Unit 9

SOFTWARE DEVELOP-MENT

1.1. Read and translate the following words and word-combinations.

Requirements, a function, reliable, an error, modify, satisfy, a developer, error-free, reusable, accomplish, top-down structured design, creative decisions, implementation, life cycle, end-user level, side by side, time consuming, inevitable bugs, rough sketches, refine, clay models, unusable, evaluate, the initial design, a structure chart, implementation, the eventual retirement of a system, a completed application, a specific programming language, a special –purpose word processor, evaluate the entire program, logical errors, divide a program into modules, pseudo code listing.

1.2. Learn key-words and word-combinations.

bootstrap loader – початковий завантажувач
coding – кодування
compiler (n) – компілятор; транслятор
definition (n) – визначення
flowchart (n) – блок-схема
implementation (n) – впровадження
interpreter (n) – інтерпретатор (програм)
linker (n) – редактор зв'язків
loader (n) – завантажувач
maintenance (n) – супровід; обслуговування
mockup (n) – модель
object program – об'єктна програма; вихідна програма
object code – об'єктна програма
program generator – генератор програм
program editor – редактор текстів програм

replacement (n) – заміна
requirements list – список вимог
source program – початкова програма
source code – початкова програма
scale (n) – масштаб
top-down structured design – низхідне структурне програмування

1.3. Read and translate the text.

The Phases of Software Development

Software development is the process of analyzing the requirements of a system , and then designing, writing, and testing the software. The goals of software development are to

1. Produce software that is easy to change and correct when errors are found
2. Modify the software to satisfy new requirements.
3. Produce parts of programs that will work with parts produced by other software developers.
4. Produce software that is reusable.

To accomplish those goals, software developers approach problems using a problem-solving process that contains several useful principles. The first prin-ciple is **top-down structured design,** which starts with the whole problem and develops more and more detail as the solution evolves. The second principle is a **systems life-cycle** approach. This proceeds from definition to design, and then from development to implementation, and finally to the eventual retirement and replacement of a system. The third principle is to make many of the important and creative decisions occur early in the definition and design stages of the life cycle. The fourth principle consists of a systematic series of actions directed toward a goal.

The process of creating software or programs consists of the following phases:

1. Define the problem.
2. Plan the solution.
3. Implement the solution.
4. Test the solution.
5. Assemble the documentation.
6. Maintain the software.

Two things must happen in the program design phase: (1) the problem must be understood and defined, and (2) a solution must be engineered. Some of the understanding can be derived from a **requirements list.** In addition to the requirements, the problem should be studied in an actual situation to understand how people use an application, the data the application processes, the hardware used to implement the application, and any other fac -tors that might affect the performance of the application. These factors should then be compared to the requirements. This process generally produces a completed application that is more naturally organized than if the requirements alone drove the initial design.

The second major step in the software-development process is the engineering of a solution. The top-down strategy – starting with the whole and working toward the parts – is a fundamental principle in all program design and allows the developer to analyze a complex problem by breaking it down into simple functions. After analyzing the requirements, the developer begins the task of actually writing the solution on paper. A developer might produce a structure chart, in which each function (or module) appears as a block and its interaction with every other function is shown. Such tools as **flowcharts, prototyping** and **pseudocode** help the developer design the program.

Implementing a solution requires answers to several planning and design questions. Does software that will solve the problem already exist? How have others solved similar problems? Are the requirements so unique that new software must be written from scratch? The answers to these questions involve defining (1) the scale of the application ; (2) the scope of the application; and (3) the type of data involved in the application.

After the problem has been defined and the solution planned, there is a wide range of development tools from which to choose: **programming languages ,program generators and interface builders.**

If the solution involves the use of a programming language, a program must be coded. **Coding** a program means writing that program in a specific programming language. This is done with a program editor, which is a special-purpose word processor that allows the programmer to enter, edit, change, add, and delete programming-language statements The program coded by the programmer cannot be directly executed by a computer unless it is coded in machine language. The program written in a programming language is called a **source program,** or simply **source code.** It must be

translated into machine language, which is called an **object program** or **object code.** The translator program is called a **compiler** or **interpreter.** Both compilers and interpreters perform the functions of (1) checking program instructions for correct spelling and grammar and then (2) translating correct program instructions into a machine language, but they work slightly differently.

An interpreter interactively translates each programming statement into an immediately usable sequence of machine-language instructions. For example, when you write a program on most personal computers, you type in the program statements and then run the program. An interpreter translates the first statement in the program and executes it, then translates the second statement and executes it, and so on, until the program is finished.

The goal of testing is to find errors. A major form of testing involves running the program using different sample sets of data and simulating the conditions under which the finished program will be run. The testing phase involves working out all the logical errors in the program – errors that a compiler cannot detect. It is best to test all the possible conditions that can occur in a program, both expected and unexpected. When a program is divided into modules, each module can be exhaustively tested, then combined with other modules, and tested together until the final program can be verified. A completed program should not only do what it is supposed to do (i.e., perform its intended function), but also it should not do anything that it is not supposed to do.

Documentation falls into two categories: program documentation and user documentation. Program documentation is the written information necessary to support the software. All software has a life cycle in which it must be maintained. Some of the maintenance involves fixing any errors that might have slipped through the testing phases; other maintenance involves additions or enhancements that might have to be incorporated into the program. Program documentation also includes a pseudo code listing or a flowchart and formal specification of the inputs, control data, and outputs. For the software user, the documentation that matters is the set of user or instruction manuals. These documents should (1) explain how to use the software and (2) cover everything that can possibly happen while the software is being used.

The last phase in the process is called **maintenance.** Software needs to be revised continuously to remove bugs or errors, improve performance,

accommodate new hardware or software, or add new requirements. Maintenance is usually an expensive process because as the software is modified it usually becomes larger, more complex, and more error-prone. Also, the documentation that accompanies the software must be modified.

1.4. Give English equivalents of the following words and word-combinations.

Підхід до життєвого циклу системи, процес вирішення проблеми, від розробки до впровадження, технологія програмування, створення програмного забезпечення, структурна схема, основні принципи, блок-схема, впровадження прикладної програми, задовольняти вимоги, заміна системи, макетування, проектування програми, ефективність прикладної програми, початковий завантажувач, підтримувати життєвий цикл системи, процес створення програми, підтримувати програмне забезпечення, використовувати програмне забезпечення, знаходити помилки, моделювати умови.

1.5. Give the definition of the following terms in English.

A linker, a loader, a program generator, an interface builder, a source program, an object program.

1.6. Translate the following sentences into English.

1. Дотримання підходу до життєвого циклу системи вважається стандартною процедурою при її створенні.
2. Блок-схема складається із стандартних символів, які означають тип операцій.
3. Псевдокод є альтернативою блок-схеми, і з ним зазвичай легше працювати.
4. Редактор зв'язків – це програма, яка з'єднує модулі в єдину програму.
5. Налагодження програми відноситься до процесу виявлення помилок у програмі її розробником.

Debugged	pseudo code	CASE	a linker
top-down design	software development	error message	

1. A program will not work properly until it is … .
2. … is a written description of a program using English statements.

1.7. Fill in the blanks with the words from the box. You can use the word more than once.

3. Software developers often use … to apply the computer to the tasks of defining, designing, and developing the software.
4. … is a program that links modules together and produces an entire program.
5. One of the goals of … is to produce software that is reusable.
6. An advantage of … is the ability to write statements that are flexible enough to be coded in different programming languages.
7. Using the … programmer starts with the whole and works toward the parts.
8. Sometimes the … points only to a general area in which the error occurred.

1.8. Answer the questions.

1. What are the goals of software development?
2. What phases does the process of working out software consist of?
3. What principles do software developers apply while creating a programmme?
4. What criteria are necessary in the program design phase?
5. Why is a requirements list necessary to a software developer?
6. How do you build an application with an interface builder?
7. Why are program generators called nonprocedural languages?
8. What is the function of a bootstrap loader?
9. What do we call a program written in a programming language?
10. What functions do compilers and interpreters perform?

1.9. Write a brief report comparing two program generators.

2.1. Read and translate the following words and phrases.

A structured approach, software-development projects, intractable, interrelated concepts, in general terms, a template, the lower-level functions, a set of interrelated modules, the design logic, verify, from beginning to end, modularization, basic premises, encompass, loop, a software entity, gain favour, a step-by-step set, exponentially, a directory, incorporate, a binary search, simplify, complexity, in terms of, accommodating, at a similar rate, back and forth, in addition.

2.2. Learn key words and word-combinations.

associate (v) – з'єднувати; сполучати
autonomy (n) – автономність
binary search – двійковий пошук
bottom-up design – висхідне проектування
class (n) – клас; категорія
debug (v) – налаштовувати
facilitate (v) – полегшувати
hierarchy chart – структурна схема
interconnection (n) – взаємозв'язок
library (n) – бібліотека
looping – організація циклу
module (n) – модуль
object-oriented programming – об'єктно-орієнтоване програмування
package (n) – модуль; пакет
procedure (n) – процедура; процес
selection (n) – вибір; селекція
sequence (n) – послідовність
similarity (n) – подібність
simplify (v) – спрощувати
structure chart – структурна схема
structured programming – структурне програмування

2.3. Read and translate the text.

Structured and Object-Oriented Programming

Structured programming is the process of dividing the program into smaller units or modules, which allows clearer expression of the problem and simpler organization. A structured programming language facilitates the process. A structured approach can help to formalize design activities and decisions so that teams of developers can work together on software-development projects. Three interrelated concepts form the basis of a structured approach to programming: top-down design, modularization, and **the standardized control structures**.

Top-down design is a popular technique for structured design. First, it consists of specifying the solution to a problem in general terms and, then, breaking the solution down into finer and finer details. Top-down design is

similar to creating an outline and later filling it in. It is often contrasted to **bottom-up design**. Top-down design refers to starting with the whole and working toward the parts, whereas bottom-up design refers to starting with the parts and trying to build a whole, and it is generally used when coding a program.

A tool that is often used in top-down design is a **structure chart,** sometimes called a **hierarchy chart**, which is a diagram that shows the components of a program and the interconnection between components. The developer analyzes the problem by starting at the top or the most general function and carefully works down to the lower-level functions. The developer need not look at any details of the processes involved. The task is to determine what needs to be done, isolate the processing steps, and break the program into a set of interrelated modules. The resulting diagram of modules becomes a useful visual method for checking the design logic.

The concept of **modularization** helps a software developer to divide a complex program into smaller subprograms. A **module** is a set of instructions that can be tested and verified independent of its use in a larger program. In other words, it can be coded, debugged, and tested without having to write an entire program. It is much easier to write a small module, test and verify it, and then combine it with other modules to form a program than it is to write and test an entire large program from beginning to end.

How do independent modules work as part of a larger program? Just as programs accept input, process it, and produce output, modules perform similar functions. Modules communicate to one another by passing data back and forth. Inputs specify the information needed for the module to do its task. The output is the result of the processing done by the module. In this way, a module becomes a well-defined processing task that can be developed and then tested separately. Modern programming languages support modularization directly through features called **units, packages, functions, procedures**, and even **modules.** Large-software development projects are so complex that modularization is a necessity.

Another basic premises of structured programming is that all the programs can be written using three basic control structures, which are statements in the program that controls the order in which the instructions are executed. The basic control structures are **sequence**, **selection**, and **looping** – the process of repeating the execution of a set of instructions. A fourth control structure, called the **case-control structure,** is often included in

structured discussions. These control structures are designed to help maintain the modular program structure.

An algorithm is a step-by-step set of instructions for solving a specific problem. In the context of developing software, algorithms are sets of instructions for computational problems. There are problems for which no known algorithms exist, such as getting a computer to play a perfect game of chess. There are also problems for which an algorithm exists, but the computer time needed to solve it grows exponentially with the size of the problem. Such problems are called intractable.

There are also problems for which both an algorithm and a reasonable time-solution exist. For example, using a technique called a **binary search**, determine how many comparisons it would take in the worst case to find a name in a one-million-name telephone directory. (In the worst case of a sequential search, one million comparisons would be required.) The answer is 20. Also, as the telephone directory grows exponentially larger, the number of comparisons required, does not increase at a similar rate. A one-billion-name telephone directory would, in the worst case, require only 30 comparisons. Many algorithms exist for sorting, searching, statistical calculating, and other mathematical procedures. If you need to sort some data quickly, for example, you can look up a published algorithm for fast sorting, copy it, and incorporate it into your program. These public algorithms are an excellent source problem-solving information, and if fit the problem, they can easily be used by the developer.

In addition to structured programming, another approach for dealing with the complexity of software is called object-oriented programming, which is gaining favour among software developers.

Object-oriented programming is a technique in which the developer breaks the problem into modules called **objects** that contain both data and instructions and can perform specific tasks. The developer then organizes the program around the collection of objects. It is characterized by the following concepts.

1. **Autonomy.** The ability to deal with software entities (objects) as units that interact only in defined, controllable ways.
2. **Similarity.** The ability to describe a software entity in terms of its differences from another entity. In other words, a new object can inherit characteristics from an old one.

Classification. The ability to represent an arbitrary number of data items that all have the same behaviour and the same characteristics with a

single software entity, called a class. The class serves as a template from which specific objects are created. A collection of classes associated with a particular environment is called a library.

Classes describe what objects can exist, the characteristics of those objects, and what kinds of procedures can be performed with the objects. Classification and similarity allow new system elements to be defined by leveraging the specifications of existing classes. For example, if we wanted to show our invoice on a display screen as well as print it, we could add a new "display invoice" object that inherits the general characteristics of invoices from an invoice class. A software developer would create the display invoice object by programming the differences between the print-invoice object and the display-invoice object. In the process, previously defined characteristics are used when appropriate.

Objects and their classes provide another technique for understanding and defining a problem phase in the process of designing a program. Perhaps what is more important, the object model is very accommodating to engineering a solution. In most cases, the object model can be used directly as a blueprint when creating an object-oriented program.

2.4. Give English equivalents of the following words and word-combinations.

Взаємозв'язок між компонентами, функції нижчого рівня, візуальний метод, таким чином, підтримувати модуляризацію, виконувати подібні функції, застосовувати специфікацію, успадкувати загальні характеристики, особливе операційне середовище, засіб, довільна кількість даних, шаблон, поведінка, специфічний об'єкт, інший підхід, покроковий, двійковий пошук, подібність.

2.5. Translate the following word-combinations into English so that you could form compound nouns.

Розробка програмного забезпечення, вимоги користувача, список вимог, засоби розробки, довідник даних, словник бази даних, генератор звітів, рівень користувача, повідомлення про помилку, документація програми, інструкція для користувача, розробники програмного забезпечення, генератор програм.

2.6. Translate the following sentences into English.

1. Алгоритм вказує, які операції оброблення даних і в якій послідовності необхідно виконати, щоб одержати розв'язок задачі.

2. Застосовуючи об'єктно-орієнтоване програмування, розробник розбиває проблему на модулі, які називаються об'єктами.
3. Об'єкт є автономним модулем, який містить в собі дані і команди.
4. Прихильники об'єктно-орієнтованого програмування вважають, що воно прискорює процес створення програми та знижує витрати на її розробку.
5. Однією з головних переваг модуляризації є те, що модулі можна перевіряти по ходу розробки проекту.

2.7. Fill in the blanks with the words from the box.

hierarchy chart	object-oriented programming	an object model
an object	a library	

1. … is a diagram that shows the components of the program and the interconnections between them.
2. … is gaining favour among software developers.
3. A collection of classes associated with a particular environment is called … .
4. … is very accommodating to engineering a solution.
5. … is a self-contained module that contains data and instructions.

2.7. Explain the following terms in your own words.

Binary search, bottom-up design, debug, library, hierarchy chart, structured programming.

2.8. Answer the questions.

1. What is the difference between structured and object-oriented programming?
2. What tool is often used in top-down design?
3. Explain the concept of modularization.
4. What is one of the major advantages of modularization?
5. Explain the concept of an algorithm. Give an example of intractable problems.
6. In what cases do you use a technique called a binary search?
7. What are the three basic control structures of structured programming and what is their function?
8. What do classes describe?
9. What is one of the major advantages of modularization?

Test Yourself

Match the following key terms to their appropriate definitions:

A.

1. ____Software development
2. _____Top-down design
3. _____Systems life-cycle approach
4. _____Prototyping
5. _____Flowchart
6. _____Pseudocode
7. _____Source program
8. _____Software engineering
9. _____Programmer
10. ____Computer-aided software engineering
11. ____Coding

a. A program written in a programming language such as C++ or Delphi.
b. The process of analyzing the requirements of a system, then designing, writing, and testing the software.
c. Proceeding from definition to design, development to implementation, and the eventual retirement and replacement of a system.
d. A trained person who create computer programs.
e. The application of scientific and mathematical principles to the design and development of producing software.
f. A written description of a program using English statements.
g. Starting with the whole program and developing more and more details as the solution evolves.
h. Writing a program in a specific programming language.
i. A set of tools to automate the tasks involved in designing and developing large-scale or complex software projects.
j. A graphic way of representing the thinking that goes into solving a problem.
k. The process of building working models of the application, testing them, and modifying them in response to new ideas.

B.

12. ____Object program
13. ____Interpreter
14. ____Compiler

15. ____ Debugging
16. ___Linker
17. ____Loader
18. _____Program generator
19. ____Report generator
20. ____Nonprocedural language
21. ____Interface builder

a. A program that brings programs stored in a program library into memory for execution.
b. A program that combines several modules into one executable program.
c. A program that has been translated into machine language.
d. A tool for assembling a user interface, also called front end, from a library of predefined graphic objects.
e. A special type of program generator that can generate only one type of program: a report.
f. A program that interactively translates each programming statement into an immediately usable machine-language instruction.
g. A program that translates correct high-level programming statements into a machine-language instructions all at one time.
h. Defining problems in terms of the results desired instead of procedural programming terms.
i. A way to generate a program based on writing specifications about the problem to be solved.
j. The process of uncovering errors in a program.

C.

22. ____Program testing
23. ____Program documentation
24. ____Structured programming
25. ____Bottom-up design
26. ____Structure chart
27. ____Module
28. ____Algorithm
29. ____Control structure
30. ____Object-oriented programming
31. ____Object