objects and "event-driven" programming: buttons, menus, and other elements of graphical user interfaces (GUIs). Visual Basic can also be used within other Microsoft software to program small routines. Visual Basic was succeeded in 2002 by Visual Basic .NET, a vastly different language based on C#, a language with similarities to C++.

The open-source language ***Python*** was developed by Dutch programmer Guido van Rossum in 1991. It was designed as an easy-to-use language, with features such as using indentation instead of brackets to group statements. Python is also a very compact language, designed so that complex jobs can be executed with only a few statements. In the 2010s, Python became one of the most popular programming languages, along with Java and JavaScript.

[https://www.computerlanguage.com/results.php?definition=object-oriented+programming]
[https://www.britannica.com/technology/computer-programming-language/]

## 1. Text-based Assignments

*1.1. Give English equivalents of the following words and word combinations:*

В котором данные и методы; самодостаточный; сохранять в базе данных; особенности; наследование свойств; пример; обеспечивать; модульность кода; имеющий тенденцию; добавление функций; способность повторного использования; основное преимущество; время запуска; в ответ на; программный режим; влиять на что-л.; сохранять; легкий в использовании; добавление отступов; вместо скобок; компактный язык.

*1.2. Put the phrases below into the right word order. Use them in the sentences of your own*

a) called self-contained "objects" entities – _____
b) types data user-defined – _____
c) a hierarchy is of added the bottom at – _____
d) an arrow change from shape to a its line – _____
e) programs the efficiency while of preserving C – _____
f) with complex statements executed jobs can only be a few – _____

*1.3. Match the following words with their definitions*

| | |
|---|---|
| 1. Modularity e | a) a database that maintains a set of separate, related files (tables), but combines data elements from the files for queries and reports when required. |
| 2. relational database a | b) run a program, which causes the computer to carry out its instructions. |
| 3. polymorphism f | c) a database that is managed by an object-oriented database management system (ODBMS). |
| 4. object database c | d) In object technology, the creation of self-contained modules that contain both the data and the processing. |
| 5. encapsulation d | e) the characteristic of a system that has been divided into smaller subsystems which interact with each other. |
| 6. execute b | f) meaning many shapes. In object technology, polymorphism is exhibited when a request (message) produces different results based on the object that it is sent to. |

*1.4. Answer the following questions on the text*

1) Where are the data stored in OOP?
2) What are the major features in OOP?
3) What does encapsulation refer to and what does it ensure?
4) How are classes created?
5) What is the difference between C++ and C #?

6) When was the open-source language Python developed?
7) What does ‘a very compact language’ mean?

*1.5. Read the text again and decide if the following statements are true or false*

1) Object-oriented programming languages, such as C++ and Java, provide a formal set of rules for creating and managing objects. The data are stored in a traditional relational database. T
2) There is one major feature in object-oriented programming that makes them different than non-OOP languages. It is polymorphism. F
3) Classes are created in hierarchies, and inheritance allows the structure and methods in one class to be passed down the hierarchy. T
4) Object-oriented programming allows procedures about objects to be created whose exact type is not known until runtime. T
5) The C++ language, developed in 2000s, extended C by adding objects to it while preserving the efficiency of C programs. F
6) In the early 1990s Java was designed by Sun Microsystems, Inc., as a programming language for the World Wide Web (WWW). T
7) Visual Basic can also be used within other Microsoft software to program small routines. T
8) Java and JavaScript are not used for programming small and portable devices, such as mobile telephones. T
9) In the 2010s, Python became one of the most popular programming languages, along with Java and JavaScript. TF

## 2. Focus on Grammar

*2.1.* We can join two independent clauses (sentences) together using ***conjunctive*** adverbs. Conjunctive adverbs show cause and effect, sequence, contrast, comparison, or other relationships.
The most common of these are:

| | | |
|---|---|---|
| Accordingly | Indeed | Otherwise |
| Afterwards | Likewise | Similarly |
| Also | Moreover | Still |
| Consequently | Nevertheless | Therefore |
| However | Nonetheless | |

*2.2. Learn the following rules and examples. Make up your own sentences with some of the conjunctive adverbs listed above.*

## Learning the Rules: Examples of Conjunctive Adverbs

### Rule 1: Complete Sentences Connected With an Adverbial Require a Semicolon

***Complete sentence +; + adverbial connecting word + complete sentence.***

E.g. Jeffrey doesn't want to learn programming languages; *nevertheless*, his mother is making him attend classes on programming.

*Nevertheless* is the adverbial connecting word in the sentence above. It's functioning the same way as a coordinating connecting word such as *but, so,* and *yet*. However, the difference here is that a coordinating conjunction does not require a semicolon, while an adverbial connecting word does.

### Rule 2: You Can Use Connecting Adverbials with a Single Main Clause

You can use connecting adverbials at the beginning, middle, and end of a main clause. Here are some examples:

a) Frank was put on hold by his cable company for nearly two hours. *Eventually*, he got in touch with a customer service rep.
b) Jan has never gotten the high score in GTA. She is determined, *nonetheless*, to improve her score.
c) There was something about studying for his exam on programming that made him anxious. He had no trouble studying informatics, *however*.

### Rule 3: Depending on the Sentence, You Might Not Need a Comma When Using a Connecting Adverbial

Sometimes, a break in the sentence is too weak to justify halting the reader with a comma. In fact, a comma can make a sentence sound choppy.

E.g. Harrison *certainly* didn't like it when his teacher called on him to answer a question in class. Harrison, certainly, didn't like it when his teacher called on him to answer a question in class.

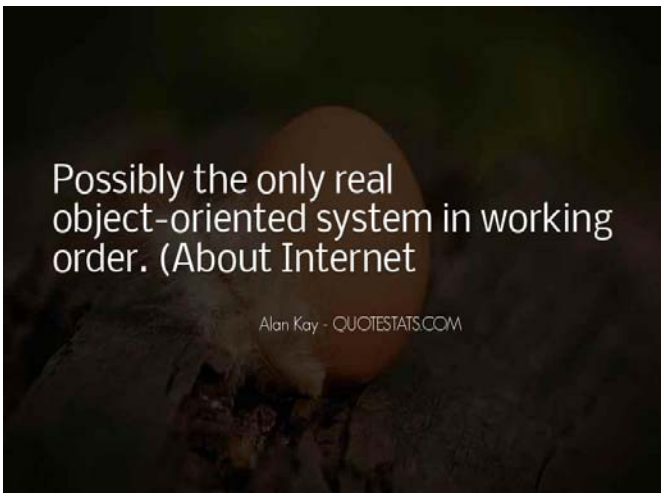
These are important skills for both academic and professional writing. Look at more conjunctive adverb examples below.

a) Paul copied his classmate's homework. *As a result*, his teacher docked his grade.
b) Stacy went to the computer store; *however*, they were out of her favorite smartphone.
c) Your work isn't bad; *in fact*, you probably deserve a raise.
d) She developed writers block; *consequently*, she didn't write another novel for years.
e) They forgot class was canceled for the week; *undoubtedly*, they had trouble figuring out how to spend their free time.

## 3. Speaking and Writing

*3.1. Here are some amazing quotes and sayings about Object Oriented you must read. Discuss them with your partner*

- Object-oriented programming offers a sustainable way to write spaghetti code. It lets you accrete programs as a series of patches. - *Author: Paul Graham*
- Mac blinked. That smile should be registered as a deadly weapon. - *Author: Kaje Harper*
- Never walk away from someone who deserves help; your hand is God's hand for that person. - *Author: Eugene H. Peterson*
- Object-oriented programming as it emerged in Simula 67 allows software structure to be based on real-world structures, and gives programmers a powerful way to simplify the design and construction of complex programs. - *Author: David Gelernter*
- Every dependency is like a little dot of glue that causes your class to stick to the things it touches. - *Author: Sandi Metz*
- I invented the term 'Object-Oriented', and I can tell you I did not have C++ in mind. - *Author: Alan Kay*




*3.2. Explain in your own words each feature in object-oriented programming*

*3.3. Suggested topics for Presentations and Reports*

1) What are the benefits of OOP?
2) Criticism of OOP. What's Wrong With Object-Oriented Programming?
3) Why is OOP so popular?
4) Object-oriented terminology

## 4. Additional reading

*4.1. Read and translate the following text*

### Document formatting languages

Document formatting languages specify the organization of printed text and graphics. They fall into several classes: text formatting notation that can serve the same functions as a word processing program, page description languages that are interpreted by a printing device and, most generally, markup languages that describe the intended function of portions of a document.

**TeX** was developed during 1977–86 as a text formatting language by Donald Knuth, a Stanford University professor, to improve the quality of mathematical notation in his books. Text formatting systems, unlike

WYSIWYG ("What You See Is What You Get") word processors, embed plain text formatting commands in a document, which are then interpreted by the language processor to produce a formatted document for display or printing. TeX marks italic text, for example, as {\it this is italicized}, which is then displayed as *this is italicized*.

TeX largely replaced earlier text formatting languages. Its powerful and flexible abilities gave an expert precise control over such things as the choice of fonts, layout of tables, mathematical notation, and the inclusion of graphics within a document. It is generally used with the aid of "macro" packages that define simple commands for common operations, such as starting a new paragraph; LaTeX is a widely used package. TeX contains numerous standard "style sheets" for different types of documents, and these may be further adapted by each user. There are also related programs such as BibTeX, which manages bibliographies and has style sheets for all of the common bibliography styles, and versions of TeX for languages with various alphabets.

**PostScript** is a page-description language developed in the early 1980s by Adobe Systems Incorporated on the basis of work at Xerox PARC (Palo Alto Research Center). Such languages describe documents in terms that can be interpreted by a personal computer to display the document on its screen or by a microprocessor in a printer or a typesetting device.

PostScript commands can, for example, precisely position text, in various fonts and sizes, draw images that are mathematically described, and specify colour or shading. PostScript uses postfix, also called reverse Polish notation, in which an operation name follows its arguments. Thus, "300 600 20 270 arc stroke" means: draw ("stroke") a 270-degree arc with radius 20 at location (300, 600). Although PostScript can be read and written by a programmer, it is normally produced by text formatting programs, word processors, or graphic display tools.

The success of PostScript is due to its specification's being in the public domain and to its being a good match for high-resolution laser printers. It has influenced the development of printing fonts, and manufacturers produce a large variety of PostScript fonts.

**SGML (standard generalized markup language)** is an international standard for the definition of markup languages; that is, it is a metalanguage. Markup consists of notations called tags that specify the function of a piece of text or how it is to be displayed. SGML emphasizes

descriptive markup, in which a tag might be "<emphasis>." Such a markup denotes the document function, and it could be interpreted as reverse video on a computer screen, underlining by a typewriter, or italics in typeset text.

SGML is used to specify DTDs (document type definitions). A DTD defines a kind of document, such as a report, by specifying what elements must appear in the document—e.g., <Title>—and giving rules for the use of document elements, such as that a paragraph may appear within a table entry but a table may not appear within a paragraph. A marked-up text may be analyzed by a parsing program to determine if it conforms to a DTD. Another program may read the markups to prepare an index or to translate the document into PostScript for printing. Yet another might generate large type or audio for readers with visual or hearing disabilities (3400).

[https://www.typescriptlang.org/docs/handbook/typescript-from-scratch.html]

*4.2. Speak about TeX, PostScript, SGML and their characteristics*

*4.3. Translate the following sentences*

1) Объектно-ориентированное программирование представляет собой путь для овладения профессией программиста. С момента изобретения компьютера методологии программирования драматически изменяются, приспосабливаясь к растущей сложности программ.
2) С ростом программ был изобретен язык Ассемблер, так что программист мог работать с большими и более сложными программами, используя символическое представление для машинных инструкций.
3) В конце концов были введены языки высокого уровня, дающие программисту больше средств для решения проблемы сложности программ. Первым широко распространенным языком был FORTRAN. Хотя FORTRAN был очень впечатляющим первым шагом, его трудно считать языком, обеспечивающим ясность и легкость понимания программ.
4) Вехами в развитии программирования являются методы, которые служат решению проблемы возрастающей сложности программ. На каждом этапе этого пути новый подход включает в себя лучшие элементы предыдущих методов.

5) Объектно-ориентированное программирование впитало в себя лучшие идеи структурного программирования и комбинирует их с новыми мощными концепциями, позволяющими увидеть задачу программирования в новом свете.
6) Объектно-ориентированное программирование позволяет легко разложить задачу на подгруппы взаимодействующих частей. Затем можно преобразовать эти подгруппы в единицы, называемые объектами.
7) Все объектно-ориентированные языки имеют три общие концепции: инкапсуляцию, полиморфизм и наследование.

# UNIT 7. Elements of Programming. Control Structures

---

*Learning objectives*

- to understand the difference between control structures and data structures
- to acquire basic knowledge about three basic control structures
- to consider an example of a subprogram and its function

*Key words and phrases. Give Russian equivalents and remember the meanings of the key words and phrases used in the text*

To express algorithms; procedural languages; sequence; iterative; to assign values; to find the roots; conditional; quadratic equation; looping; variable; to assume; discriminant; subroutine; absolute-value function; multiplication; to be implemented; repetitions; approximation; tractable subprograms; a simple restatement; a square root; paths of execution; recursive subprograms; factorial function.

*Before reading the text watch the video from https://study.com/academy/lesson/5-basic-elements-of-programming.html. Render its content in Russian*

*Read the following text and do the exercises given after it*

Despite notational differences, contemporary computer languages provide many of the same programming structures. These include basic control structures and data structures. The former provide the means to express algorithms, and the latter provide ways to organize information.

**Control structures**

Programs written in procedural languages, the most common kind, are like recipes, having lists of ingredients and step-by-step instructions for using them. The three basic control structures in virtually every procedural language are:

1. Sequence—combine the liquid ingredients, and next add the dry ones.

2. Conditional—if the tomatoes are fresh then simmer them, but if canned, skip this step.

3. Iterative—beat the egg whites until they form soft peaks.

Sequence is the default control structure; instructions are executed one after another. They might, for example, carry out a series of arithmetic operations, assigning results to variables, to find the roots of a quadratic equation $ax^2 + bx + c = 0$. The conditional IF-THEN or IF-THEN-ELSE control structure allows a program to follow alternative paths of execution. Iteration, or looping, gives computers much of their power. They can repeat a sequence of steps as often as necessary, and appropriate repetitions of quite simple steps can solve complex problems.

These control structures can be combined. A sequence may contain several loops; a loop may contain a loop nested within it, or the two branches of a conditional may each contain sequences with loops and more conditionals. In the "pseudocode" used in this article, "*" indicates multiplication and "←" is used to assign values to variables. The following programming fragment employs the IF-THEN structure for finding one root of the quadratic equation, using the quadratic formula:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

The quadratic formula assumes that $a$ is nonzero and that the discriminant (the portion within the square root sign) is not negative (in order to obtain a real number root). Conditionals check those assumptions:

```
IF a = 0 THEN
ROOT ← −c/b
ELSE
DISCRIMINANT ← b*b − 4*a*c
IF DISCRIMINANT ≥ 0 THEN
ROOT ← (−b + SQUARE_ROOT(DISCRIMINANT))/2*a
ENDIF
ENDIF
```

The SQUARE_ROOT function used in the above fragment is an example of a subprogram (also called a procedure, subroutine, or function). A subprogram is like a sauce recipe given once and used as part of many other recipes. Subprograms take inputs (the quantity needed) and produce results (the sauce). Commonly used subprograms are generally in a collection or library provided with a language. Subprograms may call other subprograms in their definitions, as shown by the following routine (where ABS is the absolute-value function). SQUARE_ROOT is implemented by using a WHILE (indefinite) loop that produces a good

approximation for the square root of real numbers unless $x$ is very small or very large. A subprogram is written by declaring its name, the type of input data, and the output:

```
FUNCTION SQUARE_ROOT(REAL x) RETURNS REAL
ROOT ← 1.0
WHILE ABS(ROOT*ROOT − x) ≥ 0.000001
AND WHILE ROOT ← (x/ROOT + ROOT)/2
RETURN ROOT
```

Subprograms can break a problem into smaller, more tractable subproblems. Sometimes a problem may be solved by reducing it to a subproblem that is a smaller version of the original. In that case the routine is known as a recursive subprogram because it solves the problem by repeatedly calling itself. For example, the factorial function in mathematics ($n! = n\cdot(n-1)\cdots3\cdot2\cdot1$—i.e., the product of the first $n$ integers), can be programmed as a recursive routine:

```
FUNCTION FACTORIAL(INTEGER n) RETURNS INTEGER
IF n = 0 THEN RETURN 1
ELSE RETURN n * FACTORIAL(n−1)
```

The advantage of recursion is that it is often a simple restatement of a precise definition, one that avoids the bookkeeping details of an iterative solution.

At the machine-language level, loops and conditionals are implemented with branch instructions that say “jump to” a new point in the program. The “goto” statement in higher-level languages expresses the same operation but is rarely used because it makes it difficult for humans to follow the “flow” of a program. Some languages, such as Java and Ada, do not allow it.

## 1. Text-based Assignments

*1.1. Give English equivalents of the following words and word combinations:*

Различия в написании; современный; управляющая структура; пооперационная инструкция; последовательность; управляющая структура по умолчанию; выполнять арифметические операции; находить корень; позволять; соответствующие повторения; решать проблемы; содержать; оператор цикла с условием; допущения;

алгоритм вычислений; функция абсолютного значения; рекурсивная подпрограмма; точное определение.

*1.2. Find synonyms for the following words from the text*

1. Contemporary, 2. control, 3. structure, 4. to assign, 5. to declare, 6. root, 7. precise, 8. to follow, 9. routine, 10. to repeat.

1) command, 2) present (new, late), 3) to iterate, 4) manner of making, 5) to give (value), 6) to proclaim, 6) procedure (operation), 7) stem, 8) well-defined, 9) to practice, 10) order.

*1.3. Match the following words with their definitions*

| | |
|---|---|
| 1. Conditionals g | a) an operation that requires successive executions of instructions or processes. |
| 2. Iterative operation b | b) a set of instructions in a program that perform a task. Programs are made up of many routines, which are also called "subroutines" and very often "functions." |
| 3. Loop d | c) to be represented by a figure, symbol, or formula. |
| 4. While loop f | d) the action of doing something over and over again. |
| 5. Routine a | e) a mathematical statement saying that two amounts or values are the same, for example 6x4=12x2. |
| 6. Equation e | f) a loop that continues to repeat while a condition is true. |
| 7. Express c | g) statements that only run under certain conditions. |

*1.4. Answer the following questions on the text*

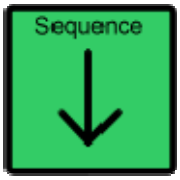
1) How many basic control structures are there in virtually every procedural language?
2) What may a sequence contain?
3) How is SQUARE_ROOT implemented?
4) What is the main advantage of recursion?
5) Which type of programming structure requires each instruction to be performed in order, with no possibility of skipping an action or branching off to another action?

6) The if/else statement conditionally evaluates two statements. Is it correct?
7) Which control structure is used for repeated operations?
8) Which control structure is used to group statements that provide a single logical operation together?

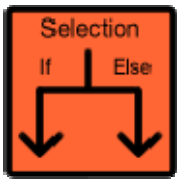
*1.5. Use the correct word to fill in the gaps in the following sentences*

| | |
|---|---|
| a single logical operation | a series of statements |
| variables | a test |
| default | to alter |
| a subprogram | **`While`** |

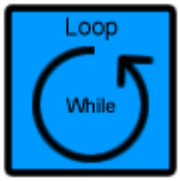
## Program Flow of Control




Sequence is composed of a) _____ which are executed one by one from top to bottom. Sequence is the b) _____ flow of control for many programming languages. All of the programs illustrated so far have used this flow of control for their execution.




Selection is used c) _____ the flow of control when a choice needs to be made between two or more actions. Often the choice is based on the state of some d) _____ in the program. This control structure is commonly specified using the keywords **`If`** and **`Else`**.




Loop is a control structure that causes a set of statements to be executed repeatedly. With each loop iteration, e) _____ is performed to determine whether the loop should continue or end. Often this control structure is specified using the key word f) _____ .


Subprograms are a way of grouping statements that provide g) _____ . An example subprogram might be SquareRoot which could find the square root of a number and return the result to the main program. The keyword **`Call`** indicates h) _____.

*Key answers to task 1.5.*

a) a series of statements
b) default
c) to alter
d) variables
e) to test
f) while
g) a single logical operation
h) a subprogram

## 2. Focus on Grammar

*2.1. Study the table of Functions of the Infinitive and make up your own sentences*

| | | |
|---|---|---|
| **Subject** | To break a problem into smaller, more tractable subprograms is not difficult. | Разбить проблему на более мелкие, более понятные подпрограммы несложно. |
| **Adverbial modifier** of purpose (can be introduced by *in order* and *so as*) | We can also use the computer's memory to store other types of data such as letters and characters like 'a', '?', or 'Z'. | Мы также можем использовать память компьютера для хранения других типов данных, таких как буквы и символы, такие как «a», «?» или «Z». |
| **Adverbial modifier** of result (it chiefly occurs after adjectives modified by the adverbs *enough* and *too*, and after the conjunction *as*) | The finds are too few to be spoken about.<br><br>The rule has been so formulated as to be easily observed by everybody. | Находок слишком мало, чтобы о них (можно было) говорить.<br><br>Правило было сформулировано таким образом, чтобы все могли легко его соблюдать. |
| **Predicative** | A typical way of specifying the type of a variable is to write the type name before the variable identifier. | Типичным способом указания типа переменной является запись имени типа перед идентификатором переменной. |
| **Attribute** | This is the main advantage to be taken into consideration. | Это основное преимущество, которое нужно учитывать. |
| **Object** | The conditional IF-THEN or IF-THEN-ELSE control structure allows a program to follow alternative paths of execution. | Условная управляющая структура IF-THEN или IF-THEN-ELSE позволяет программе следовать альтернативным способам выполнения. |

*2.2. Comment on the form and functions of the Infinitives and translate the following sentences*

1) Input is one of the two elements that are used by every program because every program needs some data to work with.
2) The bank wants to make sure it isn't someone who's not you trying to access your account.
3) Computers can perform all kinds of mathematical operations and functions, from the simple addition or subtraction needed to update your checking account balance after a withdrawal or deposit, to the complex calculus needed to put a satellite into orbit.
4) To use these elements, one imports them.
5) We can use assignment statements to give new names to existing functions.
6) One of our goals in this chapter is to isolate issues about thinking procedurally.
7) My aim was to create a program that could interact with humans like the modern-day chatbots.
8) To display output on the terminal the ‘echo’ command is used followed by the text to display.

## 3. Discussion

*3.1. Discuss the following questions with your partner*

1) How do programs work, and how can you build them?
2) How many important control structures do algorithms require? What are they?

*3.2. Suggested topics for Presentations and Reports*

1) The role of SemiColon in various Programming Languages
2) Structures in C++
3) A two-way selection and a multi-way selection control structures.

## 4. Additional Reading

*4.1. Read and translate the following text about some declarative languages*

### Declarative languages

Declarative languages, also called nonprocedural or very high level, are programming languages in which (ideally) a program specifies what is to be done rather than how to do it. In such languages there is less difference between the specification of a program and its implementation than in the procedural languages described so far. The two common kinds of declarative languages are logic and functional languages.

***Logic programming languages***, of which PROLOG (*pro*gramming in *log*ic) is the best known, state a program as a set of logical relations (e.g., a grandparent is the parent of a parent of someone). Such languages are similar to the SQL database language. A program is executed by an "inference engine" that answers a query by searching these relations systematically to make inferences that will answer a query. PROLOG has been used extensively in natural language processing and other AI programs.

***Functional languages*** have a mathematical style. A functional program is constructed by applying functions to arguments. Functional languages, such as LISP, ML, and Haskell, are used as research tools in language development, in automated mathematical theorem provers, and in some commercial projects.

***Scripting languages*** are sometimes called little languages. They are intended to solve relatively small programming problems that do not require the overhead of data declarations and other features needed to make large programs manageable. Scripting languages are used for writing operating system utilities, for special-purpose file-manipulation programs, and, because they are easy to learn, sometimes for considerably larger programs.

Perl was developed in the late 1980s, originally for use with the UNIX operating system. It was intended to have all the capabilities of earlier scripting languages. Perl provided many ways to state common operations and thereby allowed a programmer to adopt any convenient style. In the 1990s it became popular as a system-programming tool, both for small utility programs and for prototypes of larger ones. Together with

other languages discussed below, it also became popular for programming computer Web “servers.” (2200)

[https://www.britannica.com/technology/computer-programming-language/Visual-Basic]

*4.2. Describe some declarative languages in class*

# UNIT 8. Elements of Programming. Data Structures

---

*Learning objectives*

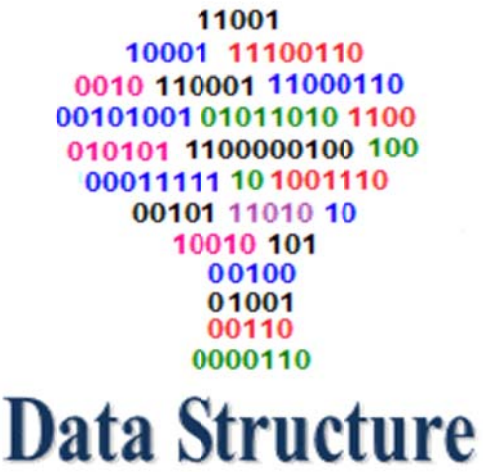
- to understand basic and advanced concepts of data structure
- to consider different types of data structures

*Key words and phrases. Give Russian equivalents and remember the meanings of the key words and phrases used in the text*

Types of data; to specify; to keep track; integers; real numbers; character strings; the array; a collection of vectors; record components or fields; to sum; dynamic allocation; a bintree; abstract data types; to hide; appropriate; to omit; information hiding; reusability; abstraction; to make public; lookup operation; insertion operation.

*Before reading the text watch the video from https://www.youtube.com/watch?v=DuDz6B4cqVc&ab_channel=CrashCourse. Render its content in Russian and explain how you understand the concept data structure*

*Read the following text and do the exercises given after it*




Whereas control structures organize algorithms, data structures organize information. In particular, data structures specify types of data, and thus which operations can be performed on them, while eliminating the need for a programmer to keep track of memory addresses. Simple data structures include integers, real numbers, Booleans (true/false), and characters or character strings. Compound data structures are formed by combining one or more data types.

The most important compound data structures are the array, a homogeneous collection of data, and the record, a heterogeneous collection. An array may represent a vector of numbers, a list of strings, or a collection of vectors (an array of arrays, or mathematical matrix). A record might store employee information—name, title, and salary. An array of records, such as a table of employees, is a collection of elements, each of which is heterogeneous. Conversely, a record might contain a vector—i.e., an array.

Record components, or fields, are selected by name; for example, E.SALARY might represent the salary field of record E. An array element is selected by its position or index; *A*[10] is the element at position 10 in array *A*. A FOR loop (definite iteration) can thus run through an array with index limits (FIRST TO LAST in the following example) in order to sum its elements:

```
FOR i ← FIRST TO LAST
SUM ← SUM + A[i]
```

Arrays and records have fixed sizes. Structures that can grow are built with dynamic allocation, which provides new storage as required. These data structures have components, each containing data and references to further components (in machine terms, their addresses). Such self-referential structures have recursive definitions. A bintree (binary tree) for example, either is empty or contains a root component with data and left and right bintree "children." Such bintrees implement tables of information efficiently. Subroutines to operate on them are naturally recursive; the following routine prints out all the elements of a bintree (each is the root of some subtree):

```
PROCEDURE TRAVERSE(ROOT: BINTREE)
IF NOT(EMPTY(ROOT))
TRAVERSE(ROOT.LEFT)
PRINT ROOT.DATA
TRAVERSE(ROOT.RIGHT)
ENDIF
```

Abstract data types (ADTs) are important for large-scale programming. They package data structures and operations on them, hiding internal details. For example, an ADT table provides insertion and lookup operations to users while keeping the underlying structure, whether an array, list, or binary tree, invisible. In object-oriented languages, classes are ADTs and objects are instances of them. The following object-oriented

pseudocode example assumes that there is an ADT bintree and a "superclass" COMPARABLE, characterizing data for which there is a comparison operation (such as "<" for integers). It defines a new ADT, TABLE, that hides its data-representation and provides operations appropriate to tables. This class is polymorphic—defined in terms of an element-type parameter of the COMPARABLE class. Any instance of it must specify that type, here a class with employee data (the COMPARABLE declaration means that PERS_REC must provide a comparison operation to sort records). Implementation details are omitted.

```
CLASS TABLE OF <COMPARABLE T>
PRIVATE DATA: BINTREE OF <T>
PUBLIC INSERT(ITEM: T)
PUBLIC LOOKUP(ITEM: T) RETURNS BOOLEAN
END

CLASS PERS_REC: COMPARABLE
PRIVATE NAME: STRING
PRIVATE POSITION: {STAFF, SUPERVISOR, MANAGER}
PRIVATE SALARY: REAL
PUBLIC COMPARE (R: PERS_REC) RETURNS BOOLEAN
END

EMPLOYEES: TABLE <PERS_REC>
```

TABLE makes public only its own operations; thus, if it is modified to use an array or list rather than a bintree, programs that use it cannot detect the change. This information hiding is essential to managing complexity in large programs. It divides them into small parts, with "contracts" between the parts; here the TABLE class contracts to provide lookup and insertion operations, and its users contract to use only the operations so publicized.

## Advantages of Data structures

The following are the advantages of a data structure:

- **Efficiency:** If the choice of a data structure for implementing a particular ADT is proper, it makes the program very efficient in terms of time and space.
- **Reusability:** The data structure provides reusability means that

multiple client programs can use the data structure.

- **Abstraction:** The data structure specified by an ADT also provides the level of abstraction. The client cannot see the internal working of the data structure, so it does not have to worry about the implementation part. The client can only see the interface.

[https://www.javatpoint.com/data-structure-tutorial]

## 1. Text-based Assignments

*1.1. Give English equivalents of the following words and word combinations:*

Сконструировать алгоритмы; в частности; целые числа; булевские значения; составные структуры данных; собрание однородных данных; запись; наоборот; представлять; массивы; индекс; определенная итерация; динамическое распределение памяти; двоичное (бинарное) дерево; примеры; представление данных; обнаруживать изменения; последовательный поиск; возможность повторного использования; абстрактный тип данных *(уникальный тип данных, определённый в терминах, применяемых к объектам операций (т.е. набора функций доступа).*

*1.2. Match the following words with their definitions:*

| | |
|---|---|
| 1. Tree topology<br>c | a) a group of related data values (called elements) that are grouped together. They must be the same data type. |
| 2. Pseudocode<br>e | b) either a pointer or an array with only one dimension. In computer graphics, the term describes a line with a starting and ending point. |
| 3. Array<br>a | c) a special type of structure where many connected elements are arranged like the branches of a tree. For example, they are frequently used to organize the computers in a corporate network, or the information in a database. |
| 4. Field<br>f | d) a function or method that repeatedly calculates a smaller part of itself to arrive at the final result. It is similar to iteration. |

| | |
|---|---|
| 5. Data structure g | e) a computer programming language that resembles plain English that cannot be compiled or executed, but explains a resolution to a problem. |
| 6. Recursive b | f) a single item of data contained in a column within a database or software program. For example, it may be a customer name, address, or phone number. |
| 7. Vector d | g) a predefined format for efficiently storing, accessing, and processing data in a computer program. |

*1.3. Answer the following questions on the text*

1) What do data structures do in terms of computer programming?
2) What do simple data structures include?
3) How is an array element selected?
4) What do an ADT table provide?
5) What is a tree data structure?
6) Which of the following data structure (arrays, records, pointers) can't store the non-homogeneous data elements?

*1.4. Read the text again and decide if the following statements are true or false.*

1) Data structures specify types of data, and thus which operations can be performed on them, keeping the need for a programmer to keep track of memory addresses.
2) Compound data structures are formed by combining one or more data types.
3) The most important compound data structures are the array, a homogeneous collection of data, and the record, a heterogeneous collection.
4) An array of records, such as a table of employees, is a collection of elements, each of which is homogeneous.
5) An array element is selected by its position or index; *A*[10] is the element at position 10 in array *A*.
6) Arrays and records do not have fixed sizes. Structures that can grow are built with dynamic allocation, which provides new storage as required.

7) Abstract data types are important for large-scale programming. They package data structures and operations on them, hiding internal details.
8) For example, an ADT table provides insertion and lookup operations to users while keeping the underlying structure, whether an array, list, or binary tree, invisible.
9) The data structure specified by an ADT cannot provide the necessary level of abstraction. The client cannot see the internal working of the data structure.
10) The client can only see the interface.

*Key answers to the task 1.2.*
1.c); 2.e); 3.a); 4.f); 5.g); 6.d); 7.b).

## 2. Focus on Grammar

*2.1. Complete the sentences using the bare infinitive or to-infinitive of the verbs in brackets*

1) She saw me (turn off) the computer.
2) Our boss usually encourages all the staff (take) the refresher courses.
3) We made her (check) all the receipts once again.
4) They didn't dare (interrupt) my presentation.
5) He decided (continue) his programming classes.
6) What makes you (think) so?
7) He advised me (show) my design project to the manager.
8) They ordered me (leave) the office immediately.
9) We consider this company (be) the most reliable partner.
10) He helped me (analyze) all the given information.
11) I heard the negotiations (stop).
12) My parents didn't let me (stay) there late.
13) We told our secretary (check) the mail in the evening.
14) I heard them (speak) in a loud voice.
15) Some scientists consider Mars (be covered) with vegetation.
16) Another possibility was (use) quartz.

## Infinitive Constructions

**Complex Object** ( The Objective with the infinitive construction)

| *Subject Predicate* | *Noun in the common case* / *Pronoun in the objective case* | *+ infinitive* |
|---|---|---|
| He /believed | the results /of this test | to have been plotted/ in the diagram. |

E.g. *The circumstances forced him to leave the town. – Обстоятельства заставили (вынудили) его уехать из города.*

NOTE

After the verbs *to see, to notice* when they denote sense perception the infinitive of the verb *"to be"* is not used. Instead a subordinate clause is used. E.g. *We saw that he was in. – Я увидел, что он дома.*

**Complex Subject** (The Subjective Infinitive Construction)

| *Noun in the common case or pronoun in the nominative case* | *A finite verb* | *Infinitive* | |
|---|---|---|---|
| The painter | | | |
| He | seemed | to see | nothing. |
| The treaty | is said | to have been signed | yesterday. |

**The for-to-infinitive construction**

| | *preposition* | *noun/ pronoun* | *infinitive* |
|---|---|---|---|
| They waited | for | the door | to open. |
| It is useless | for | me | to speak to him. |

*E.g. It was easy for me to answer that question. — Мне легко было ответить на этот вопрос.*

*2.2. Sort out the sentences given below into corresponding columns. The first two are done for you. Translate them*

| | |
|---|---|
| Complex Object | 1, |
| Complex Subject. | 2, |

1) The return makes one love the farewell.
2) His policy is no policy. And this in itself is supposed to be a policy.
3) Don't let what you cannot do interfere with what you can do.
4) The traditional toast 'Bottoms up' is known to be absolutely taboo in the Navy.
5) The last drop makes the cup run over.
6) Ambassadors are said to be eyes and ears of states.
7) Diplomacy is thought to be the art of jumping into troubled waters without making a splash.
8) Experience is said to be a comb which nature gives us when we are bold.
9) It often happens that things turn out to be different from what they at first appear to be.
10) You can take the horse to the water but you can't make him drink it.
11) The decision is sure to be adopted tomorrow and we might get acquainted with it.
12) It is true, however, as Wilkinson pointed out, that fast transitions are more likely to have been observed than slow ones.
13) The engineer wants the workers to use soft rubber for electrical insulation.

## 3. Discussion

*3.1. Give a short explanation of the terms 'Control Structure' and 'Data Structure'*

*3.2 Discuss the following questions in class. You can use some additional Internet resources if necessary*

1) Which language is best for Data Structure and algorithms? Most competitive programmers use C++. Can you explain why it is so?
2) How do you start learning DSA?
3) Is learning data structure and algorithms hard?
4) What are Python data structures?
5) How many days does it take to learn data structure and algorithms?

*3.3. Make a report on the future prospects of using various programming languages*

| S.No. | Languages | Future Scope |
|---|---|---|
| 1. | Python | Python, without a doubt, has a bright future in the programming language development area, particularly in the disciplines of data visualisation, artificial intelligence, data science and machine learning. |
| 2. | Java | Java is widely utilised in many businesses. It may also be used to make a variety of goods and has a wide range of uses. It is currently the most widely used programming language, so it's pretty worth learning. |
| 3. | C++ | C++ has a wide range of applications, and studying it is never a bad thing. It is a very simple language to pick up and understand. In the industry, it has a wide range of applications. Along with graphic designs and 3-D models, it's also employed in games. |
| 4. | C | Although C is out of date in some applications, it is not going away anytime soon. It has a wide range of real-world applications, and it will continue to be used in the industry for many years to come. |
| 5. | C# | C# is a language that is gaining in popularity and is likely to remain so in the coming years due to its effective capabilities in producing games and its resilience, both of which benefit the gaming industry. It's also quite beneficial in business applications. |
| 6. | Javascript | JavaScript is a widely-used programming language. It is so extensively used that another programming language may take a long time to replace it. It is also used in artificial intelligence and other fields, in addition to web development. |

| | | |
|---|---|---|
| | | This language should be at the top of anyone's learning priority list. |
| 7. | Ruby | In today's world, Ruby is still utilised for a large number of applications. As a result, it's a great language to learn because you'll be able to create complex apps in no time. It also has robust technology. Therefore it is still relevant today. |

## 4. Additional Reading

*4.1. Read and translate the following text*

**C#** (/si ʃɑːrp/ *see sharp*) is a general-purpose, multi-paradigm programming language. C# encompasses static typing, strong typing, lexically coped, imperative, declarative, functional, generic, object-oriented (class-based), and component-oriented programming disciplines. During the development of the .NET Framework, the class libraries were originally written using a managed code compiler system called *"Simple Managed C"* (SMC).

In January 1999, Anders Hejlsberg formed a team to build a new language at the time called Cool, which stood for "C-like Object Oriented Language". Microsoft had considered keeping the name "Cool" as the final name of the language, but chose not to do so for trademark reasons. By the time the .NET project was publicly announced at the July 2000 Professional Developers Conference, the language had been renamed C#, and the class libraries and ASP.NET runtime had been ported to C#.

Hejlsberg is C#'s principal designer and lead architect at Microsoft, and was previously involved with the design of Turbo Pascal, Embarcadero Delphi (formerly CodeGear Delphi, Inprise Delphi and Borland Delphi), and Visual J++. In interviews and technical papers he has stated that flaws in most major programming languages (e.g. C++, Java, Delphi, and Smalltalk) drove the fundamentals of the Common Language Runtime (CLR), which, in turn, drove the design of the C# language itself.

James Gosling, who created the Java programming language in 1994, and Bill Joy, a co-founder of Sun Microsystems, the originator of Java, called C# an "imitation" of Java; Gosling further said that "[C# is] sort of

Java with reliability, productivity and security deleted." Klaus Kreft and Angelika Langer (authors of a C++ streams book) stated in a blog post that "Java and C# are almost identical programming languages. Boring repetition that lacks innovation," "Hardly anybody will claim that Java or C# are revolutionary programming languages that changed the way we write programs," and "C# borrowed a lot from Java - and vice versa. Now that C# supports boxing and unboxing, we'll have a very similar feature in Java." In July 2000, Hejlsberg said that C# is "not a Java clone" and is "much closer to C++" in its design.

Since the release of C# 2.0 in November 2005, the C# and Java languages have evolved on increasingly divergent trajectories, becoming two quite different languages. One of the first major departures came with the addition of generics to both languages, with vastly different implementations. C# makes use of reification to provide "first-class" generic objects that can be used like any other class, with code generation performed at class-load time.

Furthermore, C# has added several major features to accommodate functional-style programming, culminating in the LINQ extensions released with C# 3.0 and its supporting framework of lambda expressions, extension methods, and anonymous types. These features enable C# programmers to use functional programming techniques, such as closures, when it is advantageous to their application. The LINQ extensions and the functional imports help developers reduce the amount of boilerplate code that is included in common tasks like querying a database, parsing an xml file, or searching through a data structure, shifting the emphasis onto the actual program logic to help improve readability and maintainability (2800)

[ https://en.wikipedia.org/wiki/C_Sharp_(programming_language)]

*4.2. Ask an appropriate question for the response*

1) **C#** is a general-purpose, multi-paradigm programming language.
2) Hejlsberg is C#'s principal designer and lead architect at Microsoft.
3) In January 1999.
4) These features enable C# programmers to use functional programming techniques, such as closures, when it is advantageous to their application.
5) The LINQ extensions and the functional imports.

*4.3. Translate the following sentences*

1) Массив – структурированный тип данных, состоящий из фиксированного числа однотипных элементов, объединённых одним именем, где каждый элемент имеет свой номер (индекс).
2) Как мы уже отмечали ранее, алгоритмам требуются две важные управляющие структуры: для итераций и для выбора.
3) Обе они поддерживаются в Python в различных формах. Программисты могут выбирать тот способ, который будет более уместным в данных обстоятельствах.
4) Для итераций Python предлагает стандартный оператор while и очень мощный оператор for.
5) Операторы выбора позволяют программистам задавать вопросы и выполнять различные действия, основываясь на ответе.
6) Большинство языков программирования предоставляют две версии полезных конструкций: ifelse и if. Простой пример бинарного использования оператора ifelse:

```
if n<0:

   print("Sorry, value is negative")

else:

   print(math.sqrt(n))
```

7) Поскольку большинство синтаксических конструкций Perl основаны на языке C, то для программистов, знающих языки C, C++, C#, Java, JavaScript, Python или PHP, синтаксис Perl будет очень знакомым.
8) В некоторых случаях (например, при записи атрибутов файла в Unix) нагляднее изобразить числа в восьмеричной системе счисления.
9) Весьма удобно, что преобразования между строками и числами выполняются автоматически в зависимости от контекста выражения, в котором они используются.
10) В языке Perl для уточнения смысла языковых конструкций часто используется понятие контекста, под которым понимается программное окружение элемента языка (переменной, подпрограммы и так далее), определяющее его использование.

# UNIT 9. Web – Development. Types of Web – development

---

*Learning objectives*

- to acquire basic knowledge about web development and its types
- to consider the difference between Web development and web design

*Key words and phrases. Give Russian equivalents and remember the meanings of the key words and phrases used in the text*

Intranet; functionality; content management systems (CMS); plain text; Web content development; client liaison; network security configuration; Agile methodologies; front-end developer; back-end developer; full-stack developer; database technology; layout; fonts; to run smoothly; drop-down menu; scrollbars; checkout function; User Experience Design; User Interface Design; visual design; to be broken down into; relational database management system (RDBMS).

*Read the following text and do the exercises given after it*

Web development is the process of building websites and applications for the internet, or for a private network known as an intranet. Web development is not concerned with the design of a website; rather, it's all about the coding and programming that powers the website's functionality.

From the most simple, static web pages to social media platforms and apps, from e-commerce websites to content management systems

(CMS) - all the tools we use via the internet on a daily basis have been built by developers.

Web development can range from developing a simple single static page of plain text to complex web applications, electronic businesses, and social network services. A more comprehensive list of tasks to which Web development commonly refers, may include Web engineering, Web design, Web content development, client liaison, client-side/server-side scripting, Web server and network security configuration, and e-commerce development.

For larger organizations and businesses, Web development teams can consist of hundreds of people (Web developers) and follow standard methods like Agile methodologies while developing Web sites. Web development may be a collaborative effort between departments rather than the domain of a designated department.

There are three kinds of Web developer specialization: front-end developer, back-end developer, and full-stack developer. *Front-end developers* are responsible for behavior and visuals that run in the user browser, while back-end developers deal with the servers.

## Types of web development

Web development can be broken down into three layers: client-side coding (frontend), server-side coding (backend) and database technology.

*Client-side*

Client-side scripting, or frontend development, refers to everything that the end user experiences directly. Client-side code executes in a web browser and directly relates to what people see when they visit a website. Things like layout, fonts, colours, menus and contact forms are all driven by the frontend.

*Server-side*

Server-side scripting, or backend development, is all about what goes on behind the scenes. The backend is essentially the part of a website that the user doesn't actually see. It is responsible for storing and organizing data, and ensuring that everything on the client-side runs smoothly. It does this by communicating with the front-end. Whenever something happens on the client-side - say, a user fills out a form - the browser sends a request to the server-side. The server-side "responds" with relevant information in the form of frontend code that the browser can then interpret and display.

*Database technology*

Websites also rely on database technology. The database contains all the files and content that are necessary for a website to function, storing it in such a way that makes it easy to retrieve, organize, edit, and save. The database runs on a server, and most websites typically use some form of relational database management system (RDBMS).

To summarize: the frontend, backend, and database technology all work together to build and run a fully functional website or application, and these three layers form the foundation of web development.

| FRONTEND DEVELOPERS | BACKEND DEVELOPERS | FULL STACK DEVELOPERS |
|---|---|---|
| ▪ Code the fronted of a website; i.e. the part that the user sees and interacts with.<br>▪ Bring the web designer's designs to life using HTML, JavaScript and CSS.<br>▪ Ensure responsive design. | ▪ Work behind-the-scenes, building and maintaining the technology needed to power the frontend.<br>▪ Ensure that everything the frontend developer builds is fully functional.<br>▪ Create and manage the database. | ▪ Experts in both frontend and backend development.<br>▪ Guide on strategy and best practices.<br>▪ Well –versed in both business logic and user experience. |

Fig. 10. What does a web developer do?

## The difference between web development and web design

Just like with software engineering, you might also hear the terms "web development" and "web design" used interchangeably, but these are two very different things.

Imagine a web designer and web developer working together to build a car: the developer would take care of all the functional components, like the engine, the wheels and the gears, while the designer would be responsible for both the visual aspects - how the car looks, the layout of the dashboard, the design of the seats - and for the user experience provided by the car, so whether or not it's a smooth drive.

Web designers design how the website looks and feels. They model the layout of the website, making sure it's logical, user-friendly and pleasant to use. They consider all the different visual elements: what color schemes and fonts will be used? What buttons, drop-down menus and scrollbars should be included, and where? Web design also considers the information architecture of the website, establishing what content will be included and where it should be placed.

Web design is an extremely broad field, and will often be broken down into more specific roles such as User Experience Design, User Interface Design, and Information Architecture.

It is the web developer's job to take this design and develop it into a live, fully functional website. A frontend developer takes the visual design as provided by the web designer and builds it using coding languages such as HTML, CSS and JavaScript. A backend developer builds the more advanced functionality of the site, such as the checkout function on an e-commerce site.

In short, a web designer is the architect, while the web developer is the builder or engineer.

[https://en.wikipedia.org/wiki/Web_development]
[https://careerfoundry.com/en/blog/web-development/]

## 1. Text-based Assignments

*1.1. Give English equivalents of the following words and word combinations:*

Создавать вебсайт, статическая веб-страница *(страница, которая создана заранее и хранится для последующей отправки клиентам),* инструмент, сложные веб приложения, перечень задач, исполнение скриптов на сервере, поддержание связей с клиентами, совместные усилия, ответственный отдел, разработчик пользовательских интерфейсов, разработчик полного цикла, шрифтовой комплект, посылать запрос, извлекать, взаимозаменяемо, раскрывать падающее меню, выполняться на сервере, полагаться на что-л., код клиентской стороны.

*1.2. Match the following synonyms from the text*

| | |
|---|---|
| 1. to save h | a) of the sight or vision |
| 2. complex d | b) outline |
| 3. to execute j | c) to prepare, to supply |
| 4. design i | d) complicated, composed of several elements |
| 5. foundation f | e) constituent |
| 6. to provide c | f) basis |
| 7. to designate g | g) to appoint, to assign |
| 8. component e | h) to keep, to preserve |
| 9. visual a | i) project, scheme |
| 10. configuration b | j) to accomplish |

*1.3. Match the following words and phrases from the text with their meanings*

| | |
|---|---|
| 1. Backend b | a) a collection of interlinked web pages on the World Wide Web |
| 2. domain c | b) all of the behind-the-scenes digital operations that it takes to keep the front end of a website running, such as the coding, style, and plugins |
| 3. Web designer f | c) the address for a website as entered into the browser |
| 4. website a | d) the part of the website or app that the user sees. If the back end of your website is everything behind-the-scenes, this is what happens onstage |
| 5. frontend d | e) system software for creating and managing databases that makes it possible for end users to create, protect, read, update and delete data in a database |
| 6. database management system e | f) an IT professional who is responsible for designing the layout, visual appearance and the usability of a website |

*1.4. Read the text again and decide if the following statements are true or false.*

1) Web development is the process of building websites and applications for a private network. F
2) Web development can range from developing a simple single static page of plain text to complex web applications, electronic businesses, and social network services. T
3) There are two kinds of Web developer specialization: front-end developer and back-end developer. T
4) The backend is not very actual because the user doesn't actually see it. F
5) Web design is an extremely broad field. T
6) Backend developers ensure that everything the frontend developer builds is fully functional. T
7) A full-stack developer builds the more advanced functionality of the site, such as the checkout function on an e-commerce site. F(T?)
8) Sometimes Websites do not rely on database technology. F

*1.5. Answer the following questions on the text*

1) What does a full-stack developer do?
2) What is a backend developer responsible for?
3) What does a frontend developer do?
4) Can you name any types of web development?
5) What is the difference between web development and web design?
6) How many people can Web development team consist of?
7) What tasks does Web development include?

## 2. Focus on Grammar

*2.1. Study the table of Participle I and Participle II*

<table>
<tr><td colspan="2">Вид</td><td colspan="2">(Active)</td><td colspan="2">(Passive)</td></tr>
<tr><td rowspan="3"></td><td>Present Participle<br>Simple</td><td colspan="2">developing -разрабатывающий; разрабатывая (вообще) While developing his first website he can make some mistakes.</td><td colspan="2">being developed - разрабатываемый; будучи разработан (вообще)</td></tr>
<tr><td rowspan="2">Present Participle<br>Perfect</td><td>having</td><td>III</td><td>having been</td><td>III</td></tr>
<tr><td colspan="2">having developed - разработав, (уже, до чего-то)<br>Having developed the program our company can offer its maintenance.</td><td colspan="2">having been developed – (уже) был разработан</td></tr>
<tr><td colspan="2" rowspan="2">Participle II<br>(Past Participle)</td><td colspan="2" rowspan="2">--------</td><td>-</td><td>III</td></tr>
<tr><td colspan="2" rowspan="2">developed – разработанный</td></tr>
<tr><td></td><td colspan="2"></td></tr>
</table>

*2.2. Read and translate the sentences. Comment on the functions of Participle I, II*

1) Software engineering is the discipline that aims to provide methods and procedures for developing software systems.
2) Extensive simulation and prototyping are sometimes used to capture and analyze the system requirements concerned with human interaction.
3) Researchers in the Cockrell School of Engineering at The University of Texas have developed a new, open-source computer programming framework that could make the web significantly more energy efficient, allowing people to save more battery power while browsing on mobile devices.
4) Having heard the gift of the report, Mr Smith did not dispute it.
5) Turning off his computer, he went out to the terrace.
6) He entered, puzzled but interested.
7) At last she heard her name called.
8) Mobile device users spend nearly two-thirds of their time browsing the web.
9) And each answer made was written down quickly upon the sheets of paper.

*2.3. Choose the right variant*

1) Have you had this article _____?
   a) typed b) typing c) type
2) We shall not be able to catch the train_____ at seven?
   a) left b)leaving c) having been left
3) The letter _____ yesterday was not welcome.
   a) receiving b) having received c) received
4) _____ his report, the clerk started writing down the latest figures.
   a) finished b) being finished c) having finished
5) I won't be able to go anywhere tomorrow as I'll have my new furniture _____.
   a) delivering b) being delivered c) delivered
6) _____ articles for her course paper, she began ____ money as a journalist while she was attending college.
   a) writing, earning; b) having written, earn; c) having written, to earn
7) She turned to me for help, _____ how to deal with the problem.
   a) not having known b) not being known c) not knowing
8) Alice didn't like her computer classes; she thought they were _____ .
   a) bored b) being boring c) boring

## 3. Discussion

*3.1. Look at some websites. Make notes on the differences in design between them. Make a report about navigation bars, the categories and animations they use. What design features can you notice?*

*3.2. Discuss the following questions:*

1) Why do people have personal websites?
2) Have you ever visited anyone's personal home page? What was it like? Why do companies have websites?
3) What is a difference between a website and a webpage?

## 4. Additional reading

*4.1. Read and translate the text about other development tools*

### Other web development tools

Web development tools (often called devtools or inspect element) allow web developers to test and debug their code. They are different from website builders and integrated development environments (IDEs) in that they do not assist in the direct creation of a webpage, rather they are tools used for testing the user interface of a website or web application.

Web development tools come as browser add-ons or built-in features in web browsers. Most popular web browsers, such as Google Chrome, Firefox, Internet Explorer, Safari, Microsoft Edge and Opera, have built-in tools to help web developers, and many additional add-ons can be found in their respective plugin download centers.

Web development tools allow developers to work with a variety of web technologies, including HTML, CSS, the DOM, JavaScript, and other components that are handled by the web browser. Due to increasing demand from web browsers to do more, popular web browsers have included more features geared for developers.

Web developers will also use a text editor, such as Atom, Sublime or Visual Studio Code, to write their code; a web browser, such as Chrome or Firefox; and an extremely crucial tool: Git!

Git is a version control system where developers can store and manage their code. As a web developer, it's inevitable that you'll make constant changes to your code, so a tool like Git that enables you to track these changes and reverse them if necessary is extremely valuable. Git also makes it easier to work with other teams and to manage multiple projects at once. Git has become such a staple in the world of web development that it's now considered really bad practice not to use it.

Another extremely popular tool is GitHub, a cloud interface for Git. While we explain more about what it is and how to use it in our GitHub guide, essentially this tool offers all the version control functionality of Git, but also comes with its own features such as bug tracking, task management and project wikis.

GitHub not only hosts repositories; it also provides developers with a comprehensive toolset, making it easier to follow best practices for coding.

It is considered the place to be for open-source projects, and also provides a platform for web developers to showcase their skills.

Wouldn't it be great if you could edit your HTML and CSS in real-time, or debug your JavaScript, all while viewing a thorough performance analysis of your website?

Google's built-in Chrome Developer Tools let you do just that. Bundled and available in both Chrome and Safari, they allow developers access into the internals of their web application. On top of this, a palette of network tools can help optimize your loading flows, while a timeline gives you a deeper understanding of what the browser is doing at any given moment.

Google release an update every six weeks–so check out their website as well as the Google Developers YouTube channel to keep your skills up-to-date. (2500)

*4.2. Write a short summary of the text for additional reading*

*4.3. Give a short explanation of the terms 'Web development' and 'Web development tools'*

# UNIT 10. Some Basic Elements of a Web Page

---

*Learning objectives*

- to review the essential elements of a web page
- to understand what impact each element has and how it contributes to the general user experience
- to learn some computer terms used in web page development

*Key words and phrases. Give Russian equivalents and remember the meanings of the key words and phrases used in the text*

Header, to scroll the page, website layout, trial version, call-to-action button, to hide, hamburger menu, sticky header, two-layer navigation, brand identity, to catch attention, hero section, to grab attention, footer, slider, carousel, internal search, the search query, shortcut, misinterpretation, lower bounce rate, slimmer, to get aware of, a search box, logo of the company, to apply a technique, website content.

döner menü

yanlı yorum

*Read the following text and do the exercises given after it*

***Header*** is the upper (top) part of the webpage. Being the area people see before scrolling the page in their first seconds on the website, the header is an element of strategic importance. It is expected from the header to provide the core navigation around the website so that users could scan it in split seconds and jump to the main pages that can help them. Headers are also referred to as site menus and positioned as an element of primary navigation in the website layout. Headers may include a bunch of meaningful layout elements, for example:

- basic elements of brand identity, usually a logo
- call-to-action button
- links to basic categories of website content
- links to the social networks
- basic contact information (telephone number, e-mail address, etc.)
- switcher of the languages in case of the multilingual interface

- search field
- subscription field or button
- links to interaction with the product such as trial version, downloading from the AppStore, etc.

What makes a header a vital element contributing to web usability is the fact that it is placed in the most scannable zone of a web page. Whatever is the scanning pattern users stick to on a website, it starts from the top part of the page, scanned from left to right for languages using the same reading and writing pattern. Some of the popular design practices for web headers include:

- hamburger menu: hiding the set of links to different pages or sections behind the hamburger button called so as it consists of horizontal lines looking like a typical bread-meat-bread hamburger.
- sticky header: header that doesn't hide away but sticks to the top part of the page when users are scrolling the page down. This way core navigation area is available at any point of interaction, which can be helpful in terms of content-heavy pages with long scrolling.
- two-layer navigation: a sort of double set of navigation sites in the header to separate two different routes of navigation that are both important for web usability.

One more widely-used pattern for website headers is making a logo clickable and opening or refreshing the home page after it's clicked. If you are interested in how it works, visit https://blog.tubikstudio.com/anatomy-of-web-page

***A call-to-action (CTA) button*** is an element of a user interface aimed at encouraging a user to take a certain action. This action presents a conversion for a particular page or screen (for example, buy, contact, subscribe, etc.). In other words, it turns a passive user into an active one. This type of button differs from all the other buttons on the page or screen due to its engaging nature: it has to catch attention and stimulate users to do the required action.

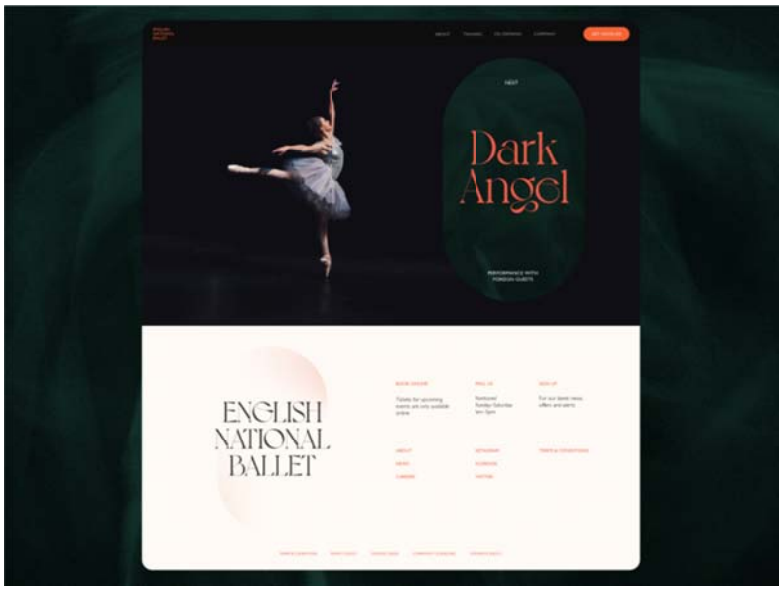
Effective call-to-action buttons are easy to notice; designers intentionally create them so that website visitors could see them in split seconds and respond. That's why they are usually bold buttons containing microcopy with a particular call to action (e.g., "Learn more" or "Buy it

now"), which explains the main action for this page and encourages a user to do it.

***Hero section*** is the above-the-fold (pre-scroll area of the web page containing the element that presents the strong visual hook: a hero image, slider, catchy piece of typography, video, or anything else attracting visitors' attention and transfers a needed message to them. The main idea is that the visual hook in the hero section instantly grabs attention and allows for setting the quick visual, emotional, and informative connection with the users, engaging them to scroll or push the buttons to learn more.

***Footer*** is the lower (bottom) part of the web page which usually marks its end. Being another common zone of global website navigation, the footer provides the additional field for useful links and data users may be interested in finding. Footers can include:

- brand identity signs, usually the name and logo of the company or product
- links to user support sections, for example, FAQ page, About page, Privacy Policy, Terms and Conditions, Support Team, etc.
- credits to website creators
- contact forms and information
- links to company or product accounts in social networks
- testimonials and badges
- certification signs
- subscription field or button.



***Slider*** is an interactive element that applies a technique of a slideshow or carousel to present different products, offers, etc. It is especially popular as a part of e-commerce and business websites to present a sort of gallery of products or services.

***Internal search*** is a functionality that enables a visitor to browse the content inside the website and shows it according to the search query. Tuned correctly, it shows the relevant content, and this way provides the shortcut to what the user needs. Thus, the internal search saves the user's time and effort, amplifies usability and desirability of the digital product, helps retain users, and increases conversion rates. The interactive element responsible for the internal search in the user interface is a search field, also called a search box or search bar: it enables a user to type in the search query and, this way find the pieces of content that are needed.

If your website is made of 50+ pages, it's high time you considered applying the internal search. Well-designed and easily found search field enables the user to jump to the necessary point without browsing through the numerous pages and menus. In case you have a single-page website, if your app or website is concise and not heavily packed with content, the internal search is not needed. One more example is also here https://blog.tubikstudio.com/anatomy-of-web-page/

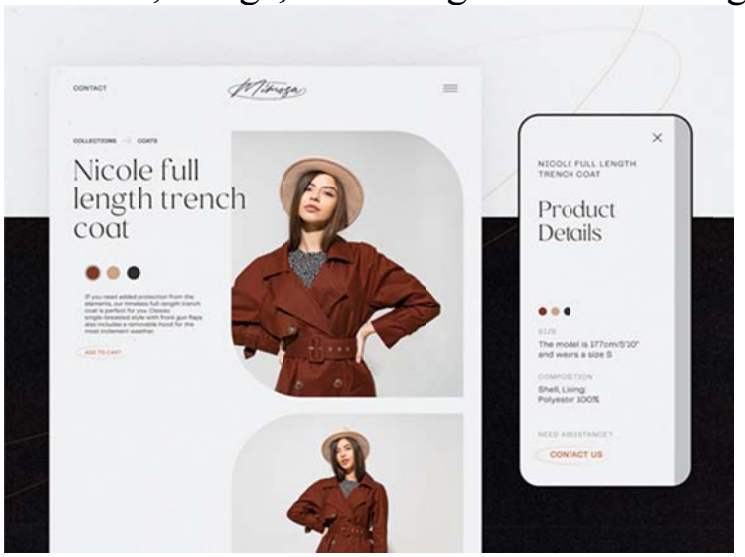
***Breadcrumbs*** are navigation elements used to support users in a journey around the website: they get aware of where they are on the website and can get used to the website structure more easily. So, breadcrumbs present the secondary level of navigation and increase website usability in case it has lots of pages.

Some of the benefits of breadcrumbs are:

- increased findability
- fewer clicks needed
- effective use of screen space line with plain-looking text elements that don't need much space
- no misinterpretation
- lower bounce rate: breadcrumbs are a great support for first-time visitors or people that have no everyday experience of dealing with complex websites.

As well as with internal website search, breadcrumbs are helpful in cases when the website has multiple pages and a complex hierarchy consisting of multiple layers. Breadcrumbs are common – and expected by users – in big e-commerce websites and platforms, media and news

websites, blogs, and magazines covering a wide range of topics, etc.




[Abridged from https://blog.tubikstudio.com/anatomy-of-web-page/]

## 1. Text-based Assignments

*1.1. Give English equivalents of the following words and word combinations:*

Шапка веб сайта; мгновение ока; набор элементов; «залипать» на сайте; шаблон; скрывать ссылки; меню «гамбургер» *(иконка из трех горизонтальных линий, при нажатии на которую открывается меню)*; страницы с большим содержанием; удобство пользования веб страницами; обновление начальной страницы; привлекательная природа; намеренно; вверху страницы; привлекать внимание; визуальный крючок; слайдер; подвал (футер); хлебные крошки (навигационная цепочка); поисковая строка; коэффициент конверсии; показатель «ненужных просмотров»; линия экрана; внутренний поиск.

*1.2. Match the following computer terms with their definitions*

| | |
|---|---|
| 1. Web page c | a) usually the top area of a website, containing the company logo, main navigation, phone number. |
| 2. Layout d | b) anything on a website that asks the user to take an action. Usually, this is something such as 'buy now', 'call us today', 'order now', 'don't see what you're looking for? Call us now on xxx', 'ready to give it a try? Start a free trial now'. |

| | |
|---|---|
| 3. Header a | c) a single document, generally written in HTML/XHTML, meant to be viewed in a web browser. In many cases, web pages also include other coding and programming (such as PHP, Ruby on Rails, or ASP). |
| 4. Footer e | d) describes what is on a page and where, the page structure. |
| 5. Breadcrumbs f | e) usually, the bottom area of a webpage, consisting of links to internal pages including legal information etc, Copywrite info etc. |
| 6. A Call-to-Action b | f) the small links under the Header that show the containing sections of a given page, usually displayed as 'home > category > subcategory > current page'. These exist to help users navigate and understand site structure. |

*1.3. Answer the following questions on the text*

1) What does header include? What makes a header a vital element contributing to web usability?
2) Is footer the lower part or the top part of the web page?
3) What is the role of slider?
4) Which principle? "The ease of use / how user friendly the website is". Accessibility, Usability, Clarity or Content?
5) What is internal search enable a visitor to do?
6) Do breadcrumbs present the primary or the secondary level of navigation?
7) Why does a call-to-action button differ from all the other buttons?

*1.4. Read the text again and decide if the following statements are true or false*

1) A call-to-action button is an element of strategic importance. T
2) Headers are also referred to as site menus and positioned as an element of primary navigation in the website layout. T
3) Some of the popular design practices for web headers include: hamburger menu, sticky header and internal search. T
4) Two-layer navigation is a sort of double set of navigation sites in the header to separate two different routes of navigation. T

5) A call-to-action (CTA) button turns a passive user into an active one. T
6) The main idea is that the visual hook in the hero section instantly grabs attention and allows for setting the quick visual, emotional, and informative connection with the users. T
7) Footers can include: call-to-action button, links to basic categories of website content, links to the social networks, basic contact information (telephone number, e-mail address, etc.). T
8) The internal search amplifies usability and desirability of the digital product, helps retain users, and increases conversion rates. But it takes a lot of time. F
9) Breadcrumbs are helpful in cases when the website has multiple pages and a complex hierarchy consisting of multiple layers. T

## 2. Focus on Grammar

*2.1. Revise the following constructions with Participles*

| | | |
|---|---|---|
| *1. The Objective - with - the - Participle I Construction (Complex Object with the Participle)* | употребляется тогда, когда говорящий хочет подчеркнуть, что действие, выраженное причастием, на завершено и протекает в момент речи. | I saw **her working** on her project. |
| *2. The Subjective Participial Construction (Complex Subject)* | употребляется с глаголами чувственного и умственного восприятия в страдательном залоге. Конструкция характерна для письменной речи. | Jane was found **working** on her project. |
| *3. The Nominative Absolute Participial Construction* | выражает действие, не связанное с действием, обозначенным глаголом-сказуемым предложения. Сам оборот состоит из существительного в общем падеже (реже местоимения в именительном падеже) и причастия. Этот оборот характерен для письменной речи . | **The article having been translated**, the student showed it to the teacher. После того как (когда) статья была переведена, студент показал её преподавателю. (обстоятельство времени) |

*2.2. Point out the complex object and complex subject with the participle constructions. Translate the following sentences into Russian*

1) I heard him moving about, and presently he was back a new printer.
2) Walking into the center of the great empty office, he stood still.
3) Lifting the telephone, she answered, 'Yes?'
4) She liked to watch him doing things, printing, coding, and designing.
5) He was always late on principle, his principle being that punctuality is the thief of time.
6) You seem wanting to get out of it.
7) Having been checked our papers, she didn't feel fit to work.
8) I don't like people coming too close.
9) The rule having been answered, we passed on analyzing the sentences.
10) There was no money, Hilbert having used all the possessed.
11) Nobody having anything more to say, he went out.
12) The task was understood as being too difficult for the students.
13) Christian seemed enduring a profound spiritual crisis.
14) We watched the plane landing.

*2.3. Make up your own sentences with Participle 1 as parenthesis (independent element).*

Participle 1 is the headword of the phrase, the meaning of which is a comment on the whole sentence or some part of it.

*Allowing for* – делая поправку на
*Generally speaking* – вообще говоря
*Judging by/ from* – судя по
*Joking aside* – кроме шуток, шутки в сторону
*Leaning aside* – не говоря о
*Putting it mildly* – мягко выражаясь
*Taking into consideration* – принимая во внимание
*Talking/ speaking of* – к вопросу о, говоря о

E.g. Generally speaking it's you duty to discuss all the necessary details with our customer.

## 3. Discussion

*3.1. Watch this video and give your comments on the top 5 websites https://www.youtube.com/watch?v=AmHEfTSBXiY&ab_channel=Flux*

*3.2. Discuss the following questions*

1) What is good Web design? What is important in Web design?
2) Is Web Design graphic design?
3) How do I start a website?
4) Is HTML and CSS enough to create a website?
5) What are the different types of website layouts?

*3.3. Explain the following terms in English*

Scrolling, trial version, multilingual interface, hero section, e-commerce, breadcrumbs, conversion rates, search bar, logo.

*3.4. Choose your answers to the questions and discuss them with your partner. If you are not sure you can find the necessary information in the Internet*

### Web pages and web apps test questions

1) What is a dynamic website?
   A) A website that has interactive element
   B) A website that has no form of interactivity
   C) A website that is very large
2) What web development languages are most websites written in?
   A) High level languages like C++ and Java
   B) HTML and CSS with some scripting languages like JavaScript and PHP
   C) Pseudo code
3) What is a mashup *(приложение, комбинирующее в себе контент с различных источников)*?
   A) Two or more websites that have been joined together
   B) A website or application which mixes code from different external sources

C) A website that isn't working properly

4) What are cookies?
   A) Viruses that are downloaded onto your computer
   B) Programs that are stored on your computer that tell a website if you have been to that site before
   C) Text files that are stored on your computer that tell a website if you have been to that site before

5) What identifies the kind of device that is accessing the website?
   A) The server
   B) A protocol
   C) The web browser

6) What is the difference between the client-side scripts and the Server-Style Scripts?
   A) Client-side scripts are programs that are processed by the web browser and server-style scripts are processed by the web server
   B) Client-side scripts are programs that are processed by the web server and server-style scripts are processed by the web browser
   C) Client-side scripts process the static parts of the web site and server-style scripts process the dynamic parts of the web site

7) How do search engines rank the website result they receive?
   A) Websites appear higher up a list of results purely based on how many hits have occurred on that website altogether, since it was created
   B) Websites appear higher up a list of search results because they have paid the web browser to be ranked higher
   C) Websites appear higher up a list of results because they are judged to be more important by the search algorithm. This is measured by the popularity of a site and how many connections it has with other websites

8) What is cloud computing?
   A) Cloud computing is storing and using services online, rather than storing them locally on a device such as a hard drive
   B) Cloud computing is accessing your home computer over the internet
   C) Cloud computing is accessing stored data online using only mobile devices

## 4. Additional reading

*4.1. Read and translate the text about other elements of a Web Page*

**The menu** is one of the core navigation elements in user interfaces. It is a graphical control that presents the options of interactions with the interface. Basically, it can be the list of commands – in this case, options will be presented with verbs marking possible actions like, for example, "save," "delete," "buy," "send," etc. A menu can also present the categories along which the content is organized in the given interface, and this can be the high time for using nouns marking them.

Menus can have different locations in the interface (side menus, header menus, footer menus, etc.) and different ways of appearance and interaction (drop-down menus, drop-up menus, sliding menus, etc.)

Some popular types found on diverse websites are:

*Classic horizontal menu:* the most common and well-recognized type of menu which presents the core navigation organized as a horizontal line in the website header, mentioned above

*Sidebar menu:* quite a classic type, presents a vertical list of options sticking to the left or right side of the web page

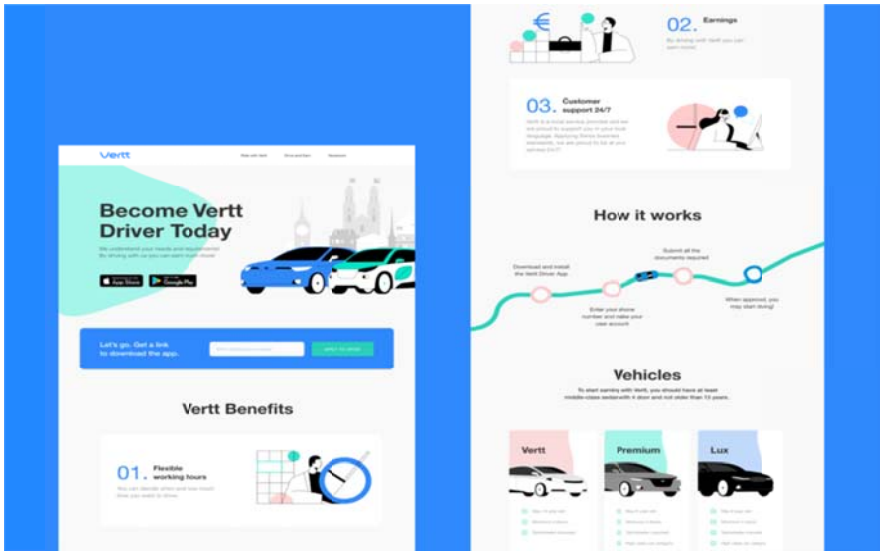
*Dropdown menu:* a more complex type of menu for content-heavy websites; here, the list of additional options opens below the primary one when it's clicked or hovered. Another similar option is the dropup menu, when the list opens up, not down, but the essence is the same.

*Megamenu:* that's the complex expandable menu in which the big list of multiple choices is presented in a two-dimensional dropdown layout; this approach is effective for cases with a vast number of options.

*Hamburger menu:* when the hamburger button (typically marked with three horizontal lines) is clicked, the menu expands. This option saves space and is often applied to mobile versions of websites. One more example is also here https://blog.tubikstudio.com/anatomy-of-web-page/

**Form** is an interactive element that allows users to send information to the system or server. In a nutshell, it is a digital version of any real paper form we have to fill in to provide someone with the arranged information; however, digital forms can have more options and functionality to make this process even more smooth, clear, and user-friendly. As it is a traditional and well-recognized pattern of collecting the data, users deal with forms quite often in their digital lives, starting from

the process of registration, adding personal or financial details, making payments, sending feedback, subscribing to a newsletter, etc.




As forms present the actual point of communication between the user and the digital product, they have to be super simple and easy to use. And the simpler the UI element should be, the more designers' thoughts and effort should be put into it, right? Make the logic of data input thought-out, the navigation of the form intuitive, and the number of required actions minimized.

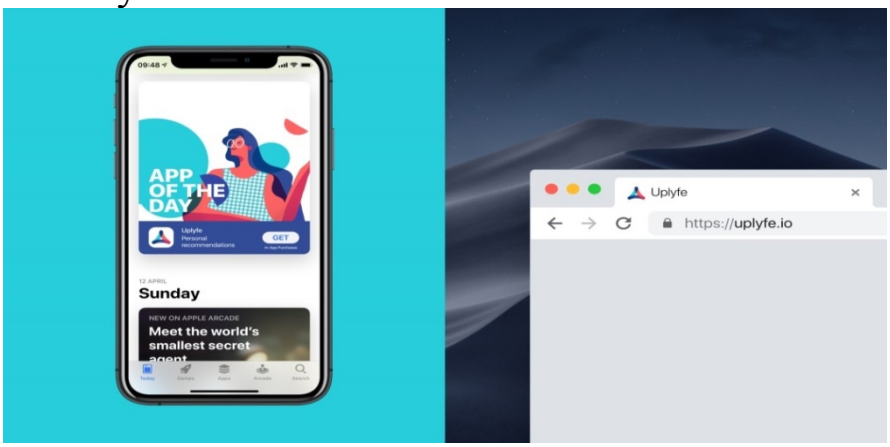
Cards, also called tiles, are layout elements that help arrange and visualize homogeneous data or content in a scannable and easy-to-use way. Cards are usually organized in a sort of grid, but each card looks like a separate piece in this system. Cards can combine different types of content about a particular item. For example, a product preview card on a catalog page can include an image, a title, basic functionality of adding it to a shopping cart or saving to the wishlist, etc.

Art Institute blog uses ultra-minimalistic cards, separated only by negative space but organized clearly to be distinguished.

Video is not a really basic part of a web page, but with the progress of web development solutions and technical abilities, we can find it more and more often on the website of different kinds these days. A catchy video crafted with an understanding of the target audience is a tool attracting customers' attention as well as a well-checked method of informing them quickly and brightly. Video content activates several channels of perception – audio, visual, motion – simultaneously, and usually does that wrapped in telling a story. Such a combination of factors often makes a video presentation strong, emotional, and memorable.

We can come across many other types of videos that help users quickly catch the idea of a product, set the atmosphere, send the needed message, engage in trying the service, demonstrate how the tool, app, or software works, share feedback from users, and so on, and so forth. However, there are essential points to consider, such as loading time, contrast issue, responsiveness, and other pitfalls that can spoil user experience in the case of video integration into a web page.

**Favicon,** also known as URL icon or bookmark icon, is a special type of symbol representing the product or brand in the URL-line of the browser and in the bookmark tab. It allows users to get a quick visual connection with it while they are browsing. This interface element proved itself effective for productive website promotion and good recognizability of its visual identity. Being super small, it makes a great contribution to web usability.




## Tags

That's another element of secondary navigation level, often found in blogs and websites with plenty of homogenous content. Tag is presented with a keyword or phrase that enables users to jump directly to the items

marked up with it. Tags are actually pieces of metadata that provide quick access to specific content categories, so they support navigation with the additional way of content classification. Moreover, tags are often the elements that users create by themselves, so they become an alternative to the names of categories that are fixed by the website and can't be changed by users. [https://blog.tubikstudio.com/anatomy-of-web-page/]

*4.2. Work with a partner. Ask some questions to each other on the text about some other elements of a web Page*

*4.3. Translate the following sentences into English*

1) User interface (UI) элементы – это части, которые дизайнеры используют для создания приложений или веб-сайтов.
2) Они добавляют интерактивность в пользовательский интерфейс, предоставляя пользователю точки соприкосновения при навигации по ним.
3) Хлебные крошки (навигационная цепочка, англ. Breadcrumbs) – это элемент навигации по сайту, который представляет собой путь от корня сайта, до текущей страницы, на которой в настоящий момент находится пользователь.
4) Хлебные крошки обычно представляют собой полосу в верхней части страницы, обычно под шапкой сайта.
5) Нотификации дают юзеру понять, что есть что-то новое, например, сообщение или какое-то системное уведомление.
6) Следующей обязательной составляющей частью web-страницы являются Элементы навигации – гиперссылки, связывающие данный документ с другими разделами сайта.
7) Элементы навигации могут быть выполнены в виде текстовых строк, графических объектов, то есть кнопок, либо активных компонентов.
8) Если web-страница является стартовым документом, в нижней ее части также размещают счетчик посещений – небольшой сценарий, вызывающий установленный на сервере CGI-скрипт, который фиксирует каждое открытие документа в браузере пользователей, изменяя значение индикатора счетчика.
9) Благодаря этому web-мастер без труда определит количество посетителей, навестивших его страничку в течение какого-либо времени.

# UNIT 11. Application Development and Types of Application Development Methodologies

---

*Learning objectives*

- to acquire basic knowledge about application development
- to consider three categories most application development methodologies can be grouped into
- to consider advantages and disadvantages of three methods in application development

*Key words and phrases. Give Russian equivalents and remember the meanings of the key words and phrases used in the text*

Application development, freelance developer, software development life-cycle (SDLC), to emerge; waterfall, to line out, sequence, prototype, to divert, to accommodate, meticulous, to train junior programmers, Rapid Application development (RAD), sprint, Agile project management, highly skilled, deadline; iterative, to stick to, planned schedule, to suit the needs, to attach.

yönlendirmek Karılamak titiz Anı/seri bağlı kalmak baglamak/eklemek

*Read the following text and do the exercises given after it*

Application development is the process of designing, building, and implementing software applications. It can be done by massive organizations with large teams working on projects, or by a single freelance developer. Application development defines the process of how the application is made, and generally follows a standard methodology.

You must consider the size of the project, how specific the requirements are, how much the customer will want to change things, how large the development team is, how experienced the development team is, and the deadline for the project.

Application development is closely linked with the software development life-cycle (SDLC).

The basic stages of SDLC are: *Planning, Analysis, Design, Construction, Testing, Implementation, Support.*

The way that application development teams have accomplished these seven tasks has changed a lot in the last few decades, and numerous types of application development methods have emerged. Each methodology must provide a solution for the seven stages of the SDLC.

Most application development methodologies can be grouped into one of three categories: *Waterfall, RAD, Agile.*

**Waterfall**

The key words for the waterfall method of application development are planning and sequence. The entire project is mapped out in the planning and analysis stages. The customer comes with a very explicit list of features and functionalities for the application. Then, a project manager takes the whole process and maps it out amongst the team.

This application development method is called waterfall because once you go down, you can't go back up; everything flows downward. The development team works together over a set of time, building exactly what is lined out according to the specifications. After the architecture is designed, then only can the construction begin. The entire application is built, and then it is all tested to make sure that it is working properly. Then, it is shown to the customer and ready to be implemented.

The waterfall method assumes that the project requirements are clear and the customer and project manager have a unified and clear vision about the end result.

The advantage of the waterfall method is that it is very meticulous. It's also a good application development method to use for big projects that need to have one unifying vision. The waterfall method is also a good way to train junior programmers on parts of development without having to turn an entire project to them.

The disadvantages are that changes happen all the time. Even if the development team is able to build exactly what the customer originally wanted (which doesn't always happen), the market, technology, or the

organization may have changed so much that it is effectively useless and a waste of time.

**Rapid Application Development (RAD) Methodology**

In many ways, RAD was the opposite of the waterfall method.

RAD is based mostly on prototypes, meaning that the goal is to produce a working version of the application as quickly as possible, and then to continuously iterate after that. The application development team and the customer work very closely with each other throughout the process. RAD teams are usually small and only involve experienced developers who are skilled in many disciplines. If a project needs to divert from the original plan, RAD should be able to accommodate that easily.

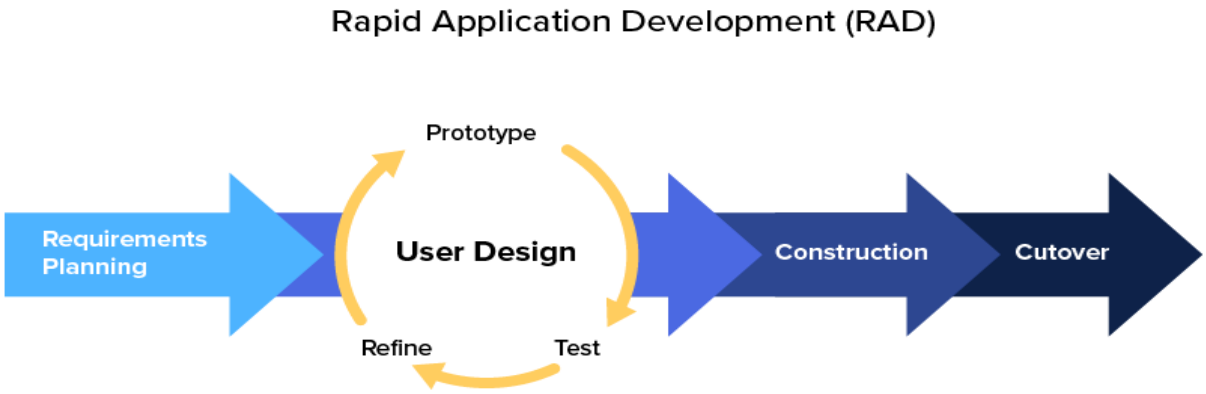



Fig. 11. Rapid Application Development

In the RAD model, as each iteration is completed, the product gets more and more refined. The early prototypes are often very rough, but give a picture of what can be. Each iteration then looks more like the finished product.

RAD's advantages are a quick and highly flexible team and a very close relationship with the customer. If changes are expected, RAD will be able to accommodate these much faster than waterfall. RAD is also never too attached to a prototype and is always willing to change it to suit the needs of the customer.

However, RAD isn't a perfect application development method. RAD requires highly skilled (and highly paid) programmers to work on a project that may change in complexity by the day. There's also less adherence to deadlines and more of a focus on adding features, which can extend delivery dates. RAD requires a lot of input from customers who may not always be available or know what they need. Additionally, for

some applications, having a prototype is not useful without seeing the entire product.

**Agile Methodology**

Agile application development is very similar to RAD, but also includes some changes to make it more suitable to larger projects. Agile is iterative, like RAD, but focuses on building features one at a time. Each feature is built in a methodical way in the team, but the customer is involved to see the features and sign off on them before the next feature is developed.

Agile uses sprints, or set of time when a certain feature should be built, tested, and presented. It tries to incorporate the entire SDLC for a feature into each sprint. This, ideally, helps to stick to a planned schedule, but also allow for frequent reviews.

Agile doesn't focus on prototypes, but only presents completed work after the sprint is over. So while the customer is informed more often than waterfall, the customer only ever sees finished work, unlike RAD.

Agile project management methodology is also more team or squad based. With RAD, you are working directly with a programmer. With Agile, the application development team will also include testers, UX designers, technical writers, and many others.

[Abridged from https://kissflow.com/low-code/rad/types-of-application-development-methodologies]

## 1. Text-based Assignments

*1.1. Give English equivalents of the following words and word combinations:*

Внедрение программных приложений; внештатный разработчик; требования; стандартная методология; выполнить задачу; относить в одну из категорий; намечаться; подробный перечень; функции приложения; менеджер проекта; в соответствии со спецификациями; определенный период времени; пустая трата времени; быстрая разработка приложений; рабочая версия; заказчик; отклониться; привязываться к прототипу; в соответствии с потребностями; ориентирована на группу; запланированный график; метод гибкой разработки приложений.

*1.2. Match the following computer terms with their definitions*

| | |
|---|---|
| 1. RAD c | a) a linear, sequential approach to the software development life cycle (SDLC) that is popular in software engineering and product development |
| 2. software testing d | b) a repeatable fixed time-box during which a "Done" product of the highest possible value is created |
| 3. sprint b | c) a form of Agile software development methodology that prioritizes rapid prototype releases and iterations. |
| 4. implement f | d) a method to check whether the actual software product matches expected requirements and to ensure that software product is Defect free. It involves execution of software/system components using manual or automated tools to evaluate one or more properties of interest |
| 5. prototype g | f) to recognize and use an element of code or a programming resource that is written into the program |
| 6. waterfall method a | g) an original model, form or an instance that serves as a basis for other processes. In software technology, this term is a working example through which a new model or a new version of an existing product can be derived |

*1.3. Match the following synonyms from the text*

1. complexity
2. entire
3. to suit
4. to incorporate
5. properly
6. management
7. agile
8. to assume
9. meticulous

a) detail, very careful
b) fitly, suitably
c) smart, active, quick
d) to correspond, to match
e) full, complete
f) to unite, combine, mix, consolidate
g) administration, guidance
h) complication, entanglement, involved character
i) to suppose

*1.4. Answer the following questions on the text*

1) What factors are there those go into how application development is done?
2) How many stages does software development life-cycle include?
3) Why is one of the application development methods called waterfall?
4) Is RAD based mostly on prototypes, isn't it? Is it a perfect application development method?
5) What does Agile application development use? Does it focus on prototypes?
6) What is the goal of Rapid Application Development?

*1.5. Complete the following table using the information about advantages and disadvantages of different methods in application development. You can use some additional information from the Internet if necessary*

| | **Advantages** | **Disadvantages** |
|---|---|---|
| **Waterfall** | 1. Meticulous, good for big projects.<br>2. … | 1. Changes happen all the time.<br>2…. |
| **RAD methodology** | | |
| **Agile methodology** | | |

## 2. Focus on Grammar

*2.1. Study the following table and make up your own sentences with Gerunds used in different functions*

| **Functions of the Gerund** | Example | Translation |
|---|---|---|
| **Subject** | Having a prototype is not useful without seeing the entire product. | Иметь прототип бесполезно, если не видеть весь продукт. |
| **Compound Predicate** | The negotiations are still far from being ended.<br>We began discussing the needs of our customers . | Переговоры еще отнюдь не закончены.<br>Мы начали обсуждать потребности наших заказчиков. |

| | | |
|---|---|---|
| **Object** | I remember having turned off my computer. | Я помню, что выключил компьютер. |
| **Attribute** | The proposal for reducing the working hours is now being discussed. | Предложение о сокращении рабочей недели сейчас обсуждается. |
| **Adverbial Modifier** | We cannot use the device without testing it. | Мы не можем использовать этот прибор без его проверки. |

*2.2. Read and translate the sentences. Comment on the functions of the Gerunds*

1) It's no use talking to the headmaster.
2) Finding a parking space is really difficult in this part of the city.
3) She has always dreamt of becoming a good programmer.
4) You can't learn without making mistakes.
5) Solving such a problem is not an easy task.
6) I suggest telephoning the hospitals before asking the police to look for him.
7) She took a long time to get over losing her dog.
8) Tom is proud of donating his free time to the charity.
9) Agile is iterative, like RAD, but focuses on building features one at a time.

*2.3. Choose the right variant*

1) You must keep on _____ the computer until you understand how _____ all of the programs.

a) practice, to use; b) practicing, using; c) practicing, to use

2) Will you excuse me for _____ an obvious precaution?

a) taking b) take c) to take

3) I'll never forget _____ my first entrance examination. It was a complete failure.

a) to take b) having been taken c) taking

4) If people delay _____ their bills, they only incur more and more interest charges.

a) to pay b) paying c) to be paying

5) We both sat in silence for some little time after _____ to this extraordinary story.

a) listening b) listen c) having been listen

6) I look forward to _____ you the next time I'm in town. I'll be sure to let you _____ ahead of time so that we can plan to get together.
   a) see, to know; b) see, knowing; c) seeing, know
7) His _____ a bad mark did not surprise anybody.
   a) receiving b) being received c) having received
8) He finished _____ this file form the Internet.
   a) downloading b) being downloaded c) having downloaded

## 3. Discussion

*3.1. Discuss the following questions concerning Application development*

1) Why is software testing important?
2) How will your application make money?
3) Web apps are relatively easy to maintain. Can you explain why it is so?
4) Users interact with different web browsers, and as a result, the usage patterns and performance metrics used to create a product roadmap are more challenging to collect. Is it advantage or disadvantage for Web Apps?
5) What should you look for in a development team?
6) How much does it cost to develop an App?

*3.2. Prepare a one minute speech on the following topic and present it to the class*

'The best ideas are of little use if they cannot be implemented. Good application design alone is not enough; efficient, high-quality development is also required'

*3.3. Prepare a short report about one of the methods of application development*

## 4. Additional Reading

*4.1. Read and translate the following text about World Wide Web display languages*

### HTML

The World Wide Web is a system for displaying text, graphics, and audio retrieved over the Internet on a computer monitor. Each retrieval unit is known as a Web page, and such pages frequently contain "links" that allow related pages to be retrieved. HTML (*h*yper*t*ext *m*arkup *l*anguage) is the markup language for encoding Web pages. It was designed by Tim Berners-Lee at the CERN nuclear physics laboratory in Switzerland during the 1980s and is defined by an SGML DTD. HTML markup tags specify document elements such as headings, paragraphs, and tables. They mark up a document for display by a computer program known as a Web browser. The browser interprets the tags, displaying the headings, paragraphs, and tables in a layout that is adapted to the screen size and fonts available to it.

HTML documents also contain anchors, which are tags that specify links to other Web pages. An anchor has the form <A HREF= "http://www.britannica.com"> Encyclopædia Britannica</A>, where the quoted string is the URL (uniform resource locator) to which the link points (the Web "address") and the text following it is what appears in a Web browser, underlined to show that it is a link to another page. What is displayed as a single page may also be formed from multiple URLs, some containing text and others graphics.

### XML

HTML does not allow one to define new text elements; that is, it is not extensible. XML (extensible markup language) is a simplified form of SGML intended for documents that are published on the Web. Like SGML, XML uses DTDs to define document types and the meanings of tags used in them. XML adopts conventions that make it easy to parse, such as that document entities are marked by both a beginning and an ending tag, such as <BEGIN>…</BEGIN>. XML provides more kinds of hypertext links than HTML, such as bidirectional links and links relative to a document subsection.

## Web scripting

Web pages marked up with HTML or XML are largely static documents. Web scripting can add information to a page as a reader uses it or let the reader enter information that may, for example, be passed on to the order department of an online business. CGI (common gateway interface) provides one mechanism; it transmits requests and responses between the reader's Web browser and the Web server that provides the page. The CGI component on the server contains small programs called scripts that take information from the browser system or provide it for display. A simple script might ask the reader's name, determine the Internet address of the system that the reader uses, and print a greeting. Scripts may be written in any programming language, but, because they are generally simple text-processing routines, scripting languages like PERL are particularly appropriate.

Another approach is to use a language designed for Web scripts to be executed by the browser. JavaScript is one such language, designed by the Netscape Communications Corp., which may be used with both Netscape's and Microsoft's browsers. JavaScript is a simple language, quite different from Java. A JavaScript program may be embedded in a Web page with the HTML tag <script language="JavaScript">. JavaScript instructions following that tag will be executed by the browser when the page is selected. In order to speed up display of dynamic (interactive) pages, JavaScript is often combined with XML or some other language for exchanging information between the server and the client's browser. In particular, the XMLHttpRequest command enables asynchronous data requests from the server without requiring the server to resend the entire Web page. This approach, or "philosophy," of programming is called Ajax (*a*synchronous *J*avaScript *a*nd *X*ML).

VB Script is a subset of Visual Basic. Originally developed for Microsoft's Office suite of programs, it was later used for Web scripting as well. Its capabilities are similar to those of JavaScript, and it may be embedded in HTML in the same fashion.

Behind the use of such scripting languages for Web programming lies the idea of component programming, in which programs are constructed by combining independent previously written components without any further language processing. JavaScript and VB Script programs were designed as components that may be attached to Web browsers to control how they display information.

[https://www.britannica.com/technology/computer-programming-language]

*4.2. Ask five questions on the content of the text for additional reading*

*4.3. Translate the following sentences*

1) Покупки в приложении, оплата подписки или премиум-версии, размещение рекламы, продажа данных – все эти способы монетизации можно использовать, даже если вы распространяете приложение не бесплатно.
2) Внутренняя архитектура зависит от функционала мобильного приложения и выбранного способа обработки и хранения данных.
3) Обычно мы составляем два списка – характеристик, которыми должно обладать приложение, и ключевых визуальных элементов. Они становятся фундаментом для всех будущих архитектурных работ.
4) В конце каждого спринта обсуждайте его результаты с заинтересованными сторонами.
5) Разработка мобильного приложения не заканчивается с его публикацией в сторах. Даже за умеренно популярными приложениями стоит целая история обновлений.
6) В понятие разработки мобильных приложений для смартфонов, планшетов и прочих мобильных устройств входит написание программного кода с целью создания программ, которые будут работать на определенных мобильных платформах.
7) На сегодняшний день существует 2 основные платформы мобильных операционных систем – Android и iOS, и менее популярные Windows Phone и Symbian).

# UNIT 12. Game Engine

---

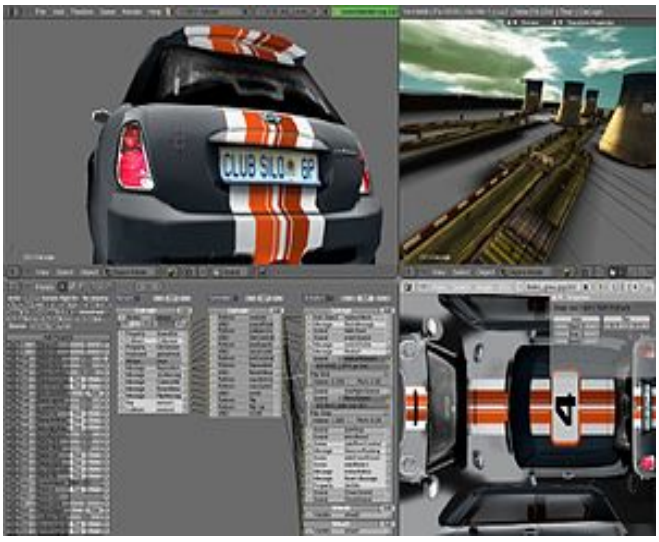
*Learning objectives:*

- to acquire basic knowledge about game engines and their types
- to consider what game engines provide

*Key words and phrases. Give Russian equivalents and remember the meanings of the key words and phrases*

Game engine, software framework, to construct games, implementer, a suite of tools, a rendering engine, “middleware”, platform abstraction, a component-based architecture, to comprise, “graphics engine”, to become outdated, a game application, multiple platform, artificial intelligence, to simplify, to run on, a custom engine, scene graph, training simulation basitletirmek

*Read the following text and do the exercises given after it*



A game engine is a software framework primarily designed for the development of video games, and generally includes relevant libraries and support programs. The "engine" terminology is similar to the term "software engine" used in the software industry.

Game engine can also refer to the development software utilizing this framework, typically offering a suite of tools and features for developing games.

Developers can use game engines to construct games for video game consoles and other types of computers. The core functionality typically provided by a game engine may include a rendering engine ("renderer") for 2D or 3D graphics, a physics engine or collision detection (and collision response), sound, scripting, animation, artificial intelligence,

networking, streaming, memory management, threading, localization support, scene graph, and video support for cinematics.

Game engine implementers often economize on the process of game development by reusing/adapting, in large part, the same game engine to produce different games or to aid in porting games to multiple platforms.

In many cases, game engines provide a suite of visual development tools in addition to reusable software components. These tools are generally provided in an integrated development environment to enable simplified, rapid development of games in a data-driven manner. Game-engine developers often attempt to preempt implementer needs by developing robust software suites which include many elements a game developer may need to build a game. Most game-engine suites provide facilities that ease development, such as graphics, sound, physics and artificial-intelligence (AI) functions. These game engines are sometimes called "middleware" because, as with the business sense of the term, they provide a flexible and reusable software platform which provides all the core functionality needed, right out of the box, to develop a game application while reducing costs, complexities, and time-to-market — all critical factors in the highly competitive video-game industry.

Fig. 12. The screen of Quake

Like other types of middleware, game engines usually provide platform abstraction, allowing the same game to run on various platforms (including game consoles and personal computers) with few, if any, changes made to the game source-code. Often, programmers design game engines with a component-based architecture that allows specific systems in the engine to be replaced or extended with more

specialized (and often more expensive) game-middleware components. Some game engines comprise a series of loosely connected game middleware components that can be selectively combined to create a custom engine, instead of the more common approach of extending or customizing a flexible integrated product. However achieved, extensibility remains a high priority for game engines due to the wide variety of uses for which they are applied. Despite the specificity of the name "game engine", end-users often re-purpose game engines for other kinds of interactive applications with real-time graphical requirements - such as marketing demos, architectural visualizations, training simulations, and modeling environments.

Some game engines only provide real-time 3D rendering capabilities instead of the wide range of functionality needed by games. These engines rely upon the game developer to implement the rest of this functionality or to assemble it from other game-middleware components. These types of engines are generally referred to as a "graphics engine", "rendering engine", or "3D engine" instead of the more encompassing term "game engine". This terminology is inconsistently used, as many full-featured 3D game engines are referred to simply as "3D engines". Examples of graphics engines include: Crystal Space, Genesis3D, Irrlicht, OGRE, RealmForge, Truevision3D, and Vision Engine.

Modern game- or graphics-engines generally provide a scene graph - an object-oriented representation of the 3D game-world which often simplifies game design and can be used for more efficient rendering of vast virtual worlds.

As technology ages, the components of an engine may become outdated or insufficient for the requirements of a given project. Since the complexity of programming an entirely new engine may result in unwanted delays (or necessitate that a project restart from the beginning), an engine-development team may elect to update their existing engine with newer functionality or components.

[https://en.wikipedia.org/wiki/Game_engine]

## 1. Text-based Assignments

*1.1. Give English equivalents of the following words and word combinations:*

Соответствующие библиотеки; игровая консоль; механизм визуализации; графический движок; многопоточность; поддержка локализации; специалист по внедрению, разработчик; способ управления данными; упреждать; надежное ПО; межплатформенное ПО; срок вывода на рынок; игровой исходный код; нестандартный движок; включать ряд компонентов; каждый раз по разному, непостоянно; тренировочное моделирование; нежелательные задержки; гибкий интегрированный продукт; быть замененным; более эффективная передача.

*1.2. What do the following prefixes and suffixes mean? Translate these words from the text and determine their part of speech*

Functionality, insufficient, interactive, unwanted, reusable, implementer, complexity, selectively, restart, discontinued, currently, informative, indefinitely.

*1.3. Match the following economic terms with their definitions*

| | |
|---|---|
| 1. A graphics engine e | a) a core component of a complex software system. |
| 2. Competitive d | b) the art of creating three-dimensional images or animations showing the attributes of a proposed architectural design. |
| 3. A scene graph f | c) a physics model typically implemented in software for use in computer games |
| 4. Visualization c | d) strongly desiring to be more successful than others; as good as or better than others of a comparable nature |
| 5. A game engine a | e) any technique for creating images, diagrams, or animations to communicate a message |
| 6. Architectural rendering, b | f) a general data structure commonly used by vector based graphics editing applications and |

| | |
|---|---|
| architectural illustration, or visualization | modern computer games |

*1.4. Read the text again and decide if the following statements are true or false*

1) The "engine" terminology is not entirely similar to the term "software engine" used in the software industry. T
2) Developers can use game engines to construct games for video game consoles and other types of computers. T
3) The core functionality typically provided by a game engine may include "3D engine" for 2D or 3D graphics. T
4) In many cases, game engines provide a suite of visual development tools in addition to reusable software components. T
5) Complexity of programming is the only requirement in today's demanding market of video-game industry. F
6) Modern game- or graphics-engines generally provide a scene graph - an object-oriented representation of the 3D game-world. T(F?)
7) End-users often re-purpose game engines for other kinds of interactive applications with real-time graphical requirements. F
8) Game engine implementers never economize on the process of game development because they want to produce different games. FF

*1.5. Answer the following questions on the text*

1) What is a game engine?
2) What does a game engine include?
3) What types of engines are referred to as "graphics" engines? Can you give any examples of them?
4) What do game engines provide?
5) Is it necessary to know how to add realism to a character, how to add graphics effects or how to animate a sprite? Or is this all taken care by the game engine?
6) If you're only aiming for a small 2D project, you probably don't need a chunky engine that's going to come with a lot of features you don't need. Do you agree with this statement?

7) What can you say about development of engines as technology ages?

## 2. Focus on Grammar

*2.1. State whether the –ing forms given in the following sentences are Participles or Gerunds. In the case of Participles define the noun or pronoun they qualify. Translate the sentences into Russian*

1) Supporting multiple programs and users is the function of mainframe operating systems.
2) Designing webpages you needn't learn how to program in HTML.
3) There exists special-purpose memory where writing is seldom necessary.
4) Programming involves analyzing the problem to be solved.
5) The data being transmitted is of great importance.
6) The aim of our seminar is studying basic stages of programming.
7) Howard Aiken completed a fully automatic calculator using standard machine components.
8) While solving the arithmetical problem the computer failed.
9) Using the appropriate CAD software the designer can perform various analyses on the object.
10) The Web is an Internet service making web pages available to millions of users.

*2.2. Study the following table Conditionals*

| | ***if*** | **condition** | **result** |
|---|---|---|---|
| 1 type | | **Present Simple** | **will + base verb** |
| | If<br>If | Tara is free tomorrow,<br>they do not pass their exam,<br><br>There is **a real possibility that the condition will happen**. | he will invite her.<br>their teacher will be sad. |
| 2 type | | **Past Simple** | **would + base verb** |
| | If<br>If | it snowed next July,<br>I won the lottery,<br><br>There is **an unreal possibility that the condition will happen**. | would you be surprised?<br>I would buy a car. |

| 3 type | | **Past Perfect** | **would have + past participle** |
|---|---|---|---|
| | If<br>If | Tara had been free yesterday,<br>it had rained yesterday,<br><br>Both the condition and result are **impossible now**. | I would have invited her.<br>what would you have done? |

*2.3. Open the brackets to form Conditionals*

1) If we _____ (not / work) harder, we_____ (not pass) the exam. (1)
2) If the students _____ (not be) late for the exam, they _____ (pass). (3)
3) If she _____ (have) her laptop with her, she _____ (email) me. (2)
4) If she _____ (not go) to the meeting, I_____ (not go) either. (1)
5) If the teacher _____ (give) us lots of homework this weekend, I _____(not be) happy. (3)
6) If I _____(want) a new car, I _____(buy) one. (2)
   We _____ (not have) so many arguments if you _____ (not be) so stubborn! (2)
7) I _____ (not meet) Amanda, if I _____ (not go) to the party. (3)
8) If you _____ (arrive) early, it_____ (be) less stressful. (3)
9) If we _____ (not be) so tired, we_____ (go) out. (2)
10) If you _____ (buy) the present, I_____(wrap) it up. (1)
11) If Lucy _____ (not quit) her job soon, she _____ (go) crazy. (1)

*2.4. Rewriting Conditionals. How can you express these ideas using conditional constructions?*

**Model:** I don't know whether I'm going to Australia. Let's suppose I go. What can I do there?
– If <u>I went</u> to Australia, I <u>could see</u> the Sydney Harbour Bridge.

1) You went to the lecture. I'm sure you saw him. You can't have avoided seeing him.
2) How can I get experience in speaking English?

3) I hope I won't fail my end of semester exams. How can I make sure I pass?
4) Every time we finished an online quiz, we celebrated by having a coffee.
5) I always feel much happier when the sun is shining.
6) I wish the library was open right now. I want to borrow a book.

## 3. Discussion

*3.1. Explain the concept of game engine in your own words*

*3.2. Choose one of the game engines mentioned and prepare a report on the topic "*What are the advantages and disadvantages *of this game engine"*

Unity, Unreal Engine, GameMaker, CryEngine, MonoGame, Construct

## 4. Additional Reading

*4.1. Read and translate the following text. You can use a dictionary if necessary*

### Artificial Intelligence

Artificial intelligence is the simulation of human intelligence processes by machines, especially computer systems. Specific applications of AI include expert systems, natural language processing, speech recognition and machine vision.

As the hype around AI has accelerated, vendors have been scrambling to promote how their products and services use AI. Often what they refer to as AI is simply one component of AI, such as machine learning. AI requires a foundation of specialized hardware and software for writing and training machine learning algorithms. No one programming language is synonymous with AI, but a few, including Python, R and Java, are popular.

In general, AI systems work by ingesting large amounts of labeled training data, analyzing the data for correlations and patterns, and using these patterns to make predictions about future states. In this way, a chatbot that is fed examples of text chats can learn to produce lifelike

exchanges with people, or an image recognition tool can learn to identify and describe objects in images by reviewing millions of examples.

AI programming focuses on three cognitive skills: learning, reasoning and self-correction.

**Learning processes.** This aspect of AI programming focuses on acquiring data and creating rules for how to turn the data into actionable information. The rules, which are called algorithms, provide computing devices with step-by-step instructions for how to complete a specific task.

**Reasoning processes.** This aspect of AI programming focuses on choosing the right algorithm to reach a desired outcome.

**Self-correction processes.** This aspect of AI programming is designed to continually fine-tune algorithms and ensure they provide the most accurate results possible.

AI is important because it can give enterprises insights into their operations that they may not have been aware of previously and because, in some cases, AI can perform tasks better than humans. Particularly when it comes to repetitive, detail-oriented tasks like analyzing large numbers of legal documents to ensure relevant fields are filled in properly, AI tools often complete jobs quickly and with relatively few errors.

This has helped fuel an explosion in efficiency and opened the door to entirely new business opportunities for some larger enterprises. Prior to the current wave of AI, it would have been hard to imagine using computer software to connect riders to taxis, but today Uber has become one of the largest companies in the world by doing just that. It utilizes sophisticated machine learning algorithms to predict when people are likely to need rides in certain areas, which helps proactively get drivers on the road before they're needed. As another example, Google has become one of the largest players for a range of online services by using machine learning to understand how people use their services and then improving them. In 2017, the company's CEO, Sundar Pichai, pronounced that Google would operate as an "AI first" company.

Today's largest and most successful enterprises have used AI to improve their operations and gain advantage on their competitors.

Artificial neural networks and deep learning artificial intelligence technologies are quickly evolving, primarily because AI processes large amounts of data much faster and makes predictions more accurately than humanly possible.

While the huge volume of data being created on a daily basis would bury a human researcher, AI applications that use machine learning can take that data and quickly turn it into actionable information. As of this writing, the primary disadvantage of using AI is that it is expensive to process the large amounts of data that AI programming requires.

**Advantages**

- Good at detail-oriented jobs;
- Reduced time for data-heavy tasks;
- Delivers consistent results; and
- AI-powered virtual agents are always available.

**Disadvantages**

- Expensive;
- Requires deep technical expertise;
- Limited supply of qualified workers to build AI tools;
- Only knows what it's been shown; and
- Lack of ability to generalize from one task to another. (3500)

[https://www.techtarget.com/searchenterpriseai/definition/AI-Artificial-Intelligence]

*4.2. Answer the following questions:*

1) How does AI work?
2) Why is artificial intelligence important?
3) What are the advantages and disadvantages of artificial intelligence?

*4.3. Translate the following sentences*

1) Игровые движки предоставляют средства разработки, которые могут быть использованы программистами, чтобы упростить их работу. Короче говоря, предоставляют инструменты и функциональные возможности для разработки игры.
2) HTML5 движки пользуются популярностью среди разработчиков игр. Один из таких Turblenz, открытая платформа для разработчиков игр.
3) Он включает в себя все основные функции, которые необходимы, чтобы разработать, интегрировать и монетизировать игру. Кроме того, нет никаких ограничений в использовании, так как он доступен по лицензии MIT.

4) Термин «игровой движок» появился в середине 1990-х в контексте компьютерных игр жанра шутер от первого лица, похожих на популярную в то время Doom.
5) Большинство игровых движков разработано и настроено для того, чтобы запустить определённую игру на определённой платформе.
6) И даже наиболее обобщённые многоплатформенные движки подходят для построения игр определённого жанра, например, шутеров первого лица или гонок.
7) В данном контексте можно более аккуратно сказать, что игровой движок становится не оптимальным при его применении не для той игры или той платформы, для которой разработан.
8) Данный эффект проявляется от того, что программное обеспечение представляет собой набор компромиссов, основанных на тех предположениях, какой должна быть игра.

## TEXTS FOR ADDITIONAL READING

### TEXT 1. Education-oriented languages

1. *Read and translate the text about education-oriented languages*

**BASIC**

BASIC (beginner's all-purpose symbolic instruction code) was designed at Dartmouth College in the mid-1960s by John Kemeny and Thomas Kurtz. It was intended to be easy to learn by novices, particularly non-computer science majors, and to run well on a time-sharing computer with many users. It had simple data structures and notation and it was interpreted: a BASIC program was translated line-by-line and executed as it was translated, which made it easy to locate programming errors.

Its small size and simplicity also made BASIC a popular language for early personal computers. Its recent forms have adopted many of the data and control structures of other contemporary languages, which makes it more powerful but less convenient for beginners.

**Pascal**

About 1970 Niklaus Wirth of Switzerland designed Pascal to teach structured programming, which emphasized the orderly use of conditional and loop control structures without GOTO statements. Although Pascal resembled ALGOL in notation, it provided the ability to define data types with which to organize complex information, a feature beyond the capabilities of ALGOL as well as FORTRAN and COBOL. User-defined data types allowed the programmer to introduce names for complex data, which the language translator could then check for correct usage before running a program.

During the late 1970s and '80s, Pascal was one of the most widely used languages for programming instruction. It was available on nearly all computers, and, because of its familiarity, clarity, and security, it was used for production software as well as for education.

**Logo**

Logo originated in the late 1960s as a simplified LISP dialect for education; Seymour Papert and others used it at MIT to teach mathematical thinking to schoolchildren. It had a more conventional syntax than LISP and featured "turtle graphics," a simple method for generating computer graphics. (The name came from an early

project to program a turtlelike robot.) Turtle graphics used body-centred instructions, in which an object was moved around a screen by commands, such as “left 90” and “forward,” that specified actions relative to the current position and orientation of the object rather than in terms of a fixed framework. Together with recursive routines, this technique made it easy to program intricate and attractive patterns.

**Hypertalk**

Hypertalk was designed as “programming for the rest of us” by Bill Atkinson for Apple’s Macintosh. Using a simple English-like syntax, Hypertalk enabled anyone to combine text, graphics, and audio quickly into “linked stacks” that could be navigated by clicking with a mouse on standard buttons supplied by the program. Hypertalk was particularly popular among educators in the 1980s and early ’90s for classroom multimedia presentations. Although Hypertalk had many features of object-oriented languages (described in the next section), Apple did not develop it for other computer platforms and let it languish; as Apple’s market share declined in the 1990s, a new cross-platform way of displaying multimedia left Hypertalk all but obsolete (*see* the section World Wide Web display languages

2. *Ask five question on the text*

## TEXT 2

*1. Read and translate the text and suggest your title for it*

Object-oriented analysis and design (OOAD) is a software engineering approach that models a system as a group of interacting objects. Each object represents some entity of interest in the system being modeled, and is characterized by its class, its state (data elements), and its behavior.

Various models can be created to show the static structure, dynamic behavior, and run-time deployment of these collaborating objects. There are a number of different notations for representing these models, such as the Unified Modeling Language (UML).

Object-oriented analysis (OOA) applies object-modeling techniques to analyze the functional requirements for a system. Object-oriented design

(OOD) elaborates the analysis models to produce implementation specifications. OOA focuses on what the system does, OOD on how the system does it

An object-oriented system is composed of objects. The behavior of the system results from the collaboration of those objects. Collaboration between objects involves sending messages to each other. Sending a message differs from calling a function in that when a target object receives a message, it itself decides what function to carry out to service that message. The same message may be implemented by many different functions, the one selected depending on the state of the target object.

The implementation of "message sending" varies depending on the architecture of the system being modeled, and the location of the objects being communicated with.

Object-oriented analysis (OOA) looks at the problem domain, with the aim of producing a conceptual model of the information that exists in the area being analyzed. Analysis models do not consider any implementation constraints that might exist, such as concurrency, distribution, persistence, or how the system is to be built. Implementation constraints are dealt during object-oriented design (OOD). Analysis is done before the Design.

The sources for the analysis can be a written requirements statement, a formal vision document, interviews with stakeholders or other interested parties. A system may be divided into multiple domains, representing different business, technological, or other areas of interest, each of which are analyzed separately.

The result of object-oriented analysis is a description of what the system is functionally required to do, in the form of a conceptual model. That will typically be presented as a set of use cases, one or more UML class diagrams, and a number of interaction diagrams. It may also include some kind of user interface mock-up. The purpose of object oriented analysis is to develop a model that describes computer software as it works to satisfy a set of customer defined requirements.

Object-oriented design (OOD) transforms the conceptual model produced in object-oriented analysis to take account of the constraints imposed by the chosen architecture and any non-functional – technological or environmental – constraints, such as transaction throughput, response time, run-time platform, development environment, or programming language.

The concepts in the analysis model are mapped onto implementation classes and interfaces. The result is a model of the solution domain, a detailed description of how the system is to be built. (3200)

[https://www.brainkart.com/article/What-is-OOAD(Object-oriented-analysis-and-design)-_9969/]

*2. Give a two-minute talk about: a) object-oriented analysis, b) object-oriented design*

## TEXT 3. Rendering and its Features

*1. Read and translate the following text about rendering or image synthesis*

Rendering or image synthesis is the process of generating a photorealistic or non-photorealistic image from a 2D or 3D model by means of a computer program. The resulting image is referred to as the render. Multiple models can be defined in a scene file containing objects in a strictly defined language or data structure. The scene file contains geometry, viewpoint, texture, lighting, and shading information describing the virtual scene. The data contained in the scene file is then passed to a rendering program to be processed and output to a digital image or raster graphics image file. The term "rendering" is analogous to the concept of an artist's impression of a scene. The term "rendering" is also used to describe the process of calculating effects in a video editing program to produce the final video output.

Rendering is one of the major sub-topics of 3D computer graphics, and in practice it is always connected to the others. It is the last major step in the graphics pipeline, giving models and animation their final appearance. With the increasing sophistication of computer graphics since the 1970s, it has become a more distinct subject.

Rendering has uses in architecture, video games, simulators, movie and TV visual effects, and design visualization, each employing a different balance of features and techniques. A wide variety of renderers are available for use. Some are integrated into larger modeling and animation packages, some are stand-alone, and some are free open-source projects. On the inside, a renderer is a carefully engineered program based on multiple disciplines, including light physics, visual perception, mathematics, and software development.

Though the technical details of rendering methods vary, the general challenges to overcome in producing a 2D image on a screen from a 3D representation stored in a scene file are handled by the graphics pipeline in a rendering device such as a GPU. A GPU is a purpose-built device that assists a CPU in performing complex rendering calculations. If a scene is to look relatively realistic and predictable under virtual lighting, the rendering software must solve the rendering equation. The rendering equation doesn't account for all lighting phenomena, but instead acts as a general lighting model for computer-generated imagery.

In the case of 3D graphics, scenes can be pre-rendered or generated in real-time. Pre-rendering is a slow, computationally intensive process that is typically used for movie creation, where scenes can be generated ahead of time, while real-time rendering is often done for 3D video games and other applications that must dynamically create scenes. 3D hardware accelerators can improve real-time rendering performance. (2700)

[ https://en.wikipedia.org/wiki/Rendering_(computer_graphics)]

*2. What is the main idea of rendering?*

*3. Write a short summary of the text according to the plan (note that you don't have to use all the phrases).*

План аннотации:
1) название статьи (текста):
– Текст называется….
– Название статьи, которую я прочел…
– Название статьи указывает на то, что в статье говорится о…
2) автор статьи, где и когда она была опубликована:
– Статья написана…. и опубликована в ….
– К сожалению, имя автора неизвестно.
– Текст взят из интернет-источника.
3) основная идея статьи (текста):
– Основная идея текста заключается в том, что…
– Статья посвящена проблеме…
– Цель статьи состоит в том, чтобы…
4) содержание статьи, факты, названия, цифры:
– Автор начинает изложение материала с …

– Автор привлекает внимание читателя к вопросу…
– Особое внимание уделяется…
– Согласно данным, представленным в статье…
– Подчеркивается (отмечается/ утверждается/ доказывается/ опровергается/ ставится под сомнение)….
– Автор приводит подробный анализ…
– Далее описываются факты… (приводятся примеры того как…/ описываются случаи…).
– В заключении автор отмечает (утверждает/ подчеркивает, акцентирует внимание на..)
– Подводя итог, автор приходит к выводу…
5) ваше мнение:
– Статья показалась мне интересной (информативной), т.к. в ней приводятся…
– Считаю необходимым привести некоторые примеры из жизни (из других статей), описывающие те же явления, что и в данном тексте.
– Идеи, предложенные автором, могут быть применены в таких областях, как …
– Некоторые положения кажутся противоречивыми, т.к….
– Статья может быть полезна для тех, кто учится на … (работает в сфере…).

## TEXT 4. The Four Types of Artificial Intelligence

1. *Read and translate the following text about types of AI and make up a short summary of it. Follow the plan given above*


**Reactive Machines**

A reactive machine follows the most basic of AI principles and, as its name implies, is capable of only using its intelligence to perceive and react to the world in front of it. A reactive machine cannot store a memory and as a result cannot rely on past experiences to inform decision making in real-time.

Perceiving the world directly means that reactive machines are designed to complete only a limited number of specialized duties. Intentionally narrowing a reactive machine's worldview is not any sort of cost-cutting measure, however, and instead means that this type of AI will be more trustworthy and reliable — it will react the same way to the same stimuli every time.

A famous example of a reactive machine is Deep Blue, which was designed by IBM in the 1990's as a chess-playing supercomputer and defeated international grandmaster Gary Kasparov in a game. Deep Blue was only capable of identifying the pieces on a chess board and knowing how each moves based on the rules of chess, acknowledging each piece's present position, and determining what the most logical move would be at that moment. The computer was not pursuing future potential moves by its opponent or trying to put its own pieces in better position. Every turn was viewed as its own reality, separate from any other movement that was made beforehand.

Another example of a game-playing reactive machine is Google's AlphaGo. AlphaGo is also incapable of evaluating future moves but relies on its own neural network to evaluate developments of the present game, giving it an edge over Deep Blue in a more complex game. AlphaGo also bested world-class competitors of the game, defeating champion Go player Lee Sedol in 2016.

Though limited in scope and not easily altered, reactive machine artificial intelligence can attain a level of complexity, and offers reliability when created to fulfill repeatable tasks.

**Limited Memory**

Limited memory artificial intelligence has the ability to store previous data and predictions when gathering information and weighing potential decisions — essentially looking into the past for clues on what may come next. Limited memory artificial intelligence is more complex and presents greater possibilities than reactive machines.

Limited memory AI is created when a team continuously trains a model in how to analyze and utilize new data or an AI environment is built so models can be automatically trained and renewed. When utilizing limited memory AI in machine learning, six steps must be followed: Training data must be created, the machine learning model must be created, the model must be able to make predictions, the model must be

able to receive human or environmental feedback, that feedback must be stored as data, and these these steps must be reiterated as a cycle.

**Theory of Mind**

Theory of Mind is just that – theoretical. We have not yet achieved the technological and scientific capabilities necessary to reach this next level of artificial intelligence.

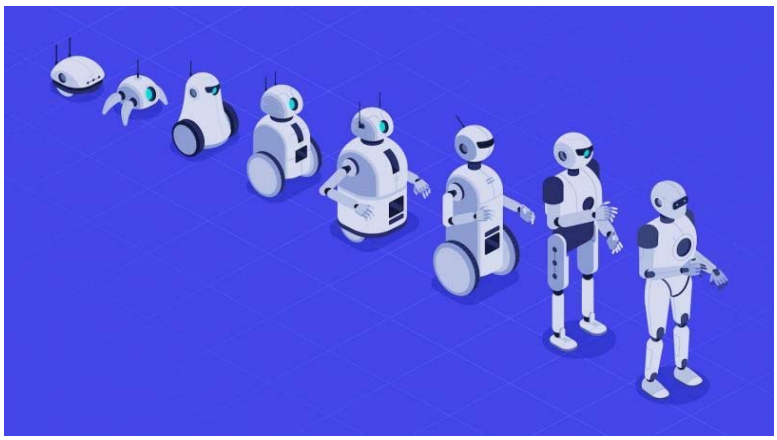
The concept is based on the psychological premise of understanding that other living things have thoughts and emotions that affect the behavior of one's self. In terms of AI machines, this would mean that AI could comprehend how humans, animals and other machines feel and make decisions through self-reflection and determination, and then will utilize that information to make decisions of their own. Essentially, machines would have to be able to grasp and process the concept of "mind," the fluctuations of emotions in decision making and a litany of other psychological concepts in real time, creating a two-way relationship between people and artificial intelligence.

**Self-awareness**

Once Theory of Mind can be established in artificial intelligence, sometime well into the future, the final step will be for AI to become self-aware. This kind of artificial intelligence possesses human-level consciousness and understands its own existence in the world, as well as the presence and emotional state of others. It would be able to understand what others may need based on not just what they communicate to them but how they communicate it.

Self-awareness in artificial intelligence relies both on human researchers understanding the premise of consciousness and then learning how to replicate that so it can be built into machines. (4300)

[https://www.govtech.com/computing/understanding-the-four-types-of-artificial-intelligence.html]



2. *Answer the following questions: Why is AI important? How does AI work?*

3. *Prepare a report on a brief history of Artificial Intelligence*

## TEXT 5. Basic Concepts of Linux

*1. Read and translate the text about Linux system*

Linux looks and feels much like any other UNIX system; indeed, UNIX compatibility has been a major design goal of the Linux project. However, Linux is much younger than most UNIX systems. Its development began in1991, when a Finnish university student, Linus Torvalds, began developing a small but self-contained kernel for the 80386 processor, the first true 32-bitprocessor in Intel's range of PC-compatible CPUs. of arbitrary files (but only read-only memory mapping was implemented in 1.0).

A range of extra hardware support was included in this release. Although still restricted to the Intel PC platform, hardware support had grown to include floppy-disk and CD-ROM devices, as well as sound cards, a range of mice, and international keyboards. Floating-point emulation was provided in the kernel for 80386 users who had no 80387 math coprocessor. System V UNIX-style interprocess communication (IPC), including shared memory, semaphores, and message queues, was implemented.

Linux kernel is composed entirely of code written from scratch specifically for the Linux project, much of the supporting software that makes up the Linux system is not exclusive to Linux but is common to a number of UNIX-like operating systems. In particular, Linux uses many tools developed as part of Berkeley's BSD operating system, MIT's X Window System, and the Free Software Foundation's GNU project.

This sharing of tools has worked in both directions. The main system libraries of Linux were originated by the GNU project, but the Linux community greatly improved the libraries by addressing omissions, inefficiencies, and bugs. Other components, such as the GNU C compiler (gcc), were already of sufficiently high quality to be used directly in Linux. The network administration tools under Linux were derived from

code first developed for 4.3 BSD, but more recent BSD derivatives, such as FreeBSD, have borrowed code from Linux in return. Examples of this sharing include the Intel floating-point-emulation math library and the PC sound-hardware device drivers.

The Linux system as a whole is maintained by a loose network of developers collaborating over the Internet, with small groups or individuals having responsibility for maintaining the integrity of specific components.

A small number of public Internet file-transfer-protocol (FTP) archive sites act as de facto standard repositories for these components. The File System Hierarchy Standard document is also maintained by the Linux community as a means of ensuring compatibility across the various system components.

This standard specifies the overall layout of a standard Linux file system; it determines under which directory names configuration files, libraries, system binaries, and run-time data files should be stored.

**Linux Distributions**

In theory, anybody can install a Linux system by fetching the latest revisions of the necessary system components from the FTP sites and compiling them. In Linux's early days, this is precisely what a Linux user had to do. As Linux has matured, however, various individuals and groups have attempted to make this job less painful by providing standard, precompiled sets of packages for easy installation.

These collections, or distributions, include much more than just the basic Linux system. They typically include extra system-installation and management utilities, as well as precompiled and ready-to-install packages of many of the common UNIX tools, such as news servers, web browsers, text-processing and editing tools, and even games.

The first distributions managed these packages by simply providing a means of unpacking all the files into the appropriate places. One of the important contributions of modern distributions, however, is advanced package management. Today's Linux distributions include a package-tracking database that allows packages to be installed, upgraded, or removed painlessly. (3900)

[Abridged from https://www.brainkart.com/article/Linux-System---Basic-Concepts_9864/]

2. *Test yourself. If you don't know the answers for some questions, you can find them in Internet later for homework*

**What do you know about Linux?**

1. Linux development began in …?
   a) 1978
   b) 1991
   c) 2001
2. How many basic elements or components of Linux are there?
   a) 5
   b) 3
   c) 6
3. What is the main component of Linux OS?
   a) hardware
   b) kernel
   c) shell
4. What is Linux shell?
   a) a user interface present between user and kernel.
   b) a type of process that can be in a number of different states.
   c) a resource manager that acts as a bridge between hardware and software.

5) What is CLI?
   a) a command-line program that usually accepts text as input to execute or run functions of the operating system.
   b) a human-computer interface that allows users to interact with electronic devices through graphical icons and visual indicators.
6. The Linux system as a whole is maintained …
   a) by long-term public support.
   b) by a loose network of developers collaborating over the Internet.
7. Name command that is used to remove files.
   a) delete
   b) dr
   c) rm
   d) None of the above
8. Name the person who developed Linux.
   a) Linus Torvalds
   b) Dennis Ritchie

c) Prof. Andrew S. Tannenbaum
d) None of the above

9. Maximum size (in bytes) of the filename in Linux can be?
   a) 64 bytes
   b) 32 bytes
   c) 128 bytes
   d) 255 bytes
10. What are the characteristics of Linux OS?
   a) It is an open-source and free-to-use.
   b) It is not open source and is not free to use.
   c) It has very low hardware requirements and facilitates powerful support for networking.
   d) None of the above.

*Answer key to the task: 1.b); 2.a); 3.b); 4.a); 5.a); 6.b); 7.b); 8.d); 9.c); 10.a).*

## TEXT 6. A Developer's Guide to Communicating with Clients

*1. Read and translate the following text about a successful conversation between a software developer and a client*

If you are a developer that does not always feel comfortable to communicate with the clients and all you want to do is just code and more code, then this article is created with you in my mind.

**1) Informing clients that you have received a task**

The first thing that crossed my mind as important is noting clients you have received a task that has been assigned to you. Maybe some of you might think how this is a no brainer but this little step is very important in everyday communication.

**Situation A)** The client assigns a task. A developer is available to look at it.

*Solution:* You should inform the client that you will look at it right away and come back to him with the more informed feedback or you can take a quick look and in the first response provide some questions to better understand what needs to be done.

**Situation B)** The client assigns a task. A developer is not free to look at it immediately.

*Solution:* Thank them for providing information regarding the task and inform them that you will start with it as soon as possible. This way you have let them know that you are not available at the moment, but as soon as you will be, you will work on it.

**Do not:**

- Ignore the task for a day or couple of days while you don't become available for resolving process.
- Share additional information of why you cannot take this task right away. You will just complicate things that shouldn't be complicated.

**2) Frequently updating the client about the task progress**

In most cases, a developer would inform the client that he has started with the task and he would not give any updates until he's not finished. If we are talking about a task that needs a day or two to be completed, it is definitely right approach, as we don't want to burden the client with too much updates. However, more time-consuming tasks that need at least 7-10 days to be fulfilled should be given updates from time to time.

The status updates shouldn't be:

- Too big or too short
- Too often (i.e. if you are working on it for 7 days, 3 updates are enough)

*Solution:* You can start with it now and at the end of the day update the client about the progress. At least you have done something.

*Solution:* You should be honest with what is going on. You were busy with the emergency task(s) and you are starting with this task now. You will apologies and prioritise this task. Maybe the client will not be satisfied with these explanations at the moment, but don't be afraid. He will appreciate your honesty after some time.

**Do not:**

- Lie to the client that you have done something which in reality you have not even started yet. Remember that lies always come out in the end.

**3) Avoiding writing too long posts**

You should remember that the client doesn't have the time to read long posts, especially if something can be said in short and without complications. If you have written too long post, it might seem that you

didn’t understand the task well. So always try to simplify things, if you can.

**4) Notifying the client on the time about your vacations and obligations**

Inform the client about your planned vacation in advance so that he can plan tasks towards you. Also, don’t take any complex tasks knowing that in the middle of their progress you will be gone and not available to check upon them. When you are about to go on a vacation, name another developer to be on the service if something is urgent.

In the situation when you got sick or have some unexpected things to deal with, ask your teammates or team leader to write a quick note that you will be unavailable for a specific period of time on projects you work. With these approaches you will receive a lot of respect from the client.

But what should you do if the project is progressing slowly because of the client and now you have a 2 weeks delay from release? This is something that definitely requires individual approach and a deeper analysis of the situation. If you are very indecisive about how to react, the following question will maybe help to make a right decision:

- How important is this project for your company?
- Will this project really be launched in 2 weeks or maybe in 4-6 weeks?
- Can someone fulfill your position while you are away?

If you are willing to give it a try and move your vacation towards the project, you should let the client know that you’ve decided to stay in order to get things done, but make a clear deadline for how long you will be available. This way you will not feel guilty if another deadline is passed.

If you have mastered these skills in your everyday work then clients probably enjoy working with you.

**What to do in the situation when the client is angry and rude?**

From my experience, it is important to not be emotionally involved with a message and to respond cool-headed. The tone of the message should be calm and with respect. Don’t be afraid even if it is your mistake. Examine the situation – why did this happen and can it be fixed. Try to propose a solution. If you are not sure how to handle this situation, ask for your team leader to jump in.The fact is that good communication is sure evidence of a future long-term relationship. (4190)

[Abridged from https://inchoo.net/life-at-inchoo/developers-guide-communicating-clients/ by Antonija Tadic]

3. *Imagine that you're in one of the situations mentioned above. Dramatize a dialogue with your group mates in class on the topic 'communicating with a client'*

## TEXT 7. Polymorphism, Encapsulation, and Inheritance Work Together

*1. Read and translate the following text*

When properly applied, polymorphism, encapsulation, and inheritance combine to produce a programming environment that supports the development of far more robust and scaleable programs than does the process-oriented model. A well-designed hierarchy of classes is the basis for reusing the code in which you have invested time and effort developing and testing. Encapsulation allows you to migrate your implementations over time without breaking the code that depends on the public interface of your classes. Polymorphism allows you to create clean, sensible, readable, and resilient code.

Of the two real-world examples, the automobile more completely illustrates the power of object-oriented design. Dogs are fun to think about from an inheritance standpoint, but cars are more like programs. All drivers rely on inheritance to drive different types (subclasses) of vehicles. Whether the vehicle is a school bus, a Mercedes sedan, a Porsche, or the family minivan, drivers can all more or less find and operate the steering wheel, the brakes, and the accelerator. After a bit of gear grinding, most people can even manage the difference between a stick shift and an automatic, because they fundamentally understand their common superclass, the transmission.

People interface with encapsulated features on cars all the time. The brake and gas pedals hide an incredible array of complexity with an interface so simple you can operate them with your feet! The implementation of the engine, the style of brakes, and the size of the tires have no effect on how you interface with the class definition of the pedals.

The final attribute, polymorphism, is clearly reflected in the ability of car manufacturers to offer a wide array of options on basically the same vehicle. For example, you can get an antilock braking system or traditional brakes, power or rack-and-pinion steering, and 4-, 6-, or 8-cylinder engines. Either way, you will still press the brake pedal to stop, turn the

steering wheel to change direction, and press the accelerator when you want to move. The same interface can be used to control a number of different implementations.

As you can see, it is through the application of encapsulation, inheritance, and polymorphism that the individual parts are transformed into the object known as a car. The same is also true of computer programs. By the application of object-oriented principles, the various parts of a complex program can be brought together to form a cohesive, robust, maintainable whole.

As mentioned at the start of this section, every Java program is object-oriented. Or, put more precisely, every Java program involves encapsulation, inheritance, and polymorphism. Although the short example programs shown in the rest of this chapter and in the next few chapters may not seem to exhibit all of these features, they are nevertheless present. As you will see, many of the features supplied by Java are part of its built-in class libraries, which do make extensive use of encapsulation, inheritance, and polymorphism.

[https://www.brainkart.com/article/Object-Oriented-Programming_10384/]

*2. Ask five questions on the text about polymorphism, encapsulation, and inheritance*

## TEXT 8. Role of Process in Software Quality

The need for software products of high quality has pressured those in the profession to identify and quantify quality factors such as usability, testability, maintainability, and reliability, and to identify engineering practices that support the production of quality products having these favorable attributes. Among the practices identified that contribute to the development of high-quality software are project planning, requirements management, development of formal specifications, structured design with use of information hiding and encapsulation, design and code reuse, inspections and reviews, product and process measures, education and training of software professionals, development and application of CASE tools, use of effective testing techniques, and integration of testing activities into the entire life cycle. In addition to identifying these individual best technical and managerial practices, software researchers realized that it was important to integrate them within the context of a high-quality software development process.

What is a process in software development?

- Process, in the software engineering domain, is the set of methods, practices, standards, documents, activities, policies, and procedures that software engineers use to develop and maintain a software system and its associated artifacts, such as project and test plans, design documents, code, and manuals.

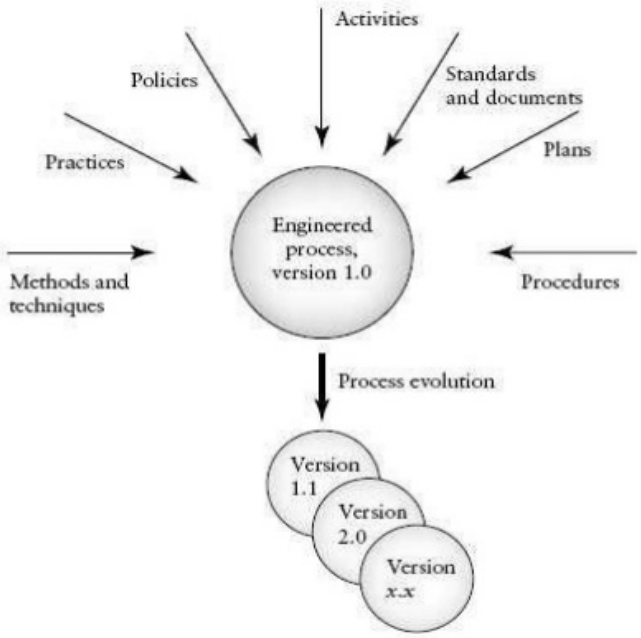



Fig. 13. Components of an engineering process

It also was clear that adding individual practices to an existing software development process in an ad hoc way was not satisfactory. The software development process, like most engineering artifacts, must be engineered. That is, it must be designed, implemented, evaluated, and maintained. As in other engineering disciplines, a software development process must evolve in a consistent and predictable manner, and the best technical and managerial practices must be integrated in a systematic way. These models allow an organization to evaluate its current software process and to capture an understanding of its state. Strong support for incremental process improvement is provided by the models, consistent with historical process evolution and the application of quality principles. The models have received much attention from industry, and resources have been invested in process improvement efforts with many successes recorded.

All the software process improvement models that have had wide acceptance in industry are high-level models, in the sense that they focus

on the software process as a whole and do not offer adequate support to evaluate and improve specific software development sub processes such as design and testing. Most software engineers would agree that testing is a vital component of a quality software process, and is one of the most challenging and costly activities carried out during software development and maintenance. (2900)

[https://www.brainkart.com/article/Role-of-process-in-software-quality_9136/]

*2. Explain in your own words the following terms 'process', 'software development process', 'models' and 'techniques'*

## TEXT 9. Mobile Computing Applications

*1. Read and translate the following text about mobile computing*

*For Estate Agents*

Estate agents can work either at home or out in the field. With mobile computers they can be more productive. They can obtain current real estate information by accessing multiple listing services, which they can do from home, office or car when out with clients. They can provide clients with immediate feedback regarding specific homes or neighbourhoods, and with faster loan approvals, since applications can be submitted on the spot. Therefore, mobile computers allow them to devote more time to clients.

*Emergency Services*

Ability to receive information on the move is vital where the emergency services are involved. Information regarding the address, type and other details of an incident can be dispatched quickly, via a CDPD system using mobile computers, to one or several appropriate mobile units which are in the vicinity of the incident. Here the reliability and security implemented in the CDPD system would be of great advantage.

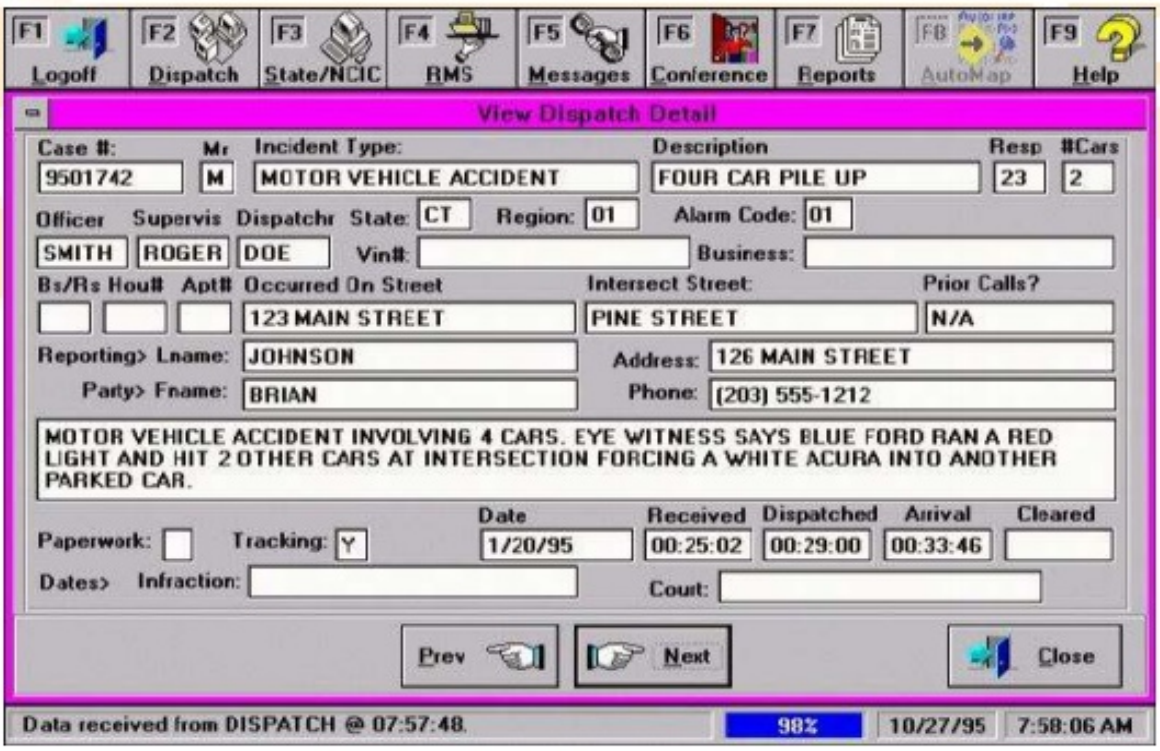



Fig. 14. Police incident information screen

*In courts*

Defence counsels can take mobile computers in court. When the opposing counsel references a case which they are not familiar, they can use the computer to get direct, real-time access to on-line legal database services, where they can gather information on the case and related precedents. Therefore mobile computers allow immediate access to a wealth of information, making people better informed and prepared.

*In companies*

Managers can use mobile computers in, say, critical presentations to major customers. They can access the latest market share information. At a small recess, they can revise the presentation to take advantage of this information. They can communicate with the office about possible new offers and call meetings for discussing responds to the new proposals. Therefore, mobile computers can leverage competitive advantages.

*Stock Information Collation/Control*

In environments where access to stock is very limited i.e.: factory warehouses. The use of small portable electronic databases accessed via a mobile computer would be ideal. Data collated could be directly written to a central database, via a CDPD network, which holds all stock information hence the need for transfer of data to the central computer at a later date is not necessary. This ensures that from the time that a stock count is

completed, there is no inconsistency between the data input on the portable computers and the central database.

*Credit Card Verification*

At Point of Sale (POS) terminals in shops and supermarkets, when customers use credit cards for transactions, the intercommunication required between the bank central computer and the POS terminal, in order to effect verification of the card usage, can take place quickly and securely over cellular channels using a mobile computer unit. This can speed up the transaction process and relieve congestion at the POS terminals.

*Taxi/Truck Dispatch*

Using the idea of a centrally controlled dispatcher with several mobile units (taxis), mobile computing allows the taxis to be given full details of the dispatched job as well as allowing the taxis to communicate information about their whereabouts back to the central dispatch office. This system is also extremely useful in secure deliveries ie: Securicor. This allows a central computer to be able to track and receive status information from all of its mobile secure delivery vans. Again, the security and reliability properties of the CDPD system shine through.

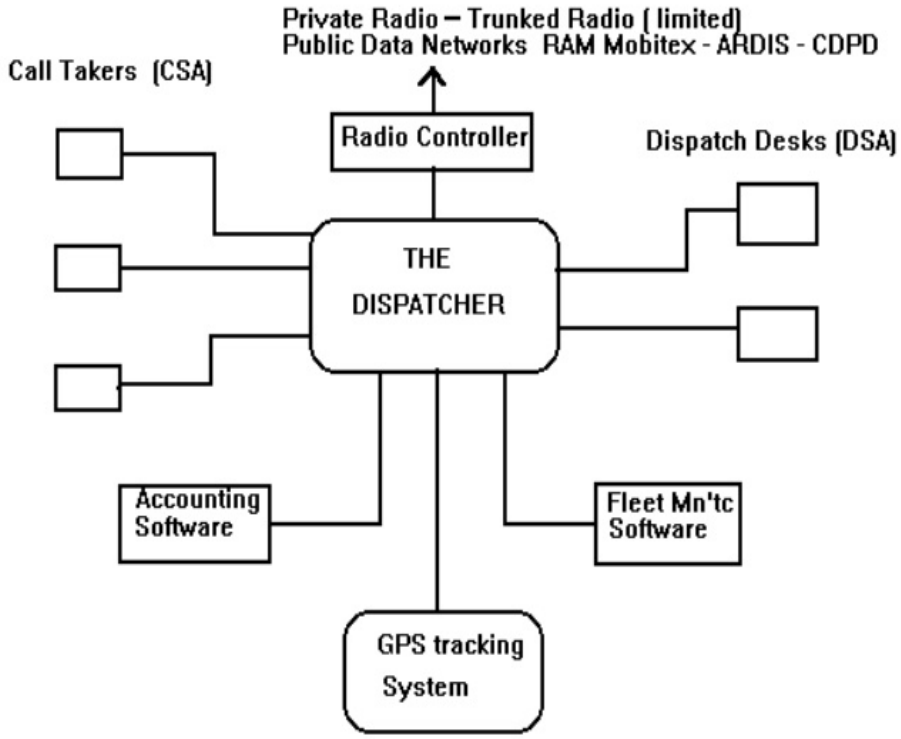



Fig. 15. Taxi Dispatch Network

*Electronic Mail/Paging*

Usage of a mobile unit to send and read emails is a very useful asset for any business individual, as it allows him/her to keep in touch with any colleagues as well as any urgent developments that may affect their work. Access to the Internet, using mobile computing technology, allows the individual to have vast arrays of knowledge at his/her fingertips. Paging is also achievable here, giving even more intercommunication capability between individuals, using a single mobile computer device. (4000)

[https://www.brainkart.com/article/Mobile-Computing-Applications_9874/]

3. *Explain in your own words each type of the mobile applications described above*

# COMPUTER TERMS GLOSSARY

**abstraction.** The separation of the logical properties of data or function from its implementation in a computer program. See: encapsulation, information hiding, software engineering.

**access time.** The time interval between the instant at which a call for data is initiated and the instant at which the delivery of the data is completed.

**accuracy.** ) (1) A qualitative assessment of correctness or freedom from error. (2) A quantitative measure of the magnitude of error. Contrast with precision. (3) The measure of an instrument's capability to approach a true or absolute value. It is a function of precision and bias.

**algorithm.** (1) A finite set of well-defined rules for the solution of a problem in a finite number of steps. (2) Any sequence of operations for performing a specific task.

**analog device.** A device that operates with variables represented by continuously measured quantities such as pressures, resistances, rotations, temperatures, and voltages.

**application software.** Software designed to fill specific needs of a user; for example, software for navigation, payroll, or process control. Contrast with support software; system software.

**architectural design.** (1) The process of defining a collection of hardware and software components and their interfaces to establish the framework for the development of a computer system. (2) The result of the process in (1).

**archival database.** An historical copy of a database saved at a significant point in time for use in recovery or restoration of the database.

**archive file.** A file that is part of a collection of files set aside for later research or verification, for security purposes, for historical or legal purposes, or for backup.

**arithmetic logic unit.** The [high speed] circuits within the CPU which are responsible for performing the arithmetic and logical operations of a computer.

**array.** An n-dimensional ordered set of data items identified by a single name and one or more indices, so that each element of the set is individually addressable; e.g., a matrix, table, or vector.

**assembler.** A computer program that translates programs [source code files] written in assembly language into their machine language equivalents [object code files]. Contrast with compiler, interpreter.

**assembly language.** A low level programming language, that corresponds closely to the instruction set of a given computer, allows symbolic naming of operations and

addresses, and usually results in a one-to-one translation of program instructions [mnemonics] into machine instructions. See: low-level language.

**asynchronous.** Occurring without a regular time relationship, i.e., timing independent.

**BIOS.** basic input/output system.

**BASIC.** An acronym for Beginners All-purpose Symbolic Instruction Code, a high-level programming language intended to facilitate learning to program in an interactive environment.

**basic input/output system.** Firmware that activates peripheral devices in a PC. Includes routines for the keyboard, screen, disk, parallel port and serial port, and for internal services such as time and date. It accepts requests from the device drivers in the operating system as well from application programs. It also contains autostart functions that test the system on startup and prepare the computer for operation. It loads the operating system and passes control to it.

**batch processing.** Execution of programs serially with no interactive processing. Contrast with real time processing.

**bias.** A measure of how closely the mean value in a series of replicate measurements approaches the true value. See: accuracy, precision, calibration.

**binary.** The base two number system. Permissible digits are "0" and "1".

**bit.** A contraction of the term binary digit. The bit is the basic unit of digital data. It may be in one of two states, logic 1 or logic 0.

**block.** (1) A string of records, words, or characters that for technical or logical purposes are treated as a unity. (2) A collection of contiguous records that are recorded as a unit, and the units are separated by interblock gaps. (3) In programming languages, a subdivision of a program that serves to group related statements, delimit routines, specify storage allocation, delineate the applicability of labels, or segment parts of the program for other purposes. In FORTRAN, a block may be a sequence of statements; in COBOL, it may be a physical record.

**block diagram.** A diagram of a system, instrument or computer, in which the principal parts are represented by suitably annotated geometrical figures to show both the basic functions of the parts and the functional relationships between them.

**block length.** (1) The number of records, words or characters in a block. (2) A measure of the size of a block, usually specified in units such as records, words, computer words, or characters.

**boot.** (1) To initialize a computer system by clearing memory and reloading the operating system. (2) To cause a computer system to reach a known beginning state.

**bootstrap.** A short computer program that is permanently resident or easily loaded into a computer and whose execution brings a larger program, such an operating system or its loader, into memory.

**buffer.** A device or storage area [memory] used to store data temporarily to compensate for differences in rates of data flow, time of occurrence of events, or amounts of data that can be handled by the devices or processes involved in the transfer or use of the data.

**bug.** A fault in a program which causes the program to perform in an unintended or unanticipated manner. See: anomaly, defect, error, exception, fault.

**bus.** A common pathway along which data and control signals travel between different hardware devices within a computer system.

**byte.** A sequence of adjacent bits, usually eight, operated on as a unit.

**CAM.** computer aided manufacturing. The automation of manufacturing systems and techniques, including the use of computers to communicate work instructions to automate machinery for the handling of the processing [numerical control, process control, robotics, material requirements planning] needed to produce a workpiece.

**CASE.** computer aided software engineering. An automated system for the support of software development including an integrated tool set, i.e., programs, which facilitate the accomplishment of software engineering methods and tasks such as project planning and estimation, system and software requirements analysis, design of data structure, program architecture and algorithm procedure, coding, testing and maintenance.

**COTS.** configurable, off-the-shelf software. Application software, sometimes general purpose, written for a variety of industries or users in a manner that permits users to modify the program to meet their individual needs.

**CPU.** central processing unit. The unit of a computer that includes the circuits controlling the interpretation of program instructions and their execution. The CPU controls the entire computer. It receives and sends data through input-output channels, retrieves data and programs from memory, and conducts mathematical and logical functions of a program.

**C.** A general purpose high-level programming language. Created for use in the development of computer operating systems software. It strives to combine the power of assembly language with the ease of a high-level language.

**C++.** An object-oriented high-level programming language.

**change tracker.** A software tool which documents all changes made to a program.

**client-server.** A term used in a broad sense to describe the relationship between the receiver and the provider of a service. In the world of microcomputers, the term

client-server describes a networked system where front-end applications, as the client, make service requests upon another networked system.

**COBOL.** Acronym for COmmon Business Oriented Language. A high-level programming language intended for use in the solution of problems in business data processing.

**code audit.** An independent review of source code by a person, team, or tool to verify compliance with software design documentation and programming standards.

**code inspection.** A manual [formal] testing [error detection] technique where the programmer reads source code, statement by statement, to a group who ask questions analyzing the program logic, analyzing the code with respect to a checklist of historically common programming errors, and analyzing its compliance with coding standards.

**coding.** (1) In software engineering, the process of expressing a computer program in a programming language. (2) The transforming of logic and data from design specifications (design descriptions) into a programming language.

**comparator.** A software tool that compares two computer programs, files, or sets of data to identify commonalities or differences. Typical objects of comparison are similar versions of source code, object code, data base files, or test results.

**compatibility.** The capability of a functional unit to meet the requirements of a specified interface.

**compilation.** Translating a program expressed in a problem-oriented language or a procedure oriented language into object code.

**compiler.** (1) A computer program that translates programs expressed in a high-level language into their machine language equivalents. (2) The compiler takes the finished source code listing as input and outputs the machine code instructions that the computer must have to execute the program.

**computer.** A functional unit that can perform substantial computations, including numerous arithmetic operations, or logic operations, without human intervention during a run.

**computer aided design.** The use of computers to design products. CAD systems are high speed workstations or personal computers using CAD software and input devices such as graphic tablets and scanners to model and simulate the use of proposed products.

**computer language.** A language designed to enable humans to communicate with computers. See: programming language.

**computer system.** A functional unit, consisting of one or more computers and associated peripheral input and output devices, and associated software.

**configuration.** (1) The arrangement of a computer system or component as defined by the number, nature, and interconnections of its constituent parts. (2) In configuration management, the functional and physical characteristics of hardware or software as set forth in technical documentation or achieved in a product.

**control flow.** In programming languages, an abstraction of all possible paths that an execution sequence may take through a program.

**controller.** Hardware that controls peripheral devices such as a disk or display screen. It performs the physical data transfers between main memory and the peripheral device.

**cross-compiler.** A compiler that executes on one computer but generates assembly code or object code for a different computer.

**DOS.** disk operating system.

**data.** Representations of facts, concepts, or instructions in a manner suitable for communication, interpretation, or processing by humans or by automated means.

**data analysis.** Evaluation of the data structure and usage in the code to ensure each is defined and used properly by the program. Usually performed in conjunction with logic analysis.

**data bus.** A bus used to communicate data internally and externally to and from a processing unit or a storage device.

**data element.** (1) A named unit of data that, in some contexts, is considered indivisible and in other contexts may consist of data items. (2) A named identifier of each of the entities and their attributes that are represented in a database.

**data flow analysis.** A software V&V task to ensure that the input and output data and their formats are properly defined, and that the data flows are correct.

**data integrity.** The degree to which a collection of data is complete, consistent, and accurate.

**data set.** A collection of related records. Syn: file.

**data structure.** A physical or logical relationship among data elements, designed to support specific data manipulation functions.

**data validation.** A process used to determine if data are inaccurate, incomplete, or unreasonable. The process may include format checks, completeness checks, check key tests, reasonableness checks and limit checks.

**database.** A collection of interrelated data, often with controlled redundancy, organized according to a schema to serve one or more applications. The data are stored so that they can be used by different programs without concern for the data structure or organization.

**dead code.** Program code statements which can never execute during program operation. Such code can result from poor coding style, or can be an artifact of previous versions or debugging efforts. Dead code can be confusing, and is a potential source of erroneous software changes.

**debugging.** Determining the exact nature and location of a program error, and fixing the error.

**design.** The process of defining the architecture, components, interfaces, and other characteristics of a system or component.

**design phase.** The period of time in the software life cycle during which the designs for architecture, software components, interfaces, and data are created, documented, and verified to satisfy requirements.

**design requirement.** A requirement that specifies or constrains the design of a system or system component.

**design standards.** Standards that describe the characteristics of a design or a design description of data or program components.

**developer.** A person, or group, that designs and/or builds and/or documents and/or configures the hardware and/or software of computerized systems.

**development methodology.** A systematic approach to software creation that defines development phases and specifies the activities, products, verification procedures, and completion criteria for each phase.

**different software system analysis.** Analysis of the allocation of software requirements to separate computer systems to reduce integration and interface errors related to safety. Performed when more than one software system is being integrated.

**digital.** Pertaining to data [signals] in the form of discrete integral values.

**disk operating system.** An operating system program; e.g., DR-DOS from Digital Research, MS-DOS from Microsoft Corp., OS/2 from IBM, PC-DOS from IBM, System-7 from Apple.

**driver.** A program that links a peripheral device or internal function to the operating system, and providing for activation of all device functions.

**dynamic analysis.** Analysis that is performed by executing the program code.

**editing.** Modifying the content of the input by inserting, deleting, or moving characters, numbers, or data.

**embedded software.** Software that is part of a larger system and performs some of the requirements of that system; e.g., software used in an aircraft or rapid transit system. Such software does not provide an interface with the user. See: firmware.

**encapsulation.** A software development technique that consists of isolating a system function or a set of data and the operations on those data within a module and providing precise specifications for the module.

**end user.** A person, device, program, or computer system that uses an information system for the purpose of data processing in information exchange.

**error.** A discrepancy between a computed, observed, or measured value or condition and the true, specified, or theoretically correct value or condition.

**error detection.** Techniques used to identify errors in data transfers. See: check summation, cyclic redundancy check [CRC], parity check, longitudinal redundancy.

**execution trace.** A record of the sequence of instructions executed during the execution of a computer program. Often takes the form of a list of code labels encountered as the program executes.

**FTP.** file transfer protocol.

**failure.** The inability of a system or component to perform its required functions within specified performance requirements.

**failure analysis.** Determining the exact nature and location of a program error in order to fix the error, to identify and fix other similar errors, and to initiate corrective action to prevent future occurrences of this type of error.

**fault.** An incorrect step, process, or data definition in a computer program which causes the program to perform in an unintended or unanticipated manner.

**file.** (1) A set of related records treated as a unit; e.g., in stock control, a file could consists of a set of invoices. (2) The largest unit of storage structure that consists of a named collection of all occurrences in a database of records of a particular record type.

**file maintenance.** The activity of keeping a file up to date by adding, changing, or deleting data.

**file transfer protocol.** (1) Communications protocol that can transmit binary and ASCII data files without loss of data. See: Kermit, Xmodem, Ymodem, Zmodem. (2) TCP/IP protocol that is used to log onto the network, list directories, and copy files. It can also translate between ASCII and EBCDIC.

**firmware.** The combination of a hardware device; e.g., an IC; and computer instructions and data that reside as read only software on that device. Such software cannot be modified by the computer during processing.

**FORTRAN.** An acronym for FORmula TRANslator, the first widely used high-level programming language. Intended primarily for use in solving technical problems in mathematics, engineering, and science.

**functional analysis.** Verifies that each safety-critical software requirement is covered and that an appropriate criticality level is assigned to each software element.

**functional configuration audit.** An audit conducted to verify that the development of a configuration item has been completed satisfactorily, that the item has achieved the performance and functional characteristics specified in the functional or allocated configuration identification, and that its operational and support documents are complete and satisfactory.

**gigabyte.** Approximately one billion bytes; precisely 230 or 1,073,741,824 bytes.

**graph.** A diagram or other representation consisting of a finite set of nodes and internode connections called edges or arcs.

**graphic software specifications.** Documents such as charts, diagrams, graphs which depict program structure, states of data, control, transaction flow, HIPO, and cause-effect relationships; and tables including truth, decision, event, state-transition, module interface, exception conditions/responses necessary to establish design integrity.

**hardware.** Physical equipment, as opposed to programs, procedures, rules, and associated documentation.

**high-level language.** A programming language which requires little knowledge of the target computer, can be translated into several different machine languages, allows symbolic naming of operations and addresses, provides features designed to facilitate expression of data structures and program logic, and usually results in several machine instructions for each program statement. Examples are PL/1, COBOL, BASIC, FORTRAN, Ada, Pascal, and "C".

**I/0.** input/output.

**ISO.** International Organization for Standardization.

**implementation.** The process of translating a design into hardware components, software components, or both.

**implementation phase.** The period of time in the software life cycle during which a software product is created from design documentation and debugged.

**implementation requirement.** A requirement that specifies or constrains the coding or construction of a system or system component.

**incremental development.** A software development technique in which requirements definition, design, implementation, and testing occur in an overlapping, iterative [rather than sequential] manner, resulting in incremental completion of the overall software product.

**information hiding.** The practice of "hiding" the details of a function or structure, making them inaccessible to other parts of the program.

**input/output.** Each microprocessor and each computer needs a way to communicate with the outside world in order to get the data needed for its programs and in order to communicate the results of its data manipulations. This is accomplished through I/0 ports and devices.

**installation.** The phase in the system life cycle that includes assembly and testing of the hardware and software of a computerized system. Installation includes installing a new computer system, new software or hardware, or otherwise modifying the current system.

**installation and checkout phase.** The period of time in the software life cycle during which a software product is integrated into its operational environment and tested in this environment to ensure that it performs as required.

**instruction.** (1) program statement that causes a computer to perform a particular operation or set of operations. (2) In a programming language, a meaningful expression that specifies one operation and identifies its operands, if any.

**integrated circuit (IC).** Small wafers of semiconductor material [silicon] etched or printed with extremely small electronic switching circuits. Syn: chip.

**interface.** A shared boundary between two functional units, defined by functional characteristics, common physical interconnection characteristics, signal characteristics, and other characteristics, as appropriate. The concept involves the specification of the connection of two devices having different functions.

**interpret.** To translate and execute each statement or construct of a computer program before translating and executing the next.

**interpreter.** A computer program that translates and executes each statement or construct of a computer program before translating and executing the next. The interpreter must be resident in the computer each time a program [source code file] written in an interpreted language is executed.

**invalid inputs.** Test data that lie outside the domain of the function the program represents.

**KB.** kilobyte. Approximately one thousand bytes. This symbol is used to describe the size of computer memory or disk storage space. Because computers use a binary number system, a kilobyte is precisely 210 or 1024 bytes.

**key element.** An individual step in an critical control point of the manufacturing process.

**life cycle methodology.** The use of any one of several structured methods to plan, design, implement, test. and operate a system from its conception to the termination of its use. See: waterfall model.

**linkage editor.** A computer program that creates a single load module from two or more independently translated object modules or load modules by resolving cross references among the modules and, possibly, by relocating elements. Syn: link editor

**loader.** A program which copies other [object] programs from auxiliary [external] memory to main [internal] memory prior to its execution.

**low-level language.** The advantage of assembly language is that it provides bit-level control of the processor allowing tuning of the program for optimal speed and performance. For time critical operations, assembly language may be necessary in order to generate code which executes fast enough for the required operations. The disadvantage of assembly language is the high-level of complexity and detail required in the programming. This makes the source code harder to understand, thus increasing the chance of introducing errors during program development and maintenance.

**Mb.** megabit. Approximately one million bits. Precisely 1024 K bits, 220 bits, or 1,048,576 bits.

**MB.** megabyte. Approximately one million bytes. Precisely 1024 K Bytes, 220 bytes, or 1,048,576 bytes. See: kilobyte.

**MIPS.** million instructions per second.

**machine code.** Computer instructions and definitions expressed in a form [binary code] that can be recognized by the CPU of a computer. All source code, regardless of the language in which it was programmed, is eventually converted to machine code. Syn: object code.

**macroinstruction.** A source code instruction that is replaced by a predefined sequence of source instructions, usually in the same language as the rest of the program and usually during assembly or compilation.

**main memory.** A non-moving storage device utilizing one of a number of types of electronic circuitry to store information.

**main program.** A software component that is called by the operating system of a computer and that usually calls other software components. See: routine, subprogram.

**maintenance.** Activities such as adjusting, cleaning, modifying, overhauling equipment to assure performance in accordance with requirements. Maintenance to a software system includes correcting software errors, adapting software to a new environment, or making enhancements to software.

**measure.** A quantitative assessment of the degree to which a software product or process possesses a given attribute.

**megahertz.** A unit of frequency equal to one million cycles per second.

**memory.** Any device or recording medium into which binary data can be stored and held, and from which the entire original data can be retrieved.

**menu.** A computer display listing a number of options; e.g., functions, from which the operator may select one. Sometimes used to denote a list of programs.

**metric, software quality.** A quantitative measure of the degree to which software possesses a given attribute which affects its quality.

**microcode.** Permanent memory that holds the elementary circuit operations a computer must perform for each instruction in its instruction set.

**microprocessor.** A CPU existing on a single IC. Frequently synonymous with a microcomputer.

**modeling.** Construction of programs used to model the effects of a postulated environment for investigating the dimensions of a problem for the effects of algorithmic processes on responsive targets.

**module.** (1) In programming languages, a self- contained subdivision of a program that may be separately compiled. (2) A discrete set of instructions, usually processed as a unit, by an assembler, a compiler, a linkage editor, or similar routine or subroutine. (3) A packaged functional hardware unit suitable for use with other components. See: unit.

**multi-processing.** A mode of operation in which two or more processes [programs] are executed concurrently [simultaneously] by separate CPUs that have access to a common main memory. Contrast with multi-programming.

**multi-programming.** A mode of operation in which two or more programs are executed in an interleaved manner by a single CPU.

**multi-tasking.** A mode of operation in which two or more tasks are executed in an interleaved manner.

**network.** (1) An arrangement of nodes and interconnecting branches. (2) A system [transmission channels and supporting hardware and software] that connects several remotely located computers via telecommunications.

**OOP.** object oriented programming. A technology for writing programs that are made up of self-sufficient modules that contain all of the information needed to manipulate a given data structure. The modules are created in class hierarchies so that the code or methods of a class can be passed to other modules. New object modules can be easily created by inheriting the characteristics of existing classes.

**object.** In object oriented programming, A self contained module [encapsulation] of data and the programs [services] that manipulate [process] that data.

**object code.** A code expressed in machine language ["1"s and "0"s] which is normally an output of a given translation process that is ready to be executed by a computer.

**object oriented design.** A software development technique in which a system or component is expressed in terms of objects and connections between those objects.

**object oriented language.** A programming language that allows the user to express a program in terms of objects and messages between those objects. Examples include C++, Smalltalk and LOGO.

**object program.** A computer program that is the output of an assembler or compiler.

**operating system.** Software that controls the execution of programs, and that provides services such as resource allocation, scheduling, input/output control, and data management. Usually, operating systems are predominantly software, but partial or complete hardware implementations are possible.

**Oracle.** A relational database programming system incorporating the SQL programming language. A registered trademark of the Oracle Corp.

**original equipment manufacturer.** A manufacturer of computer hardware.

**PROM.** programmable read only memory.

**parallel.** (1) Pertaining to the simultaneity of two or more processes. (2) Pertaining to the simultaneous processing of individual parts of a whole, such as the bits of a character or the characters of a word, using separate facilities for the various parts. (3) Term describing simultaneous transmission of the bits making up a character, usually eight bits [one byte].

**parallel processing.** See: multi-processing, multi- programming.

**parameter.** A constant, variable or expression that is used to pass values between software modules.

**parity.** An error detection method in data transmissions that consists of selectively adding a 1-bit to bit patterns [word, byte, character, message] to cause the bit patterns to have either an odd number of 1-bits [odd parity] or an even number of 1-bits [even parity].

**Pascal.** A high-level programming language designed to encourage structured programming practices.

**path.** A sequence of instructions that may be performed in the execution of a computer program.

**peripheral device.** Equipment that is directly connected a computer. A peripheral device can be used to input data; e.g., keypad, bar code reader, transducer, laboratory test equipment; or to output data; e.g., printer, disk drive, video system, tape drive, valve controller, motor controller. Syn: peripheral equipment.

**pixel.** (1) In image processing and pattern recognition, the smallest element of a digital image that can be assigned a gray level. (2) In computer graphics, the smallest element of a display surface that can be assigned independent characteristics. This term is derived from the term "picture element".

**platform.** The hardware and software which must be present and functioning for an application program to run [perform] as intended. A platform includes, but is not limited to the operating system or executive software, communication software, microprocessor, network, input/output hardware, any generic software libraries, database management, user interface software, and the like.

**program.** A sequence of instructions suitable for processing. Processing may include the use of an assembler, a compiler, an interpreter, or another translator to prepare the program for execution. The instructions may include statements and necessary declarations.

**program design language.** A specification language with special constructs and, sometimes, verification protocols, used to develop, analyze, and document a program design.

**programmable logic device.** A logic chip that is programmed at the user's site.

**programmable read only memory.** A chip which may be programmed by using a PROM programming device. It can be programmed only once. It cannot be erased and reprogrammed. Each of its bit locations is a fusible link.

**programming language.** A language used to express computer programs.

**PROM programmer.** Electronic equipment which is used to transfer a program [write instructions and data] into PROM and EPROM chips.

**protocol.** A set of semantic and syntactic rules that determines the behavior of functional units in achieving communication.

**prototyping.** Using software tools to accelerate the software development process by facilitating the identification of required functionality during analysis and design phases. A limitation of this technique is the identification of system or software problems and hazards.

**pseudocode.** A combination of programming language and natural language used to express a software design. If used, it is usually the last document produced prior to writing the source code.

**qualification, operational.** Establishing confidence that process equipment and sub-systems are capable of consistently operating within established limits and tolerances.

**qualification, process performance.** Establishing confidence that the process is effective and reproducible.

**RAM.** random access memory. Chips which can be called read/write memory, since the data stored in them may be read or new data may be written into any memory address on these chips. The term random access means that each memory location [usually 8 bits or 1 byte] may be directly accessed [read from or written to] at random.

**ROM.** read only memory. A memory chip from which data can only be read by the CPU. The CPU may not store data to this memory. The advantage of ROM over RAM is that ROM does not require power to retain its program. This advantage applies to all types of ROM chips; ROM, PROM, EPROM, and EEPROM.

**real time.** Pertaining to a system or mode of operation in which computation is performed during the actual time that an external process occurs, in order that the computation results can be used to control, monitor, or respond in a timely manner to the external process. Contrast with batch. See: conversational, interactive, interrupt, on-line.

**real time processing.** A fast-response [immediate response] on-line system which obtains data from an activity or a physical process, performs computations, and returns a response rapidly enough to affect [control] the outcome of the activity or process; e.g., a process control application. Contrast with batch processing.

**record.** (1) a group of related data elements treated as a unit. [A data element (field) is a component of a record, a record is a component of a file (database)].
**relational database.** Database organization method that links files together as required. Relationships between files are created by comparing data such as account numbers and names. A relational system can take any two or more files and generate a new file from the records that meet the matching criteria.

**reliability.** The ability of a system or component to perform its required functions under stated conditions for a specified period of time.

**requirement.** (1) A condition or capability needed by a user to solve a problem or achieve an objective. (2) A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents. (3) A documented representation of a condition or capability as in (1) or (2).

**requirements phase.** The period of time in the software life cycle during which the requirements, such as functional and performance capabilities for a software product, are defined and documented.

**robustness.** The degree to which a software system or component can function correctly in the presence of invalid inputs or stressful environmental conditions. See: software reliability.

**routine.** A subprogram that is called by other programs and subprograms. Note: This term is defined differently in various programming languages.

**SOPs.** standard operating procedures.

**SQL.** structured query language.

**safety.** Freedom from those conditions that can cause death, injury, occupational illness, or damage to or loss of equipment or property, or damage to the environment.

**server.** A high speed computer in a network that is shared by multiple users. It holds the programs and data that are shared by all users.

**simulation.** (1) Use of an executable model to represent the behavior of an object. During testing the computational hardware, the external environment, and even code segments may be simulated. (2) A model that behaves or operates like a given system when provided a set of controlled inputs. Contrast with emulation.

**software.** Programs, procedures, rules, and any associated documentation pertaining to the operation of a system.

**software characteristic.** An inherent, possibly accidental, trait, quality, or property of software; e.g., functionality, performance, attributes, design constraints, number of states, lines or branches.

**software design description.** A representation of software created to facilitate analysis, planning, implementation, and decision making. The software design description is used as a medium for communicating software design information, and may be thought of as a blueprint or model of the system.

**software development process.** The process by which user needs are translated into a software product. the process involves translating user needs into software requirements, transforming the software requirements into design, implementing the design in code, testing the code, and sometimes installing and checking out the software for operational activities.

**software documentation.** Technical data or information, including computer listings and printouts, in human readable form, that describe or specify the design or details, explain the capabilities, or provide operating instructions.

**software element.** A deliverable or in- process document produced or acquired during software development or maintenance.

**software engineering.** The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; i.e., the application of engineering to software.

**software item.** Source code, object code, job control code, control data, or a collection of these items. Contrast with software element.

**software life cycle.** Period of time beginning when a software product is conceived and ending when the product is no longer available for use. The software life cycle is typically broken into phases denoting activities such as requirements, design, programming, testing, installation, and operation and maintenance.

**software reliability.** (1) the probability that software will not cause the failure of a system for a specified time under specified conditions. The probability is a function of the inputs to and use of the system in the software. The inputs to the system determine whether existing faults, if any, are encountered. (2) The ability of a program to perform its required functions accurately and reproducibly under stated conditions for a specified period of time.

**source code.** (1) Computer instructions and data definitions expressed in a form suitable for input to an assembler, compiler or other translator. (2) The human readable version of the list of instructions [program] that cause a computer to perform a task.

**source program.** A computer program that must be compiled, assembled, or otherwise translated in order to be executed by a computer. See: source code.

**specification.** A document that specifies, in a complete, precise, verifiable manner, the requirements, design, behavior,or other characteristics of a system or component, and often, the procedures for determining whether these provisions have been satisfied.

**specification, product.** A document which describes the as built version of the software.

**specification, requirements.** A specification that documents the requirements of a system or system component. It typically includes functional requirements, performance requirements, interface requirements, design requirements [attributes and constraints], development [coding] standards, etc.

**spiral model.** A model of the software development process in which the constituent activities, typically requirements analysis, preliminary and detailed design, coding, integration, and testing, are performed iteratively until the software is complete.

**standard operating procedures.** Written procedures [prescribing and describing the steps to be taken in normal and defined conditions] which are necessary to assure control of production and processes.

**storage device.** A unit into which data or programs can be placed, retained and retrieved.

**structured programming.** Any software development technique that includes structured design and results in the development of structured programs.

**structured query language.** A language used to interrogate and process data in a relational database. Originally developed for IBM mainframes, there have been many implementations created for mini and micro computer database applications. SQL commands can be used to interactively work with a data base or can be embedded with a programming language to interface with a database.

**subprogram.** A separately compilable, executable component of a computer program. Note: This term is defined differently in various programming languages.

**subroutine.** A routine that returns control to the program or subprogram that called it. Note: This term is defined differently in various programming languages.

**support software.** Software that aids in the development and maintenance of other software; e.g., compilers, loaders, and other utilities.

**syntax.** The structural or grammatical rules that define how symbols in a language are to be combined to form words, phrases, expressions, and other allowable constructs.

**system analysis.** A systematic investigation of a real or planned system to determine the functions of the system and how they relate to each other and to any other system.

**system design.** A process of defining the hardware and software architecture, components, modules, interfaces, and data for a system to satisfy specified requirements.

**system life cycle.** The course of developmental changes through which a system passes from its conception to the termination of its use; e.g., the phases and activities associated with the analysis, acquisition, design, development, test, integration, operation, maintenance, and modification of a system. See: software life cycle.

**system software.** (1) Application- independent software that supports the running of application software. (2) Software designed to facilitate the operation and maintenance of a computer system and its associated programs; e.g., operating systems, assemblers, utilities.

**terminal.** A device, usually equipped with a CRT display and keyboard, used to send and receive information to and from a computer via a communication channel.

**test.** An activity in which a system or component is executed under specified conditions, the results are observed or recorded and an evaluation is made of some aspect of the system or component.

**test phase.** The period of time in the software life cycle in which the components of a software product are evaluated and integrated, and the software product is evaluated to determine whether or not requirements have been satisfied.

**test procedure.** A formal document developed from a test plan that presents detailed instructions for the setup, operation, and evaluation of the results for each defined test.

**testing.** (1) The process of operating a system or component under specified conditions, observing or recording the results, and making an evaluation of some aspect of the system or component. (2) The process of analyzing a software item to detect the differences between existing and required conditions, i.e. bugs, and to evaluate the features of the software items.

**testing, compatibility.** The process of determining the ability of two or more systems to exchange information. In a situation where the developed software replaces an already working program, an investigation should be conducted to assess possible comparability problems between the new software and other programs or systems

**testing, unit.** (1) Testing of a module for typographic, syntactic, and logical errors, for correct implementation of its design, and for satisfaction of its requirements. (2) Testing conducted to verify the implementation of the design for one software element; e.g., a unit or module; or a collection of software elements.

**time sharing.** A mode of operation that permits two or more users to execute computer programs concurrently on the same computer system by interleaving the execution of their programs. May be implemented by time slicing, priority-based interrupts, or other scheduling methods.

**top-down design.** Pertaining to design methodology that starts with the highest level of abstraction and proceeds through progressively lower levels.

**transaction.** (1) A command, message, or input record that explicitly or implicitly calls for a processing action, such as updating a file. (2) An exchange between and end user and an interactive system. (3) In a database management system, a unit of processing activity that accomplishes a specific purpose such as a retrieval, an update, a modification, or a deletion of one or more data elements of a storage structure.

**unambiguous.** (1) Not having two or more possible meanings. (2) Not susceptible to different interpretations. (3) Not obscure, not vague. (4) Clear, definite, certain.

**unit.** (1) A separately testable element specified in the design of a computer software element. (2) A logically separable part of a computer program. Syn: module.

**UNIX.** A multitasking, multiple-user (time-sharing) operating system developed at Bell Labs to create a favorable environment for programming research and development.

**utility program.** A computer program in general support of the processes of a computer; e.g., a diagnostic program, a trace program, a sort program.

**utility software.** Computer programs or routines designed to perform some general support function required by other application software, by the operating system, or by the system users. They perform general functions such as formatting electronic media, making copies of files, or deleting files.

**VV&T.** validation, verification, and testing.

**valid input.** Test data that lie within the domain of the function represented by the program.

**validation.** (1) Establishing documented evidence which provides a high degree of assurance that a specific process will consistently produce a product meeting its predetermined specifications and quality attributes.

**validation, software.** Determination of the correctness of the final program or software produced from a development project with respect to the user needs and requirements. Validation is usually accomplished by verifying each stage of the software development life cycle.

**verification, software.** In general the demonstration of consistency, completeness, and correctness of the software at each stage and between each stage of the development life cycle. See: validation, software.

**virus.** A program which secretly alters other programs to include a copy of itself, and executes when the host program is executed. The execution of a virus program compromises a computer system by performing unwanted or unintended functions which may be destructive.

**WAN.** wide area network.

**watchdog timer.** A form of interval timer that is used to detect a possible malfunction.

# CONCLUSION

The manual “Professional English for Software Developers” supplies the key lexical and grammatical material for teaching and learning English for specific purposes in IT sphere of specialization.

The teacher’s work with “Professional English for Software Development” and the assessment of students’ knowledge have shown its conformity to real interests, psychological needs and abilities of the senior and graduate students of Ufa State Aviation Technical University. The proposed textbook is considered as a means of optimizing the learning process of foreign language professional communication of the students and graduate students. It has been complied according to the State Educational Standard and requirements of the foreign language program for technical universities.

Various tasks, dialogues, role-plays and topics suggested for presentations and discussions proved to be the effective means for improving language and social and cultural skills in different kinds of speech activity. As we can see this textbook is designed to work under the guidance of a teacher who organizes the educational process in the classroom and selects tasks for independent extracurricular work.

We hope that ‘Professional English for Software Developers ’ will be interesting and useful in mastering students’ language proficiency.

# REFERENCES AND INFORMATION RESOURCES


1. What is Software Engineering? *[Электронный ресурс]:* – URL: https://www.castsoftware.com/glossary/what-is-software-engineering-definition-types-of-basics-introduction (дата обращения 20.02.22)
2. Software. *[Электронный ресурс]:* – URL: https://searchapparchitecture.techtarget.com/definition/software (дата обращения 25.02.22)
3. Operating system, its Functions and Characteristics. *[Электронный ресурс]:* – URL: https://medium.com/computing-technology-with-it-fundamentals/operating-system-its-functions-and-characteristics-c0946e4215c6 (дата обращения 28.02.22)
4. Types of Operating Systems. *[Электронный ресурс]:* – URL: https://searchapparchitecture.techtarget.com/definition/software (дата обращения 28.02.22)
5. Types of Operating Systems. *[Электронный ресурс]:* – URL: https://www.tutorialspoint.com/operating_system/os_types.htm (дата обращения 1.03.22)
6. Software and its types. *[Электронный ресурс]:* – URL: https://www.geeksforgeeks.org/software-and-its-types.htm (дата обращения 3.03.22)
7. Coding vs Programming: Top Differences. *[Электронный ресурс]:* – URL: https://www.lighthouselabs.ca/en/blog/coding-vs-programming (дата обращения 3.03.22)
8. The first intuitive programming language for quantum computers. *[Электронный ресурс]:* – URL: https://www.sciencedaily.com/releases/2020/06/200615115820.htm (дата обращения 28.02.22)
9. How the brain is programmed for computer programming? *[Электронный ресурс]:* – URL: https://www.sciencedaily.com/releases/2021/01/210128094229.htm (дата обращения 8.03.22)
10. Ian Sommerville. Software Engineering. Tenth Edition. Pearson. 2016. – PP.200-210.
11. Software Engineering Discussion. *[Электронный ресурс]:* – URL: https://www.goodreads.com/topic/show/18012721-how-to-become-an-expert-software-engineer-and-get-any-job-you-want (дата обращения 7.04.22)
12. Programming Languages. *[Электронный ресурс]:* – URL: https://en.wikipedia.org/wiki/Programming_language (дата обращения 8.03.22)
13. Introduction to programming. *[Электронный ресурс]:* – URL: https://www.bbc.co.uk/bitesize/guides/zts8d2p/test (дата обращения 14.03.22)
14. Definition: object-oriented programming. *[Электронный ресурс]:* –

URL: https://www.computerlanguage.com/ results.php?definition=object-oriented+programming (дата обращения 24.03.22)
15. Computer Programming Language. *[Электронный ресурс]:* – URL: https://www.britannica.com/technology/computer-programming-language/ (дата обращения 24.03.22)
16. TypeScript for the New Programmer. *[Электронный ресурс]:* – URL: https://www.typescriptlang.org/docs/handbook/typescript-from-scratch.html (дата обращения 20.03.22)
17. Visual Basic. *[Электронный ресурс]:* – URL: https://www.britannica.com/technology/computer-programming-language/Visual-Basic. (дата обращения 20.03.22)
18. Data Structures Tutorial. *[Электронный ресурс]:* – URL: https://www.javatpoint.com/data-structure-tutorial (дата обращения 25.03.22)
19. C Sharp (programming language). *[Электронный ресурс]:* – URL: https://en.wikipedia.org/wiki/C_Sharp_(programming_language) (дата обращения 1.04.22)
20. Web Development. *[Электронный ресурс]:* – URL: https://en.wikipedia.org/wiki/Web_development#:~:text=Web%20development,-From%20Wikipedia%2C%20the (дата обращения 1.04.22)
21. What is web development? *[Электронный ресурс]:* – URL: https://careerfoundry.com/en/blog/web-development/ (дата обращения 11.04.22)
22. The Anatomy of a Web Page: 14 Basic Elements. *[Электронный ресурс]:* – URL: https://blog.tubikstudio.com/anatomy-of-web-page/ (дата обращения 1.04.22)
23. What is Application Development? - 3 Main Types of Application Development Methodologies. *[Электронный ресурс]:* – URL: https://kissflow.com/low-code/rad/types-of-application-development-methodologies (дата обращения 3.04.22)
24. Английский язык. Информационные технологии = English for Information Technology: учебное пособие для студентов технических и инженерно-экономических специальностей / И. Ю. Ваник, О. А. Лапко, Н. В. Сурунтович. – Минск : БНТУ, 2016. – 157 с.
25. Language types. *[Электронный ресурс]:* – URL: https://www.britannica.com/technology/computer-programming-language (дата обращения 3.04.22)
26. Game Engine. *[Электронный ресурс]:* – URL: https://en.wikipedia.org/wiki/Game_engine (дата обращения 23.02.22)
27. Artificial Intelligence. *[Электронный ресурс]:* – URL: https://www.techtarget.com/searchenterpriseai/definition/AI-Artificial-Intelligence (дата обращения 23.03.22)

28. Language Types. Education-Oriented Languages. *[Электронный ресурс]:* – URL: https://www.britannica.com/technology/computer-programming-language (дата обращения 23.04.22)
29. What is OOAD(Object-oriented analysis and design)? *[Электронный ресурс]:* – URL: https://www.brainkart.com/article/What-is-OOAD(Object-oriented-analysis-and-design)-_9969/(дата обращения 23.04.22)
30. Rendering (computer graphics). *[Электронный ресурс]:* – URL: https://en.wikipedia.org/wiki/Rendering_(computer_graphics) (дата обращения 5.04.22)
31. Understanding the Four Types of Artificial Intelligence. *[Электронный ресурс]:* – URL: https://www.govtech.com/computing/understanding-the-four-types-of-artificial-intelligence.html (дата обращения 16.04.22)
32. Linux System - Basic Concepts. *[Электронный ресурс]:* – URL: https://www.brainkart.com/article/Linux-System---Basic-Concepts_9864/ (дата обращения 7.04.22)
33. A Developer's Guide to Communicating With Clients. *[Электронный ресурс]:* – URL: https://inchoo.net/life-at-inchoo/developers-guide-communicating-clients/ (дата обращения 30.03.22)
34. Object-Oriented-Programming. *[Электронный ресурс]:* – URL: https://www.brainkart.com/article/Object-Oriented-Programming_10384/ (дата обращения 30.03.22)
35. Role of process in software quality. *[Электронный ресурс]:* – URL: https://www.brainkart.com/article/Role-of-process-in-software-quality_9136/ (дата обращения 1.04.22)
36. Mobile Computing Applications. *[Электронный ресурс]:* – URL: https://www.brainkart.com/article/Mobile-Computing-Applications_9874/ (дата обращения 10.04.22)
37. Англо-русский и русско-английский словарь. *[Электронный ресурс]:* – URL: https://www.multitran.com/
38. Computer Glossary. *[Электронный ресурс]:* – URL: https://www.tutorialspoint.com/computer_glossary.htm