Pabna University of Science and Technology

Faculty of Engineering and Technology

Department of Information and Communication Engineering

Practical Lab Report

Course Code: ICE-4204

Course Title: System Analysis and Software Testing Sessional.

Submitted To,

Md. Anwar Hossain

Professor,

Department of Information and Communication Engineering,

Pabna University of Science and Technology.

Submitted By,

Sayeeda Khan

Roll: **200604**

Session: 2019-2020

4th Year 2nd Semester

Department of Information and Communication Engineering,

Pabna University of Science and Technology.

INDEX

Experiment Number: 01

Name of the experiment: 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: The main objective of this program is to build a basic calculator that can take user input in the form of a mathematical expression (like 1+2 or 8%4), evaluate it, and then display the result in a format like 1+2=3.

This calculator supports five operations:

- Addition (+)
- Subtraction (-)
- Multiplication (*)
- Division (/)
- Modulo (%)

It performs the operation between two operands based on the operator the user provides.

Theory: This program mainly uses basic input handling, character processing, and conditional logic. In C, we can read the expression using scanf() and store the input into variables: two integers and a character (the operator). Then, based on the operator, we use an if-else or switch statement to perform the correct calculation.

Algorithm:

- Start the program by including required header files (stdio.h).
- Declare three variables:
  - Two integers to store the numbers (e.g., num1, num2)
  - One character to store the operator (e.g., op)
- Prompt the user to enter an expression in the form number operator number, like 5+3.
- Use scanf() to read the input in the format: *scanf("%d%c%d", &num1, &op, &num2);* This will store the first number in num1, the operator in op, and the second number in num2.
- Use a switch statement (or if-else) to check the value of the operator:
  - If op is '+', calculate num1 + num2
  - If op is '-', calculate num1 - num2
  - If op is '*', calculate num1 * num2
  - If op is '/', check if num2 != 0, then calculate num1 / num2
  - If op is '%', check if num2 != 0, then calculate num1 % num2

- ❖ Store the result of the operation in a separate variable (e.g., result).
- ❖ Print the full expression with the result in the format: num1 operator num2 = result (like 5+3=8)
- ❖ If the operator is not recognized, display an error message like "Invalid operator."
- ❖ End the program with a return statement.

Pseudo 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;
            printf("%d+%d=%d\n", num1, num2, result);
            break;
        case '-':
            result = num1 - num2;
            printf("%d-%d=%d\n", num1, num2, result);
            break;
        case '*':
            result = num1 * num2;
            printf("%d*%d=%d\n", num1, num2, result);
            break;
        case '/':
            if (num2 != 0) {
                result = num1 / num2;
                printf("%d/%d=%d\n", num1, num2, result);
            } else {
                printf("Division by zero is not allowed.\n");
            }
            break;
        case '%':
            if (num2 != 0) {
                result = num1 % num2;
                printf("%d%%%d=%d\n", num1, num2, result); // %% to print %
            } else {
                printf("Modulo by zero is not allowed.\n");
            }
            break;
        default:
            printf("Invalid operator.\n");
    }

    return 0;
}
```

Sample Input:

```
Enter expression (e.g., 1+2): 04+69
```

Sample Output:

```
4+69=73
```

Experiment Number: 02

Name of the experiment: 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: The goal of this C program is to take a list of integers as input, followed by an operator at the end (like +, *, etc.). The program then performs the specified operation in pairs producing a result for each pair and printing them all together.

For example, input: 4 5 7 8 20 40 +

Output: 9 15 60

Explanation: 4+5=9, 7+8=15, 20+40=60.

Theory: This program uses the idea of processing a dynamic list of integers and applying arithmetic operations in pairs. In C, we can use arrays to store the input numbers and use a loop to apply the operation on every two consecutive numbers. The program finishes input collection once it encounters the operator, which is read as a character.

This involves input reading, looping, basic arithmetic, and array handling in C.

Algorithm:

1. Start the program and include the necessary header files (stdio.h).
2. Declare an integer array to store the input numbers. Also declare variables to keep count of inputs and store the operator.
3. Prompt the user to enter numbers followed by an operator (e.g., 4 5 7 8 20 40 +).
4. Use a loop to keep reading input:
    - If the input is an integer, store it in the array.
    - If the input is a character and matches an operator (+, -, *, /, %), stop the loop and store the operator.
5. Check that the number of integers is even (because we need pairs). If it's not even, display an error.
6. Use a loop to go through the array two elements at a time:
    - For each pair (e.g., numbers at index 0 and 1, 2 and 3, etc.), apply the selected operation.
    - Store or directly print the result.
7. Use a switch statement (or if-else) to apply the correct operation depending on the operator:
    - +: add the pair
    - -: subtract second from first

- *: multiply the pair
- /: divide first by second (after checking for divide-by-zero)
- %: modulo (after checking second number is not zero)

8. Print all the results in one line, separated by spaces.
9. End the program.

Pseudo Code:

```
#include <stdio.h>
int main() {
    int numbers[100], count = 0;
    char input[1000];
    char op;
    printf("Enter even number of integers followed by an operator (e.g., 4 5 7 8 +):\n");
    // Read whole line of input
    fgets(input, sizeof(input), stdin);
    // Scan integers and the operator
    int i = 0, offset = 0;
    while (sscanf(input + offset, "%d%n", &numbers[count], &i) == 1) {
        count++;
        offset += i;
    }
    // Scan operator after all numbers
    sscanf(input + offset, " %c", &op);
    if (count % 2 != 0) {
        printf("Please enter an even number of integers.\n");
        return 1;
    }
    printf("Output:\n");
    int j =0;
    for (j; j < count; j += 2) {
        int a = numbers[j];
        int b = numbers[j + 1];
        switch (op) {
            case '+': printf("%d ", a + b); break;
            case '-': printf("%d ", a - b); break;
            case '*': printf("%d ", a * b); break;
            case '/': b != 0 ? printf("%d ", a / b) : printf("NaN "); break;
            case '%': b != 0 ? printf("%d ", a % b) : printf("NaN "); break;
            default: printf("Invalid operator\n"); return 1;
        }
    }
    printf("\n");
    return 0;
}
```

Sample Input:

```
Enter even number of integers followed by an operator (e.g., 4 5 7 8 +):
4 5 7 8 9 45 +
```

Sample Output:

```
Output:
9 15 54
```

Experiment Number: 03

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

Objective: The objective of this program is to check whether a given number or string is a palindrome. A palindrome is something that reads the same forward and backward — like 121 or madam. The program will take input and determine if it's a palindrome or not, then display the result.

Theory: In C, we can use either strings or integers as input. To check if a number is a palindrome, we reverse its digits and compare it with the original number. For a string, we compare characters from the beginning and end, moving toward the center. If all pairs match, it's a palindrome.

This requires string handling, looping, conditional checks, and possibly type conversion if we want to handle both numbers and strings together.

Algorithm:

1. Start the program and include the necessary headers like stdio.h and string.h.
2. Declare a character array to store input (so that we can handle both numbers and strings as strings).
3. Prompt the user to enter a string or number.
4. Read the input using scanf() and store it as a string (even numbers will be treated as strings).
5. Calculate the length of the string using strlen().
6. Use a loop to compare characters:
    - Compare the first character with the last, the second with the second-last, and so on.
    - If any mismatch is found, it's not a palindrome.
7. If all characters match during the comparison, declare it as a palindrome.
8. Print the result to the user — whether the input is a palindrome or not.
9. End the program.

Pseudo Code:

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

int isPalindrome(char str[]) {
   int start = 0;
   int end = strlen(str) - 1;
```

```
    while (start < end) {
        if (str[start] != str[end])
            return 0;  // Not a palindrome
        start++;
        end--;
    }
    return 1;  // It's a palindrome
}

int main() {
    char input[100];

    printf("Enter a string or number: ");
    scanf("%s", input);

    // Optional: convert to lowercase for case-insensitive comparison
    for (int i = 0; input[i]; i++) {
        input[i] = tolower(input[i]);
    }

    if (isPalindrome(input)) {
        printf("'%s' is a palindrome.\n", input);
    } else {
        printf("'%s' is not a palindrome.\n", input);
    }

    return 0;
}
```

Sample Input:

```
Enter a string or number: 1221
Enter a string or number: 1236
Enter a string or number: Level
Enter a string or number: Sayeeda
```

Sample Output:

```
'1221' is a palindrome.
'1236' is not a palindrome.
'level' is a palindrome.
'sayeeda' is not a palindrome.
```

Experiment Number: 04

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

ATM System Specifications

An ATM system helps people access their bank accounts easily and safely. Here's what a basic ATM should do:

- User Login: When you insert your card, you need to enter a PIN. The system checks if the PIN is right before letting you do anything.
- Account Options: You can pick from your accounts, like checking or savings, to do transactions.
- Transactions: You can withdraw money, check your balance, deposit cash, or even transfer money between accounts.
- Security: The system should lock you out after a few wrong PIN tries to keep your money safe.
- Easy Use: The screen should show clear instructions so anyone can follow along.
- Error Handling: If there's a problem, like not enough money or a card stuck in the machine, the ATM should let you know in simple words.
- Records: Every action you do should be saved securely so the bank can check it if needed.

Common Bugs Found in ATM Systems (Real-life Problems)

- PIN Issues: Sometimes, if you enter the wrong PIN many times, the system might not block the card like it should, which is a big risk.
- Balance Errors: The amount shown can sometimes be wrong if the system didn't update properly after a recent transaction.
- Cash Not Dispensed but Money Deducted: The ATM may take the money out of your account but fail to give you cash because of a jam or hardware problem.
- Input Problems: If you enter weird things (like letters instead of numbers), the machine might crash or behave strangely.
- Confusing Messages: The error messages can be too vague, leaving users unsure what went wrong.
- Session Doesn't End: If you walk away without logging out, someone else could use your account.
- Hardware Glitches: Sometimes the card reader or keypad stops working, and the ATM freezes or shuts down.
- Security Holes: If data isn't handled carefully, hackers might steal PINs or account info.

Experiment Number: 05

Name of the experiment: 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 program aims to calculate the factorial of a given number. Factorial means multiplying all positive integers from 1 up to that number. For example, factorial of 5 (written as 5!) is 1×2×3×4×5 = 120. The program will do this calculation using either a for loop or a while loop, and then compare the results from both methods to confirm they are the same.

Theory: The factorial function is a fundamental concept in mathematics, often used in combinations, permutations, and probability. In programming, loops are ideal for factorial because the operation involves repeated multiplication.

- A for loop is convenient when the number of iterations is known upfront, like multiplying numbers from 1 to n.
- A while loop can achieve the same but offers more flexibility by continuing as long as a condition is true.

Both loops will multiply numbers sequentially to get the factorial. After computing factorial with both loops, comparing results verifies that the logic is correct.

Algorithm:

1. Start the program.
2. Input a positive integer n from the user.
3. Initialize two variables to store factorial results from both loops, for example, fact_for = 1 and fact_while = 1.
4. Calculate factorial using for loop:
    - Loop variable i starts at 1 and runs until i is equal to n.
    - In each iteration, multiply fact_for by i.
5. Calculate factorial using while loop:
    - Initialize a counter variable j to 1.
    - While j is less than or equal to n, multiply fact_while by j and increment j.
6. Compare both factorial results:
    - If they are equal, print a confirmation message that both methods give the same answer.
    - Otherwise, print an error message (this usually should not happen if logic is correct).
7. Output the factorial result.
8. End the program.

Pseudo Code:

```
#include <stdio.h>
// Function using for loop
unsigned long long factorial_for(int n) {
    unsigned long long fact = 1;
    for (int i = 1; i <= n; i++) {
        fact *= i;
    }
    return fact;
}
// Function using while loop
unsigned long long factorial_while(int n) {
    unsigned long long fact = 1;
    int i = 1;
    while (i <= n) {
        fact *= i;
        i++;
    }
    return fact;
}
int main() {
    int number;
    printf("Enter a non-negative integer: ");
    scanf("%d", &number);
    if (number < 0) {
        printf("Factorial is not defined for negative numbers.\n");
        return 0;
    }
    unsigned long long result_for = factorial_for(number);
    unsigned long long result_while = factorial_while(number);
    printf("Factorial using for loop: %llu\n", result_for);
    printf("Factorial using while loop: %llu\n", result_while);
    if (result_for == result_while)
        printf("Results verified: Both methods give the same result.\n");
    else
        printf("Results mismatch! Please check the code.\n");
    return 0;
}
```

Sample Input:

```
Enter a non-negative integer: 6
```

Sample Output:

```
Factorial using for loop: 720
Factorial using while loop: 720
Results verified: Both methods give the same result.
```

Experiment Number: 06

Name of the experiment: 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 goal of this program is to take several numbers as input from the user, store them in an array, and then calculate:

1. The sum of all elements
2. The average of those elements

To do this, we will use a do-while loop for iteration and two separate functions:

- One function to calculate the sum
- Another function to calculate the average

Theory: An array is a collection of values stored in a single variable. To process each element, loops are used. A do-while loop is a post-tested loop, meaning it always runs at least once.

We break the logic into functions for better organization:

- Function 1: Calculates and returns the sum of array elements.
- Function 2: Calculates and returns the average, possibly by calling the sum function or reprocessing the array.

Using user-defined functions improves readability and reusability of code, which is a good programming practice.

Algorithm:

1. Start the program.
2. Declare an integer array to hold numbers.
3. Input the total number of elements, say n.
4. Input n numbers from the user and store them in the array using a do-while loop.
5. Define Function 1 (sumFunction):
    - Input: the array and its size.
    - Logic: Use a do-while loop to iterate through each element and add them to a sum variable.
    - Return the sum.
6. Define Function 2 (averageFunction):
    - Input: the same array and its size.
    - Logic: Call sumFunction to get the sum.
    - Divide the sum by n to get the average.
    - Return the average.

7. Call both functions from main and print their results.
8. End the program.

Pseudo Code:

```
#include <stdio.h>

// Function to calculate sum
int calculateSum(int arr[], int size) {
    int sum = 0;
    for (int i = 0; i < size; i++)
        sum += arr[i];
    return sum;
}

// Function to calculate average
float calculateAverage(int sum, int size) {
    return (float)sum / size;
}

int main() {
    int arr[100], n, i = 0;

    printf("Enter the number of elements (max 100): ");
    scanf("%d", &n);

    if (n <= 0 || n > 100) {
        printf("Invalid array size.\n");
        return 0;
    }

    printf("Enter %d integers:\n", n);
    do {
        scanf("%d", &arr[i]);
        i++;
    } while (i < n);

    int sum = calculateSum(arr, n);
    float avg = calculateAverage(sum, n);

    printf("Sum = %d\n", sum);
    printf("Average = %.2f\n", avg);

    return 0;
}
```

Sample Input:

```
Enter the number of elements (max 100): 5
Enter 5 integers:
4 5 9 10 12
```

Sample Output:

```
Sum = 40
Average = 8.00
```

Experiment Number: 07

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

Objective: The purpose of this program is to understand and demonstrate two specific exceptions in Java:

1. ClassNotFoundException – occurs when Java tries to load a class that isn't available or doesn't exist.
2. EOFException – occurs when an input stream reaches the end unexpectedly during reading.

This program helps us learn how to handle these exceptions properly using try-catch blocks.

Theory:

1. ClassNotFoundException

This is a **checked exception**. It happens when a program tries to load a class using functions like Class.forName("SomeClass"), but the class is not available in the classpath. This is common in JDBC or Reflection-based applications where class names are loaded dynamically.

2. EOFException (End of File Exception)

This is also a **checked exception**, which occurs during input operations — especially when reading from a file or stream using classes like ObjectInputStream or DataInputStream. If the program tries to read more data than exists in the file, this exception is thrown.

Both exceptions are part of **java.lang** or **java.io** packages and must be handled using try-catch blocks to avoid crashes.

Algorithm:

1. Start the program.
2. ClassNotFoundException part:
    - Use Class.forName("DummyClass"), where "DummyClass" is a class that does not exist.
    - Wrap this line in a try-catch block.
    - Catch ClassNotFoundException and display a message like "Class not found".
3. EOFException part:
    - Create a small data file and write one or two values into it using ObjectOutputStream.
    - Open the file using ObjectInputStream and try reading multiple times, more than the available data.
    - Wrap this read operation in a try-catch block.

- Catch EOFException and display a message like “Reached end of file unexpectedly.”

4. End the program.

Pseudo Code:

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

int main() {
   FILE *file;
   char ch;

   // --------- Simulate ClassNotFound by trying to open a non-existent file ---------
   file = fopen("non_existing_class.txt", "r");
   if (file == NULL) {
      printf("ClassNotFound simulation: File 'non_existing_class.txt' not found.\n");
   } else {
      fclose(file);
   }

   // --------- EOFException Simulation ---------
   file = fopen("input_Lab07.txt", "r");

   if (file == NULL) {
        printf("EOFException simulation: 'input.txt' file not found. Create this file to test EOF.\n");
   } else {
      printf("Reading characters until EOF:\n");

      while ((ch = fgetc(file)) != EOF) {
         putchar(ch);
      }

      printf("\nReached end of file (EOF).\n");
      fclose(file);
   }

   return 0;
}
```

Sample Output:

```
ClassNotFound simulation: File 'non_existing_class.txt' not found.
Reading characters until EOF:
Hi There My Name is Sayeeda Khan I am the final semester student at Department of Information and Communication Engineering, Pabna University of Science and Technology.
Reached end of file (EOF).
```

Experiment Number: 08

Name of the experiment: 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 aim is to create a program that reads a list of positive integers from an input.txt file and performs addition, subtraction, multiplication, and division on them. The results should be saved in an output.txt file, with proper formatting as shown in the sample output.

Theory: In real-world programming, file handling is essential for dealing with large data. This program shows how we can read values from a file, perform calculations, and write results back to another file. We focus on:

- Reading input from a text file (input.txt) containing multiple sets of numbers.
- Applying four arithmetic operations:
    - Addition: sum of all numbers.
    - Subtraction: subtract numbers one by one from the first.
    - Multiplication: product of all numbers.
    - Division: divide first number by the rest one-by-one (integer division).
- Writing output into output.txt with labeled cases like Case-1, Case-2, etc.

This program strengthens the understanding of file I/O, loops, arrays, and basic arithmetic logic.

Algorithm:

1. Start the program.
2. Open the input.txt file in read mode.
3. Open the output.txt file in write mode.
4. Repeat until end of input file:
    - Read a line of integers.
    - Store the numbers in an array.
    - Initialize variables: sum, sub, mul, div.
    - Addition: Set sum = 0, loop through array, add each value.
    - Subtraction: Set sub = first number, then subtract the rest one by one.
    - Multiplication: Set mul = 1, loop through array, multiply each value.
    - Division: Set div = first number, then divide one-by-one by the remaining numbers using integer division.
5. Use a case counter (e.g., Case-1, Case-2, etc.) to label each set.

6. Write the results of each arithmetic operation into output.txt following the given format.
7. After processing all lines, close both files.
8. End the program.

Pseudo Code:

```
#include <stdio.h>
int main() {
    FILE *fin, *fout;
    int a, b, caseNo = 1;
    // Open files
    fin = fopen("input_Lab08.txt", "r");
    fout = fopen("output_Lab08.txt", "w");
    if (fin == NULL) {
        printf("Error: Cannot open input.txt\n");
        return 1;
    }
    if (fout == NULL) {
        printf("Error: Cannot create output.txt\n");
        fclose(fin);
        return 1;
    }
    // Read pairs from input file and process
    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; // avoid division by 0

        fprintf(fout, "Case-%d: %d %d %d %d\n", caseNo, sum, diff, prod, div);
        caseNo++;
    }
    printf("Processing complete. Check output.txt\n");
    fclose(fin);
    fclose(fout);
    return 0;
}
```

Sample Input:

```
5 5 9 8
```

Sample Output:

```
Case-1: 10 0 25 1
Case-2: 17 1 72 1
```

Experiment Number: 09

Name of the experiment: 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 modern biomedical engineering by designing and developing reliable, safe, and efficient software systems used in medical devices, diagnostics, and healthcare applications. In this field, the focus is on solving real-life medical problems using technology.

Key Contributions:

- Medical Imaging Software: Software engineers develop programs to analyze MRI, CT, and X-ray images using advanced image processing and machine learning.
- Patient Monitoring Systems: Real-time monitoring tools for heart rate, oxygen level, and blood pressure rely on embedded software for accurate and stable functioning.
- Health Information Systems: Electronic Health Records (EHR) and hospital management systems are designed using software engineering principles to store and secure patient data.
- Bio-signal Processing: Signals like ECG or EEG are processed and analyzed using custom software to assist doctors in diagnosis.
- Simulation and Modeling: Software tools simulate human body responses to surgeries or drug interactions for training and research.

Summary: In biomedical engineering, software engineering bridges the gap between medicine and technology, improving diagnosis, treatment, and patient care.

Role of Software Engineering in Artificial Intelligence (AI)

Artificial Intelligence is heavily dependent on software engineering to build smart systems that can learn, reason, and act like humans. Software engineering ensures that these systems are scalable, maintainable, and robust.

Key Contributions:

- Algorithm Development: Software engineers implement AI algorithms such as machine learning, deep learning, and reinforcement learning in real-world applications.
- Data Handling: AI models require massive amounts of data, and software engineers create systems to collect, clean, and manage that data effectively.
- Model Deployment: Software engineering ensures AI models are properly integrated into web apps, mobile apps, or devices.
- Performance and Optimization: Engineers fine-tune software for better speed, accuracy, and efficiency in AI tasks.

- Ethics and Security: Secure coding practices and ethical AI use are ensured through proper software engineering processes.

Summary: In AI, software engineering is the backbone that turns theoretical models into real-world intelligent systems.

Role of Software Engineering in Robotics

In robotics, software engineering is responsible for programming the robot's "brain." It controls everything from sensing the environment to making decisions and taking actions.

Key Contributions:

- Robot Control Systems: Engineers write software that controls motors, sensors, and actuators in robots.
- Path Planning & Navigation: Robots use algorithms to move intelligently, avoid obstacles, and navigate environments — all implemented through software.
- Human-Robot Interaction (HRI): Voice commands, gesture control, and facial recognition are developed with software tools.
- Embedded Systems Programming: Robotics relies on embedded software that runs on microcontrollers and ensures real-time performance.
- Autonomous Behavior: From warehouse robots to drones and humanoids, all autonomous behavior is guided by code designed using software engineering methods.

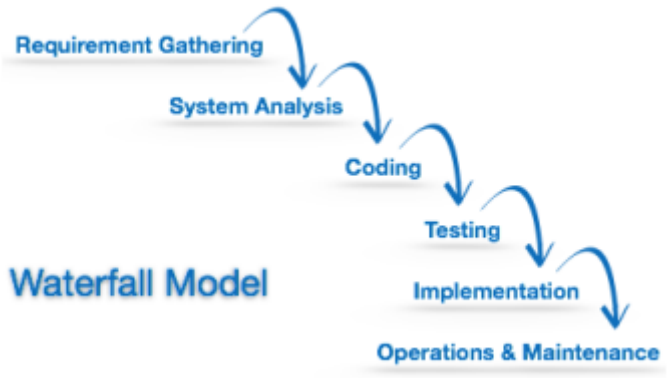
Summary: In robotics, software engineering transforms mechanical machines into intelligent, responsive, and autonomous systems.

Conclusion

Software engineering acts as the foundation for innovation in Biomedical Engineering, AI, and Robotics. It ensures that these technologies are not only functional but also efficient, user-friendly, and reliable. Whether it's saving lives through medical devices, creating intelligent AI models, or building smart robots — software engineering is at the core of it all.

Experiment Number: 10

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



Phases of the Waterfall Model (Detailed Explanation)

The Waterfall Model is one of the oldest and simplest software development methodologies. It follows a linear and sequential approach, where the output of one phase becomes the input for the next. Each phase has specific goals and deliverables.

1. Requirement Analysis

In this phase, the development team collects all necessary requirements from the client or stakeholders. These requirements describe what the software should do, including:

- Functional requirements (e.g., login system, payment gateway).
- Non-functional requirements (e.g., performance, security, usability).

The team prepares a Software Requirement Specification (SRS) document, which acts as the agreement between developers and clients.

- Goal: Understand and finalize “what” is to be built.

2. System Design

Using the SRS, the design team plans how to build the software. This includes:

- High-level system architecture (how components interact).
- Database schema and data flow.
- Interface design and user experience planning.

Design documents are created, including:

- HLD (High-Level Design)
- LLD (Low-Level Design)

- Goal: Plan the structure and blueprint of the system before coding.

3. Implementation (Coding)

Now, developers start writing the code based on the finalized design. Tasks are divided among different teams or modules, and the software begins to take shape. Programming languages, tools, and technologies are chosen according to the design.

- Goal: Convert the design into a working software product.

4. Integration and Testing

Once coding is completed, the different modules are integrated and tested as a complete system. This includes:

- Unit Testing (module-wise testing)
- System Testing (whole system)
- Integration Testing (check if modules work together)
- User Acceptance Testing (final user review)

- Goal: Detect and fix bugs or errors to ensure the system behaves as expected.

5. Deployment

After successful testing, the final software is delivered to the customer and installed in the live environment. Training and user manuals may be provided at this stage.

- Goal: Make the software available for real-world use.

6. Maintenance

After deployment, the software may need updates due to:

- New user requirements
- Bug fixes
- Changing environments (hardware, operating system)

This phase ensures the software remains functional and up to date over time.

- Goal: Keep the software running smoothly post-deployment.

Most Dominant Phase: Requirement Analysis

Among all the phases, Requirement Analysis is the most dominant and crucial in the Waterfall Model.

Reasons:

1. Foundation of All Other Phases: If the requirements are wrong, unclear, or incomplete, then the design, code, and testing will all be based on incorrect assumptions — causing the entire project to fail.
2. No Going Back: The Waterfall model is a rigid model. Once a phase is completed, it is difficult and expensive to go back and make changes. So, requirements must be clear from the start.

3. Cost of Mistakes: Any error made in this phase can remain hidden until the final phases, where fixing it could cost more money, time, and effort.
4. Defines Project Success: A project with well-understood and well-documented requirements has a much higher chance of successful delivery.
5. Customer Satisfaction: This phase ensures the final product will meet the client's expectations if the needs are properly analyzed and documented at the beginning.

Conclusion: The Waterfall Model divides the software development life cycle into structured, non-overlapping stages. While each phase plays an important role, the Requirement Analysis phase is the most dominant, as it shapes the entire direction and success of the project. A small mistake here can cause big failures later, making it the foundation upon which all other phases rest.

Experiment Number: 11

Name of the experiment: Using COCOMO model estimate effort for specific problem in industrial domain.

Problem Context (Industrial Domain Example)

Let's consider a Factory Automation Software that controls assembly line operations and machinery using sensor data. The system also includes dashboards for engineers and alerts for maintenance teams.

Suppose the estimated size of the software is 50 KLOC (Kilo Lines of Code), and we are using the Basic COCOMO Model to estimate the effort.

COCOMO Model: COCOMO (Constructive Cost Model) is a software cost estimation model developed by Barry Boehm. It estimates:

- Effort (in person-months),
- Time (in months), and
- Cost based on the project size (in KLOC).

There are 3 versions:

1. Basic COCOMO – quick estimation
2. Intermediate COCOMO – considers cost drivers
3. Detailed COCOMO – includes each development phase

COCOMO Project Categories

To use the model, we must first identify which type of software we're dealing with:

Project Type	Description	Typical Domain
Organic	Simple software with small teams, good experience	Accounting tools, inventory apps
Semi-detached	Mixed experience and medium complexity	Compilers, database apps
Embedded	Complex systems with hardware/software integration	Real-time control systems, industrial apps

Since this is factory automation software, which interacts with hardware and involves real-time decisions — it falls under the Embedded category.

Basic COCOMO Model Formula

The formula for effort estimation is:

$$Effort = a \times (KLOC)^b$$

Where:

- PM = Person-Months (total effort)
- KLOC = Thousands of Lines of Code
- a, b = Constants depending on the project type

For Embedded systems:

- a=3.
- b=1.20

Effort Estimation (Example)

Let's say the estimated size of the system is 50 KLOC.

$$Effort = 3.6 \times (50)^{1.20}$$

First calculate $(50)^{1.20}$**:**

$$(50)^{1.20} \approx 109.95$$

$$Effort = 3.6 \times 109.95 = 395.82 \text{ person} - \text{months}$$

Time Estimation

To estimate the development time (in months):

$$Time\ (TDEV) = c \times (Effort)^{d}$$

For Embedded:

- c=2.5
- d=0.32

$$Time\ (TDEV) = 2.5 \times (395.82)^{0.32} \approx 17.42 \text{ months}$$

Final Answer

- Estimated Effort: ≈ 396 person-months
- Estimated Development Time: ≈ 17.4 months

Summary

Using the COCOMO Basic Model, we estimated that developing a 50 KLOC industrial factory automation software (embedded system) will take approximately:

- 396 person-months of effort, and
- 17.4 months of calendar time.

Experiment Number: 12

Name of the experiment: 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".

Software Crisis

The term software crisis refers to the challenges faced in software development such as:

- Projects taking longer than expected,
- Software not working correctly,
- Difficulties in maintaining and understanding large codebases.

It happens because of poor planning, fast delivery without proper testing, unclear requirements, and complex systems.

Common Reasons Behind Software Crisis

1. Lack of proper testing
2. Poor time estimation
3. Unclear or incomplete requirements
4. Over-complex design
5. Inadequate team skills or communication
6. Insufficient documentation
7. No proper debugging or error-handling mechanisms

Case 1: Airport Ticket Reservation Software Crash

The software was working well for the first 12 hours but crashed exactly at 12.00 PM noon the next day. It took the developers 5 hours to fix the issue.

Possible Reason (Cause of Crisis):

- The bug might have been caused by a time format issue (for example, confusion between AM/PM or 12-hour vs. 24-hour clock).

- It may also have lacked real-time stress testing or testing during transitions between AM and PM.
- The software might not have been tested for long continuous usage (like 24 hours).

Solution:

- Perform extensive time-based testing, especially around time boundaries like 12:00 AM and 12:00 PM.
- Use automated test cases that simulate real-world time and load.
- Add proper error-handling routines to avoid complete system failure.
- Have a rollback plan or backup system in case of live failures.

Case 2: Financial System with Undetected Error

The customer reported a malfunction, but the software was so large and complex that the developers couldn't find the issue quickly.

Possible Reason (Cause of Crisis):

- The software likely lacked modularity and proper documentation, making it hard to trace bugs.
- It's possible the team didn't use debugging tools or logging systems.
- The design may have been too tightly coupled, meaning changes in one part affected many other parts.

Solution:

- Use modular programming to separate parts of the system for easier debugging.
- Add detailed logs and trace messages to find the root of issues.
- Maintain clear documentation for each module and function.
- Apply code reviews and regular testing during development to reduce future errors.
- Use version control and error tracking tools to handle complex software better.

Summary Table

Case	Cause	Solution
Case 1	Time-based bug, lack of boundary testing	Time simulation tests, better error handling, backup systems
Case 2	Complex structure, poor documentation	Modular design, logging, debugging tools, regular code reviews