PABNA UNIVERSITY OF SCIENCE & TECHNOLOGY

DEPARTMENT OF

INFORMATION AND COMMUNICATION ENGINEERING

FACULTY OF ENGINEERING AND TECHNOLOGY

Lab Report

Course Title: System Analysis and Software Testing Sessional

Course Code: ICE-4204

Submitted By:	*Submitted To:*
Name: Gulam Mustofa **Roll: 200615** **Session: 2019-20** **4th Year 2nd Semester,** **Department of ICE,** **Pabna University of Science and Technology.**	**Md. Anwar Hossain** **Professor,** **Department of ICE,** **Pabna University of Science and Technology.**

Submission Date: 20-05-2025 **Signature**

INDEX

Problem No: 01

Problem Name: Write a program in "JAVA" or "C" to develop a simple calculator that would be able to take a number, an operator (addition/subtraction/multiplication/ division/modulo) and another number consecutively as input and the program will display the output after pressing "=" sign. Sample input: 1+2; 8%4; Sample output: 1+2=3; 8%4=0.

Objective

Using Java or C, the objective of this lab task is to create and construct a basic calculator that can carry out addition, subtraction, multiplication, division, and modulo. An expression should be entered into the calculator, which should then determine the operator, carry out the appropriate operation, and show the outcome in an understandable manner (1+2=3). Additionally, this exercise helps with the comprehension of basic programming ideas like conditional statements, error handling, and input parsing. Basic software testing will also be carried out to guarantee that the calculator operates correctly.

Algorithm

1. Start the program
2. Prompt the user to enter a simple arithmetic expression (e.g., 5+3, 8%4).
3. Read the input expression as a string.
4. Parse the string to separate.
5. Use a conditional structure (such as if-else or switch-case) to identify the operator and perform the corresponding operation.
6. Store the result of the operation.
7. Display the full expression along with the result
8. Perform testing with different inputs to verify
9. End the program.

Source Code:

```
#include<bits/stdc++.h>
using namespace std;
int main(){
        double num1,num2;
        cout<<"Enter num1: "; cin>>num1;
        cout<<"Enter num2: "; cin>>num2;
        char charecter,press;
        cout<<"Enter +,-,*,/,%: ";  cin>>charecter;
   cout<<"Enter =: ";  cin>>press;

        if(press=='='){
                if(charecter=='+'){
                        cout<<num1<<"+"<<num2<<" = "<<num1+num2<<endl;
                }
```

```
                else if(charecter=='-'){
                        cout<<num1<<"-"<<num2<<" = "<<num1-num2<<endl;
                }
                else if(charecter=='*'){
                        cout<<num1<<"*"<<num2<<" = "<<num1*num2<<endl;
                }
                else if(charecter=='/'){
                        cout<<num1<<"/"<<num2<<" = "<<num1/num2<<endl;
                }
                else{
                        cout<<num1<<"%"<<num2<<" = "<<int(num1)%int(num2)<<endl;
                }
        }
}
```

Output:

Enter num1: 6

Enter num2: 7

Enter +,-,*,/,%: *

Enter =: =

6*7 = 42

Problem No:02

Problem Name: Write a program in "JAVA" or "C" that will take two 'n' integers as input until a particular operator and produce 'n' output.Sample input: 4 5 7 8 20 40 +; Sample output: 9 15 60.

Objective:

This lab experiment's goal is to create a Java or C program that accepts two sets of n integers (for example, 4 5 7 and 8 20 40) and an arithmetic operator (for example, +, -, *, /, %) as input. The program then operates element-wise on the respective pairs of numbers. The operator will then be applied to each pair, and the program will show n outputs. Understanding loop structures, array handling, input parsing, and using logic for batch arithmetic operations in Java or C will all be aided by this exercise.

Algorithm:

1. Start program
2. Prompt the user to enter a sequence of 2n integers followed by an operator (e.g., 4 5 7 8 20 40 +).
3. Read the full input as a string or tokenized input.
4. Separate the input into:
    a. First n number
    b. Second n number
    c. Input operator
5. Store the numbers in two arrays
6. Perform the loop for n integer value
7. Store and display the result for each pair
8. End the program

Source Code:

```
#include <bits/stdc++.h>

using namespace std;

int main()

{

   int n;

   cout<<"Enter number: ";    cin>>n;

   vector<double>vec(2*n);
```

```
    int j = 2*n;
    cout<<"Enter numbers: "<<endl;
    for(int i=0; i<j; i++)
        cin>>vec[i];
    char ch;
    cout<<"Enter operator: ";    cin>>ch;
    if(ch == '+'){
        for(int i=0; i<j; i+=2){
            cout<<vec[i]<<" + "<<vec[i+1]<<" = "<<vec[i] + vec[i+1]<<endl;}}
    if(ch == '-'){
        for(int i=0; i<j; i+=2){
            cout<<vec[i]<<" - "<<vec[i+1]<<" = "<<vec[i] - vec[i+1]<<endl;}}
    if(ch == '*')    {
        for(int i=0; i<j; i+=2)        {
            cout<<vec[i]<<" * "<<vec[i+1]<<" = "<<vec[i] * vec[i+1]<<endl;}}
    if(ch == '/')    {
        for(int i=0; i<j; i+=2)        {
            cout<<vec[i]<<" / "<<vec[i+1]<<" = "<<vec[i] / vec[i+1]<<endl;}}
    return 0;
}
```

Input:

```
Enter number: 3
Enter numbers:
4  5
7  8
```

Output

```
Enter operator: +
4 + 5 = 9
7 + 8 = 15
20 + 40 = 60
```

Problem No: 03

Problem Name: Write a program in "JAVA" or "C" to check weather a number or string is palindrome or not.

N.B: Your program must not take any test case number such as 1 or 2 for the desired cases from the user. Program users will insert a number or string as input directly and the program will display the exact result in the output console.

Objective

The objective of this lab experiment is to create a program in Java or C that checks whether a given number or string is a palindrome. A palindrome is a sequence that reads the same backward as forward. The program should directly accept input from the user, process the input, and print whether it is a palindrome or not without asking for a specific test case number. This task helps in understanding string manipulation, conditional logic, and the use of loops in programming.

Algorithm:

1. Start
2. Prompt the user to enter a string or number as input.
3. Read the input as a string.
4. Store the original input in a variable .
5. Reverse the input string:
    a. You can do this by iterating from the end of the string to the beginning and storing characters in a new string (e.g., reversed).
6. Compare the original string with the reversed string:
    a. If they are the same, print "Input is a palindrome".
    b. Otherwise, print "Input is not a palindrome".
7. End

Source Code:

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
   string string1, string2, number1,number2;
   int number;
   char ch;
   cout<<"Enter 's' for string checking or 'n' for number checking: ";    cin>>ch;
   if(ch == 's'){
      cout<<"Enter a word without space: ";        cin>>string1;
      string2 = string1;
      reverse(string2.begin(), string2.end());
      if(string1 == string2){
         cout<<"The given word is palindrome."<<endl;
      }
      else{
         cout<<"The given word is not palindrome."<<endl;
      }
   }
   if(ch == 'n'){
      cout<<"Enter number without space: ";        cin>>number;
      number1 = to_string(number);
      number2 = number1;
      reverse(number2.begin(), number2.end());
      if(number1 == number2){
         cout<<"The given number is palindrome."<<endl;
      }
      else{
         cout<<"The given number is not palindrome."<<endl;
```

```
        }
    }
    return 0;
}
```

Output:

Enter 's' for string checking or 'n' for number checking: n

Enter number without space: 567765

The given number is palindrome.

Problem No: 04

Problem Name: Write down the ATM system specifications and report the various bugs.

ATM System Specifications

To provide a comprehensive overview, this section outlines the specifications of a hypothetical **ATM (Automated Teller Machine)** system, followed by a discussion of potential bugs that may arise within such a system.

1. Authentication

a. The user inserts their ATM card into the machine.

b. The system prompts the user to enter their Personal Identification Number (PIN).

c. The PIN is validated against the bank's database. If the PIN is incorrect, the user is allowed for up to **three attempts** before the card is **blocked**.

2. Menu Options

After successful authentication, the user is presented with a set of options:

a. Withdraw Cash

b. Deposit Cash

c. Check Balance

d. Transfer Funds

e. Change PIN

f. Mini Statement

g. Exit

3. Cash Withdrawal

a. The user selects the withdrawal option and enters the desired amount.

b. The system checks the user's account balance.

c. If the balance is sufficient, the system verifies whether the ATM has enough cash to dispense.

d. If valid, the amount is deducted and cash is dispensed.

e. A printed receipt is provided if requested.

4. Cash Deposit

a. The user selects the deposit option and enters the amount.

b. The ATM prompts the user to insert cash or checks.

c. The deposited amount is verified and added to the user's account.

d. A printed receipt is provided if requested.

5. Balance Inquiry

a. The user selects the balance inquiry option.

b. The ATM retrieves and displays the current account balance from the bank's database.

6. Funds Transfer

a. The user selects the transfer option and enters the recipient’s account number and amount.

b. The system verifies both the recipient’s account and the sender’s available balance.

c. If valid, the amount is transferred to the recipient’s account.

7. PIN Change

a. The user selects the PIN change option.

b. The user is prompted to enter their current PIN followed by a new PIN.

c. If the current PIN is valid, the new PIN is updated in the system.

8. Mini Statement

a. The user selects the mini-statement option.

b. The ATM prints a mini-statement showing the last few transactions.

9. Error Handling and Security

a. In case of errors or hardware failures, the system **rolls back** the transaction and displays an appropriate **error message**.

b. Security features include:

   i. Card retention for blocked cards

   ii. Data encryption

   iii. Secure transaction logging

Potential Bugs in the ATM System

1. Authentication Bugs

a. **Incorrect PIN Lock**: The system locks the user out before allowing all 3 attempts.

b. **PIN Bypass**: The system allows access without valid PIN verification.

c. **Card Reading Issues**: Valid cards are wrongly identified as invalid due to hardware malfunction.

2. Transaction Bugs

a. **Cash Dispense Failure**: Cash is not dispensed, but the account is still debited.

b. **Incorrect Deduction**: Incorrect amount deducted during withdrawal or transfer.

c. **Receipt Error**: Receipt is missing or shows wrong transaction details.

3. Interface Bugs

a. **Wrong Navigation**: System navigates to incorrect menu options.

b. **Unresponsive Interface**: Interface freezes or lags during user input.

c. **Misleading Messages**: Confusing or incorrect confirmation/error messages.

4. Cash Deposit Bugs

- **Miscounted Deposits**: The system miscounts or fails to detect deposited items.
- **Duplicate Credits**: Deposit is credited more than once.

5. Balance Inquiry Bugs

a. **Incorrect Display**: Balance shown is outdated or incorrect due to sync issues.

6. Funds Transfer Bugs

a. **Wrong Account Transfer**: Funds go to an incorrect recipient even with valid details.

b. **Duplicate Transfer**: Multiple transfers processed for a single request.

7. PIN Change Bugs

a. **Failed Update**: New PIN not saved properly.

b. **Weak PIN Handling**: No enforcement of strong PIN rules (length, characters).

8. Security Bugs

a. **Data Exposure**: Sensitive data shown on screen or receipts.

b. **Transaction Replay**: Unauthorized repeat of previous transactions.

c. **Logging Errors**: Failed transactions recorded as successful and vice versa.

9. Hardware/Network Bugs

a. **Card Retention Failure**: ATM fails to retain blocked or expired cards.

b. **Network Issues**: Frequent disconnections cause incomplete transactions.

c. **Device Malfunction**: Dispenser, printer, or card reader errors.

10. User Experience Bugs

a. **Poor Accessibility**: Interface is not user-friendly for people with disabilities.

b. **Unclear Instructions**: On-screen instructions are vague or confusing.

These bugs can severely affect the **functionality**, **security**, and **user satisfaction** of the ATM system. Therefore, **thorough software testing, routine maintenance, and security auditing** are essential to ensure reliable and secure ATM operations.

Bug ID	Bug Description	Type	Severity	Reproducible?
ATM-B001	System accepts wrong PIN after 3 invalid attempts	Logical	High	Yes
ATM-B002	Balance displays incorrect amount after withdrawal	Functional	High	Yes
ATM-B003	System crashes when special characters are entered in PIN	Input Validation	Medium	Yes
ATM-B004	Mini-statement shows duplicate transactions	Data Handling	Medium	Yes
ATM-B005	Withdrawal amount exceeds balance but is still allowed	Logical	Critical	Yes
ATM-B006	No session timeout implemented	Security	Medium	Yes

Bug ID	Bug Description	Type	Severity	Reproducible?
ATM-B007	User can deposit negative amount	Input Validation	High	Yes
ATM-B008	Font size too small for display	UI/UX	Low	Yes
ATM-B009	ATM does not return to home screen after transaction	Navigation	Medium	Yes
ATM-B010	Incorrect error message displayed on insufficient funds	Functional	Low	Yes

Problem No: 05

Problem Name: Write a program in "JAVA" or "C" to find out the factorial of a number using while or for loop. Also verify the results obtained from each case.

Objective:

The objective of this experiment is to write a program in Java or C to calculate the factorial of a given number using loops. The program should compute the factorial using both types of loops separately and verify that both give the same result. This helps to understand loop control structures, logic development, and testing in programming.

Algorithm:

Using for loop:

1. Start
2. Input the number n
3. Initialize fact = 1
4. Loop from i = 1 to n
    - Multiply fact = fact * i
5. Print fact
6. End

Source Code:

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
   int number;
   cout<<"Enter a number to find factorial: ";    cin>>number;
   long long int fact=1;
   int j=1;
   for(int i = number; i>0; i--)
   {
      cout<<"Factorial in iteration "<<j<<" is : "<<fact<<" * "<<i<<" = "<<fact<<endl;
      fact = fact * i;
      j +=1;
   }
   cout<<endl<<number<<"! = "<<fact<<endl;
   return 0;
}
```

Output:

Enter a number to find factorial: 10

Factorial in iteration 1 is : 1 * 10 = 1

Factorial in iteration 2 is : 10 * 9 = 10

Factorial in iteration 3 is : 90 * 8 = 90

Factorial in iteration 4 is : 720 * 7 = 720

Factorial in iteration 5 is : 5040 * 6 = 5040

Factorial in iteration 6 is : 30240 * 5 = 30240

Factorial in iteration 7 is : 151200 * 4 = 151200

Factorial in iteration 8 is : 604800 * 3 = 604800

Factorial in iteration 9 is : 1814400 * 2 = 1814400

Factorial in iteration 10 is : 3628800 * 1 = 3628800

10! = 3628800

Problem No: 06

Problem Name: Write a program in "JAVA" or "C" that will find sum and average of array using do while loop and 2 user defined function.

Objective:

The objective of the program is to utilize a do-while loop and two user-defined functions to get the average and sum of the entries in an array. This teaches students how to divide jobs into modular functions in C or Java and how to use do-while to control loops.

Algorithm:

1. Start the program.
2. Input the number of elements n for the array.
3. Declare an array of size n.
4. Input all elements of the array.
5. Define the function
6. Call the first user-defined function sum():
    a. Initialize sum = 0
    b. Use do-while loop to iterate through the array
    c. Add each element to sum
    d. Return sum
7. Call the second user-defined function avg ():
    a. Accept the sum and n as parameters
    b. Compute average as average = sum / n
    c. Return average
8. Display the sum and average
9. End the program

Source Code:

```
#include <bits/stdc++.h>
using namespace std;


void sum(vector<int>arr1){
    int i = 0, sum=0;
    do{
        sum = sum + arr1[i];
        i++;
    }while(i<arr1.size());
    cout<<"Sum : "<<sum;
}
void avg(vector<int>arr1){
    int i = 0;
    float sum =0;
    do{
        sum = sum + arr1[i];
        i++;
    }while(i<arr1.size());
    cout<<"Average : "<<sum/arr1.size();
}
int main(){
    int n;
    cout<<"Enter array size: ";    cin>>n;
    vector<int>arr(n);
    cout<<"Enter array elements: ";
    for(int i = 0; i<n; i++){cin>>arr[i];}
    cout<<endl;
    sum(arr);
    cout<<endl;
```

```
    avg(arr);
    return 0;
}
```

Output:

Enter array size: 5

Enter array elements: 9 8 7 6 5

Sum : 35

Average : 7

Problem No: 07

Problem Name: Write a simple "JAVA" program to explain classNotFound Exception and endOfFile(EOF) exception.

Objective:

The objective is to understand Java exception handling by demonstrating how to manage ClassNotFoundException when loading a class dynamically and EOFException when reading past the end of a file, using a simple program that handles these exceptions gracefully.

Algorithm:

1. Import necessary I/O and utility classes.
2. Begin the program execution.
3. Attempt to load a class at runtime.
    a. Handle exception if the class is not found.
4. Open a stream to read data.
5. Read data in a loop until the end is reached.
    a. Handle exception when end of data is encountered.
6. Close the stream and end the program.

Source Code:

```
package Study;
import java.io.*;
public class CEException {
   public static void main(String args[]) throws Exception {
      try {
            Class.forName("Study.ExistClass");
            System.out.println("Class Found inside package.\n---------");
         } catch(ClassNotFoundException e) {
               System.out.println("CLassNotFoundException");
         }
            System.out.println("EOF exception for output ");
            DataInputStream dis = new DataInputStream(new FileInputStream("input.txt"));
            while(true) {
               try{
                     byte ch;
                     ch = dis.readByte();
                     System.out.print((char)ch);
               } catch(EOFException e) {
                     System.out.println("\nEnd of file reached");
                     break;
               }
            }
            dis.close();
      }
}
```

```
//Other class
package Study;
public class ExistClass
{
            public static void getDepartmentName()
            {
               System.out.println("Information and Communication Engineering");
            }
}
```

Output:

Problem No: 08

Problem Name: Write a program in "JAVA" or "C" that will read a input.txt file containing n positive integers and calculate addition, subtraction, multiplication and division in separate output.txt file.

Sample input: 5 5 9 8; Sample output: Case-1:10 0 25 1; Case-2: 17 1 72 1.

Objective:

The objective of this program is to read a series of positive integers from a file and perform basic arithmetic operations on them. These operations include addition, subtraction, multiplication, and division for each pair of integers. The results of these operations will be written into an output file in a clear and structured format. This helps in understanding how arithmetic operations can be performed in C programming. It also provides practical knowledge of how to handle files for reading input and writing output. Overall, the task enhances understanding of both computation and file management in C.

Algorithm:

1. Start the program.
2. Open the input file in read mode and the output file in write mode.
3. Check if files opened successfully.
4. Read two integers at a time from the input file.
5. For each pair:
    a. Perform addition, subtraction, multiplication, and division.
    b. Write the results to the output file with proper formatting.
6. Repeat steps 4–5 until all data is processed.
7. Close both files.
8. End the program.

Source Code:

```
#include <stdio.h>
#include <stdlib.h>

int main() {
    FILE *fin, *fout;
    int a, b, caseNum = 1;

    // Open input and output files
    fin = fopen("input.txt", "r");
    fout = fopen("output.txt", "w");

    if (fin == NULL || fout == NULL) {
        printf("Error opening file.\n");
        return 1;
    }

    // Read input in pairs
    while (fscanf(fin, "%d %d", &a, &b) == 2) {
        int sum = a + b;
        int diff = a - b;
        int prod = a * b;
        int div = (b != 0) ? a / b : 0; // Prevent division by zero

        fprintf(fout, "Case-%d: %d %d %d %d\n", caseNum, sum, diff, prod, div);
        caseNum++;
    }

    // Close files
    fclose(fin);
    fclose(fout);

    printf("Results written to output.txt successfully.\n");
    return 0;
}
```

Input: 5 5 9 8

Output:

Case-1: 10 0 25 1

Case-2: 17 1 72 1

Problem No: 09

Problem Name: Explain the role of software engineering in Biomedical Engineering and in the field of Artificial Intelligence and Robotics.

Role of Software Engineering in Biomedical Engineering:

Software engineering plays a critical role in biomedical engineering by providing the tools and methodologies needed to design, develop, and maintain software solutions for healthcare and medical applications. The integration of software engineering in biomedical engineering facilitates advancements in patient care, diagnosis, treatment, and medical research. Here are some key roles:

1. **Medical Device Software Development:** Biomedical engineers often work on designing and developing medical devices such as MRI machines, CT scanners, and pacemakers. Software engineering is crucial for developing the embedded systems and software that control these devices, ensuring they operate safely, effectively, and reliably.
2. **Clinical Decision Support Systems (CDSS):** Software engineering enables the development of CDSS, which helps healthcare providers make data-driven decisions. These systems leverage algorithms, machine learning, and data analytics to provide insights, predict patient outcomes, and suggest treatment plans.
3. **Health Informatics:** Software engineering is key in developing Electronic Health Records (EHR) systems, telemedicine applications, and health management systems. These systems manage patient data, enable remote consultations, and streamline communication between healthcare professionals.
4. **Medical Imaging and Signal Processing:** Biomedical engineers work on software that processes and analyzes medical images (e.g., MRI, CT scans) and biomedical signals (e.g., ECG, EEG). Advanced software engineering techniques, including machine learning and computer vision, are used to develop algorithms for image enhancement, segmentation, and disease detection.
5. **Wearable Health Technology and Mobile Health Apps:** The development of wearable devices (e.g., fitness trackers, glucose monitors) and mobile health applications relies heavily on software engineering for creating user interfaces, data collection, and real-time monitoring capabilities.
6. **Simulations and Modeling:** Software engineering facilitates the creation of simulations and models of biological systems, which are essential for understanding complex physiological processes and conducting virtual experiments.

Role of Software Engineering in Artificial Intelligence and Robotics:

Software engineering is foundational in the fields of Artificial Intelligence (AI) and Robotics, as it involves creating software that can learn, adapt, and perform tasks autonomously. Key roles include:

1. **Development of AI Algorithms and Models:** Software engineering provides the frameworks and tools necessary to design, develop, and deploy AI algorithms, including machine learning models, deep learning networks, natural language processing, and computer vision systems. Engineers must ensure these models are efficient, scalable, and secure.
2. **Robotics Control Software:** Robotics relies on software to control and manage robot behavior, including motion planning, manipulation, navigation, and perception. Software engineering ensures that robotic control systems are robust, real-time, and capable of adapting to different environments.
3. **Integration of Sensors and Actuators:** In robotics, software engineering plays a crucial role in integrating various sensors (e.g., LIDAR, cameras, gyroscopes) and actuators (e.g., motors, grippers). This involves writing low-level drivers, data processing algorithms, and middleware to ensure seamless communication between hardware components.
4. **AI-Driven Robotics:** Software engineering is critical in developing AI-driven robotic systems that can perceive their environment, make decisions, and perform tasks autonomously. This includes implementing reinforcement learning, computer vision, and natural language understanding to enable robots to interact intelligently with humans and their environment.
5. **Simulation and Testing Environments:** Robotics and AI development often require extensive simulation and testing before deploying in real-world environments. Software engineering is involved in creating realistic simulators and virtual environments where robots and AI models can be trained and tested.
6. **Ethics, Security, and Compliance:** Both AI and robotics require adherence to ethical guidelines, privacy considerations, and security protocols. Software engineers are responsible for implementing data privacy measures, cybersecurity protocols, and ensuring compliance with legal and ethical standards.
7. **Human-Robot Interaction (HRI):** Developing intuitive and safe interfaces for human-robot interaction is a key area where software engineering plays a crucial role. It involves creating user-friendly interfaces, real-time feedback systems, and interactive software that can adapt to user behaviors.

Conclusion

In both Biomedical Engineering and AI/Robotics, software engineering is indispensable for developing robust, efficient, and secure systems that meet the complex requirements of these fields. It provides the foundation for innovation, enabling advanced healthcare solutions and intelligent robotic systems that can revolutionize the way we interact with technology and improve human life.

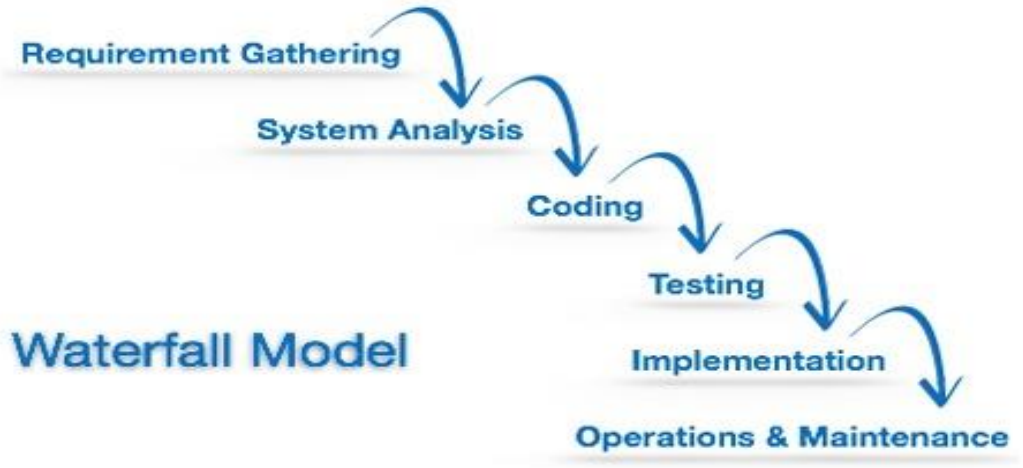
Problem No: 10

Problem Name: Study the various phases of Water-fall model. Which phase is the most dominated one?

Objective:

- To understand the working of the Waterfall model in software development.
- To study the various sequential phases involved in the Waterfall model.
- To identify and analyze the most dominant phase of the Waterfall model.
- To learn how each phase depends on the completion of the previous phase.

Theory: The **Waterfall Model** is a **linear and sequential software development process.** Each phase must be completed before the next begins. Below are the standard phases:



1. **Requirement Analysis:**
   This phase involves collecting and documenting all software requirements from the client. It defines what the system should do and ensures clarity for the development process.
2. **System Design:**
   Based on the gathered requirements, system architecture and module-level designs are created. This serves as a blueprint for coding.
3. **Implementation (Coding):**
   The design is translated into source code using suitable programming languages. Each module is developed and tested individually.
4. **Integration and Testing:**
   All modules are combined and tested as a complete system. This phase checks for defects and ensures the software meets requirements.
5. **Deployment:**
   The fully tested software is delivered and installed in the user environment. It becomes operational for end users.
6. **Maintenance:**
   Post-deployment, the software is monitored, bugs are fixed, and updates or enhancements are applied as needed.

Most Dominant Phase:
The most important phase is **Requirement Analysis** because it sets the foundation for the entire project. If requirements are unclear or incorrect, the following phases will be affected, often leading to costly errors. Since the Waterfall Model is sequential, mistakes in this phase are hard to fix later, making it the most critical stage.

Problem No: 11

Problem Name: Using COCOMO model estimate effort for specific problem in industrial domain

The **COCOMO (Constructive Cost Model)** is a widely used software cost estimation model that helps in calculating the required **effort**, **time**, and **cost** of a software development project based on its size and complexity. It is especially useful in **industrial domains**, where accurate planning and resource allocation are critical due to fixed requirements and stringent constraints.

Types of COCOMO Models:

1. **Basic COCOMO** – Provides a quick estimate based on project size (KLOC).
2. **Intermediate COCOMO** – Adds cost drivers like reliability, complexity, and team experience.
3. **Detailed COCOMO (COCOMO II)** – Includes all cost drivers plus factors such as development tools, reuse, and team capability.

Basic COCOMO Formula:

The **effort** (in person-months) is calculated as:

$$\text{Effort (E)} = a \times (\text{KLOC})^b$$

Where:

- **KLOC** = Thousands of Delivered Source Lines of Code
- **a** and **b** = constants depending on the project type

Project Type	a	b
Organic	2.4	1.05
Semi-Detached	3.0	1.12
Embedded	3.6	1.20

Given Problem:

- **Domain:** Industrial
- **Project Type:** Semi-Detached (moderately complex with some experience and constraints)
- **Project Size:** 50 KLOC

Calculation:

Using the **semi-detached** formula:

$$\text{Effort (E)} = 3 \times (50)^{1.12}$$

Result:

Estimated Effort = 239.87 person-months

This means that **approximately 240 person-months** of effort are needed to complete the software project under the **Basic COCOMO model** assuming it is a **semi-detached industrial project**.

Conclusion:

The Basic COCOMO model provides a quick and effective way to estimate the effort required for software development. For a 50 KLOC semi-detached industrial project, **about 240 person-months** would be required. For more accuracy, factors such as team capability, software reliability, and tool support should be included using the **Intermediate or Detailed COCOMO models**.

Problem No: 12

Problem Name: Identify the reasons behind software crisis and explain the possible solutions for the following scenario:

Case 1: "Air ticket reservation software was delivered to the customer and was installed in an airport 12.00 AM (mid night) as per the plan. The system worked quite fine till the next day 12.00 PM (noon). The system crashed at 12.00 PM and the airport authorities could not continue using software for ticket reservation till 5.00 PM. It took 5 hours to fix the defect in the software".

Case 2: "Software for financial systems was delivered to the customer. Customer conformed the development team about a mal-function in the system. As the software was huge and complex, the development team could not identify the defect in the software".

Solution:

The term **"software crisis"** refers to the major challenges faced during software development, especially between the 1960s and 1980s. During that time, many software projects were delivered late, went over budget, or failed to meet user expectations. The main causes of this crisis include:

1. **High Complexity:** As software systems became larger and more advanced, they became harder to design, build, and maintain.
2. **Weak Requirement Analysis:** Many projects started with unclear or incomplete requirements, leading to software that did not satisfy user needs.
3. **Poor Project Management:** Bad planning, missing documentation, and poor resource allocation often caused delays and failures.
4. **Lack of Proper Testing:** Releasing software without thoroughly testing it led to critical bugs and system failures.
5. **Changing Requirements:** Requirements often changed during development, making it hard for developers to keep up.
6. **No Common Standards:** The absence of universal standards resulted in inconsistent development practices and product quality.
7. **Communication Issues:** Poor communication among developers, clients, and users led to misunderstandings about software features and expectations.

Case Study 1: Air Ticket Reservation System

Scenario:
An air ticket booking system was running well but suddenly crashed at 12:00 PM the following day and remained down for five hours. This points to a **time-related bug** that activated at a specific moment.

Possible Causes:

1. **Time-Based Error:** A bug might have been triggered exactly at 12:00 PM due to boundary conditions or overflow.
2. **Limited Testing:** The system might not have been tested for scenarios that involve time-specific functions.
3. **Resource Leakage:** A memory or system resource leak could have caused the crash after extended use.
4. **Weak Error Handling:** The system may not have been designed to handle unexpected errors correctly.
5. **Database Problem:** It could be a database issue like connection failure or data corruption occurring at that time.

Suggested Solutions:

1. **Thorough Testing:** Conduct time-bound, performance, and stress tests to catch these issues early.
2. **Code Review & Debugging:** Go through the code carefully to identify the root of the time-related error.
3. **Logging and Monitoring:** Add system logs and monitoring tools to help trace the problem when it happens.
4. **Stronger Error Handling:** Improve the code to handle unexpected errors without crashing the system.
5. **Backup Systems:** Use failover systems to keep services running even if the main system goes down.

Case Study 2: Financial Software System

Scenario:
A financial application delivered to the client had a malfunction, but due to its complexity, the development team was unable to identify the issue quickly.

Possible Causes:

1. **Lack of Modularity:** The software may have been built as one large unit, making it hard to isolate problems.
2. **Poor Documentation:** Without proper documentation, understanding the system's flow and structure is difficult.
3. **Bad Maintenance Design:** The system might not have been built with easy maintenance in mind.
4. **Complicated Dependencies:** Too many connections between different parts of the system can hide where the error is.
5. **Limited Testing:** The testing might not have covered all use cases or edge scenarios.

Suggested Solutions:

1. **Use Modular Design:** Break the system into smaller parts to make debugging and updates easier.
2. **Better Documentation:** Maintain complete documentation that explains the system’s architecture and components.
3. **Automated Testing:** Set up automated tests to catch issues early and ensure continuous improvement.
4. **Advanced Debugging Tools:** Use tools like static code analysis, runtime logs, and debugging software to track down issues.
5. **Team Training:** Train the development team in modern techniques like modularization and large system maintenance.
6. **Work with the Client:** Collaborate with the customer to understand how the issue happens and find a solution faster.

Conclusion

By identifying the core problems and applying proper solutions, software teams can better manage such failures. Addressing issues like complexity, weak documentation, and poor testing practices is crucial to avoiding future software crises and delivering reliable, efficient systems.