Pabna University of Science & Technology

Department of Information & Communication Engineering

Faculty of Engineering & Technology

<u>LAB REPORT</u>

B.Sc.(Engineering), 4th Year 2nd Semester Examination-2023
Course code: ICE-4204
Course Title: System Analysis & Software Testing Sessional

Submitted By:

Name:Z.H.M Khairul Basar
Roll:190624
Semester: 4th Year 2nd Semester
Session: 2018-2019
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 No: 01

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

Objective:

To develop a basic calculator using the C programming language that takes an arithmetic expression in the format number1 operator number2 = and displays the result

Algorithm:

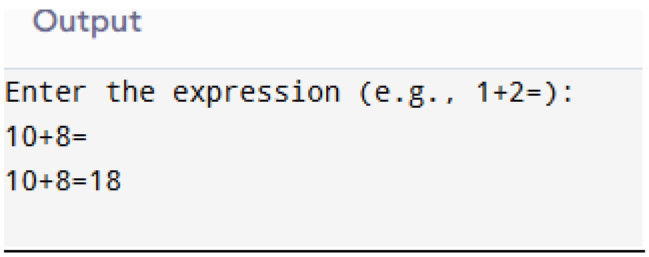
1. **Start**
2. **Input** the expression in the format: operand1 operator operand2
3. Parse the first number (operand1)
4. Parse the operator (+, -, *, /, %)
5. Parse the second number (operand2)
6. **Wait for the '=' input**
7. **Evaluate** the expression based on the operator
8. **Display** the result in the format: operand1 operator operand2 = result
9. **End**

Source Code:

```
#include <stdio.h>

int main() {
  int num1, num2, result;
  char op, eq;

  printf("Enter expression (e.g., 1+2=): ");
  scanf("%d%c%d%c", &num1, &op, &num2, &eq);

  if (eq != '=') {
    printf("Invalid format. Please end the expression with '='.\n");
    return 1;
  }
```

```
    switch(op) {
        case '+':
            result = num1 + num2;
            printf("%d%c%d=%d\n", num1, op, num2, result);
            break;
        case '-':
            result = num1 - num2;
            printf("%d%c%d=%d\n", num1, op, num2, result);
            break;
        case '*':
            result = num1 * num2;
            printf("%d%c%d=%d\n", num1, op, num2, result);
            break;
        case '/':
            if (num2 == 0) {
                printf("Division by zero error!\n");
                return 1;
            }
            result = num1 / num2;
            printf("%d%c%d=%d\n", num1, op, num2, result);
            break;
        case '%':
            if (num2 == 0) {
                printf("Modulo by zero error!\n");
                return 1;
            }
            result = num1 % num2;
            printf("%d%c%d=%d\n", num1, op, num2, result);
            break;
        default:
            printf("Invalid operator!\n");
            return 1;
    }

    return 0;
}
```

<u>Output:</u>



Output

```
Enter the expression (e.g., 1+2=):
10+8=
10+8=18
```

Problem N0: 02

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

Objective:
To develop a C program that reads two sets of n integers followed by an arithmetic operator, and performs the given operation (+, -, *, /, %) on each corresponding pair of integers.

Algorithm:

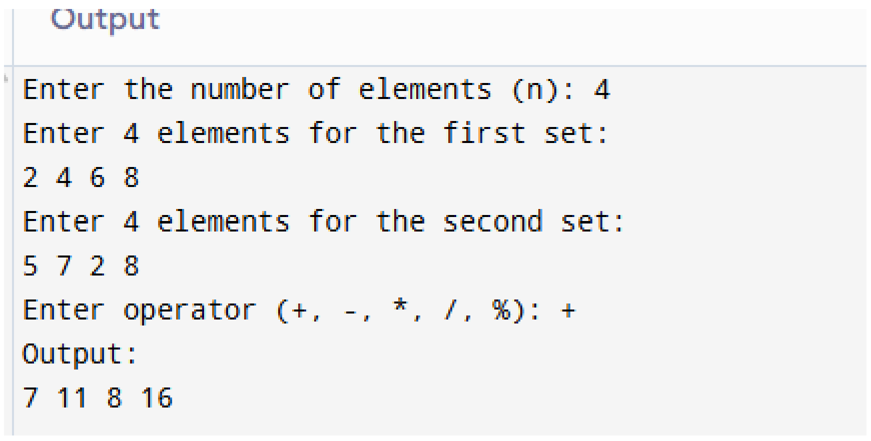
1. **Start**
2. Initialize an array to store integers
3. Read integers until a non-integer (the operator) is encountered
4. If the number of integers is odd, display error and exit
5. Read the operator (+, -, *, /, %)
6. For each pair in the input, apply the operator:

   a[i] op a[i+1]
7. Print the result for each operation
8. **End**

Source Code:

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

int main() {
    int arr[100], count = 0;
    char ch;
    printf("Enter even number of integers followed by an operator (e.g. 4 5 7 8 +):\n");

    // Read integers until a non-digit character (the operator) is encountered
    while (scanf("%d", &arr[count]) == 1) {
        count++;
    }

    // Now read the operator (non-digit character that stopped the input loop)
    scanf(" %c", &ch);
    // Check for even number of integers
    if (count % 2 != 0) {
        printf("Error: Odd number of integers entered.\n");
        return 1;
    }

    printf("Output: ");
    for (int i = 0; i < count; i += 2) {
        int a = arr[i];
        int b = arr[i + 1];
        int result;

        switch (ch) {
            case '+':
                result = a + b;
                break;
            case '-':
                result = a - b;
                break;
            case '*':
                result = a * b;
                break;
            case '/':
                if (b == 0) {
                    printf("Error(div0) ");
                    continue;
                }
                result = a / b;
                break;
            case '%':
                if (b == 0) {
                    printf("Error(mod0) ");
                    continue;
                }
                result = a % b;
                break;
            default:
                printf("Invalid operator!\n");
                return 1;
        }

        printf("%d ", result);
    }

    printf("\n");
    return 0;
}
```

Output:



```
Enter the number of elements (n): 4
Enter 4 elements for the first set:
2 4 6 8
Enter 4 elements for the second set:
5 7 2 8
Enter operator (+, -, *, /, %): +
Output:
7 11 8 16
```

Problem No: 03

Problem Name: Write a program in "JAVA" or "C" to check weather a number or string is palindrome or not. N.B: your program must not take any test case number such as 1 or 2 for the desired cases from the user. Program user will insert a number or string as input directly and the program will display the exact result in the output console.

Objective:
To develop a C program that takes an input (either a number or a string) and checks whether it is a palindrome without using any test case number or predefined conditions.

Algorithm:

1. **Start**
2. Take a **single input** from the user (as a string to handle both text and numbers)
3. Initialize two pointers:

    i.start at beginning of string

    ii.end at end of string

4. While start < end:

    i.If characters at start and end don’t match → Not a palindrome

    ii.Otherwise, move start++ and end--

5. If the loop completes, it is a palindrome
6. Print the result
7. **End**

Source Code:

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

int main() {
    char input[100];
    int start, end, isPalindrome = 1;

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

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

    start = 0;
    end = strlen(input) - 1;

    while (start < end) {
        if (input[start] != input[end]) {
            isPalindrome = 0;
            break;
        }
        start++;
        end--;
    }

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

    return 0;
}
```

Output:

```
Enter a number or string: alamin

"alamin" is not a palindrome.


Enter a number or string: 2002

"2002" is a palindrome.
```

Problem No: 04

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

Solution:

ATM System Specifications and Bug Report:

System Specifications

1. Functional Requirements

1. **Card Authentication**
    - Accepts debit/credit cards with magnetic stripe or chip
    - Validates card expiration date
    - Verifies card against stolen/lost card database
    - Limits to 3 invalid PIN attempts before retaining card
2. **Transaction Processing**
    - Cash withdrawals (multi-denomination support)
    - Balance inquiries
    - Fund transfers between accounts
    - Mini-statement generation
    - PIN change functionality
    - Deposit processing (cash/check)
3. **Security Features**
    - Encrypted PIN entry
    - Session timeout after 2 minutes of inactivity
    - Transaction limits (daily/transaction-based)
    - EMV chip authentication for chip cards
    - Anti-skimming detection
4. **Hardware Integration**
    - Card reader (magnetic + chip)

- Cash dispenser (with counterfeit detection)
- Receipt printer
- Deposit acceptor
- Touchscreen/keypad interface
- Camera surveillance

5. **Network Communication**
   - Real-time authorization with bank servers
   - Fallback to offline mode when network fails
   - End-to-end encryption for all transactions
   - Daily reconciliation process
6. **User Interface**
   - Multilingual support
   - Accessibility features (voice guidance, tactile buttons)
   - Clear transaction confirmation screens
   - Receipt option selection

Bug Report:

Critical Bugs

1. **Card Retention Failure**
   - After 3 failed PIN attempts, system sometimes fails to retain the card
   - Reproduction rate: ~1 in 50 occurrences
   - Security risk: Allows unlimited PIN attempts
2. **Cash Dispensing Error**
   - Occasionally dispenses incorrect denominations (e.g., 50insteadof50*insteadof*20)
   - Occurs more frequently when dispensing large amounts
   - Hardware logs show cassette misalignment during these events
3. **Double Debit Issue**

- Network timeout sometimes causes duplicate transactions
- Particularly occurs during peak hours
- Requires manual reversal by bank staff

High Severity Bugs:

4. **Session Timeout Bypass**
    - Pressing specific key combinations maintains session beyond timeout period
    - Security risk if user walks away from active session
5. **Balance Display Error**
    - Occasionally shows incorrect account balance
    - Appears to be caching issue during network latency
    - Refresh shows correct balance
6. **Deposit Processing Fault**
    - Some check deposits are misread (amount recognition error)
    - Occurs with handwritten checks more frequently
    - Requires manual verification

Medium Severity Bugs:

7. **Receipt Printer Jam**
    - Printer frequently jams when printing long receipts
    - Requires service call to clear
    - Design flaw in paper path mechanism
8. **Language Selection Reset**
    - System reverts to default language after each transaction
    - Inconvenience for non-default language users
9. **Touchscreen Calibration**
    - Lower right corner becomes unresponsive after prolonged use
    - Requires periodic recalibration

Low Severity Bugs:

10. **UI Alignment Issue**
    - Some screen elements misaligned in certain language versions
    - Cosmetic issue only
11. **Audit Log Timestamp**
    - Logs occasionally show incorrect timestamps during daylight saving transitions
12. **Card Ejection Sound**
    - Missing audio confirmation when card is returned
    - Accessibility concern

Recommended Actions:

1. Immediate patch for card retention and cash dispensing issues (critical)
2. Network transaction logic overhaul to prevent duplicate debits
3. Firmware update for deposit module's OCR capabilities
4. Scheduled maintenance for hardware recalibration
5. UI/UX review for language persistence and accessibility features

System Performance Metrics:

- Uptime: 99.2%
- Average transaction time: 45 seconds
- Most frequent service call: Cash dispenser jams (38% of calls)
- Customer complaint rate: 1.2 per 10,000 transactions

Problem No: 05

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

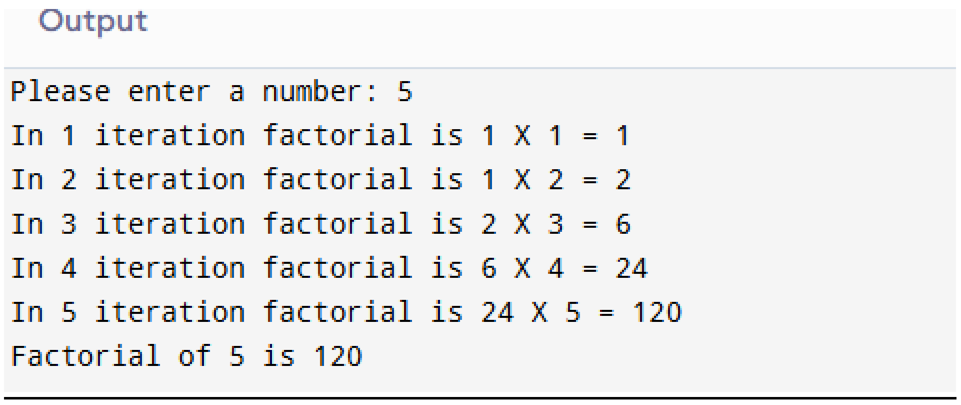
Algorithm:

```
1. Start
2. Prompt user to enter a positive integer n
3. Initialize fact1 = 1, fact2 = 1
4. Use a for loop to compute fact1 = n!
5. Use a while loop to compute fact2 = n!
6. Compare fact1 and fact2
7. Print both factorial results
8. If both results match, print "Verified!"
9. End
```

Source Code:

```c
#include <stdio.h>
int main() {
   int n, i;
   unsigned long long fact = 1, prev;
   printf("Please enter a number: ");
   scanf("%d", &n);
   for (i = 1; i <= n; i++) {
      prev = fact;
      fact *= i;
      printf("In %d iteration factorial is %llu X %d = %llu\n", i, prev, i, fact);
   }
   printf("Factorial of %d is %llu\n", n, fact);
   return 0;
}
```

Output:



```
Please enter a number: 5
In 1 iteration factorial is 1 X 1 = 1
In 2 iteration factorial is 1 X 2 = 2
In 3 iteration factorial is 2 X 3 = 6
In 4 iteration factorial is 6 X 4 = 24
In 5 iteration factorial is 24 X 5 = 120
Factorial of 5 is 120
```

Problem No: 06

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

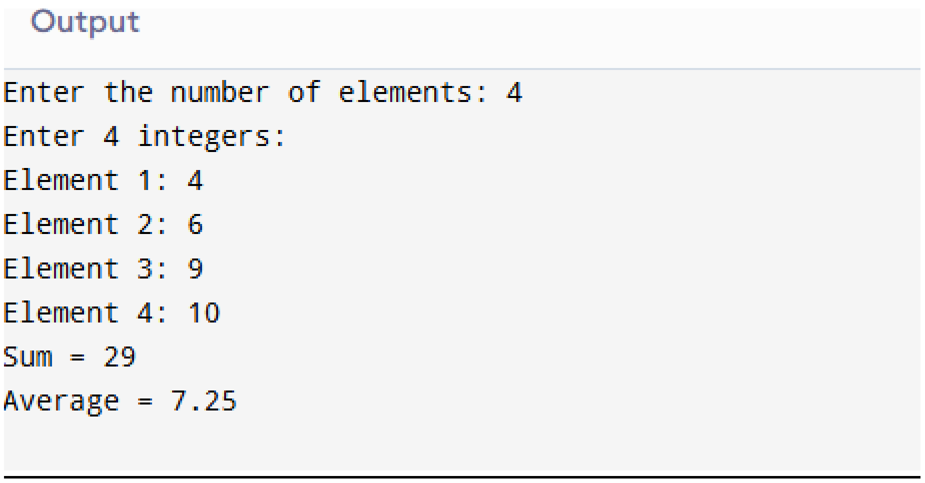
Algorithm:

1. **Start**
2. Prompt the user to enter the size of the array n
3. Declare an array of size n
4. Read n elements from the user using a do...while loop
5. Call calculateSum() function to get the sum
6. Call calculateAverage() function to compute the average using the sum
7. Display the sum and average
8. **End**

Source Code:

```
#include <stdio.h>

// Function to calculate sum of array elements
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 elements
float calculateAverage(int sum, int n) {
    return (float)sum / n;
}

int main() {
    int n, i = 0;
    int arr[100]; // Assumes array size won't exceed 100

    printf("Enter the number of elements: ");
    scanf("%d", &n);
    // Input elements using do...while loop
    printf("Enter %d integers:\n", n);
    do {
        printf("Element %d: ", i + 1);
        scanf("%d", &arr[i]);
        i++;
    } while (i < n);

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

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

    return 0;
}
```

Output:



```
Enter the number of elements: 4
Enter 4 integers:
Element 1: 4
Element 2: 6
Element 3: 9
Element 4: 10
Sum = 29
Average = 7.25
```

Problem No: 07

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

Algorithm:

For ClassNotFoundException

1. Use Class.forName("ClassName") to load a class dynamically.
2. If the class does **not exist**, ClassNotFoundException is thrown.
3. Catch and handle the exception.

For EOFException

1. Write some data to a file using ObjectOutputStream.
2. Read the data using ObjectInputStream in a loop.
3. When end of file is reached, EOFException is thrown.
4. Catch and handle the exception.

Source Code:

```
import java.io.*;

// Main class
public class ExceptionDemo {

    public static void main(String[] args) {

        // Demonstrating ClassNotFoundException
        try {
            // Attempt to load a non-existent class
            Class.forName("com.example.NonExistentClass");
        } catch (ClassNotFoundException e) {
            System.out.println("ClassNotFoundException caught: " + e.getMessage());
        }

        // Demonstrating EOFException
        String filename = "data.txt";

        // Write sample integers to file
        try (ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream(filename))) {
            out.writeInt(100);
            out.writeInt(200);
            out.writeInt(300);
        } catch (IOException e) {
            System.out.println("IOException during writing: " + e.getMessage());
        }

        // Read the integers and intentionally trigger EOFException
        try (ObjectInputStream in = new ObjectInputStream(new FileInputStream(filename))) {
            while (true) {
                int num = in.readInt(); // will throw EOFException after last integer
                System.out.println("Read: " + num);
            }
        } catch (EOFException e) {
            System.out.println("EOFException caught: End of file reached.");
        } catch (IOException e) {
            System.out.println("IOException during reading: " + e.getMessage());
        }
    }
}
```

Output:

```
ClassNotFoundException caught: com.example.NonExistentClass
Read: 100
Read: 200
Read: 300
EOFException caught: End of file reached.
```

Problem No: 08

Problem Name: Write a program in "JAVA" or "C" that will read a input.txt file containing n positive integers and calculate addition, subtraction, multiplication and division in separate output.txt file. Sample input: 5 5 9 8; Sample output: Case-1:10 0 25 1; Case-2: 17 1 72 1.

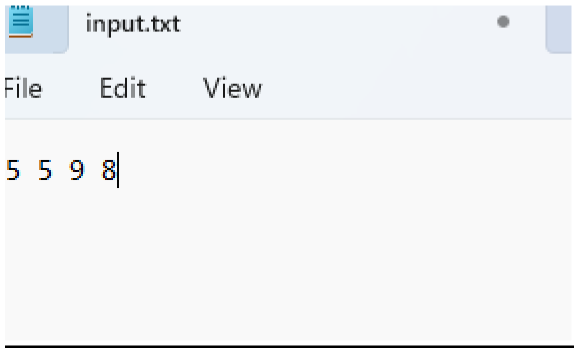
Algorithm:

1. Open input.txt for reading
2. Read all positive integers into an array
3. For every **pair of numbers** (a, b):
    - Calculate a+b, a-b, a*b, a/b (use integer division)
    - Write results in format: Case-i: sum diff prod div
4. Save all results to output.txt
5. Close files

Source Code:

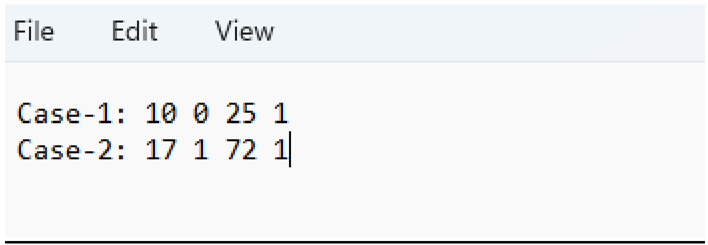
```
#include <stdio.h>
#include <stdlib.h>

int main() {
    FILE *fin = fopen("input.txt", "r");
    FILE *fout = fopen("output.txt", "w");

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

    int numbers[100];
    int n = 0;

    // Read integers until EOF or max limit
    while (fscanf(fin, "%d", &numbers[n]) == 1) {
        if (numbers[n] <= 0) {
            printf("Only positive integers allowed.\n");
            fclose(fin);
            fclose(fout);
            return 1;
        }
        n++;
        if (n >= 100) break;  // avoid overflow
    }

    if (n < 2) {
        printf("Not enough numbers in input file.\n");
        fclose(fin);
        fclose(fout);
        return 1;
    }

    // Process pairs: (numbers[0], numbers[1]), (numbers[2], numbers[3]), ...
    int case_num = 1;
    for (int i = 0; i < n - 1; i += 2) {
        int a = numbers[i];
        int b = numbers[i + 1];

        int addition = a + b;
        int subtraction = a - b;
        int multiplication = a * b;
        int division = (b != 0) ? (a / b) : 0;  // avoid divide by zero

        fprintf(fout, "Case-%d: %d %d %d %d\n", case_num, addition, subtraction, multiplication,
division);
        case_num++;
    }

    fclose(fin);
    fclose(fout);

    printf("Operation completed. Check output.txt for results.\n");
    return 0;
}
```

Input file:



```
5 5 9 8
```

Output file:



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

Problem No: 09

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

Solution:

The Role of Software Engineering in Biomedical Engineering, AI, and Robotics

Software engineering plays a critical role in modern technological fields, particularly in biomedical engineering, artificial intelligence (AI), and robotics. Here's a detailed explanation of its importance in each domain:

1. Software Engineering in Biomedical Engineering

Medical Device Development

- **Embedded systems programming**: Developing software for pacemakers, insulin pumps, and other implantable devices
- **Safety-critical systems**: Implementing fail-safes and redundancy in life-support equipment
- **Regulatory compliance**: Ensuring software meets FDA (Class I-III) and other medical device regulations

Medical Imaging and Diagnostics

- **Image processing algorithms**: Developing software for MRI, CT, and ultrasound image reconstruction
- **Computer-aided diagnosis**: Creating machine learning models for tumor detection, fracture identification, etc.
- **DICOM standard implementation**: Building systems that handle medical imaging data exchange

Health Information Systems

- **Electronic Health Records (EHR)**: Designing secure, interoperable patient data systems
- **Telemedicine platforms**: Developing real-time video consultation and remote monitoring solutions
- **Clinical decision support**: Creating systems that assist physicians with treatment recommendations

Biomedical Data Analysis

- **Genomics software**: Tools for DNA sequencing and analysis
- **Bioinformatics pipelines**: Processing and interpreting biological data
- **Wearable health tech**: Developing algorithms for fitness trackers and medical wearables

2. Software Engineering in Artificial Intelligence

Core AI Development

- **Algorithm implementation**: Efficient coding of machine learning models (neural networks, decision trees, etc.)
- **Framework development**: Creating libraries like TensorFlow, PyTorch, or scikit-learn
- **Optimization**: Improving model performance through software techniques

AI System Architecture

- **Distributed computing**: Building systems for parallel training of large models
- **Pipeline design**: Creating reproducible ML workflows from data ingestion to deployment
- **Model serving**: Developing infrastructure to deploy AI models at scale

AI Applications

- **Computer vision**: Software for image recognition, object detection
- **Natural language processing**: Chatbots, translation systems, text analysis
- **Recommendation systems**: Personalization engines for e-commerce and content platforms

AI Safety and Ethics

- **Bias detection**: Software tools to identify and mitigate model biases
- **Explainability**: Developing interfaces that explain AI decisions
- **Robustness testing**: Creating frameworks to test AI system vulnerabilities

3. Software Engineering in Robotics

Robot Control Systems

- **Real-time operating systems**: Software for deterministic control loops
- **Motion planning**: Algorithms for path finding and obstacle avoidance

- **Sensor fusion**: Integrating data from multiple sensors (LiDAR, cameras, IMUs)

Robot Software Architecture

- **Middleware development**: ROS (Robot Operating System) and similar frameworks
- **Simulation environments**: Building virtual testing platforms like Gazebo
- **Firmware development**: Low-level code for motor controllers and actuators

Autonomous Systems

- **Perception systems**: Software for object recognition and scene understanding
- **Decision making**: Implementing behavior trees and state machines
- **Localization and mapping**: SLAM (Simultaneous Localization and Mapping) algorithms

Human-Robot Interaction

- **Gesture recognition**: Software for natural human-robot interfaces
- **Voice control**: Integrating speech recognition systems
- **Collaborative robotics**: Software for safe human-robot workspaces

Cross-Cutting Contributions of Software Engineering

Common Challenges Across Domains

1. **Safety and reliability**: Especially critical in biomedical and robotics applications
2. **Real-time performance**: Many applications require deterministic timing
3. **System integration**: Combining hardware and software components
4. **Data security**: Protecting sensitive medical and personal data
5. **Verification and validation**: Proving systems work as intended

Emerging Trends

- **AI-powered medical diagnostics**: Combining biomedical engineering with machine learning
- **Surgical robotics**: Merging robotics with medical applications
- **Edge AI**: Implementing AI directly on medical devices and robots
- **Digital twins**: Virtual models of physical systems for testing and monitoring

Software Engineering Methodologies Applied

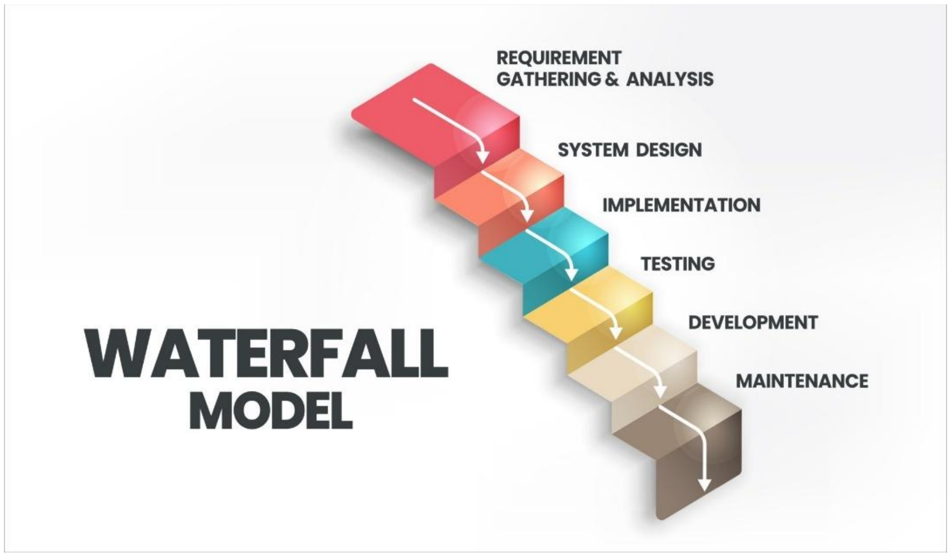
- **Agile development**: For iterative improvement of complex systems
- **Model-based design**: Using simulations to validate systems before deployment
- **Formal methods**: Mathematical verification of critical systems
- **DevOps/MLOps**: Continuous integration/deployment for AI and robotic systems

The integration of software engineering principles in these fields has enabled groundbreaking advancements - from AI-assisted cancer detection to autonomous surgical robots and smart prosthetics. As these technologies continue to evolve, software engineering will remain fundamental to their development, deployment, and ongoing improvement.

Problem Name: 10

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

Solution:



The **Waterfall Model** is a **linear, sequential** approach to software development, where each phase must be completed before moving to the next. It consists of the following phases:

1. **Requirements Gathering & Analysis**
    - Involves collecting and documenting all system requirements.
    - Stakeholders (clients, users, analysts) define what the software should do.
    - Output: **Software Requirements Specification (SRS) document**.
2. **System Design**
    - Translates requirements into a technical blueprint.
    - Defines system architecture, data structures, and algorithms.
    - Output: **Design Document (HLD & LLD - High-Level & Low-Level Design)**.
3. **Implementation (Coding)**
    - Developers write code based on the design documents.
    - Programming languages, frameworks, and tools are used.
    - Output: **Working software modules**.

4. **Testing**
    - The software is tested for defects (unit, integration, system, and acceptance testing).
    - Bugs are identified and fixed.
    - Output: **Test Reports & Debugged Software**.
5. **Deployment (Release & Maintenance)**
    - The software is deployed to the production environment.
    - Maintenance includes bug fixes, updates, and enhancements.
    - Output: **Live System & Maintenance Logs**.

Which Phase is the Most Dominant?

The **most dominant phase** in the Waterfall Model is:

1. Requirements Gathering & Analysis

Why?

- **Foundation of the entire project**: If requirements are incorrect or incomplete, all subsequent phases will be affected.
- **Difficult to modify later**: Since Waterfall is sequential, changing requirements mid-project requires restarting the process.
- **High cost of errors**: Mistakes in requirements lead to **expensive rework** in later stages.
- **Client & stakeholder dependency**: Proper documentation ensures alignment between developers and users.

Other Important Phases:

- **Testing** (if defects are found late, fixing them is costly).
- **Design** (poor architecture leads to scalability issues).

But **Requirements Phase dominates** because it sets the direction for the entire project.

Problem No: 11

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

Solution:

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:

● Sizeofthe 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 COCOMOModelType:

COCOMOhasseveral models based on the project type:

1. Basic COCOMO:Forearly, rough estimates.

2. Intermediate COCOMO: Includes cost drivers to refine the estimate.

3. Detailed COCOMO: Provides a more detailed estimate by including a detailed analysis of

project characteristics.

3. Apply Basic COCOMO Model

Basic COCOMOFormula: Effort=a×(KLOC)b}

Where:

● Effort is measured in person-months.

● KLOC is the estimated size of the software in thousands of lines of code.

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

Project Types:

1. **Organic** (simple projects):

a=2.4

$b=1.05$

2. **Semi-Detached** (medium complexity projects):

   $a=3.0$

   $b=1.12$

3. **Embedded**(complex projects):

   $a=3.6$

   $b=1.20$

4. Apply Intermediate or Detailed COCOMO Model (if needed)

● **Intermediate COCOMOadds** cost drivers for various factors affecting the project such as product reliability, software engineering capabilities, and hardware constraints.

● **Detailed COCOMOfurther** 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: $\text{Effort}=3.0\times(50)1.12$

Calculate: $(50)1.12\approx55.08$ $\text{Effort}=3.0\times55.08\approx165.24$

Problem No: 12

Problem Name: I dentify the reasons behind software crisis and explain the possible solutions for the

following scenario:

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

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

Solution:

The "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. Lackof 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 deployment.

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 software system.

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. CodeReviews 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.

Summary

- **Software Crisis Reasons:** Complexity, unclear requirements, poor project management, lack of testing, and inadequate documentation.
- **Case1Solutions:** Improved testing, robust monitoring, contingency planning, and post-deployment support.
- **Case2Solutions:** 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.