Pabna University of Science and Technology

Faculty of Engineering and Technology

Department of Information and Communication Engineering

Lab Report

Course Title: **System Analysis and Software Testing Sessional**

Course Code: **ICE-4204**

Submitted By:	**Submitted To:**
Farhan Masud **Roll: 200625** **Session**: 2019-2020 4th Year 2nd semester Dept. of ICE, PUST.	**Md. Anwar Hossain** **Professor** Dept. of ICE, PUST.

Date of Submission: **24/05/2025**

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.

Objectives: To implement a calculator to handle basic arithmetic operation : addition, subtraction, multiplication, division, modulo.

Theory : A simple calculator program perform basic arithmetic operation such as : addition, subtraction, multiplication and modulo. It takes two input number and one operator. The program reads these inputs. It uses conditional logic to determine the correct operation and finally display the result. This type of program helps in understanding input handling, expression parsing, and basic control structures in programming.

Algorithm:

1. Start
2. Read the input expression (e.g., 1+2) as a string.
3. Parse the first number, operator, and second number from the input.
4. Use a conditional structure (if or switch) to:
    - Perform addition if operator is +
    - Perform subtraction if operator is –
    - Perform multiplication if operator is *
    - Perform division if operator is /
    - Perform modulo if operator is %
5. Calculate the result based on the operator.
6. Display the full expression with result (e.g., 1+2=3).
7. End

Source Code:

```
#include<stdio.h>
int main() {
int num1, num2, result;
char operator;
printf("Enter expression (e.g., 1+2): ");
scanf("%d%c%d", &num1, &operator, &num2);

switch (operator) {
case '+': result = num1 + num2; break;
case '-': result = num1 - num2; break;
case '*': result = num1 * num2; break;
case '/':
      if (num2 != 0)
           result = num1 / num2;
       else {
              printf("Division by zero error.\n");

                return 1;
            }
             break;
```

```
            case '%':
                if (num2 != 0)
                    result = num1 % num2;
                else {
                    printf("Modulo by zero error.\n");
                     return 1;
                    }
                    break;
                   default:
                       printf("Invalid operator.\n");
                        return 1;
                    }
                    printf("%d%c%d=%d\n", num1, operator, num2, result);
                    return 0;
                   }
```

Input : Sample input: 1+2; 8%4

Output: Sample output: 1+2=3; 8%4=0.

Problem no : 2

Problem Name: Write a program in "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 in a single input stream

Theory: The program read multiple pair of integers followed by an operator. The entered operator should be checked. Addition is occurred if there I s '+',subtraction occurred if there is ''-",multiplication occurred if there is "*", division occurred if there is "/" and modulo occurred if there is "%". Display the desired output.

Algorithm:

1. Start
2. Initialize two arrays to store the two sequences of n integers.
3. Read input values until an operator (+, -, *, /, %) is encountered.
4. Count the number of integers and divide them into two equal parts.
5. Store the first half in array A and the second half in array B.
6. Identify the operator at the end of input.
7. For each index i from 0 to n-1:
   Perform the operation A[i] op B[i]:
   - + → addition
   - - → subtraction
   - * → multiplication
   - / → division (check for divide-by-zero)
   - % → modulo
8. Print the resulting n outputs.
9. End

Source Code :

```
#include<stdio.h>
#include
#include
int main() {
int numbers[100];
int count = 0;
char input[1000], *token;
char operator;
printf("Enter integers followed by an operator (e.g., 4 5 7 8 20 40 +):\n");
fgets(input, sizeof(input), stdin);
```

```
token = strtok(input, " \n");
 while (token != NULL){
            if (strchr("+-*/%", token[0]) && strlen(token) == 1) {
               operator = token[0];
                   break;
               }
               numbers[count++] = atoi(token);
                token = strtok(NULL, " \n");
              }
              if (count % 2 != 0) {
                  printf("Invalid input: odd number of integers.\n");
                   return 1;
                }
             for (int i = 0; i < count; i += 2) {
                  int a = numbers[i];
                  int b = numbers[i + 1];
                  int result;
                  switch (operator) {
                      case '+': result = a + b; break;
                      case '-': result = a - b; break;
                      case '*': result = a * b; break;
                      case '/': result = (b != 0) ? a / b : 0; break;
                      case '%': result = (b != 0) ? a % b : 0; break;
                      default:
                            printf("Invalid operator.\n");
                            return 1;
                           }
                            printf("%d ", result);
                        }
                       printf("\n");
                       return 0;
                       }
```

Input: Enter integers followed by an operator :

4 5 7 8 20 40 +

Output : 9 15 60

Problem no : 03

Problem Name: Write a program in ‘’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 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: 1221 or madam. At first taken input from the user. Then the input is checked by comparing characters from the begging and end, moving towards the center. If all characters match the input considered as palindrome.

Algorithm :

1. Start
2. Take a string input from the user (treat both numbers and strings as character arrays).
3. Determine the length of the input.
4. Initialize two pointers: one at the start, one at the end.
5. While start < end:
    - Compare characters at both pointers.
    - If any pair doesn't match, it's not a palindrome → stop.
    - Move the pointers towards the center.
6. If all characters match, it's a palindrome.
7. Display the result.
8. End

Source Code:

```
#include<stdio.h>
#include<string.h> #include<ctype.h>
int isPalindrome(char str[]) {
    int left = 0;
    int right = strlen(str) - 1;
    while (left < right) {
        // Convert both characters to lowercase to make it case-insensitive
        char l = tolower(str[left]);
        char r = tolower(str[right]);
```

```
        If(l!=r){
            return 0; // Not a palindrome
            }
            left++; right--;
            }
```

```
                    return 1; // Is a palindrome }
                }
int main() {
char input[1000];
printf("Enter a number or string: ");
scanf("%s", input);
if (isPalindrome(input)) {
     printf("%s is a palindrome.\n", input);
    } else {
          printf("%s is not a palindrome.\n", input);
      }
      return 0;
}
```

Input : Enter a number or string: I want to be a good person

Enter a number or string: 4004

Output: I want to be a good person is not a palindrome.

4004 is not a palindrome.

Problem Number: 04

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

Objectives : To indicate the useful and non-functional prerequisites of an ATM (Computerized Teller Machine) framework and recognize potential bugs or issues in its operations to progress unwavering quality, security, and client involvement.

Theory : An Mechanized Teller Machine (ATM) gives clients with managing an account administrations such as cash withdrawals, stores, adjust request, and finance exchanges without the require for human interaction.

1. **Core functional Fundamental:**
    - **Authentication**
        Card insertion
        PIN verification (usually 4 or 6 digits)
    - **Transaction Services**

        Cash Withdrawal

        Balance Inquiry

        Mini Statement Fund Transfer (within bank or inter-bank)

        Deposit (cash or check)

    - **User Interface :**
        Multilingual support
        Touchscreen or button-based navigation
        Receipt printing option
    - **Security :**
        Encrypted PIN and transaction processing
        Session timeout
        Camera surveillance (hardware)
    - **Admin Services :**
        Cash loading and denomination management
        System diagnostics
        Logs of usage and errors

2. **Non-Functional Requirements:**
    - Availability: 24/7 operation
    - Performance: Transactions processed within a few seconds
    - Scalability: Support for integration with multiple banks and accounts
    - Reliability: Must handle power failures gracefully (e.g., UPS backup)
    - Maintainability: Logs, alerts, and remote diagnostics

3. **Hardware Interfaces**:
    - Card Reader (EMV chip/magnetic strip)
    - PIN Pad / Touchscreen
    - Cash Dispenser
    - Receipt Printer

- Deposit Module
- Network Modem or Router

4. **Software Interfaces:**

   - Bank Core System API
   - Encryption Libraries
   - Operating System (typically Linux or a secure embedded OS)
   - Database for local logging

Common Bugs in ATM Systems:

1. **Transaction Bugs:**
   - **Double Deduction:** Amount is debited twice due to a timeout or retry logic bug.
   - **Amount Not Dispensed:** Transaction success is shown but cash isn't dispensed.
   - **Wrong Balance Display:** After a transaction, the balance does not reflect accurately**.**
   - **Deposit Not Updated:** Deposited amount doesn't reflect in the account.
   - **Mini Statement Inaccuracy:** Transactions are missing or incorrectly ordered.
2. **UI/UX Bugs :**

   Incorrect Language Switching: Selecting a language doesn't change the UI language.
   Unresponsive Buttons: Hardware or software glitches make screen/buttons unusable.
   Wrong Instructions: Prompts don't match the transaction context

3. **Security Bugs**

   - **Session Doesn't Expire:** No timeout when the user walks away.
   - **PIN Not Masked:** PIN entry visible on the screen.
   - **No Encryption in Transmission:** PIN or transaction data sent as plain text.

Problem No: 05

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

Objectives: To write a C program that calculates the factorial of a given number using both for and while loops, and verifies that both methods produce the same result.

Theory:

Loops(for/while):

- Used to iterate from 1 to n and multiply each value to get the factorial.
- Integer Variables:

  One variable to hold the input number (n).

  One variable to store the result (fact), initialized to 1.

- Verification: Running both for and while loop versions and comparing the outputs confirms correction.
- Input Validation: Factorial is only defined for non-negative integers.

Algorithm:

1. Start
2. Initialize a variable fact = 1
3. Read input number n
4. If n < 0, display "Invalid input"
5. Otherwise,
   - Loop from i = 1 to n
   - Multiply fact = fact*i
6. Display fact
7. End

Source Code:

```
#include<stdio.h>
int factorialForLoop(int n) {
int result = 1;
for (int i = 1; i <= n; i++) {
result *= i;
}
return result;
}
```

```
int factorialWhileLoop(int n)
{
int result = 1;
int i = 1;
while (i <= n) {
 result *= i;
i++;
}
return result;
 }

int main()
{
int number;
printf("Enter a positive integer: ");
scanf("%d", &number);

 if (number < 0) {
 printf("Factorial is not defined for negative numbers.\n");
  return 1;
}


int factFor = factorialForLoop(number);
int factWhile = factorialWhileLoop(number);
printf("Factorial using for loop: %d\n", factFor);
printf("Factorial using while loop: %d\n", factWhile);
if (factFor == factWhile) {
 printf(" ✅ Both results match. Factorial is %d.\n", factFor")
}
 else
  {
printf(" ❌ )
}
  return 0;
 }
```

Input : Enter a positive integer: 5

Output:

Factorial using for loop: 120

Factorial using while loop: 120

Both results match. Factorial is 120**.**

Problem no : 06

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

Objectives : To write a C program that calculates the sum and average of elements in an array using a do-while loop and two user-defined functions—one for computing the sum and another for calculating the average.

Theory:

To find the sum and average of elements in an array in C:

- An array is a collection of similar data types stored in contiguous memory locations.
- The sum is the total of all elements.
- The average is the sum divided by the total number of elements.
- The task will be divided into two functions:
    1. calculateSum() – computes the sum using a do...while loop.
    2. calculateAverage() – calculates the average using the sum and number of elements.

Algorithm:

Main function:

1. Declare an array and variables for sum and average.
2. Take input for array size n.
3. Read n array elements from the user.
4. Call findSum() function to get the sum of array elements.
5. Call findAverage() function to compute the average using the sum and n.
6. Display the sum and average.

Function 1: findSum(int arr[],int sum):

1. Initialize sum = 0, i = 0
2. Use do-while loop:
    - Add arr[i] to sum
    - Increment i
    - Continue while i < n
3. Return sum

Function 2 : indAverage(int sum, int n)

1. Calculate average = (float)sum / n
2. Return average

Source Code:

```
#include<stdio.h>
// Function to calculate sum of array
int calculateSum(int arr[], int n) {
int sum = 0, i = 0;
do { sum += arr[i]; i++;
 } while (i < n);
 return sum;
}
// Function to calculate average of array
float calculateAverage(int sum, int n) {
return (float)sum / n;
}
int main()
{
int arr[100], n, i;
printf("Enter the number of elements in the array: ");
scanf("%d", &n);
if (n <= 0)
{
 printf("Invalid number of elements.\n");
return 1;
}
printf("Enter %d integers:\n", n);
 i = 0;
do
{
 printf("Element %d: ", i + 1);
 scanf("%d", &arr[i]);
 i++;
} while (i < n);
int sum = calculateSum(arr, n);
float average = calculateAverage(sum, n);
printf("\nSum of array elements = %d\n", sum);
printf("Average of array elements = %.2f\n", average);
return 0;
 }
```

Input :

Enter the number of elements in the array: 5

Enter 5 integers: Element 1: 5

Element 2: 8

Element 3: 9

Element 4: 8

Element 5: 2

Output:

Sum of array elements = 32

Average of array elements = 6.40

Problem No: 07

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

Objective :

To write a simple Java program that demonstrates the use of exception handling for ClassNotFoundException and EOFException, showing how these exceptions occur during dynamic class loading and file input operations, and how to handle them gracefully using try catch blocks.

Theory:

ClassNotFoundException

- **Definition:** ClassNotFoundException is a checked exception in Java that occurs when an application tries to load a class dynamically at runtime using methods like Class.forName() or ClassLoader.loadClass(), but the class with the specified name cannot be found in the classpath.
- **Importance :** Handling this exception is essential in programs that rely on dynamic class loading, such as plugin systems or applications using reflection.

EOFException (End of File Exception)

- **Definition:** EOFException is a subclass of IOException in Java. It signals that an end of file or stream has been reached unexpectedly during input operations, especially when using data streams like DataInputStream or ObjectInputStream.
- **Importance**: Catching this exception allows programs to detect when no more data can be read from a stream, enabling graceful termination of reading loops and avoiding infinite waiting or errors.

Source Code :

```
import java.io.*;
public class ExceptionDemo {
  public static void demoClassNotFoundException(){
      try{
       // Try to load a class that doesn't exist
            Class.forName("com.example.NonExistentClass");
               }catch(ClassNotFoundException e) {
               System.out.println("Caught ClassNotFoundException: " + e.getMessage());
             }
 }
```

```
public static void demoEOFException() {
          try {
            FileInputStream fis = new FileInputStream("testdata.dat");
            DataInputStream dis = new DataInputStream(fis);

             while (true) {
              // Intentionally read more data than file contains to cause EOFException
                int number = dis.readInt();
                System.out.println("Read number: " + number);
                 }
               } catch (EOFException e) {
                    System.out.println("Caught EOFException: Reached end of file.");
                  } catch (IOException e) {
                    System.out.println("IOException: " + e.getMessage());
                   }
         }
         public static void main(String[] args){
            System.out.println("Demo: ClassNotFoundException");
             demoClassNotFoundException();
             System.out.println("\nDemo: EOFException");
            demoEOFException();
       }
}
```

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: 100 25 1; **Objectives: Case-2:** 171 72 1;

Objectives:

- To read a list of positive integers from a text file (input.txt).
- To perform arithmetic operations (addition, subtraction, multiplication, division).
- To write the results of each operation into an output file (output.txt).
- To use file input/output handling and loops in C.

Theory:

- **Input**: Reads integers from a file (e.g., input.txt).
- **Operations**: Computes sum, difference, product, and integer division iteratively.
- **Output**: Writes results to separate files (sum.txt, difference.txt, product.txt, division.txt).
- **Error Handling**: Checks for file opening/reading errors and division by zero.
- **File Closing**: Ensures all files are properly closed.

Source Code:

```
#include<stdio.h>
int main() {
FILE *inputFile, *outputFile;
inputFile = fopen("input.txt", "r");
outputFile = fopen("output.txt", "w");
if (inputFile == NULL || outputFile == NULL) {
        printf("Error opening file.\n");
         return 1;
}
int num, count = 0;
int sum = 0, first = 1, subtraction = 0;
int product = 1;
float division = 0;
int numbers[100]; // assuming at most 100 integers
```

```
// Reading numbers from input.txt
while (fscanf(inputFile, "%d", &num) == 1) {
numbers[count++] = num;
sum += num;
product *= num;
}
if (count == 0) {
fprintf(outputFile, "No numbers found in input file.\n");
fclose(inputFile);
fclose(outputFile);
return 0;
}
// Initialize subtraction and division with the first number
subtraction = numbers[0];
division = (float) numbers[0];
for (int i = 1; i < count; i++) {
 subtraction -= numbers[i];
  if (numbers[i] != 0)
   division /= numbers[i];
  else {
    fprintf(outputFile, "Division by zero encountered. Cannot continue.\n");
    fclose(inputFile);
    fclose(outputFile);
    return 1;
     }
}

fprintf(outputFile, "Sum: %d\n", sum);
fprintf(outputFile, "Subtraction (left-to-right): %d\n", subtraction);
fprintf(outputFile, "Multiplication: %d\n", product);
fprintf(outputFile, "Division (left-to-right): %.2f\n", division);
fclose(inputFile);
fclose(outputFile);
return 0;
 }
```

Output:

Creating file and writing one object...

Reading two integers from file...

First int: 42

Caught EOFException: Reached end of file unexpectedly.

Problem No -09

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

Objectives:

- To understand the applications of software engineering in Biomedical Engineering.
- To explore the importance of software development in Artificial Intelligence (AI) and Robotics.
- To analyze how software engineering contributes to innovation, safety, and efficiency in these fields.

Theory: Software engineering is the disciplined application of engineering principles to the design, development, testing, and maintenance of software. It plays a critical role in several advanced fields, especially Biomedical Engineering, Artificial Intelligence (AI), and Robotics.

In Biomedical Engineering

Software engineering enables the development of advanced systems that help in diagnosis, treatment, monitoring, and research.

Applications include:

- Medical Devices: MRI machines, ECG, pacemakers use embedded software for accurate operations.
- Medical Imaging: Software processes images from X-rays, MRIs, CT scans for better diagnostics.
- Electronic Health Records (EHRs): Systems for storing and managing patient data securely.
- Bioinformatics: Software helps analyze genes, proteins, and other biological data.
- Wearable Devices: Devices like fitness trackers or glucose monitors rely on real-time software.

Software engineering is essential in creating intelligent, adaptive, and interactive systems.

Applications include:

- **Machine Learning Models:** Tools for training and deploying AI models rely on well structured software.
- **Computer Vision:** AI systems that recognize faces, gestures, or objects use advanced image-processing algorithms.
- **Robot Operating Systems (e.g., ROS):** Provide a framework for communication and control between robot components.
- **Autonomous Robots:** Use software for navigation, decision-making, and interaction with the environment.
- **Speech & Language Processing:** Software powers assistants like Siri or ChatGPT.

Role of Software Engineering in Artificial Intelligence (AI):

1. Algorithm Development:

- Machine Learning: Software engineers design and implement algorithms for machine learning and deep learning models that enable computers to learn from data.
- Natural Language Processing (NLP): They develop algorithms for understanding and generating human language.

2. AI System Integration:
    - Deployment: Engineers integrate AI models into applications, ensuring that they work effectively in real-world scenarios.
    - Scalability: They optimize AI systems for performance and scalability to handle large amounts of data and complex tasks.
3. Tool Development: Frameworks and Libraries: Engineers create and maintain AI frameworks (e.g.,TensorFlow, PyTorch) and libraries that simplify the development of AI applications.
4. Ethics and Safety:
    - Bias and Fairness: Software engineers work on reducing bias in AI systems and ensuring that they operate fairly and ethically.
    - Security: They implement measures to secure AI systems from vulnerabilities and attacks

Role of Software Engineering in Robotics:

1. Control Systems:
    - Robot Programming: Software engineers develop the control algorithms that govern robotic movements and actions.
    - Sensors and Actuators: They work on integrating sensors and actuators to enable robots to interact with their environment.
2. Robotic Software Frameworks:
    - Robot Operating System (ROS): Engineers develop and utilize software frameworks like ROS to facilitate the development, simulation, and control of robots.
3. Simulation and Testing:
    - Virtual Environments: Software engineers create virtual environments for simulating robotic operations and testing before deployment in real-world scenarios.
    - Performance Evaluation: They design tools for evaluating and improving robotic performance and reliability
4. Human-Robot Interaction:
    - User Interfaces: Engineers develop interfaces that allow humans to control and interact with robots effectively.
    - Collaboration: They work on systems that enable robots to collaborate with humans in shared workspaces.

Problem No -10

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

Objectives:

- To study and understand each phase of the Waterfall model.
- To analyze the flow of software development in a sequential model.
- To identify the most dominant and critical phase of the Waterfall model.

Theory: The Waterfall Model is one of the earliest models of the Software Development Life Cycle (SDLC). It is a linear sequential model where each phase must be completed before the next begins. The model flows in a single direction — like a waterfall — hence the name.

Phases of the Waterfall Model:

1. **Requirement Analysis:**
   - All possible requirements of the system are gathered and documented.
   - The goal is to understand what the customer needs.
2. **System Design:**
   - The system's architecture and design are prepared from the requirements.
   - Defines hardware, software, data, and system interfaces.
3. **Implementation (Coding):**
   - Actual source code is written in this phase using appropriate programming languages and tools.
4. **Testing o After coding:**
   - the system is tested thoroughly for bugs and errors.
   - This phase ensures the software meets the desired requirements**.**
5. **Deployment:**
   - The finished product is delivered to the customer and deployed in the target environment.
6. **Maintenance :**
   - Once the software is in use, issues are fixed and improvements are made.
   - Includes bug fixes, upgrades, and minor enhancements.

Most Dominated Phase

Requirements Analysis is often considered the most critical phase in the Waterfall model. Here's why:

1. Foundation for All Subsequent Phases:
   - The success of the entire project depends on how well the requirements are gathered and documented. If the requirements are incorrect or incomplete, it affects all subsequent phases, leading to potential project failures.
2. Cost of Changes:
   - Changes made after the requirements phase are more costly and time-consuming. Once the system design and implementation have started, making changes based on incorrect or missed requirements can be expensive.
3. Client Satisfaction:

- Properly understanding and documenting the requirements ensures that the final product meets the client's needs and expectations. This leads to higher client satisfaction and reduces the risk of project failure.

4. Risk Management:
    - Identifying and understanding requirements early helps in identifying potential risks and addressing them before they impact the development process.

In summary, while all phases of the Waterfall model are important, the Requirements Analysis phase is the most dominant as it lays the groundwork for the entire project and impacts all subsequent phases. Proper requirements analysis is crucial for the successful completion and quality of the software project.

Problem No -11

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

Objectives:

- To apply the COCOMO (Constructive Cost Model) for software effort estimation.
- To estimate the development time and effort for a sample industrial problem.
- To understand how project size and type influence the cost estimation.

Theory: The COCOMO Model (Constructive Cost Model) is a widely used model for estimating the cost, effort, and time required to develop software. Developed by Barry Boehm, it is based on the size of the software in KLOC (Kilo Lines of Code). There are three categories:

1. Organic: Small, simple software projects.
2. Semi-Detached: Medium complexity.
3. Embedded: Complex software with tight hardware, software, or regulatory constraints.

The COCOMO (Constructive Cost Model) model is used to estimate the effort, cost, and schedule for software projects based on the size of the software and various project characteristics. To estimate effort using the COCOMO model, follow these steps:

1. **Gather Information**
   Inputs Needed:
   - Size of the Software: Typically measured in Lines of Code (LOC) or Function Points (FP).
   - Project Attributes: Various cost drivers such as software complexity, team experience, and development environment.
2. **Choose a COCOMO Model Type**
   COCOMO has several models based on the project type:
   - Basic COCOMO: For early, rough estimates.
   - Intermediate COCOMO: Includes cost drivers to refine the estimate.
   - Detailed COCOMO: Provides a more detailed estimate by including a detailed analysis of project characteristics.
3. **Apply Basic COCOMO Model**

   Basic COCOMO Equations:

   Effort (E) = a × (KLOC)^b

   Development Time (TDEV) = c × (E)^d

   Where:

   Effort is measured in person-months.

   KLOC is thee stimated size of the software in thousands of lines of code.

   a and b are coefficients that vary based on the project type.

Project Types:

- ❖ Organic (simple projects):

✓ a=2.4

✓ b=1.05

❖ Semi-Detached (medium complexity projects):

✓ a=3.0

✓ b=1.12

❖ Embedded (complex projects):

✓ a=3.6

✓ b=1.20

4. **Apply Intermediate or Detailed COCOMO Model (if needed)**
    - Intermediate COCOMO adds cost drivers for various factors affecting the project such as product reliability, software engineering capabilities, and hardware constraints.
    - Detailed COCOMO further refines the estimate by analyzing each phase of the development lifecycle (e.g., requirements analysis, design, coding, testing)

Example Calculation

Assumptions:

- Software size: 50,000 LOC (50 KLOC)
- Project Type: Semi-Detached

Using Basic COCOMO:

- Coefficients for Semi-Detached: a=3.0, b=1.12
- Effort: Effort=3.0×(50)1.12

Calculate: (50)1.12≈55.08 Effort=3.0×55.08≈165.24

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 at 12.00 AM (midnight) 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 informed the development team about a malfunction in the system. As the software was huge and complex, the development team could not identify the defect in the software."

Objectives:

- To identify and understand the reasons behind software crisis.
- To analyze real-world cases of software failure.
- To suggest appropriate preventive and corrective solutions for each case.

Theory:

The "software crisis" refers to the challenges and difficulties faced in developing and maintaining software systems. Key reasons include:

1. Complexity:
    - Description: Software systems are often complex and difficult to manage, leading to bugs and inefficiencies.
    - Impact: Complexity increases the chance of errors and makes debugging and maintenance more challenging.
2. Unclear Requirements:
    - Description: Poorly defined or changing requirements can lead to software that does not meet user needs.
    - Impact: Misalignment between the delivered software and user expectations.
3. Poor Project Management:
    - Description: Inadequate planning, scheduling, and resource management can lead to missed deadlines and budget overruns.
    - Impact: Projects may be delivered late or with reduced functionality.
4. Lack of Testing:
    - Description: Inadequate or incomplete testing can result in undetected bugs and issues.
    - Impact: Software may fail under real-world conditions, leading to crashes and malfunctions.
5. Inadequate Documentation:
    - Description: Lack of proper documentation can hinder understanding and maintenance of the software.
    - Impact: Difficulty in troubleshooting and extending the software.

Case 1: Air Ticket Reservation Software

Scenario: The software crashed 12 hours after installation, causing a 5-hour downtime for ticket reservations.

Possible Solutions:

1. Improved Testing:
    - Solution: Implement comprehensive testing including stress testing, load testing, and end-to-end testing to identify potential issues before deploymen.
    - Benefit: Helps in detecting and fixing issues that could cause crashes under real-world conditions.
2. Robust Monitoring and Logging:
    - Solution: Implement real-time monitoring and detailed logging to detect anomalies and diagnose problems quickly.
    - Benefit: Facilitates faster identification of the root cause of issues and reduces downtime.
3. Contingency Planning:
    - Solution: Develop a contingency plan and backup systems to handle unexpected failures.
    - Benefit: Minimizes disruption to operations and allows for quick recovery.
4. Post-Deployment Support:
    - Solution: Ensure that support teams are available immediately after deployment to address any issues that arise.
    - Benefit: Provides rapid response to problems, reducing the impact on users.

Case 2: Financial Systems Software

Scenario: The development team struggles to identify defects in a large and complex financial

Possible Solutions:

1. Enhanced Debugging Tools:
    - Solution: Utilize advanced debugging tools and techniques to analyze and trace complex issues within the software.
    - Benefit: Helps in pinpointing defects more efficiently in complex systems.
2. Modular Design:
    - Solution: Design the software in modular components or services, making it easier to isolate and test individual parts.
    - Benefit: Reduces complexity and makes it easier to identify and fix defects.
3. Code Reviews and Pair Programming:
    - Solution: Conduct regular code reviews and use pair programming to catch defects early in the development process.
    - Benefit: Improves code quality and helps in identifying issues before they become significant problems.
4. Automated Testing:
    - Solution: Implement automated testing frameworks to continuously test the software for defects.
    - Benefit: Ensures that changes do not introduce new issues and helps in catching defects early.
5. Clear Documentation and Requirements:

- Solution: Maintain clear and detailed documentation of the software's design, architecture, and requirements.
- Benefit: Helps developers understand the system better and identify defects more effectively.

Summery

- **Case1 Solutions:** Improved testing, robust monitoring, contingency planning, and post-deployment support.
- **Case2 Solutions:** Enhanced debugging tools, modular design, code reviews, automated testing, and clear documentation.

Addressing these issues with the proposed solutions can significantly improve software quality and reduce the impact of defects.