Programming Methodologies

tutorialspoint

SIMPLY EASY LEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia

https://twitter.com/tutorialspoint

About the Tutorial

When programs are developed to solve real-life problems like inventory management, payroll processing, student admissions, examination result processing, etc., they tend to be huge and complex. Programming Methodology is the approach to analyzing such complex problems by planning the software development and controlling the development process.

In this tutorial, we will cover the top-down approach to programming, also called **modular programming**. We will also learn about requirement gathering, problem definition and identifying unique solution to the given problems. In addition, we will throw light on the best practices for code optimization.

Audience

This tutorial is designed for anyone who wants to learn about programming methodologies and how to use them to design solutions to a given problem.

Prerequisites

There are no prerequisites for this tutorial except a desire to learn how to write good programs. However, it would definitely help if the readers have some prior experience in writing codes in any programming language.

Disclaimer & Copyright

Table of Contents

About the Tutorial i

Audience i

Prerequisites i

Disclaimer & Copyright i

Table of Contents ii

1. PROGRAMMING METHODOLOGIES – INTRODUCTION 1

Types of Programming Methodologies 1

Top-down or Modular Approach 2

Bottom-up Approach 3

2. PROGRAMMING METHODOLOGIES – UNDERSTANDING THE PROBLEM 4

Requirement Gathering 4

Problem Definition 5

3. PROGRAMMING METHODOLOGIES – IDENTIFYING THE SOLUTION 6

Flowcharting 6

Data Flow Diagram 7

Pseudocode 7

Identifying Mathematical Operations 8

4. PROGRAMMING METHODOLOGIES – APPLYING MODULAR TECHNIQUES 9

Advantages of Modular Programming 9

Identifying the Modules 9

Step-by-Step Solution 10

Control Structures 10

5. PROGRAMMING METHODOLOGIES – WRITING THE ALGORITHM 13

6. PROGRAMMING METHODOLOGIES – FLOWCHART ELEMENTS 15

7. PROGRAMMING METHODOLOGIES – USING CLEAR INSTRUCTIONS 17

Clarity of Expressions **17**

Simplicity of Instructions **18**

8. PROGRAMMING METHODOLOGIES – CORRECT PROGRAMMING TECHNIQUES 19

Proper Identifier Names **20**

Comments **20**

Indentation **21**

9. PROGRAMMING METHODOLOGIES – DEBUGGING 22

Syntax Errors **22**

Semantic Errors **23**

Runtime Errors **23**

Code Optimization **24**

Execution Time Optimization **24**

Memory Optimization **25**

10. PROGRAMMING METHODOLOGIES – PROGRAM DOCUMENTATION 26

Advantages of Documentation **26**

Example Documents **26**

11. PROGRAMMING METHODOLOGIES – PROGRAM MAINTENANCE 28

Types of Maintenance **28**

Maintenance Tools **29**

1. Programming Methodologies – Introduction

When programs are developed to solve real-life problems like inventory management, payroll processing, student admissions, examination result processing, etc. they tend to be huge and complex. The approach to analyzing such complex problems, planning for software development and controlling the development process is called **programming methodology**.

Types of Programming Methodologies

There are many types of programming methodologies prevalent among software developers:

Procedural Programming

Problem is broken down into procedures, or blocks of code that perform one task each. All procedures taken together form the whole program. It is suitable only for small programs that have low level of complexity.

Example: For a calculator program that does addition, subtraction, multiplication, division, square root and comparison, each of these operations can be developed as separate procedures. In the main program each procedure would be invoked on the basis of user's choice.

Object-oriented Programming

Here the solution revolves around entities or objects that are part of problem. The solution deals with how to store data related to the entities, how the entities behave and how they interact with each other to give a cohesive solution.

Example: If we have to develop a payroll management system, we will have entities like employees, salary structure, leave rules, etc. around which the solution must be built.

Functional Programming

Here the problem, or the desired solution, is broken down into functional units. Each unit performs its own task and is self-sufficient. These units are then stitched together to form the complete solution.

Example: A payroll processing can have functional units like employee data maintenance, basic salary calculation, gross salary calculation, leave processing, loan repayment processing, etc.

Logical Programming

Here the problem is broken down into logical units rather than functional units. **Example**: In a school management system, users have very defined roles like class teacher, subject teacher, lab assistant, coordinator, academic in-charge, etc. So the

software can be divided into units depending on user roles. Each user can have different interface, permissions, etc.

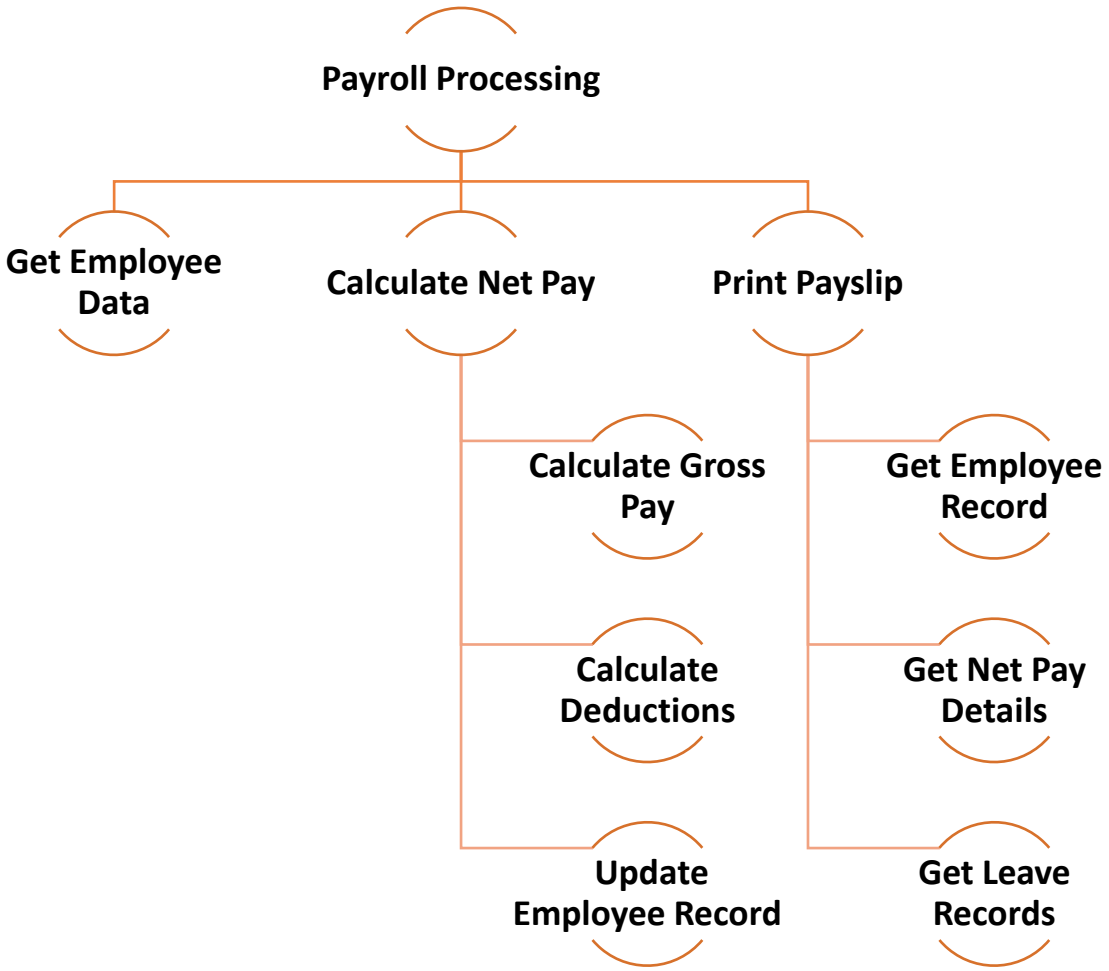
Software developers may choose one or a combination of more than one of these methodologies to develop a software. Note that in each of the methodologies discussed, problem has to be broken down into smaller units. To do this, developers use any of the following two approaches:

- Top-down approach
- Bottom-up approach

Top-down or Modular Approach

The problem is broken down into smaller units, which may be further broken down into even smaller units. Each unit is called a **module**. Each module is a self-sufficient unit that has everything necessary to perform its task.

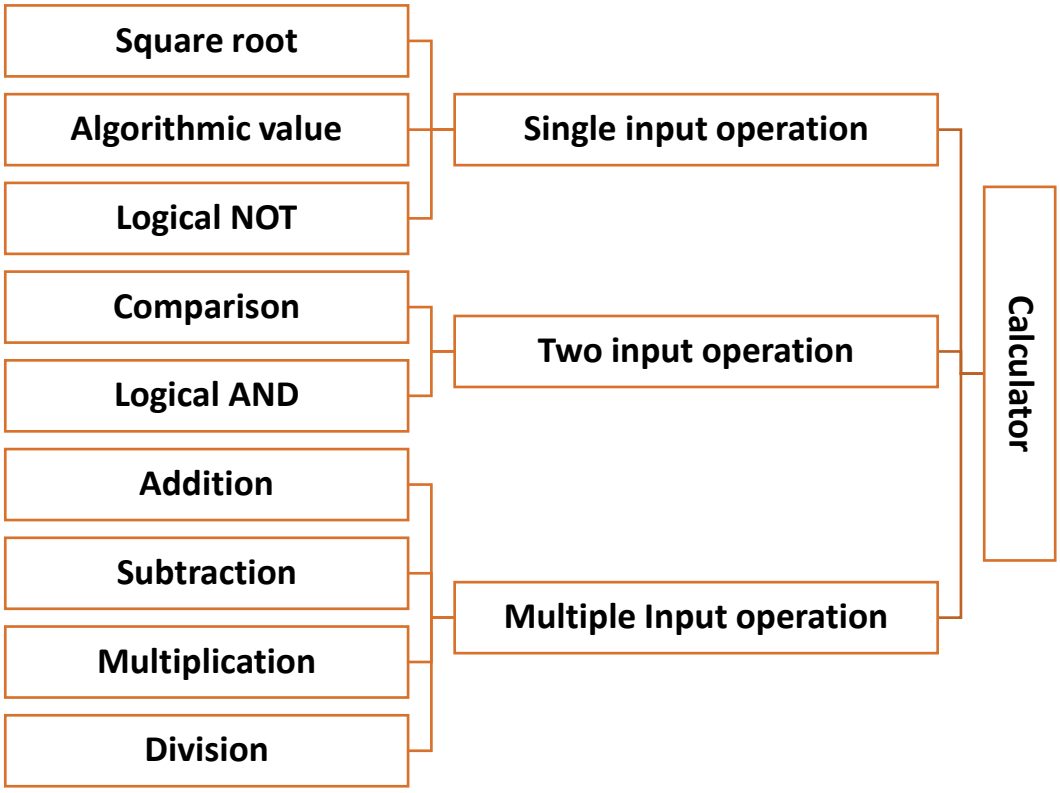
The following illustration shows an example of how you can follow modular approach to create different modules while developing a payroll processing program.



Bottom-up Approach

In bottom-up approach, system design starts with the lowest level of components, which are then interconnected to get higher level components. This process continues till a hierarchy of all system components is generated. However, in real-life scenario it is very difficult to know all lowest level components at the outset. So bottoms up approach is used only for very simple problems.

Let us look at the components of a calculator program.



2. Programming Methodologies — Understanding the Problem

A typical software development process follows these steps:

- Requirement gathering
- Problem definition
- System design
- Implementation
- Testing
- Documentation
- Training and support
- Maintenance

The first two steps assist the team in understanding the problem, the most crucial first step towards getting a solution. Person responsible for gathering requirement, defining the problem and designing the system is called **system analyst**.

Requirement Gathering

Usually, clients or users are not able to clearly define their problems or requirements. They have a vague idea of what they want. So system developers need to gather client requirements to understand the problem that needs to be resolved, or what needs to be delivered. Detailed understanding of the problem is possible only by first understanding the business area for which the solution is being developed. Some key questions that help in understanding a business include:

- What is being done?
- How is it being done?
- What is the frequency of a task?
- What is the volume of decisions or transactions?
- What are the problems being encountered?

Some techniques that help in gathering this information are:

- Interviews
- Questionnaires
- Studying existing system documents
- Analyzing business data

System analysts needs to create clear and concise but thorough requirements document in order to identify SMART – specific, measurable, agreed upon, realistic and time-based – requirements. A failure to do so results in:

- Incomplete problem definition
- Incorrect program goals
- Re-work to deliver required outcome to client
- Increased costs
- Delayed delivery

Due to the depth of information required, requirement gathering is also known as **detailed investigation**.

Problem Definition

After gathering requirements and analyzing them, problem statement must be stated clearly. Problem definition should unambiguously state what problem or problems need to be solved. Having a clear problem statement is necessary to:

- Define project scope
- Keep the team focused
- Keep the project on track
- Validate that desired outcome was achieved at the end of project

End of ebook preview

If you liked what you saw…

Buy it from our store @ **https://store.tutorialspoint.com**