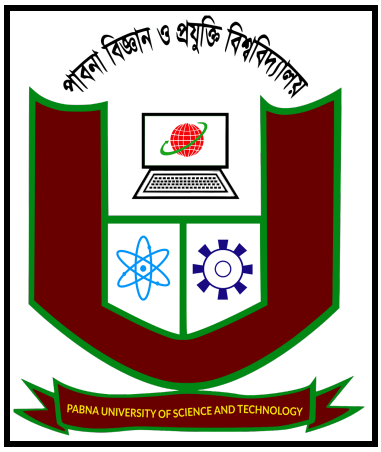

Pabna University of Science and Technology

Faculty of Engineering and Technology

Department of Information and Communication Engineering

4th year 2nd semester

Lab Report

Course Title: System Analysis and Software Testing Sessional

Course Code: ICE-4202

Session:2018-2019

Submitted by:	***Submitted To:***
Name: Md. Imran Hossain **Roll:190615** **Registration no:1065340** **Dept. of Information and Communication Engineering** **PUST**	**Md. Anwar Hossain** **Professor** **Dept. of Information and Communication Engineering** **PUST**

INDEX

08	Write a program in "JAVA" or "C" that will read an input.txt file containing n positive integers and calculate addition, subtraction, multiplication, and division in a separate output.txt file. Sample input: 5 5 9 8; Sample output: Case-1: 10 0 25 1; Case-2: 17 1 72 1.
09	Explain the role of software engineering in Biomedical Engineering and in the field of Artificial Intelligence and Robotics.
10	Study the various phases of the waterfall model. Which phase is the most dominant one?
11	Using the COCOMO model to estimate the effort for a specific problem in the industrial domain.
12	Identify the reasons behind the 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 (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. The customer confirmed with 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.

Problem Number: 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 afterpressing "=" sign. Sample input: 1+2; 8%4; Sample output: 1+2=3; 8%4=0.

Objectives:

- To implement a basic calculator that performs arithmetic operations: addition, subtraction, multiplication, division, and modulo.

- To take input in the format of `<number><operator><number>` and display the result after pressing the "=" key.

Algorithm:

1. Start the program.
2. Read a mathematical expression from the user in the format: operand1 operator operand2 (e.g., 1+2).
3. Parse the input to extract two operands and the operator.
4. Perform the calculation based on the operator:
    - `+` → Addition
    - `-` → Subtraction
    - `*` → Multiplication
    - `/` → Division (check for divide-by-zero)
    - `%` → Modulo (check for divide-by-zero)
5. Display the result in the format: operand1 operator operand2 = result
6. End the program.

Code (C Language):

```
#include<stdio.h>

int main() {

    int num1, num2;

    char operator;

    int result;

    printf("Enter the first number: ");
```

```
scanf("%d", &num1);
printf("Enter the second number: ");
scanf("%d", &num2);
printf("Enter the operator: ");
scanf("%c",operator);

switch (operator) {
    case '+':
        result = num1 + num2;
        printf("The result is: %d+%d=%d\n", num1, num2, result);
        break;
    case '-':
        result = num1 - num2;
        printf("The result is: %d-%d=%d\n", num1, num2, result);
        break;
    case '*':
        result = num1 * num2;
        printf("The result is: %d*%d=%d\n", num1, num2, result);
        break;
    case '/':
        if (num2 == 0) {
            printf("Error: Division by zero is not allowed.\n");
        } else {
            result = num1 / num2;
```

```
            printf("The result is: %d/%d=%d\n", num1, num2, result);
        }
        break;
    case '%':
        if (num2 == 0) {
            printf("Error: Modulo by zero is not allowed.\n");
        } else {
            result = num1 % num2;
            printf("The result is: %d%%%d=%d\n", num1, num2, result);
        }
        break;
    default:
        printf("Invalid operator.\n");
  }


  return 0;
}
```

Input:

Enter the first number: 9

Enter the second number: 3

Enter the operator: +

Output:

The result is: 9 + 3 = 12.

Problem Number: 02

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

Objectives:

- To read a list of integers followed by an operator.
- To perform the given arithmetic operation on every **pair of consecutive integers.**
- To output the result of each pair-wise operation.

Algorithm:

1. Start the program.
2. Read input integers from the user until an operator is encountered.
3. Store all the integers in an array.
4. Check that the total number of integers is even (so they can be paired).
5. Apply the operator to every pair (i.e., 0 & 1, 2 & 3, ...).
6. Print the result for each operation.
7. End the program.

Code (C Language):

```
#include <stdio.h>

#include <ctype.h>


#define MAX 100
```

```
int main() {
    int numbers[MAX], count = 0;
    char operator;

    printf("Enter input (e.g., 4 5 7 8 20 40 +):\n");

    while (scanf("%d", &numbers[count]) == 1) {
        count++;
    }
    scanf(" %c", &operator);
    if (count % 2 != 0) {
        printf("Invalid input: Odd number of integers.\n");
        return 1;
    }
    printf("Output:\n");
    for (int i = 0; i < count; i += 2) {
        int a = numbers[i];
        int b = numbers[i + 1];
        switch (operator) {
            case '+':
                printf("%d ", a + b);
                break;
            case '-':
                printf("%d ", a-b);
```

```
            break;
        case '*':
            printf("%d ", a * b);
            break;
        case '/':
            if (b == 0)
                printf("Error ");
            else
                printf("%d ", a / b);
            break;
        case '%':
            if (b == 0)
                printf("Error ");
            else
                printf("%d ", a % b);
            break;
        default:
            printf("Invalid operator.\n");
            return 1;
      }
   }
   printf("\n");
   return 0;
}
```

Input:

4 5 7 8 20 40 +

Output:

9 15 60

Problem Number: 03

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

Objectives:

- To develop a program that checks whether an input (number or string) is a palindrome.
- The program should accept the input directly, without asking for a test case count or input type.
- It should identify palindromes regardless of whether the input is a number or a string.

Algorithm:

1. Start the program.
2. Take a string input from the user.
3. Find the length of the input.
4. Compare the characters from the beginning and end, moving toward the center.
5. If all characters match in reverse order, it's a palindrome.
6. Display the result.
7. End the program.

Code (C Language):

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
int main() {
    char input[100];
    int len, i, isPalindrome = 1;
    printf("Enter a number or string: ");
    scanf("%s", input);
    len = strlen(input);
    for (i = 0; i < len / 2; i++) {
        if (input[i] != input[len - i - 1]) {
            isPalindrome = 0;
            break;
        }
    }
    if (isPalindrome)
        printf("%s is a palindrome.\n", input);
    else
        printf("%s is not a palindrome.\n", input);
    return 0;
}
```

Input:

Madam

12321

hello

Output:

madam is a palindrome.

12321 is a palindrome.

hello is not a palindrome.

Problem Number: 04

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

ATM System Specifications

1. System Overview:

An Automated Teller Machine (ATM) system allows users to perform basic banking transactions without the need for a human teller. This system provides services like balance inquiry, cash withdrawal, PIN change, mini statement, and fund transfer.

2. Functional Specifications:

Feature	Description
Card Authentication	User inserts a card and enters a valid PIN to access services.
PIN Validation	The system checks whether the entered PIN is correct.
Balance Inquiry	Displays the current balance in the user's account.
Cash Withdrawal	Allows users to withdraw cash if the balance is sufficient and limits are not exceeded.

Mini Statement	Displays the last 5–10 transactions.
Fund Transfer	Transfers money to another valid account number.
PIN Change	Allows users to change their ATM PIN.
Card Ejection	Ejects the card at the end of the session or upon user request.
Session Timeout	Ends the session after a fixed time of inactivity.

3. Non-Functional Specifications:

- **Security**: Data encryption during transactions and PIN entry.
- **Availability**: 24/7 uptime with backup in case of network failure.
- **Performance**: Fast response time (under 3 seconds per operation).
- **Usability**: Simple and intuitive user interface with clear instructions.
- **Hardware Integration**: Integration with card reader, keypad, cash dispenser, printer, and screen.

4. Assumptions and Constraints:

- ATM is connected to a core banking server via a secure network.
- Users have already registered bank accounts.
- ATM dispenses fixed denominations (e.g., ₹100, ₹500, ₹2000 notes).
- Daily withdrawal limit (e.g., ₹25,000/day).

Common Bugs in ATM Systems

Bug Category	Bug Description
Authentication Bugs	Accepts invalid PINs; allows too many incorrect attempts; doesn't block the card after 3 wrong attempts.
UI Bugs	Misaligned screen options, overlapping text, and non-responsive buttons.
Session Bugs	Session doesn't time out after inactivity; allows multiple simultaneous sessions.
Withdrawal Bugs	Allows overdraft; dispenses incorrect denominations; fails to update balance.
Concurrency Bugs	Two users accessing the same account simultaneously causes inconsistency.
Printing Bugs	Mini statement fails to print; printer stuck, but system proceeds.
Input Validation Bugs	Accepts invalid account numbers or alphabets in numeric fields.
Crash Bugs	The system crashes during network failure or when a card is forcefully removed.
Fund Transfer Bugs	Transfers to non-existent accounts; transfers incorrect amount.

PIN Change Bugs	Allows reuse of old PIN; fails to confirm new PIN properly.

Problem Number: 05

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

Objectives:

- Write a C program that calculates the factorial of a number using both `for` and `while` loops.
- To verify that both loops return the same correct result.

Algorithm:

1. Start the program.
2. Take a positive integer input from the user.
3. Use a `for` loop to calculate the factorial of the number and store the result.
4. Use a `while` loop to calculate the factorial of the same number again.
5. Compare the results obtained from both loops.
6. Display both results and the verification status.

Code (C Language):

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

```
int main() {
    int n, i;
    unsigned long long factorial_for = 1, factorial_while = 1;
    printf("Enter a positive integer: ");
    scanf("%d", &n);
    // Using for loop
    for (i = 1; i <= n; i++) {
        factorial_for *= i;
    }
    // Using while loop
    i = 1;
    while (i <= n) {
        factorial_while *= i;
        i++;
    }
    // Display results
    printf("Factorial using for loop: %llu\n", factorial_for);
    printf("Factorial using while loop: %llu\n", factorial_while);
    // Verification
    if (factorial_for == factorial_while)
        printf("Verification: Both results are equal.\n");
    else
        printf("Verification: Results do not match. Error detected.\n");
    return 0;
```

}

Input:

Enter a positive integer: 5

Output:

Factorial using a for loop: 120

Factorial using while loop: 120

Verification: Both results are equal.

Problem Number: 06

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

Objectives

- To learn how to use a `do-while` loop in C for array processing.
- To understand how to use user-defined functions to modularize code.
- To calculate the **sum** and **average** of elements in an array using functions.

Algorithm

1. Start the program.
2. Take the number of elements `n` from the user.
3. Read `n` integers into an array.
4. Define a function `calculateSum()` to compute the sum using a `do-while` loop.

5. Define another function, `calculateAverage()`, to compute the average using the sum.

6. Print the sum and average.

7. End the program.

Code (C Language):

```
#include <stdio.h>

// Function to calculate the sum using a do-while loop
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
float calculateAverage(int sum, int n) {
    return (float)sum / n;
}

int main() {
    int n, i, arr[100], sum;
    float avg;

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

    printf("Enter %d elements:\n", n);
    for (i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

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

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

```
    printf("Average = %.2f\n", avg);

    return 0;
}
```

Input:

Enter the number of elements: 5
Enter 5 elements: 10 20 30 40 50

Output:
Sum = 150
Average = 30.00

Problem Number: 07

Problem Name: Demonstrating `ClassNotFoundException` and `EOFException` in Java.

Objectives:

- To understand the cause and handling of `ClassNotFoundException`.
- To demonstrate `EOFException` while reading from a file or stream.
- To apply `try-catch` blocks for handling exceptions in Java.

Algorithm

1. Create a Java class.
2. Use `Class.forName()` to try loading a class that does not exist to trigger `ClassNotFoundException`.

3. Create a file and read from it using `ObjectInputStream` until EOF is reached to show `EOFException`.
4. Catch and handle both exceptions separately.
5. Display custom error messages.

Code (Java):

```
import java.io.*;

public class ExceptionDemo {

    public static void main(String[] args) {

        // 1. Demonstrate ClassNotFoundException

        try {

            Class.forName("NonExistentClass"); // This class does not exist

        } catch (ClassNotFoundException e) {

            System.out.println("Caught ClassNotFoundException: " + e.getMessage());

        }


        try {

            FileOutputStream fos = new FileOutputStream("data.txt");

            ObjectOutputStream oos = new ObjectOutputStream(fos);

            oos.writeInt(10);

            oos.writeInt(20);

            oos.close();

            FileInputStream fis = new FileInputStream("data.txt");

            ObjectInputStream ois = new ObjectInputStream(fis);

            while (true) {
```

```
            int num = ois.readInt(); // Read until EOF
            System.out.println("Read: " + num);
        }
    } catch (EOFException eof) {
        System.out.println("Caught EOFException: End of file reached.");
    } catch (IOException ioe) {
        System.out.println("Caught IOException: " + ioe.getMessage());
    }
  }
}
```

Input:

This program doesn't require user input. It creates and reads from a file internally.

Output:

Caught ClassNotFoundException: NonExistentClass

Read: 10

Read: 20

Caught EOFException: End of file reached.

Problem Number: 08

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

Objectives

- To read input from a file (`input.txt`).
- To perform arithmetic operations (addition, subtraction, multiplication, division) on integer pairs.
- To write the results to a file (`output.txt`) in a case-wise manner.

Algorithm

1. Open `input.txt` in read mode.
2. Read all integers into an array.
3. For every **two consecutive integers**, perform:
    - Addition
    - Subtraction
    - Multiplication
    - Integer division (check for division by zero)
4. Write the result in `output.txt` in the format:
   `Case-x: ADD SUB MUL DIV`
5. Close all files.

Code (C Language):

```
#include <stdio.h>

#include <stdlib.h>

int main() {

   FILE *fin, *fout;

   int numbers[100], n = 0;
```

```
int a, b, i = 0, caseNo = 1;
// Open input file
fin = fopen("input.txt", "r");
if (fin == NULL) {
    printf("Error opening input.txt\n");
    return 1;
}
// Read integers from file
while (fscanf(fin, "%d", &numbers[n]) != EOF) {
    n++;
}
fclose(fin);
// Open output file
fout = fopen("output.txt", "w");
if (fout == NULL) {
    printf("Error opening output.txt\n");
    return 1;
}
// Process pairs of numbers
for (i = 0; i < n - 1; i += 2) {
    a = numbers[i];
    b = numbers[i + 1];
    int add = a + b;
    int sub = a - b;
```

```
        int mul = a * b;
        int div = (b != 0) ? (a / b) : 0;
        fprintf(fout, "Case-%d: %d %d %d %d\n", caseNo++, add, sub, mul, div);
    }
    fclose(fout);
    printf("Results written to output.txt\n");
    return 0;
}
```

Input:

5 5 9 8

Output:

Case-1: 10 0 25 1

Case-2: 17 1 72 1

Problem Number: 09

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

Role of Software Engineering in Biomedical Engineering

1. Medical Software Development

- **Diagnostic tools** like MRI/CT scan software rely on complex algorithms developed by software engineers.

- Development of **electronic health records (EHR)** systems that store and manage patient data.

2. Medical Device Programming

- Software engineers develop embedded software for medical devices like pacemakers, insulin pumps, ECG machines, etc.

- These systems must be **real-time**, **fail-safe**, and comply with **medical regulations** (e.g., FDA standards).

3. Simulation & Modeling

- Creating simulations of biological processes (like blood flow or organ function) to assist doctors and researchers.

- Helps in **drug development**, surgery planning, and **personalized medicine**.

4. Data Analysis & Machine Learning

- Software engineering enables the integration of **AI and data science** in biomedical research.

- Example: analyzing **genomic data**, predicting disease risks, and **AI-based diagnostics**.

5. Human-Computer Interaction (HCI)

- Designing **user-friendly interfaces** for medical staff and patients (e.g., telemedicine apps, diagnostic tools).

- Ensures usability, accessibility, and patient safety.

Role of Software Engineering in Artificial Intelligence & Robotics

1. Algorithm Development

- Software engineers implement **AI algorithms** like machine learning, deep learning, and natural language processing.
- Applications: image recognition, autonomous decision-making, voice assistants, etc.

2. Robot Control Systems

- Development of control software that manages robotic actions (e.g., path planning, object detection, motor control).
- Involves **real-time programming**, sensor fusion, and feedback systems.

3. Simulation and Testing

- Engineers create **virtual environments** to simulate AI behaviors and robot operations before real-world deployment.
- Ensures **safe and optimized performance**.

4. Integration with Hardware

- Software engineering bridges the gap between AI models and robotic hardware (motors, sensors, actuators).
- Example: programming microcontrollers or using ROS (Robot Operating System).

5. Ethics and Safety

- Engineers must build **ethical AI systems** that are safe, transparent, and explainable.
- Critical in autonomous vehicles, surgical robots, and military drones.

Summary

Field	Contribution of Software Engineering
Biomedical Engineering	Development of medical software, simulation tools, device programming, and data analysis systems.
AI & Robotics	Design of intelligent systems, robot control, real-time processing, simulation, and safe human interaction.

Problem Number: 10

Problem Name: Study the various phases of the waterfall model. Which phase is the most dominant one?

Study of the Waterfall Model Phases

The **Waterfall Model** is one of the earliest and simplest models of the software development life cycle (SDLC). It is **sequential**, meaning each phase must be completed before the next one begins.

📑 Phases of the Waterfall Model

1. ### 1. Requirements Gathering and Analysis

    - Stakeholders define the **system's requirements**.
    - Outputs: Software Requirement Specification (SRS) document.
    - No coding or design is done here, only documentation and understanding of user needs.

2. ### 2. System Design

    - Based on requirements, the system architecture and design components are planned.
    - Decisions on software architecture, data models, and technology stack are made.

3. **3. Implementation (Coding)**
    - The actual **coding** of the system is done based on the design.
    - Developers write the code in modules or units.
4. **4. Integration and Testing**
    - All modules are integrated and tested as a whole system.
    - Purpose: Find bugs and ensure that the system behaves as expected.
    - Includes **unit testing, integration testing, and system testing**.
5. **5. Deployment**
    - The software is installed at the user's site.
    - Can be a staged or full release.
6. **6. Maintenance**
    - Post-deployment phase to fix any issues or to update the software.
    - Can include bug fixing, performance improvements, or new features.

Most Dominated Phase: Requirements Gathering and Analysis

Reasons:

- **Foundation Phase**: All other phases depend heavily on correctly understood requirements.
- **Change is Costly Later**: Mistakes in this phase propagate through all stages and become **expensive** to fix in later stages.

- **Customer Involvement**: This is the primary phase where end-users are actively involved.
- **Scope Definition**: Determines the scope, cost, timeline, and feasibility of the entire project.

Summary Table

Phase	Description
Requirements Analysis	Define "what" the system should do
System Design	Plan "how" to implement the requirements
Implementation	Code based on design
Testing	Validate the system functionality
Deployment	Release the system to end-users
Maintenance	Post-release bug fixes and updates

Conclusion: While all phases are important, the **Requirements Gathering and Analysis** phase is considered the **most dominant** in the Waterfall model because it sets the foundation for the entire project. Any errors here can lead to project failure or major rework.

Problem Number: 11

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

Objectives

- To understand how to use the Basic COCOMO Model.
- To estimate the effort (person-months), development time, and number of people required to develop a software system in an industrial domain.
- To apply COCOMO to a realistic inventory management system.

Problem Description

Estimate the development effort for an Inventory Management System (IMS) for a warehouse using the Basic COCOMO Model. The estimated size of the project is 50 KLOC (Kilo Lines of **Code).**

COCOMO Model Background

COCOMO estimates the effort required for software development using the formula:

Effort Estimation (E):

E=a×(KLOC)b(Person-Months)E = a \times (KLOC)^b \quad \text{(Person-Months)}E=a×(KLOC)b(Person-Months)

Development Time (D):

D=c×(E)d(Months)D = c \times (E)^d \quad \text{(Months)}D=c×(E)d(Months)

Where the constants a, b, c, and d depend on the project type:

Project Type	a	b	c	d

Organic	2.4	1.05	2.5	0.38
Semi-Detachd	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

Assumption

- Project Type: Semi-Detached (due to intermediate complexity and mixed team experience)
- Estimated Size: 50 KLOC

Algorithm / Calculation

Using Semi-Detached model constants:

- a = 3.0
- b = 1.12
- c = 2.5
- d = 0.35

1. **Effort (E):**

$E=3.0\times((50)^{1.12} \approx 3.0\times87.1 = 261.3$ Person-Months

2. **Development Time (D):**

$D = 2.5 \times (261.3)^{0.35} \approx 2.5 \times 7.6 = 19.0$ Months

3. **Team Size:**

$People = D/E = 261.3/19.0 \approx 13.7 \Rightarrow 14 People$

Output

- **Effort**: ~261.3 Person-Months
- **Development Time**: ~19 Months
- **Team Size**: ~14 Developers

Conclusion

Using the Basic COCOMO Model, we estimated the following for a 50 KLOC industrial Inventory Management System:

- Total effort: ~261 person-months
- Development time: ~19 months
- Required team size: ~14 people

This demonstrates the practicality of using COCOMO for early-phase cost estimation in real-world software engineering projects.

Problem Number: 12

Problem Name: Identify the reasons behind the 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 (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. The customer confirmed with 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.

Software Crisis: Reasons & Solutions

The term **software crisis** refers to the set of problems faced during software development, especially in the early decades (1970s–90s), such as:

- Projects running over time and budget
- Software being unreliable or difficult to maintain
- Lack of quality control
- Difficulty in managing large and complex software system**s**

Common Reasons Behind Software Crisis

1. Inadequate Requirements Analysis
2. Poor Software Design
3. Inadequate Testing and Debugging
4. Lack of Maintenance Planning
5. Unclear Documentation
6. Overconfidence in Technology
7. Lack of Skilled Developers
8. Failure to Predict Edge Cases

Case-wise Analysis and Solutions

Case 1 Analysis: Air Ticket Reservation System Crashed at Noon

Problem Summary:

- System delivered and installed at 12:00 AM
- Crashed exactly at 12:00 PM (noon)
- Took 5 hours to fix

Possible Causes:

- Time Format Misinterpretation: The System may have used a 12-hour format without distinguishing AM/PM.
- Faulty Time Check Logic: Could be an unhandled edge case at the 12-hour boundary.
- Insufficient Testing: Testing may have only been done from 9 AM to 11 AM; boundary testing at 12 PM was missed.

Solutions:

- Thorough Testing with Boundary Values (e.g., 11:59 AM → 12:00 PM)
- Implement proper time format validation (24-hour or include AM/PM markers)
- Unit Testing and Regression Testing at critical time transition points
- Use of automated test suites to simulate long-run behavior

Case 2 Analysis: Financial Software Malfunction, Hard to Locate Defect

Problem Summary:

- Software delivered, but it had a malfunction
- Due to a large and complex codebase, defects couldn't be easily found

Possible Causes:

- Poor Code Organization and documentation

- Lack of modularity and traceability
- No logging or debugging aids built in
- Inadequate testing during development
- Possibly, tight deadlines led to reduced code review

Solutions:

- Modular Design: Divide large systems into smaller, manageable modules
- Use of Version Control and automated debugging tools
- Built-in logging mechanisms to trace software behavior
- Apply Static Code Analysis Tools to find potential bugs
- Maintain detailed and updated documentation
- Encourage peer code reviews and unit testing

General Preventive Measures (For Both Cases)

Area	Preventive Solution
Design	Use of software engineering principles, such as modularity, abstraction, and design patterns
Testing	Implement unit, integration, system, and acceptance testing rigorously

Documentation	Maintain clear and up-to-date documentation for future debugging
Tools	Use of debuggers, profilers, loggers, and monitoring tools
Training	Continuous training of developers on best practices and new technologies
Risk Management	Anticipate edge cases and define fallback mechanisms

---

Conclusion

The software crisis is mainly due to a lack of planning, testing, and scalability practices. In both cases provided:

- Case 1 highlights the need for better boundary testing and time handling.
- Case 2 emphasizes the importance of modular design, debugging support, and maintainability.