PABNA UNIVERSITY OF SCIENCE AND TECHNOLOGY

FACULTY OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF INFORMATION AND COMMUNICATION ENGINEERING

LAB REPORT

COURSE CODE: ICE-4204

COURSE TITLE: System Analysis and Software Testing Sessional

SUBMITTED BY

Naima Islam

Roll: 200601

Session: 2019-20

4th year 2nd Semester

Department of Information and Communication Engineering, Pabna University of Science and Technology.

SUBMITTED TO

Md. Anwar Hossain

Professor

Department of Information and Communication Engineering, Pabna University of Science and Technology.

DATE OF SUBMISSION: 21-05-2025

SIGNATURE

Index

Problem: 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.

Objectives: To Implement a simple calculator that can handle basic arithmetic operations: addition, subtraction, multiplication, division, and modulo.

Theory: A simple calculator program performs basic arithmetic operations such as addition, subtraction, multiplication, division, and modulo. It takes user input in the form of an expression like number1 operator number2 =, for example, 1+2=. The program reads this input, extracts the two numbers and the operator, and then performs the corresponding calculation. Based on the operator provided, it uses conditional logic to determine the correct operation and finally displays the complete expression along with the result. This type of program helps in understanding input handling, expression parsing, and basic control structures in programming.

Algorithm: The algorithm of this problem is writing in below:

1. Start the program.
2. Declare three variables:
    - Two integers: num1, num2 to store the numbers.
    - One character to store the operator (+, -, *, /, %).
    - One character press to store the = sign.
3. Prompt the user to:
    - Enter the first number → store in num1.
    - Enter the operator → store in character.
    - Enter the second number → store in num2.
    - Enter the equal sign (=) → store in press.
4. Check if the entered character is =:
    - If true, perform the arithmetic operation based on the operator:
        - If +, add the two numbers and print result.
        - If -, subtract and print result.
        - If *, multiply and print result.
        - If /, divide and print result (cast to float to handle decimals).
        - If %, perform modulo and print result.
5. End the program.

Code:

```
#include <stdio.h>

int main() {
   int num1, num2;
   char charecter, press;

   printf("Enter num1: ");
   scanf("%d", &num1);

   printf("Enter +,-,*,/,%c: ", '%');
```

```
    scanf(" %c", &charecter);

    printf("Enter num2: ");
    scanf("%d", &num2);

    printf("Enter =: ");
    scanf(" %c", &press);

    if (press == '=') {
        if (charecter == '+')
            printf("%d+%d = %d\n", num1, num2, num1 + num2);
        else if (charecter == '-')
            printf("%d-%d = %d\n", num1, num2, num1 - num2);
        else if (charecter == '*')
            printf("%d*%d = %d\n", num1, num2, num1 * num2);
        else if (charecter == '/')
            printf("%d/%d = %.2f\n", num1, num2, (float)num1 / num2);
        else
            printf("%d%%%d = %d\n", num1, num2, num1 % num2);
    }

    return 0;
}
```

Input: Enter num1: 65
Enter +,-,*,/,%: %
Enter num2: 55
Enter =: =
Output: 65%55 = 10

Problem: 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.

Objectives: To Understand how to process mixed input (integers and operators) in a single input stream.

Theory: This program is designed to take a sequence of 2n integers followed by an arithmetic operator and perform pairwise operations on each corresponding pair of integers. The input format includes numbers grouped in two sets of n values (e.g., 4 5 7 and 8 20 40) followed by an operator (e.g., +, -, *, /). The program reads and stores the first n integers and the next n integers separately, then applies the given operator to each pair (first of the first set with the first of the second set, and so on). It then outputs the result of each operation in sequence. This type of program demonstrates the use of arrays, loops, and basic arithmetic operations in programming.

Algorithm: The algorithm of this problem is writing in below:

1. Start the program.
2. Declare an integer array vec[] to store numbers and a count variable to track how many numbers were entered.
3. Use a character array input[] to accept mixed input (number/operator).
4. Prompt the user to enter values one by one.
5. In a while loop:
    - Use scanf to read the input as a string.
    - If the first character is an arithmetic operator (+, -, *, /, %):
        - Store it in variable op and break the loop.
    - Else, use sscanf to convert the string to an integer and store it in the array.
6. After input ends:
    - Loop over the array by steps of 2 (i.e., in pairs: vec[0] & vec[1], vec[2] & vec[3], etc.).
    - For each pair, perform the operation indicated by op and print the result.
    - If there is an odd number of inputs, show a warning for the last unmatched value.
7. End the program.

Code:

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

int main() {
   int vec[100], count = 0;
   char input[20];
   char op;

   printf("Enter numbers one by one (operator to stop):\n");

   while (1) {
      scanf("%s", input);
```

```
        if (input[0] == '+' || input[0] == '-' || input[0] == '*' || input[0] == '/' || input[0] == '%') {
            op = input[0];
            break;
        }

        sscanf(input, "%d", &vec[count]);
        count++;
    }

    for (int i = 0; i < count; i += 2) {
        if (i + 1 >= count) {
            printf("Insufficient number for last operation.\n");
            break;
        }

        printf("%d %c %d = ", vec[i], op, vec[i + 1]);

        if (op == '+')
            printf("%d\n", vec[i] + vec[i + 1]);
        else if (op == '-')
            printf("%d\n", vec[i] - vec[i + 1]);
        else if (op == '*')
            printf("%d\n", vec[i] * vec[i + 1]);
        else if (op == '/')
            printf("%.2f\n", (float)vec[i] / vec[i + 1]);
        else
            printf("%d\n", vec[i] % vec[i + 1]);
    }

    return 0;
}
```

Input: Enter numbers one by one (operator to stop): 4 5 10 15 12 45 +
Output: 4 + 5 = 9
10 + 15 = 25
12 + 45 = 57

Problem Number: 03

Problem Name: Write a program in "JAVA" or "C" to check whether 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 user will insert a number or string as input directly and the program will display the exact result in the output console.

Objectives: To learn how to check for palindrome properties.

Theory: A palindrome is a number or string that reads the same forward and backward, such as "121" or "madam." This program is designed to take a single input from the user—either a number or a string—without asking for a separate test case identifier. The input is then checked by comparing characters from the beginning and end, moving toward the center. If all corresponding characters match, the input is considered a palindrome. This program demonstrates concepts like string manipulation, loops, and conditional checks in programming.

Algorithm: The algorithm of this problem is writing in below:

1. Start
2. Declare two character arrays:
   string1[100] – to store the original input
   string2[100] – to store the reversed version
3. Prompt the user to enter a number or string.
4. Read the input into string1.
5. Copy the contents of string1 to string2 using strcpy().
6. Reverse string2 by calling reverseStr():
   - Find the length of string2 using strlen().
   - Loop from index 0 to len/2:
     - Swap characters at positions i and len-1-i.
7. Compare string1 and string2 using strcmp():
   - If they are equal:
     - Print "It is palindrome."
   - Else:
     - Print "It is not palindrome."
8. End

Code:

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

void reverseStr(char* str) {
   int len = strlen(str);
   for (int i = 0; i < len / 2; i++) {
      char tmp = str[i];
      str[i] = str[len - 1 - i];
      str[len - 1 - i] = tmp;
   }
}
```

```
int main() {
   char string1[100], string2[100];
   printf("Enter a number or string: ");
   scanf("%s", string1);
   strcpy(string2, string1);
   reverseStr(string2);

   if (strcmp(string1, string2) == 0)
      printf("It is palindrome.\n");
   else
      printf("It is not palindrome.\n");

   return 0;
}
```

Input: Enter a number or string: nayan
Output: It is palindrome.

Problem Number: 04

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

Objectives: To understand the working and components of an Automated Teller Machine (ATM) system, simulate basic banking operations such as user authentication, cash withdrawal, and balance inquiry, and identify common system bugs and their causes.

Theory: An Automated Teller Machine (ATM) is an electronic banking device that enables customers to perform basic financial transactions without the need for a human teller. These transactions include withdrawing cash, checking account balances, depositing funds, transferring money between accounts, and printing mini statements. ATMs are accessible 24/7 and are connected to banking networks, allowing users to access their bank accounts using a debit or ATM card and a Personal Identification Number (PIN) for secure authentication.

ATM System Specifications:

- **Hardware Components:**
  - Card Reader: Reads data from ATM/debit cards using magnetic stripe or chip.
  - Keypad: For PIN entry and transaction inputs.
  - Display Screen: Provides user interface to display messages and instructions.
  - Cash Dispenser: Dispenses requested cash denominations.
  - Receipt Printer: Prints transaction receipts for users.
  - Security Camera: Monitors user activity for security.
  - Communication Module: Connects ATM to banking network over secure channels.
- **Software Components**
  - Operating System: Embedded OS (e.g., Windows Embedded, Linux).
  - ATM Application Software:
    - ✓ User authentication (PIN verification)
    - ✓ Transaction processing (withdrawal, deposit, balance inquiry)
    - ✓ Communication with bank server
  - Encryption Module: For secure PIN entry and data transmission.
  - Logging Module: For maintaining audit trails and error logs.
- **Functional Specifications**
  - User Login: Card + PIN-based authentication.
  - Cash Withdrawal: Allows users to withdraw up to a daily limit.
  - Balance Inquiry: Displays account balance.
  - Deposit Function: Accepts cash or cheque deposits (if supported).
  - Mini Statement: Shows recent transactions.
  - Multi-language Support: Optional for user convenience.
  - Timeout Handling: Session timeout for inactivity.

    - Error and Exception Handling: For hardware/software failures.
- **Security Specifications**
    - End-to-End Encryption: For data from user to bank server.
    - PIN Block Format: Uses secure ISO PIN block formats.
    - Tamper Detection: Triggers alerts if physical intrusion is detected.
    - Session Logs and Audit Trails: For fraud and transaction analysis.

Commonly Reported Bugs in ATM Systems:

Bug Type	Description
Card Retention Bug	ATM may retain user card unexpectedly or fail to eject.
Cash Dispense Error	Machine dispenses incorrect amount or fails to dispense.
Double Debit Bug	Account debited twice for one transaction due to communication failure.
PIN Entry Bug	Incorrect masking or input errors during PIN entry.
Screen Freeze	UI becomes unresponsive due to software crash or timeout handling error.
Receipt Printing Error	Blank receipts or no receipt printed.
Network Timeout	Transaction fails due to server connection issues.
Language Display Bug	Language switch causes broken text or incorrect instructions.
Security Flaws	Weak encryption or failure to log out after session ends.
Session Expiry Bug	Transaction proceeds after timeout or fails prematurely.

Problem Number: 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.

Objectives: To understand the concept of factorial and the loop structures.

Theory: A factorial program calculates the product of all positive integers up to a given number n, denoted as n!. For example, the factorial of 5 is $5! = 5\times4\times3\times2\times1 = 120$. This program takes a number as input and uses either a while loop or a for loop to compute the factorial iteratively. In each iteration, the program multiplies the current result with the loop variable until it reaches the input number. The result can then be verified by comparing it with a known factorial value or using different loop structures to ensure correctness. This program helps in understanding loop control and the concept of iterative computation in programming.

Algorithm: The algorithm of this problem is writing in below:

1. Start
2. Declare the following variables:
    - num as integer → to store the input number.
    - i as integer → to use in loops.
    - fact_for and fact_while as unsigned long long → to store factorial results.
3. Display message: "Enter a positive integer:"
4. Read the value of num from user input.

Factorial using For Loop:

5. Initialize fact_for = 1.
6. Print: "Calculating factorial using for loop:"
7. Loop from i = 1 to num (inclusive):
    - Multiply fact_for = fact_for × i
    - Print the current iteration and value of fact_for.

Factorial using While Loop:

8. Initialize i = 1 and fact_while = 1.
9. Print: "Calculating factorial using while loop:"
10. Repeat while i <= num:

- Multiply fact_while = fact_while × i
- Print the current iteration and value of fact_while
- Increment i by 1

Final Output and Verification:

11. Print the final values of:

- fact_for
- fact_while

12. Compare fact_for and fact_while:

- If they are equal → print "Both methods give the same result. Verified!"
- Else → print "Mismatch in results. Something went wrong!"

13. End

Code:

```
#include <stdio.h>

int main() {
    int num, i;
    unsigned long long fact_for = 1, fact_while = 1;

    // Take input
    printf("Enter a positive integer: ");
    scanf("%d", &num);

    // Factorial using for loop with iteration display
    printf("\nCalculating factorial using for loop:\n");
    for (i = 1; i <= num; i++) {
        fact_for *= i;
        printf("Iteration %d: fact_for = %llu\n", i, fact_for);
    }

    // Factorial using while loop with iteration display
    printf("\nCalculating factorial using while loop:\n");
    i = 1;
    while (i <= num) {
        fact_while *= i;
        printf("Iteration %d: fact_while = %llu\n", i, fact_while);
        i++;
    }

    // Final results
    printf("\nFinal Result:\n");
    printf("Factorial using for loop: %llu\n", fact_for);
    printf("Factorial using while loop: %llu\n", fact_while);

    // Verification
    if (fact_for == fact_while)
        printf("Both methods give the same result. Verified!\n");
    else
        printf("Mismatch in results. Something went wrong!\n");

    return 0;
}
```

Input: Enter a positive integer: 6
Output: Calculating factorial using for loop:
Iteration 1: fact_for = 1
Iteration 2: fact_for = 2
Iteration 3: fact_for = 6
Iteration 4: fact_for = 24
Iteration 5: fact_for = 120
Iteration 6: fact_for = 720
Calculating factorial using while loop:
Iteration 1: fact_while = 1

Iteration 2: fact_while = 2
Iteration 3: fact_while = 6
Iteration 4: fact_while = 24
Iteration 5: fact_while = 120
Iteration 6: fact_while = 720
Final Result:
Factorial using for loop: 720
Factorial using while loop: 720
Both methods give the same result. Verified!

Problem Number: 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.

Objectives: To learn how to calculate the sum and average of array elements using a do-while loop.

Theory: This program calculates the sum and average of elements in an array using a do-while loop and two user-defined functions. One function is responsible for computing the sum of all array elements, and the other calculates the average using the sum and the total number of elements. The program takes input from the user to populate the array, then processes the data using a do-while loop, which ensures that the loop runs at least once. This approach reinforces the use of loops, functions, and array manipulation, encouraging modular and structured programming practices.

Algorithm: The algorithm of this problem is writing in below:

1. Start the program.
2. Declare a variable n to store the size of the array.
3. Prompt the user to enter the array size and read the value into n.
4. Declare an array arr of size 100.
5. Prompt the user to enter n elements into the array.
6. Use a for loop to read n elements into the array arr.
7. Call the function sum(arr, n):
    - Inside the sum function:
        - Initialize sum = 0 and i = 0.
        - Use a do-while loop to iterate through the array:
            - Add each element arr[i] to sum.
            - Increment i.
        - Repeat the loop while i < n.
        - Print the total sum.
8. Call the function avg(arr, n):
    - Inside the avg function:
        - Initialize sum = 0 and i = 0.
        - Use a do-while loop to iterate through the array:
            - Add each element arr[i] to sum.
            - Increment i.
        - Repeat the loop while i < n.
        - Calculate the average by dividing the sum by n.
        - Print the average with two decimal precision.
9. End the program.

Code:

```
#include <stdio.h>

void sum(int arr[], int n) {
   int sum = 0, i = 0;
   do {
      sum += arr[i++];
   } while (i < n);
```

```
    printf("Sum : %d\n", sum);
}

void avg(int arr[], int n) {
    int i = 0, sum = 0;
    do {
        sum += arr[i++];
    } while (i < n);
    printf("Average : %.2f\n", (float)sum / n);
}

int main() {
    int n;
    printf("Enter array size: ");
    scanf("%d", &n);

    int arr[100];
    printf("Enter array elements: ");
    for (int i = 0; i < n; i++)
        scanf("%d", &arr[i]);

    sum(arr, n);
    avg(arr, n);

    return 0;
}
```

Input: Enter array size: 7
Enter array elements: 12 45 63 4 8 14 0
Output: Sum: 146
Average: 20.86

Problem Number: 07

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

Objectives: To understand and demonstrate the handling of ClassNotFoundException and EOFException in Java.

Theory: In Java, ClassNotFoundException and EOFException are two common exceptions that occur during runtime under specific conditions. ClassNotFoundException is thrown when the Java ClassLoader tries to load a class at runtime using its name (often through reflection) and the class cannot be found in the classpath. This usually happens in programs that use dynamic class loading. On the other hand, EOFException (End-Of-File Exception) is a checked exception that occurs when an input operation is attempted beyond the end of a file, typically when reading from a file or stream using classes like ObjectInputStream or DataInputStream. These exceptions help in identifying problems related to class availability and file reading boundaries, ensuring more reliable and controlled program execution.

Algorithm: The algorithm of this problem is writing in below:

1. Start
2. Try to Load a Class Dynamically
    - Use Class.forName("Study.ExistClass") to attempt to load a class named ExistClass from the Study package.
    - If class is found:
        - Print "Class Found inside package.\n---------".
    - If class is not found:
        - Catch the ClassNotFoundException and print "CLassNotFoundException".
3. Print: "EOF exception for output " to indicate that the program will now demonstrate end-of-file handling.
4. Try to Read from a File (input.txt)
    - Open the file input.txt using FileInputStream wrapped by DataInputStream.
    - Loop indefinitely:
        - Try to read one byte at a time using readByte().
        - Convert the byte to a character and print it.
        - If end-of-file is reached:
            - Catch the EOFException.
            - Print "\nEOFException caught: End of file reached.".
            - Break the loop.
    - Close the input stream (dis.close()).
5. Catch Exceptions Related to File Handling
    - If the file is not found (FileNotFoundException), print an error message.
    - If an input/output error occurs (IOException), print the exception message.
6. End

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 ");

      try {
         DataInputStream dis = new DataInputStream(new FileInputStream("input.txt"));
         while (true) {
            try {
               byte ch = dis.readByte();
               System.out.print((char) ch);
            } catch (EOFException e) {
               System.out.println("\nEOFException caught: End of file reached.");
               break;
            }
         }
         dis.close();
      } catch (FileNotFoundException e) {
         System.out.println("FileNotFoundException: Make sure 'input.txt' exists in the working
directory.");
      } catch (IOException e) {
         System.out.println("IOException occurred: " + e.getMessage());
      }
   }
}
```

Output:

Class Found inside package.

EOF exception for output

My name is Naima Islam. I am a good girl.

EOFException caught: End of file reached.

Problem Number: 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.

Objectives: To understand file handling using C by reading from and writing to files.

Theory: This program is designed to read a list of positive integers from an input file (input.txt) and perform basic arithmetic operations—addition, subtraction, multiplication, and division—on them. The results of these operations are then written to an output file (output.txt). The program processes the data in cases, typically grouping the numbers in pairs or specified formats to compute results like sum, difference, product, and quotient. It uses file handling techniques to read and write data, loops for iteration, and basic arithmetic logic for calculations. This task reinforces the use of file input/output, control structures, and modular programming for real-world data processing.

Algorithm: The algorithm of this problem is writing in below:

1. Start the program.
2. Open the Input.txt file for reading using freopen.
3. Open the Output.txt file for writing using freopen.
4. Declare two integer variables a and b.
5. Use a while loop to read pairs of integers (a and b) from the input file until the end of the file is reached:
    - Read two integers from the file using scanf.
    - If reading is successful (i.e., two integers are read), continue; otherwise, exit the loop.
6. For each pair (a, b):
    - Calculate and print the sum of a and b.
    - Calculate and print the subtraction of b from a.
    - Calculate and print the multiplication of a and b.
    - Check if b is not zero:
        - If true, calculate and print the division of a by b (as float).
        - If false, print a message that division is undefined.
    - Print a separator line (----------) after each case.
7. Close the input and output streams (automatically done when the program ends).
8. End the program.

Code:

```
#include <stdio.h>

int main() {
   freopen("Input.txt", "r", stdin);
   freopen("Output.txt", "w", stdout);

   int a, b;
```

```
    while (scanf("%d %d", &a, &b) == 2) {
        printf("Sum of %d and %d is %d\n", a, b, a + b);
        printf("Subtraction of %d and %d is %d\n", a, b, a - b);
        printf("Multiplication of %d and %d is %d\n", a, b, a * b);
        if (b != 0)
            printf("Division of %d and %d is %.2f\n", a, b, (float)a / b);
        else
            printf("Division of %d and %d is undefined (division by zero)\n", a, b);

        printf("------------------------\n");
    }

    return 0;
}
```

Input: 34 72 59 84 2552 2663

Output:

Sum of 34 and 72 is 106
Subtraction of 34 and 72 is -38
Multiplication of 34 and 72 is 2448
Division of 34 and 72 is 0.47

Sum of 59 and 84 is 143
Subtraction of 59 and 84 is -25
Multiplication of 59 and 84 is 4956
Division of 59 and 84 is 0.70

Sum of 2552 and 2663 is 5215
Subtraction of 2552 and 2663 is -111
Multiplication of 2552 and 2663 is 6795976
Division of 2552 and 2663 is 0.96

Problem Number: 09

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

Objectives: To study and understand the applications and significance of software engineering in the development of biomedical systems and in the advancement of artificial intelligence and robotics technologies.

Theory: Software engineering is a disciplined and systematic approach to the design, development, testing, deployment, and maintenance of software applications. It applies engineering principles to ensure that software is reliable, efficient, scalable, maintainable, and meets user requirements.

- **Role of Software Engineering in Biomedical Engineering:** Software engineering plays a critical role in developing reliable, efficient, and safe software systems used in healthcare and biomedical applications. For this aspect, some key contributions are following:
  - ✓ **Medical Imaging Software**: Development of applications for processing and analyzing CT, MRI, and X-ray images.
  - ✓ **Patient Monitoring Systems**: Software for continuous tracking of vital signs like ECG, blood pressure, and oxygen levels.
  - ✓ **Electronic Health Records (EHR)**: Secure and efficient management of patient data through well-structured software systems.
  - ✓ **Biomedical Device Control**: Programming embedded systems for devices like pacemakers, infusion pumps, and prosthetics.
  - ✓ **Simulation and Modeling**: Creating simulations of physiological systems for education, diagnosis, and research.
  - ✓ **Regulatory Compliance**: Ensuring software meets standards (like FDA or ISO) for medical device software validation.

- **Role of Software Engineering in Artificial Intelligence and Robotics:** Software engineering is fundamental in designing, developing, testing, and maintaining the complex systems that power AI and robotics. For this aspect, some key contributions are following:
  - ✓ **Algorithm Development**: Implementing AI models for machine learning, deep learning, and decision-making.
  - ✓ **Robot Control Systems**: Writing real-time control software for robotic motion, perception, and interaction.
  - ✓ **Natural Language Processing (NLP)**: Developing software for speech recognition, text understanding, and dialogue systems.
  - ✓ **Computer Vision**: Building software for object detection, tracking, and image interpretation in autonomous systems.

- ✓ **Autonomous Systems**: Integrating AI algorithms with sensors and actuators for self-driving vehicles, drones, and service robots.
- ✓ **System Integration and Testing**: Ensuring seamless interaction between hardware, software, and AI modules with robust testing.

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 Number: 10

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

Objectives: To understand the structure and flow of the Waterfall Model in software development.

Theory: The Waterfall Model is a sequential software development process in which progress flows in a linear and downward direction through several distinct phases—Requirement Gathering, System Analysis, Coding, Testing, Implementation, and Operations & Maintenance. Each phase must be completed before the next one begins, and there is typically no overlap between the phases. This model assumes that all requirements can be gathered at the beginning of the project and that they will not change throughout the development process. It is best suited for projects with clearly defined requirements and where changes are unlikely during development.

Figure 01: Waterfall Model

Among all the phases, **Requirement Gathering** is considered the most dominant one in the Waterfall Model. This is due to the following reasons:

- **Foundation for All Other Phases**
  - The Requirement Gathering phase lays the groundwork for the entire project. It involves understanding and documenting what the users need from the software.
  - If requirements are unclear or incomplete, the mistakes will carry forward into design, coding, testing, and beyond, causing major issues.
- **No Going Back**
  - The Waterfall Model does not support going back once a phase is completed. So, if there is an error or misunderstanding in the requirements, it is very difficult to fix it later without redoing all the subsequent phases.
  - This makes the accuracy of the Requirement phase absolutely critical.
- **Influences Project Success**
  - Since all design decisions, coding logic, and test cases are derived from the gathered requirements, any flaw here can affect the quality and success of the entire project.
- **Higher Cost of Mistakes Later**
  - Errors found in later phases (like testing or implementation) that originated from poor requirements are more expensive and time-consuming to correct.

- **Best Fit for Known Projects**
    - The Waterfall Model works best when requirements are well-known and have little chance of changing, which reinforces the importance of getting them right at the beginning.

Conclusion: In the Waterfall Model, the Requirement Gathering phase is the most dominant because it determines the direction and success of all following phases. Any mistake in this phase can lead to significant problems down the line, with limited opportunity to correct them due to the model's rigid, non-iterative structure.

Problem Number: 11

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

Objectives: To estimate the software development effort required for an industrial domain project using the COCOMO (Constructive Cost Model).

Solution: The COCOMO model is a well-established software cost estimation technique developed by Barry Boehm. It provides a mathematical formula to estimate the effort, time, and cost of a software development project based on the size of the project, typically measured in Kilo Lines of Code (KLOC). The model exists in different forms:

- Basic COCOMO model
- Intermediate COCOMO model
- Detailed COCOMO model

Basic COCOMO model: The Basic COCOMO model offers a simplified approach to estimate effort and development time based solely on the size of the software in Kilo Lines of Code (KLOC). It classifies projects into three categories:

  - Organic: Simple projects with small teams and well-understood requirements.
  - Semi-detached: Intermediate projects with mixed teams and moderate complexity.
  - Embedded: Complex projects with tight hardware/software integration and real-time constraints.

This model is useful for quick, rough estimates during the early stages of a project.

Intermediate COCOMO model: Intermediate COCOMO model builds upon the basic model by introducing a set of cost drivers that affect the software development process. These drivers include factors such as product reliability, team experience, and development environment. By incorporating these additional attributes, the intermediate model provides more accurate and customized estimates suited to the specific nature of the project.

Detailed COCOMO model: The Detailed COCOMO model extends the intermediate version by applying cost drivers to each phase of the software development lifecycle individually, such as design, coding, testing, and maintenance. This model offers the highest level of accuracy and is typically used for large, complex projects where detailed planning and resource allocation are critical.

To estimate the software development effort required for an industrial domain project using the COCOMO (Constructive Cost Model). This experiment aims to analyze the size of the software in terms of Kilo Lines of Code (KLOC) and compute the required effort (in person-months), development time, and personnel using the basic COCOMO model.

Basic COCOMO Model Formula: The estimation formulas used in the Basic COCOMO model are:

$$Effort\ (person-months) = a * (KLOC)^b$$

Where:

- KLOC = Thousands of Lines of Code
- a, b = Constants depending on the project type

$$Development\ Time(months) = c * (effort)^d$$

Where:

- c, d = Constants depending on the project type

For an industrial domain project (e.g., an automated packaging system or control circuit software), the category is often Semi-detached or Embedded, depending on the hardware interaction level and complexity. The COCOMO model helps in:

- Planning and budgeting industrial software projects
- Evaluating feasibility
- Improving project scheduling

Conclusion: By applying COCOMO, industrial software development teams can better estimate the required effort and avoid underestimation or overestimation, leading to improved project management and resource allocation.

Problem Number: 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".

Objectives: The objective of this study is to estimate the software development effort for a specific problem in the industrial domain using the COCOMO (Constructive Cost Model) model, and to understand the reasons behind software crisis through the analysis of real-world failure scenarios. The goal is to demonstrate how software estimation techniques and lessons from crisis situations can improve planning, development, and maintenance of software systems in critical domains like airline reservations and financial systems.

Theory: The Software Crisis refers to the challenges faced in software development, especially in the areas of reliability, maintainability, cost, complexity, and delayed delivery. It emerged when demand for software outgrew the ability to develop and maintain it effectively.

Common causes of software crisis include:

- Increasing complexity of systems
- Poor project management
- Lack of proper testing
- Incomplete requirements
- Inadequate documentation
- Failure to predict future software behavior (e.g., during peak load)
- Inability to debug large and complex systems

Possible solutions to the software crisis:

- Use of formal software engineering practices
- Agile methodologies for iterative testing and feedback
- Robust testing strategies (unit, integration, regression, system testing)
- Strong version control and debugging tools
- Modular software design and documentation
- Continuous monitoring and maintenance

Case Analysis and Solution

Case 1: Airline Ticket Reservation System Crash
Problem Summary: The software worked fine from midnight till noon, then crashed at 12.00 PM. It took 5 hours to identify and fix the defect.
Reason behind the failure:

- Time-based bug (likely a 12-hour to 24-hour time conversion issue)
- Inadequate testing for boundary time conditions
- Possibly no real-time stress testing was done for 24-hour runtime scenarios

Possible Solutions:

1. Perform boundary value testing—especially for time-related variables (midnight/noon cases).
2. Automate time-based simulation tests—simulate software running for 24 hours before actual deployment.
3. Log monitoring and exception handling—implement real-time logs that help quickly trace runtime errors.
4. Regression testing after every deployment to catch potential bugs before they affect end users.
5. Failover mechanisms—include backup systems or automated restart services to minimize downtime.

Case 2: Financial System Malfunction

Problem Summary: After delivery, the customer reported a malfunction. Due to the software's huge size and complexity, the team failed to identify the defect.

Reason behind the failure:

- Large, **monolithic** codebase without modular structure
- Lack of proper documentation and traceability
- Insufficient debugging tools and error logs
- Possibly no use of proper design patterns or architecture

Possible Solutions:

1. Modular Design—break down the system into smaller, testable modules for better fault isolation.
2. Proper Documentation—maintain detailed software design, logic flow, and change logs.
3. Logging and Monitoring—use detailed logging to trace the malfunction point.
4. Unit and Integration Testing—test individual components before integration.
5. Automated Debugging Tools—use profilers, debuggers, and static analyzers to detect code-level issues.
6. Code Reviews and Refactoring—ensure that complex parts of code are reviewed by multiple team members and simplified if possible.

Conclusion: The software crisis in both cases emerged due to lack of robust testing, code modularity, and real-world scenario handling. By applying engineering principles, modular design, and automated testing, such failures can be minimized. This experiment shows the importance of proactive development and testing strategies in critical software systems like airline reservations and financial applications.