# Thadomal Shahani Engineering College

**Bandra (W.), Mumbai- 400 050.**

## CERTIFICATE

Certify that Mr./~~Miss~~ Parth Sandeep Dabholkar of Computer Department, Semester V with Roll No. 2103032 has completed a course of the necessary experiments in the subject Software Engineering under my supervision in the **Thadomal Shahani Engineering College** Laboratory in the year 2023 - 2024

Teacher In-Charge

Head of the Department

Date 21 October, 2023

Principal

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

# CONTENTS


| SR. NO. | EXPERIMENTS | PAGE NO. | DATE | TEACHERS SIGN. |
|---|---|---|---|---|
| 1. | Write a detailed Problem Statement for any case study. Justify which process model would be suited to it. | 1-2 | 18/7/23 | |
| 2. | Application of Agile Process model on the project. | 3-5 | 25/7/23 | |
| 3. | Develop SRS document in IEEE Format. | 6-20 | 1/8/23 | |
| 4. | Develop Data Flow Diagram for project | 21-23 | 8/8/23 | |
| 5. | Develop Activity and State Diagram. | 24-30 | 8/8/23 | |
| 6. | Identify scenarios and develop Use case Diagram for the project. | 31-33 | 22/8/23 | |
| 7. | Create project schedule using Gantt Chart. | 34-36 | 29/8/23 | |
| 8. | Conduct Function Point Analysis for project. | 37-42 | 5/9/23 | |
| 9. | Application of COCOMO for cost estimation | 43-46 | 5/9/23 | |
| 10. | Develop a RMMM plan for project | 47-53 | 12/9/23 | |
| 11. | Case study: Github for Version Control | 54-64 | 26/9/23 | |
| 12. | Develop test cases using White Box Testing | 65-68 | 3/10/23 | |
| 13. | Assignment 1 | 69-74 | 1/8/23 | |
| 14. | Assignment 2 | 75-80 | 12/9/23 | |

# EXPERIMENT NO: 1

**AIM**: Write a detailed Problem Statement for any case study. Justify which process model would be best suited to apply it.

---

## Background:

The problem at hand revolves around the lack of a robust and automated system for detecting and reporting potholes in a timely and accurate manner. Manual inspection methods are inefficient and often fail to identify all potholes, leading to a delayed response in repairs and an increased risk of accidents. The absence of a reliable mechanism for road maintenance authorities to be alerted promptly about potholes hinders their ability to address the issue effectively. Additionally, the lack of accurate data on the location and extent of potholes complicates maintenance planning and resource allocation.

## Problems Faced Due to Potholes:

Safety Hazards: Potholes pose risks to drivers, cyclists, and pedestrians, leading to accidents, injuries, and fatalities.

Vehicle Damage: Potholes cause tire blowouts, misalignment, suspension damage, and costly repairs.

Traffic Congestion: Drivers slow down or swerve to avoid potholes, leading to traffic congestion and delays.

Increased Maintenance Costs: Potholes necessitate frequent road repairs, straining maintenance budgets.

Economic Impact: Vehicle damage and repairs result in financial burden for individuals and businesses.

Environmental Impact: Potholes contribute to vehicle emissions due to increased traffic congestion.

Public Dissatisfaction: Poor road conditions affect public perception of infrastructure quality.

Liability Issues: Local authorities may face legal challenges if accidents are attributed to neglected potholes.

## Problem Statement:

Design a system capable of using cameras or sensors to scan roads and identify potholes automatically. This technology will enhance road safety by promptly detecting these hazardous road conditions and alerting drivers and relevant authorities. The system's real-time notifications will help prevent accidents and enable timely road repairs, ensuring safer journeys for everyone.

## Model used and Justification:

- Pothole detection systems often require close collaboration with stakeholders to ensure the accuracy and effectiveness of detection algorithms.
- The iterative nature of Scrum allows for incremental improvements, which can be crucial for refining complex algorithms over time.
- Regular feedback through ceremonies like Sprint Reviews helps fine-tune the detection system's performance based on real-world testing.
- The adaptability of Scrum enables adjustments to changing priorities, which can be valuable if new pothole types or features need to be incorporated.

1. Iterative Development: Scrum's iterative approach accommodates evolving requirements and allows gradual system enhancement.

2. Stakeholder Engagement: Regular feedback in Scrum ensures alignment with real-world pothole detection needs.

3. Cross-Functional Teams: Pothole detection involves hardware and software; Scrum's teamwork addresses both aspects effectively.

4. Time-to-Market: Short sprints in Scrum facilitate quicker deployment, addressing pothole detection issues sooner.

5. Risk Management: Frequent reviews and retrospectives enable early identification of risks and adaptation.

6. User-Centric Focus: Scrum's iterative cycles lead to a pothole detection system closely aligned with user requirements.

7. Flexibility: Ability to adapt to changes in sensor technology or detection algorithms through Scrum's approach.

8. Transparency: Scrum ceremonies provide clear project visibility, ensuring everyone is informed about progress.

9. Continuous Improvement: Retrospectives promote ongoing enhancements for better pothole detection accuracy.

10. Collaboration: Scrum's collaborative environment helps hardware and software teams integrate effectively.

# EXPERIMENT – 2

**AIM:** Application of Agile Process Model on the project using JIRA Software

## THEORY:

The Agile process model is a software development approach that emphasizes flexibility, collaboration, customer involvement, and iterative progress. It was introduced as a response to the limitations of traditional software development methodologies, such as the Waterfall model. Agile methodologies prioritize adaptability and delivering working software in small, incremental releases. Here's a detailed explanation of the Agile process model:

**Key Principles of Agile:**

1. **Individuals and Interactions Over Processes and Tools:** Agile places a strong emphasis on effective communication and collaboration among team members. It values face-to-face interactions and encourages open dialogue.

2. **Working Software Over Comprehensive Documentation:** While documentation is important, Agile prioritizes working software. It means that the primary goal is to produce functional, usable software rather than exhaustive documentation.

3. **Customer Collaboration Over Contract Negotiation:** Agile welcomes customer involvement throughout the development process. Instead of defining all requirements upfront, Agile allows customers to provide feedback and adapt to changing needs.

4. **Responding to Change Over Following a Plan:** Agile acknowledges that change is inevitable. It encourages teams to be flexible and responsive to changing requirements, even late in the development process.

**Agile Methodologies:**

There are several Agile methodologies, including:

1. **Scrum:** In Scrum, work is organized into time-boxed iterations called "sprints." A Scrum team typically consists of a Product Owner, Scrum Master, and Development Team. The Development Team is responsible for delivering a potentially shippable product increment at the end of each sprint.

2. **Kanban:** Kanban is based on visualizing work on a Kanban board. It focuses on limiting work in progress and optimizing flow. Teams using Kanban pull work from a backlog as capacity allows.

3. **Extreme Programming (XP):** XP is known for its technical practices, such as test-driven development, continuous integration, and pair programming. It also emphasizes customer involvement and frequent releases.

4. **Lean Software Development:** Lean principles, derived from manufacturing, are applied to software development. It aims to reduce waste, increase efficiency, and deliver value to the customer.

5. **Dynamic Systems Development Method (DSDM):** DSDM is an Agile methodology that includes principles for project governance and the entire project lifecycle. It's particularly well-suited for larger, more complex projects.

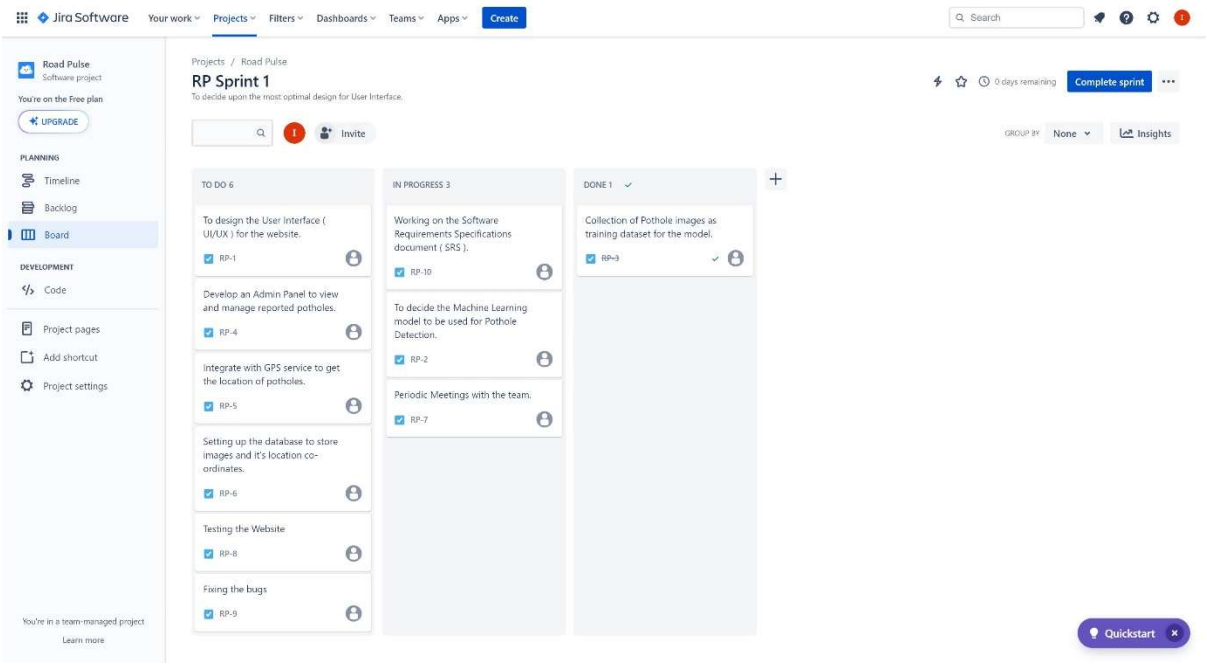
**Benefits of Agile:**

- Faster delivery of working software.
- Customer satisfaction through continuous collaboration.
- Adaptability to changing requirements.
- Improved quality through frequent testing and feedback.
- Enhanced team collaboration and morale.

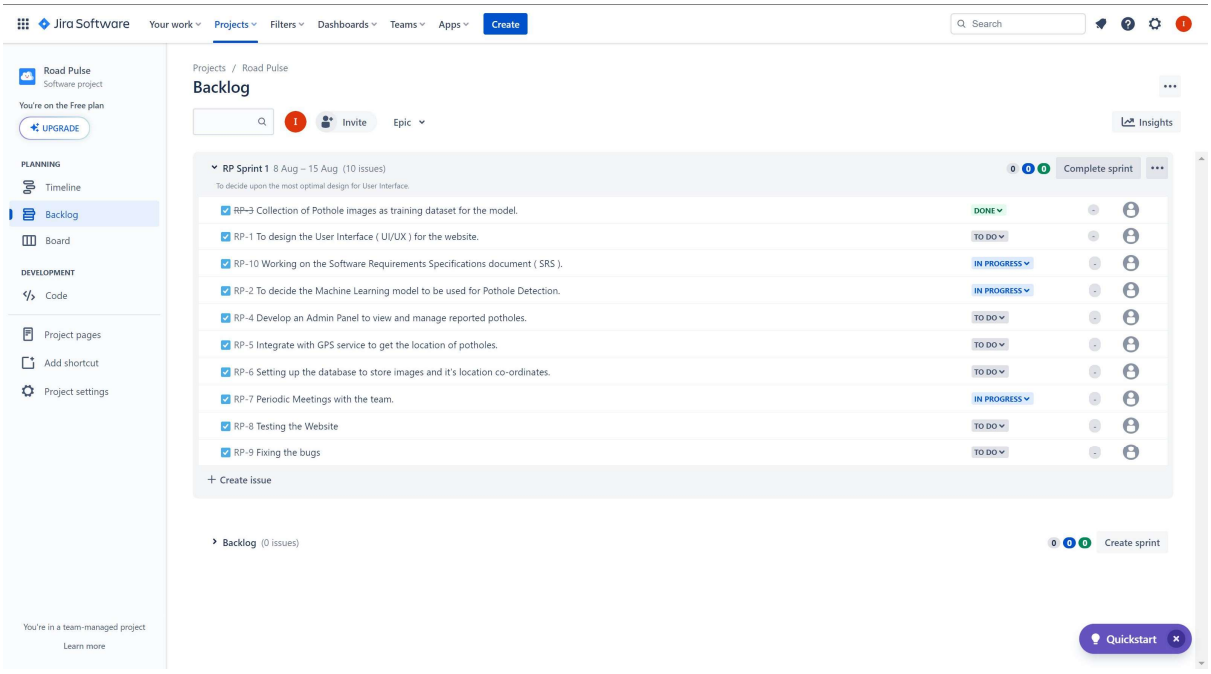
**Challenges of Agile:**

- Adapting to the Agile mindset and practices can be challenging for traditional organizations.
- Maintaining documentation and visibility can be a concern in some Agile implementations.
- Balancing flexibility with predictability in larger projects can be tricky.

## Application of Kanban Model in JIRA Software (KANBAN BOARD):




## BACKLOGS:



# Software Requirements Specification

**for**

# Pothole Detection System

Prepared By:

1. Parth Dabholkar
2. Vedant Devkar
3. Viraj Dighe
4. Ayush Bhanushali

**Thadomal Shahani Engineering College**

# Table of Contents

**Table of Contents ……ii**

**1. Introduction……1**

1.1 Purpose……1

1.2 Intended Audience and Reading Suggestions……..1

1.3 Scope……1

**2. Overall Description……2**

2.1 Product Perspective……2

2.2 Product Functions……2

2.3 User Classes and Characteristics……4

2.4 Operating Environment……6

2.5 Implementation Constraints……8

2.6 Assumptions and Dependencies……..8

**3. Functional Requirements……9**

3.1 Real-time Video Feed Processing……9

3.2 Pothole Detection Algorithm……10

3.3 Alert Generation and Reporting……10

3.4 Data Storage and Analysis……10

3.5 User Authentication and Authorization……10

3.6 Public Reporting (Future Enhancement)……10

3.7 Error Handling and Logging……11

3.8 Security Measures……..11

3.9 Responsive User Interface……..11

3.10 Performance Metrics……11

**4. Non-Functional Requirements……11**

4.1 Performance Requirements……..11

4.2 Safety Requirements……12

4.3 Security Requirements……..12

4.4 Software Quality Attributes……..12

4.5 Business Rules……..13

# 1.Introduction

## 1.1 Purpose

The purpose of a Pothole Detection System is to enhance road safety and infrastructure maintenance by accurately identifying and locating potholes on roadways. This technology aims to reduce vehicle damage and potential accidents caused by these road defects. By utilizing advanced sensors and algorithms, the system detects and reports potholes in real-time, enabling timely repairs and maintenance. Its functionality benefits both drivers, who can navigate roads more safely, and road maintenance teams, who can prioritize and efficiently address road repairs. Ultimately, the system contributes to smoother and safer travel experiences for all road users.

## 1.2 Intended Audience and Reading Suggestions

### Intended Audience:

The document targets engineers, developers, project managers, stakeholders, road maintenance personnel, and regulatory authorities associated with the Pothole Detection System.

### Reading Suggestions:

1. "Image Processing and Analysis for Pothole Detection: A Review" - Offers technical insights into pothole detection algorithms.

2. "Smart Road Infrastructure Management System" - Explores the context of intelligent road systems.

3. "Transportation Infrastructure Management" - Provides a broader understanding of transportation infrastructure and technology.

4. "Intelligent Transportation Systems: Functional Design for Effective Traffic Management" - Covers intelligent transportation systems and their practical applications.

5. “Online Resources from Transportation Agencies” - Presents real-world applications and user perspectives.

## 1.3 Scope

1. Pothole Detection: Accurate identification of potholes using sensors and algorithms on road surfaces.

2. Real-time Monitoring: Continuous and immediate tracking of road conditions for timely detection.

3. Location Precision: Pinpointing geographic coordinates of potholes for efficient maintenance planning.

4. Alerts and Insights: Generating alerts for drivers and maintenance teams and providing data-driven insights on pothole distribution and severity.

5. Integration and Performance: Integration with existing systems, adherence to defined performance metrics for accuracy and responsiveness.

# 2. Overall Description

## 2.1 Product Perspective

The Pothole Detection System specified in this SRS is a new and self contained product aimed at addressing road safety and maintenance challenges through automated pothole detection using Python based technology. It is not a replacement for existing systems nor a follow on member of a product family.

### 2.1.1 Context and Origin

The origin of this product stems from the growing concerns related to road safety and the need for efficient road maintenance. Potholes pose a significant risk to both vehicles and pedestrians, leading to accidents and increased maintenance costs. The traditional manual inspection methods are time consuming and often result in delayed repairs. The Pothole Detection System addresses this issue by leveraging modern computer vision techniques to identify potholes accurately and in real time.

### 2.1.2 Role within the Larger System

The Pothole Detection System operates as a component of a larger urban infrastructure and road maintenance ecosystem. While the system itself is self contained, it interfaces with existing road cameras and may also collaborate with the local traffic control center. It plays a pivotal role in detecting potholes and generating alerts, which contribute to the overall road safety and maintenance efforts.

## 2.2 Product Functions

The "Product Functions" section of the pothole detection system outlines the specific capabilities and tasks that the system can perform. Here's a breakdown of the product functions for your system:

### 2.2.1 Capture Data:

Collect road surface data using cameras, sensors, or other hardware components installed on vehicles or fixed installations.

### 2.2.2. Preprocess Data:

Apply image enhancement and noise reduction techniques to improve the quality of captured data.

### 2.2.3. Analyse Data:

Utilize computer vision algorithms and machine learning models to Analyse the preprocessed data for the presence of potholes.

### 2.2.4. Detect Potholes:

Identify and classify potholes based on size, depth, and severity.

### 2.2.5. Geotagging:

Associate geographic coordinates with detected potholes to accurately pinpoint their locations.

### 2.2.6. Severity Assessment:

Determine the severity level of each detected pothole based on predefined criteria (e.g., minor, moderate, severe).

### 2.2.7. Realtime Monitoring:

Continuously monitor road conditions and perform pothole detection in real-time.

### 2.2.8. Alert Generation:

Generate alerts and notifications for maintenance personnel when potholes are detected, providing information about location and severity.

### 2.2.9. User Interface:

Create a user-friendly interface (web or mobile app) for users to access pothole information, view maps, and receive alerts.

### 2.2.10. Reporting:

Allow users to report new or missed potholes, contributing to data accuracy and system improvement.

### 2.2.11. Maintenance Scheduling:

Assist maintenance teams in scheduling repairs by providing information about the location and severity of detected potholes.

### 2.2.12. Integration with Navigation Apps:

Integrate with navigation applications to provide real-time pothole alerts to drivers, enhancing road safety.

### 2.2.13. System Health Monitoring:

Monitor the health of the system's components, detect failures, and provide alerts to administrators.

### 2.2.14. Data Storage:

Store historical data on detected potholes, allowing for trend analysis and infrastructure planning.

### 2.2.15. User Management:

Manage user access, roles, and permissions within the user interface.

### 2.2.16. Integration with Maintenance Databases:

Integrate with existing road maintenance databases to cross reference detected potholes with previous records.

### 2.2.17. Configurable Settings:

Allow administrators to configure system settings, such as detection sensitivity and alert thresholds.

### 2.2.18. Data Visualization:

Present pothole data on interactive maps, enabling users to visualize and explore road conditions.

### 2.2.19. Data Export:

Provide options to export pothole data and reports for further analysis and reporting.

### 2.2.20. System Documentation:

Generate and manage documentation for users, administrators, and maintenance personnel.

## 2.3 User Classes and Characteristics

The "User Classes and Characteristics" section of the pothole detection system describes the different types of users who will interact with the system and outlines their specific characteristics, roles, and requirements. Here's an overview of the user classes and their characteristics:

### 2.3.1. Drivers:

- Characteristics: Everyday road users, including car drivers, motorcyclists, and truck drivers.

- Roles: Indirect interaction with the system through their vehicles equipped with cameras or sensors.

- Requirements: Expect timely alerts about detected potholes to avoid damage to vehicles and ensure road safety.

### 2.3.2. Maintenance Personnel:

- Characteristics: Road maintenance workers and crews responsible for repairing potholes.

- Roles: Receive alerts about detected potholes and use system data to plan and schedule maintenance work.

- Requirements: Need accurate location information, severity assessment, and the ability to mark potholes as repaired.

### 2.3.3. Administrators:

- Characteristics: System administrators responsible for overseeing system operation, maintenance, and user management.

- Roles: Monitor system health, manage user accounts, configure system settings, and ensure smooth operation.

- Requirements: Access to comprehensive system reports, monitoring tools, and administrative controls.

### 2.3.4. App Users (Web/Mobile):

- Characteristics: General public, drivers, and maintenance personnel using the system's user interface.

- Roles: Interact with the system to view pothole information, access maps, and report new potholes.

- Requirements: User-friendly interface, real-time alerts, easy access to pothole data, and seamless navigation.

### 2.3.5. Road Maintenance Authorities:

- Characteristics: Government or local agency representatives responsible for road infrastructure and safety.

- Roles: Collaborate with the system to receive pothole data, plan maintenance activities, and allocate resources.

- Requirements: Integration capabilities to incorporate system data into existing road maintenance processes.

### 2.3.6. Developers and Technicians:

- Characteristics: Technical experts involved in the development, maintenance, and troubleshooting of the system.

- Roles: Develop and maintain the system, troubleshoot issues, and ensure continuous operation.

- Requirements: Access to system logs, debugging tools, and documentation.

### 2.3.7. Data Analysts and Planners:

- Characteristics: Professionals responsible for analyzing pothole data and using insights for infrastructure planning.

- Roles: Analyze trends in pothole data, make informed decisions about road maintenance, and allocate resources effectively.

- Requirements: Access to historical pothole data, data export functionality, and data visualization tools.

## 2.4 Operating Environment

The Pothole Detection System operates in a specific environment that encompasses the hardware, operating system, and software components it interacts with. Ensuring compatibility and seamless interaction with these elements is crucial for the system's accurate and efficient functioning.

### 2.4.1 Hardware Platform:

The Pothole Detection System is designed to run on hardware with the following specifications or higher:

Processor: Dual-core 2.0 GHz or higher

Memory: 4 GB RAM

Storage: 100 GB available disk space

Camera: High-definition (1080p or higher) video feed from road cameras

Network Connectivity: Stable internet connection for data transmission and alerts

### 2.4.2 Operating System:

The system is compatible with the following operating systems:

Windows 10 or higher

Linux distributions (e.g., Ubuntu 18.04 or higher)

Python Version:

The system relies on Python for its algorithm and processing. It is compatible with Python 3.6 or higher. The specific Python libraries used in the algorithm should also be listed.

### 2.4.3 Dependencies:

The Pothole Detection System relies on the following software components and applications:

OpenCV: Computer vision library for image and video analysis.

NumPy: Numerical computing library for data manipulation.

Network Protocol Libraries: Libraries for handling video feed streaming and data transmission (e.g., HTTP, MQTT).

### 2.4.4 Database:

If the system involves data storage and analysis, it may require a compatible database system:

MySQL 5.7 or higher

PostgreSQL 10 or higher

### 2.4.5 Camera Hardware:

The system expects to receive video feeds from road cameras that meet the following criteria:

High-definition resolution (1080p or higher)

Compatible video streaming protocols (e.g., RTSP)

### 2.4.6 External Interfaces:

The Pothole Detection System may need to interact with external systems and services:

Local Traffic Control Center: For receiving maintenance alerts and integrating with the broader road infrastructure.

GPS Data: Integration with GPS data can enhance the accuracy of pothole location determination.

### 2.4.7 Network Requirements:

The system relies on a stable and reliable internet connection for:

Receiving live video feeds from cameras.

Transmitting alerts and reports to relevant stakeholders.

Accessing software updates and patches.

## 2.5 Implementation Constraints

Several constraints impact the implementation of the Pothole Detection System. These constraints encompass technical, regulatory, and design considerations that guide developers in creating a functional and compliant system.

### 2.5.1 Hardware Limitations:

The Pothole Detection System's performance is dependent on the quality and capabilities of the cameras capturing video feeds. Cameras with resolutions below 1080p might lead to decreased accuracy in pothole detection.

### 2.5.2 Regulatory Compliance:

The system must adhere to data protection regulations and privacy standards. Video feeds should not invade private property or capture sensitive information unrelated to pothole detection.

### 2.5.3 Programming Language and Libraries:

The system's implementation is constrained to using Python 3.6 or higher due to the chosen algorithm's compatibility. The system relies on specific libraries such as OpenCV and NumPy for image analysis and processing.

### 2.5.4 Communication Protocols:

The Pothole Detection System must support standard communication protocols for real-time video feed streaming, such as RTSP. Additionally, any communication with external systems (e.g., Traffic Control Center) must comply with security protocols and standards.

## 2.6 Assumptions and Dependencies

### 2.6.1 Assumptions

1. Hardware Installation: The system assumes that hardware components (cameras, sensors, GPS devices) are properly installed and functioning on vehicles, fixed installations, or drones.

2. Data Connectivity: Adequate data connectivity, such as cellular or wireless networks, is assumed to be available for real-time data transmission between the hardware components and the central processing system.

3. User Familiarity: Users are assumed to have a basic level of familiarity with mobile devices or web interfaces, as they will interact with the system through user-friendly applications.

4. Data Quality: The system assumes that captured data from cameras and sensors are of sufficient quality for accurate processing and analysis.

5. Road Infrastructure: The system assumes that roads and road conditions conform to standard norms and regulations, allowing for consistent data collection.

### 2.6.2 Dependencies

1. Hardware Performance: The accuracy of pothole detection depends on the performance and capabilities of installed cameras and sensors. Any limitations or failures in these hardware components can impact data quality and detection accuracy.

2. Network Connectivity: The system relies on stable and reliable network connectivity to transmit data from the field to the central processing system and to deliver real-time alerts to users and maintenance personnel.

3. External System Integration: If the system integrates with navigation apps or local road maintenance databases, it is dependent on the availability and compatibility of the APIs or interfaces provided by those external systems.

4. Data Accuracy: The effectiveness of pothole detection relies on accurate data preprocessing, image analysis, and machine learning algorithms. Any inconsistencies or errors in data processing can affect the reliability of detected potholes.

# 3. Functional Requirements

## 3.1 Real-time Video Feed Processing:

REQ-1: The system shall continuously capture and process live video feeds from road cameras.

If a camera feed is unavailable or interrupted, the system shall log the error and attempt reconnection.

The system shall process video frames at a rate of at least 10 frames per second (fps).

If the frame processing rate drops below 10 fps, the system shall trigger an alert for review.

## 3.2 Pothole Detection Algorithm:

REQ-2: The system shall utilize a Python-based algorithm to analyze video frames and identify potential potholes.

The algorithm shall differentiate between road surface irregularities and actual potholes.

If the algorithm's detection confidence falls below 70%, the system shall log the detection as "low confidence."

## 3.3 Alert Generation and Reporting:

REQ-3: When a pothole is detected, the system shall generate an alert specifying the pothole's location.

The system shall transmit alerts to designated maintenance personnel via email and SMS.

If the alert transmission fails, the system shall retry the transmission up to three times.

The system shall log failed alert transmissions for administrative review.

## 3.4 Data Storage and Analysis:

REQ-4: The system shall store detected pothole data in a centralized database.

Each pothole entry shall include timestamp, location, and detection confidence level.

The system shall allow authorized users to query the database for historical pothole data.

In case of database connection issues, the system shall maintain a local cache and sync data when connection is restored.

## 3.5 User Authentication and Authorization:

REQ-5: Authorized administrators shall have access to the system's backend.

User authentication shall be based on username and password.

Failed login attempts shall trigger an account lockout for 15 minutes.

## 3.6 Public Reporting (Future Enhancement):

REQ-6: Public users shall be able to report potential potholes through a mobile app.

Submitted reports shall include a photo, GPS location, and optional description.

The system shall generate alerts for maintenance crews based on user-submitted reports.

### 3.7 Error Handling and Logging:

REQ-7: The system shall log all errors, exceptions, and alerts for diagnostic and auditing purposes.

Errors shall be categorized and prioritized based on severity.

### 3.8 Security Measures:

REQ-8: The system shall encrypt transmitted data to ensure data integrity and confidentiality.

User passwords shall be securely hashed before storage.

### 3.9 Responsive User Interface:

REQ-9: The user interface shall provide real-time updates on camera feeds and detection status.

The system shall display a user-friendly error message for any failed operations or invalid inputs.

### 3.10 Performance Metrics:

REQ:10 The system shall log processing times for each frame analysis.

The system shall track the accuracy of pothole detection based on ground truth data.

## 4. Non-Functional Requirements:

### 4.1 Performance Requirements

4.1.1 The system must achieve a minimum accuracy of 90% in detecting and classifying potholes from sensor data.

4.1.2 The time taken to process and analyse captured data should not exceed 200 milliseconds.

4.1.3 The system should support a minimum of 100 concurrent data streams from different sensors.

4.1.4 The response time for notifying authorities about detected potholes should be within 2 seconds.

4.1.5 The system's performance metrics and accuracy should be regularly monitored and reported to stakeholders.

## 4.2 Safety Requirements

4.2.1 The system should operate without causing any interference or disruption to the normal functioning of the vehicle's control systems.

4.2.2 False positive rates should be minimized to avoid distracting drivers with unnecessary alerts.

4.2.3 The system must not introduce any risks to the safety of pedestrians, cyclists, or other road users during data collection and analysis.

4.2.4 All system components and sensors must adhere to relevant safety standards and regulations.

4.2.5 The system should be able to adapt to varying weather and road conditions to maintain reliable operation.

## 4.3 Security Requirements

4.3.1 Data transmitted between sensors and the central system must be encrypted using industry-standard encryption protocols.

4.3.2 The system should implement strong user authentication and role-based access control to prevent unauthorized access to data and control features.

4.3.3 Personally identifiable information (PII) of users and drivers should be securely stored and processed in compliance with data protection regulations.

4.3.4 The system should have mechanisms in place to detect and prevent tampering, unauthorized modifications, and unauthorized access to stored data.

4.3.5 Regular security audits and vulnerability assessments should be conducted to ensure the system's resistance to potential cyber threats.

## 4.4 Software Quality Attributes

4.4.1 The system should maintain high availability, with a target uptime of 99.9%.

4.4.2 The user interface should be intuitive and easy to use, requiring minimal training for users to operate effectively.

4.4.3 The system's architecture should be designed to support scalability, enabling it to accommodate a growing number of sensors and users.

4.4.4 Regular performance testing and optimization should be carried out to ensure smooth operation even under peak loads.

4.4.5 Comprehensive documentation should be provided to aid in system setup, configuration, and troubleshooting.

## 4.5 Business Rules

4.5.1 Users should have the ability to manually report potholes, which will be considered alongside automated detections.

4.5.2 The system should prioritize detected potholes based on factors such as their severity, geographic location, and potential impact on road safety.

4.5.3 Integration with existing road maintenance processes should be seamless, allowing for efficient scheduling and execution of repairs.

4.5.4 The system should offer customizable notification preferences for authorities and maintenance crews.

4.5.5 Regular reviews of business rules and system processes should be conducted to ensure alignment with changing road maintenance strategies and priorities.

# EXPERIMENT – 4

**AIM:** Develop Data Flow Diagram (DFD) for the project using Lucid-Chart

## THEORY:

DFD is the abbreviation for Data Flow Diagram. The flow of data of a system or a process is represented by DFD. It also gives insight into the inputs and outputs of each entity and the process itself. DFD does not have control flow and no loops or decision rules are present. Specific operations depending on the type of data can be explained by a flowchart. It is a graphical tool, useful for communicating with users ,managers and other personnel. it is useful for analyzing existing as well as proposed system.

**It provides an overview of**

● What data is system processes.

● What transformation are performed.

● What data are stored.

● What results are produced , etc.

**Components of DFD:**

The Data Flow Diagram has 4 components:

● **Process** Input to output transformation in a system takes place because of process function. The symbols of a process are rectangular with rounded corners, oval, rectangle or a circle. The process is named a short sentence, in one word or a phrase to express its essence

● **Data Flow** Data flow describes the information transferring between different parts of the systems. The arrow symbol is the symbol of data flow. A relatable name should be given to the flow to determine the information which is being moved. Data flow also represents material along with information that is being moved. Material shifts are modeled in systems that are not merely informative. A given flow should only transfer a single type of information. The direction of flow is represented by the arrow which can also be bi-directional.

● **Warehouse** The data is stored in the warehouse for later use. Two horizontal lines represent the symbol of the store. The warehouse is simply not restricted to being a data file rather it can be anything like a folder with documents, an optical disc, a filing cabinet. The data warehouse can be viewed independent of its implementation. When the data flow

from the warehouse it is considered as data reading and when data flows to the warehouse it is called data entry or data updating.

● **Terminator** The Terminator is an external entity that stands outside of the system and communicates with the system. It can be, for example, organizations like banks, groups of people like customers or different departments of the same organization, which is not a part of the model system and is an external entity. Modeled systems also communicate with terminator.

**Rules for creating DFD**

● The name of the entity should be easy and understandable without any extra assistance(like comments).

● The processes should be numbered or put in ordered list to be referred easily.

● The DFD should maintain consistency across all the DFD levels.

● A single DFD can have a maximum of nine processes and a minimum of three processes.

**Symbols Used in DFD**

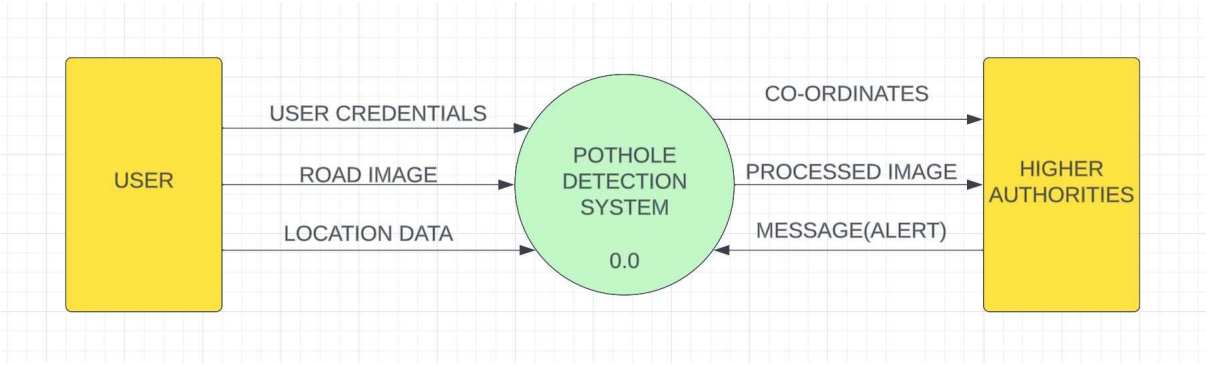
● **Square Box**: A square box defines source or destination of the system. It is also called entity. It is represented by rectangle.

● **Arrow or Line**: An arrow identifies the data flow i.e. it gives information to the data that is in motion.

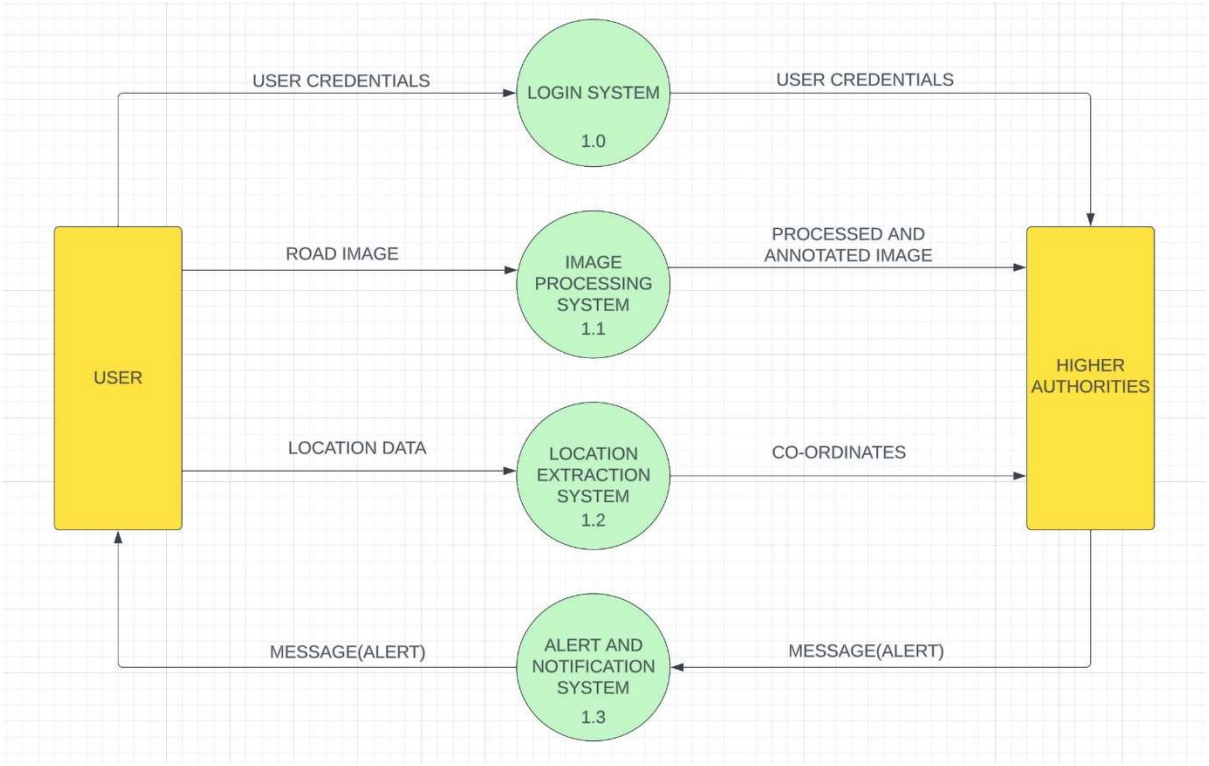
● **Circle or bubble chart**: It represents as a process that gives us information. It is also called processing box.

● **Open Rectangle:** An open rectangle is a data store. In this data is store either temporary or permanently.

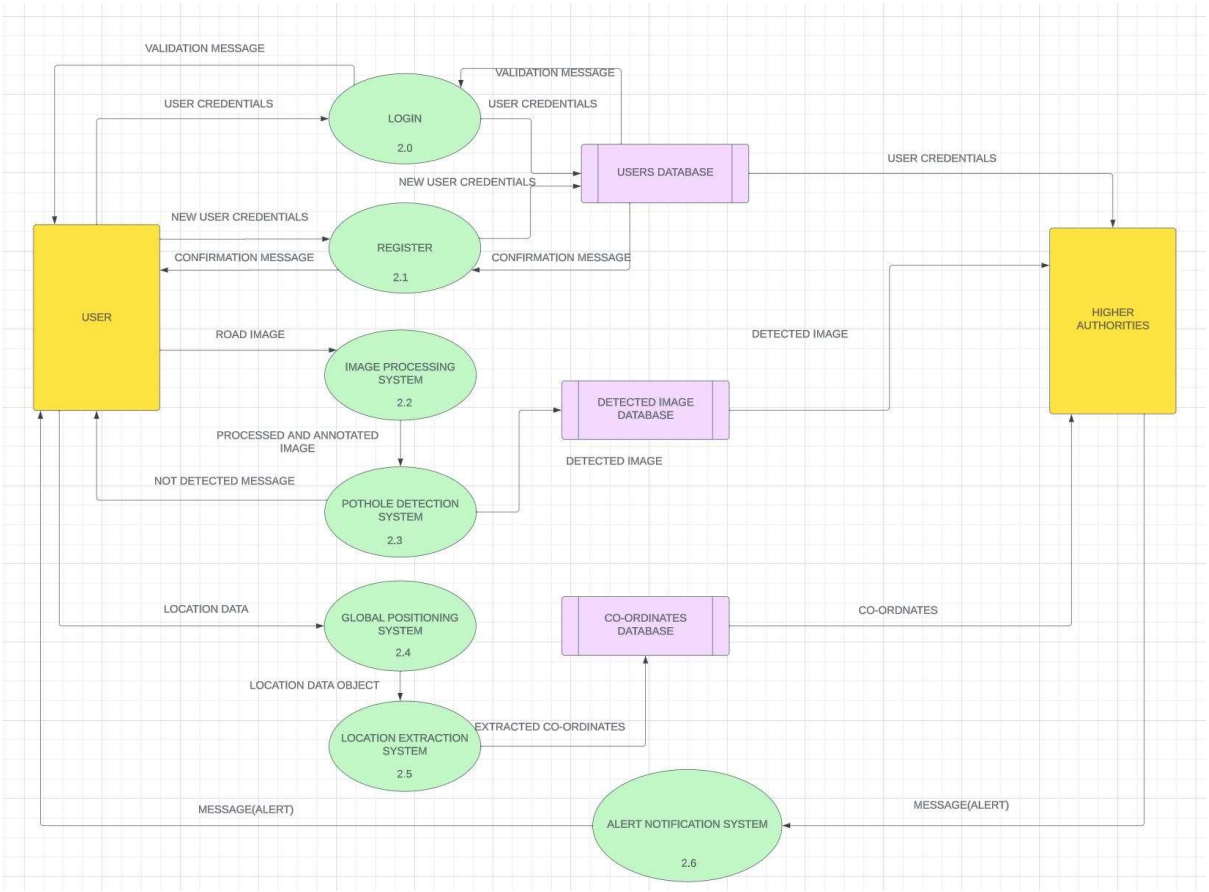
## LEVEL-0 DFD



## LEVEL-1 DFD




## LEVEL-2 DFD



# EXPERIMENT – 5

**Aim**: Develop Activity & State Diagram for the project (Smart Draw, Lucid Chart)

**Theory:**

We use Activity Diagrams to illustrate the flow of control in a system and refer to the steps involved in the execution of a use case. We model sequential and concurrent activities using activity diagrams. So, we basically depict workflows visually using an activity diagram. An activity diagram focuses on condition of flow and the sequence in which it happens. We describe or depict what causes a particular event using an activity diagram. UML models basically three types of diagrams, namely, structure diagrams, interaction diagrams, and behavior diagrams. An activity diagram is a behavioral diagram i.e., it depicts the behavior of a system. An activity diagram portrays the control flow from a start point to a finish point showing the various decision paths that exist while the activity is being executed. We can depict both sequential processing and concurrent processing of activities using an activity diagram. They are used in business and process modelling where their primary use is to depict the dynamic aspects of a system. An activity diagram is very similar to a flowchart.

**Components of an Activity Diagram**

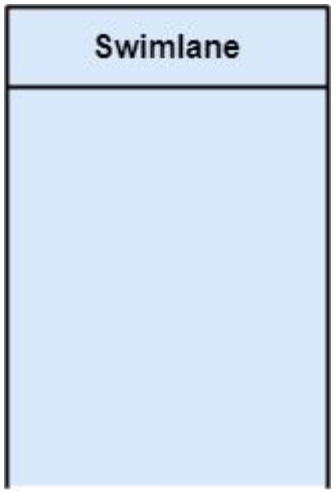
**Activities**

The categorization of behavior into one or more actions is termed as an activity. In other words, it can be said that an activity is a network of nodes that are connected by edges. The edges depict the flow of execution. It may contain action nodes, control nodes, or object nodes.

The control flow of activity is represented by control nodes and object nodes that illustrates the objects used within an activity. The activities are initiated at the initial node and are terminated at the final node.

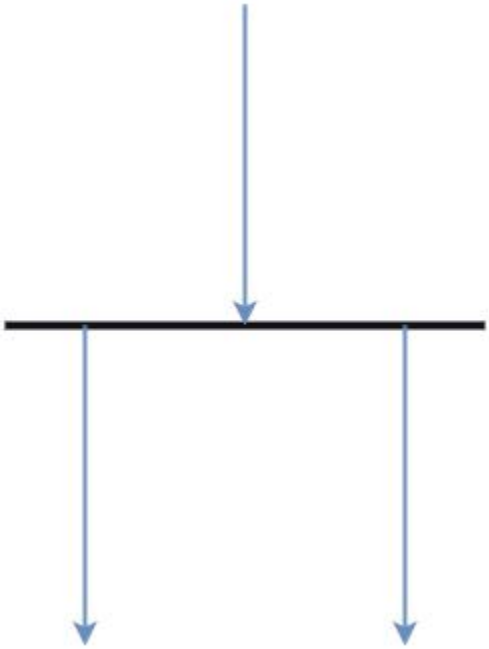

**Activity partition /swimlane**

The swimlane is used to cluster all the related activities in one column or one row. It can be either vertical or horizontal. It used to add modularity to the activity diagram. It is not necessary to incorporate swimlane in the activity diagram. But it is used to add more transparency to the activity diagram.



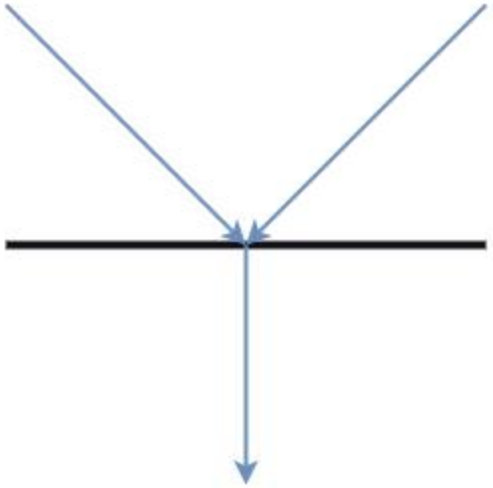

**Forks**

Forks and join nodes generate the concurrent flow inside the activity. A fork node consists of one inward edge and several outward edges. It is the same as that of various decision parameters. Whenever a data is received at an inward edge, it gets copied and split crossways various outward edges. It split a single inward flow into multiple parallel flows.



**Join Nodes**

Join nodes are the opposite of fork nodes. A Logical AND operation is performed on all of the inward edges as it synchronizes the flow of input across one single output (outward) edge.



**Pins**

It is a small rectangle, which is attached to the action rectangle. It clears out all the messy and complicated thing to manage the execution flow of activities. It is an object node that precisely represents one input to or output from the action.

Notation of an Activity diagram

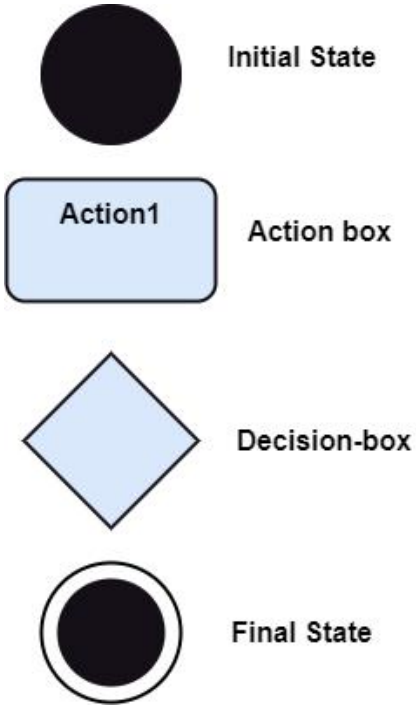
Activity diagram constitutes following notations:

**Initial State:** It depicts the initial stage or beginning of the set of actions.

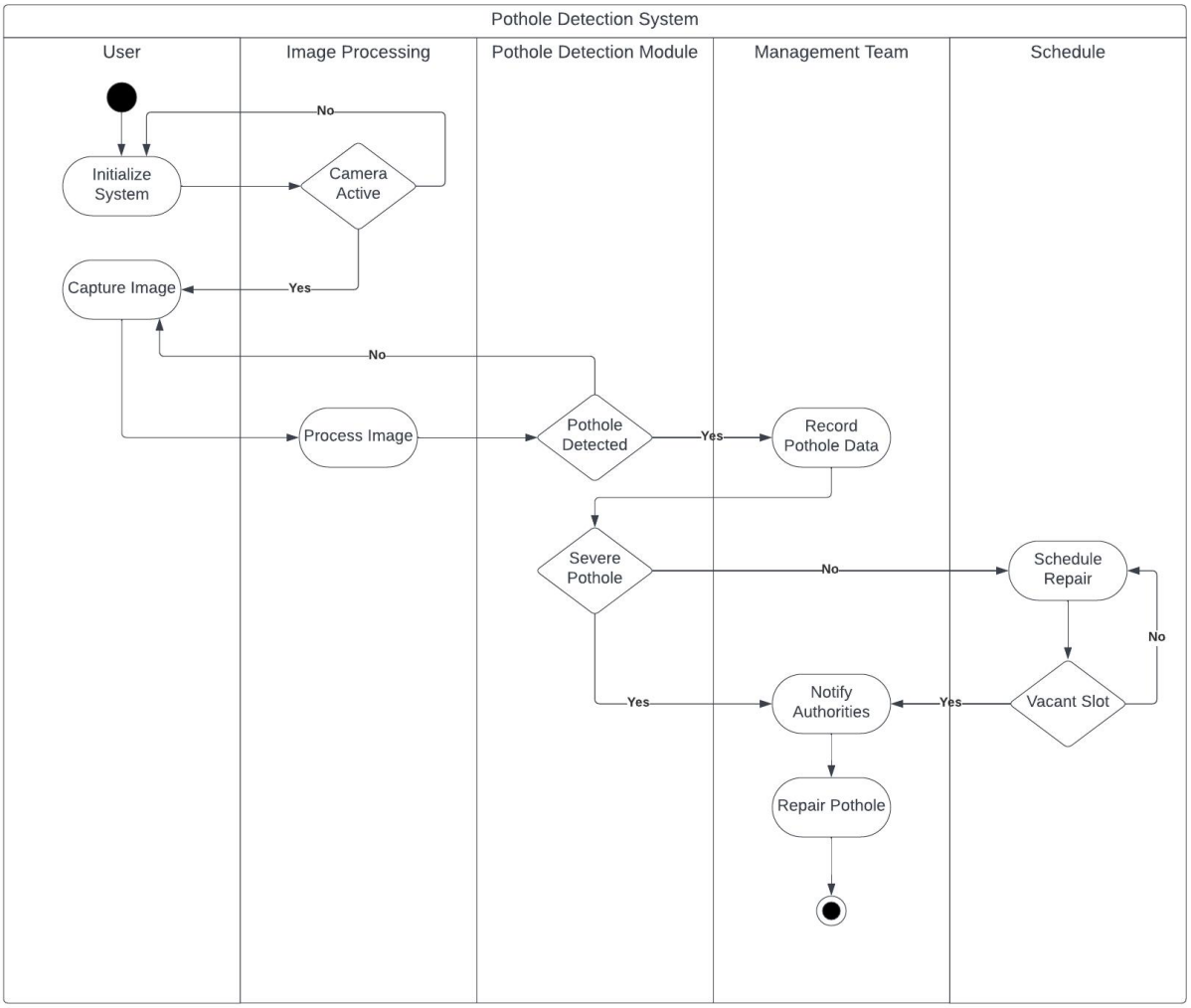
**Final State:** It is the stage where all the control flows and object flows end.

**Decision Box:** It makes sure that the control flow or object flow will follow only one path.

**Action Box:** It represents the set of actions that are to be performed.



# ACTIVITY DIAGRAM



A state diagram is a graphical representation of a finite state machine (FSM) that depicts the various states an object can be in and the transitions between those states. State diagrams are widely used in various fields such as software engineering, control systems, and even in modeling complex processes. Here are some theories and concepts related to state diagrams:

States: States are the fundamental elements of a state diagram. They represent a condition or a situation in the system or object being modeled. In software engineering, for example, a state can represent the state of an object, such as "idle," "active," or "error."

Transitions: Transitions are the arrows or lines connecting states in a state diagram. They indicate how the system can move from one state to another in response to certain events or conditions. Transitions often have labels that describe the events or conditions triggering the state change.

Initial State: An initial state, often denoted by a filled circle, represents the starting point of the system or object's behavior. It's the state in which the system is in when it is first activated.

Final State: A final state, often denoted by a double circle, represents the end point of a particular behavior or process. When the system reaches a final state, it has completed a specific task or process.

Events: Events are occurrences that trigger a change in state. Events can be external inputs or internal conditions that cause a transition from one state to another. For example, in a vending machine, an event could be inserting a coin.

Actions: Actions are associated with transitions and represent the activities or operations that occur when a transition is taken. These actions can be functions, procedures, or changes in variables that occur when moving from one state to another.

Guard Conditions: Guard conditions are conditions that must be satisfied for a transition to occur. They are often written near the transition arrow and specify when a particular transition should be taken.

Hierarchical States: State diagrams can be hierarchical, meaning that states can contain sub-states. This allows for the modeling of complex systems with various levels of granularity.

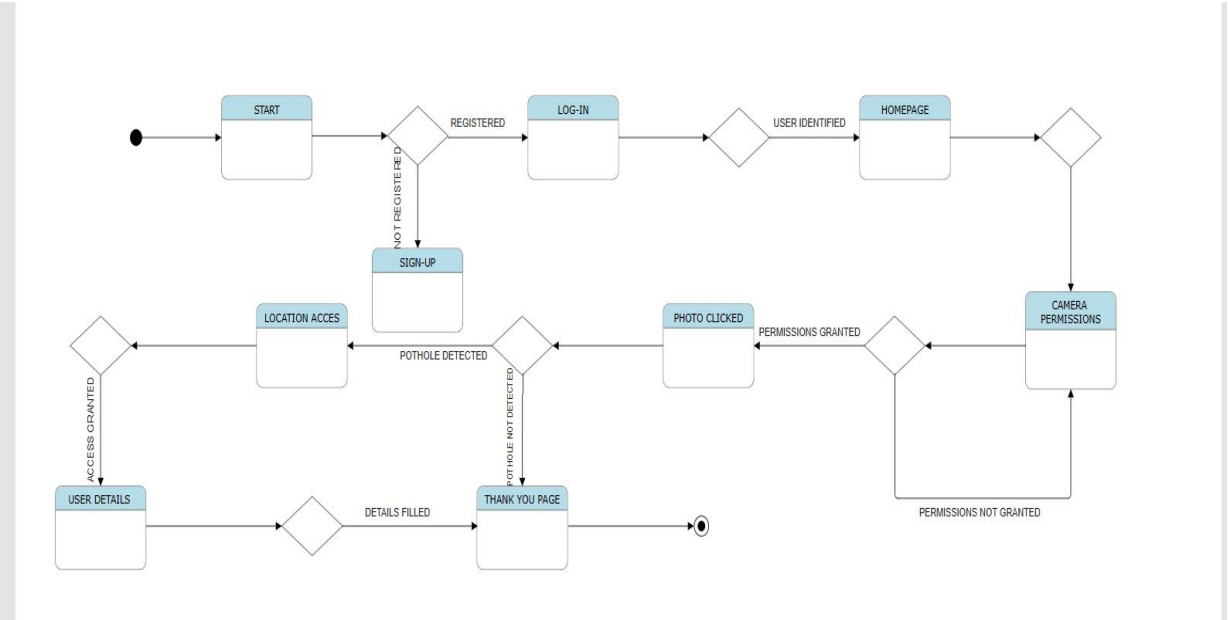
Parallel States: In some state diagrams, it's possible to have parallel or concurrent states that can coexist and transition independently. This is useful for modeling systems with multiple concurrent activities.

History States: Some state diagrams allow for history states, which remember the last sub-state that was active within a superstate. This can be helpful for returning to the last known state in complex state machines.

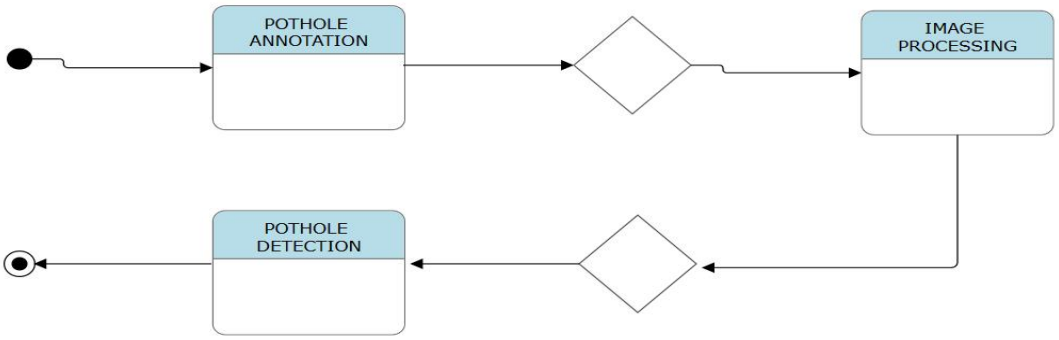
Orthogonal States: In more advanced state diagrams, orthogonal states can exist within a superstate, and transitions between these orthogonal states can occur independently.

State diagrams are a fundamental tool for designing and understanding complex systems, helping to visualize the behavior and flow of states in a clear and comprehensible manner. They are often used in software design, control systems, and any domain where systems can be described in terms of discrete states and state transitions.

CLASS : USER STATE DIAGRAM




CLASS B: ML MODEL STATE DIAGRAM



# EXPERIMENT – 6

Aim: Identify scenarios & Develop Use Case Diagram for the project.

## Theory:

In UML, use-case diagrams model the behavior of a system and help to capture the requirements of the system.

Use-case diagrams describe the high-level functions and scope of a system. These diagrams also identify the interactions between the system and its actors. The use cases and actors in use-case diagrams describe what the system does and how the actors use it, but not how the system operates internally.

Use-case diagrams illustrate and define the context and requirements of either an entire system or the important parts of the system. You can model a complex system with a single use-case diagram, or create many use-case diagrams to model the components of the system. You would typically develop use-case diagrams in the early phases of a project and refer to them throughout the development process.

Use-case diagrams are helpful in the following situations:

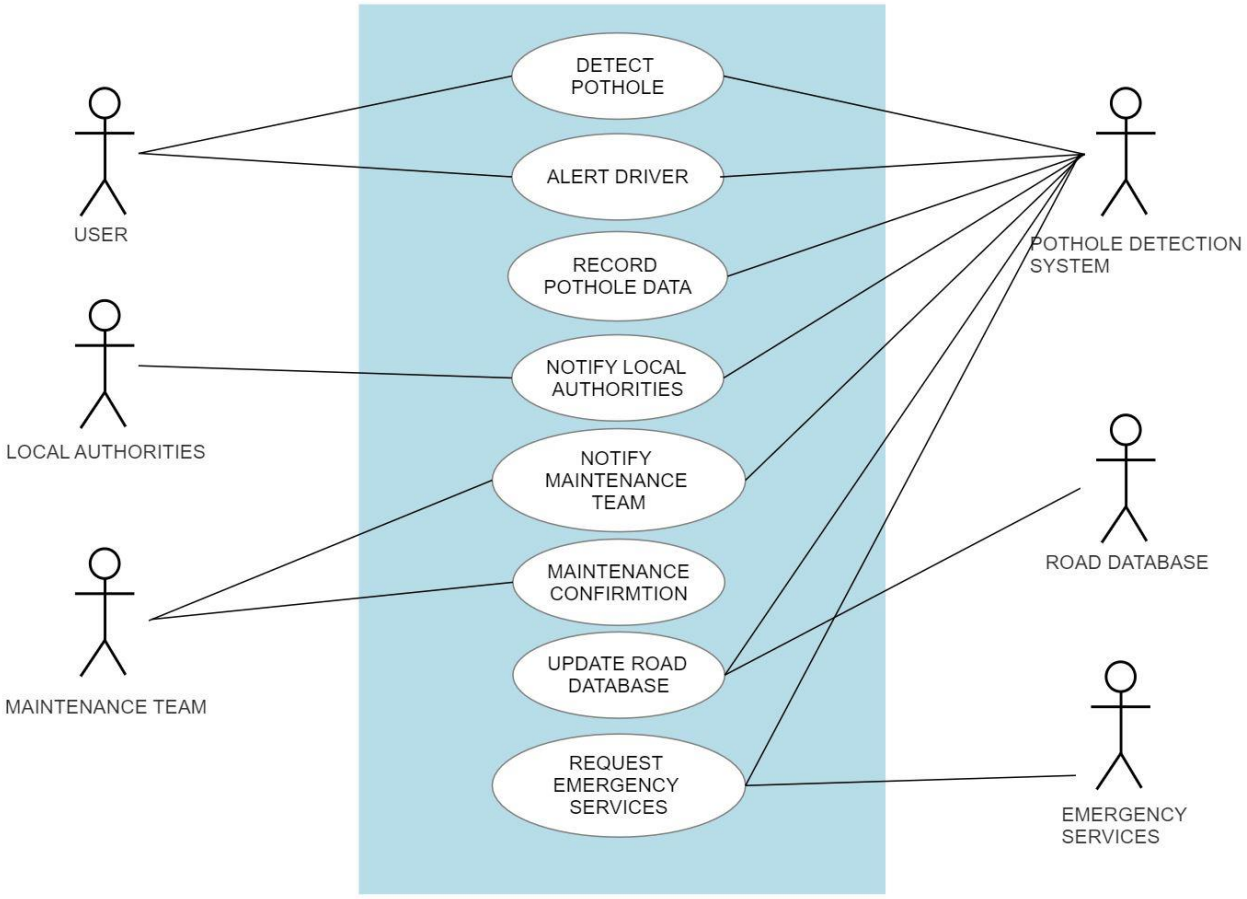
- Before starting a project, you can create use-case diagrams to model a business so that all participants in the project share an understanding of the workers, customers, and activities of the business.
- While gathering requirements, you can create use-case diagrams to capture the system requirements and to present to others what the system should do.
- During the analysis and design phases, you can use the use cases and actors from your use-case diagrams to identify the classes that the system requires.
- During the testing phase, you can use use-case diagrams to identify tests for the system.

The following topics describe model elements in use-case diagrams:

- **Use cases**
  A use case describes a function that a system performs to achieve the user's goal. A use case must yield an observable result that is of value to the user of the system.
- **Actors**
  An actor represents a role of a user that interacts with the system that you are modeling. The user can be a human user, an organization, a machine, or another external system.

- **Subsystems**
  In UML models, subsystems are a type of stereotyped component that represent independent, behavioral units in a system. Subsystems are used in class, component, and use-case diagrams to represent large-scale components in the system that you are modeling.
- **Relationships in use-case diagrams**
  In UML, a relationship is a connection between model elements. A UML relationship is a type of model element that adds semantics to a model by defining the structure and behavior between the model elements.

## OUTPUT


DETECT POTHOLE
ALERT DRIVER
RECORD POTHOLE DATA
NOTIFY LOCAL AUTHORITIES
NOTIFY MAINTENANCE TEAM
MAINTENANCE CONFIRMTION
UPDATE ROAD DATABASE
REQUEST EMERGENCY SERVICES
USER
LOCAL AUTHORITIES
MAINTENANCE TEAM
POTHOLE DETECTION SYSTEM
ROAD DATABASE
EMERGENCY SERVICES

# Experiment No.7

**Aim**: Create a project schedule using gantt chart.

**Theory**: A Gantt chart in software engineering is a visual representation of a project schedule, typically used to plan, track, and manage software development projects. Here's a theoretical explanation of the Gantt chart in software engineering:

1. Project Phases and Tasks:

A Gantt chart breaks down a software development project into phases and tasks. These phases may include requirements gathering, design, development, testing, and deployment. Each phase is represented as a separate row on the Gantt chart, and the tasks within each phase are listed under their respective phase.

2. Timeline:

The horizontal axis of the Gantt chart represents time, usually in days, weeks, or months. The timeline spans from the project's start date to its end date. Each task is scheduled based on when it should begin and how long it is expected to take. The timeline provides a clear visual representation of the project's duration.

3. Task Dependencies:

Gantt charts allow for the depiction of task dependencies. Tasks often depend on the completion of other tasks before they can start. Dependencies are shown as arrows connecting tasks. This helps project managers and team members understand the order in which tasks should be completed.

4. Milestones:

Milestones are significant points in a project that mark the completion of a major phase or the achievement of a critical objective. Milestones are usually represented as diamonds on the Gantt chart. They help project stakeholders easily identify key project events.

5. Resource Allocation:

A Gantt chart may also include information about resource allocation. This can include the assignment of specific team members or teams to each task and the allocation of tools or equipment as needed. Resource allocation ensures that the right resources are available when needed.

6. Tracking Progress:

Throughout the project, the Gantt chart is a valuable tool for tracking progress. As tasks are completed, they are shaded or marked to indicate their status. This allows project managers and team members to quickly assess whether the project is on schedule or if any tasks are behind.

7. Communication:

Gantt charts are powerful communication tools. They provide a clear, visual representation of the project schedule that can be easily shared with stakeholders, team members, and clients. This transparency enhances communication and ensures that everyone is on the same page regarding project timelines and progress.

8. Iterative and Adaptive Planning:

In agile software development methodologies, Gantt charts may be used in an iterative and adaptive manner. As project teams work in sprints or iterations, the Gantt chart is updated to reflect the evolving project plan. This adaptability is important for projects with changing requirements.

9. Risk Management:

Gantt charts can be used to identify potential project risks. If a task falls behind schedule, it may impact the overall project timeline. Project managers can use the Gantt chart to analyze and mitigate these risks by adjusting the schedule, allocating more resources, or reorganizing tasks.

In summary, a Gantt chart in software engineering is a comprehensive project management tool that offers a structured way to plan, track, and manage software development projects. It provides a visual representation of the project's timeline, dependencies, and resource allocation, helping project managers and teams ensure successful project execution.

# GANTT CHART

**POTHOLE DETECTION SYSTEM** — PROJECT NAME

**## 6-20-2023** — STAIRT DATE

**8-10-2023** — LAST UPDATE DAT E

| Task ID | Task Name | Start Date | End Date | % Completed | 6-20-2023 | 6-23-2023 | 6-27-2023 | 6-29-2023 | 7-1-2023 | 7-9-2023 | 7-16-2023 | 7-23-2023 | 7-30-2023 | 8-1-2023 | 8-2-2023 | 8-16-2023 | 8-24-2023 | 8-25-2023 | 8-31-2023 | 9-1-2023 | 9-15-2023 | 9-16-2023 | 9-30-2023 | 10-1-2023 | 10-8-2023 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | COLLECTION OF DATA SETS | 6-20-2023 | 6-20-2023 | 100% | | | | | | | | | | | | | | | | | | | | | |
| | DECIDE THE ML MODEL | 6-20-2023 | 6-23-2023 | 100% | | | | | | | | | | | | | | | | | | | | | |
| | WORKING ON SRS | 6-27-2023 | 6-29-2023 | 100% | | | | | | | | | | | | | | | | | | | | | |
| | FIRST TEAM MEETING | 7-1-2023 | 7-1-2023 | 100% | | | | | | | | | | | | | | | | | | | | | |
| | TRAINING THE MODEL | 7-9-2023 | 7-16-2023 | 100% | | | | | | | | | | | | | | | | | | | | | |
| | TESTING THE MODEL | 7-23-2023 | 7-30-2023 | 50% | | | | | | | | | | | | | | | | | | | | | |
| | SECOND TEAM MEETING | 8-1-2023 | 8-1-2023 | 100% | | | | | | | | | | | | | | | | | | | | | |
| | DESIGN THE UI/UX | 8-2-2023 | 8-16-2023 | 20% | | | | | | | | | | | | | | | | | | | | | |
| | INTEGRATE GPS SERVICE | 8-24-2023 | 8-24-2023 | 0% | | | | | | | | | | | | | | | | | | | | | |
| | SETTING UP DATABASE | 8-25-2023 | 8-31-2023 | 0% | | | | | | | | | | | | | | | | | | | | | |
| | DEVELOP ADMIN PANEL | 9-1-2020 | 9-15-2023 | 0% | | | | | | | | | | | | | | | | | | | | | |
| | FIXING THE BUGS | 9-16-2023 | 9-30-2023 | 0% | | | | | | | | | | | | | | | | | | | | | |
| | FINAL TESTING AND DEPLO | 10-1-2023 | 10-8-2023 | 0% | | | | | | | | | | | | | | | | | | | | | |

# EXPERIMENT - 8

**AIM :** Conduct Function Point Analysis (FPA) for the project.

## THEORY:

Function Point Analysis (FPA) is a software metric used to measure the functional size of a software application based on the user's perspective. It quantifies the functionality offered by an application by considering inputs, outputs, inquiries, internal files, and external interfaces. FPA helps in estimating the effort required for development, maintenance, and assessing the project's complexity. It is technology-independent and is used to compare projects and evaluate productivity. FPA is often used in software project management to make informed decisions about resource allocation and project planning.

## CALCULATIONS:

**For a Pothole Detection System, you can identify the following functionality types:**

1. **External Inputs (EIs):**

- Capture video or images from cameras mounted on vehicles.
- Process and analyze captured images to detect potential potholes.
- Record the location and severity of detected potholes.
- Allow users to report potholes manually.

2. **External Outputs (EOs):**

- Display a real-time map with pothole locations to users.
- Generate reports on pothole locations and severity.

3. **External Inquiries (EQs):**

- Provide information about the number of potholes detected in a specific area.

4. **Internal Logical Files (ILFs):**

- Maintain a database of pothole data, including location and severity.
- Store user profiles and authentication information.

5. **External Interface Files (EIFs):**

- Interface with GPS data from vehicles to correlate pothole locations accurately.

**Identify and Document the Function Points:**

For each functionality type, identify specific functions or data elements and assign complexity levels (Low, Average, High).

1. **EIs:**

- Capture video or images (Average)
- Image processing and pothole detection (High)
- Record pothole information (Average)
- User-reported pothole (Average)

2. **EOs:**

- Real-time map display (High)
- Generate pothole reports (Average)

3. **EQs:**

- Provide pothole information (Low)

4. **ILFs:**

- Pothole database maintenance (Average)
- User profiles and authentication (Low)

5. **EIFs:**

- Interface with GPS data (Average)

**Now, let's calculate the unadjusted function points (UFP):**

- UFP = (Total number of Low Complexity functions) + (Total number of Average Complexity functions * 2) + (Total number of High Complexity functions * 3)

- UFP = (2 Low Complexity functions) + (4 Average Complexity functions * 2) + (2 High Complexity functions * 3)

- **UFP = 2 + 8 + 6 = 16 unadjusted function points**

- This gives us a total of 16 unadjusted function points for the Pothole Detection System.

**Here's a list of common VAF factors and their weights:**

1. **Data Communications**: The level of data exchange between the system and external entities (e.g., GPS data).

**Weight**: 0.05

2. **Distributed Data Processing**: The extent to which the processing is distributed across different systems.

   **Weight**: 0.05

3. **Performance**: The response time, throughput, and resource utilization requirements.

   **Weight**: 0.10

4. **Heavily Used Configuration**: If the software is intended to be used in a heavily configured environment.

   **Weight**: 0.10

5. **Transaction Rate**: The rate at which transactions or data are processed.

   **Weight**: 0.15

6. **Online Data Entry**: The complexity of online data entry screens.

   **Weight**: 0.10

7. **End-User Efficiency**: The degree to which the system is designed for efficient use by end-users.

   **Weight**: 0.05

8. **Complex Processing**: The complexity of the algorithms used for pothole detection.

   **Weight**: 0.10

9. **Reusability**: The extent to which components or modules can be reused.

   **Weight**: 0.05

10. **Installation Ease**: The ease of installing the system on vehicles.

    **Weight**: 0.05

11. **Operational Ease:** The ease of operating and maintaining the system.

    **Weight**: 0.05

12. **Multiple Sites**: If the system is used in multiple locations or regions.

    **Weight**: 0.10

13. **Facilitate Change**: The ability of the system to accommodate changes in requirements.

    **Weight**: 0.07

14. **Security and Privacy**: The level of security and privacy requirements.

    **Weight**: 0.08

**Now, assign scores to each of these factors on a scale from 0 to 5, where 0 represents no influence, and 5 represents strong influence:**

Data Communications: 4

Distributed Data Processing: 3

Performance: 4

Heavily Used Configuration: 2

Transaction Rate: 3

Online Data Entry: 3

End-User Efficiency: 4

Complex Processing: 4

Reusability: 2

Installation Ease: 3

Operational Ease: 3

Multiple Sites: 2

Facilitate Change: 3

Security and Privacy: 4

**Now, calculate the VAF as the weighted average of these scores:**

- VAF = (Sum of scores * Weight) / 100
- VAF = [(40.05) + (30.05) + (40.10) + (20.10) + (30.15) + (30.10) + (40.05) + (40.10) + (20.05) + (30.05) + (30.05) + (20.10) + (30.07) + (40.08)] / 100
- VAF = (0.20 + 0.15 + 0.40 + 0.20 + 0.45 + 0.30 + 0.20 + 0.40 + 0.10 + 0.15 + 0.15 + 0.20 + 0.21 + 0.32) / 100
- VAF = 3.43 / 100
- **VAF ≈ 0.0343**

So, the Value Adjustment Factor (VAF) for this simplified example of a Pothole Detection System is approximately 0.0343. This factor will be used to adjust the unadjusted function points (UFP) to calculate the adjusted function points (AFP).

To estimate the effort and resources required for your Pothole Detection System project, you can use the Adjusted Function Points (AFP) we calculated earlier (AFP ≈ 1) along with historical data or industry-standard metrics for effort per function point.

Effort estimation can vary widely depending on the specific context and technology used in your project. It's best to use historical data from similar projects or industry benchmarks to make a more accurate estimate. Typically, you would consider factors such as team productivity, technology stack, project complexity, and more.

**Here's a simplified example:**

1. **Historical Data:** Let's assume that based on historical data for similar projects, the average effort required per function point is 20 person-hours.

2. **Effort Estimation** :

- Effort = AFP * Effort per Function Point
- Effort = 1 (AFP) * 20 person-hours per function point
- Effort = 20 person-hours

This simplified estimation suggests that the Pothole Detection System project would require approximately 20 person-hours of effort. However, please note that this is a very rough estimate, and actual effort estimation should involve a more comprehensive analysis of project-specific factors.

To allocate resources, you would also need to consider factors like team size, skill levels, project duration, and any constraints or limitations specific to your organization or project. The effort estimation is just one component of resource planning, and you would need to integrate it with other project management processes to effectively allocate resources.

**In short, to validate and refine your project estimates for the Pothole Detection System:**

1. Involve stakeholders and experts for feedback.

2. Analyze and clarify project requirements.

3. Identify and mitigate potential risks.

4. Review and adjust the Function Point Analysis (FPA).

5. Allocate and plan resources.

6. Create a detailed project plan.

7. Use historical data for reference.

8. Consider prototyping and testing.

9. Implement monitoring and control mechanisms.

10. Document assumptions and constraints.

11. Maintain open communication with stakeholders.

12. Be flexible and adapt as the project progresses.

# EXPERIMENT – 9

Aim: Application of COCOMO model for cost estimation of the project.

## Theory:

Cocomo (Constructive Cost Model) is a regression model based on LOC, i.e number of Lines of Code. It is a procedural cost estimate model for software projects and is often used as a process of reliably predicting the various parameters associated with making a project such as size, effort, cost, time, and quality.

Different models of Cocomo have been proposed to predict the cost estimation at different levels, based on the amount of accuracy and correctness required. All of these models can be applied to a variety of projects, whose characteristics determine the value of the constant to be used in subsequent calculations.

Software projects under COCOMO model strategies are classified into 3 categories, **organic, semi-detached, and embedded.**

**Organic**: A software project is said to be an organic type if-

- Project is small and simple.
- Project team is small with prior experience.
- The problem is well understood and has been solved in the past.
- Requirements of projects are not rigid, such a mode example is payroll processing system.

**Semi-Detached Mode:** A software project is said to be a Semi-Detached type if-

- Project has complexity.
- Project team requires more experience,better guidance and creativity.
- The project has an intermediate size and has mixed rigid requirements such a mode example is a transaction processing system which has fixed requirements.
- It also includes the elements of organic mode and embedded mode.
- Few such projects are- Database Management System(DBMS), new unknown operating system, difficult inventory management system.

**Embedded Mode:** A software project is said to be an Embedded mode type if-

- A software project has fixed requirements of resources .
- Product is developed within very tight constraints.

- A software project requiring the highest level of complexity, creativity, and experience requirement fall under this category.
- Such mode software requires a larger team size than the other two models.

**Types of COCOMO Models**

COCOMO consists of a hierarchy of three increasingly detailed and accurate forms. Any of the three forms can be adapted according to our requirements. These are types of COCOMO model:

- Basic COCOMO Model
- Intermediate COCOMO Model
- Detailed COCOMO Model

**Basic COCOMO Model**

The basic COCOMO is employed for rough calculations, limiting software estimation precision. This is because the model only considers lines of source code and constant values derived from software project types rather than other elements significantly impacting the software development process.

**Intermediate COCOMO Model**

The Intermediate COCOMO model expands the Basic COCOMO model, which considers a collection of cost drivers to improve the accuracy of the cost-estimating model.

**Advanced/Detailed COCOMO Model**

The model contains all qualities of both Basic COCOMO and Intermediate COCOMO techniques for each Software Engineering process. The model considers each project's development phase (analysis, design, and so on). It is a complex model, considering the fact that it considers more parameters. Here, the whole software is parted into different modules and then the COCOMO model is applied to all the modules separately. Finally, the efforts are joined to calculate the total result.

The six phases of the Advanced/Detailed COCOMO Model are:

- Planning and Requirements
- System Design
- Detailed Design
- Module Code and Testing
- Integration and Testing
- Cost Constructive Model

## COCOMO Model Parameters:

1. **Object Points (OP):** 4000 Object Points
2. **Productivity Rate (PROD):** 12 Object Points per person-month
3. **Development Time (D):** 10 months (as an example)
4. **Personnel Continuity (PVOL):** Nominal (1.00)

## Scale Factors:

- **Precedentness (PREC):** Very Low (0.82)
- **Development Flexibility (FLEX):** High (1.10)
- **Architecture/Risk Resolution (RESL):** Very High (1.30)
- **Team Cohesion (TEAM):** High (1.05)
- **Process Maturity (PMAT):** Nominal (1.00)

## Cost Drivers:

- **Required Software Reliability (RELY):** High (1.21)
- **Database Size (DATA):** High (1.14)
- **Product Complexity (CPLX):** High (1.17)
- **Required Reusability (RUSE):** Nominal (1.00)
- **Documentation Match to Life-Cycle Needs (DOCU):** High (1.23)
- **Execution Time Constraint (TIME):** High (1.29)
- **Storage Constraint (STOR):** Nominal (1.00)

## Effort Adjustment Factor (EAF) Calculation:

$$EAF = PREC \times FLEX \times RESL \times TEAM \times PMAT \times RELY \times DATA \times CPLX \times RUSE \times DOCU \times TIME \times STOR \times PVOL$$

Substituting the values:

$$EAF = 0.82 \times 1.10 \times 1.30 \times 1.05 \times 1.00 \times 1.21 \times 1.14 \times 1.17 \times 1.00 \times 1.23 \times 1.29 \times 1.00 \times 1.00$$

$$EAF \approx 2.98$$

## Effort and Project Size Estimation:

$$EB = OP/PROD \times EAF$$

$$EB = 4000 \text{ OP}/(12 \text{ OP})/\text{person-month} \times 2.98$$

$$EB \approx 995.33 \text{ person-months}$$

## Estimated Project Size (KLOC):

Estimated Project Size (KLOC)=$EB \times PROD$

Estimated Project Size (KLOC)≈995.33 person-months × (12 OP/person-month)/1000

Estimated Project Size (KLOC) ≈ 11.94 KLOC

## Productivity Rate:

Productivity Rate=Estimated Project Size (KLOC) / Development Time (months)

Productivity Rate≈11.94 KLOC / 10 months

Productivity Rate ≈ 1.194 KLOC/month

## Summary:

- **Effort (EB):** Approximately 995.33 person-months
- **Estimated Project Size (KLOC):** Approximately 11.94 KLOC
- **Productivity Rate:** Approximately 1.194 KLOC/month

# EXPERIMENT – 10

**Aim**: Develop a Risk Mitigation, Monitoring and Management Plan (RMMM) for the project.

## Theory:

In most cases, a risk management approach can be found in the software project plan. This can be broken down into three sections: risk mitigation, monitoring, and management (RMMM). All work is done as part of the risk analysis in this strategy. The project manager typically uses this RMMM plan as part of the overall project plan.

Some development teams use a Risk Information Sheet(RIS) to document risk. For faster information handling, such as creation, priority sorting, searching, and other analyses, this RIS is controlled by a database system. Risk mitigation and monitoring will begin after the RMMM is documented and the project is launched.

**Risk Mitigation**

Risk Mitigation is a technique for avoiding risks (Risk Avoidance).

The following are steps to take to reduce the risks:

- Identifying the risk.
- Getting rid of the causes that lead to the production of risk.
- Controlling the relevant documents regularly.
- Conducting timely reviews to accelerate the process.

**Risk Monitoring**

Risk monitoring is an activity used to track a project's progress.

The following are the critical goals of the task.

- To see if the risks that were anticipated actually happen.
- To verify that the risk aversion steps defined for risk are adequately implemented.
- To gather information for future risk assessments.
- To determine which risks generate which problems throughout the project.

**Risk Management and Planning**

Risk management and planning are based on the assumption that the mitigation effort failed and the risk has become a reality. When a risk becomes a reality and produces serious problems, the project manager is in charge of this responsibility. It is easier to manage risks

if the project manager successfully implements project mitigation to eliminate risks. This demonstrates how a manager will respond to each risk. The risk register is the key objective of the risk management plan. This risk register identifies and prioritizes potential dangers to a software project.

**Drawbacks of RMMM**

- It raises the cost of the project.
- Time will be needed more.
- A larger project's implementation of an RMMM could prove to be a time-consuming process in and of itself.
- RMMM cannot guarantee a project will be risk-free; in fact, hazards could materialise after the project has been delivered.
- Not All Organisations Should Use RMMM.
- Data security is an issue for RMMM.
- Focus Loss from Automation

# RMMM PLAN

## Risk 1: Inaccurate Pothole Detection (Impact Scale: 4/5)

- Nature of Risk: Technology limitation or algorithm inefficiency may cause the system to inaccurately detect potholes or produce false positives/negatives.

- Risk Category: Technical

- Probability: Moderate (60%)

Risk Mitigation, Monitoring, and Management (RMMM) Plan:

- Mitigation Strategy: Thoroughly test and fine-tune the detection algorithms using diverse datasets and real-world testing. Involve experts to refine the algorithms and incorporate machine learning to improve accuracy over time.

- Monitoring: Regularly monitor the system's performance and conduct validation tests in various road conditions to identify inaccuracies. Continuously gather user feedback and reports to improve the algorithm further.

- Management: Establish a dedicated team to address and resolve detection inaccuracies promptly. Have a protocol in place to update and improve the algorithm periodically based on new data and advancements in technology.

## Risk 2: Hardware Failure (Impact Scale: 2/5)

- Nature of Risk: Malfunction or breakdown of hardware components like cameras, sensors, or processing units.

- Risk Category: Technical/Operational

- Probability: Low (30%)

RMMM Plan:

- Mitigation Strategy: Use high-quality and redundant hardware components, and conduct regular maintenance to prevent hardware failures. Have a backup system in place to take over in case of a failure.

- Monitoring: Implement real-time monitoring of hardware health and performance. Set up automated alerts for potential issues and conduct periodic inspections and maintenance.

- Management: Maintain a readily available stock of replacement components and a clear procedure to follow in the event of hardware failure. Coordinate with relevant technicians or vendors for timely repairs or replacements.

## Risk 3: Data Privacy and Security Breach (Impact Scale: 3/5)

- Nature of Risk: Unauthorized access, hacking, or data leaks compromising sensitive information collected by the pothole detection system.

- Risk Category: Security

- Probability: Moderate (50%)

RMMM Plan:

- Mitigation Strategy: Implement strong encryption protocols, access controls, and regular security audits. Ensure compliance with relevant data privacy laws and standards.

- Monitoring: Conduct regular security assessments and penetration testing to identify vulnerabilities. Monitor system access and usage to detect any suspicious activities.

- Management: Have an incident response plan in place to handle potential breaches, including steps for containment, investigation, and communication to stakeholders. Involve legal and IT teams for appropriate actions and notifications.

## Risk 4: Weather Conditions Affecting Performance (Impact Scale: 5/5)

- Nature of Risk: Adverse weather conditions like heavy rain, snow, or extreme temperatures affecting the system's functionality and accuracy.

- Risk Category: Environmental

- Probability: High (80%)

RMMM Plan:

- Mitigation Strategy: Design the system to be resilient to various weather conditions by using weatherproof materials and testing it under extreme conditions during development.

- Monitoring: Monitor weather forecasts and have mechanisms in place to temporarily disable the system during severe weather events to prevent false detections.

- Management: Develop protocols to recalibrate or reset the system after exposure to extreme weather. Conduct regular checks and maintenance post adverse weather conditions to ensure optimal functionality.

## Risk 5: Budget Overruns (Impact Scale: 3/5)

- Nature of Risk: Unexpected costs exceeding the budget allocated for the project due to unforeseen expenses or resource limitations.

- Risk Category: Financial

- Probability: Moderate (40%)

RMMM Plan:

- Mitigation Strategy: Conduct thorough project planning, identify potential cost areas, and allocate a contingency budget to handle unforeseen expenses.

- Monitoring: Regularly monitor project expenses and track them against the allocated budget. Implement cost controls and approval processes for additional expenditures.

- Management: Establish a financial oversight committee to review project expenditures periodically and make informed decisions on managing costs. Implement strategies to reduce costs without compromising project objectives if needed.

## Risk Table:

| Risk | Risk Category | Probability | Impact (1-5) | RMMM |
|---|---|---|---|---|
| Inaccurate Pothole Detection | Technical | 60% | 4 | Thoroughly test and fine-tune the detection algorithms using diverse datasets and real-world testing. Involve experts to refine the algorithms and incorporate machine learning to improve accuracy over time. |
| Hardware Failure | Technical/ Hardware | 30% | 2 | Use high-quality and redundant hardware components, and conduct regular maintenance to prevent hardware failures. Have a backup system in place to take over in case of a failure. |
| Data Privacy and Security Breach | Security | 50% | 3 | Implement strong encryption protocols, access controls, and regular security audits. Ensure compliance with relevant data privacy laws and standards. |
| Weather Conditions Affecting Performance | Environmental | 80% | 5 | Design the system to be resilient to various weather conditions by using weatherproof materials and testing it under extreme conditions during development |
| Budget Overruns | Financial | 40% | 3 | Conduct thorough project planning, identify potential cost areas, and allocate a contingency budget to handle unforeseen expenses. |

# EXPERIMENT 11

**Aim:** Case study: GitHub for version control

## Theory:

Configuration Management:

Configuration management is a systems engineering process for establishing consistency of a product's attributes throughout its life. In the technology world, configuration management is an IT management process that tracks individual configuration items of an IT system. IT systems are composed of IT assets that vary in granularity. An IT asset may represent a piece of software, or a server, or a cluster of servers. The following focuses on configuration management as it directly applies to IT software assets and software asset CI/CD.

Software configuration management is a systems engineering process that tracks and monitors changes to a software systems configuration metadata. In software development, configuration management is commonly used alongside version control and CI/CD infrastructure.


Configuration management helps engineering teams build robust and stable systems through the use of tools that automatically manage and monitor updates to configuration data. Complex software systems are composed of components that differ in granularity of size and complexity. For a more concrete example consider a microservice architecture. Each service in a microservice architecture uses configuration metadata to register itself and initialize.

Some examples of software configuration metadata are:

- Specifications of computational hardware resource allocations for CPU, RAM, etc.
- Endpoints that specify external connections to other services, databases, or domains
- Secrets like passwords and encryption keys

It's easy for these configuration values to become an afterthought, leading to the configuration to become disorganized and scattered. Imagine numerous post-it notes with passwords and URLs blowing around an office. Configuration management solves this challenge by creating a "source of truth" with a central location for configuration.

Git is a fantastic platform for managing configuration data. Moving configuration data into a Git repository enables version control and the repository to act as a source of truth. Version control also solves another configuration problem: unexpected breaking changes. Managing unexpected changes through the use of code review and version control helps to minimize downtime.

Configuration values will often be added, removed, or modified. Without version control this can cause problems. One team member may tweak a hardware allocation value so that the software runs more efficiently on their personal laptop. When the software is later deployed to a production environment, this new configuration may have a suboptimal effect or may break.

Version control and configuration management solve this problem by adding visibility to configuration modifications. When a change is made to configuration data, the version control system tracks it, which allows team members to review an audit trail of modifications.

Configuration version control enables rollback or "undo" functionality to configuration, which helps avoid unexpected breakage. Version control applied to the configuration can be rapidly reverted to a last known stable state.

## Version control:

Version control, also known as source control, is the practice of tracking and managing changes to software code. Version control systems are software tools that help software teams manage changes to source code over time. As development environments have accelerated, version control systems help software teams work faster and smarter. They are especially useful for DevOps teams since they help them to reduce development time and increase successful deployments.

Version control software keeps track of every modification to the code in a special kind of database. If a mistake is made, developers can turn back the clock and compare earlier versions of the code to help fix the mistake while minimizing disruption to all team members.

For almost all software projects, the source code is like the crown jewels - a precious asset whose value must be protected. For most software teams, the source code is a repository of the invaluable knowledge and understanding about the problem domain that the developers have collected and refined through careful effort. Version control protects source code from both catastrophe and the casual degradation of human error and unintended consequences. Software developers working in teams are continually writing new source code and changing existing source code. The code for a project, app or software component is typically organized in a folder structure or "file tree". One developer on the team may be working on a new feature while another developer fixes an unrelated bug by changing code, each developer may make their changes in several parts of the file tree.
Version control helps teams solve these kinds of problems, tracking every individual change by each contributor and helping prevent concurrent work from conflicting. Changes made in one part of the software can be incompatible with those made by another developer working at the same time. This problem should be discovered and solved in an orderly manner without blocking the work of the rest of the team. Further, in all software development, any change can introduce new bugs on its own and new software can't be trusted until it's tested. So testing and development proceed together until a new version is ready.

Good version control software supports a developer's preferred workflow without imposing one particular way of working. Ideally it also works on any platform, rather than dictate what operating system or tool chain developers must use. Great version control systems facilitate a smooth and continuous flow of changes to the code rather than the frustrating and clumsy mechanism of file locking - giving the green light to one developer at the expense of blocking the progress of others.

Software teams that do not use any form of version control often run into problems like not knowing which changes that have been made are available to users or the creation of incompatible changes between two unrelated pieces of work that must then be painstakingly untangled and reworked. If you're a developer who has never used version control you may have added versions to your files, perhaps with suffixes like "final" or "latest" and then had to later deal with a new final version. Perhaps you've commented out code blocks because you want to disable certain functionality without deleting the code, fearing that there may be a use for it later. Version control is a way out of these problems.

Version control software is an essential part of the every-day of the modern software team's professional practices. Individual software developers who are accustomed to working with a capable version control system in their teams typically recognize the incredible value version control also gives them even on small solo projects. Once accustomed to the powerful benefits of version control systems, many developers wouldn't consider working without it even for nonsoftware projects.

## Benefits of version control systems:

Using version control software is a best practice for high performing software and DevOps teams. Version control also helps developers move faster and allows software teams to preserve efficiency and agility as the team scales to include more developers. Version Control Systems (VCS) have seen great improvements over the past few decades and some are better than others. VCS are sometimes known as SCM (Source Code Management) tools or RCS (Revision Control System). One of the most popular VCS tools in use today is called Git. Git is a Distributed VCS, a category known as DVCS, more on that later. Like many of the most popular VCS systems available today, Git is free and open source. Regardless of what they are called, or which system is used, the primary benefits you should expect from version control are as follows.

1. A complete long-term change history of every file. This means every change made by many individuals over the years. Changes include the creation and deletion of files as well as edits to their contents. Different VCS tools differ on how well they handle renaming and moving of files. This history should also include the author, date and written notes on the purpose of each change. Having the complete history enables going back to previous versions to help in root cause analysis for bugs and it is crucial when needing to fix problems in older versions of software. If the software is being actively worked on, almost everything can be considered an "older version" of the software.

2. Branching and merging. Having team members work concurrently is a no-brainer, but even individuals working on their own can benefit from the

ability to work on independent streams of changes. Creating a "branch" in VCS tools keeps multiple streams of work independent from each other while also providing the facility to merge that work back together, enabling developers to verify that the changes on each branch do not conflict. Many software teams adopt a practice of branching for each feature or perhaps branching for each release, or both. There are many different workflows that teams can choose from when they decide how to make use of branching and merging facilities in VCS.

3. Traceability. Being able to trace each change made to the software and connect it to project management and bug tracking software such as Jira, and being able to annotate each change with a message describing the purpose and intent of the change can help not only with root cause analysis and other forensics. Having the annotated history of the code at your fingertips when you are reading the code, trying to understand what it is doing and why it is so designed can enable developers to make correct and harmonious changes that are in accord with the intended long-term design of the system. This can be especially important for working effectively with legacy code and is crucial in enabling developers to estimate future work with any accuracy.

While it is possible to develop software without using any version control, doing so subjects the project to a huge risk that no professional team would be advised to accept. So the question is not whether to use version control but which version control system to use.

## Github commands:

### Getting & Creating Projects

| Command | Description |
|---|---|
| git init | Initialize a local Git repository |
| git clone ssh://git@github.com/[username]/[repository-name].git | Create a local copy of a remote repository |

### Basic Snapshotting

| Command | Description |
|---|---|
| git status | Check status |
| git add [file-name.txt] | Add a file to the staging area |
| git add -A | Add all new and changed files to the staging area |
| git commit -m "[commit message]" | Commit changes |
| git rm -r [file-name.txt] | Remove a file (or folder) |

### Branching & Merging

| Command | Description |
|---|---|
| git branch | List branches (the asterisk denotes the current branch) |
| git branch -a | List all branches (local and remote) |
| git branch [branch name] | Create a new branch |

| Command | Description |
| --- | --- |
| git branch -d [branch name] | Delete a branch |
| git push origin --delete [branch name] | Delete a remote branch |
| git checkout -b [branch name] | Create a new branch and switch to it |
| git checkout -b [branch name] origin/[branch name] | Clone a remote branch and switch to it |
| git branch -m [old branch name] [new branch name] | Rename a local branch |
| git checkout [branch name] | Switch to a branch |
| git checkout - | Switch to the branch last checked out |
| git checkout -- [file-name.txt] | Discard changes to a file |
| git merge [branch name] | Merge a branch into the active branch |
| git merge [source branch] [target branch] | Merge a branch into a target branch |
| git stash | Stash changes in a dirty working directory |
| git stash clear | Remove all stashed entries |

Sharing & Updating Projects

| Command | Description |
| --- | --- |
| git push origin [branch name] | Push a branch to your remote repository |
| git push -u origin [branch name] | Push changes to remote repository (and remember the branch) |
| git push | Push changes to remote repository (remembered branch) |
| git push origin --delete [branch name] | Delete a remote branch |
| git pull | Update local repository to the newest commit |
| git pull origin [branch name] | Pull changes from remote repository |

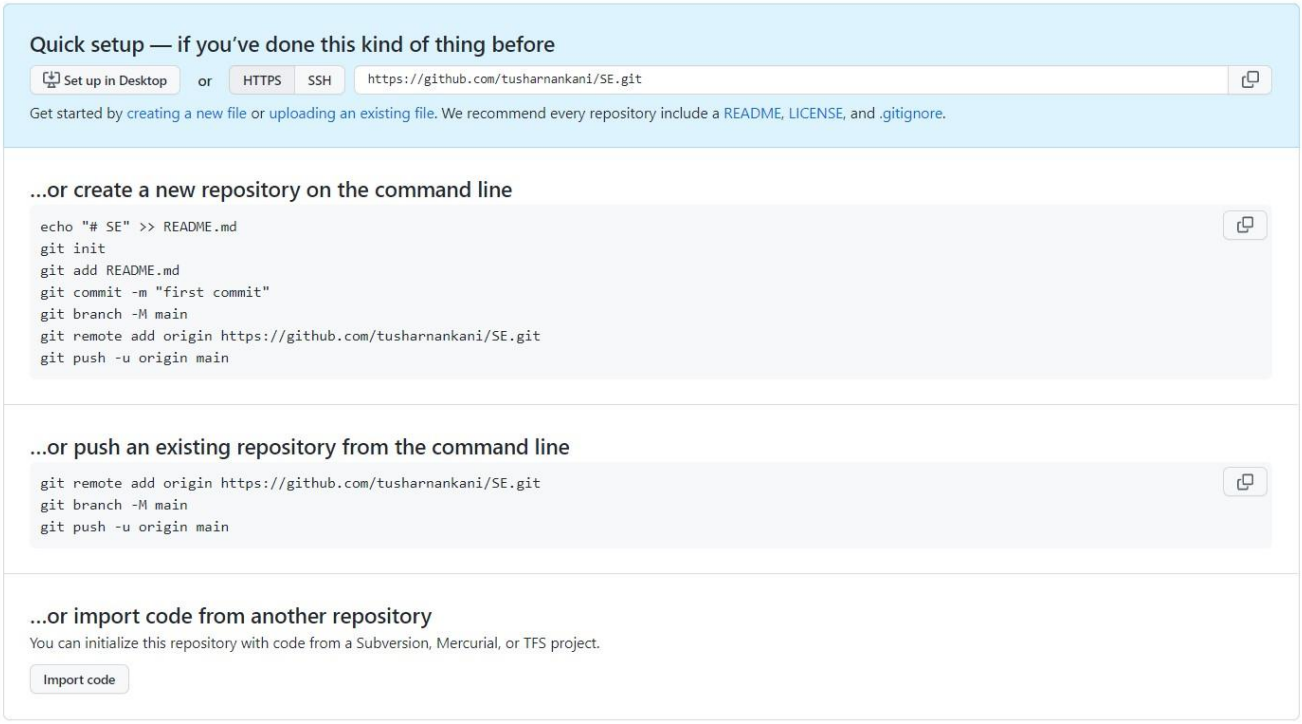
| Command | Description |
| --- | --- |
| git remote add origin ssh://git@github.com/[username]/[repository-name].git | Add a remote repository |
| git remote set-url origin ssh://git@github.com/[username]/[repository-name].git | Set a repository's origin branch to SSH |

Inspection & Comparison

| Command | Description |
| --- | --- |
| git log | View changes |
| git log --summary | View changes (detailed) |
| git log --oneline | View changes (briefly) |
| git diff [source branch] [target branch] | Preview changes before merging |

OUTPUT:

1. Creating a repository




2. Initial Commit

```
C:\Users\Tushar Nankani\Desktop\COLLEGE\SEM 5\SE>mkdir SE && cd SE

C:\Users\Tushar Nankani\Desktop\COLLEGE\SEM 5\SE\SE>git init
Initialized empty Git repository in C:/Users/Tushar Nankani/Desktop/COLLEGE/SEM 5/SE/SE/.git/

C:\Users\Tushar Nankani\Desktop\COLLEGE\SEM 5\SE\SE>git add -A

C:\Users\Tushar Nankani\Desktop\COLLEGE\SEM 5\SE\SE>git commit -m "first commit"
[master (root-commit) b11dbfd] first commit
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 Exp1.docx

C:\Users\Tushar Nankani\Desktop\COLLEGE\SEM 5\SE\SE>git remote add origin https://github.com/tusharnankani/SE.git
```

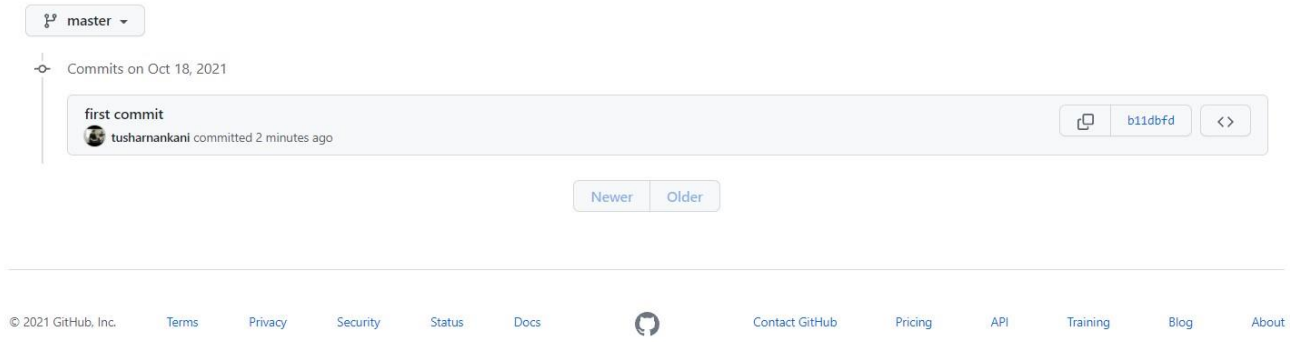
```
C:\Users\Tushar Nankani\Desktop\COLLEGE\SEM 5\SE\SE>git push -u origin master
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 10.65 KiB | 10.65 MiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/tusharnankani/SE.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.

C:\Users\Tushar Nankani\Desktop\COLLEGE\SEM 5\SE\SE>
```

3. Commit history

```
C:\Users\Tushar Nankani\Desktop\COLLEGE\SEM 5\SE\SE>git log
commit b11dbfd95e69bf67a119cae1ffd0537d93ff4f2e (HEAD -> master, origin/master)
Author: tusharnankani <tusharnankani3@gmail.com>
Date:   Mon Oct 18 03:01:29 2021 +0530

    first commit
```



4. Git commands

```
C:\Users\Tushar Nankani\Desktop\COLLEGE\SEM 5\SE\SE>git --help
usage: git [--version] [--help] [-C <path>] [-c <name>=<value>]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p | --paginate | -P | --no-pager] [--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           <command> [<args>]

These are common Git commands used in various situations:

start a working area (see also: git help tutorial)
   clone             Clone a repository into a new directory
   init              Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)
   add               Add file contents to the index
   mv                Move or rename a file, a directory, or a symlink
   restore           Restore working tree files
   rm                Remove files from the working tree and from the index
   sparse-checkout   Initialize and modify the sparse-checkout

examine the history and state (see also: git help revisions)
   bisect            Use binary search to find the commit that introduced a bug
   diff              Show changes between commits, commit and working tree, etc
   grep              Print lines matching a pattern
   log               Show commit logs
   show              Show various types of objects
   status            Show the working tree status

grow, mark and tweak your common history
   branch            List, create, or delete branches
   commit            Record changes to the repository
   merge             Join two or more development histories together
   rebase            Reapply commits on top of another base tip
   reset             Reset current HEAD to the specified state
   switch            Switch branches
   tag               Create, list, delete or verify a tag object signed with GPG

collaborate (see also: git help workflows)
   fetch             Download objects and refs from another repository
   pull              Fetch from and integrate with another repository or a local branch
   push              Update remote refs along with associated objects
```

# EXPERIMENT – 12

**AIM:** Develop test cases for the project using White Box Testing (JUnit)

## THEORY:

White box testing is a software testing technique that examines the internal structure and logic of an application's code. This type of testing is also known as clear box testing, glass box testing, or structural testing. White box testing focuses on validating the correctness of the code, the flow of data, and the behavior of various components within the software. It is typically performed by software developers or dedicated quality assurance (QA) professionals with knowledge of the codebase.

**Objective**: The main objective of white box testing is to ensure that the code functions correctly, adheres to design specifications, and is free from logic errors, vulnerabilities, and defects. It aims to assess the quality of the code, its efficiency, and its ability to handle various inputs and scenarios.

**Scope**: White box testing primarily focuses on the internal workings of the software. Testers have access to the source code, database schema, and system architecture to design test cases.

**Techniques**: There are various techniques used in white box testing:

a. Statement Coverage: This technique ensures that every line of code is executed at least once during testing.

b. Branch Coverage: It verifies that every branch of conditional statements is executed, including both the true and false conditions.

c. Path Coverage: This technique checks every possible path through the code to identify any logical issues or dead code.

d. Cyclomatic Complexity: This metric calculates the complexity of the code based on the number of independent paths through the code, helping in the identification of complex areas that might require more extensive testing.

e. Data Flow Analysis: Examines how data is used and propagated within the code, looking for potential issues like uninitialized variables or variable overflows.

**Testing Levels**:

Unit Testing: This is the lowest level of white box testing and focuses on individual functions or methods within the code.

Integration Testing: This level verifies the interactions between different modules or components, ensuring they work together correctly.

System Testing: At this level, the entire system is tested to confirm that it meets the overall requirements and functions as expected.

Tools: White box testing often employs various automated testing tools and frameworks that assist in code analysis, coverage measurement, and defect detection. Some popular tools include JUnit, NUnit, SonarQube, and CodeSonar.

**Challenges**:

White box testing can be time-consuming and may require in-depth knowledge of the codebase.

Testers may need to continuously update test cases as the code evolves.

It may not guarantee the absence of all defects, as some issues may still be overlooked.

**Advantages**:

It provides in-depth insight into the code's quality and reliability.

Helps identify issues early in the development process, reducing debugging efforts later.

Can uncover hidden defects and vulnerabilities.

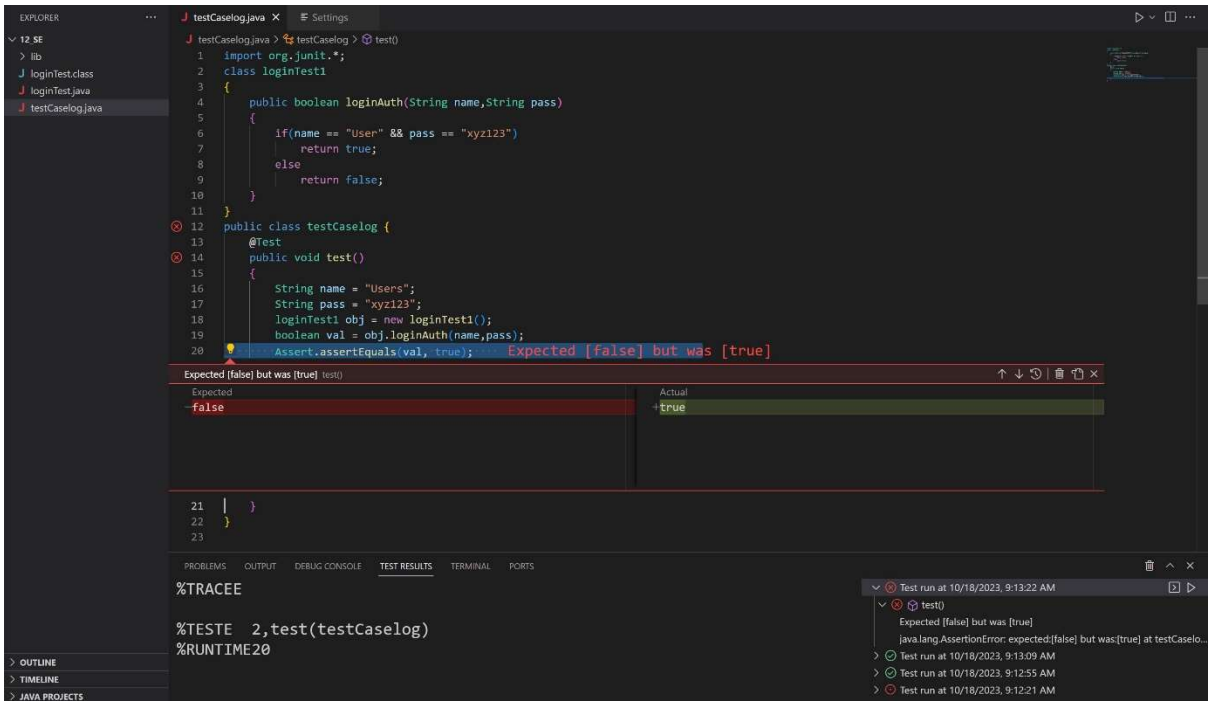
**Limitations:**

It may not guarantee that the software behaves correctly in all real-world scenarios.

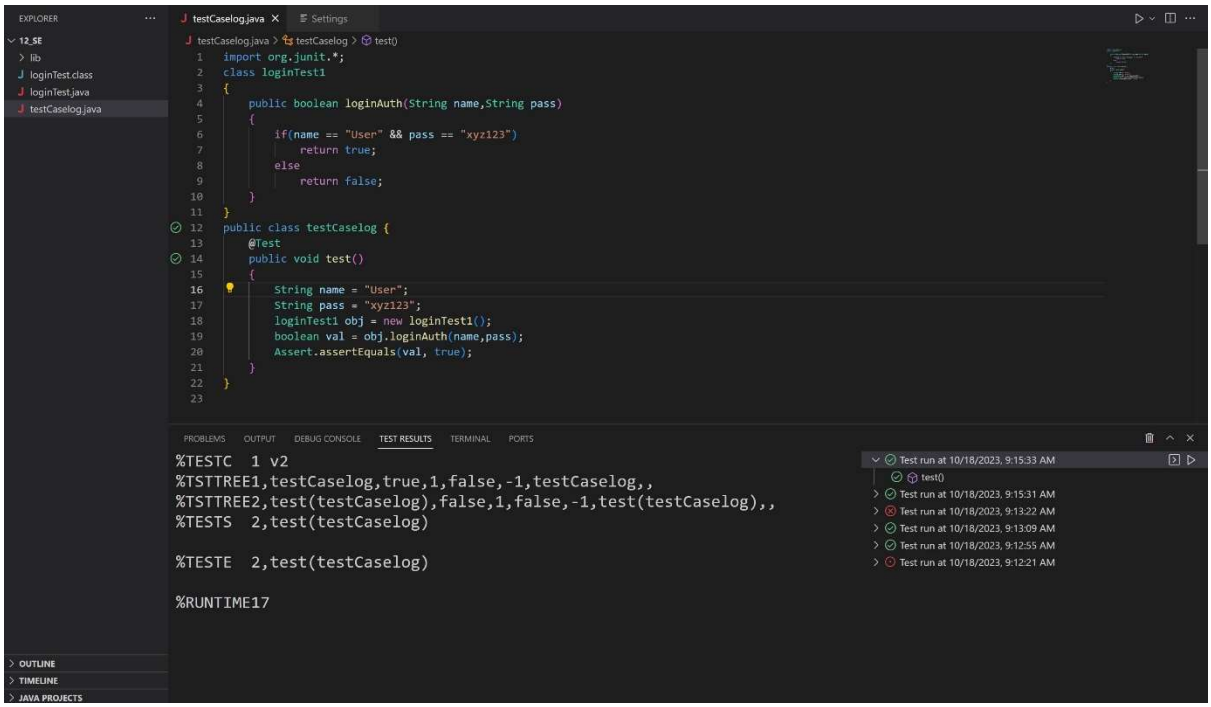
Testers might be biased by their knowledge of the code, potentially missing issues a user might encounter.

## TEST CASE 1: CHECK FOR VALID USER CREDENTIALS

### Invalid User Test Case

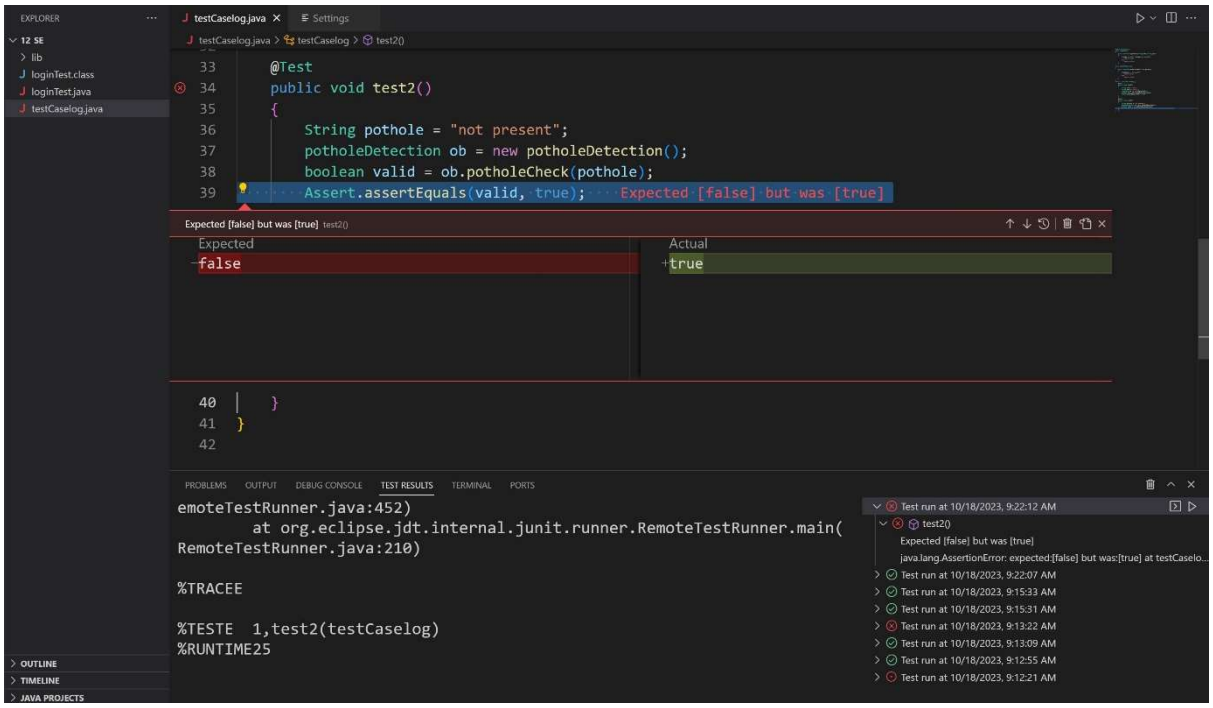



### Valid User Test Case

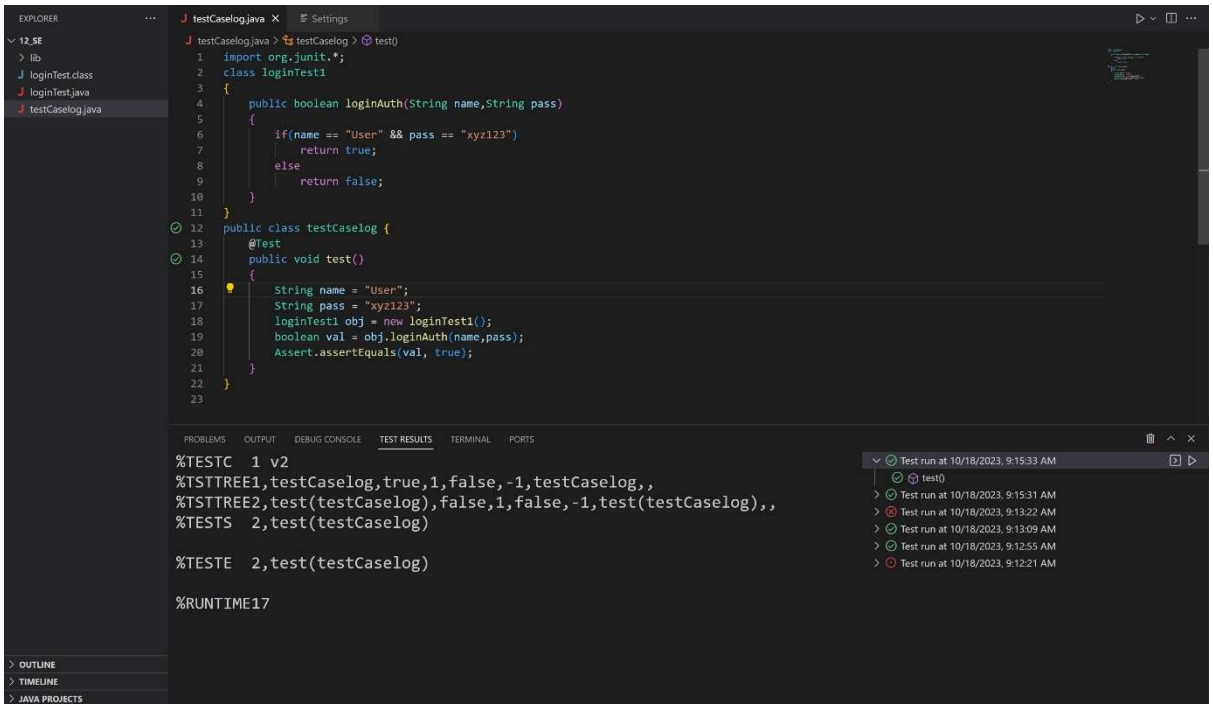


## TEST CASE 2: CHECK WHETHER POTHOLE DETECTED OR NOT

## Pothole not detected




## Pothole Detected



Parth D.
32
C12

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

# ASSIGNMENT NO:1

Q] Architectural Design: Explain all the types with examples.

⇒ Software architecture suggests "the overall structure of the software and the ways in which that structure provides conceptual integrity for a system."
Architectural design is the backbone of any software design and it is responsible for the overall structure of the system. In nearly all the models of software process, architectural design is considered as the first stage in the software design and development process. The output of the architectural design process is an architectural model that explains the overall structure of the system and also explains the working of the system and the communication among various components of the system.

No. of different models can be used to represent architecture

(1) Structural Model: Represent architecture as an organized collection of components.

(2) Framework model: Increase level of design abstraction by identifying repeatable architectural design framework.

(3) Dynamic model: Addresses behaviour of the program architecture and indicating how the system or structure configuration may change as a function.

(4) Process Model: focus on design of the business or

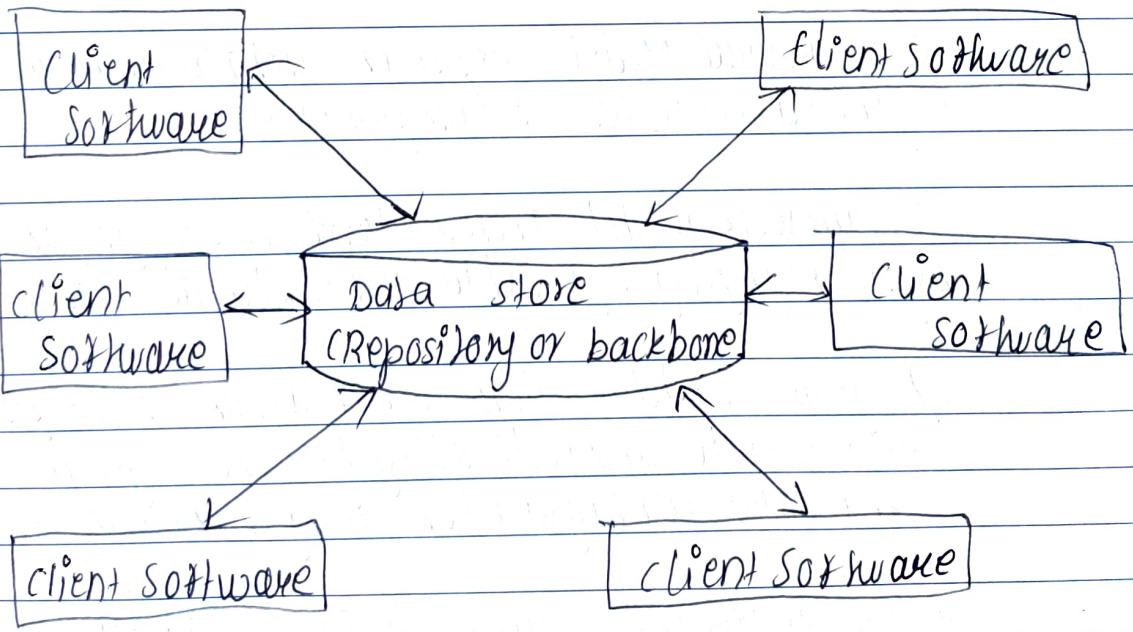
technical process that the system must accommodate.

(e) Functional model: used to represent the functional hierarchy of the system.

Style describes a system category that encompasses:
- A set of components that perform a function required by a system.
- A set of connectors that enable communication cordinations and co-operations among components.
- Constraints that define how components can be integrated to form the system.
- semantic models that enable designer to understand the overall properties of the system.

It can be represented by:

(A) Data Centred Architecture:

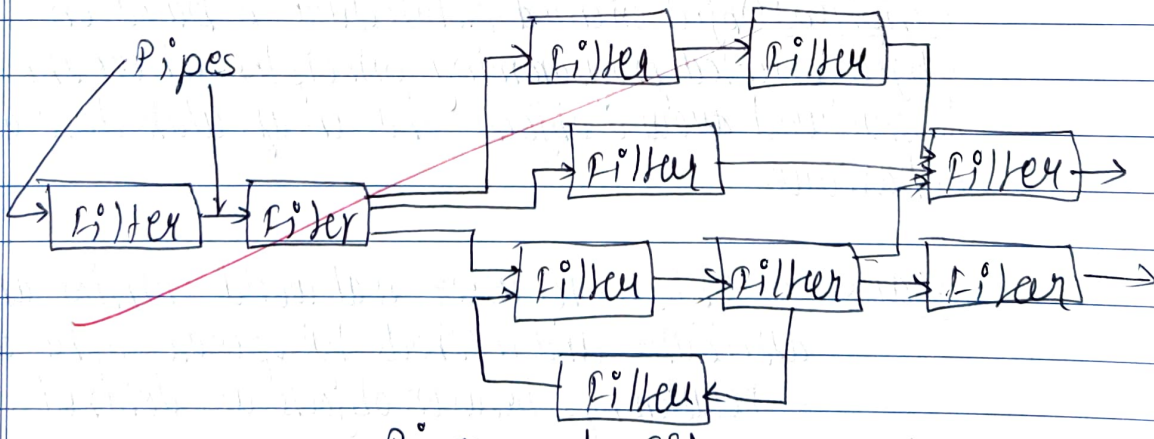


Description: Data-Centred Architecture places a strong emphasis on how data is organized, stored and accessed within a system. It often revolves around a central data repository or data base.

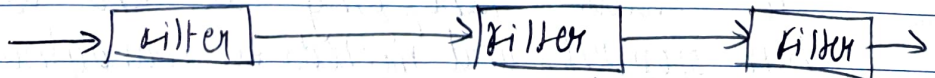
Details: - Data is considered the core element of the system.
- Components within the system interact with the data repository to read or write data.
- It's common to use database with structured schemas, like SQL databases, to manage the data

Example: A Customer Relationship Management (CRM) system that stores customer information, interactions and transactions in a central database and allows various modules to access and manipulate this data.

(B) Data Flow Architecture:




Pipes and Filters



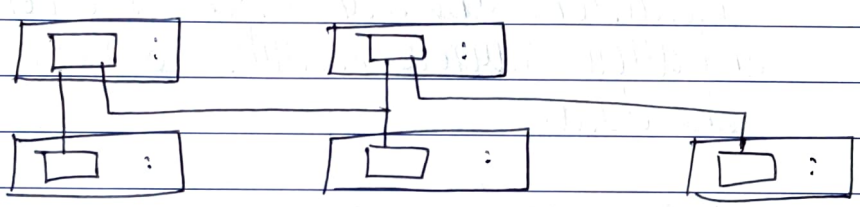

Batch sequential

Description: Data flow architecture focusses on movement and transformation of data as it flows through various processing steps within a system.

Details:- It models how data is acquired, processed and delivered to its destination.
- Typically used in data processing applications, data pipelines or system involving data transformations.

Example:- A real time stock trading system where data is continuosly flowing in being analyzed for patterns, and then being used to trigger buy/sell decision

(c) Object oriented Architecture:



Description: Object oriented Architecture is based on object oriented programming principles, where the system is structured around objects that encapsulate data and behaviour.

Details:- Objects represent real-world entities and have attributes and methods that operate on that data.
- Relationships between objects are defined.

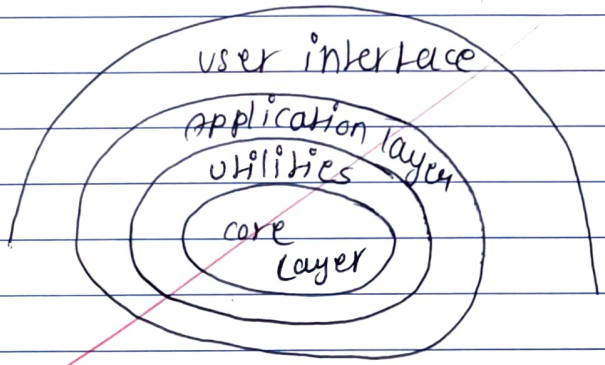
Example: An e-commerce platform where you have objects like "Product", "Shopping Cart" and "User", each with their properties and methods for managing products, adding to carts and processing orders.

## (D) Layered Architecture :

Description : Layered architecture divides the system into distinct layers, each with its own specific role and responsibility.

Details :- Communication typically flows in a vertical manner, from one layer to another layer.
- Layers abstract and seperate concerns, making the system more modular and maintainable.

Example : A software application with a presentation layer, buisness logic layer and data access layer, where each layer has a well defined purpose.



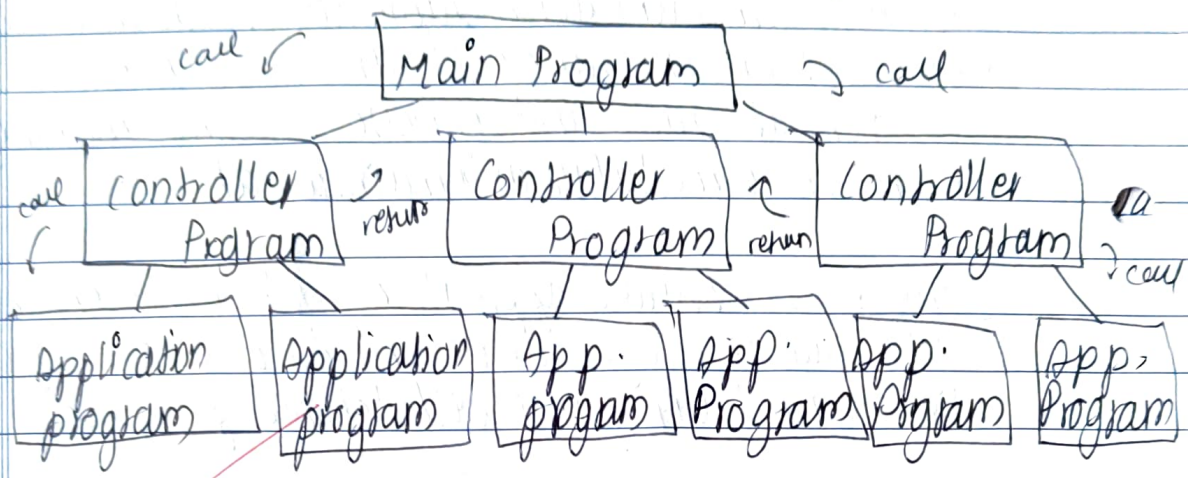

## (E) Call and return architecture :

Call and return architecture, also known as procedural or imperative architecture, focusses on controlling the flow of execution through a series of procedure or function calls.

Details :- It's characterized by a linear flow of control, where functions are called and return results.
- It's commonly used in procedural programming

languages like C.

Example: A simple command line calculator program where you have functions for addition, subtraction, multiplication and division and the program executes these functions based on the user input.



Parth D.
32
C12

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

# Assignment no: 2

Q] Explain in detail:

a) Software Maintainance:

The software maintainance is an activity that usually begins after the software product delivered at the client's end. In software maintainance, the modifications are carried out or the updates in the software are taken place. In software maintainance phase, no major changes are implemented. In software maintainance, the changes are done in the existing program or the some small new functionality is added. Three decades ago, software maintainance was given the name as iceberg, where the potential problems reside inside and it is not visible to the clients. If maintainance is not done properly, it will sink the entire ship, as icebergs do. The maintainance of the software involves more than 60% of all the efforts taken by the development team in developing the actual product.

Types of software maintainance:

(1) Corrective maintainance
(2) Adaptive maintainance
(3) Perfective maintainance
(4) Preventive maintainance.

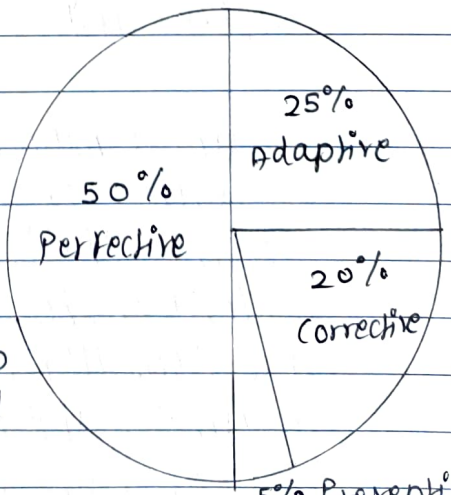


Fig 1.

Fig. 1 Illustrates the distribution of activities into above mentioned types of maintainance.

(a) Corrective maintainance : The corrective maintainance is a type of maintainance in which the errors are fixed when it is observed during the use of software. In this the errors may be caused due to the lacity software design, incorrect logic and improper coding. All these errors are called as residual errors and they prevent the final specification. In case of system failure, the corrective maintainance approach is initiated to locate the original specification. Corrective maintainance normally used for more than 20% of all the maintainance activities.

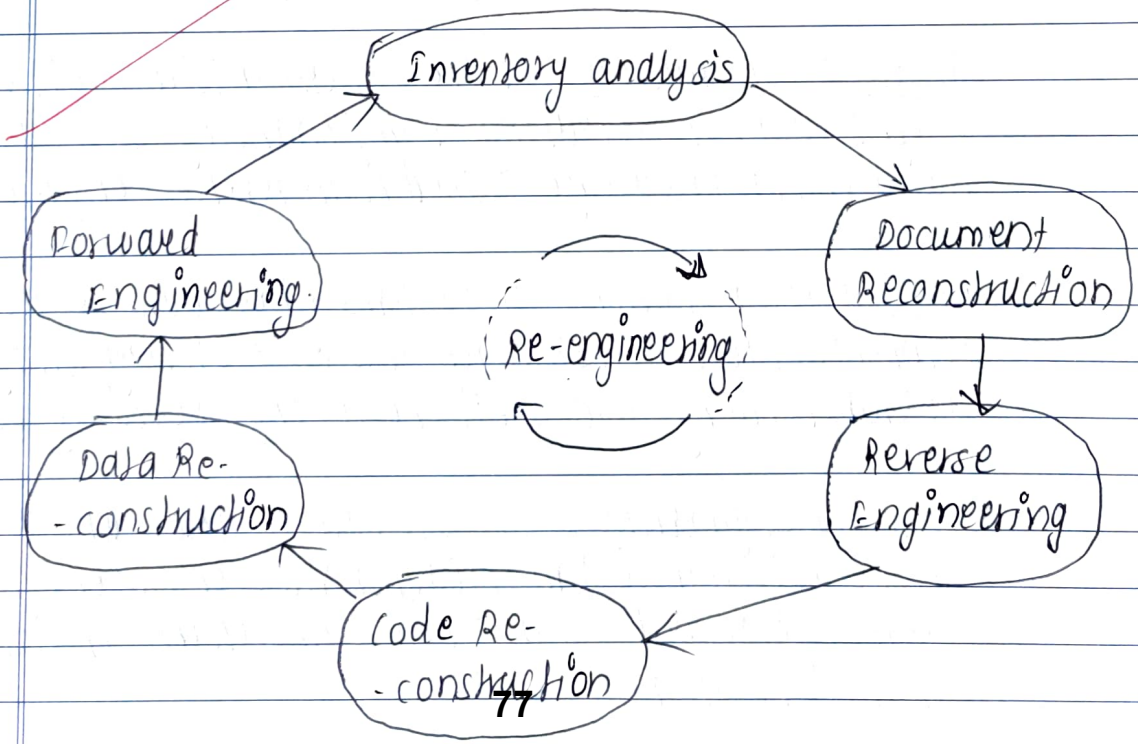
(b) Adaptive maintainance: The adaptive maintainance means the implementation of modification of the system. The changes may be of hardware or the OS environment. Adaptive maintainance normally used for more than 25% of all the maintainance activities.

(c) Perfective maintainance: Perfective maintainance deals with the modified and changed user requirements. The functional enhancements are taken into consideration in Perfective maintainance. The function and efficiency of the code is continuosly improved. Perfective maintainance normally used for more than 50% of all maintainance activities and it is the largest among all.

(d) Preventive maintainance: Preventive maintainance is used to prevent the possible errors to occur. Thus, in the activity, the complexity is minimized and the quality of the program is enhanced. Adaptive maintainance normally used for 5% of all the maintainance activities and it is the smallest among all the types of maintainance activities.

b) Re-engineering:

Software re-engineering is a process of software development which is done to improve the maintainability of a software system. Re-engineering is the examination and alteration of the system to reconstitute it in a new form. This process encompasses a combination of subprocesses like reverse engineering, forward engineering, reconstructing etc. Software re-engineering, also known as software reconstructing or software renowation, refers to the process of improving or upgrading existing software systems to improve their quality, maintainability or functionality. It involves reusing the existing software artifacts, such as code, design and documentation, and transforming them to meet the new or updated requirements. The primary goal of software re-engineering is to improve the quality and maintainability of the software system, while minimizing the risks and costs associated with the redevelopment of the system from scratch.



The steps involved in software re-engineering are:

(a) Inventory Analysis: This step involves creating a comprehensive inventory of all components, assets and resources within the existing system. It helps in understanding what you have before making any changes.

(b) Document Re-construction: In this step, existing documentation such as system application, user manuals and source code documentation, is reconstructed and updated. This is essential for understanding how the system works and what it is supposed to do.

(c) Reverse Engineering: Reverse engineering involves examining the existing system or software to understand its design, structure and functionality. This is often done by analyzing binaries, source code, or even physical components. It helps in gaining insights into the system's inner workings.

(d) Code Reconstruction: If the system involves software, this step focuses on recreating or implementing the source code based on the knowledge gained from reverse engineering.

(e) Data Reconstruction: In systems where data is a significant component, data reconstruction involve recovering, migrating or transforming data to fit the re-engineered system.

(f) Forward Engineering: once you have a clear understanding of the existing system through reverse engineering, you can move forward to design and implement improvements or modifications.

c) Reverse Engineering:

The reverse engineering is the discipline of software engineering, where the knowledge and design information is extracted from the source code or it is reproduced. The reverse engineering is a process where the system is analyzed at higher level of abstraction. It is also called as going backward through all the development cycles. Following are some important purposes of reverse engineering:-

(i) Security auditing
(ii) Enabling additional features.
(iii) Used as learning tools
(iv) Develop compatible products.

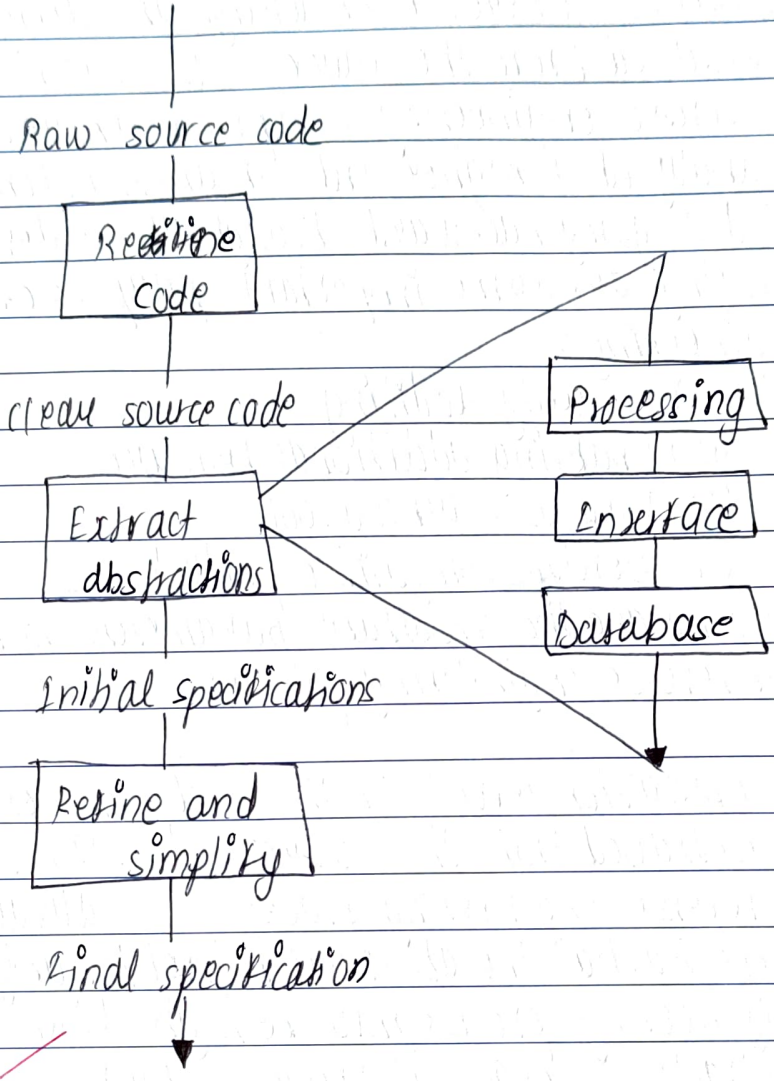
Following are the important parameters to be considered for reverse engineering process:

(a) Abstraction level: In this level, the design information is extracted from the source code. It is the highest level in reverse engineering process. It is always ~~accepted~~ expected that the abstraction level for any reverse engineering process must be high. When the level of abstraction is high, it becomes easy for the software developer to understand the program.

(b) Completeness: These are the details available through the abstraction level of reverse engineering process. For example from the given process source code it is very easy to develop the complete procedural design.

(c) Directionality: It is a method of extracting information from the source code and making it available to software

developers. The software developers can use this information during the maintainance activities.




1) Raw or dirty source code obtained from the abstraction level taken as input.
2) This code is refined or reconstructed.
3) From clean code, abstraction is extracted.
4) From this initial specification, the code is refined and simplified.
5) Now, we get the final specification.