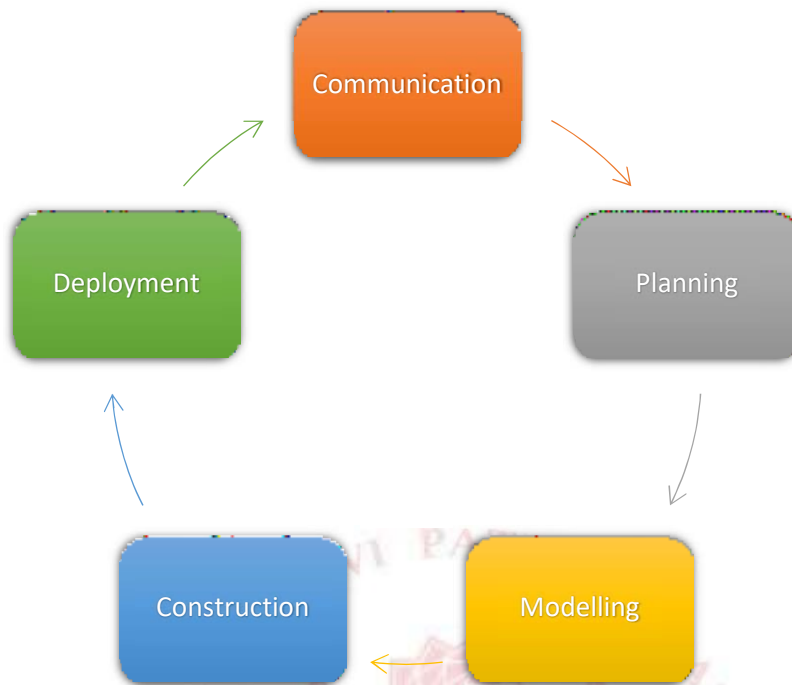


PRACTICAL-01

- ❖ **Aim:** - Study of Software Development Life Cycle (SDLC) & analysing various activities conducted at various phases.
- ❖ **Objective:** - Learning about Software development life cycle and its phases along with different models of SDLC.
- **SDLC:**
 - SDLC known as Software Development Life Cycle is a process used by the software industry to design, develop and test high quality software. The SDLC aims to produce a high-quality software that meets or exceeds customer expectations, reaches completion within times and cost estimates.
 - SDLC is a framework defining tasks performed at each step in the software development process.
- **SDLC is Required For:**
 - The SDLC is important because it helps ensure that the right people are involved in the right activities at the right times. Using a structured approach to developing software helps ensure that your project will be successful.
 - **Some of the SDLC's Benefits Are:**
 - Understanding your requirements and the goal of the software
 - Identify risks at an early stage
 - Plan how you will deliver your solution in stages, such as building prototypes or writing functional specifications
 - Measure your progress relative to your goals and ensure everything is on track
- **There are 5 Stages in Software Development Life Cycle:**
 1. Communication
 2. Planning
 3. Modelling
 4. Construction
 5. Deployment



➤ **STAGE 1: - COMMUNICATION:**

- Communication refers to the exchange of information, ideas, and feedback between various stakeholders involved in the software development process. Effective communication is vital throughout the entire SDLC to ensure a successful and well-coordinated development effort.
- It is the first and the foremost thing for the development of the software. Communication is necessary to know the actual demand of the client.

➤ **STAGE 2: - PLANNING:**

- In the context of SDLC (Software Development Life Cycle), planning is a crucial phase that occurs early in the software development process. It involves defining the project's objectives, scope, timeline, resources, and deliverables. The planning phase lays the foundation for the entire development effort and ensures that the project is set up for success.
- It basically means drawing a map for reducing the complication of development

➤ **STAGE 3: - MODELING:**

- Modelling refers to the process of creating visual representations or abstractions of the software system being developed. Models are used to understand, analyse, and document various aspects of the

SOFTWARE ENGINEERING

Subject Code: 21CS2506

software, including its structure, behaviour, and interactions. Modelling plays a crucial role in the design and development phases of the SDLC.

- It is basically that is created according to the client for better understanding.

➤ **STAGE 4: - CONSTRUCTION:**

- Construction refers to the phase of the development process where the actual coding and implementation of the software take place. It is the third phase in the SDLC, following the requirements gathering and analysis phase and the design phase. The construction phase is focused on transforming the design specifications into working software that meets the specified requirements. During this phase, developers write code, create software components, and integrate them to build the final product.
- It basically includes the testing and the coding of the problem.

➤ **STAGE 5: - DEPLOYMENT:**

- Deployment is the final phase of the development process, where the software is released and made available for use by end-users or customers. It involves the actual implementation of the software in the production environment, making it accessible to the intended users. Deployment marks the transition from development to the operational phase of the software.
- It includes the delivery of the software to the client for evaluation and feedback.

❖ **SDLC MODELS:**

- The Software Development Life Cycle models are basically classified into two types.

- **Evolutionary Models**
- **Conventional Models**

❖ **EVOLUTIONARY MODEL:**

- Evolutionary models in SDLC (Software Development Life Cycle) are iterative and incremental approaches to software development that emphasize continuous feedback, refinement, and evolution of the software product. Unlike traditional linear models, where development proceeds in a sequential manner, evolutionary models allow for the development of a basic version of the software early on, followed by the addition of new features and enhancements through multiple iterations. This allows for greater flexibility, adaptability, and responsiveness to changing requirements and customer feedback.

- They include **Incremental model, Component Based Processed model, Unified Process model.**

❖ CONVENTIONAL MODEL:

- Conventional models in SDLC (Software Development Life Cycle) are traditional, linear approaches to software development that follow a sequential and structured process. These models are often referred to as "waterfall" models because development flows in a single direction, with each phase dependent on the completion of the previous one. Conventional models are characterized by well- defined phases and a fixed scope, making them suitable for projects with stable and clear requirements.
- They include **Waterfall model, prototype model and RAD model.**

1. Incremental Model

- Incremental Model is a process of software development where requirements divided into multiple standalone modules of the software development cycle. In this model, each module goes through the requirements, design, implementation and testing phases. Every subsequent release of the module adds function to the previous release. The process continues until the complete system achieved.

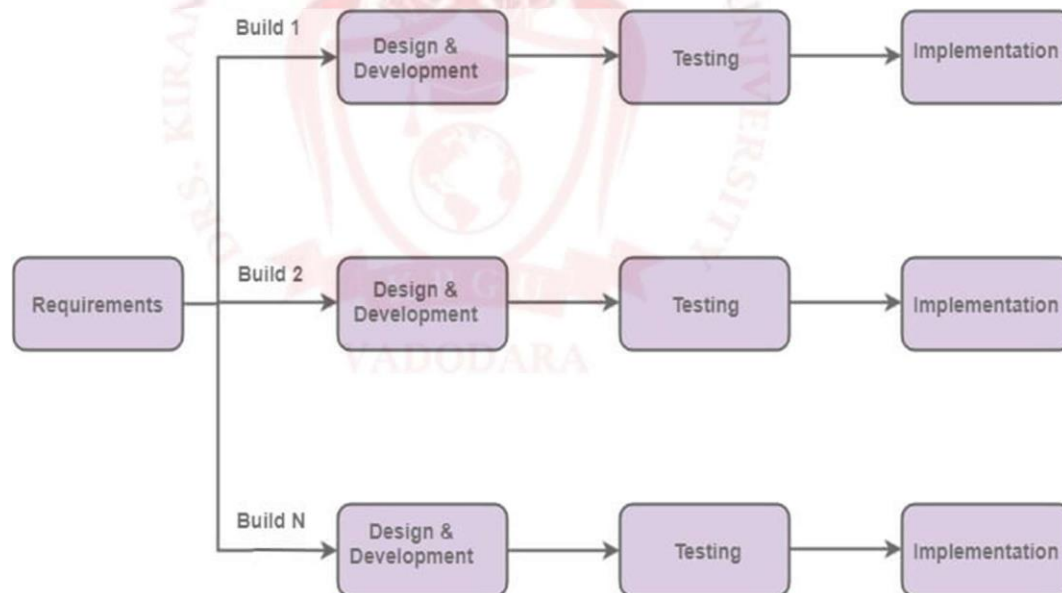


Fig: Incremental Model

❖ ADVANTAGES:

- Errors are easy to be recognized.

- Easier to test and debug
- More flexible.
- Simple to manage risk because it handled during its iteration.
- The Client gets important functionality early.

❖ **DISADVANTAGES:**

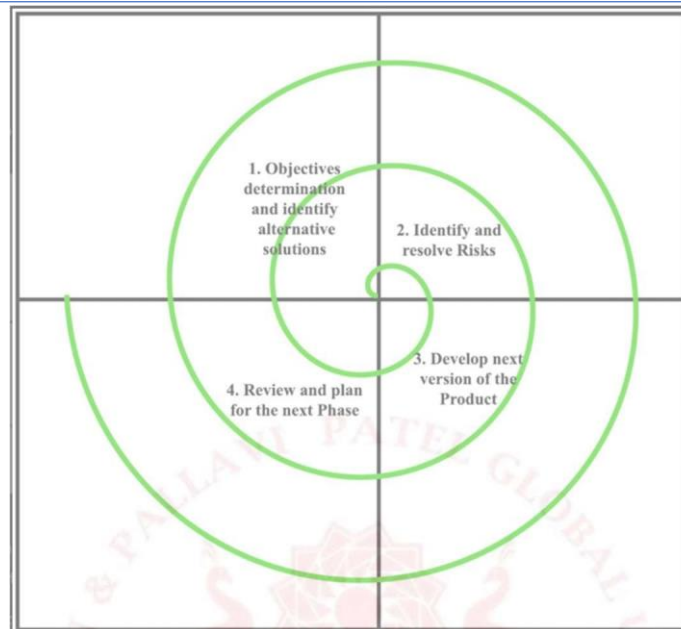
- Need for good planning
- Total Cost is high.
- Well defined module interfaces are needed.

❖ **When we use the Incremental Model?**

- When the requirements are superior.
- A project has a lengthy development schedule.
- When Software team are not very well skilled or trained.
- When the customer demands a quick release of the product.
- You can develop prioritized requirements first.

2. Spiral Model

- Spiral Model is one of the most important Software Development Life Cycle models, which provides support for Risk Handling. In its diagrammatic representation, it looks like a spiral with many loops. The exact number of loops of the spiral is unknown and can vary from project to project. Each loop of the spiral is called a Phase of the software development process. The exact number of phases needed to develop the product can be varied by the project manager depending upon the project risks. As the project manager dynamically determines the number of phases, so the project manager has an important role to develop a product using the spiral model.
- The Spiral Model is a software development life cycle (SDLC) model that provides a systematic and iterative approach to software development. It is based on the idea of a spiral, with each iteration of the spiral representing a complete software development cycle, from requirements gathering and analysis to design, implementation, testing, and maintenance.



❖ ADVANTAGES: -

- **Risk Handling:** The projects with many unknown risks that occur as the development proceeds, in that case, Spiral Model is the best development model to follow due to the risk analysis and risk handling at every phase.
- **Flexibility in Requirements:** Change requests in the Requirements at later phase can be incorporated accurately by using this model.
- **Iterative and Incremental Approach:** The Spiral Model provides an iterative and incremental approach to software development, allowing for flexibility and adaptability in response to changing requirements or unexpected events.
- **Improved Quality:** The Spiral Model allows for multiple iterations of the software development process, which can result in improved software quality and reliability
- **Improved Communication:** The Spiral Model provides for regular evaluations and reviews, which can improve communication between the customer and the development team.

❖ DISADVANTAGES: -

- **Complex:** The Spiral Model is much more complex than other SDLC models.
- **Expensive:** Spiral Model is not suitable for small projects as it is expensive.

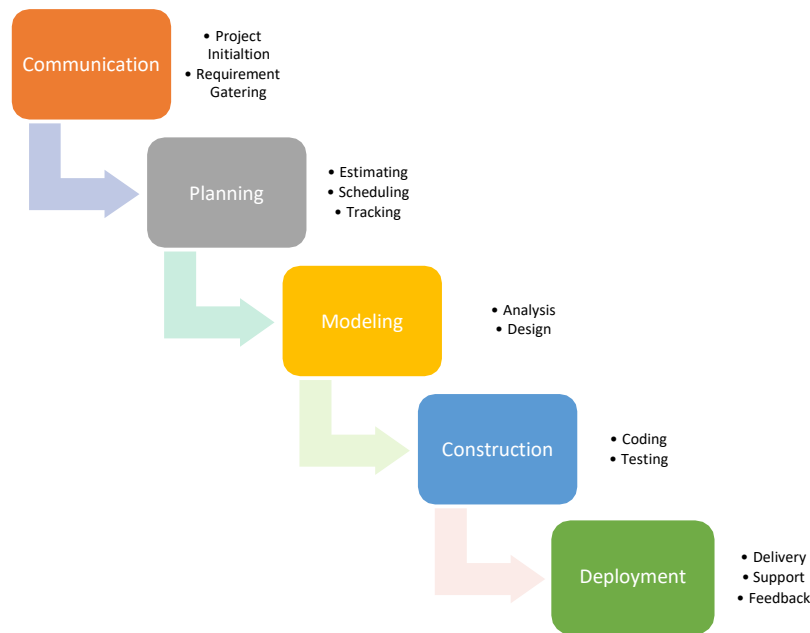
- **Too much dependability on Risk Analysis:** The successful completion of the project is very much dependent on Risk Analysis. Without very highly experienced experts, it is going to be a failure to develop a project using this model.
- **Difficulty in time management:** As the number of phases is unknown at the start of the project, so time estimation is very difficult.
- **Complexity:** The Spiral Model can be complex, as it involves multiple iterations of the software development process.

❖ **When to use spiral Model: -**

- When a project is vast in software engineering, a spiral model is utilised.
- A spiral approach is utilised when frequent releases are necessary.
- When it is appropriate to create a prototype
- When evaluating risks and costs is crucial
- The spiral approach is beneficial for projects with moderate to high risk.
- The SDLC's spiral model is helpful when requirements are complicated and ambiguous.
- If modifications are possible at any moment
- When committing to a long-term project is impractical owing to shifting economic priorities.

3. **Waterfall Model:**

- The Waterfall Model is one of the earliest and most well-known conventional models used in SDLC (Software Development Life Cycle). It is a linear and sequential approach that divides the software development process into distinct phases. Each phase must be completed before moving on to the next, and there is minimal opportunity for revisiting previous stages once they are completed. The model is called "waterfall" because progress flows in one direction, like a waterfall, from the top (requirements) to the bottom (maintenance).

**❖ ADVANTAGES: -**

- **Predictability:** The linear and well-defined nature of the model makes it predictable, as each phase has specific deliverables and outcomes.
- **Clear Documentation:** Each phase is thoroughly documented, providing a clear record of decisions, design, and requirements.
- **Suitable for Small Projects:** The Waterfall Model works well for small projects with well-defined and stable requirements.
- **Control and Management:** It allows for better control and management of the development process.

❖ DISADVANTAGES: -

- **Inflexible to Changes:** The model is not well-suited to handle changes in requirements, and making changes after a phase is completed can be costly and time-consuming.
- **Limited Customer Involvement:** Customers may not have significant involvement until the later stages, which can lead to a product that does not fully meet their needs.
- **Late Detection of Defects:** Defects may not be detected until the testing phase, which can lead to issues in the final product.
- **Long Delivery Time:** The linear approach may result in longer development times, especially for larger projects.

SOFTWARE ENGINEERING

Subject Code: 21CS2506

❖ When to use Waterfall Model: -

- The Waterfall Model is most appropriate to use under certain specific circumstances and project characteristics. It is suitable when:
 - 1) Clear and Well-Defined Requirements: The project requirements are well understood, stable, and unlikely to change significantly during development. The Waterfall Model works best when the scope is well-defined from the beginning.
 - 2) Small and Simple Projects: The model is ideal for small and straightforward projects with a limited number of features and functionalities.
 - 3) Fixed Timeframe and Budget: The project has a fixed timeframe and budget, and there is no room for flexibility in terms of scope or schedule.
 - 4) Limited Customer Involvement: The customer or end-users do not need to be extensively involved during development and are content with providing requirements upfront.
 - 5) Strict Documentation Requirements: The project requires comprehensive documentation for each phase and deliverable.
 - 6) Regulatory or Compliance Constraints: When the project requires strict adherence to regulatory standards, the Waterfall Model can help ensure thorough documentation and verification of compliance.
 - 7) Low Technical Risk: The technology and tools required for the project are well-established, and there is minimal technical risk involved.
 - 8) Sequential Dependencies: The project's tasks have a clear sequence of dependencies, where each phase builds upon the completion of the previous one.

PRACTICAL-2

❖ **Aim:** - Compare and contrast all the process model in tabular view

❖ **Objective:** - To compare all the parameters with different models of software

| Parameter | Waterfall Model | Incremental Model | Prototype Model | RAD Model | Spiral Model | Agile Model | XP Programming |
|---------------------------------|------------------|-------------------|-----------------|---------------|-----------------|----------------------|----------------|
| Clear Requirement Specification | Initial Level | Initial Level | At Medium Level | Initial Level | Initial Level | Change Incrementally | Initial Level |
| Speed to Change | Low | Medium | High | Low | Medium | High | High |
| Feedback from User | No | No | Yes | No | No | No | Yes |
| Risk Identification | At Initial Level | No | No | Yes | Yes | Yes | Yes |
| Predictability | Low | Low | High | Low | Medium | High | High |
| Any Variation Done | Yes V-Model | No | No | No | Yes, Win Spiral | No | No |
| Usability | Basic | Medium | High | Medium | Medium | Most use now a Days | Medium |
| Customer Priority | Nil | Nil | Intermediate | Nil | Intermediate | High | Intermediate |
| Industry Approach | Basic | Basic | Medium | Medium | Medium | High | Medium |
| Cost | Low | Low | High | Very High | Expensive | Much Expensive | High |
| Elasticity | No | No | Yes | Yes | No | Very High | Medium |
| Resource Organization | Yes | Yes | Yes | Yes | No | No | Yes |

PRACTICAL-3

Aim: Decide Software Process Model for your own project. Brief about process model and justify selection of process model for your project.

Description:

The meaning of Agile is swift or versatile. “**Agile process model**” refers to a software development approach based on iterative development. Agile methods break tasks into smaller iterations, or parts do not directly involve long term planning. The project scope and requirements are laid down at the beginning of the development process. Plans regarding the number of iterations, the duration and the scope of each iteration are clearly defined in advance.

Each iteration is considered as a short time "frame" in the Agile process model, which typically lasts from one to four weeks. The division of the entire project into smaller parts helps to minimize the project risk and to reduce the overall project delivery time requirements. Each iteration involves a team working through a full software development life cycle including planning, requirements analysis, design, coding, and testing before a working product is demonstrated to the client.

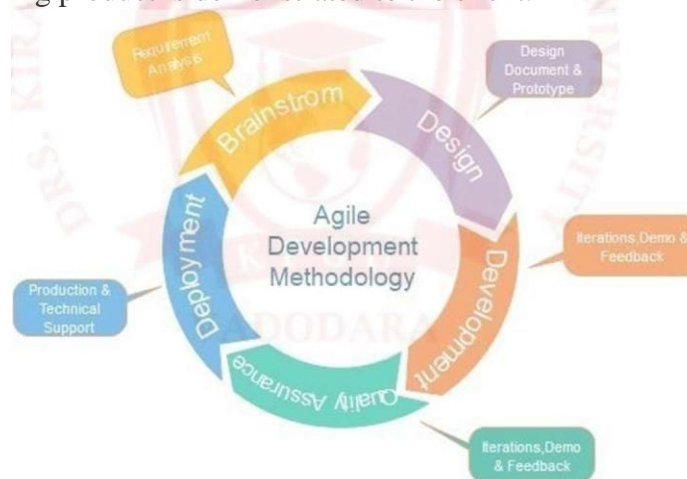


Fig. Agile Model

Advantages:

- Frequent Delivery
- Face-to-Face Communication with clients.
- Efficient design and fulfils the business requirement
- Anytime changes are acceptable. • It reduces total development time.

Disadvantages:

- Due to the shortage of formal documents, it creates confusion and crucial decisions taken throughout various phases can be misinterpreted at any time by different team members.
- Due to the lack of proper documentation, once the project completes and the developers allotted to another project, maintenance of the finished project can become a difficulty.

Phases:

1. Requirements gathering
2. Design the requirements
3. Construction/ iteration
4. Testing/ Quality assurance
5. Deployment
6. Feedback

Testing Methods:

- Scrum
- Crystal
- Dynamic Software Development Method(DSDM)
- Feature Driven Development(FDD)
- Lean Software Development
- Extreme Programming(XP)

When to use it:

The tenets of **Agile**—adaptability, iteration, continuous delivery, and short time frames, among others—make it a project management style that's better suited for ongoing projects and projects where certain details aren't known from the outset. That means if a project doesn't have clear constraints, timelines, or available resources, it's a good candidate for an Agile approach.

For example, designing and launching a new product might push a team against several unforeseen challenges. Having an Agile approach can mean the project already has the methodology in place to test products as often as needed, iterate quickly.

Why is this model being suitable for my project?

- Agile is an approach to project management that centers around incremental and iterative steps to completing projects.

SOFTWARE ENGINEERING

Subject Code: 21CS2506

- The approach prioritizes quick delivery, adapting to change, and collaboration rather than top-down management and following a set plan.
- There will require constant updates required for the histories of transaction done, cancelled orders, delivered ordered, deleted products as well as security alerts so iterative model comes in the phase.
- It involves both development and maintenance phases. The under-development product is defined as completed when it satisfies all of its functional requirements.
- The financial institutes can use a central system to share their work progress and enable errors to be caught faster.

It allows having a frequent overview, analysis, and implementation at a 'reduced cost'. So this is how the in the banking system this methodology is chosen.



Practical-4

Aim: Select project of your choice and construct Software Requirement Specification document.

Online Transaction Fraud Detection

1. Introduction:

1.1 Purpose

The purpose of this document is to comprehensively define the requirements for the Online Transaction Fraud Detection System, with a focus on its functionalities, features, performance, security, and other aspects.

1.2 Scope

The Online Transaction Fraud Detection System aims to provide a robust platform for detecting and preventing fraudulent online transactions. This system will analyze transaction data in real-time and alert relevant parties when suspicious activities are identified.

1.3 Definitions, Acronyms, and Abbreviations

- SRS: Software Requirement Specification
- OTP: One-Time Password

2. System Overview

2.1 System Description:

The Online Transaction Fraud Detection System will comprise multiple interrelated components, including data collection, analysis, reporting, and administration. It will be designed to interface with various external systems and databases to access transaction data for analysis.

3. Functional Requirements

3.1 User Registration and Management

1. The system shall provide a user registration module allowing administrators to create, modify, and deactivate user accounts.
2. User roles shall include administrators, analysts, and regular users, each with distinct privileges and access levels.

3.2 Data Collection

1. The system shall support real-time data collection from multiple sources, including web services, databases, and external APIs.
2. It shall be able to ingest and normalize data in various formats (e.g., JSON, XML, CSV) and from different transaction sources.
3. Data collection shall include user details, transaction amount, location, transaction timestamp, and other relevant information.

3.3 Data Analysis

1. The system shall employ machine learning algorithms and pattern recognition techniques to analyze transaction data.
2. Real-time data analysis will compare current transactions with historical user behavior to identify anomalies.
3. The system shall assign risk scores to transactions based on the likelihood of fraud, and classify them as low, medium, or high risk.
4. Flagged transactions shall be queued for further investigation by analysts.

3.4 Reporting

1. The system shall offer a web-based reporting dashboard for administrators and analysts.
2. Reports will include detailed information on flagged transactions, their associated risk scores, and relevant user data.
3. Users shall have the ability to generate ad-hoc reports and schedule automated reports to be sent via email.

3.5 Alerting

1. The system shall provide real-time alerting capabilities, issuing alerts for transactions with high-risk scores.
2. Alert notifications shall be delivered via email, SMS, or other predefined channels.
3. Alerts shall include essential information, such as transaction details, user data, and risk assessment

SOFTWARE ENGINEERING

Subject Code: 21CS2506

4. Non-Functional Requirements**4.1 Performance**

1. The system shall process transactions within a maximum of 2 seconds from the time of data ingestion.
2. It should be capable of handling a minimum of 1000 transactions per minute, with the ability to scale horizontally to accommodate increasing transaction volumes.

4.2 Security

1. All user data and transaction data shall be encrypted during transmission and storage.
2. Access control and authentication mechanisms shall be in place to restrict system access based on user roles.
3. The system shall maintain comprehensive audit logs for user actions and data access.

4.3 Scalability

1. The system shall be designed with scalability in mind, allowing for the addition of processing and storage resources to accommodate growing transaction data volumes.

5. Constraints

1. The system will be developed using Python for its core functionality.
2. The database system will be PostgreSQL for data storage and retrieval.
3. The system will rely on external APIs and data sources to acquire transaction data.

6. Assumptions and Dependencies

1. The system assumes a stable internet connection for real-time data collection.
2. It depends on external data sources for transaction data, which must be available and accessible via APIs.

7. Future Enhancements

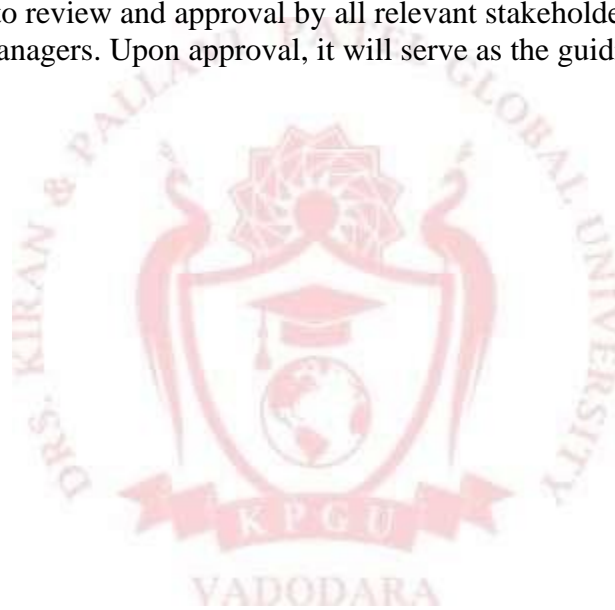
1. Integration with third-party fraud detection services to enhance analysis.
2. Support for additional data sources and formats to improve data compatibility.
3. Advanced analytics and reporting capabilities, including predictive analytics.

8. Conclusion

This comprehensive Software Requirement Specification document provides a detailed roadmap for the development and implementation of the Online Transaction Fraud Detection System. It will serve as the foundation for design, development, and testing.

9. Approval

This document is subject to review and approval by all relevant stakeholders, including clients, developers, and project managers. Upon approval, it will serve as the guiding document for the system's development.



PRACTICAL-5

- ❖ **Aim: Construct Software Project Management Plan (SPMP) document for your project. Decide effort estimation for the same.**
- ❖ **Description: This SPMP document outlines a brief plan about how project is to be shaped.**

Software Project Management Plan for Online Transaction Fraud Detection Preface:

The purpose of this document is to specify the project plan to develop the Online Transaction Fraud Detection System. This document outlines a brief plan about how project is to be shaped and also includes the milestones and deliverables. This document will serve as guide for developers, developing the project. Updates of this document will serve to record the progress of the project.

1. Introduction:

(a) Objectives:

The main purpose of the project is intended to develop an application for detection of transaction frauds. This system prevents fraudulent users from misusing the details of the credit-card of the genuine users for their personal gain, which will be designed using MySQL as the back – end and Visual Basic application package as the front – end. We tried to detect fraudulent transaction before transaction succeed. The owner is immediately alerted about the attempted fraud and the transaction is blocked. Thus, the system protects legitimate users from financial loss. The system helps in making electronic payment safer and more reliable.

(b) Major functions:

Online transaction fraud detection on e-commerce website is an application that helps to detect transaction frauds on e-commerce directly buying goods or services from the seller Via Internet using a web browser. The credit card fraud detection features use user behavior and location scanning to check for unusual patterns. These patterns include user characteristics such as user spending patterns as well as usual user geographic locations to verify his identity. If any unusual pattern is detected, the system requires re-verification. The owner is immediately alerted about the attempted fraud and the transaction is blocked. Thus, the system protects legitimate users from

financial loss. The system helps in making electronic payment safer and more reliable.

(c) Performance Issue:

The major issue is to deliver a complete whole portal which is not possible within little time and with RAD model so deliverables are broken down in the project plan into smaller deliverables and activities.

Factors critical in deciding the performances of the software are:

Internet Connection

Being predominantly a web-based application, fast internet connection is necessity. Slow connections may hamper performance.

Server performance

Performance may also be hampered by excessive net traffic.

(d) Management and technical constrains:

- ☐ The project should be completed within specified time period including Planning, Designing, Development, Testing and Deployment.
- ☐ The project should be completed within specified budget.
- ☐ The Requirement Traceability Matrix (RTM) should be correlated and completed.
- ☐ All the Entry and Exit criteria of all the stages should met.
- ☐ The product should be user-friendly, reliable and should maintain the industry standards without compromising the quality.
- ☐ The system architecture and design should be open and in a standard way such that additional functionalities can be added later without much effort.
- ☐ Database designed should me normalized.
- ☐ Transfer of rupees for booking tickets should be done through secured connection

2. Project Estimates:

Estimation technique for project:

COCOMO (Constructive Cost Estimation Model) was proposed by Boehm 1981.

Boehm postulated that any software development project can be classified into a one of the following categories based on development complexity: organic, semidetached and

embedded. In order to classify a product into the identified categories, Boehm requires us to consider not only the characteristics of the product but also of the development team and the development environment. Roughly speaking, the three product classes correspond to application, utility and system programs, respectively. Normally data processing programs are considered to be application programs. Compilers, linkers etc. are utility programs. Operating systems and real time system programs etc. are system programs. System programs interact directly with the hardware and typically involve meeting timing constraints and concurrent processing.

Also the utility programs are three times as difficult to write as application programs and, system programs are roughly three times as difficult as utility programs. Thus, the relative levels of product development complexity for the three categories (application, utility and system programs) of products are 1:3:9.

Boehm's [1981] definitions of organic, semidetached and embedded systems are elaborated as follows:

Organic: We can consider a development project to be of organic type, if the project deals with developing a well understood application program, the size of the development team is reasonably small, and the team members are experienced in developing similar types of projects.

3. Project Estimates:

Estimation technique for project:

COCOMO (Constructive Cost Estimation Model) was proposed by Boehm 1981. Boehm postulated that any software development project can be classified into one of the following categories based on development complexity: organic, semidetached and embedded. In order to classify a product into the identified categories, Boehm requires us to consider not only the characteristics of the product but also of the development team and the development environment. Roughly speaking, the three product

classes correspond to application, utility and system programs, respectively. Normally data processing programs are considered to be application programs. Compilers, linkers etc. are utility programs. Operating systems and real time system programs etc. are system programs. System programs interact directly with the hardware and typically involve meeting timing constraints and concurrent processing.

Also the utility programs are three times as difficult to write as application programs and, system programs are roughly three times as difficult as utility programs. Thus, the relative levels of product development complexity for the three categories (application, utility and system programs) of products are 1:3:9.

Boehm's [1981] definitions of organic, semidetached and embedded systems are elaborated as follows:

Organic: We can consider a development project to be of organic type, if the project deals with developing a well understood application program, the size of the development team is reasonably small, and the team members are experienced in developing similar types of projects.

Semidetached: The project can be considered of this type if the development team consists of a mixture of experienced and inexperienced staff. Team members may have limited experience on related systems but may be unfamiliar with some aspects of the system being developed.

Embedded: The project can be considered of this type, if the software being developed is strongly coupled to complex hardware, or if stringent regulations on the operational procedures exist. Here we have chosen the embedded model since all the team members are new in the field of the software development and also the software product size is relatively small.

Estimation of Development Efforts-

1. Organic Model:

$$\text{Efforts} = 2.4(\text{KLOC})^{1.05} \text{ Person-Months}$$

2. Semi-Detach Model:

$$\text{Efforts} = 3.0(\text{KLOC})^{1.12} \text{ Person-Months}$$

3. Embedded Model:

$$\text{Efforts} = 3.6(\text{KLOC})^{1.20} \text{ Person-Months}$$

Estimation of Development Time-

1. Organic model:

$$\text{Efforts} = 2.5(\text{Efforts})^{0.38} \text{ Person-Months}$$

2. Semi-Detached Model:

$$\text{Efforts} = 2.5(\text{Efforts})^{0.35} \text{ Person-Month}$$

3. Embedded Model:

$$\text{Efforts} = 2.5(\text{Efforts})^{0.32} \text{ Person-Month}$$

Cost and Effort Calculation:

Efforts: Organic model:

$$= 2.4(\text{KLOC})^{1.05}$$

$$= 2.4(1.25)^{1.05}$$

$$= 3.0$$

$$= 3 \text{ Person-months (approx.)}$$

Time Duration: Organic model:

$$= 2.5(\text{Efforts})^{0.32}$$

$$= 2.5(3)^{0.32}$$

$$= 3.79$$

$$= 4 \text{ months (approx.)}$$

3. **Schedule:**

Work breakdown structure

To execute the task successfully we need to follow the work breakdown structure.

Work Breakdown Structure:

A work breakdown structure can be defined as dividing the whole project into individual components in a hierarchical structure. The breakdown structure defines tasks that can be completed independently of other tasks, resource allocation, and assigning the responsibilities of resources in the project.

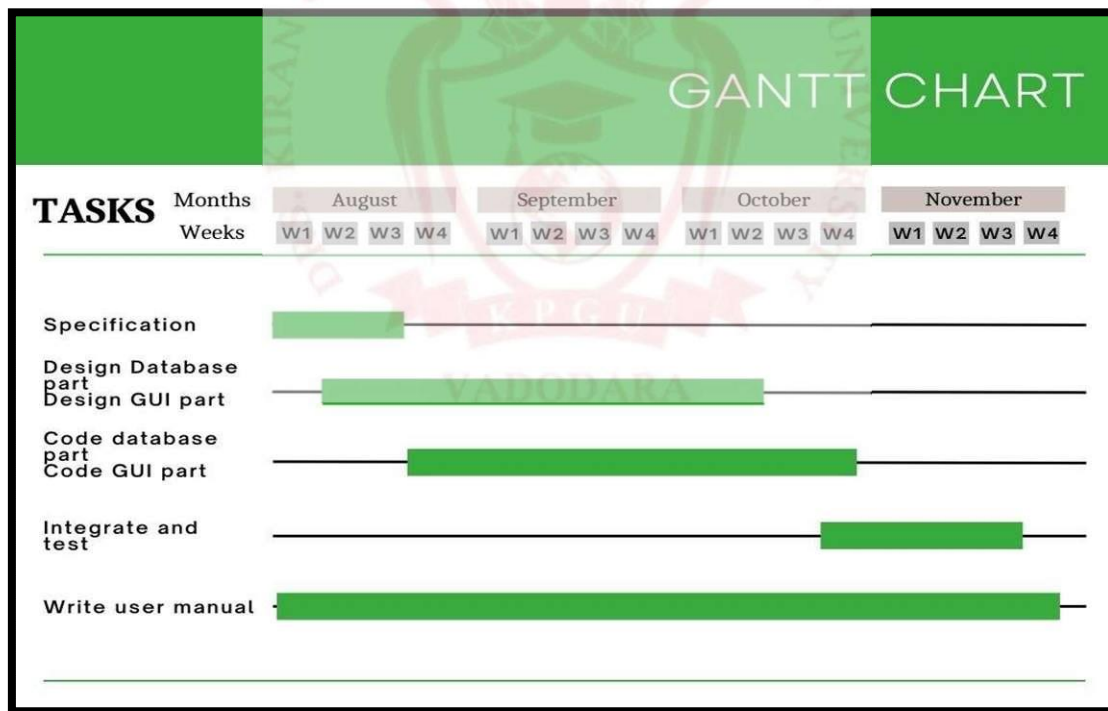
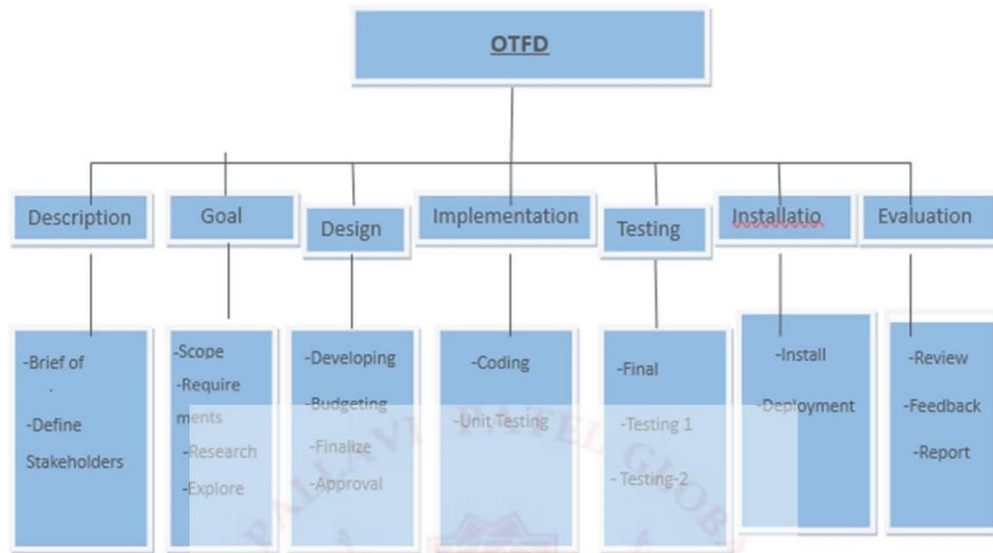
a. Description

SOFTWARE ENGINEERING

Subject Code: 21CS2506

-
- i) Brief of project
 - ii) Define stakeholders
 - b. Achieve goal
 - i) Scope
 - ii) Requirements
 - iii) Research
 - iv) Explore
 - v) Consult
 - vi) Concept development
 - vii) Plan
 - c. Design
 - i) Develop BLA
 - ii) Budgeting
 - iii) Finalize
 - iv) Approval of budget
 - v) Approval of plan
 - vi) Re-approval in requirement
 - d. Implementation
 - i) Coding
 - ii) Unit testing
 - e. Testing
 - i) Final commotion
 - ii) Testing phase 1
 - iii) Testing phase 2
 - f. Installation and deployment
 - i) installation
 - g. Evaluation
 - i) Review
 - ii) Feedback

iii) Report



4. Project Resources:

a) People

- The Stakeholders
- Team leaders

SOFTWARE ENGINEERING

Subject Code: 21CS2506

- The Software Team
- Agile Team (Implementer)
- Co-ordination and Communication Issues.

b) Hardware and Software Requirements

Software Requirements:

- An efficient OS version
- Android studio
- Key Packages
- Device binaries
- Build toolchain
- Android Development Tool kit (ADT kit)
- Platforms for development – Qpython/PySide/PyMob/Kivy etc

Hardware Requirements:

- The system shall run on any smart devices like laptop, pc, mobile, tablet etc.
- A 64-bit environment is required for Android 2.3.x (Gingerbread) and higher versions, including the master branch.
- At least 250GB of free disk space to check out the code and an extra 150 GB to build it.
- At least 4 GB of available RAM is required. 8 GB recommended.
- Processor must also be greater than 2.3 GHz.

Software Interfaces: The system shall interface with any Web Browser easily and on any Operating System like mac, android and Microsoft etc.

Hardware Interfaces: The system shall run on any smart devices like laptop, pc, mobile, tablet etc.

c) Special Resources:

- ☐ Python programming
- ☐ SQL database
- ☐ Emulator for testing a product

5. Staff Organization:

Democratic team structure is used, as the name implies there is no project manager. All are treated

SOFTWARE ENGINEERING

Subject Code: 21CS2506

as team leaders and all of them can give any suggestions for the project development. Everyone is given equal rights for making and giving changes which are beneficiary for the project.

Democratic organization leads to higher morale and job satisfaction. Consequently, it suffers from less man-power turnover.

As team is of less than 5 engineers, democratic team structure is suitable for project and for research-oriented projects.

6. Risk Management:

a) Risk Analysis:

1) Product Size Risks

- ✓ Estimated size of product in number of programs, files, transactions
- ✓ Size of database created or used by the product.
- ✓ Number of projected changes to the requirements for the product before delivery and after delivery.

2) Business Risks

- ✓ Amount and Quality of product documentation that must be produced and delivered to the customer.
- ✓ Delivery deadlines
- ✓ Costs associated with late delivery and defective product.

3) Client Risks

- ✓ Communicating with developers
- ✓ Will customer agree to spend time in formal requirements gathering meetings to identify project scope.
- ✓ When working for the first time

4) Resource Risks

- ✓ Number of Resources and skillsets

5) Technology Risks

- ✓ Requirements put excessive performance constraints on the product
- ✓ Customer demanding new technology.

b) Risk identification:

SOFTWARE ENGINEERING

Subject Code: 21CS2506

- Risks with respect to the work to be done
 - ✓ Miscommunication
 - ✓ Time out
 - ✓ Design errors
 - ✓ Illness or absence of team members
 - ✓ Server crash
- Risks with respect to the management
 - ✓ Illness or sudden absence of the project manager
- Risks with respect to the resources
 - ✓ Unavailability of the technical advisor when needed
- Risks with respect to the customer.
 - ✓ The customer changes his mind about the requirements
 - ✓ The customer is not available when needed

The Risk Manager is responsible for identifying risks and monitor risk. After identifying Risk Manager Need to solve problem by replace old system with new one. If need arise Risk Managers should modify, delete and add information to the database.

c) Risk Estimation:

Risk estimation also called risk projection, attempts to rate each risk in two ways:

- ✓ The likelihood or probability that the risk is real.
- ✓ The consequences (i.e. effect or result) of the problems associated with the risk, should it occur.

The project planner, along with other managers and technical staff, performs four risk projection activities:

- ✓ Establish a scale that reflects the supposed likelihood of a risk
- ✓ Describe the consequences of the risk,
- ✓ Estimate the impact of the risk on the project and the product,
- ✓ Note the overall accuracy of the risk projection so that there will be no misunderstanding

The intent of these steps is to consider risks in a manner that leads to prioritization. By prioritizing risks, the team can allocate resources where they will have the most impact.

7. Project Tracking & Control Plan:

The monitoring of progress is done by the Project Manager using the following means:

- ✓ Weekly Project Group Meetings
- ✓ Progress Meetings
- ✓ Project metrics
- ✓ Team leader meetings

Project cost will be tracked by counting manpower, number of resources needed, wages given to staff, etc. As per the schedule all the activities will be performed on time and frequency of report structure should be as per the standards of the organization and meeting structure would be frequently so that team members get to know where they are standing right now and can plan further meetings. After each task audit will be there so that team members get to know errors and fault of the system, after that they will recover it as soon as possible.

8. Miscellaneous plan:

a) Quality assurance plan:

Quality Assurance plan includes the techniques to test the quality of the current product. Again acceptance tests are used to see if the components are working properly when they are brought together. Additionally, performance tests will be used to see the overall quality in terms of efficiency. Results of these tests are considered frequently and these tests are run daily. If any improvement is needed already implemented parts will be improved before going on to other parts.

b) Configuration Management plan:

This part of the SPMP contains the configuration management plan for the software project, to include the methods that will be used to provide configuration identification, evaluation, and release management. Configuration identifications are done according to the needs of the customer. Similarly, evaluation is also done with customer but in- team evaluations are also done as frequently as possible. Releases are done as big milestones are achieved. They are scheduled above. Change requests are analyzed during development phases and schedule is updated accordingly. Possible changes might result in missed deadlines. Therefore, these

observations are done during development frequently.

c) Validation & verification:

Verifications are done first via various tests like unit tests and acceptance tests. These will ensure us that we do the right thing. Similarly, verification is done between team members in weekly meetings. Validation is done with the help of customer. There will be weekly meetings with the customer to validate the parts that is developed. Demos are done for this purpose. At each meeting the milestones that are met is showed to the customer and current status of the development is said to him.

d) System Testing plan:

- ✓ Recovery testing (fault tolerant)
- ✓ Security testing
- ✓ Stress testing (volume, resources)
- ✓ Performance testing (real-time, embedded system)
- ✓ Acceptance Testing
- ✓ Alpha test: at the developer's site, controlled environment
- ✓ Beta test: at one or more customer site



PRACTICAL-6

Aim: Perform modeling for your project. Draw Data Flow Diagram (DFD) using function-oriented approach.

Description:





Design:

Design phase deals with transforming the requirements, as described in the SRS document, into a form that is implemented using a programming language. The various designs of this system are shown as following:

Data flow diagram:

Data Flow diagram is a graphical representation of flow of data throughout the information system.

Data flow diagrams illustrate how data is processed by a system in terms of inputs and outputs.

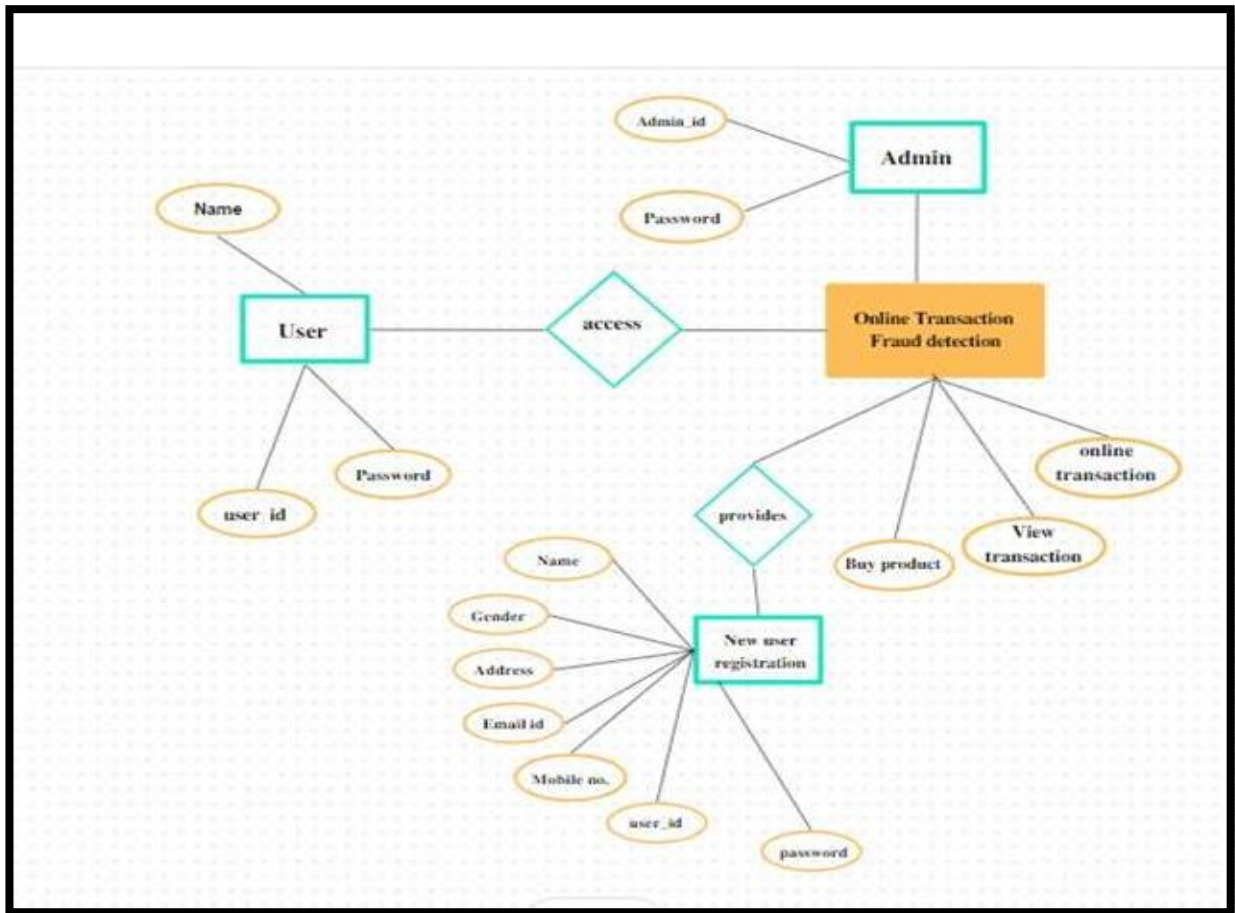
| <u>Name</u> | <u>Notation</u> | <u>Role</u> |
|-----------------|---|--|
| Process |  | Transforms incoming dataflow to output data flow. |
| Data Store |  | Repositories of data in the system. |
| Dataflow |  | Dataflow are pipelines through which packets of information flow |
| External Entity |  | External entities are objects outside the system with which the system communicates. |

PRACTICAL-7

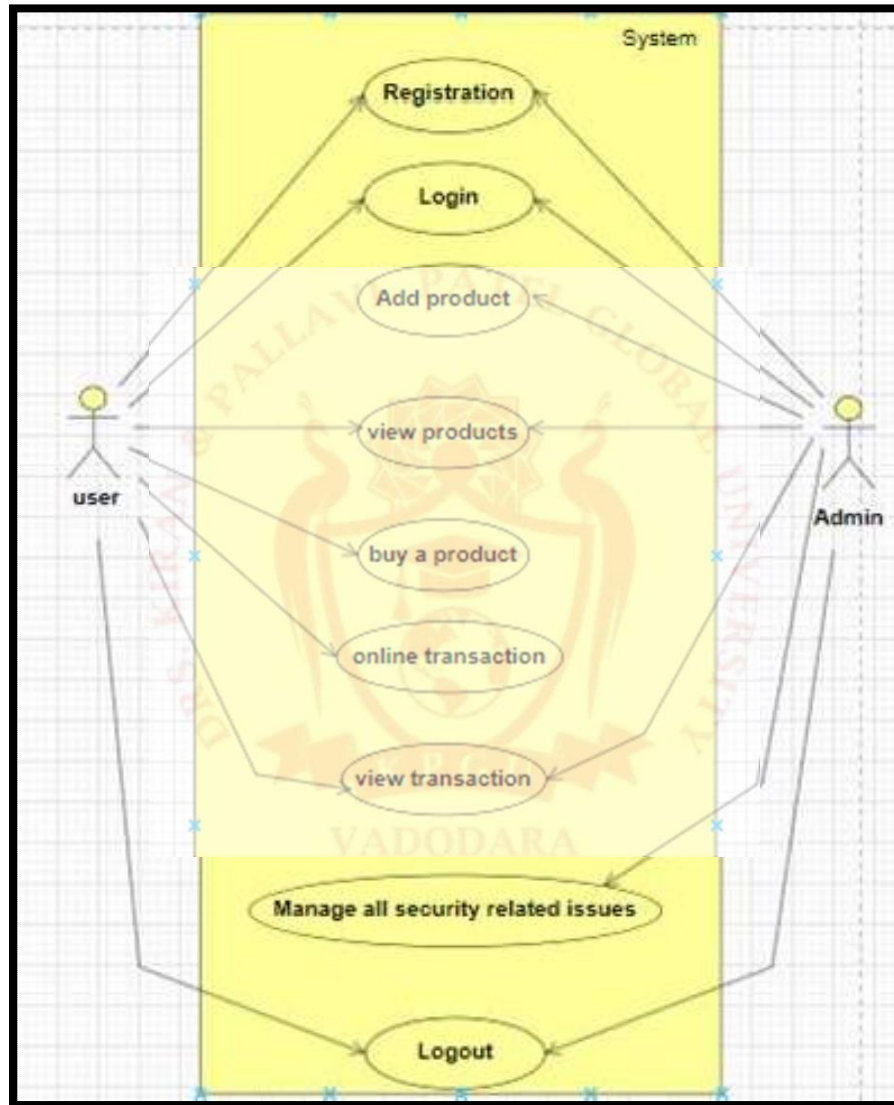
Aim: Prepare UML modeling using object-oriented approach. (Use Case, Class Diagram and Activity Diagram (using swim lane)).

- **Description:**

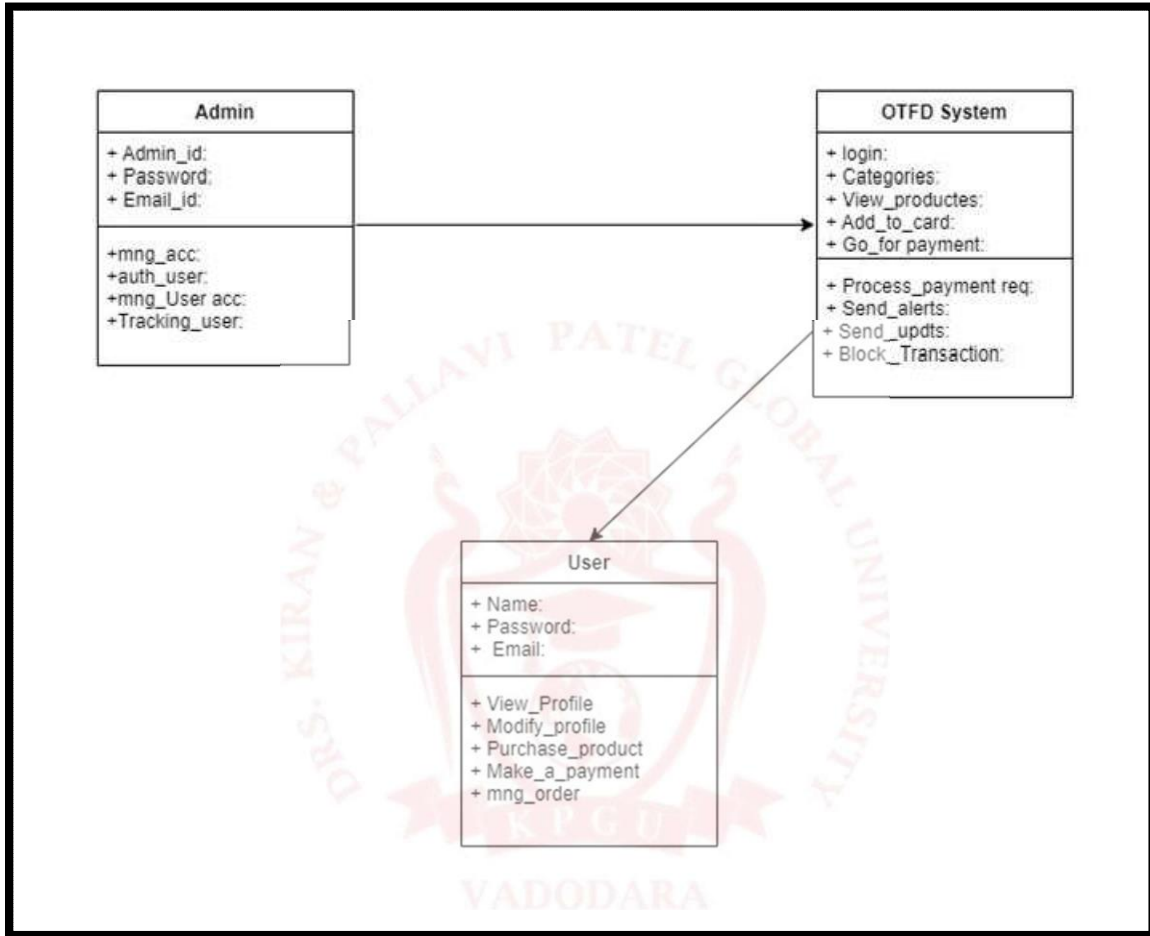
- **E-R Diagram:**



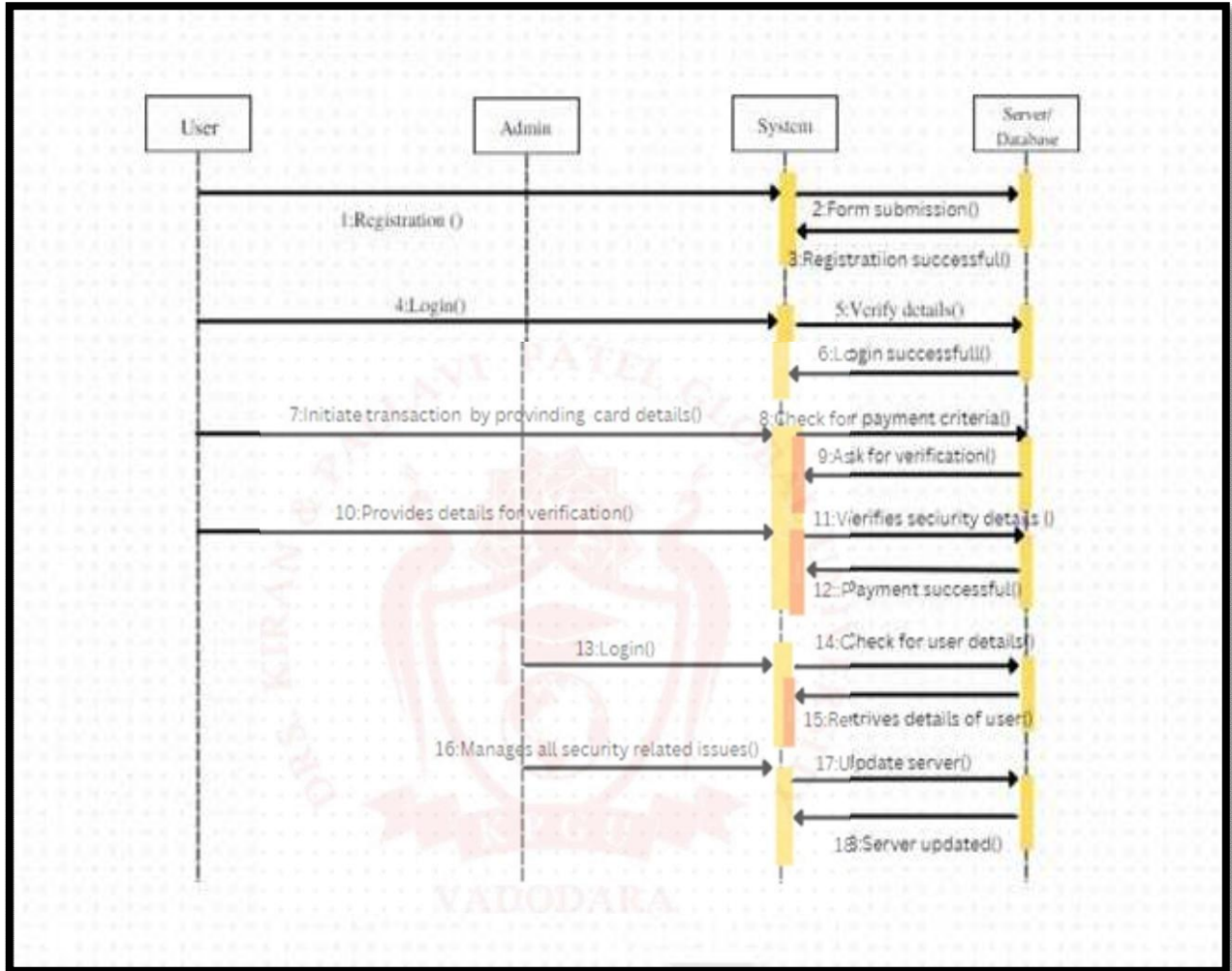
➤ Use case diagram:



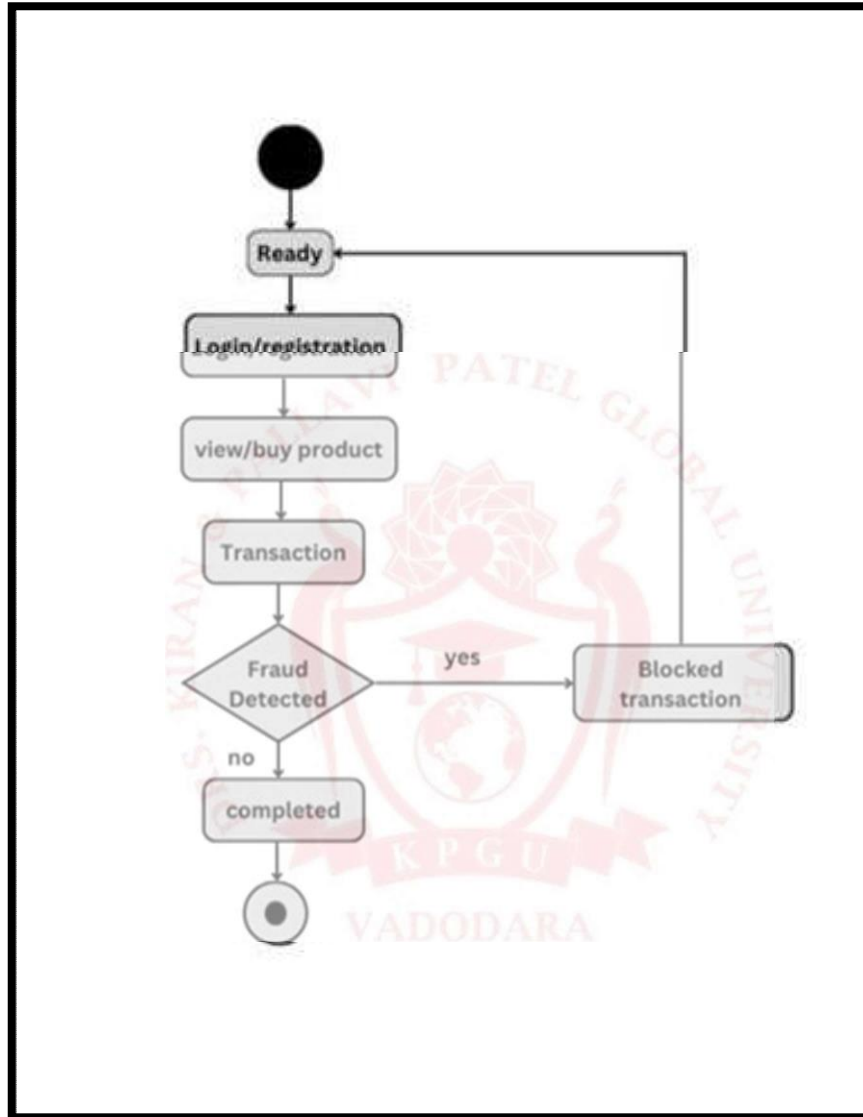
➤ **Class diagram:**



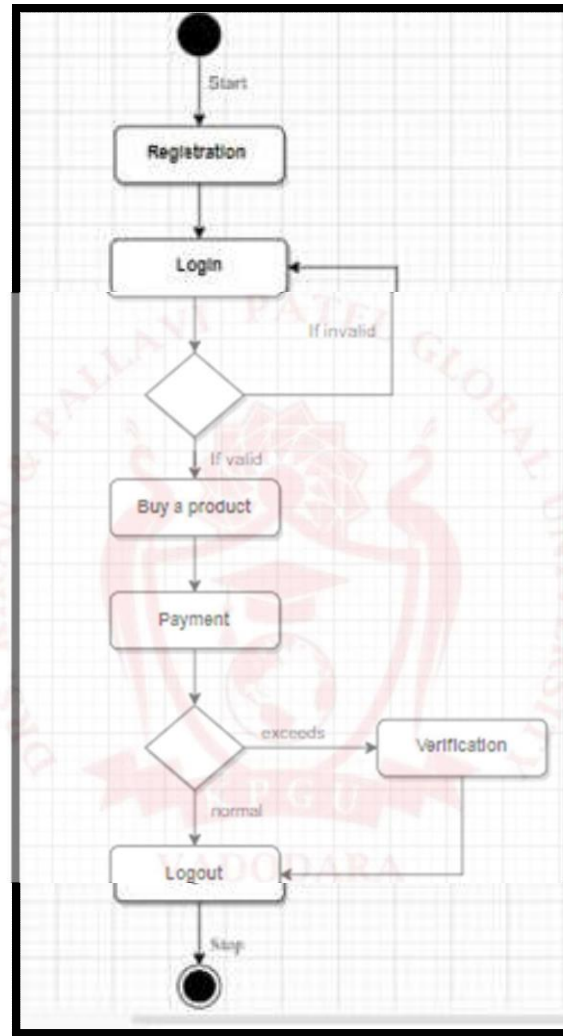
➤ **Sequence diagram:**



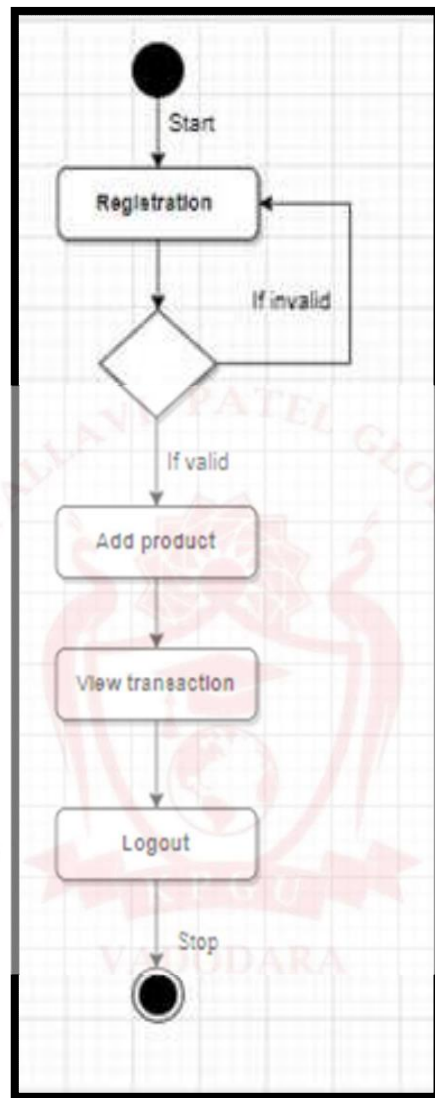
➤ **State diagram:**



➤ **Activity diagram:**



Activity diagram of user



Activity diagram of admin

PRACTICAL-8

Aim: Understand COCOMO Model with suitable example. Also decide cost estimation for your project.

Description:

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 constant to be used in subsequent calculations. These characteristics pertaining to different system types are mentioned below.

1. **Organic** – A software project is said to be an organic type if the team size required is adequately small, the problem is well understood and has been solved in the past and also the team members have a nominal experience regarding the problem.
2. **Semi-detached** – A software project is said to be a Semi-detached type if the vital characteristics such as team size, experience, knowledge of the various programming environment lie in between that of organic and Embedded. The projects classified as Semi-Detached are comparatively less familiar and difficult to develop compared to the organic ones and require more experience and better guidance and creativity. E.g.: Compilers or different Embedded Systems can be considered of Semi-Detached type.
3. **Embedded** – A software project requiring the highest level of complexity, creativity, and experience requirement fall under this category. Such software requires a larger team size than the other two models and also the developers need to be sufficiently experienced and creative to develop such complex models.

According to Boehm, software cost estimation should be done through three stages:

SOFTWARE ENGINEERING

Subject Code: 21CS2506

1. Basic Cocomo Model
2. Intermediate Cocomo Model
3. Detailed Cocomo Model

Basic Cocomo Model-

The basic COCOMO model gives an approximate estimate of the project parameters. The basic COCOMO estimation model is given by the following expressions

$$\text{Effort} = a_1 * (\text{KLOC})^{a_2} \text{PM}$$

$$\text{Tdev} = b_1 * (\text{Effort})^{b_2} \text{ months}$$

Effort is the total effort required to develop the software product, expressed in person months (PMs).

KLOC is the estimated size of the software product expressed in Kilo Lines of Code

a_1 , a_2 , b_1 , b_2 are constants for each category of software products,

Tdev is the estimated time to develop the software, expressed in months,

| Project | a_1 | a_2 | b_1 | b_2 |
|--------------|-------|-------|-------|-------|
| Organic | 2.4 | 1.05 | 2.5 | 0.38 |
| Semidetached | 3.0 | 1.12 | 2.5 | 0.35 |
| Embedded | 3.6 | 1.20 | 2.5 | 0.32 |

The effort estimation is expressed in units of person-months (PM)

Example: Assume that the size of an organic type software product has been estimated to be 32,000 lines of source code. Assume that the average salary of software engineers be Rs. 15,000/- per month. Determine the effort required to develop the software product and the nominal development time.

$$\text{Effort} = a_1 * (\text{KLOC})^{a_2} \text{PM}$$

$$= 2.4 + (32)1.05 \text{ PM}$$

$$= 91$$

$$\text{Tdev} = b_1 * (\text{Effort})^{b_2} \text{ months}$$

$$= 2.5 * (91)0.38 \text{ Months} = 14 \text{ Months}$$

Cost required to develop the product = $14 \times 15000 = \text{Rs. } 2,10,000/-$

Calculation for the project:

Estimation of Development Efforts-

1. Organic model:
Efforts = $2.4(\text{KLOC})^{1.05}$ Person-Months
2. Semi-Detached Model:
Efforts = $3.0(\text{KLOC})^{1.12}$ Person-Months
3. Embedded Model:
Efforts = $3.6(\text{KLOC})^{1.20}$ Person-Months

Estimation of Development Time-

1. Organic model:
Efforts = $2.5(\text{Efforts})^{0.38}$ Person-Months
2. Semi-Detached Model:
Efforts = $2.5(\text{Efforts})^{0.35}$ Person-Months
3. Embedded Model:
Efforts = $2.5(\text{Efforts})^{0.32}$ Person-M

Cost and Effort Calculation:

Efforts: Organic model:

$$\begin{aligned} &= 2.4(\text{KLOC})^{1.05} \\ &= 2.4(1.25)^{1.05} \\ &= 3.0 \\ &= 3 \text{ Person-months (approx.)} \end{aligned}$$

Time Duration: Organic model:

$$\begin{aligned} &= 2.5(\text{Efforts})^{0.32} \\ &= 2.5(3)^{0.32} \\ &= 3.79 \\ &= 4 \text{ months (approx.)} \end{aligned}$$

PRACTICAL-9

Aim: Understand function point. Analyze the case study. Identify error & solve it.

Description:

A function point is a "unit of measurement" to express the amount of business functionality an information system (as a product) provides to a user. Function points are used to compute a functional size measurement (FSM) of software. The cost (in dollars or hours) of a single unit is calculated from past projects.

Measure size in terms of the amount of functionality in a system. Function points are computed by first calculating an *unadjusted function point count* (UFC).

Counts are made for the following categories

- *External inputs* – those items provided by the user that describe distinct application- oriented data (such as file names and menu selections)
- *External outputs* – those items provided to the user that generate distinct application- oriented data (such as reports and messages, rather than the individual components of these)
- *External inquiries* – interactive inputs requiring a response
- *External files* – machine-readable interfaces to other systems
- *Internal files* – logical master files in the system

Multiply each number by a weight factor, according to complexity (**simple**, **average** or **complex**) of the parameter, associated with that number. The value is given by a table:

| Parameter | Simple | Average | Complex |
|---------------------|--------|---------|---------|
| User Inputs | 3 | 4 | 6 |
| User Outputs | 4 | 5 | 7 |
| User Requests | 3 | 4 | 6 |
| Files | 7 | 10 | 15 |
| External Interfaces | 5 | 7 | 10 |

- ☐ Calculate the total **UFP** (Unadjusted Function Points)
- ☐ Calculate the total **TCF** (Technical Complexity Factor)

SOFTWARE ENGINEERING

Subject Code: 21CS2506

- ☐ Sum the resulting numbers to obtain **DI** (degree of influence)
- ☐ **TCF** given by the formula: $TCF = 0.65 + 0.01 * DI$

Function Points are given by the formula: $FP = UFP * TCF$

Case Study:

The spell checker accepts as input a document file & an optional personal dictionary file. The checker lists all words not contained in either of these files. The user can query the number of words processed & the number of spelling errors found at any stage during processing.



PRACTICAL-10

Aim: Understand Cyclometric Complexity for coding. Analyze the case study.

Description:

Cyclomatic complexity is a software metric (measurement), used to indicate the complexity of a program. It is a quantitative measure of the number of linearly independent paths through a program's source code. It was developed by Thomas J. McCabe, Sr. in 1976.

Cyclomatic complexity is computed using the control flow graph of the program. One testing strategy, called basis path testing by McCabe who first proposed it, is to test each linearly independent path through the program.

Definition:

The cyclomatic complexity of a section of source code is the number of linearly independent paths within it. For instance, if the source code contained no control flow statements (conditionals or decision points), the complexity would be 1, since there would be only a single path through the code. If the code had one single-condition IF statement, there would be two paths through the code: one where the IF statement evaluates to TRUE and another one where it evaluates to FALSE, so the complexity would be 2. Two nested single-condition IFs, or one IF with two conditions, would produce a complexity of 4.

Mathematically, the cyclomatic complexity of a structured program is defined with reference to the control flow graph of the program, a directed graph containing the basic blocks of the program, with an edge between two basic blocks if control may pass from the first to the second. The complexity **M** is then defined as

$$M = E - N + 2P$$

Where,

E = the number of edges of the graph

N = the number of nodes of the graph

P = the number of connected components

For a single program (or subroutine or method), P is always equal to 1. So a simpler formula for a single subroutine is:

$$M = E - N + 2$$

Applications:

Determining the number of test cases that are necessary to achieve thorough test coverage of a particular module.

It is useful because of two properties of the cyclomatic complexity, M , for a specific module:

- M is an upper bound for the number of test cases that are necessary to achieve a complete branch coverage.
- M is a lower bound for the number of paths through the control flow graph (CFG). Assuming each test case takes one path, the number of cases needed to achieve path coverage is equal to the number of paths that can actually be taken.

All three of the above numbers may be equal:

branch coverage \leq cyclomatic complexity \leq number of paths.

Case study:

consider a program that consists of two sequential if-then-else statements:

```
if( c1() )
```

```
f1();
```

```
else
```

```
f2();
```

```
if( c2() )
```

```
f3();
```

```
else
```

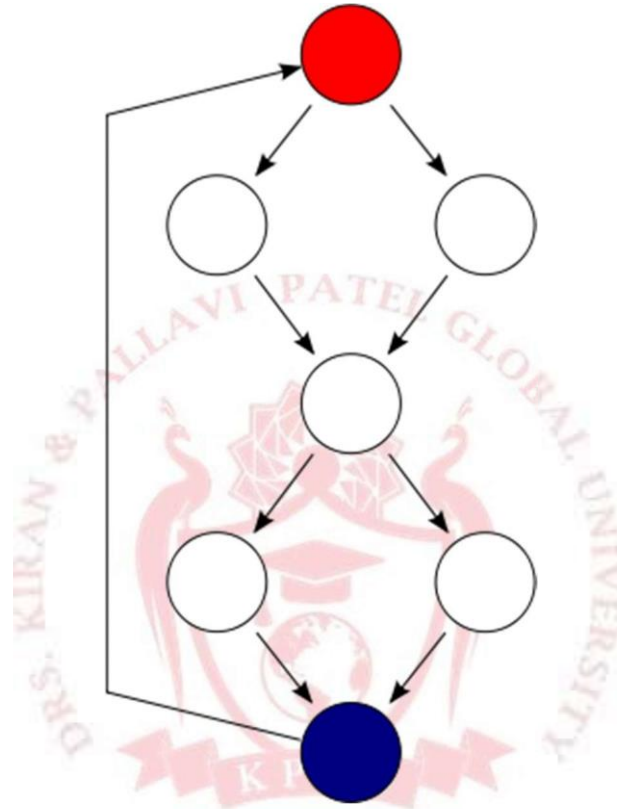
```
f4();
```

In general, in order to fully test a module all execution paths through the module should be exercised. This implies a module with a high complexity number requires more testing effort than a module with a lower value since the higher complexity number indicates more

SOFTWARE ENGINEERING

Subject Code: 21CS2506

pathways through the code. This also implies that a module with higher complexity is more difficult for a programmer to understand since the programmer must understand the different pathways and the results of those pathways.



In this example, two test cases are sufficient to achieve a complete branch coverage, while four are necessary for complete path coverage. The CC of the program is 3 (as the strongly connected graph for the program contains 9 edges, 7 nodes and 1 connected component) $(9 - 7 + 1)$.

Unfortunately, it is not always practical to test all possible paths through a program. Considering the example above, each time an additional if-then-else statement is added, the number of possible paths doubles. As the program grew in this fashion, it would quickly reach the point where testing all of the paths was impractical.

PRACTICAL-11

Aim: Study of Open Source Software/Learning for implementation of project.

Description:

Overview:

Open source is a term that originally referred to open source software (OSS). Open source software is code that is designed to be publicly accessible—anyone can see, modify, and distribute the code as they see fit.

Open source software is developed in a decentralized and collaborative way, relying on peer review and community production. Open source software is often cheaper, more flexible, and has more longevity than its proprietary peers because it is developed by communities rather than a single author or company.

Open source has become a movement and a way of working that reaches beyond software production. The open source movement uses the values and decentralized production model of open source software to find new ways to solve problems in their communities and industries.

- **The history of open source is the history of the internet**

In the 1950s and 1960s researchers developing early internet technologies and telecommunication network protocols relied on an open and collaborative research environment. The Advanced Research Projects Agency Network (ARPANET), which would later become the foundation for the modern internet, encouraged peer review and an open feedback process. User groups shared and built upon one another's source code. Forums helped facilitate conversation and develop standards for open communication and collaboration.

- **How does an open source development model work?**

An open source development model is the process used by an open source community project to develop open source software. The software is then released under an open source license, so anyone can view or modify the source code. Many open source projects are hosted on [GitHub](https://github.com), where you can access repositories or get involved in community projects.

[Linux®](#), Ansible, and Kubernetes are examples of popular open source projects.

Linux and open source:

Linux is a free, open source operating system (OS), released under the GNU General Public License (GPL). It's also become the largest open source software project in the world.

The Linux operating system was created as an alternative, free, open source version of the MINIX operating system, which was itself based on the principles and design of Unix.

Because Linux is released under an open source license, which prevents restrictions on the use of the software, anyone can run, study, modify, and redistribute the source code, or even sell copies of their modified code, as long as they do so under the same license.

What's the difference between free, closed, and open source software?

For a long time open source software held the earlier label of "free software." The free software movement was formally established by Richard Stallman in 1983 through the [GNU Project](#). The free software movement organized itself around the idea of user freedoms: freedom to see the source code, to modify it, to redistribute it—to make it available and to work for the user in whatever way the user needed it to work.

Free software exists as a counterpart to proprietary or "closed source" software. Closed source software is highly guarded. Only the owners of the source code have the legal right to access that code. Closed source code cannot be legally altered or copied, and the user pays only to use the software as it is intended—they cannot modify it for new uses nor share it with their communities.

The name "free software," however, has caused a lot of confusion. Free software does not necessarily mean free to own, just free to use how you might want to use it. "Free as in freedom, not as in beer" the community has tried to explain. [Christine Peterson, who coined the term "open source,"](#) tried to address this problem by replacing 'free software' with 'open source': "The problem with the main earlier label, 'free software,' was not its political connotations, but that—to newcomers—its seeming focus on price is distracting. A term was needed that focuses on the key issue of source code and that does not immediately confuse those new to the concept."

Peterson proposed the idea of replacing "free software" with the term "open source" to a working

group that was dedicated, in part, to shepherding open source software practices into the broader marketplace. This group wanted the world to know that software was better when it was shared when it was collaborative, open, and modifiable. That it could be put to new and better uses, was more flexible, cheaper, and could have better longevity without vendor lock-in.

Eric Raymond was one of the members of this working group, and in 1997 he published some of these same arguments in his wildly influential essay ["The Cathedral and the Bazaar"](#). In 1998, partly in response to that essay, Netscape Communications Corporation open sourced their Mozilla project, releasing the source code as free software. In its open source form, that code later became the foundation for Mozilla Firefox and Thunderbird.

What are the values of open source?

There are lots of reasons why people choose open source over proprietary software, but the most common ones are:

- **Peer review:** Because the source code is freely accessible and the open source community is very active, open source code is actively checked and improved upon by peer programmers. Think of it as living code, rather than code that is closed and becomes stagnant.
- **Transparency:** Need to know exactly what kinds of data are moving where, or what kinds of changes have happened in the code? Open source allows you to check and track that for yourself, without having to rely on vendor promises.
- **Reliability:** Proprietary code relies on the single author or company controlling that code to keep it updated, patched, and working. Open source code outlives its original authors because it is constantly updated through active open source communities. Open standards and peer review ensure that open source code is tested appropriately and often.
- **Flexibility:** Because of its emphasis on modification, you can use open source code to address problems that are unique to your business or community. You aren't locked in to using the code in any one specific way, and you can rely on community help and peer review when you implement new solutions.

SOFTWARE ENGINEERING

Subject Code: 21CS2506

- **Lower cost:** With open source the code itself is free—what you pay for when you use a company like Red Hat is support, security hardening, and help managing interoperability.
- **No vendor lock-in:** Freedom for the user means that you can take your open source code anywhere, and use it for anything, at anytime.
- **Open collaboration:** The existence of active open source communities means that you can find help, resources, and perspectives that reach beyond one interest group or one company.



PRACTICAL-12

Aim: Develop test-case scenarios for your project.

Description:

Testing phase is a very important for a successful system. In this phase before implementing the new system into operation, for eliminating bugs a test run of the system is done. After completing codes for the whole programs of the system, a test plan should be developed and run one given set of test data.

| Test Case: 1 | | Test Case Name: Register | | |
|--|--------------------------------|---------------------------------|-----------|---------|
| System: OTFD | | Subsystem: Registration | | |
| Designed By: SMC | | Design Date: October 17,2022 | | |
| Executed By: Priya Koladiya | | Execution Date: October 17,2022 | | |
| Short Description: Test case for user registration when all details are valid. | | | | |
| Pre-conditions: User must navigate to registration page. | | | | |
| Step | Action | Expected System Response | Pass/Fail | Comment |
| User enters correct details | User clicks on Register button | User should be registered | Pass | |

SOFTWARE ENGINEERING

Subject Code: 21CS2506

| Test Case: 2 | | Test Case Name: Login | | |
|--|-----------------------------|---|---------------|---------|
| System: OTFD | | Subsystem: Login | | |
| Designed By: SMC | | Design Date: August 12,2022 | | |
| Executed By: Priya Koladiya | | Execution Date: October 17,2022 | | |
| Short Description: Test case for logging user in when credentials are valid. | | | | |
| Pre-conditions: User must navigate to login page | | | | |
| Step | Action | Expected System Response | Pass/ Fail | Comment |
| User enters correct credentials | User clicks on Login button | User should be logged in and system must be created | Pass | |

| | | | | |
|--|--------------------------------|---------------------------------|-----------|---------|
| Test Case: 3 | | Test Case Name: View a product | | |
| System: OTFD | | Subsystem: Product | | |
| Designed By: SMC | | Design Date: August 12,2022 | | |
| Executed By: Priya Koladiya | | Execution Date: October 17,2022 | | |
| Short Description: Test case for user registration when all details are valid. | | | | |
| Pre-conditions: User must navigate to registration page. | | | | |
| Step | Action | Expected System Response | Pass/Fail | Comment |
| User goes to page | User clicks on Register button | User should be registered | Pass | |

SOFTWARE ENGINEERING

Subject Code: 21CS2506

| Test Case: 4 | | Test Case Name: Buy a product | | |
|--|-----------------------------|--|-----------|---------|
| System: Online News Portal | | Subsystem: Login | | |
| Designed By: Priya Koladiya | | Design Date: March 02, 2017 | | |
| Executed By: Keval Shah | | Execution Date: March 02, 2017 | | |
| Short Description: Test case for logging user in when credentials are valid. | | | | |
| Pre-conditions: User must navigate to login page. | | | | |
| Step | Action | Expected System Response | Pass/Fail | Comment |
| User enters correct credentials | User clicks on Login button | User should be logged in and session must be created | Pass | |

| Test Case: 5 | | Test Case Name: Performing transaction | | |
|---|------------------------------------|---|-----------|---------|
| System: OTFD | | Subsystem: Transaction | | |
| Designed By: Keval Shah | | Design Date: March 05, 2017 | | |
| Executed By: Priya Koladiya | | Execution Date: March 05, 2017 | | |
| Short Description: Test case for editing user profile with proper details. | | | | |
| Pre-conditions: User must be logged in, and navigated to Edit Profile page. | | | | |
| Step | Action | Expected System Response | Pass/Fail | Comment |
| User edits profile and enters proper details | User clicks on Edit Profile button | The new details must be saved in the database and user should be shown the edit profile page again with the new | Fail | |

SOFTWARE ENGINEERING

Subject Code: 21CS2506

| Test Case: 6 | | Test Case Name: Log Out | | |
|--|-------------------------------|--|-----------|---------|
| System: Online transaction fraud detection | | Subsystem: Log Out | | |
| Designed By: SMC | | Design Date: August 12, 2022 | | |
| Executed By: Priya Koladiya | | Execution Date: October 17, 2022 | | |
| Short Description: Test case for logging a user out of the system. | | | | |
| Pre-conditions: User must be logged in. | | | | |
| Step | Action | Expected System Response | Pass/Fail | Comment |
| | User clicks on Log Out button | User must be logged out of the system, session should be destroyed, and the homepage must be shown | Pass | |

PRACTICAL-13

Aim: Study various tools for testing (win runner, load runner).

Description:

Automated testing is becoming more & more important for many software projects in order to automatically verify key functionality. Various areas in which automated testing tools are used:

- Reviews, walkthroughs & inspection
- Regression testing, defect management, test management, etc
- Configuration control & version management

WinRunner:

As a functional test suite, it worked with HP QuickTest Professional & supported enterprise quality assurance. It captured, verified and replayed user interactions automatically, in order to identify defects & determine whether business processes worked as designed. The software implemented a proprietary Test Script Language (TSL) that allowed customization and parameterization of user input.

HP WinRunner was originally written by Mercury Interactive. Mercury Interactive was subsequently acquired by Hewlett Packard (HP) in 2006. It is the most commonly used automated testing tool.

Features:

1. Automatic Recovery: It is possible for the user to set up various operations that will become activated if an exception event appears. The recovery manager will give you awizard that will allow you to set up a scenario for recovery.
2. Silent Installation: The unattended installation mode can be set.
3. Support For Various Environments: WinRunner includes support for Internet Explorer 6.x & Netscape 6.x, Windows XP & Sybase's Power Builder 8, in addition to 30+ environments.

SOFTWARE ENGINEERING

4. Cost Effective: WinRunner provides the most powerful productive & cost-effective solution for verifying enterprise application functionality.
5. Support For Multiple Data Combination: WinRunner has an ability to use numerous data combinations for one test. The DataDriver Wizard has been designed for automatic processing of large amount of data.
6. Multiple Verification: It can offer checkpoints for test, URL, GUI & databases this provides ability to compare expected outcomes with real ones.

Load Runner:

HP Load Runner is a software testing tool from Hewlett-Packard. It is used to test applications, measuring system behaviour & performance under load. HP Load Runner can simulate thousands of users concurrently using application software, recording & later analysing the performance of key components of the application.

Features:

1. It has excellent monitoring & analysis interface where tester can see reports easy to understand coloured charts & graphics.
2. It uses C as a default programming language. However, it also supports other languages like Java, & Visual Basic.
3. No need to install it on the server under test. It uses native monitors.
4. It has a support for most of the commonly used protocols.
5. It has GUI generated scripts which can be modified as per the requirements.
6. This tool can quickly point out the effect of the wide area network (WAN) on application reliability, performance & response ti



DRS. KIRAN & PALLAVI PATEL GLOBAL UNIVERSITY

Established Under Gujarat Private Universities (Amendment) Act, 2021 (Gujarat Act No. 15 of 2021)

KRISHNA SCHOOL OF EMERGING TECHNOLOGY & APPLIED RESEARCH (KSET)

KPGU
Vadodara

SOFTWARE ENGINEERING



DRS. KIRAN & PALLAVI PATEL GLOBAL UNIVERSITY

Established Under Gujarat Private Universities (Amendment) Act, 2021 (Gujarat Act No. 15 of 2021)

KRISHNA SCHOOL OF EMERGING TECHNOLOGY & APPLIED RESEARCH (KSET)

KPGU
Vadodara

SOFTWARE ENGINEERING
