

Table of Contents

Table of Contents	1
Learning Objectives	2
Program.....	2
Programmer.....	2
Programming Language.....	2
Types of Languages	2
Low-Level Language	2
High-Level Language	2
Types of High-Level Languages.....	3
Program Development Cycle.....	3
Program Development Cycle Steps	3
Step 1 – Analyze Requirements.....	4
Major Tasks	4
Task 1 – Review the Requirements	4
Task 2 – Meet with the Systems Analyst and Users.....	4
Task 3 - Identify Input, Output, Processing, and Data Components.	4
Step – 2 Design Solution.....	5
Structured Design.....	5
Object-Oriented Design	6
Control Structures	7
Types of Control Structures	7
Sequence Control Structure	7
Selection Control Structure.....	7
Repetition Control Structure.....	8
Types of Repetition Structures.....	8
Do-While Repetition Structure	8
Do-Until Repetition Structure.....	9
Design Tools	9
Types of Design Tools	9
Flowcharts.....	9
Pseudocode	11
Step 3 – Validate Design.....	11
Techniques for reviewing a Solution Algorithm	11
Desk Check	11
Desk Check Steps	11
Structure Walkthrough.....	12
Step 4 – Implement Design.....	12
Step 5 – Test solution.....	12
Step 6 – Document Solution	12
Reference for Further Reading.....	13

Learning Objectives

After completing this handout, you will be able to:

- Understand the six steps in the program development cycle.
- Differentiate between structured and object-oriented design.
- Understand program design techniques.

Program

- A **computer program** is a series of instructions that directs a computer to perform tasks.

Programmer

- A **computer programmer** is a person who writes and modifies computer programs.

Programming Language

- A **programming language** is a set of words, symbols, and codes that enables a programmer to communicate instructions to a computer.
- Programmers use a variety of programming languages and tools to write, or code, a program.
- Each programming language has its own rules for writing the instructions.

Types of Languages

- Low-level language
- High-level language

Low-Level Language

- A low-level language is a programming language that is machine dependent.
- Programs written in low-level language are not easily portable.
- Each language instruction in a low-level language usually equates to a single machine instruction.

High-Level Language

- Each language instruction typically equates to multiple machine instructions.
- Programs written in high-level language are machine independent.

Types of High-Level Languages

- Procedural languages
- Visual programming languages
- Object-oriented programming languages
- Non-procedural languages and tools
- Web page programming languages
- Multimedia programming languages

Program Development Cycle

- The **program development cycle** is a series of steps programmers use to build computer programs.
- The program development cycle guides computer programmers through the development of a program.
- The program development cycle consists of six steps

Program Development Cycle Steps

1. Analyze Requirements
2. Design Solution
3. Validate Design
4. Implement Design
5. Test Solution
6. Document Solution

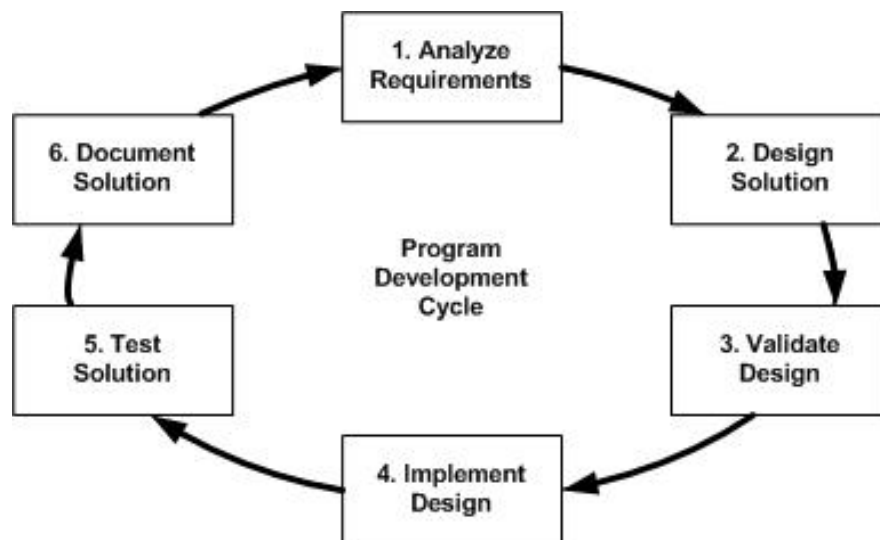


Figure 1: Program Development Cycle

Step 1 – Analyze Requirements

- The first step is to analyze the requirements of the problem the program should solve.
- The analysis step consists of three major tasks.

Major Tasks

1. Review the requirements
2. Meet with the systems analyst and users
3. Identify input, output, processing, and data components.

Task 1 – Review the Requirements

- The requirements may be in the form of deliverables such as charts, diagrams, and reports.
- The deliverables document various aspects of the users' requirements.
- Screen and report layout charts show input and output requirements.
- Structured English, decision tables, and decision trees convey processing requirements.
- The data dictionary identifies data requirements.
- By reviewing deliverables, the programmer understands the nature of the requirements.

Task 2 – Meet with the Systems Analyst and Users

- Programmers also meet the systems and users to understand the purpose of the requirements.

Task 3 - Identify Input, Output, Processing, and Data Components.

- After design specifications are established, the programmer defines the input, processing and output (IPO) requirements for the program.
- Programmers normally use the IPO chart for this task.
- An **IPO chart** identifies a program's input, its outputs, and the processing steps required to transform the inputs into the outputs.
- Programmers review the contents of the IPO chart with the systems analyst and the users.

Input	Processing	Output
Regular Time Hours Worked	Read regular time hours worked, Overtime hours worked, hourly pay rate.	Gross pay
Overtime Hours Worked	Calculate regular time pay.	
Hourly Pay Rate	If employee worked overtime, calculate overtime pay.	
	Calculate gross pay	
	Print gross pay	

Figure 2: IPO Chart

Step – 2 Design Solution

- Designing the solution involves devising a **solution algorithm** to satisfy the requirements.
- A **solution algorithm**, also called **program logic**, is a graphical or written description of the step-by-step procedures to solve the problem.
- System can be designed using the **process modeling** (structured analysis and design) or **object modeling** (object-oriented analysis and design).

Structured Design

- In **structured design**, sometimes called **top-down design**, the programmer typically begins with a general design and moves toward a more detailed design.
- This approach breaks down the original set of requirements into smaller, more manageable sections.
- The **first step** is to identify the major function of a program, sometimes called the **main routine**.
- Next, the programmer decompose breaks down) the main routine into smaller sections, called **subroutines**.
- A section of a program that performs a single function is a **module**. Subroutines are further decomposed into modules.
- Programmers use a **hierarchy chart**, also called a **structure chart**, to show program modules graphically.
- Program developed using the top-down approach benefit from the simplicity of their design.
- Program developed are usually reliable and easy to read and maintain.

- Structured design, however, does not provide a way to keep the data and the program together.

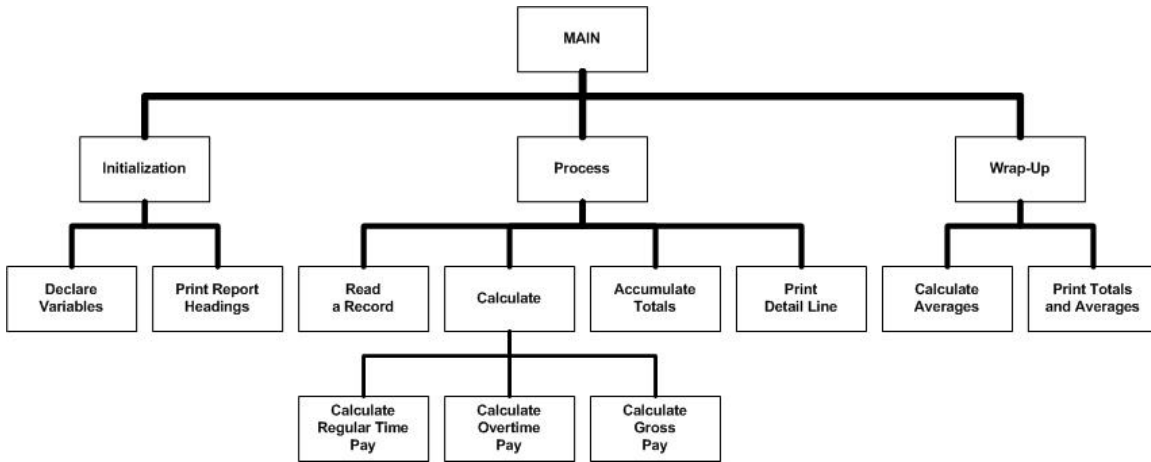


Figure 3: Hierarchy Chart

Object-Oriented Design

- With **object-oriented (OO) design**, the programmer packages the data and the program (or procedure) into a single unit, an **object**.
- The concept of packaging the data and procedures into a single object is called **encapsulation**. That is, an object encapsulated (hides) the details of the object.
- Objects are grouped into **classes**.
- To represent classes and their hierarchical relationship graphically, programmers use a **class diagram**.
- A **class diagram** provides a visual representation of each object, its attributes, and its methods.

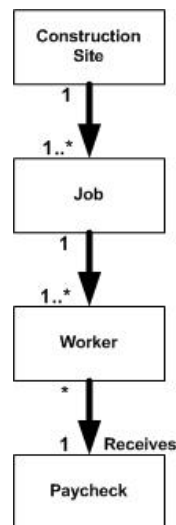


Figure 4: Class Diagram

Control Structures

- A **control structure**, also known as a construct, depicts the logical order of program instructions.

Types of Control Structures

- Sequence Control Structure
- Selection Control Structure
- Repetition Control Structure

Sequence Control Structure

- A **sequence control** structure shows one or more actions following each other in order.

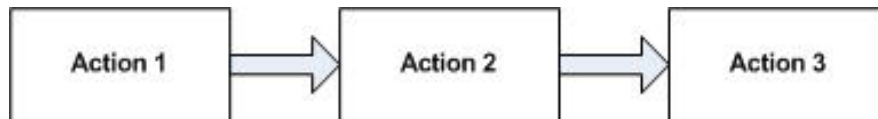


Figure 5: Sequence Control Structure

Selection Control Structure

- A **selection control structure** tells the program which action to take, based on a certain condition.
- Two common types of selection control structures are the **if-then-else** and the **case**.
- **If-then-else**: when a program evaluates the condition in an **if-then-else** control structure, it yields one of two possibilities: **true** or **false**.
- If the result is true, then the program performs one action. If the result is false, the program performs a different action.

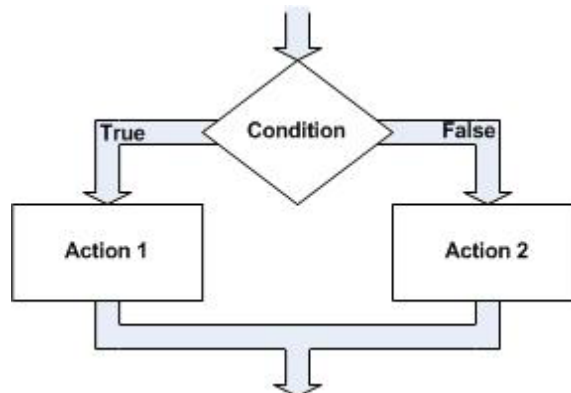


Figure 6: If-Then-Else Control Structure

- **Case**: with the case control structure, a condition can yield one of three or more possibilities.

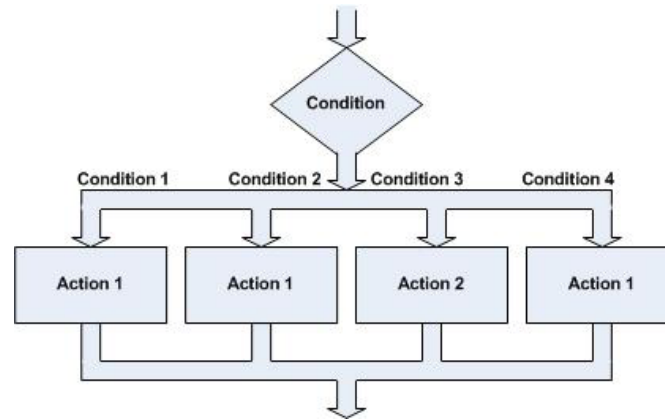


Figure 7: Case Control Structure

Repetition Control Structure

- The repetition control structure enables a program to perform one or more actions repeatedly as long as a certain condition is met.

Types of Repetition Structures

- Do-While Repetition Structure
- Do-Until Repetition Structure

Do-While Repetition Structure

- A do-while repetition structure repeats one or more times as long as a condition is true.
- This control structure tests a condition at the beginning of the loop.
- This control structure normally is used when occurrence of an event is not quantifiable or predictable.

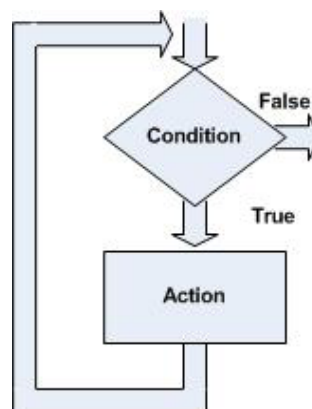


Figure 8: Do-While Control Structure

Do-Until Repetition Structure

- The do-until control structure tests the condition at the end of the loop.
- This control structure always will execute at least once.
- This control structure normally is used when occurrence of an event is known or predictable.

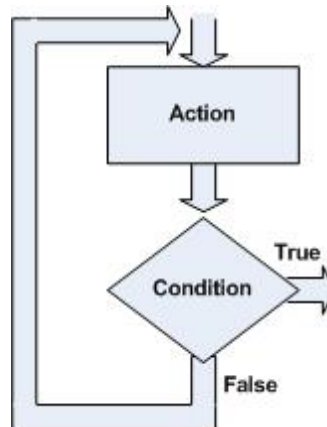


Figure 9: Do-Until Control Structure

Design Tools

- To help document a solution algorithm, programmers use design tools.

Types of Design Tools

- Flowcharts
- Pseudocode

Flowcharts

- A flowchart graphically shows the logic in a solution algorithm.
- The American National Standards Institute (ANSI) published a set of standards for program flowcharts.
- Programmers use commercial flowchart software to develop flowcharts.
- Two popular flowcharting programs are SmartDraw and Visio 2003.

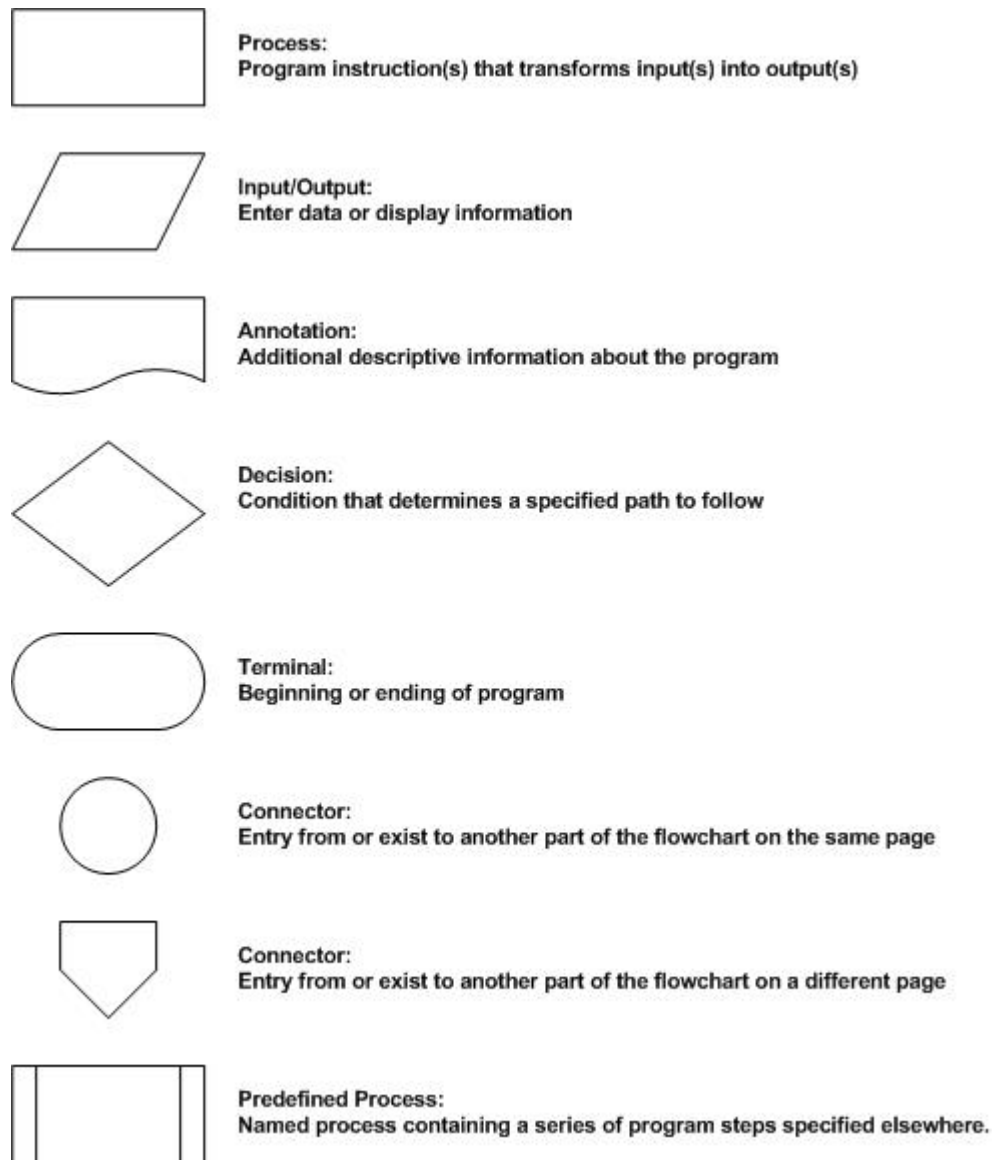


Figure 10: Flowchart Symbols

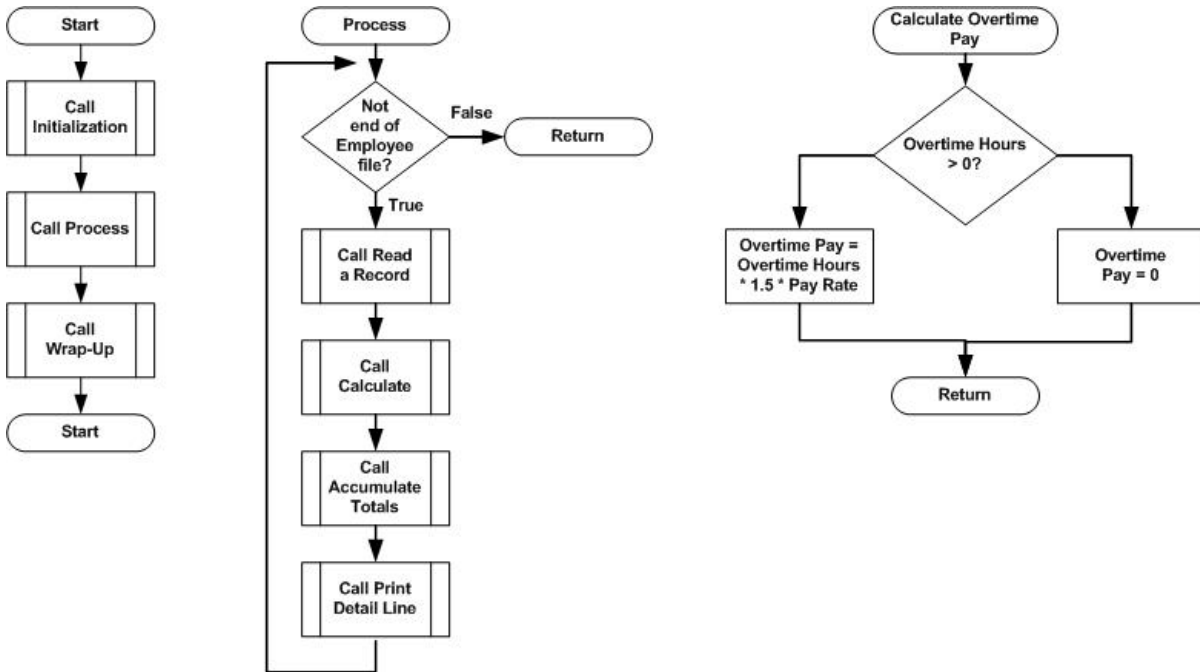


Figure 11: Flowcharts

Pseudocode

- Pseudocode uses a condensed form of English to convey program logic.

Step 3 – Validate Design

- After developing the solution algorithm using a program flowchart or Pseudocode, programmers validate, or check the program design for accuracy.
- The programmers check the logic for correctness and attempts to uncover the logic errors.
- A **logic error** is a flaw in the design that causes inaccurate results.

Techniques for reviewing a Solution Algorithm

- Desk Check
- Structured Walkthrough

Desk Check

- In desk check programmers use test data to step through the program logic.

Desk Check Steps

- Develop various sets of test data (inputs).
- Determine the expected results (outputs).
- Step through the solution algorithm using test data and record actual results.
- Compare the expected results with actual results.

- Repeat above step for each set of test data.

Structure Walkthrough

- Programmers often use a structured walkthrough to review deliverables during the program development cycle.
- The purpose of structured walkthroughs is to identify errors in the program logic and check for possible improvements in program design.

Step 4 – Implement Design

- Implementation of the design includes writing the code that translates the design into a program.

Step 5 – Test solution

- Through testing of programs is very important as many users rely on the program and its output to support their daily activities and decisions.
- The goal of program testing is to ensure the program runs correctly and is error free.
- Errors uncovered during this step usually are one of two types: (1) syntax errors or (2) logic errors.
- A syntax error occurs when the code violates the syntax, or grammar, of the programming language.
- Syntax errors are normally discovered at the time of compiling the program.
- The procedure for testing for logic errors is much like the desk checking techniques used in the Validation Design Step.
- An error that occurs while the program is running is referred to as run-time error such as divide-by-zero error.
- The process of locating and correcting syntax and logic errors in a program is known as debugging the program.
- Most programming languages include a debug utility which assists programmers with identifying syntax errors and finding logic errors.
- As a general rule, the more time and effort programmers spend analyzing and designing the solution algorithm, the less time spent debugging the program.

Step 6 – Document Solution

- In documenting the solution the programmer performs two activities: (1) review the program code and (2) review all the documentation.

- The documentation includes all charts, solution algorithms in the form of program flowcharts or Pseudocode, test data, and program code listings that contain global and internal comments.
- Proper documentation greatly reduces the amount of time a new programmer spends learning about existing programs.

Reference for Further Reading

- Discovering Computer 2005, A Gateway to Information, Complete Edition, Shelly Cashman Vermaat, Chapter # 13.