GATE INSTITUTE OF TECHNOLOGY & MANAGEMENT SCIENCES

(Approved by AICTE , Affiliated to Sri Venkateswara University , TIRUPATI)

ICET CODE : GTMT

Master of Computer Applications

SEMESTER - II

Hall Ticket No: 122531067

Name of the Student: M. Lahari

Subject: Software lab - II

Year: I st Semester: II Batch 2025 - 26

GATE EDUCATIONAL INSTITUTIONS

DEPARTMENT OF COMPUTERS

YEAR II SEMESTER II

Reg No. 122531067

CERTIFICATE

Certified that this is the bonafide record of practical work done in the laboratory by the candidate M. Lahari *during the academic year* 2025-27 *subject* Software lab - II .

No. of Practical's Conducted 16 ***No. of Practical's Attended*** 16

LECTURER **HEAD OF THE DEPARTMENT**

Submitted for the Practical Examination held on_____

Examiners

1.

2.

INDEX

S.No.	Date	Name of the Experiment	Page No.	Marks Awarded	Remarks
01	16/03/26	Breadth First search(BFS)	1 - 3		
02	23/03/26	Depth First search (DFS)	4 - 6		
03	30/03/26	A * Algorithm	7 - 9		
04	06/04/26	Minimax Algorithm	10 - 12		
05	09/04/26	Bayes theorem	13 - 15		
06	16/04/26	constraint satisfaction Problem (csp)	16 - 18		
07	20/04/26	Markov Decision Process	19 - 21		
08	23/04/26	Q - learning.	22 - 24		

INDEX

Breadth First Search (BFS)

1. Program to implement Breadth First Search (BFS) for graph traversal.

Aim :-

To implement Breadth First Search (BFS) for graph traversal.

Algorithm :-

1. Start from the initial node.
2. Add the node to the queue
3. Remove node from queue and visit it.
4. Add all unvisited neighbours to the queue.
5. Repeat Until the queue becomes empty.

Source code:

```
graph = {'A':['B','C'],'B':['D','E'],'C':['F'],'D':[],'E':[],'F':[]}
visited = []
queue = []
def bfs(start):
    visited.append(start)
    queue.append(start)
    while queue:
        node = queue.pop(0)
        print(node, end=" ")
        for n in graph[node]:
            if n not in visited:
                visited.append(n)
                queue.append(n)
bfs('A')
```

OUTPUT:

```
A B C D E F
```

Result :-

Thus, Breadth First Search is implemented successfully.

Depth first search (DFS)

2. Program to implement Depth first search (DFS) for graph traversal.

Aim :-

To implement Depth first search (DFS) for graph traversal.

Algorithm :-

1. Start from the initial node
2. visit the node and mark it as visited.
3. Recursively visit all unvisited neighbours.

SOURCE CODE:

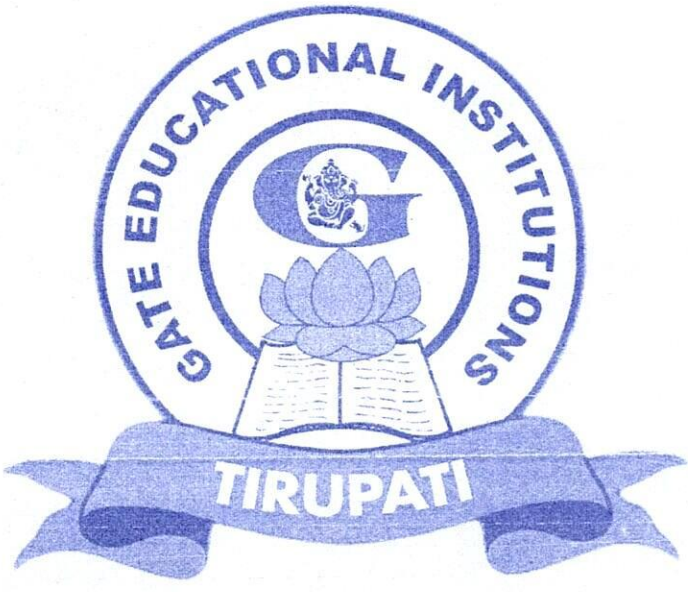
```
graph = {'A':['B','C'],'B':['D','E'],'C':['F'],'D':[],'E':[],'F':[]}
visited = set()
def dfs(node):
    if node not in visited:
        print(node, end=" ")
        visited.add(node)
        for n in graph[node]:
            dfs(n)
dfs('A')
```

OUTPUT:

```
A B D E C F
```

Result :-

Thus, Depth first search is implemented successfully



A* Algorithm

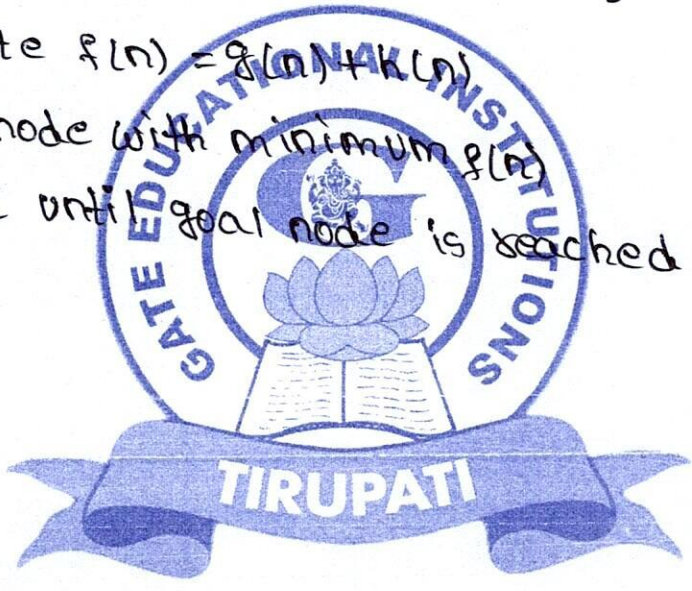
3. Program to implement A* search Algorithm for finding optimal path

Aim :-

To implement A* algorithm to find optimal path

Algorithm :-

1. Define graph and heuristic values.
2. calculate $f(n) = g(n) + h(n)$
3. Select node with minimum $f(n)$
4. Repeat until goal node is reached.



SOURCE CODE:

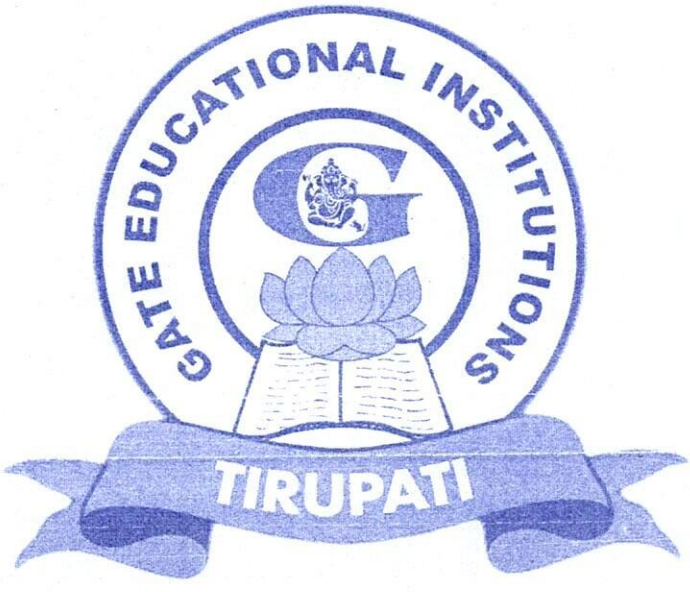
```
graph = {'A':[('B',1),('C',3)],'B':[('D',1)],'C':[('D',1)],'D':[]}
h = {'A':3,'B':1,'C':1,'D':0}
open_list = ['A']
g = {'A':0}
while open_list:
    node = min(open_list, key=lambda x: g[x] + h[x])
    print(node, end=" ")
    if node == 'D':
        break
    open_list.remove(node)
    for (n,c) in graph[node]:
      g[n] = g[node] + c
      open_list.append(n)
```

OUTPUT:

```
A B D
|
```

Result :-

Thus, A* algorithm is implemented successfully.



Minimax Algorithm

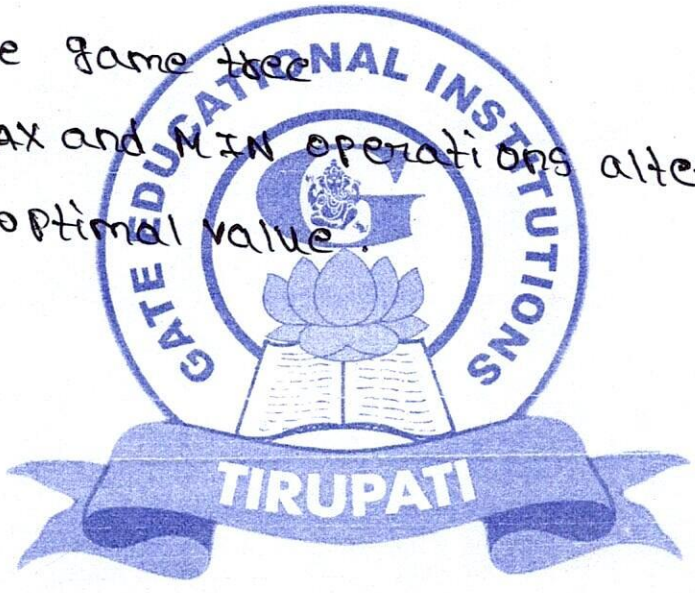
4. Program to implement Minimax Algorithm for game decision making.

Aim :-

To implement Minimax algorithm for game decision making.

Algorithm :-

1. Generate game tree
2. Apply MAX and MIN operations alternately
3. Select optimal value.



SOURCE CODE:

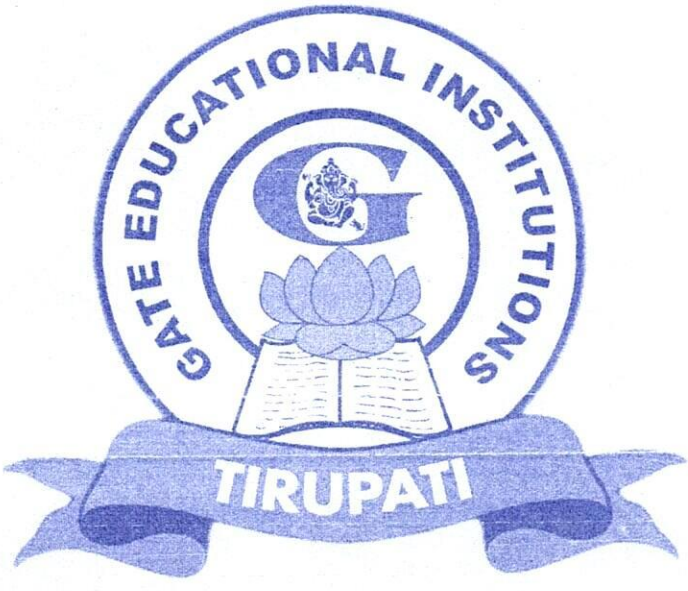
```
def minimax(depth, nodeIndex, isMax, values, height):
    if depth == height:
        return values[nodeIndex]
    if isMax:
        return max(minimax(depth+1,nodeIndex*2,False,values,height),
minimax(depth+1,nodeIndex*2+1,False,values,height))
    else:
        return min(minimax(depth+1,nodeIndex*2,True,values,height),
minimax(depth+1,nodeIndex*2+1,True,values,height))
values = [3,5,6,9,1,2,0,-1]
print(minimax(0,0,True,values,3))
```

OUTPUT:

```
----------
5
```

Result:-

Thus, Minimax algorithm is implemented successfully.



Bayes Theorem

5. Program to implement Bayes Theorem for Probability calculation.

Aim :-

To implement Bayes theorem for Probability calculation.

Algorithm :-

1. Define prior Probabilities
2. Apply Bayes formula
3. calculate Posterior Probability

SOURCE CODE:

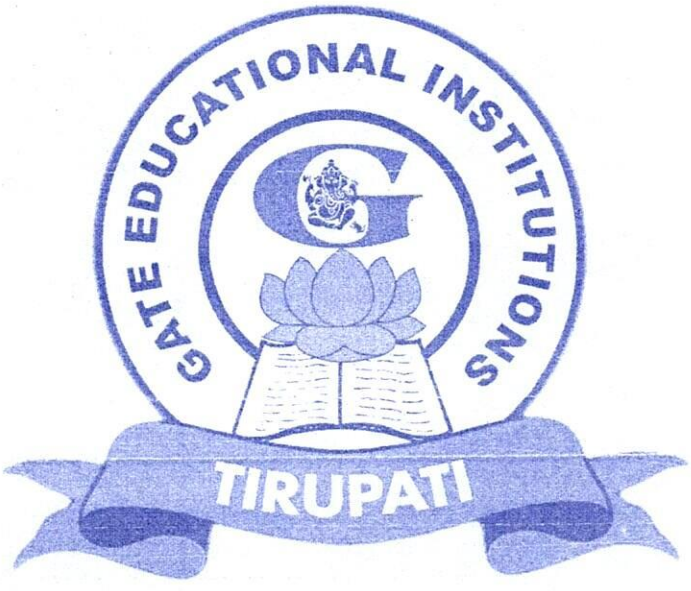
```
P_D = 0.01
P_Pos_D = 0.99
P_Pos_notD = 0.05
P_notD = 1 - P_D
P_Pos = (P_Pos_D * P_D) + (P_Pos_notD * P_notD)
P_D_Pos = (P_Pos_D * P_D) / P_Pos
print(P_D_Pos)
```

OUTPUT:

```
0.1666666666666669
```

Result:-

Thus, Bayes theorem is implemented successfully.



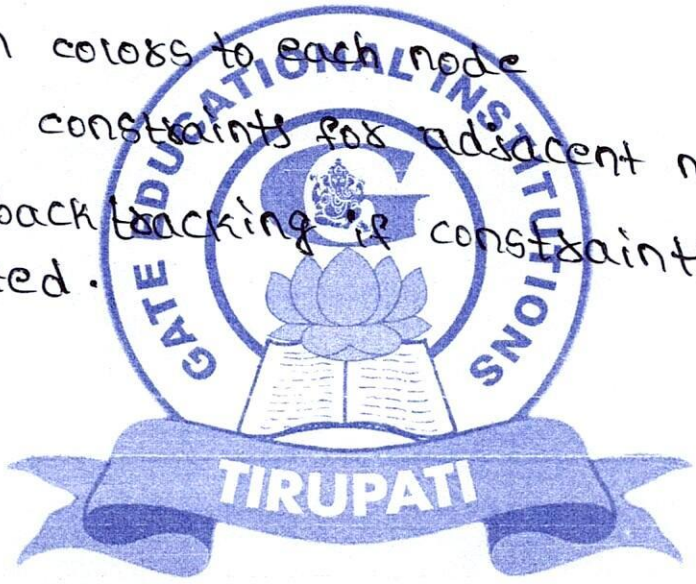
6. Program to solve Map coloring Problem using constraint satisfaction Problem (CSP)

Aim :-

To solve map coloring Problem using constraint Satisfaction Problem.

Algorithm :-

1. Assign colors to ~~each~~ node
2. check constraints for adjacent nodes
3. Use back ~~t~~tracking if constraints are violated.



SOURCECODE:

```
colors = ["Red","Green","Blue"]
graph = {'A':['B','C'],'B':['A','C'],'C':['A','B']}
solution = {}
def safe(node, color):
    for n in graph[node]:
        if n in solution and solution[n] == color:
            return False
    return True
def solve(node):
    if node == len(graph):
        return True
    nodes = list(graph.keys())
    for color in colors:
        if  safe(nodes[node], color):
            solution[nodes[node]] = color
            if solve(node+1):
                return True
            del solution[nodes[node]]
            return False
solve(0)
print(solution)
```

OUTPUT:

```
{'A': 'Red', 'B': 'Green', 'C': 'Blue'}
```

Result :-

Thus, map coloring problem is solved successfully

Markov Decision Process (MDP)

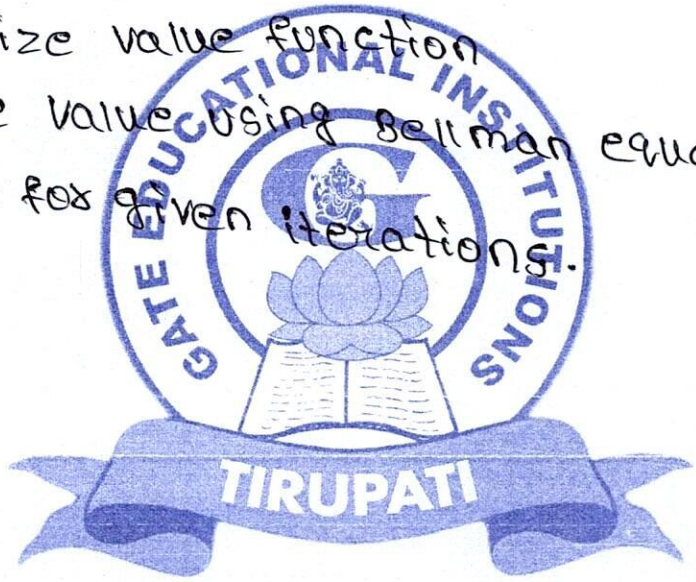
7 Program to implement value iteration using Markov Decision Process (MDP)

Aim :-

To implement value iteration using Markov Decision Process

Algorithm :-

1. Initialize value function
2. Update value using Bellman equation.
3. Repeat for given iterations.



SOURCE CODE:

```
states = [0,1]
rewards = {0:5,1:10}
gamma = 0.9
V = [0,0]
for i in range(5):
    newV = V.copy()
    for s in states:
        newV[s] = rewards[s] + gamma * V[s]
    V = newV
print(V)
```

OUTPUT:

```
[20.4755, 40.951]
```

Result :-

Thus, value iteration is implemented successfu-lly.

Q-learning

8. Program to implement Q-learning in Reinforcement learning.

Aim :-

To implement Q-learning in Reinforcement learning.

Algorithm :-

1. Initialize Q-table
2. Select random state and action
3. Update Q values using formula
4. Repeat for iterations.

SOURCE CODE:

```
import numpy as np
Q = np.zeros((2,2))
gamma = 0.8
alpha = 0.1
rewards = [[0,1],[1,0]]
for i in range(100):
    s = np.random.randint(0,2)
    a = np.random.randint(0,2)
    Q[s,a] = Q[s,a] + alpha * ( rewards[s][a] + gamma * np.max(Q[a]) - Q[s,a] )
print(Q)
```

OUTPUT:

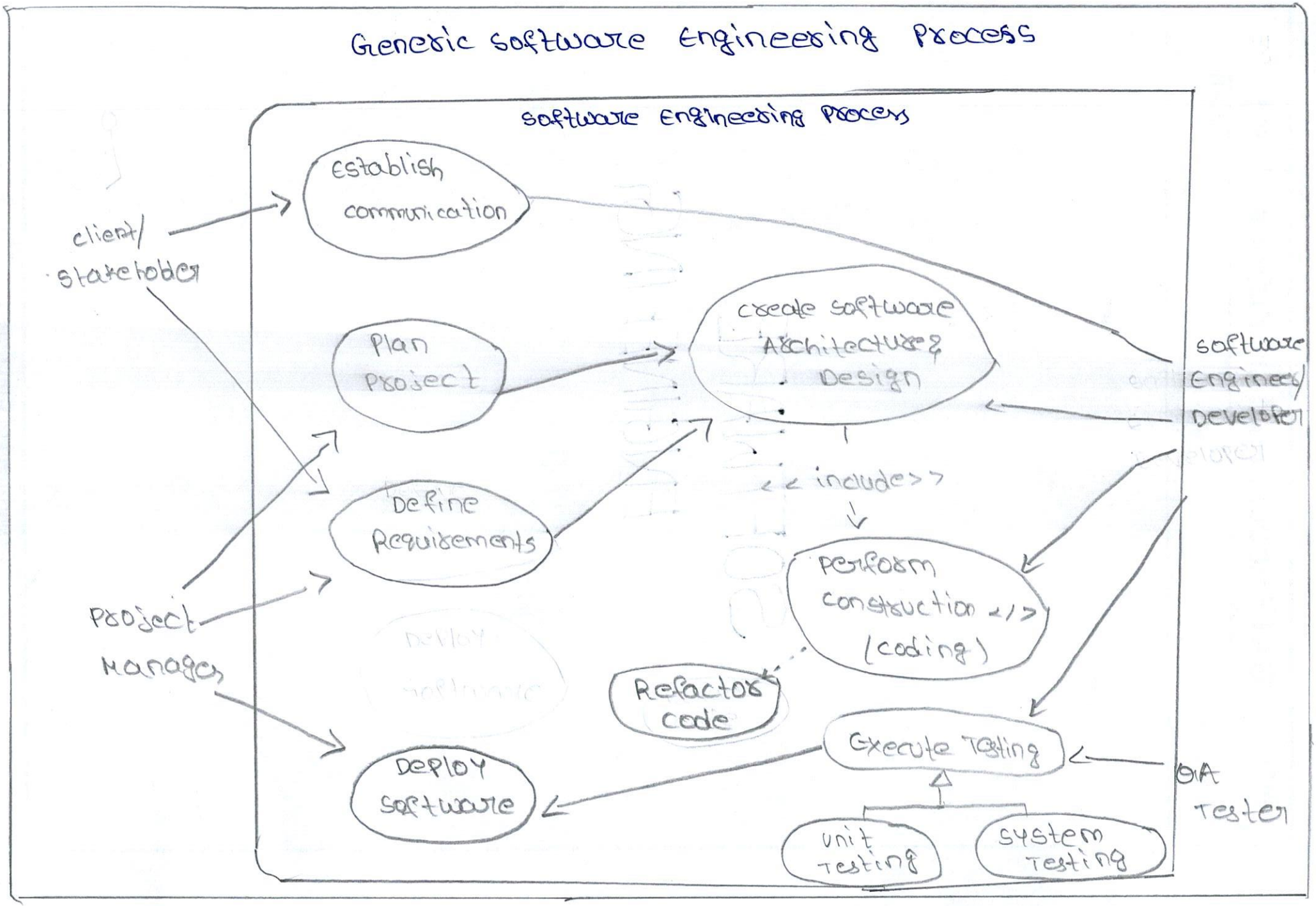
```
Output
[[1.14511186 2.14806507]
 [2.14206052 1.06274592]]

=== Code Execution Successful ===
```

Result :-

Thus, Q-learning is implemented successfully.

SOFTWARE ENGINEERING

Generic Software Engineering Process



Use case diagram :-

1: Explain the concept of software engineering. Describe its importance and discuss the generic view of the software process in detail.

Aim :-

To study and understand the concept of software engineering, its importance, and the generic view of the software process, and to demonstrate a simple implementation using Python.

Description :-

Software engineering is the systematic, disciplined, and quantifiable approach to the development, operation, and maintenance of software. It applies engineering principles to build reliable, efficient, and scalable software systems.

Objectives :-

* To understand the principles of software engineering
* To learn the importance of structured software development
* To study the phases of the software process
* To implement a simple example representing software process steps using Python

1. # Simple representation of Software Process

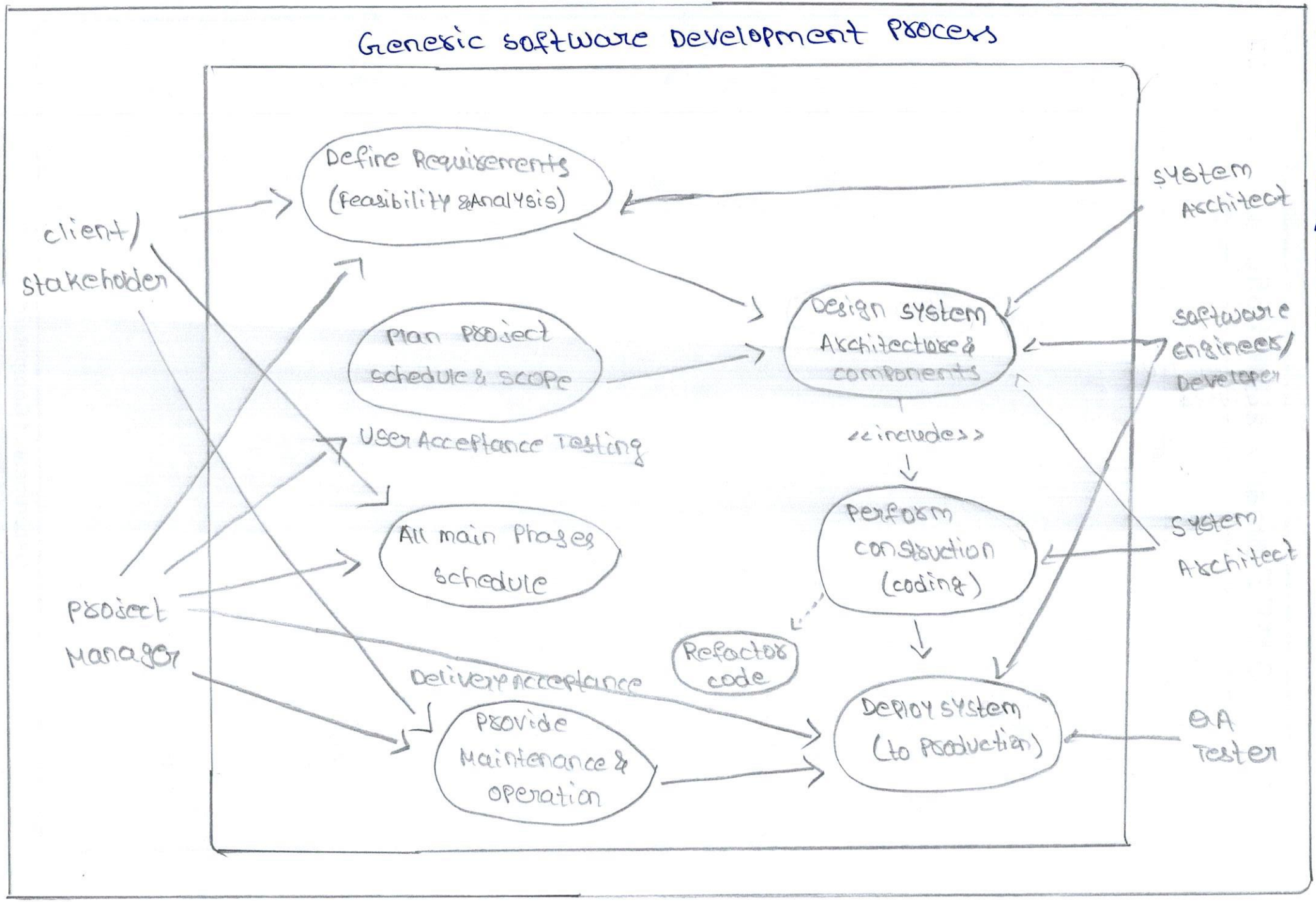
```
def requirements():
    return "Requirements gathered successfully."

def design():
    return "System design completed."

def implementation():
    return "Coding phase completed."

def testing():
    return "Testing done successfully."

def deployment():
    return "Software deployed."

def maintenance():
    return "Maintenance activity ongoing."

# Main process
print("Software Process Execution:\n")

print(requirements())
print(design())
print(implementation())
print(testing())
print(deployment())
print(maintenance())
```

Output

```
Software Process Execution:

Requirements gathered successfully.
System design completed.
Coding phase completed.
Testing done successfully.
Software deployed.
```

Use case diagram :-

Generic software Development Process



2. Simulate various software process models and compare Traditional and Agile process models using Python.

Aim :-

To simulate various software process models and compare traditional and Agile process models using Python.

Description :-

Software process models define structured approaches used to develop software systems. These models guide planning, execution, testing and delivery.

objectives :-

* understand key software development models.
* Identify differences between traditional and Agile approaches.
* Analyze advantages and limitations of each model
* Apply decision logic for selecting an appropriate model.

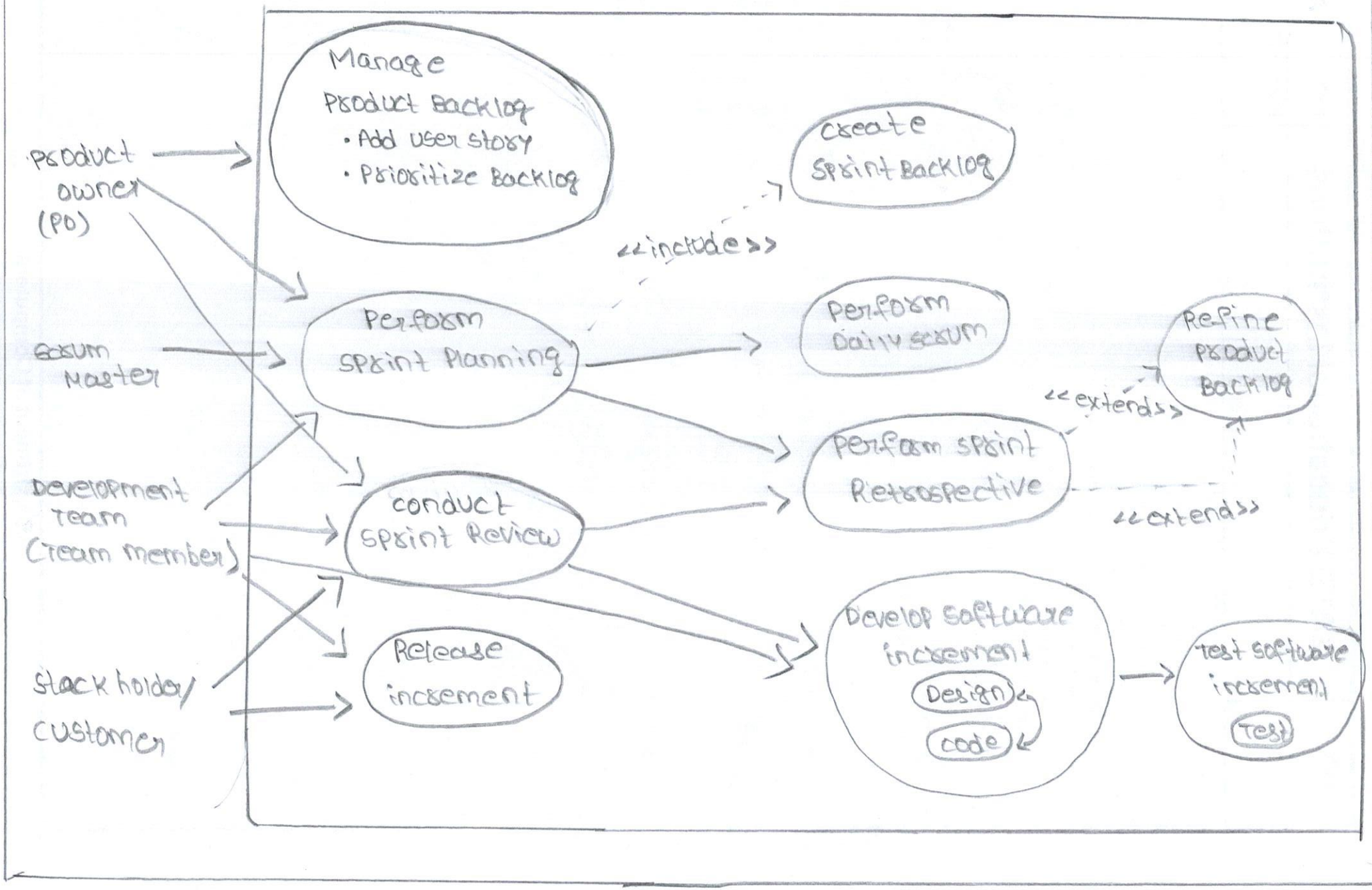
2. # Software Process Model Selector

```
def select_model(requirements_clear, project_size, need_flexibility):
    if requirements_clear and project_size == "large":
        return "Waterfall Model"
    elif not requirements_clear and need_flexibility:
        return "Agile Model"
    elif project_size == "medium":
        return "Incremental Model"
    else:
        return "Spiral Model"

# Example inputs
requirements_clear = False
project_size = "small"
need_flexibility = True

model = select_model(requirements_clear, project_size, need_flexibility)

print("Recommended Software Process Model:", model)
```

Output

```
Recommended Software Process Model: Agile Model
```

Agile (scrum) Process Model



use case diagram :-

3 Explain Agile software development and compare it with conventional approaches. write a Python program to simulate both models.

Aim :-

To study Agile software development and compare it with conventional approaches, and to implement a Python program to simulate both models.

Description :-

The Agile approach is a modern software development methodology that focuses on iterative development, continuous feedback, and adaptability to change. Instead of completing all phases sequentially like traditional models, Agile breaks the project into small units called iterations or sprints, where each cycle produces a working version of the software.

Agile emphasizes collaboration between developers and customers, frequent delivery of software, and continuous improvement through feedback.

objectives:-

* To understand Agile software development concepts.
* To study Agile principles and practices.
* To compare Agile with conventional development
* To simulate iterative vs sequential development using Python.

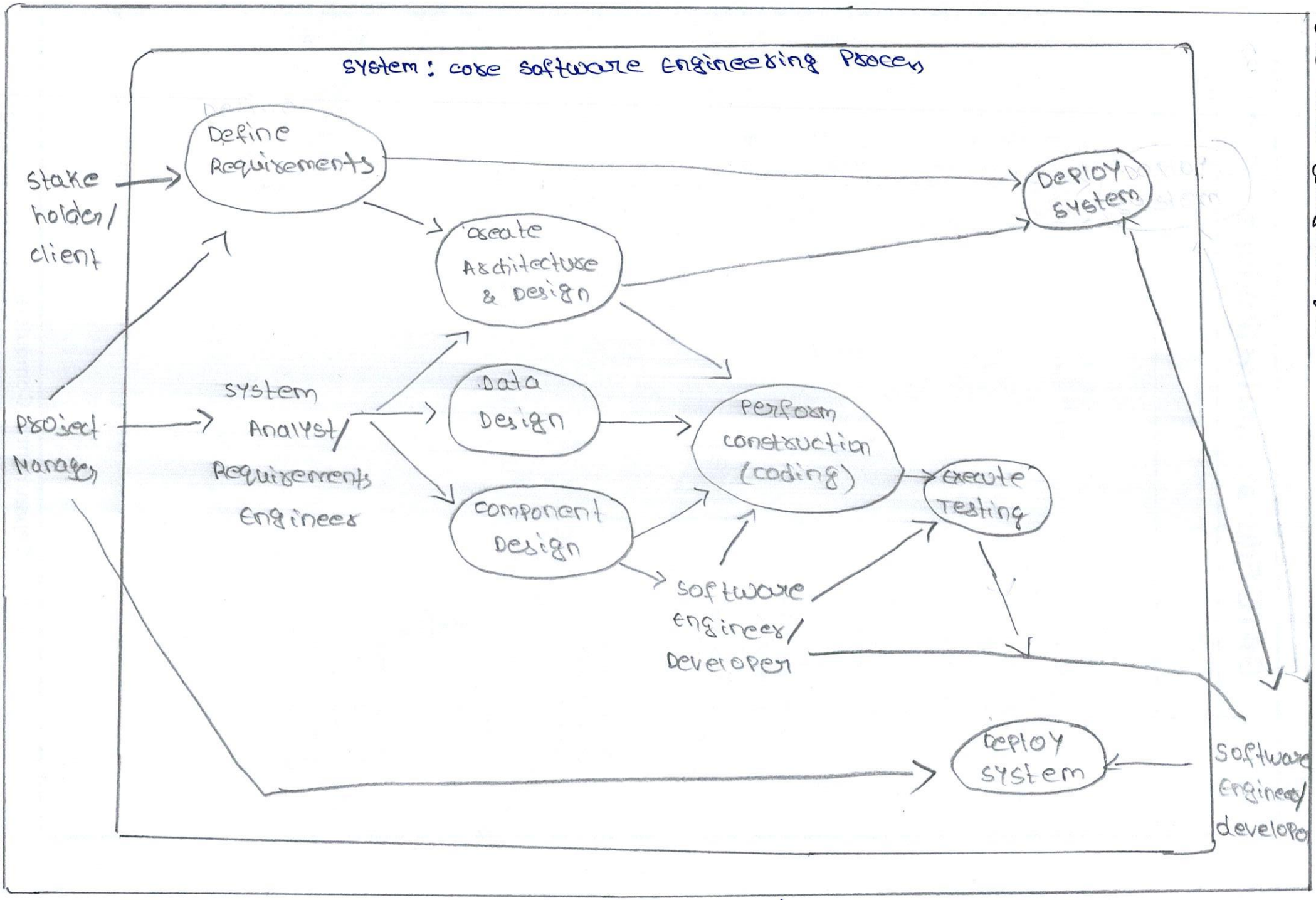
3.# Agile vs Conventional Software Development Simulation

```
def waterfall_process():
    print("WATERFALL MODEL")
    stages = ["Requirement Analysis", "Design", "Implementation", "Testing", "Deployment"]

    for stage in stages:
        print(f"Completed: {stage}")

    print("Final Product Delivered\n")


def agile_process(sprints):
    print("AGILE MODEL")

    for i in range(1, sprints + 1):
        print(f"--- Sprint {i} ---")
        print("Planning")
        print("Design")
        print("Development")
        print("Testing")
        print("Deliver Working Increment\n")


# Run both models
waterfall_process()
agile_process(3)
```

Output

```
WATERFALL MODEL
Completed: Requirement Analysis
Completed: Design
Completed: Implementation
Completed: Testing
Completed: Deployment
Final Product Delivered
```

Use case diagram :-



core Process View

Describe the software Engineering Practices Framework Explain in detail the following activities.

Aim:-

To study the software practices framework and understand its key activities: communication, planning, modeling, construction, and deployment, along with a simple python simulation of these phases.

Description :-

The software Engineering practices framework is a structured approach that defines the essential activities required to develop high quality software systems. It provides a foundation for all software process models (Agile, waterfall, spiral etc) and ensures that development is systematic, efficient and reliable.

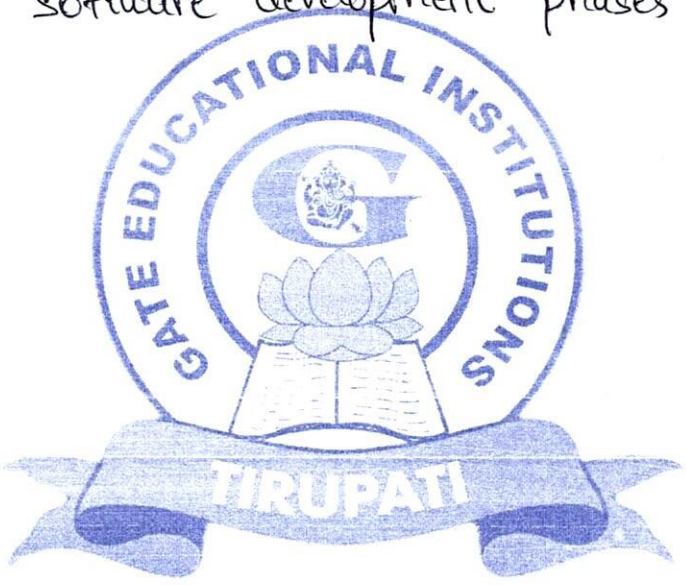
The framework is divided into five main activities:

1. communication
2. planning
3. Modeling
4. construction
5. Deployment

Each activity plays a critical role in transforming User requirement into a working software product.

Objectives :-

* To understand the phases of software engineering practices.
* To learn how software is developed step-by-step
* To analyze the important of each activity in software development.
* To simulate software development phases using python.



4# Software Engineering Practices Framework Simulation

```
def communication():
    print("1. COMMUNICATION PHASE")
    print("Requirements gathered from client.")
    print("SRS document prepared.\n")


def planning():
    print("2. PLANNING PHASE")
    print("Project schedule created.")
    print("Resources and cost estimated.\n")


def modeling():
    print("3. MODELING PHASE")
    print("System design created using UML diagrams.\n")


def construction():
    print("4. CONSTRUCTION PHASE")
    print("Coding completed.")
    print("Unit and integration testing done.\n")


def deployment():
    print("5. DEPLOYMENT PHASE")
    print("Software installed for users.")
    print("Maintenance and support started.\n")


# Execute all phases
communication()
planning()
modeling()
construction()
deployment()
```

Output

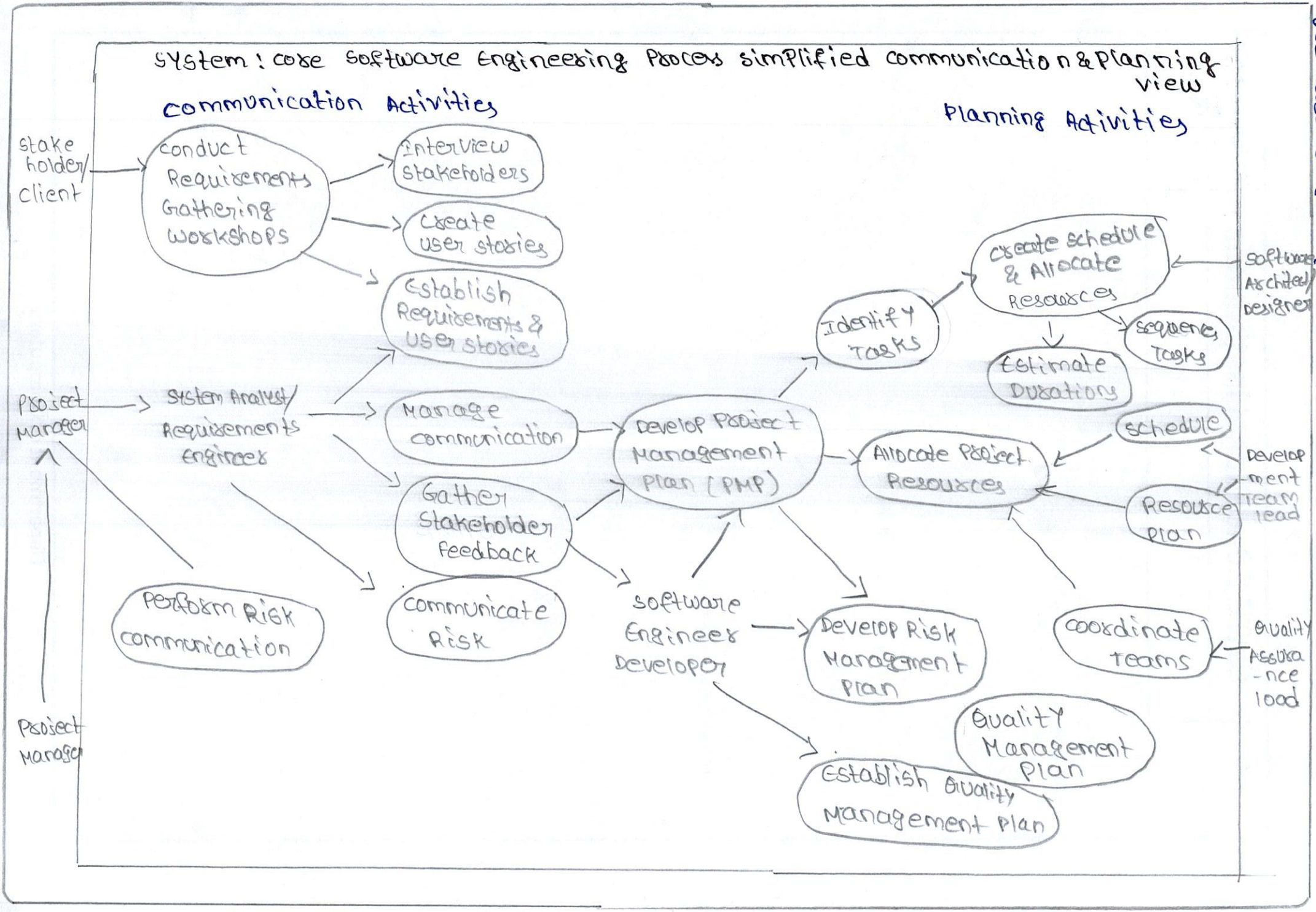
1. COMMUNICATION PHASE
Requirements gathered from client.
SRS document prepared.

2. PLANNING PHASE
Project schedule created.
Resources and cost estimated.

3. MODELING PHASE
System design created using UML diagrams.

4. CONSTRUCTION PHASE
Coding completed.
Unit and integration testing done.

5. DEPLOYMENT PHASE
Software installed for users.
Maintenance and support started.


System : core software Engineering Proces simplified communication & Planning view
communication Activities
Planning Activities
stake holder/ client
conduct Requirements Gathering workshops
Interview stakeholders
Create user stories
Establish Requirements & user stories
Project manager
System Analyst/ Requirements Engineer
Manage communication
Gather Stakeholder feedback
Perform Risk communication
communicate Risk
Develop Project Management Plan (PMP)
Identify Tasks
create schedule & Allocate Resources
Estimate Duration
sequence Tasks
Allocate Project Resources
schedule
Resource plan
software Engineer Developer
Develop Risk Management Plan
coordinate teams
Establish Quality Management Plan
Quality Management Plan
Project manager
Use case diagram :

Discuss the communication and planning activities in software engineering. Explain their importance and challenges, with suitable examples.

Aim :-

To study the communication and planning activities in software engineering, Understand their importance, challenges and demonstrate their execution How using a python program.

Description :-

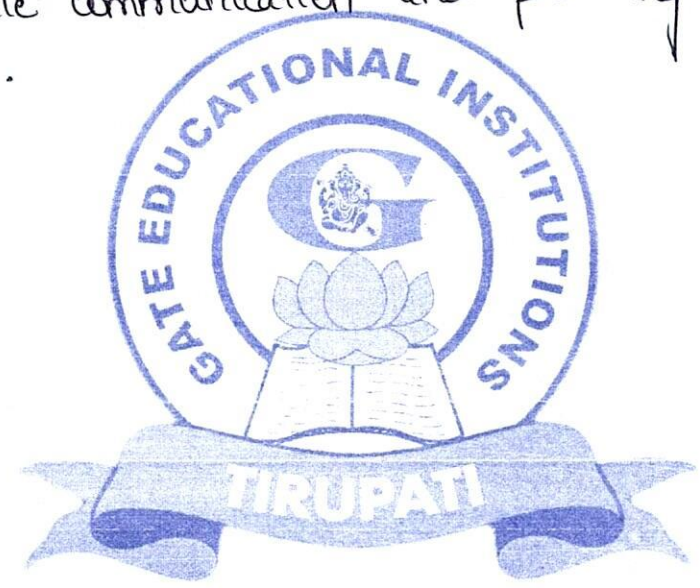
In software engineering, communication and planning are the first two major activities in the software development lifecycle.

* communication focuses on understanding what the client wants by gathering and analyzing requirements
* planning focuses on deciding how the project will be executed, including cost, schedule, resources, and risk management.

These activities ensure that the software project starts with a clear understanding and a structured roapmap.

Objectives :-

* To understand the role of communication in requirement gathering.
* To study the importance of planning in software development.
* To identify challenges in both activities.
* To simulate communication and planning phases using python.



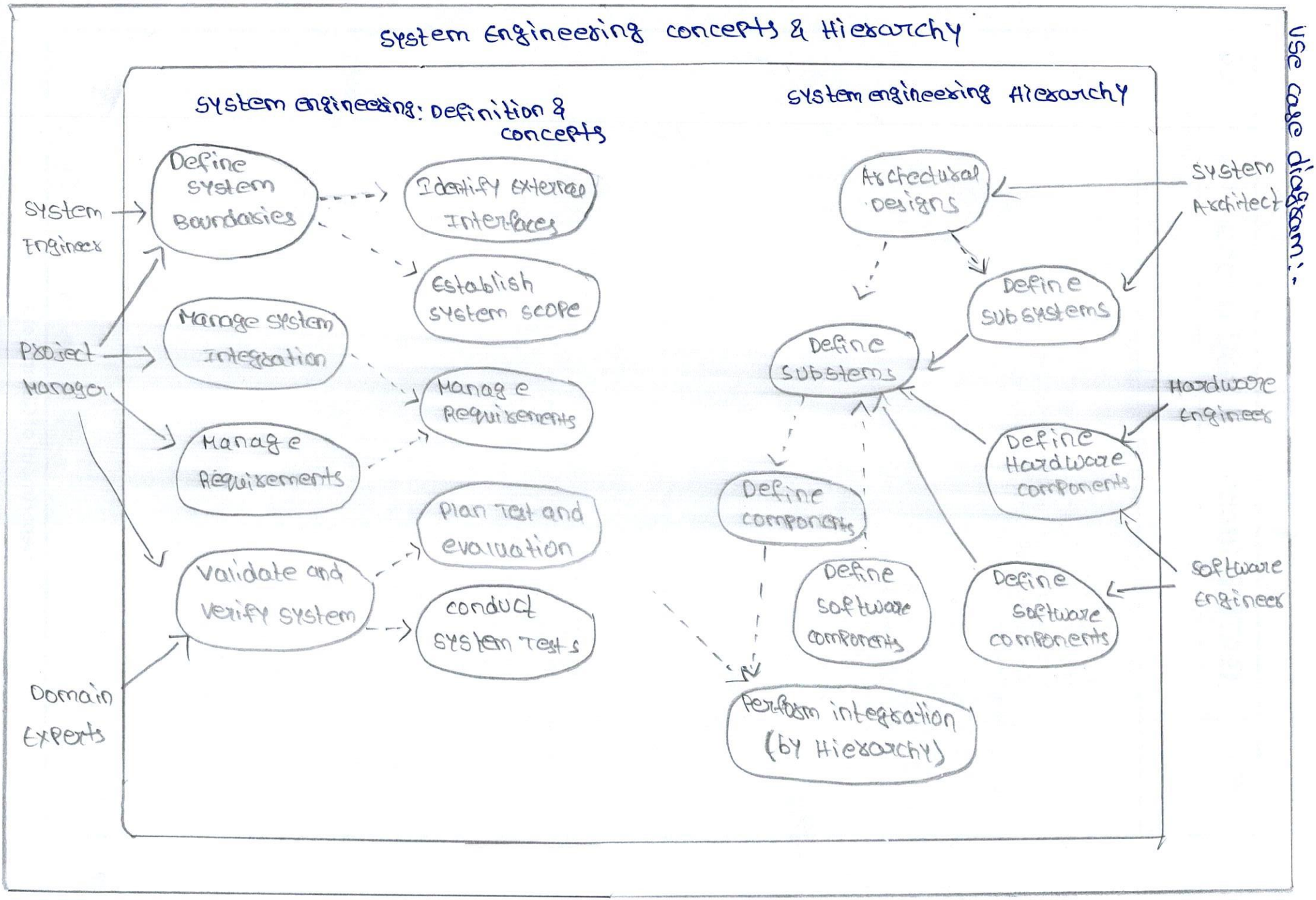
5.# Simulation of Communication and Planning in Software Engineering

```
def communication():
    print("COMMUNICATION PHASE")
    print("Client requirements gathered for Online Shopping App.")
    print("Features identified: Login, Cart, Payment, Delivery Tracking.")
    print("SRS document prepared.\n")


def planning():
    print("PLANNING PHASE")
    print("Project duration estimated: 3 months")
    print("Team assigned: 4 developers, 1 tester")
    print("Cost estimated and resources allocated.")
    print("Risk identified: Payment gateway failure\n")


# Execute phases
communication()
planning()
```

Output

```
COMMUNICATION PHASE
Client requirements gathered for Online Shopping App.
Features identified: Login, Cart, Payment, Delivery Tracking.
SRS document prepared.

PLANNING PHASE
Project duration estimated: 3 months
Team assigned: 4 developers, 1 tester
Cost estimated and resources allocated.
Risk identified: Payment gateway failure
```

System Engineering concepts & Hierarchy



Use case diagram:-

Define System Engineering. Explain the concept of computer-based systems and describe the system engineering hierarchy in detail.

Aim :—

To understand the concept of System engineering, study computer based systems, and describe the System engineering hierarchy, along with a simple python program to represent system structure.

Description :—

System Engineering is a discipline of software engineering that focus on designing, analyzing, and managing complex system over their entire life cycle. It ensures that all components of a system work together efficiently to achieve overall system goals.

A computer based system is a combination of hardware, software, data, people and procedures that interact to perform specific tasks.

System Engineering provides a top-down approach breaking a large system into smaller manageable subsystems.

Objectives :—

* To understand the concept of system engineering.
* To study computer-based system and their components.
* To learn the system engineering hierarchy.
* To learn system structure using a python program.

6.# System Engineering Hierarchy Simulation

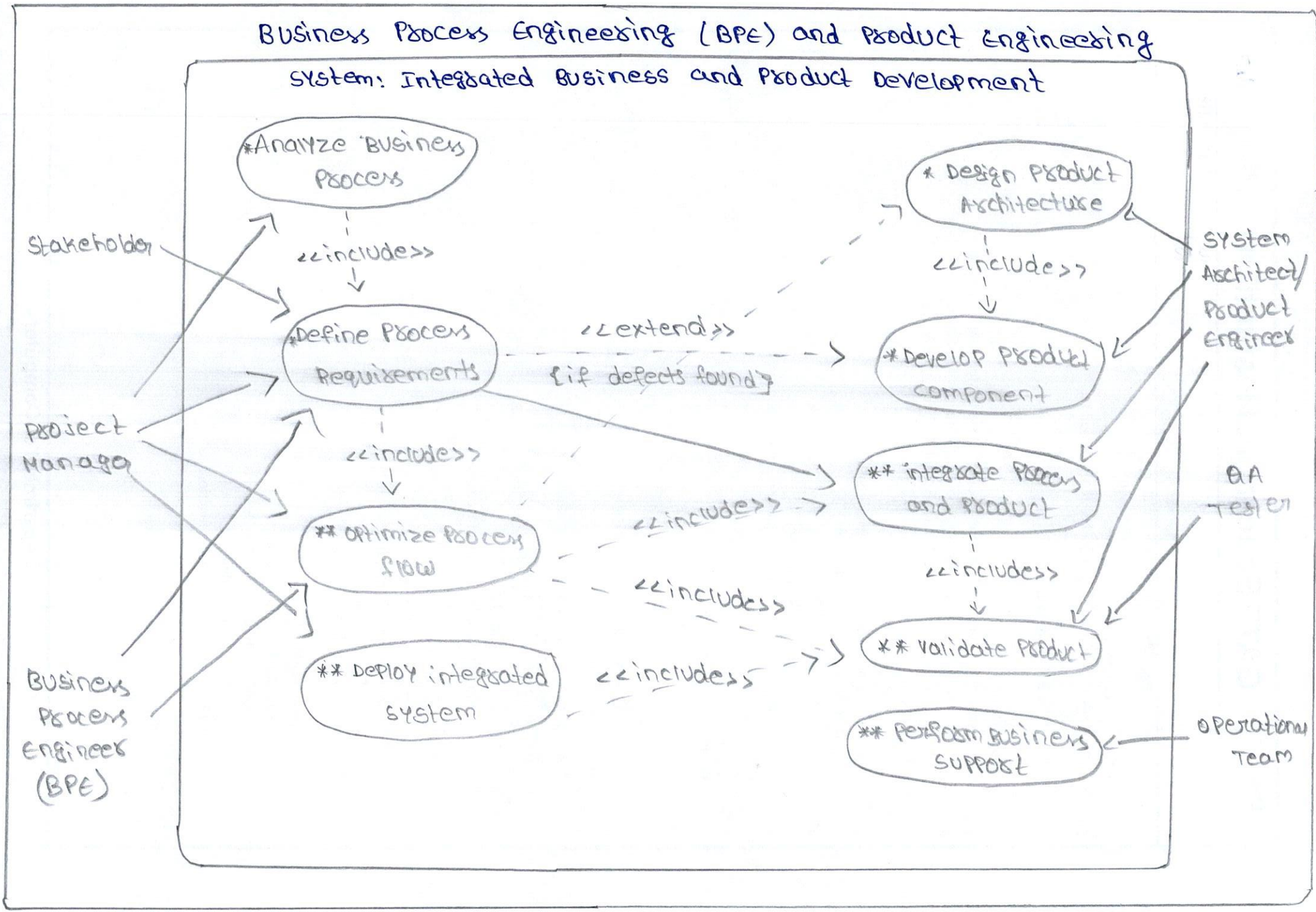
```
def system_engineering():
    print("SYSTEM ENGINEERING HIERARCHY\n")

    print("System: Banking System")

    subsystems = {
        "Account Management": ["Login Module", "Profile Module"],
        "Transaction System": ["Deposit Module", "Withdraw Module"],
        "Security System": ["Authentication Module", "Encryption Module"]
    }

    for subsystem, modules in subsystems.items():
        print(f"\n Subsystem: {subsystem}")
        for module in modules:
            print(f"   -> Module: {module}")
            print(f"      -> Component: Basic processing functions")


system_engineering()
```

Output

```
SYSTEM ENGINEERING HIERARCHY
System: Banking System

 Subsystem: Account Management
   -> Module: Login Module
      -> Component: Basic processing functions
   -> Module: Profile Module
      -> Component: Basic processing functions

 Subsystem: Transaction System
   -> Module: Deposit Module
      -> Component: Basic processing functions
   -> Module: Withdraw Module
      -> Component: Basic processing functions

 Subsystem: Security System
   -> Module: Authentication Module
      -> Component: Basic processing functions
   -> Module: Encryption Module
      -> Component: Basic processing functions
```

Use case diagram :-

Business Process Engineering (BPE) and Product Engineering



Explain Business process Engineering (BPE) and product Engineering. Discuss atter roles in system development with examples.

Aim :-

To analyze, redesign, and improve business processes to achieve higher efficiency, reduce cost, and better quality of service.

Description :-

Business procen Engineering (BPE) is a systematic approach to improving an organization's workflows. It focuses on restructuring existing business processes (or) designing completely new ones to meet busines goals using technology and management techniques. It is otten used in organization undergoing digital transformation.

Objectives :-

* Improve procen efficiency.
* Reduce operational cost
* Eliminate redundant steps
* Improve customer satisfaction.
* Automate manual processes
* Increases productivity.

7.# Business Process Engineering Simulation

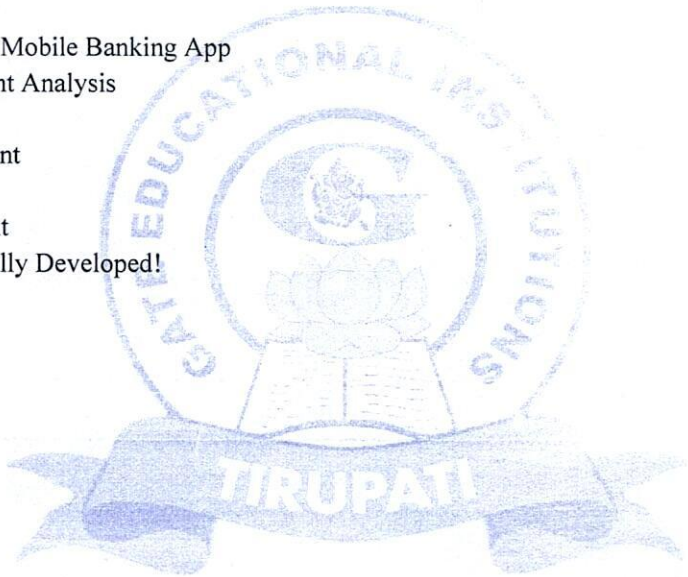
```
class BusinessProcess:
    def __init__(self, name):
        self.name = name
        self.steps = []

    def add_step(self, step):
        self.steps.append(step)

    def execute(self):
        print(f"\nExecuting Business Process: {self.name}")
        for i, step in enumerate(self.steps, start=1):
            print(f"Step {i}: {step}")
        print("Process Completed Successfully!\n")


# Product Engineering Simulation

class ProductEngineering:
    def __init__(self, product_name):
        self.product_name = product_name
        self.stages = ["Requirement Analysis", "Design", "Development", "Testing",
"Deployment"]

    def build_product(self):
        print(f"\nBuilding Product: {self.product_name}")
        for stage in self.stages:
            print(f"Stage: {stage}")
        print("Product Successfully Developed!\n")


# Example usage

# Business Process Engineering Example
order_process = BusinessProcess("Online Order Processing")
order_process.add_step("Receive Order")
order_process.add_step("Check Inventory")
order_process.add_step("Process Payment")
order_process.add_step("Dispatch Product")
order_process.execute()
```

```
# Product Engineering Example
app = ProductEngineering("Mobile Banking App")
app.build_product()
```

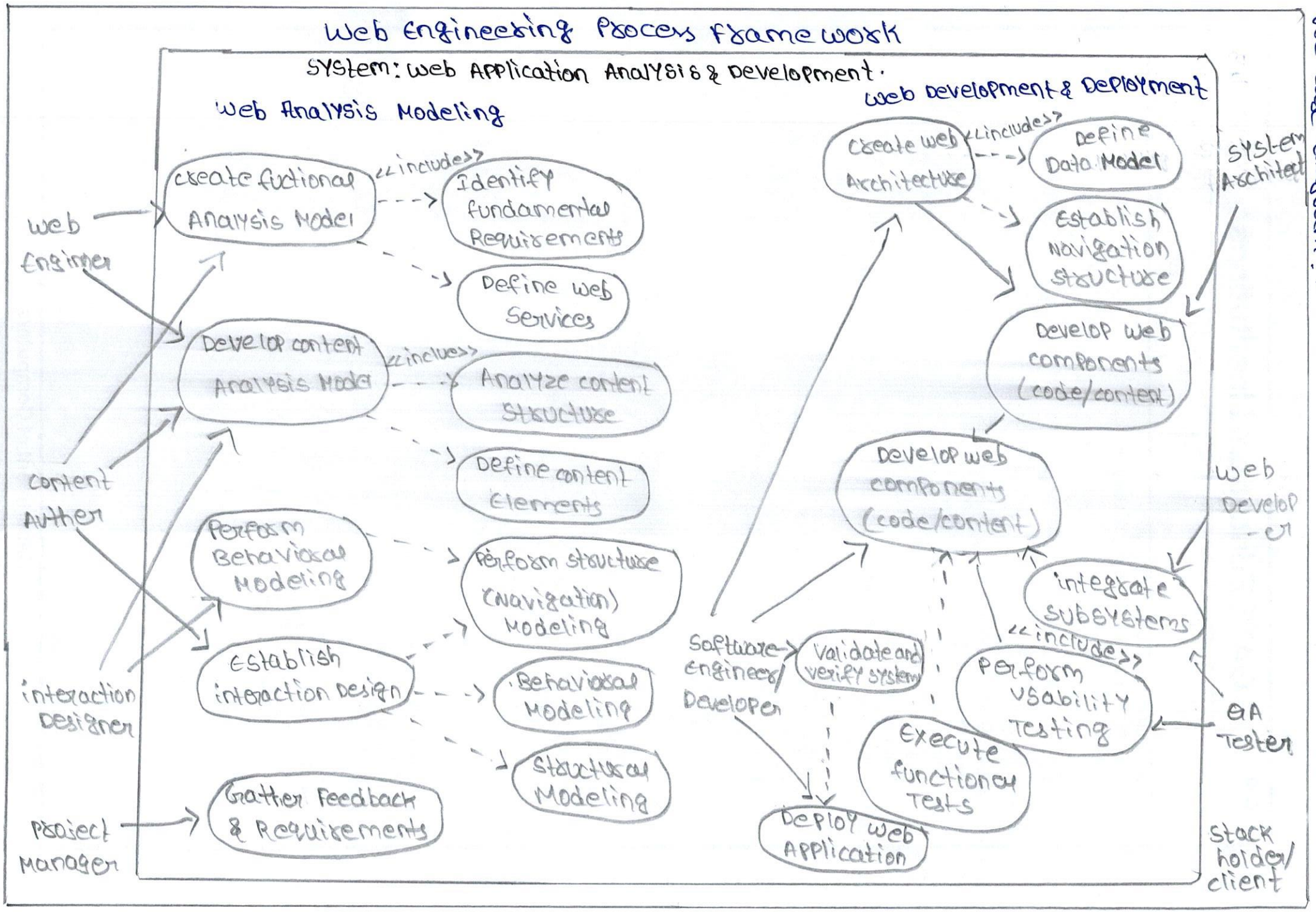
Output

```
Executing Business Process: Online Order Processing
Step 1: Receive Order
Step 2: Check Inventory
Step 3: Process Payment
Step 4: Dispatch Product
Process Completed Successfully!
```

```
Building Product: Mobile Banking App
Stage: Requirement Analysis
Stage: Design
Stage: Development
Stage: Testing
Stage: Deployment
Product Successfully Developed!
```



Web Engineering Process Framework



Use case diagram :-

Discuss the Web Engineering process. Explain different analysis models for web applications and their significance.

Aim :-

To understand the systematic process of developing web applications using web Engineering principles and to study different analysis models used for designing efficient, scalable, and user-friendly web systems.

Description :-

Web application is a disciplined approach to developing web Engineering using systematic methods, tools, and techniques similar to software engineering.

It focuses on :

* Requirements gathering
* Web application design
* content structuring
* Implementation
* Testing
* Deployment and maintance.

Unlike traditional software, web applications require attention to :

* User Experience (UI/UX)
* Navigation structure
* content management
* performance and scalability.

Objectives :-

* Develop structured and maintainable web applications.
* Improve usability and user experience
* Ensure scalability and performance.
* Manage dynamic web content efficiently.
* Reduce development time and cost
* Improve reliability and security of web systems

8. # WebEngineeringProcess

```
class WebEngineeringProcess:
    def __init__(self, project_name):
        self.project_name = project_name
        self.phases = ["Communication", "Planning", "Modeling", "Construction",
"Deployment"]

    def execute_process(self):
        print(f"\nWeb Engineering Process for: {self.project_name}")
        for phase in self.phases:
            print(f"Phase: {phase}")
        print("Web Application Development Completed Successfully!\n")


class WebAnalysisModels:
    def show_models(self):
        print("Web Application Analysis Models:\n")

        print("1. Content Model: Defines website content like text, images, videos.")
        print("2. Navigation Model: Defines user flow between pages.")
        print("3. Interaction Model: Defines user interactions like forms and clicks.")
        print("4. Architecture Model: Defines system structure (client-server).")
        print("5. Functional Model: Defines system functions like login, search.\n")


# Example execution

project = WebEngineeringProcess("Online Shopping Website")
project.execute_process()

models = WebAnalysisModels()
models.show_models()
```

Output

Web Engineering Process for: Online Shopping Website
Phase: Communication
Phase: Planning
Phase: Modeling
Phase: Construction
Phase: Deployment
Web Application Development Completed Successfully!

Web Application Analysis Models:

1. Content Model: Defines website content like text, images, videos.
2. Navigation Model: Defines user flow between pages.
3. Interaction Model: Defines user interactions like forms and clicks.
4. Architecture Model: Defines system structure (client-server).
5. Functional Model: Defines system functions like login, search.