Problem No : 01

Problem Name : Write a program in “Python” 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.

Algorithm:

Start

Input: Read an expression (`num1 operator num2=`).

Parse: Extract `num1`, `operator`, `num2` from the expression.

Evaluate: Based on the operator:

Perform addition, subtraction, multiplication, division, or modulo.

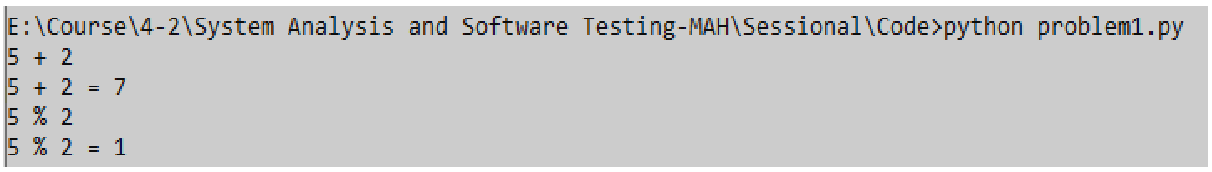
Output: Display the result (`num1 operator num2 = result`).

End.

Source Code:

```
def warning():
    print("Please Enter in this format (number1 operator number2) ")

def calculation(num1,op,num2):
    if op=="+":
        return num1+num2
    elif op=="-":
        return num1-num2
    elif op=="*":
        return num2*num1
    elif op=="/":
        return num1/num2
    elif op=="%":
        return num1%num2
    else:
        print("Please Enter Valid Operator(+,-,/,*,%)")


while True:
    try:
        num1,op,num2=input().split(" ")
        num1,num2=int(num1),int(num2)
        print(f"{num1} {op} {num2} = {calculation(num1,op,num2)}")
    except:
        print()
        warning()
        continue;
```

Output:



```
E:\Course\4-2\System Analysis and Software Testing-MAH\Sessional\Code>python problem1.py
5 + 2
5 + 2 = 7
5 % 2
5 % 2 = 1
```

Problem No : 02

Problem Name : Write a program in “Python” 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.

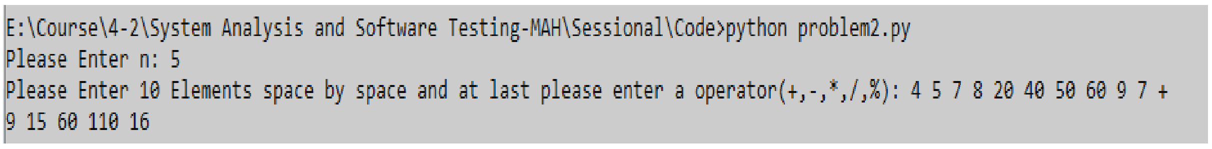
Algorithm:

```
Start
Input: Read two sets of n integers followed by an operator.
Parse: Separate the integers into two lists and extract the operator.
Evaluate: Apply the operator (+, -, *, /) element-wise between the two lists.
Output: Display the result for each pair of integers.
End.
```

Source Code:

```python
def calculation(num1,op,num2):
    if op=="+":
        return num1+num2
    elif op=="-":
        return num1-num2
    elif op=="*":
        return num2*num1
    elif op=="/":
        return num1/num2
    elif op=="%":
        return num1%num2
    else:
        print("Please Enter Valid Operator(+,-,/,*,%)")


n=int(input("Please Enter n: "))
elements=input(f"Please Enter {n*2} Elements space by space and at last please enter a operator(+,-,*,/,%): ").split(" ")
op=elements[-1]
elements=list(map(int,elements[:-1]))
for i in range(0,2*n,2):
    cal=calculation(elements[i],op,elements[i+1])
    print(cal,end=" ")
print()
```

Output:



```
E:\Course\4-2\System Analysis and Software Testing-MAH\Sessional\Code>python problem2.py
Please Enter n: 5
Please Enter 10 Elements space by space and at last please enter a operator(+,-,*,/,%): 4 5 7 8 20 40 50 60 9 7 +
9 15 60 110 16
```

Problem No : 03

Problem Name : Write a program in “Python” 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.

Algorithm:
Start
Input: Read the input (either a number or a string).
Reverse: Reverse the input.
Compare: Check if the original input is the same as the reversed input.
Output:

- If they are the same, display "Palindrome".
- If not, display "Not a palindrome".

End.

Source Code:

```
string = input("Please Enter a Number or string: ")
revString=string[::-1]
if string==revString:
   print("The Number or String is Palindrome")
else:
   print("The  Number or String is not Palindrome")
```

Output:

```
Please Enter a Number or string: I am a Student
The  Number or String is not Palindrome

Please Enter a Number or string: 2002
The Number or String is Palindrome
```

Problem No : 04

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

Solution:

1. **Card Insertion**:
   - The user inserts the ATM card into the card reader.
   - The ATM reads the card information and initiates a session.
2. **PIN Authentication**:
   - The ATM prompts the user to enter a Personal Identification Number (PIN).
   - The system validates the PIN against the bank's database.
   - If the PIN is incorrect, the user is given a few attempts before the card is blocked.
3. **Display Menu**:
   - Once authenticated, the ATM displays a menu with options:
     - Withdraw cash
     - Deposit cash
     - Check balance
     - Transfer funds
     - Change PIN
     - Print receipt
     - Exit
4. **Transaction Process**:
   - **Withdraw**:
     - The user selects an amount to withdraw.
     - The system checks if the user has sufficient funds.
     - If funds are available, the system dispenses cash and updates the balance.
   - **Deposit**:
     - The user deposits cash or checks into the ATM.
     - The system updates the balance after verifying the deposit.
   - **Balance Inquiry**:
     - The ATM displays the user's current account balance.
   - **Transfer Funds**:
     - The user transfers money between accounts (e.g., savings to checking).
   - **Change PIN**:
     - The user can update their PIN after authenticating with the old one.
5. **End Transaction**:
   - The system asks the user if they want another transaction.
   - The card is ejected after the session is complete, and the transaction receipt is printed if requested.
6. **Timeout Handling**:
   - If the user is inactive for a certain period, the session automatically ends for security purposes.

Common Bugs in the ATM System

1. **Card Reader Issues**:
   - **Bug**: Card not read properly or not detected.

   - **Impact**: Users may have to reinsert their cards multiple times, leading to frustration.
2. **PIN Validation Errors**:
   - **Bug**: Incorrect handling of invalid PIN attempts (e.g., blocking card too soon or not blocking after multiple wrong attempts).
   - **Impact**: Security risks or customer inconvenience due to premature card blocking.
3. **Screen Freeze/Unresponsiveness**:
   - **Bug**: ATM screen becomes unresponsive after selecting an option.
   - **Impact**: Users might have to restart the transaction, causing delays and confusion.
4. **Balance Update Failure**:
   - **Bug**: Balance not updated correctly after withdrawals, deposits, or transfers.
   - **Impact**: Incorrect account balances can lead to customer disputes and potential financial loss.
5. **Cash Dispenser Malfunction**:
   - **Bug**: ATM fails to dispense the requested amount of cash or dispenses incorrect amounts.
   - **Impact**: Customers may receive less money than requested, or more, leading to disputes.
6. **Deposit Malfunction**:
   - **Bug**: Deposit amount not accurately recorded or reflected in the user’s account.
   - **Impact**: Users may deposit cash or checks, but the amount may not appear in their balance, leading to frustration.
7. **Security Vulnerabilities**:
   - **Bug**: ATM does not return the card before completing a transaction or times out incorrectly.
   - **Impact**: Potential security risk where the card can be left in the ATM, or the session times out before the user completes the transaction.
8. **Receipt Printing Issues**:
   - **Bug**: Receipt contains incorrect details, or fails to print.
   - **Impact**: Users may not have proper documentation of their transaction, which could lead to confusion or disputes.
9. **ATM Crash During Transactions**:
   - **Bug**: ATM crashes or restarts in the middle of a transaction.
   - **Impact**: Loss of transaction data, leaving users unsure if their transaction was completed.
10. **Card Ejection Before Transaction Completion**:
    - **Bug**: ATM ejects the card before completing all transactions.
    - **Impact**: Potential security risk, as the session may still be active after the card is returned.

These issues highlight the importance of robust error handling, security measures, and a seamless user experience in an ATM system.

Problem No : 05

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

Algorithm:
Start
Input: Read a number `n` from the user.
Initialize: Set `factorial = 1`.
Loop:

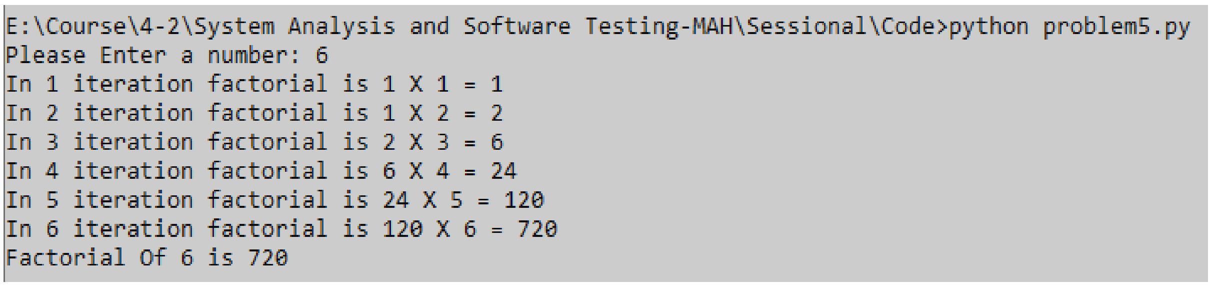
- Use a `for` or `while` loop from `i = 1` to `n`.
- Multiply `factorial` by `i` in each iteration.

Output: Display the value of `factorial`.
End.

Source Code:

```
n=int(input("Please Enter a number: "))
fact=1
for i in range(1,n+1):
   prev=fact
   fact = i*fact
   # To verify the factorial
   print(f"In {i} iteration factorial is {prev} X {i} = {fact}")
print(f"Factorial Of {n} is {fact}")
```

Output:



```
E:\Course\4-2\System Analysis and Software Testing-MAH\Sessional\Code>python problem5.py
Please Enter a number: 6
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
In 6 iteration factorial is 120 X 6 = 720
Factorial Of 6 is 720
```

Problem No : 06

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

Algorithm:

Start

Input: Read the size of the array and its elements.

Define Function 1: `calculateSum()`

- Initialize sum to 0.
- Use a `do-while` loop to iterate over the array and add elements to the sum.

Define Function 2: `calculateAverage()`

- Call `calculateSum()`.
- Compute the average by dividing sum by the number of elements.

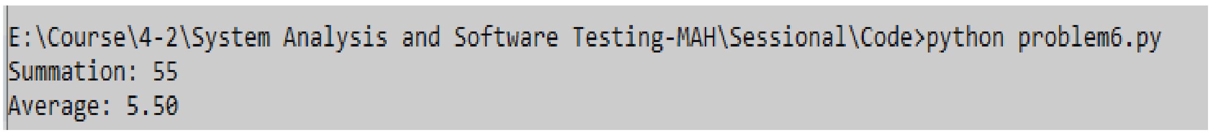
Output: Display the sum and average.

End.

Source Code:

```
def sum(array):
    n=len(array)
    summation=0
    for i in range(n):
        summation+=array[i]
    return summation


def average(array):
    n=len(array)
    summation=sum(array)
    return summation/n


array=[1,2,3,4,5,6,7,8,9,10]
print("Summation:",sum(array))
print(f"Average: {average(array):.2f}")
```

Output:



```
E:\Course\4-2\System Analysis and Software Testing-MAH\Sessional\Code>python problem6.py
Summation: 55
Average: 5.50
```

Problem No : 07

Problem Name : Write a simple “Python” program to explain classNotFound Exception and endOfFile (EOF) exception.

Algorithm:
Start
ClassNotFoundException:

- **Load Class**: Attempt to load a class using `Class.forName()`.
- **Handle Exception**: Catch `ClassNotFoundException` if the class cannot be found.
- **Output**: Display an error message.

EOFException:

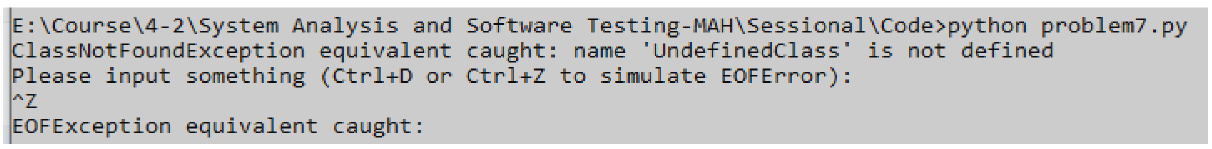
- **Read File**: Attempt to read from a file using `DataInputStream` or `BufferedReader`.
- **Handle Exception**: Catch `EOFException` if the end of the file is reached unexpectedly.
- **Output**: Display an error message.

End

Source Code:

```
# Example to simulate ClassNotFoundException and EOFError in Python
def simulate_class_not_found():
    try:
        # Trying to reference a class that is not defined
        obj = UndefinedClass()
    except NameError as e:
        print(f"ClassNotFoundException equivalent caught: {e}")

def simulate_eof_error():
    try:
        # Simulating EOFError by trying to read input with an unexpected EOF
        print("Please input something (Ctrl+D or Ctrl+Z to simulate EOFError): ")
        data = input()
    except EOFError as e:
        print(f"EOFException equivalent caught: {e}")

if __name__ == "__main__":
    # Simulate ClassNotFoundException equivalent
    simulate_class_not_found()

    # Simulate EOFException equivalent
    simulate_eof_error()
```

Output:



```
E:\Course\4-2\System Analysis and Software Testing-MAH\Sessional\Code>python problem7.py
ClassNotFoundException equivalent caught: name 'UndefinedClass' is not defined
Please input something (Ctrl+D or Ctrl+Z to simulate EOFError):
^Z
EOFException equivalent caught:
```

Problem No : 08

Problem Name : Write a program in “Python” that will read a 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.

Algorithm:
Start
Read Input:

- Open and read `input.txt` file containing `n` positive integers.

Perform Calculations:

- **Addition**: Sum all integers.
- **Subtraction**: Subtract subsequent integers from the first.
- **Multiplication**: Multiply all integers.
- **Division**: Divide the first integer by each subsequent integer.

Write Output:

- Open `output.txt` file for writing.
- Write results of each operation in a formatted way.

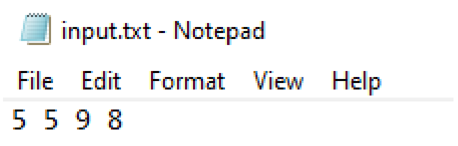
Close Files.
End

Source Code:

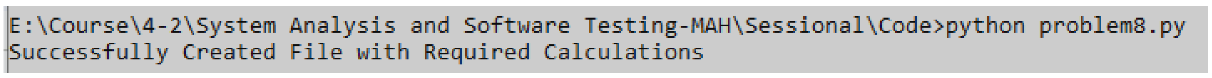
```
def calculation(num1,op,num2):
    if op=="+":
        return num1+num2
    elif op=="-":
        return num1-num2
    elif op=="*":
        return num2*num1
    elif op=="/":
        return num1/num2
    else:
        print("Please Enter Valid Operator(+,-,/,*)")


with open('input.txt', 'r') as file:
    input_data = file.read()


elements=list(map(int, input_data.replace(',', ' ').split()))
n=len(elements)
```

```
for i in range(0,n,2):
    add=calculation(elements[i],"+",elements[i+1])
    sub=calculation(elements[i],"-",elements[i+1])
    mul=calculation(elements[i],"*",elements[i+1])
    div=calculation(elements[i],"/",elements[i+1])
    result=[add,sub,mul,div]
    with open('output.txt', 'a') as file:
        file.write(f"Case: {i//2+1}: "+' '.join(map(str, result))+"\n")
print("Successfully Created File with Required Calculations")
print()
```

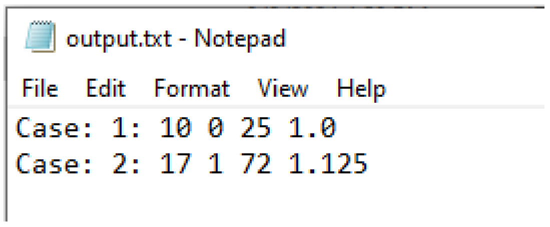
Output:



```
5 5 9 8
```



```
E:\Course\4-2\System Analysis and Software Testing-MAH\Sessional\Code>python problem8.py
Successfully Created File with Required Calculations
```



```
Case: 1: 10 0 25 1.0
Case: 2: 17 1 72 1.125
```

Problem No : 09

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

Solution:

Biomedical Engineering: Software engineering contributes to the development of medical devices, healthcare systems, data analysis tools, and simulations, ensuring that these systems are reliable and effective.
Artificial Intelligence: Software engineers develop and implement AI algorithms, integrate AI systems, create tools and frameworks, and address ethical and security issues.
Robotics: They design control systems, develop software frameworks, create simulation environments, and work on human-robot interactions to advance robotic capabilities and applications.

Role of Software Engineering in Biomedical Engineering

1. Medical Device Software:
    - Development: Software engineers design and develop software for medical devices such as pacemakers, insulin pumps, and imaging systems.
    - Testing and Validation: They ensure that the software is reliable, accurate, and complies with regulatory standards.
2. Healthcare Systems:
    - Electronic Health Records (EHR): Software engineers build and maintain EHR systems that store and manage patient data securely.
    - Clinical Decision Support Systems (CDSS): They develop systems that assist healthcare providers in making informed decisions based on data analysis and algorithms.
3. Data Analysis and Management:
    - Bioinformatics: Software engineering supports the analysis of biological data, including genomic sequences and protein structures.
    - Medical Data Analytics: Engineers create tools for processing and analyzing large datasets to improve diagnostics and treatment plans.
4. Simulation and Modeling:
    - Disease Modeling: Software engineers develop simulations to model disease progression and treatment outcomes.
    - Medical Imaging: They work on algorithms for enhancing and interpreting medical images, such as MRI and CT scans.

Role of Software Engineering in Artificial Intelligence (AI)

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

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

Role of Software Engineering in Robotics

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

Problem No : 10

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

Solution:
The Waterfall model is a sequential design process used in software development and project management. It consists of the following phases:

1. Requirements Analysis:
    - Description: Gather and document all the requirements for the software project. This phase involves understanding the client's needs and defining what the system should do.
    - Output: Requirements specification document.
2. System Design:
    - Description: Design the system architecture and detailed design based on the requirements. This includes designing the system components, interfaces, and data structures.
    - Output: System design document.
3. Implementation (or Coding):
    - Description: Convert the design into executable code. This phase involves writing and testing the code for individual components and integrating them into a complete system.
    - Output: Source code and compiled software.
4. Integration and Testing:
    - Description: Integrate the various components of the system and test them to ensure that they work together as expected. This phase involves both unit testing and system testing.
    - Output: Test reports and bug fixes.
5. Deployment:
    - Description: Deploy the system to the production environment where it will be used by end-users. This phase involves installation, configuration, and initial user training.
    - Output: Deployed software and user documentation.
6. Maintenance:
    - Description: Provide ongoing support and maintenance for the software after it has been deployed. This phase involves fixing bugs, updating the software, and making improvements as needed.
    - Output: Updated software and support documentation.

Most Dominated Phase

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

1. Foundation for All Subsequent Phases:
    - The success of the entire project depends on how well the requirements are gathered and documented. If the requirements are incorrect or incomplete, it affects all subsequent phases, leading to potential project failures.
2. Cost of Changes:

    - Changes made after the requirements phase are more costly and time-consuming. Once the system design and implementation have started, making changes based on incorrect or missed requirements can be expensive.
3. Client Satisfaction:
    - Properly understanding and documenting the requirements ensures that the final product meets the client's needs and expectations. This leads to higher client satisfaction and reduces the risk of project failure.
4. Risk Management:
    - Identifying and understanding requirements early helps in identifying potential risks and addressing them before they impact the development process.

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

Problem No : 11

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

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:

- **Size of the Software**: Typically measured in Lines of Code (LOC) or Function Points (FP).
- **Project Attributes**: Various cost drivers such as software complexity, team experience, and development environment.

2. Choose a COCOMO Model Type

COCOMO has several models based on the project type:

1. **Basic COCOMO**: For early, 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 COCOMO Formula: $\text{Effort} = a \times (\text{KLOC})^{b}$ }

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

Example Calculation

Assumptions:

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

Using Basic COCOMO:

- Coefficients for Semi-Detached: a=3.0, b=1.12
- **Effort**: Effort=$3.0\times(50)^{1.12}$

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

Summary

To estimate the effort for a specific problem in the industrial domain using the COCOMO model:

1. **Determine the size of the software** in KLOC.
2. **Select the appropriate COCOMO model type** based on the complexity of the project.
3. **Use the Basic COCOMO formula** to calculate an initial effort estimate, or use Intermediate/Detailed COCOMO for more refined estimates.
4. **Adjust for cost drivers** if using Intermediate or Detailed COCOMO.

In this example, the estimated effort for a Semi-Detached project with 50 KLOC is approximately 165.24 person-months.

Problem No : 12

Problem Name : Identify the reasons behind software crisis and explain the possible solutions for the following scenario: Case 1: “Air ticket reservation software was delivered to the customer and was installed in an airport 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. **Lack of Testing**:
   - **Description**: Inadequate or incomplete testing can result in undetected bugs and issues.
   - **Impact**: Software may fail under real-world conditions, leading to crashes and malfunctions.
5. **Inadequate Documentation**:
   - **Description**: Lack of proper documentation can hinder understanding and maintenance of the software.
   - **Impact**: Difficulty in troubleshooting and extending the software.

Case 1: Air Ticket Reservation Software

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

Possible Solutions:

1. **Improved Testing**:
   - **Solution**: Implement comprehensive testing including stress testing, load testing, and end-to-end testing to identify potential issues before 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. **Code Reviews and Pair Programming**:
    - **Solution**: Conduct regular code reviews and use pair programming to catch defects early in the development process.
    - **Benefit**: Improves code quality and helps in identifying issues before they become significant problems.
4. **Automated Testing**:
    - **Solution**: Implement automated testing frameworks to continuously test the software for defects.
    - **Benefit**: Ensures that changes do not introduce new issues and helps in catching defects early.
5. **Clear Documentation and Requirements**:
    - **Solution**: Maintain clear and detailed documentation of the software’s design, architecture, and requirements.
    - **Benefit**: Helps developers understand the system better and identify defects more effectively.

Summary

- **Software Crisis Reasons**: Complexity, unclear requirements, poor project management, lack of testing, and inadequate documentation.
- **Case 1 Solutions**: Improved testing, robust monitoring, contingency planning, and post-deployment support.
- **Case 2 Solutions**: Enhanced debugging tools, modular design, code reviews, automated testing, and clear documentation.

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