Thadomal Shahani Engineering College

Bandra (W.), Mumbai- 400 050.

CERTIFICATE

Certify that ~~ME~~/Miss Niyati Savant of Computer Department, Semester V with Roll No. 2103156 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

Rusare 21/10/23

Teacher In-Charge

Head of the Department

Date 21 October 2023

Principal

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

CONTENTS

CONTENTS

Kusari 21/10/23

Project Title:
Banking Management System

Ujjwal Rajpurohit - 2103145

Rittika Rijhwani - 2103149

Niyati Savant - 2103156

EXPERIMENT - 1

AIM- To prepare a detailed problem statement for the selected mini-project and to identify a suitable software model for the same.

THEORY-

Title:- Banking management system

Problem Statement:-

A Banking Management System is a software application designed to facilitate and streamline the various operations and processes within a banking institution. It serves as the backbone of the bank's digital infrastructure, enabling efficient management of customer accounts, transactions, loans, and other financial activities. The system is crucial for providing a seamless and secure banking experience to customers while also assisting bank employees in their day-to-day tasks.

Banking management systems are those systems that enhance operational performance by addressing a spectrum of processes from customer account management, funds and Funds Transfer and Payments, Loan and Credit Management, Interest and Fee Calculations, Security and Authentication, Reporting and Analytics, Integration with External Systems to Scalability and Performance

A bank token system is a queuing mechanism used by banks and other financial institutions to manage customer flow efficiently. It aims to reduce waiting times and improve customer service by providing customers with numbered tokens upon arrival. Customers can then wait comfortably until their turn comes, without the need to stand in long queues.
This is how the typical banking system works:-

1. Token Issuance
2. Service Categories
3. Waiting Area
4. Token Display
5. Serving Customers
6. Optional Services

In a bid to alleviate the perennial issue of long wait times at banks, the adoption of digital tokens has emerged as a transformative solution. Traditionally, customers have had to endure frustrating queues, leading to dissatisfaction and wasted time. However, the implementation of digital tokens holds the promise of enhancing customer experience and streamlining banking operations.

Token Generation: Upon arrival at the bank, customers receive a digital token either through a mobile app or a self-service kiosk. The token is assigned based on the customer's banking needs, such as transactions, consultations, or account services.

Queue Management: The digital token system efficiently organizes the queue by sending real-time notifications to customers about their wait time and position. This reduces anxiety and enables customers to engage in other activities while waiting.

Customer Engagement: The digital token system can provide information about the bank's services, promotions, or financial literacy tips while customers wait. This keeps customers engaged and informed, contributing to a positive experience.

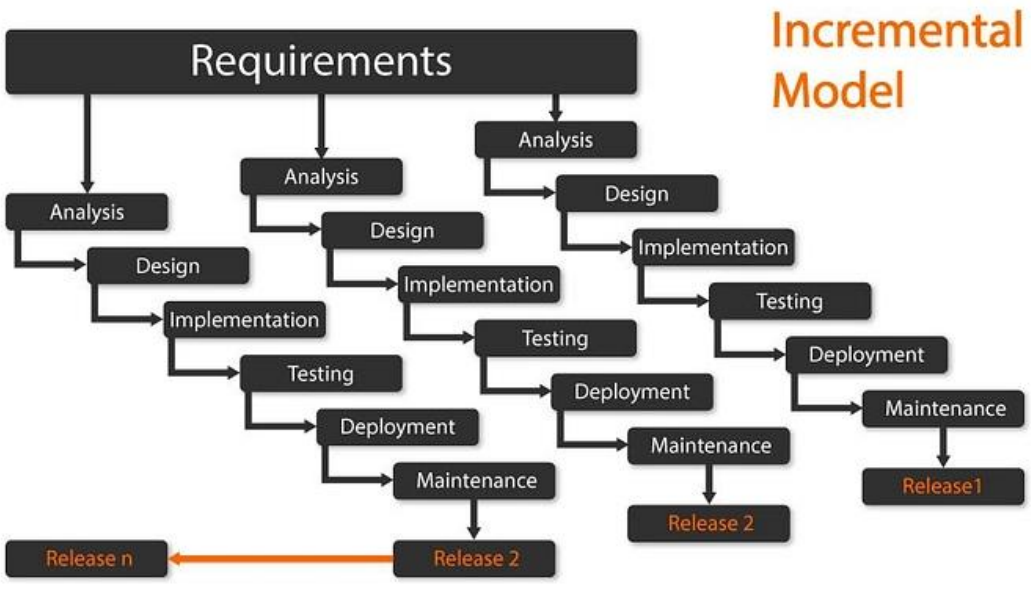
Staff Allocation: Bank staff can use the digital token data to anticipate customer requirements and allocate appropriate resources, leading to more efficient staffing and reduced wait times.

Remote Queuing: Some systems even allow customers to remotely join the queue before arriving at the bank, reducing physical presence and wait times in the banking premises.

Hence.the adoption of digital tokens in banks is a promising avenue to revolutionize the customer experience and eliminate the age-old problem of waiting in lines. By leveraging technology to streamline processes and engage customers, banks can create a more efficient and customer-centric environment that benefits both clients and financial institutions alike.

TRADITIONAL MODEL:

Incremental model



An incremental model is an iterative development approach where a software project is divided into smaller, manageable modules or increments. Each increment represents a subset of functionality that is developed, tested, and delivered incrementally. This model aims to deliver a functional piece of software at the end of each iteration, gradually adding more features and improvements with each subsequent iteration.

This approach is in contrast to the traditional waterfall model, where development proceeds linearly through distinct phases.

Stages:
The incremental model typically involves the following stages:

1. Requirements Analysis: The overall requirements of the software project are identified and divided into smaller functional requirements.
2. Design and Implementation: The design and development of the software begin, focusing on implementing the functionality identified in the first increment.
3. Testing: Testing is performed on the increment to ensure its functionality meets the requirements and to identify and fix any defects.
4. Integration: The increment is integrated with the existing software, and any issues arising from the integration are addressed.
5. Delivery: The completed increment is delivered to the customer or end-users for evaluation and feedback.
6. Feedback and Evaluation: The customer or end-users provide feedback on the delivered increment, which is used to guide the development of subsequent increments.
7. Repeat: Steps 2 to 6 are repeated for each subsequent increment, adding new features or improving existing ones.

Key characteristics and advantages:

- Phased Development: Each increment is a small version of the final product, containing a subset of the features.
- Iterative Process: Within each increment, developers work on adding, enhancing, or refining features during each iteration.
- Continuous Integration: This allows for early detection of integration issues and promotes ongoing testing and validation.
- Frequent Deliverables: At the end of each increment, a working version of the software is delivered to stakeholders, allowing them to see progress and provide feedback.
- Flexible Scope: The scope of each increment can be adjusted based on feedback, changing requirements, or new priorities.
- Risk Mitigation: By delivering functional increments at regular intervals, the incremental model reduces the risk of catastrophic project failure. It allows for early identification and mitigation of potential issues.
- Customer Involvement: Stakeholders and customers are involved throughout the development process. Their feedback is considered during each increment, ensuring that the evolving product aligns with their expectations.
- Parallel Development: Different teams can work on different increments simultaneously, which can lead to faster overall development and reduced time-to-market.
- Suitable for Large Projects: The incremental model is particularly beneficial for large and complex projects, as it provides a structured approach to manage complexity and scale.

Disadvantages:

- Complex Management: Managing multiple increments and coordinating their development, testing, and integration can be challenging.
- Potentially Higher Costs: The incremental model can lead to higher initial costs due to the need for repeated development, testing, and integration activities.
- System Integration Challenges: Integrating different increments can be complex, requiring careful planning.
- Incomplete Functionality: Early increments might lack some essential features, making it difficult for users to fully utilize the software until later increments are delivered.
- Dependency on User Feedback: Continuous user feedback is crucial, and if not obtained or acted upon effectively, it can lead to issues in subsequent increments.

Justification for using incremental model:

- Early Delivery of Core Functionality:
  Basic banking operations, such as customer registration, account management, and transaction processing, can be developed and delivered in the initial increments. This provides stakeholders with a working version of the system sooner, helping to meet urgent business needs.

- Rapid Feedback and Adaptation:
  By delivering working increments at regular intervals, stakeholders, including bank staff and customers, can provide feedback on the system's functionality and usability. This enables developers to quickly make adjustments and improvements based on real-world usage and requirements, leading to a system that better aligns with user needs.

- Reduced Risk and Improved Quality:
  With each increment, developers can identify and address potential issues, bugs, and design flaws. This iterative process leads to higher quality code and a more stable system over time.

- Flexibility and Adaptability:
  The incremental model accommodates changing requirements more effectively than linear development models. In the banking sector, regulatory changes, evolving customer expectations, and emerging technologies are common. The ability to incorporate these changes incrementally helps the banking management system remain up-to-date and compliant.

- Partial Deployment:
  With incremental development, it's possible to deploy and use specific features or modules of the banking management system as they become available. This allows the bank to start benefiting from the new system earlier, even if the entire system is not fully completed. For example, online banking features could be rolled out incrementally.

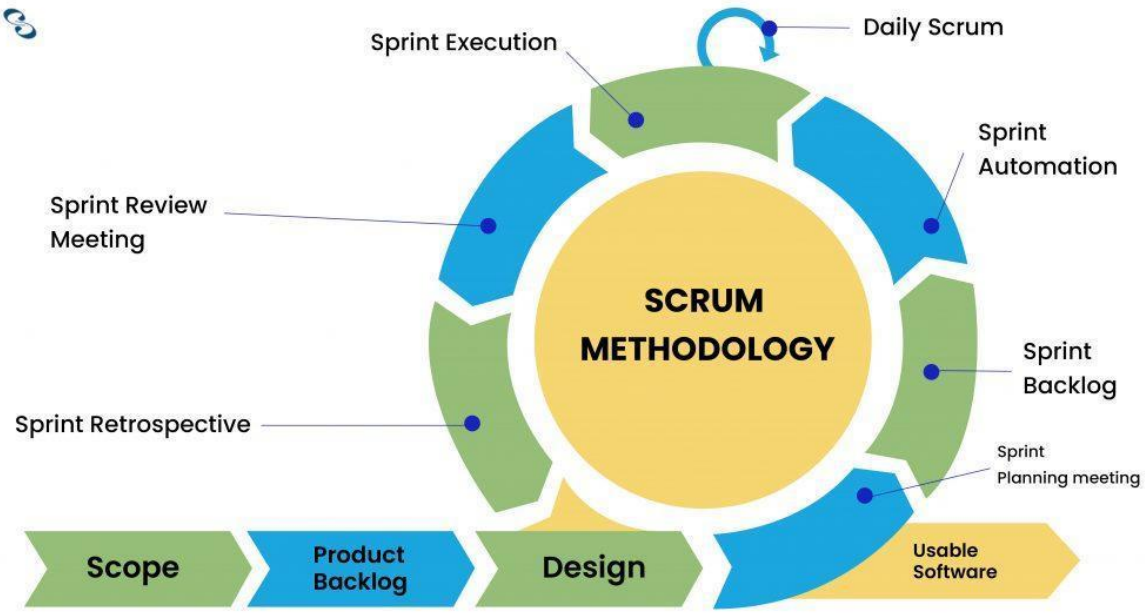
- Easier Resource Management:
  By breaking down the development process into smaller increments, resource allocation becomes more manageable. This can help in terms of staffing, budgeting, and overall project management. The development team can focus on delivering specific functionalities without feeling overwhelmed by the entire scope of the project.

- Support for Complex Systems:

Banking management systems are often complex, involving various interconnected modules and functionalities. Incremental development is particularly suitable for such systems, as it allows teams to focus on individual components while ensuring their integration within the larger framework.

AGILE MODEL:

Scrum model



Scrum is an agile development methodology used in the development of Software based on iterative and incremental processes. Scrum is an adaptable, fast, flexible, and effective agile framework that is designed to deliver value to the customer throughout the development of the project.

The primary objective of Scrum is to satisfy the customer's need through an environment of transparency in communication, collective responsibility and continuous progress. The development starts from a general idea of what needs to be built, elaborating a list of characteristics ordered by priority (product backlog) that the owner of the product wants to obtain.

Stages:
The scrum model typically involves the following stages:

1. Product Backlog Creation: The Scrum work process starts with a product owner, who, in collaboration with the team, develops user stories or requirements of the project. A product backlog is a prioritized list of all the product requirements or user stories that a scrum team maintains for a product.
2. Release Backlog: After determining which user stories will go into a particular release, the development team estimates the time duration needed to complete each item. Once the release planning has been completed, the user stories are then selected for a sprint.

3. Sprint Backlog Creation: A Sprint is a predefined timeframe within which the team performs a set of tasks from the Backlog. The duration of each Sprint lasts 2-4 weeks. Each Sprint takes a manageable chunk of the release backlog and gets it a ship-steady state. A set of product backlog items that must be delivered within a single sprint iteration is called a Sprint backlog.
4. Working on Sprint and Scrum Meetings: After the user stories for the current phase are selected, the development process begins. For tracking the current working process, a task board is commonly used, which represents particular user stories with a description of tasks needed for implementation. Further, daily scrums are conducted by development teams to monitor the progress made towards the Sprint Goal and progress performed in the Sprint Backlog, to adjust the plan for the rest of the Sprint.
5. Burndown Charts: The progress of the team is tracked through a burndown chart. It provides a day-by-day measure of the work that remains in a given sprint or release. The slope of the graph (burndown velocity) is calculated by comparing the number of hours worked to the original project estimation and displays the average rate of productivity for each day
6. Testing and Product Demonstration: If all the user stories are completed then the sprint backlog is also completed, which means a sprint completion. A sprint review is held after sprint completion, where the working software is demonstrated and presented for acceptance to the customers. Based on their feedback, stakeholders decide any further changes be enacted in the project.

Key characteristics and advantages:

- Flexibility and Adaptability: Scrum's iterative approach allows teams to adapt to changing requirements, market dynamics, and customer needs. This flexibility ensures that the project remains aligned with the current context.
- Early and Continuous Delivery: Scrum's time-boxed sprints result in frequent releases of potentially shippable increments. This enables stakeholders to see real progress and make informed decisions about the project's direction.
- Transparency and Visibility: Scrum promotes transparency by making work, progress, and issues visible to all team members and stakeholders. This transparency enhances collaboration and accountability.
- Risk Mitigation: Incremental development and regular reviews allow teams to identify and address issues early in the development process, reducing the risk of delivering a product that doesn't meet requirements or quality standards.
- Improved Communication: Scrum emphasizes regular communication through events like daily stand-ups, sprint planning, and sprint reviews. This fosters better understanding, collaboration, and alignment among team members.
- Continuous Improvement: Sprint retrospectives encourage teams to reflect on their processes and identify opportunities for improvement. This iterative improvement cycle leads to more efficient and effective practices.
- Focus on Value: Scrum encourages prioritization based on value. This ensures that the most valuable features and improvements are delivered first, maximizing the return on investment for stakeholders.
- Clear Roles and Responsibilities: Scrum defines distinct roles (Product Owner, Scrum Master, Development Team) and responsibilities, reducing ambiguity and enhancing accountability within the team.
- Effective Time Management: Scrum's time-boxed approach ensures that meetings and activities have well-defined durations, helping teams manage their time effectively and maintain a sustainable pace of work.

- Stakeholder Engagement: Scrum encourages regular involvement of stakeholders, leading to better alignment between the development team and those invested in the project's success.
- Cross-Functional Collaboration: Scrum emphasizes cross-functional teams with diverse skill sets. This collaboration fosters creativity and problem-solving by bringing different perspectives to the table.

Disadvantages:

- Scrum frameworks do not allow changes in their sprint.
- It is not a fully described model. If you want to adopt it you need to fill in the framework with your own details like Extreme Programming(XP), Kanban, and DSDM.
- It can be difficult for Scrum to plan, structure, and organize a project that lacks a clear definition.
- The daily Scrum meetings and frequent reviews require substantial resources.

Justification for using the Scrum model:

- Evolving Requirements: Banking software often experiences changing regulatory requirements, customer demands, and technological advancements. Scrum's iterative and flexible nature allows the team to adapt quickly to evolving requirements, ensuring that the software remains up-to-date and compliant.

- Customer-Centric Approach: Scrum encourages continuous feedback from stakeholders, including customers and end-users. In the context of banking software, this enables the development team to prioritize features and functionalities that directly address customer needs, resulting in a more user-friendly and valuable product.

- Frequent Deliverables: Scrum divides the project into time-boxed sprints, each resulting in a potentially shippable increment. This approach allows the banking software project to deliver valuable features and improvements at the end of each sprint, enabling stakeholders to see tangible progress and provide early feedback.

- Risk Mitigation: Banking software involves critical financial and security aspects. Scrum's incremental delivery and regular reviews mitigate the risk of developing a product that doesn't align with stakeholders' expectations. Early identification of potential issues and continuous testing contribute to improved quality and security.

- Transparency and Collaboration: Scrum emphasizes transparency through open communication and visibility of work. This is crucial in a banking context where collaboration between business stakeholders, compliance teams, developers, and testers is essential to ensure accurate and secure functionality.

- Adaptation to Market Changes: Banking is a rapidly evolving industry with changing customer preferences and competitive landscapes. Scrum's ability to prioritize work based on real-time feedback allows the development team to respond to market changes more effectively.

- Quality Focus: Each sprint in Scrum includes tasks like sprint planning, daily stand-ups, and sprint reviews. These events promote a focus on quality, regular testing, and early identification of issues, ensuring that the banking software maintains a high standard of reliability and performance.

- Flexibility for Regulatory Compliance: Banking software must adhere to various regulations and standards. Scrum's adaptive nature enables the team to incorporate compliance requirements into the development process while still responding to changes as needed.

- Motivated and Empowered Teams: Scrum promotes self-organizing teams, giving them ownership of the development process. This empowerment can lead to higher job satisfaction, increased creativity, and improved collaboration, all of which are beneficial for complex software projects.

- Continuous Improvement: Scrum's sprint retrospectives encourage the team to reflect on their processes and identify areas for improvement. This fosters a culture of continuous learning and enhancement, which can lead to more efficient development practices and a better end product.

EXPERIMENT 2

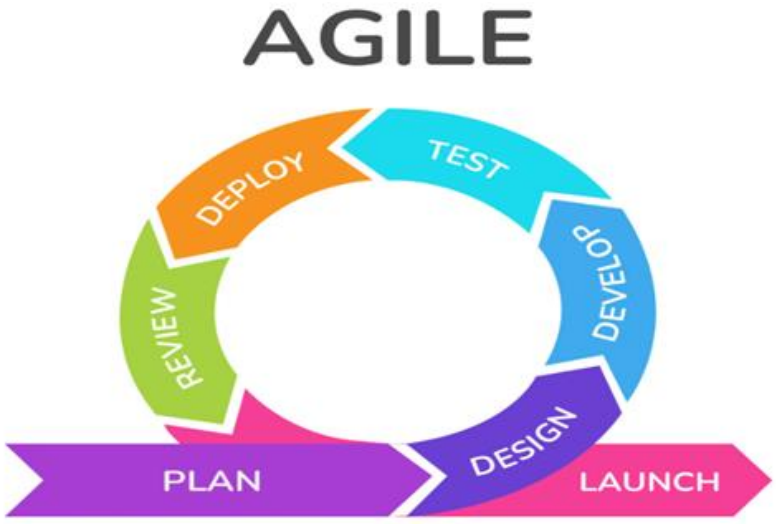
Aim: Application of Agile Process model on the project using JIRA

Theory:

Agility has become today's buzzword when describing a modern software process. Everyone is agile. An agile team is a nimble team able to appropriately respond to changes.Change is what software development is very much about. Changes in the software being built, changes to the team members, changes because of new technology, changes of all kinds that may have an impact on the product they build or the project that creates the product. An agile team recognizes that software is developed by individuals working in teams and that the skills of these people, their ability to collaborate is at the core for the success of the project.The pervasiveness of change is the primary driver for agility. Software engineers must be quick on their feet if they are to accommodate the rapid changes.

Principles of Agile Process

- Customer satisfaction
- Deliver Working software
- Measure of progress
- Late changes are welcome
- Face to face communication
- Motivated individuals
- Technical excellence
- Simplicity
- Regular Adoption



Kanban

Agile Kanban is Agile Software Development with Kanban approach. In Agile Kanban, the Kanban board is used to visualize the workflow. The Kanban board is normally put up on a wall in the project room. The status and progress of the story development tasks is tracked visually on the Kanban board with flowing Kanban cards.

Kanban board is used to depict the flow of tasks across the value stream. The Kanban board –

- · Provides easy access to everyone involved in the project.
- · Facilitates communication as and when necessary.
- · Progress of the tasks are visually displayed.
- · Bottlenecks are visible as soon as they occur.

The tasks and stories are represented by Kanban cards. The current status of each task is known by displaying the cards in separate columns on the board. The columns are labeled as To Do, Doing, and Done. Each task moves from To Do to Doing and then to Done.

Kanban Board is updated on a daily basis as the team progresses through the development.

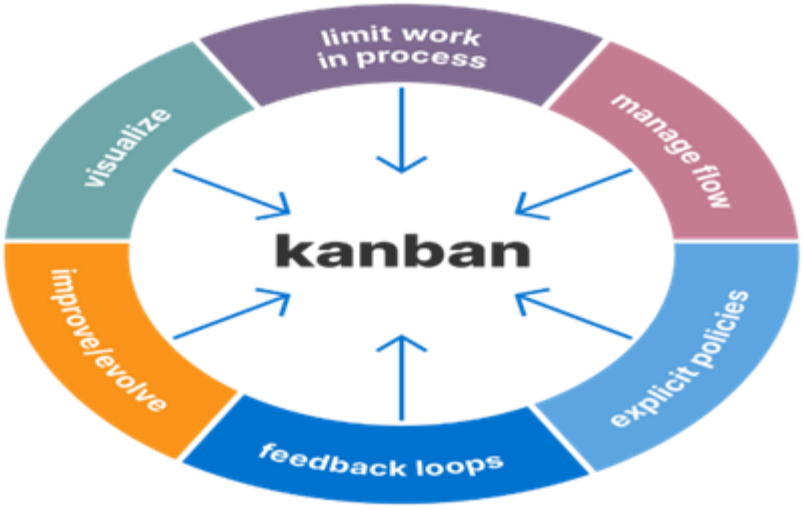
Limiting WIP: The label in the Doing column also contains a number, which represents the maximum number of tasks that can be in that column at any point of time. i.e., the number associated with the Doing column is the WIP (Work-In-Progress) Limit.

Pull Approach: Pull approach is used as and when a task is completed in the Doing column.Another card is pulled from the To Do column.

Self-directing: In Agile Development, the team is responsible for planning, tracking, reporting and communicating in the project. Team is allowed to make decisions and is accountable for the completion of the development and product quality. This is aligned to the characteristic of empowerment of the team in Kanban.

Continuous Flow:

In Agile development, there is no gate approach and the work flows across the different functions without wait-time. This contributes in minimizing the cycle time characteristic.



How to set up a simple Kanban system for a software development team.

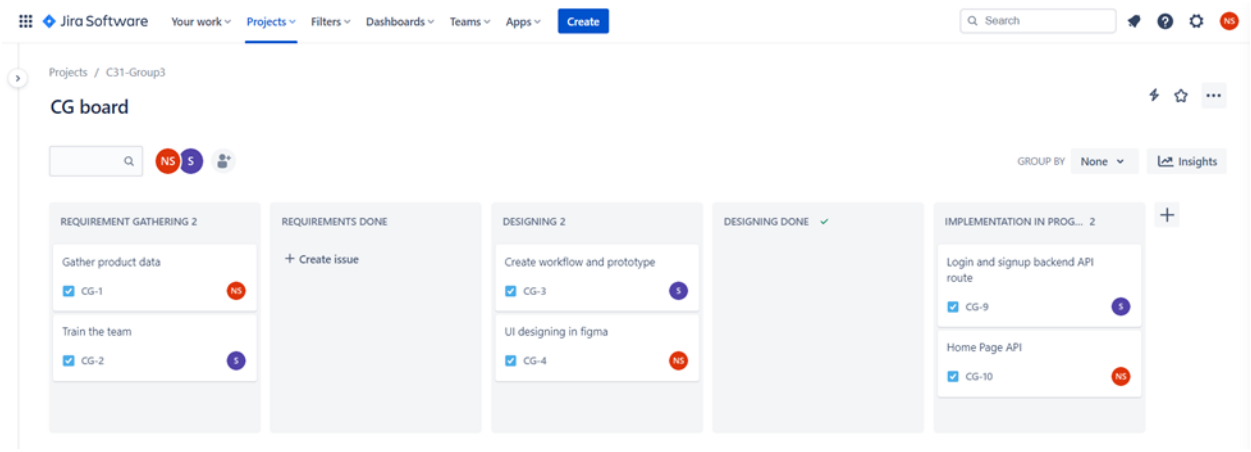
1. Define a work process flow
2. Lay out a visual Kanban board
3. Decide on limits for items in queue and work in progress
4. Place prioritized goals on the left column of the board
5. Start the board by placing stories or features in queue
6. Move features through the process flow as work is completed
7. Use the dates on the cards to calculate cycle time

Advantages of Kanban board

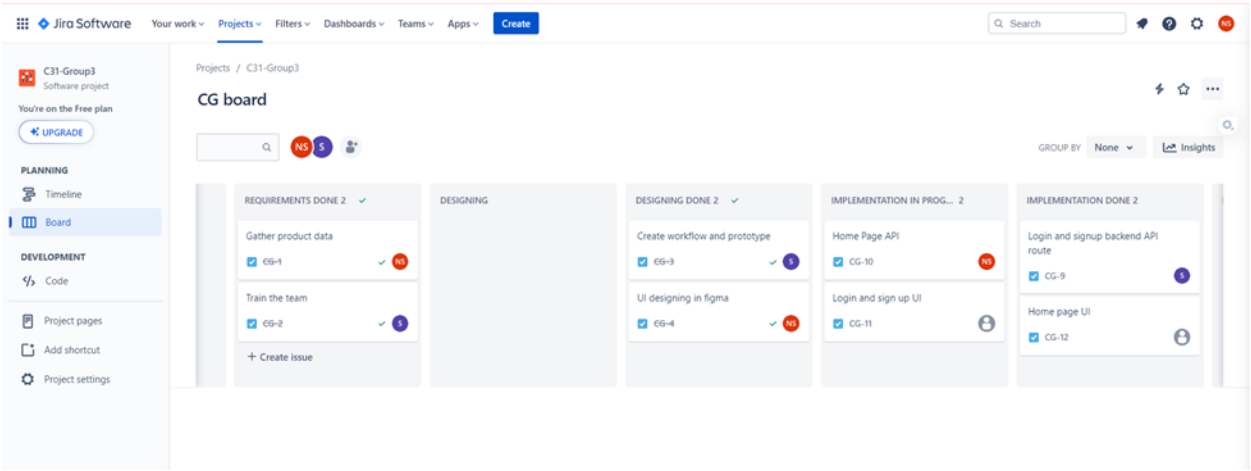
- Empowerment of Team
    a. Team is allowed to take decisions as and when required.
    b. Team collaboratively resolves the bottlenecks.
    c. Team has access to the relevant information.
    d. Team continually communicates with customer.
- Continuous Delivery

a. Focus on work completion.
b. Limited requirements at any point of time.
c. Focus on delivering value to the customer.
d. Emphasis on whole project.

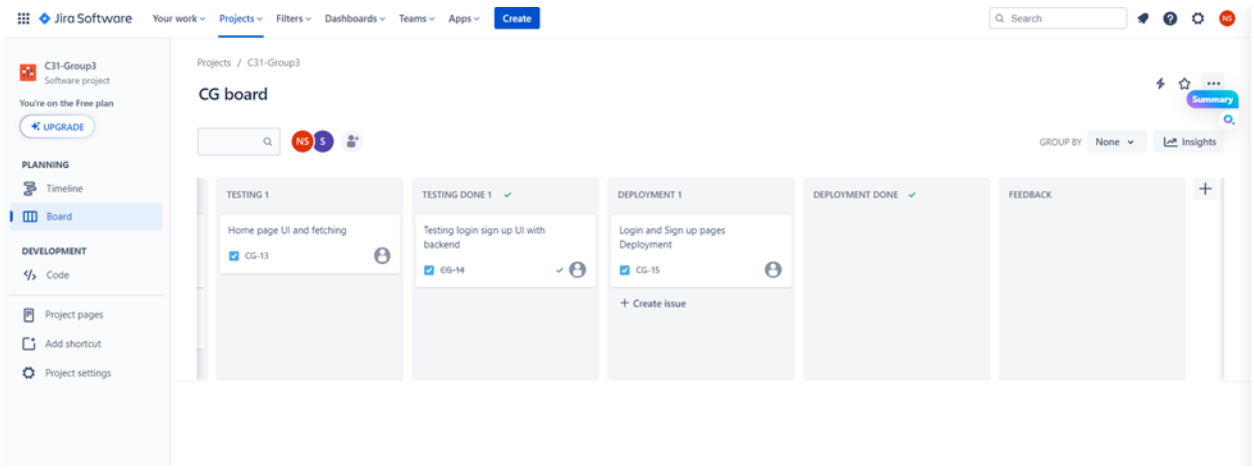
PLANING AND REQUIREMENTS GATHERING PHASE

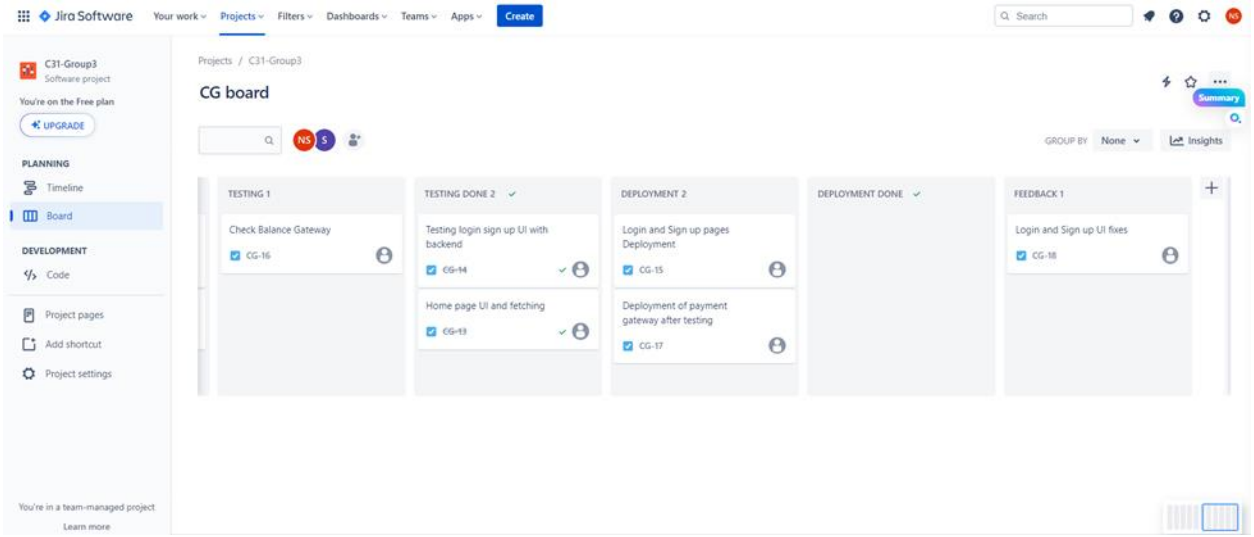


DESIGNING AND IMPLEMENTATION IN PROGRESS PHASE

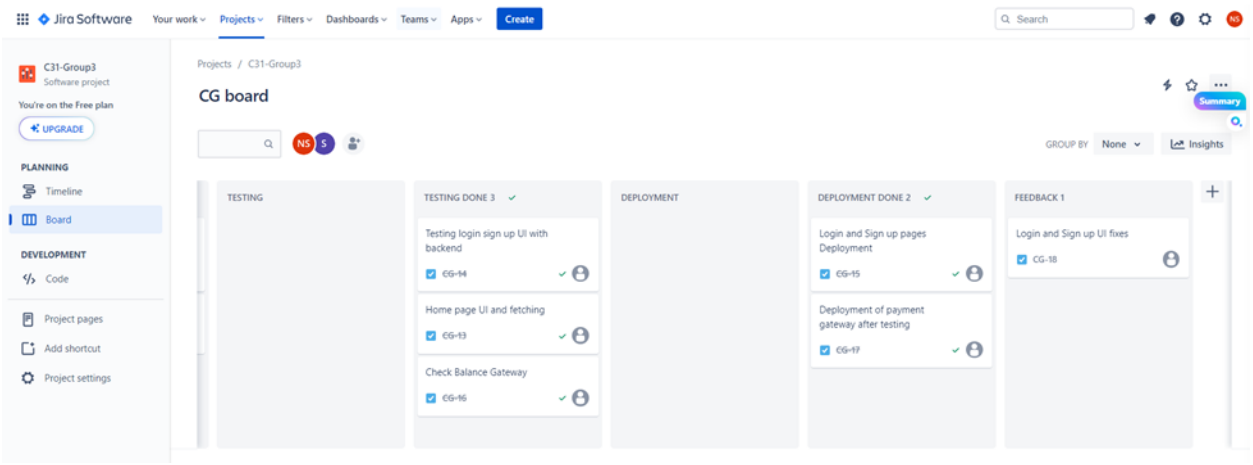


TESTING PHASE





DEPLOYMENT PHASE WITH SOME FEEDBACK DONE



Software Requirements Specification

For

Banking Management System

Version 1.0 approved

Prepared by

Rittika Rijhwani

Niyati Savant

Ujjwal Rajpurohit

Thadomal Shahani Engineering College

10th August, 2023

Table of Contents

Revision History

Name	Date	Reason For Changes	Version

1. Introduction

1.1 Purpose

The purpose of the software is to provide the users a platform for all their banking needs like withdrawing money,depositing money,fund transfer ,getting loan status,investment plans.This will help customers save time as they won't need frequent trips to their banks .

1.2 Document Conventions

Important points have been underlined and the headings and subheadings are written in bold font to provide emphasis. The points in all sections have been written in the order of decreasing priority, so that important points are not missed out. Only standard abbreviations understood by the developers of the application are used.

1.3 Intended Audience and Reading Suggestions

The intended audience is the team of developers who will be designing and implementing the Banking Management System. Also, the document is to be utilized by the testing team who will be testing and evaluating the performance and design of the application. The document consists of all the necessary information that will be required by the team of software engineers who will be working on the project.

1.4 Product Scope

The Banking Property Management System aims to simplify the banking process for its customers . To make it accessible on all electronic devices the application will be a web-based application and will be mobile responsive as well. The application will provide all necessary functionalities for checking balance,viewing transaction slips,viewing loan options. There will also be functionalities for home service,fund transfers,card services. We are also giving the option for digital tokens so that the customers need not wait in long lines.

1.5 References

Websites:

- Django Documentation: https://docs.djangoproject.com/en/3.1/
- Bootstrap Framework: https://getbootstrap.com/

2. Overall Description

2.1 Product Perspective

The Bank Management System is intended to provide an online platform for all the banking needs to get rid of all the hassle of waiting in lines or spending 2 hours on a 15 minute job.. The features of the application will allow the users to get loans,scan and pay,transfer funds and check insurance. This eliminates the need of physically visiting the bank , thus saving time for both the customers and the bank officials .

2.2 Product Functions

- By using the Bank Management Management System, the need of visiting the bank is eliminated.
- Users can cater to their banking needs from the comfort of their home.
- Saves time for both the consumers and employees.
- Multiple available options for tasks to choose from.
- Allows the employees focus more on the tasks of higher priority

2.3 User Classes and Characteristics

The users of the application can be classified into two types.The ones which are operating the account on a personal level and the ones that operate the accounts of the business. Both the users can access all the functionality of the application.

2.4 Operating Environment (OE)

Since the application is a web application it can work on any device having a browser.

- Device: Mobile Phone, Computer, Laptops, Tablets.
- Operating System: Windows, Linux distributions, Mac OS, Android
- RAM: 128 MB or more
- Disk Space: 20 MB or more.
- Browsers: Mozilla Firefox 30+, Google Chrome 27.0+, Microsoft Edge. Other browsers can also be used.
- Internet connection: Strong internet connection with speed of at least 1 Mbps for best experience.

2.5 Design and Implementation Constraints

CO-1:

The time allotted for this project is atmost 5 months.

CO-2:

The front end of the application will be made using HTML, CSS and JavaScript.

CO-3:

Python will be used as the language for the backend of the application and MySQL will be used for the database of the application.

CO-4:

The website will be in English language. Users who do not know English will face difficulties in using the website.

2.6 User Documentation

Appropriate instructions will be provided at every step in the application to ensure the users do not face any difficulties while using the application. In future, we plan to add a chatbot to guide users in case they face any difficulties. Instructions will be given while filling out forms, adding photos and important details. Proper error messages will be displayed in case the user inadvertently fills wrong information or makes any mistake while using the application.

2.7 Assumptions and Dependencies

AS-1:

The application supports only English language. We assume the users of the application will be well versed with English.

AS-2:

The users of the application should have basic knowledge of uploading images and be ready with soft copy of the documents.

DE-1:

The application will require Django web framework as a dependency, since we use Python as the backend language.

DE-2:

Bootstrap Framework will be used for the front end of the application.

3. External Interface Requirements

3.1 User Interfaces

UI-1:

The website will start with a landing page. User can sign in using their m-pin or username and password .After signing in user can see their account informations.

UI-2:

There will be a navigation bar at the top of the web page which will help users to navigate to different web pages like Home ,Loans ,Digital Tokens , Invest , Insure , Online Shopping , Cards , Accounts , Help ,Contact-us etc .

UI-3:

.Iinstructions will be provided to the users on top of forms to be filled.

Steps to will be provided to the users to take digital tokens and access other features.

UI-4:

The application will feature alert notifications and pop-ups that trigger when users make errors during interactions. These prompts will offer immediate feedback, guiding users toward rectifying their mistakes. This proactive approach enhances user understanding and reduces frustration. By addressing errors promptly, the system promotes a smoother and more intuitive user experience.

UI-5:

The interface will be responsive for all screen sizes as much as possible to provide the users a seamless experience.The interface's responsiveness will optimize it for diverse screen sizes, prioritizing a smooth user experience. By adapting layouts and content, users will enjoy seamless interactions across devices. This approach enhances usability and engagement

3.2 Hardware Interfaces

N/A

3.3 Software Interfaces

- Browsers: Google Chrome 27.0+ ,Mozilla Firefox 30+, Brave,

  Microsoft Edge are the preferred browsers.

- Operating System: Android, ios, Windows 7, 8, 10, Mac OS,

  Linux distributions.

3.4 Communication Interfaces

The application will be using HTTPS protocol.Communication between the client and server should utilize a REST-compliant web service and must be served over HTTP Secure (HTTPS).The client-server communication must be stateless. A uniform interface must separate the client roles from the server roles.

4. System Features

4.1 Opening a Bank Account

4.1.1 Description and Priority:

This feature enables users to open new bank accounts online, providing them with a convenient way to initiate the account creation process without visiting a physical branch. It is a high priority feature.

4.1.2 Response Sequences:

Upon selecting the "Open New Account" option, the user is led to the account initiation section through the navigation bar on the home page. Here, essential information like name, age, and valid documentation needs to be provided by the user via a submission form.

The system sends a confirmation email with the account details and guidelines for account activation once the details are verified.

4.1.3 Functional Requirements:

REQ-1: Cryptography libraries and API’s are used for data encryption to encrypt all sensitive user data such as bank account no’s , passwords etc.

REQ-2: Optical Character Recognition (OCR) technology extracts text and data from scanned images or documents, making them searchable and machine-readable. It can be used to verify the text content of documents

4.2 Payments

4.2.1 Description and Priority:

This feature allows users to perform various types of financial transactions, including bank transfers, NEFT and UPI.

4.2.2 Response Sequences:

Upon arriving at the homepage, the user is guided to navigate to the payments page via the navigation bar. Here, fundamental information such as the sender's and receiver's bank account numbers, the transfer amount, and the chosen payment method are required to be entered. System provides a confirmation message and updates the user's account balance accordingly, if payment is successful.

4.2.3 Functional Requirements:

REQ-1: APIs, encryption protocols (SSL/TLS), tokenization are the technologies required to implement payment gateway, the most prominent requirement of this feature.
REQ-2: Optical Character Recognition (OCR) technology extracts text and data from scanned images or documents, making them searchable and machine-readable. It can be used to verify the text content of documents
REQ-3: Secure Communication Protocols: Secure communication protocols like HTTPS, TLS, and SSL are crucial for encrypting data during online transactions

4.3 Transaction Cards

4.3.1 Description and priority:

This feature involves the management of transaction cards, including debit/credit cards, allowing users to apply, activate, and manage their cards. It is of medium priority.

4.3.2 Response Sequences:

Upon reaching the homepage, users are guided to the transaction cards section through the navigation bar. Here, they are presented with two primary choices: initiating an application for a new card or modifying details and limits for a card they already possess.

4.3.3 Functional Requirements:

REQ-1: Machine learning algorithms and anomaly systems are used for fraud detection and prevention which is very important in case of any card updations and transactions and for behavioural analysis to detect or identify suspicious transactions.

REQ-2: Near Field Communication Chips or NFC chips and contactless payment terminals are used to enable contactless payments using smartphones or cards

4.4 Login

4.4.1 Description and priority:

This feature is crucial for user authentication and access to their banking account. It is of high priority.

4.4.2 Response Sequences:

Users can access the login page either through the homepage where they are prompted to enter their registered username and password. The system then validates these credentials, ensuring the security of user accounts. Upon successful validation, users are seamlessly redirected to their personalized account dashboard, where they can access and manage their information and preferences with ease.

4.4.3 Functional Requirements:

REQ-1: Implement secure encryption for storing and transmitting user login information.

REQ-2: Implement account lockout mechanisms after a certain number of failed login attempts to enhance security.

REQ-3: Enable multi-factor authentication (MFA) for an additional layer of security.

REQ-4: Log and monitor login activities for security auditing purposes.

4.5 View Account Details

4.5.1 Description and priority:

This feature allows users to access and review their account information, including balances, transactions, and account settings. It is of medium priority.

4.5.2 Response Sequences:

Once users log in to their account, they gain access to the account dashboard, where they have the option to select "Account Details." Within this section, users can view essential information pertaining to their account. This includes an overview of their account balance, a record of their recent transactions, providing insight into their financial

activity. It also allows users to review and update their personal account information, ensuring that their profile remains accurate and up-to-date.

4.5.3 Functional Requirements:

REQ-1: Display account balance and transaction history in real-time.

REQ-2: Allow users to filter and search for specific transactions within a date range.

REQ-3: Provide options to update personal account details (e.g., contact information) if needed.

REQ-4: Implement role-based access control to ensure only authorized users can access account details.

4.6 Request for Checkbook or Token

4.6.1 Description and priority:

This feature allows users to request a checkbook or security token. It is of medium priority.

4.6.2 Response Sequences:

Upon logging into their account, users gain access to their personalized account dashboard. Among the available options, users have the choice to "Request Services." Within this section, users are presented with two primary service options: requesting a checkbook or acquiring a security token. To proceed, users simply select their preferred service and provide the requisite details, such as specifying the number of checkbooks needed or providing a reason for requesting a security token. Once all necessary information is entered, users confirm their request, ensuring an efficient process for accessing these financial services.

4.6.3 Functional Requirements:

REQ-1: Store and track checkbook and token requests for auditing purposes.

REQ-2: Implement a notification system to update users on the status of their requests.

REQ-3: Allow users to cancel pending requests if needed.

REQ-4: Verify user identity before processing requests for security tokens.

4.7 Withdraw or Deposit Money

4.7.1 Description and priority:

This feature enables users to perform financial transactions such as withdrawing or depositing money into their accounts. It is of high priority.

4.7.2 Response Sequences:

Users begin by logging into their account, gaining access to their personalized dashboard. Depending on their needs, users can then decide whether to withdraw or deposit money. Upon making their selection, they proceed to input the desired amount and select the specific account involved in the transaction. After double-checking the details, users finalize their decision by confirming the transaction.

4.7.3 Functional Requirements:

REQ-1: Ensure secure and encrypted money transfer transactions.

REQ-2: Implement daily transaction limits to prevent unauthorized transactions.

REQ-3: Provide transaction history with transaction IDs and timestamps for reference.

REQ-4: Send transaction confirmation emails or SMS to users.

4.8 Check Statement

4.8.1 Description and priority:

This feature allows users to access and download their account statements for a specified period. It is of medium priority.

4.8.2 Response Sequences:

Within the interface, users can further refine their preferences by specifying the desired account and specifying the date range for the statement they wish to retrieve. This user-centric flexibility allows individuals to retrieve relevant financial information with ease. Once these selections are made, users can view and download their account statements, available in both PDF format.

4.8.3 Functional Requirements:

REQ-1: Generate accurate and complete account statements.

REQ-2: Allow users to select different account types if they have multiple accounts.

REQ-3: Implement access controls to ensure only authorized users can access account statements.

REQ-4: Store historical account statements securely for a predefined period.

5. Other Nonfunctional Requirements

5.1 Performance Requirements

- Scalability:

The system should be designed to scale horizontally or vertically to accommodate an increase in the number of users and transactions without a significant degradation in performance.

- The system should be optimized to minimize the impact of network latency on user interactions, especially for online banking activities.

5.2 Safety Requirements

- Regular automated backups of the system's database and configuration settings must be performed, and a robust data recovery plan should be in place to ensure minimal data loss in the event of system failures or disasters.
- Role-based access control should be implemented to ensure that users can only access functionalities that they are authorized for based on their roles (e.g., customer, bank teller, manager).
- All sensitive data, including customer information, transaction details, and account balances, must be encrypted during transmission and storage using industry-standard encryption protocols.

5.3 Security Requirements

- The passwords of the users are hashed and then stored in the database so that no person can access the passwords of the users.
- The passwords should be at least 8 characters long and must have at least one uppercase character, one digit and at least one special

symbol.

- The website should HTTPS protocol for security.

- Users, especially those accessing sensitive operations or information, should be required to authenticate using multiple factors (e.g., password, OTP, biometric) to enhance account security.

5.4 Software Quality Attributes

5.4.1 Usability:

The user interface should be simple to use and not cluttered with a lot of information.

5.4.2 Reliability:

- The system should be available to users at all times, with minimal downtime for maintenance or unforeseen issues.
- All data transactions and updates should be accurately and reliably recorded to prevent data corruption.

- The system should be available to users at all times, with minimal downtime for maintenance or unforeseen issues.

5.4.3 Maintainability:

The system's architecture should be organized into modular components, making it easier to modify or update specific parts without affecting the whole system.

5.4.4 Testability:

The system should be designed in a way that allows thorough testing of all functionalities, reducing the chances of bugs and defects in production.

5.5 Business Rules

Different user roles (customer, teller, manager, administrator) have varying levels of access to system features and data.

Customers can open multiple types of accounts (savings, checking, fixed deposit, etc.).

Each account must have a unique account number.

Accounts can only be closed with a zero balance.

Transactions require proper authorization based on the user's role.

Withdrawals cannot exceed the available balance.

Alerts are sent for suspicious or large transactions.

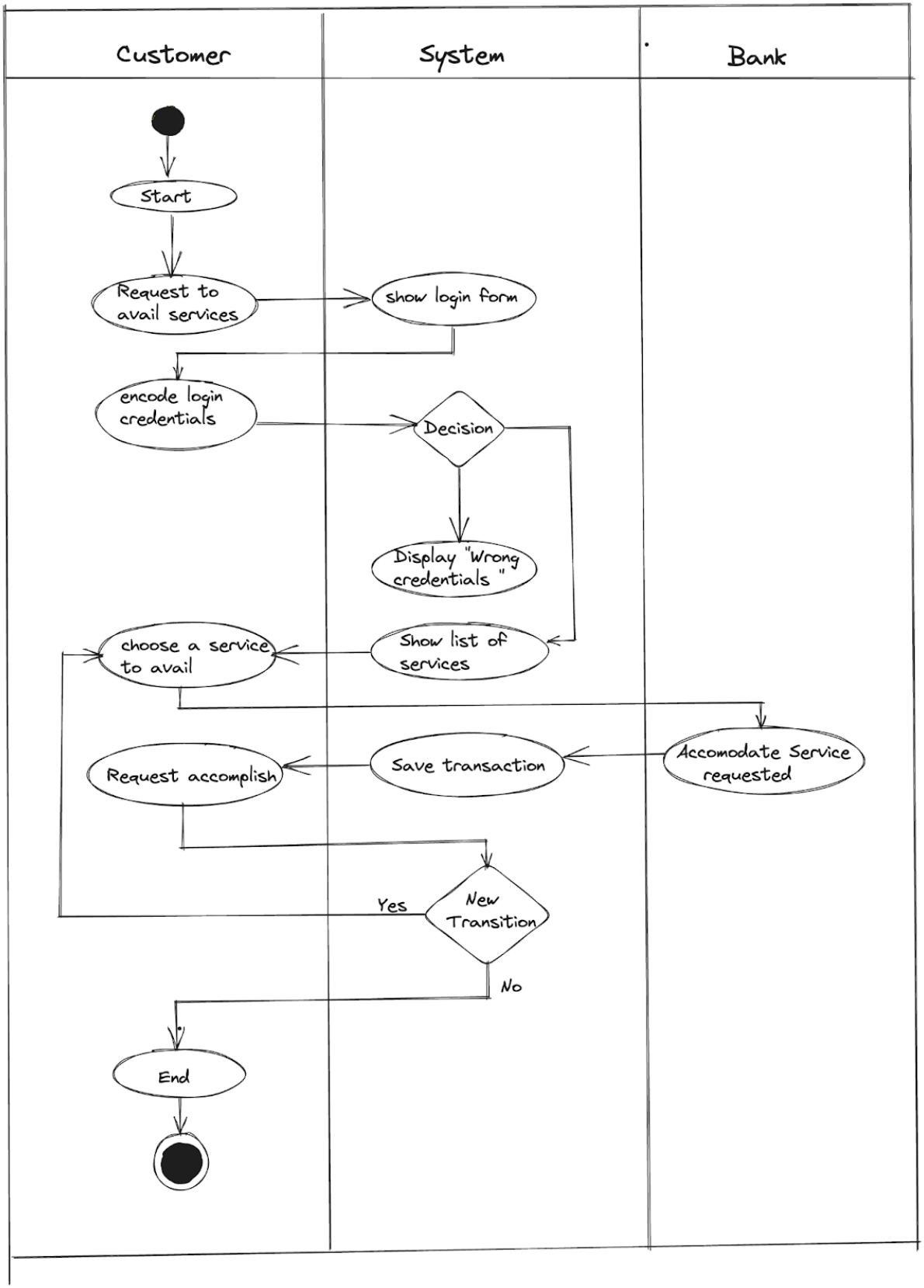
6. Other Requirements

Appendix A: Glossary

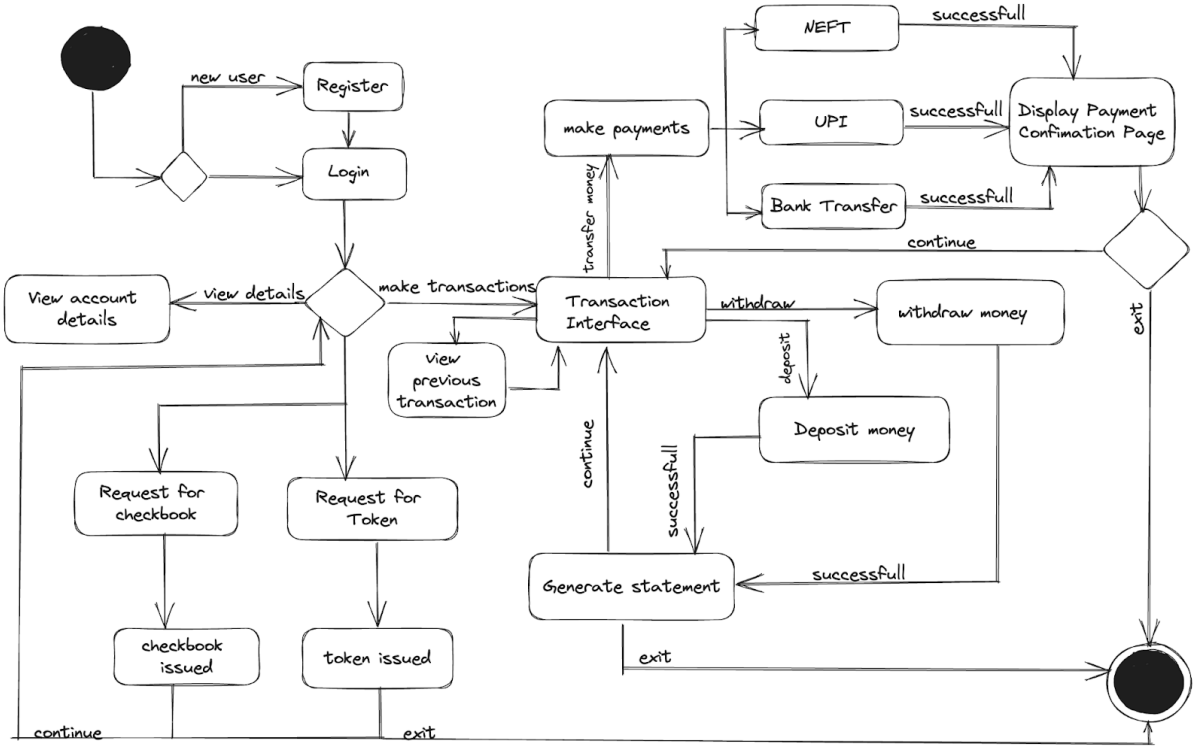
- HTTPS: Hypertext Transfer Protocol Secure
- API: Application Programming Interface
- GUI: Graphical User Interface

Appendix B: Analysis Models

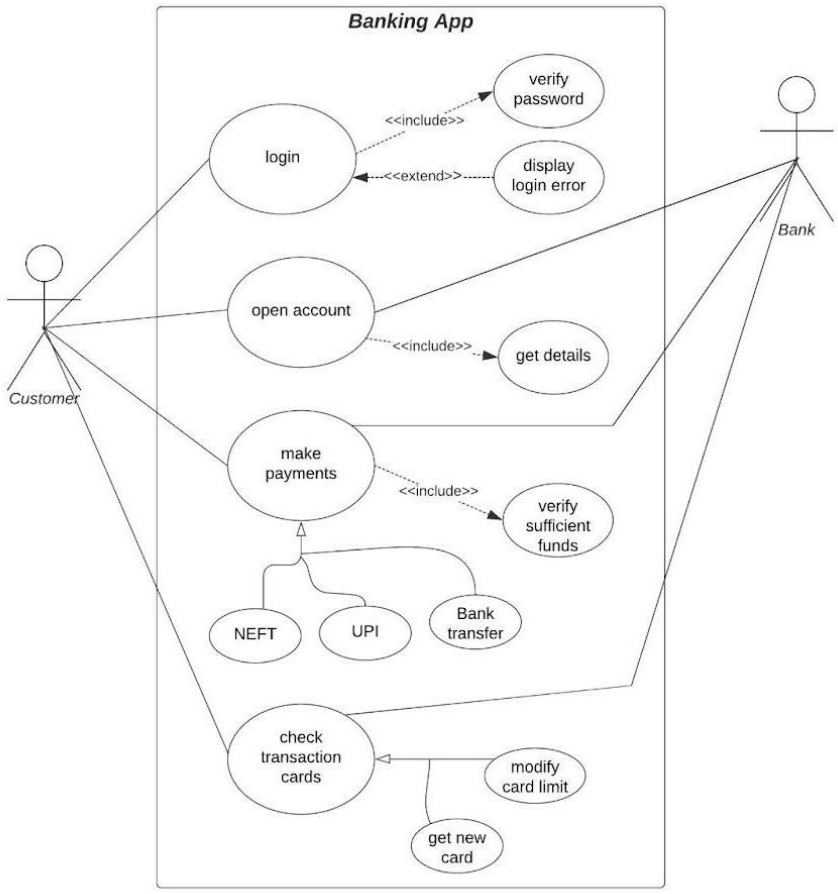
Activity Diagram of the Application



State Diagram of the Application



Use Case Diagram for the application


Banking App
verify password
<<include>>
login
<<extend>>
display login error
Bank
open account
<<include>>
get details
Customer
make payments
<<include>>
verify sufficient funds
NEFT
UPI
Bank transfer
check transaction cards
modify card limit
get new card

EXPERIMENT 4

AIM : Develop Data Flow Diagram (DFD) for the project (Smart Draw,Lucid Chart).

THEORY:

A Data Flow Diagram (DFD) is a visual representation used in software engineering to illustrate the flow of data within a system or process. It employs various symbols and arrows to depict the movement of data from its sources, through various processing steps, and ultimately to its destinations or storage locations. DFDs are valuable tools for understanding the data interactions within a system and are used to:

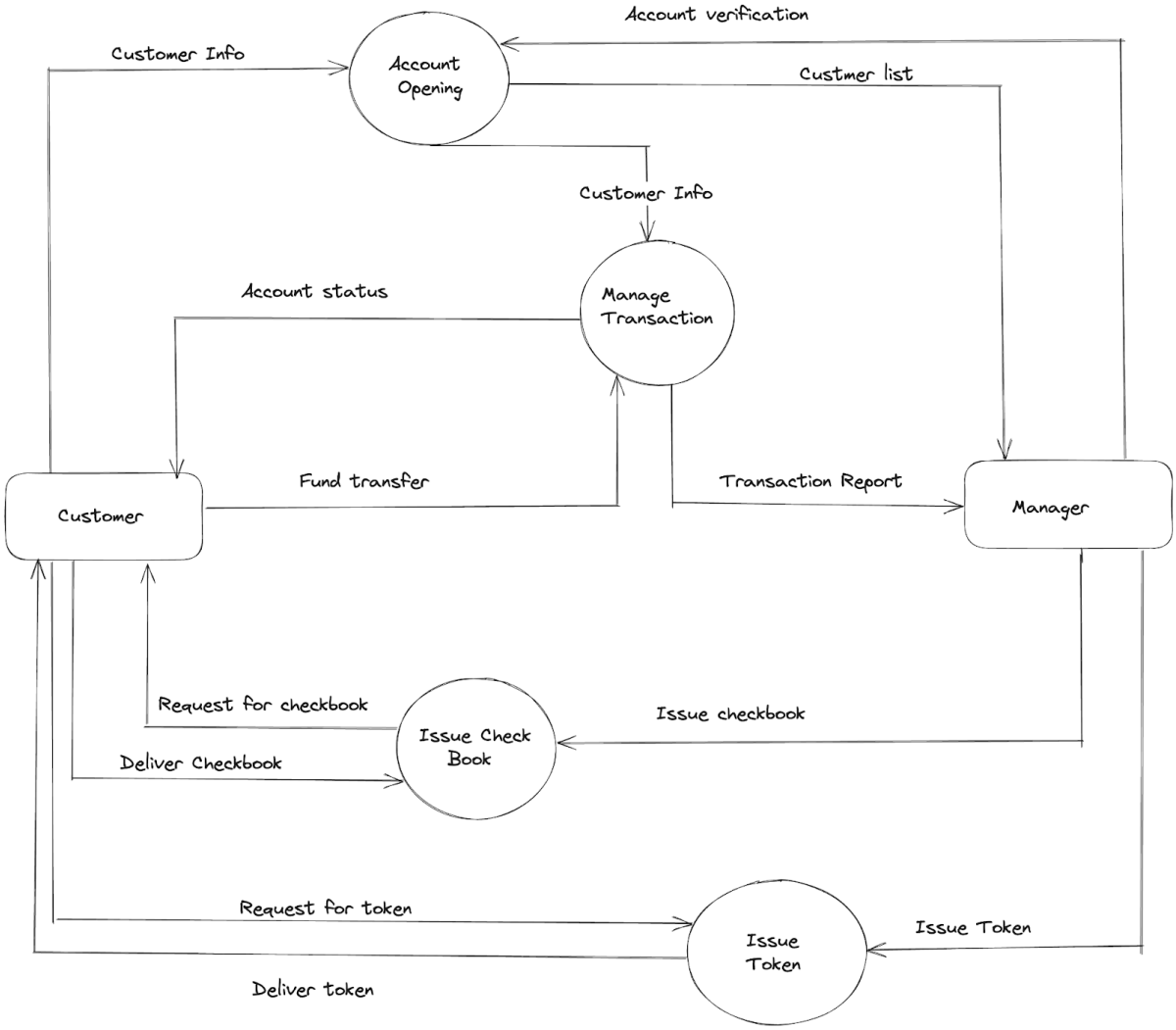
- Visualize Processes: DFDs show how data moves through different processes or functions within a system. This helps in understanding the sequence and logic of operations.
- Identify Data Sources and Destinations: DFDs highlight the origins and endpoints of data. These could be external entities, other systems, databases, or even manual inputs.
- Clarify Data Transformations: The transformation of data as it passes through different processes is shown in DFDs. This makes it easier to identify where data is modified or manipulated.
- Display Data Stores: DFDs include data stores, which are places where data is stored for later use. These could be databases, files, or other storage mechanisms.
- Support Communication: DFDs serve as a common visual language that helps different stakeholders (like developers, analysts, and users) understand how data is processed and utilized.
- Aid in System Analysis and Design: DFDs are often used during the requirements gathering phase to analyze and define the system's data flow, and they continue to be useful in the design phase.

DFDs come in various levels of complexity, from high-level overviews to detailed diagrams that dive deep into specific processes. They help in documenting the architecture of a system and facilitate communication between different teams and individuals involved in the development process.

Different Levels of DFD:

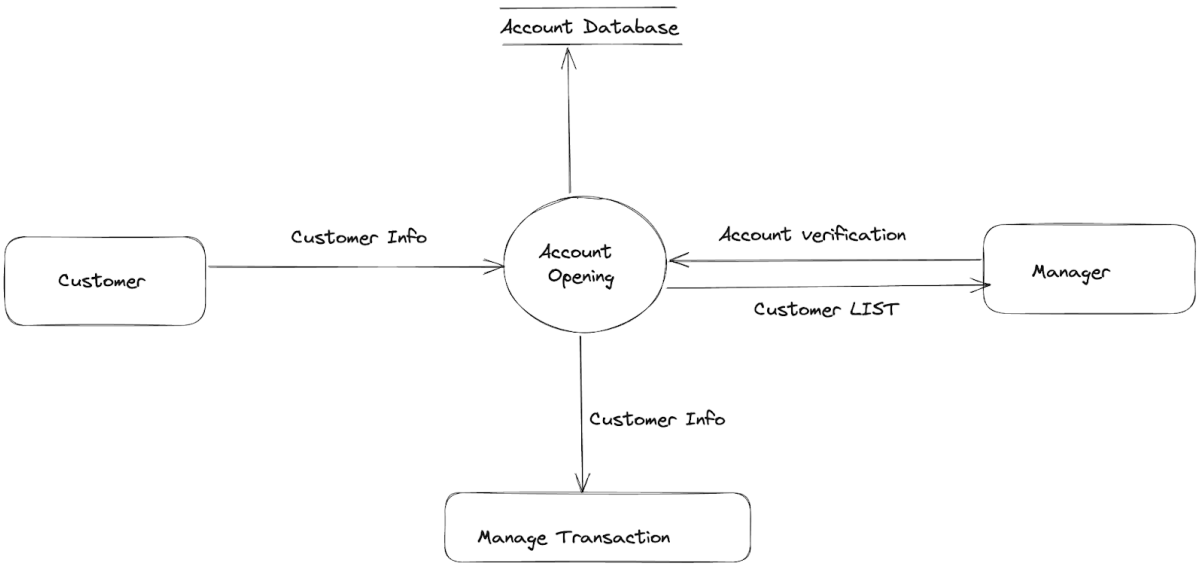
○ **DFD Level 0** is also called a Context Diagram. It’s a basic overview of the whole system or process being analyzed or modeled. It’s designed to be an at-a-glance view, showing the system as a single high-level process, with its relationship to external entities. It should be easily understood by a wide audience, including stakeholders, business analysts, data analysts and developers.

DFD Level 1 provides a more detailed breakout of pieces of the Context Level Diagram. You will highlight the main functions carried out by the system, as you break down the high-level process of the Context Diagram into its subprocesses. A Level 1 Data Flow Diagram (DFD) is a more detailed representation of a system's data flow compared to a higher-level DFD. In a Level 1 DFD, the processes identified in the context diagram (Level 0 DFD) are broken down into sub-processes or activities. The Level 1 DFD provides a clearer view of how data moves between processes, data stores, and external entities within a system.

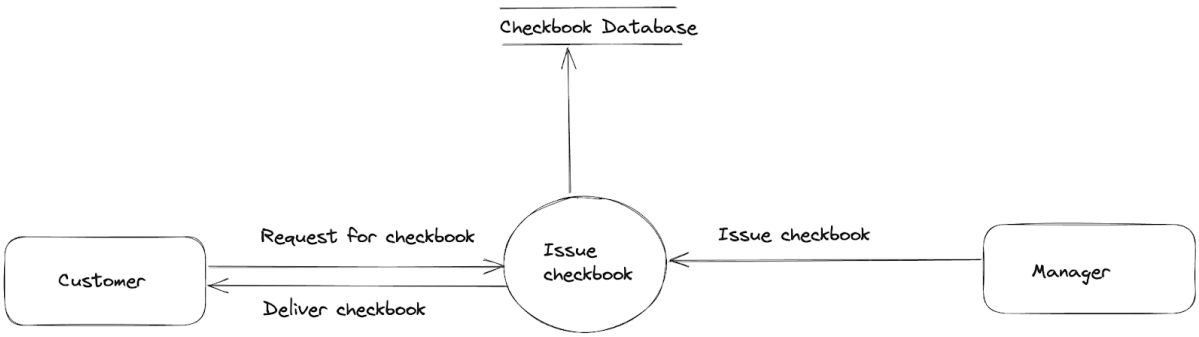


DFD Level 2 is a further breakdown of the processes identified in a Level 1 DFD. It provides even more detailed information about how data flows within a specific process or sub-process. In a Level 2 DFD, each process from the Level 1 DFD is decomposed into more granular sub-processes, and the interactions between them are depicted with greater clarity.

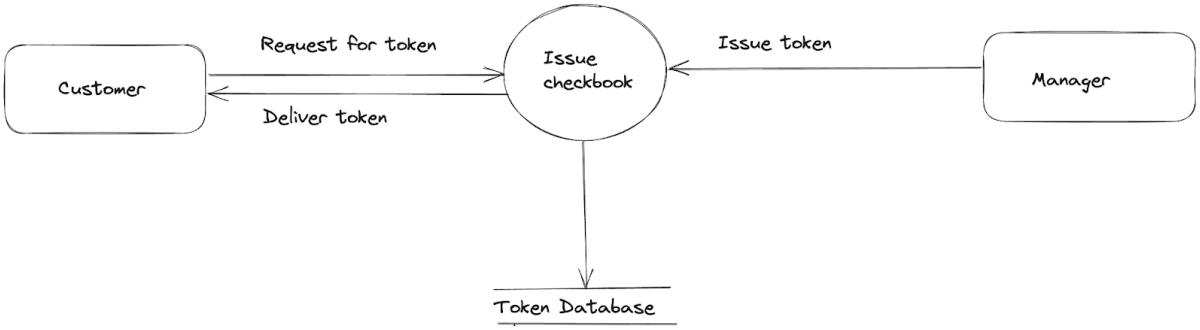
1) Account Opening



2. Checkbook issue



3. Issue token



EXPERIMENT 5

AIM :- Develop activity and state diagram for the project

THEORY

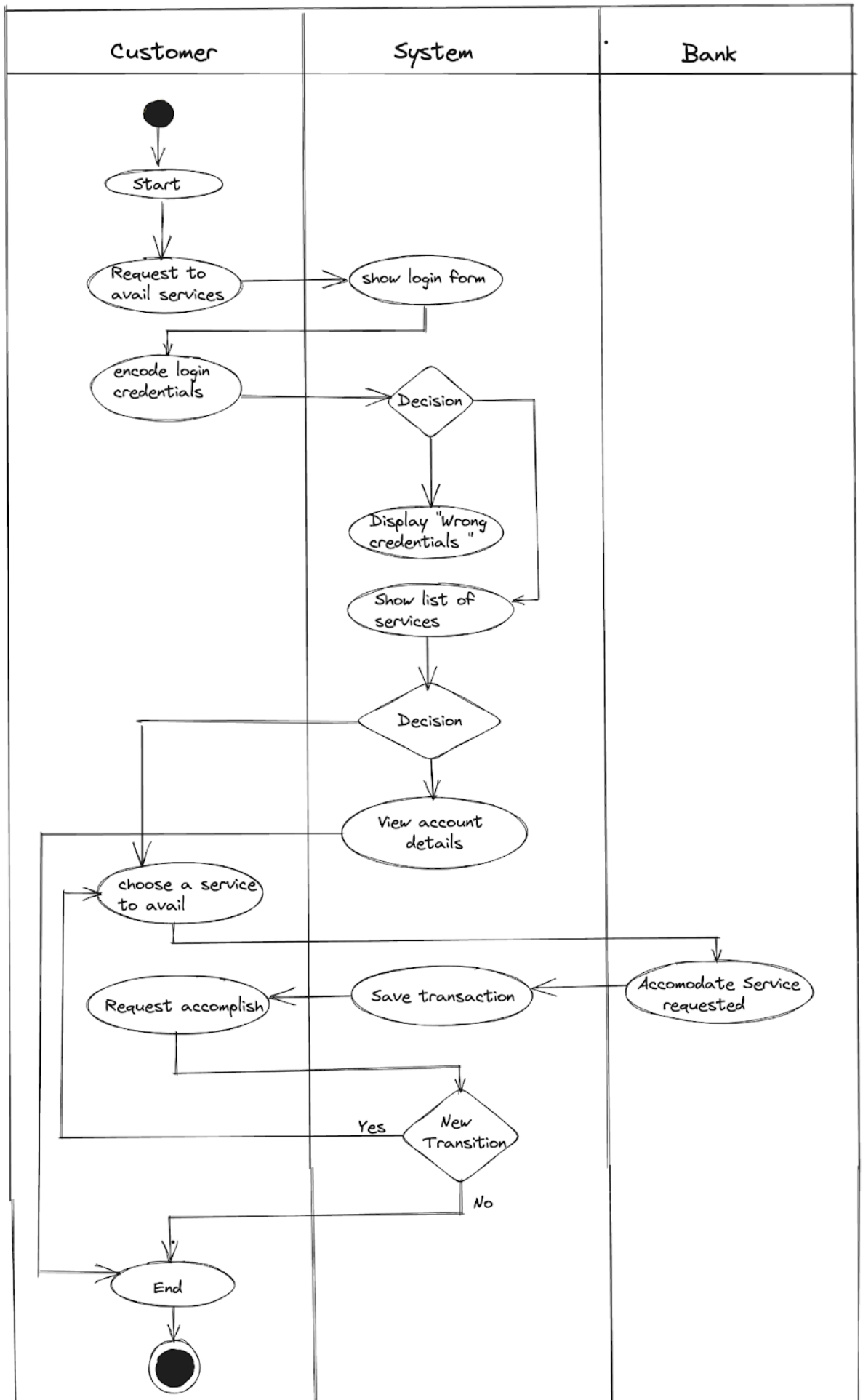
A] Activity Diagram

An activity diagram is a type of diagram used in software engineering to visually represent the flow of activities, actions, and transitions within a system, process, or workflow. It's a part of the Unified Modeling Language (UML), which is a standardized notation for modeling software systems.
Activity diagrams are especially useful for depicting complex business processes, use cases, and the logic behind the behavior of a system. They help to illustrate the sequence of actions, decision points, parallel activities, and the interaction between different elements in the system.
Here are the key components and symbols used in an activity diagram:

- Initial Node: Represents the starting point of the activity diagram.
- Activity State: Represents a specific action or task. It can have incoming and outgoing transitions.
- Decision Node: Represents a decision point where the flow of the diagram splits into different paths based on a condition.
- Merge Node: Represents a point where multiple paths rejoin into a single path after a decision point.
- Fork Node: Represents a point where the flow splits into multiple parallel activities that can occur simultaneously.
- Join Node: Represents a point where parallel activities converge back into a single flow.
- Final Node: Represents the end point of the activity diagram.
- Control Flow: Represents the flow of actions and decisions between nodes. It's typically depicted as arrows.
- Object Flow: Represents the flow of objects (data or information) between activities. It's also depicted as arrows.

When creating an activity diagram, it's important to:

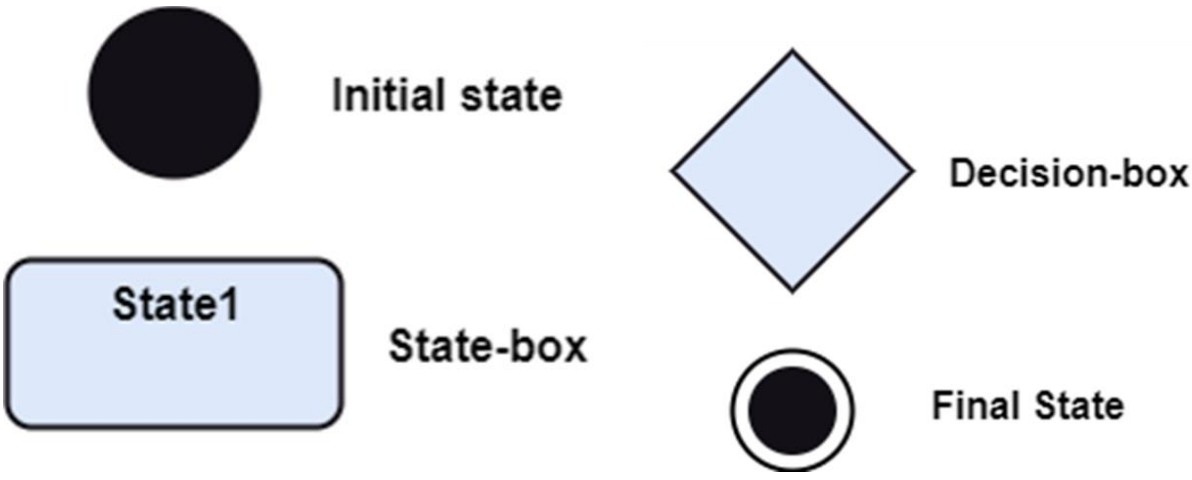
1.Clearly label each activity and decision point.
2.Use meaningful names for actions and transitions.
3.Maintain a logical flow from start to finish.
4.Use control and object flow arrows to depict the sequence of activities and data flow.

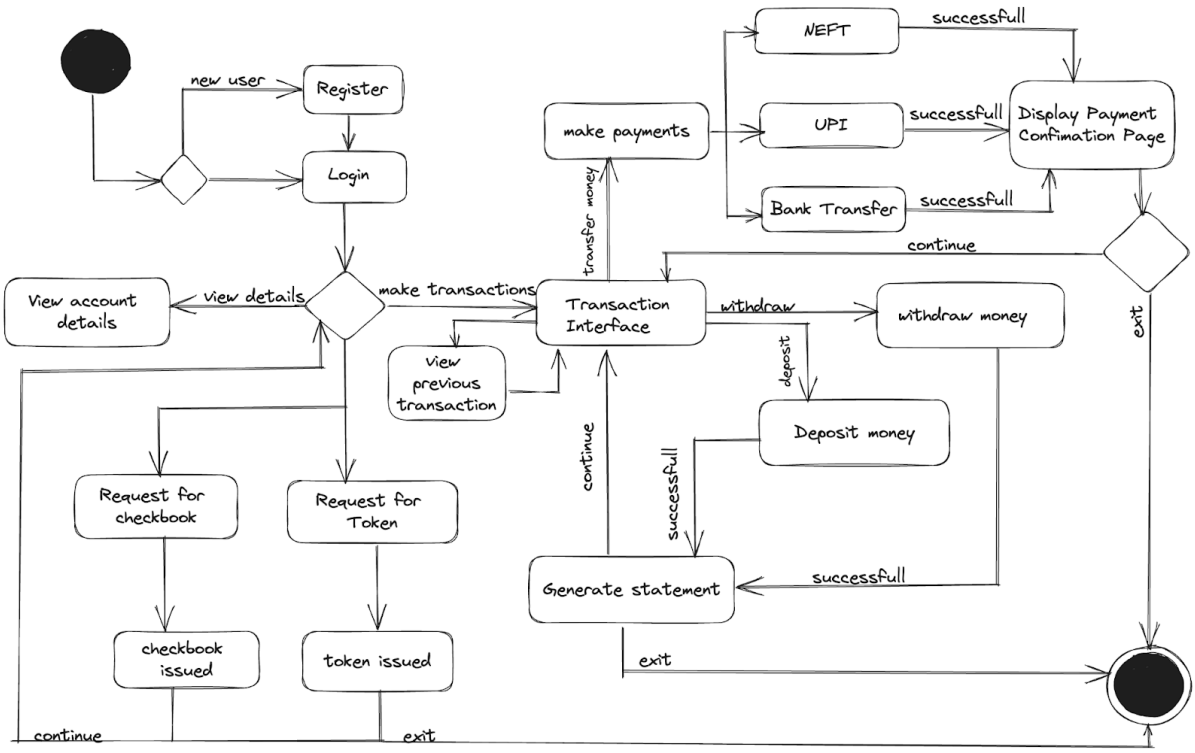

Customer
System
Bank
Start
Request to avail services
show login form
encode login credentials
Decision
Display "Wrong credentials "
Show list of services
Decision
View account details
choose a service to avail
Accomodate Service requested
Save transaction
Request accomplish
New Transition
Yes
No
End

B] State Diagram

A state diagram, also known as a state machine diagram, is a visual representation used in software engineering to model the behavior of an object, component, or system in terms of its states, transitions, and events. State diagrams are a part of the Unified Modeling Language (UML) and are particularly useful for depicting the lifecycle and dynamic behavior of objects or systems.
State diagrams consist of states, transitions, events, and actions. They are widely used to describe the behavior of complex systems, especially those with multiple states and interactions between those states. Here are the key components and symbols used in a state diagram:

- State: A state represents a condition or phase that an object or system can be in. It can represent various states that an entity can transition between, such as "Idle," "Active," "Paused," etc.
- Transition: A transition represents the change of state caused by an event. It's depicted by an arrow connecting two states and is labeled with the triggering event and optional guard conditions.
- Event: An event is a trigger that leads to a state transition. Events can be external (coming from the environment) or internal (triggered by the object itself).
- Action: An action is an operation or behavior associated with a state or a transition. It can represent what happens when a state is entered, exited, or during a transition.
- Initial State: Represents the starting point of the state machine diagram. It is often depicted as a solid circle.
- Final State: Represents the end point of the state machine diagram. It can indicate termination or completion of the process. It is often depicted as a solid circle with a hollow circle inside.
- Choice/Pseudostate: Represents a decision point in the state diagram where the next state is determined by a condition. It's often depicted as a diamond shape.
- Composite State: Represents a grouping of states within a state diagram. It's used to simplify the diagram by encapsulating related states.




new user
Register
Login
View account details
view details
make transactions
View previous transaction
Request for checkbook
Request for Token
checkbook issued
token issued
continue
exit
Transaction Interface
transfer money
make payments
NEFT
UPI
Bank Transfer
successfull
Display Payment Confimation Page
continue
exit
withdraw
withdraw money
deposit
Deposit money
continue
successfull
Generate statement
successfull
exit

EXPERIMENT 6

Aim: Identify scenarios and develop use case diagram for the project

Theory:

A use case diagram is a visual representation used in software engineering to illustrate the interactions between users (actors) and a system's various functions (use cases). It serves as a high-level view of the system's behavior, helping stakeholders understand how different components and actors interact within the system. They are a fundamental tool in the Unified Modeling Language (UML).

Components of Use Case Diagrams:

1. Actors: Actors are external entities that interact with the system. They can be humans, other systems, or even hardware components. They are represented as stick figures on the diagram and they initiate the use cases. Primary actors are on the left of the system and secondary actors are on the right.
2. Use Cases: Use cases represent specific functionalities or actions that the system can perform. They are represented as ovals on the diagram. Each use case corresponds to a particular interaction or process that the system can execute. It accomplishes a task within the system.
3. Relationships: Use cases are connected to actors through associations, which show the roles of actors in performing specific actions. Actors are linked to use cases they interact with, showing the system's functional requirements from the user's perspective. They are of two types, include and extend. The "include" relationship indicates that one use case encompasses another, meaning it always occurs when the base ude case occurs. The "extend" relationship signifies optional or conditional behavior that enhances a use case. It does not necessarily occur when base use case occurs.
4. System Boundary: Use case diagrams usually have a boundary, which defines the scope of the system being modeled. Use cases are inside the boundary, represented by a rectangle box. Actors are outside the box and relationships are arrows connecting the two.

Benefits and Applications:

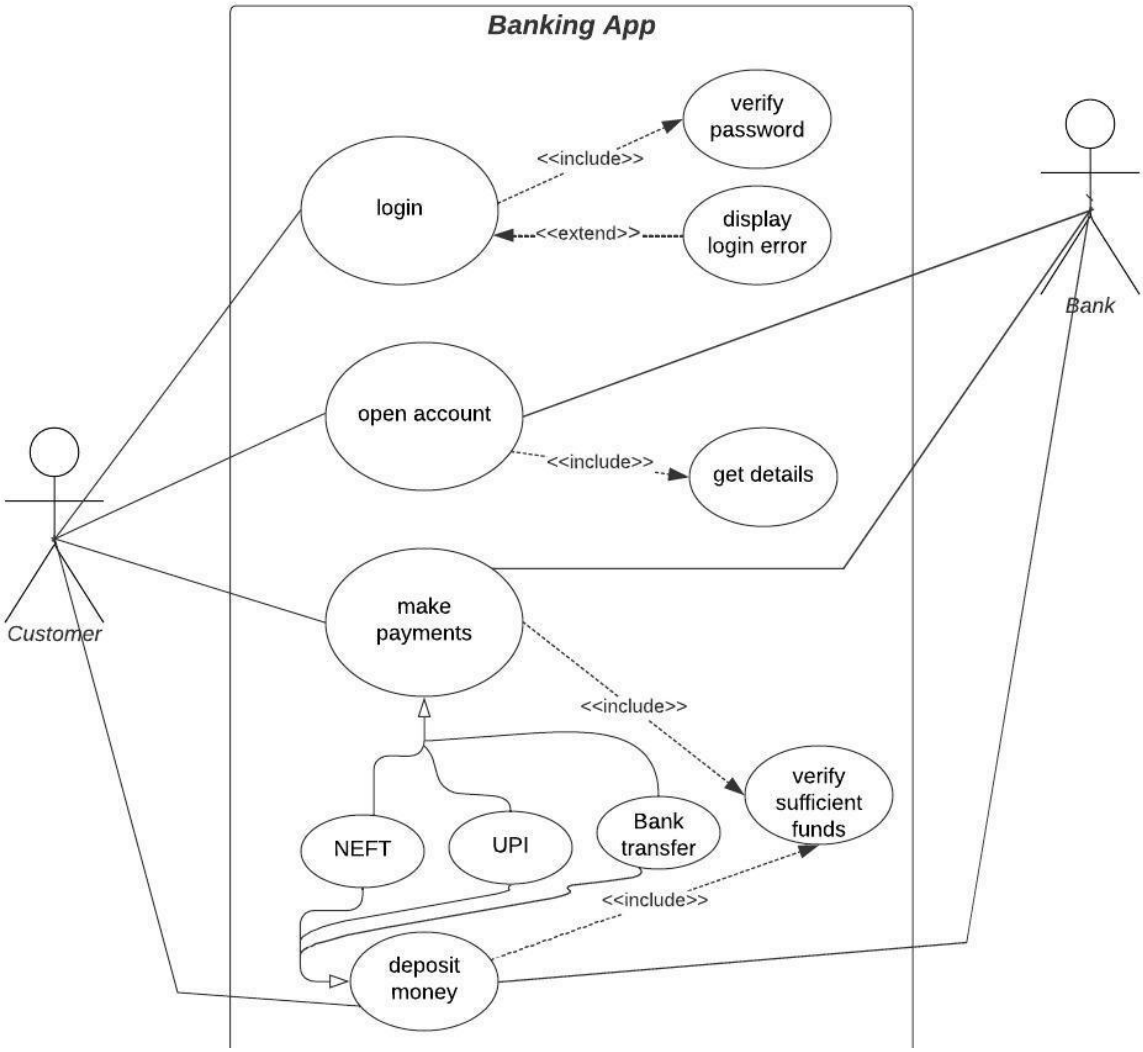
1. Requirement Communication: Use case diagrams offer a clear and intuitive way to communicate functional requirements to stakeholders, bridging the gap between technical teams and non-technical stakeholders.
2. System Design: Use case diagrams help in the early stages of system design by providing an overview of the system's major functions and the interactions between users and those functions.
3. Prioritization: By visualizing interactions between actors and use cases, teams can prioritize development efforts based on the critical functionalities that benefit users the most.
4. Testing and Validation: Use case diagrams can guide the creation of test cases, ensuring that the system is validated against user interactions and requirements.
5. Project Management: These diagrams aid project managers in identifying potential risks and estimating project complexity by highlighting the number of interactions and use cases.

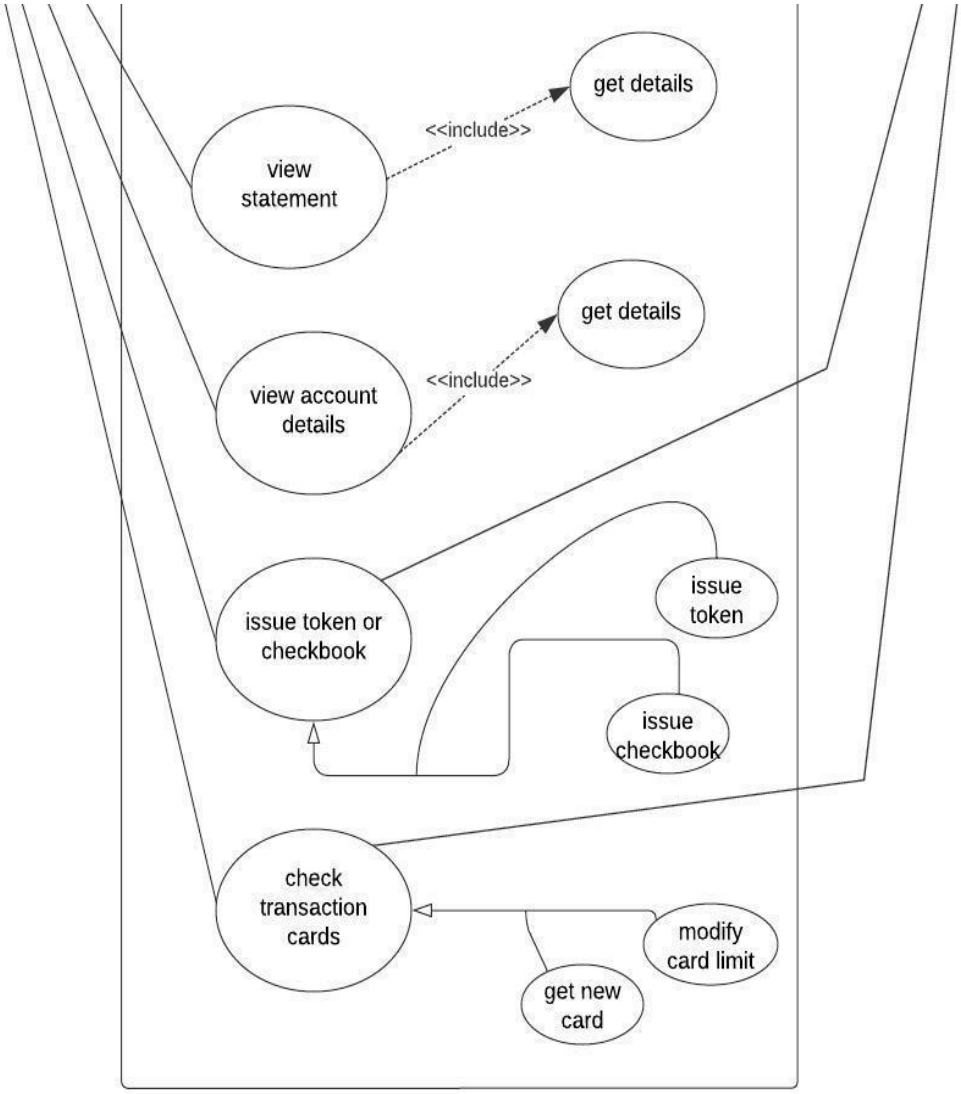
Disadvantages:

1. Simplicity: While use case diagrams provide a high-level overview, they might not capture all intricate details of complex systems.

2. Ambiguity: There's a potential for misinterpretation, as the diagrams focus on what the system should do rather than how it accomplishes it.
3. Dynamic Aspects: Use case diagrams don't show the sequence of events or the dynamic behavior of the system. They're static representations of functionality.

Diagram:




get details
<<include>>
view statement
get details
<<include>>
view account details
issue token or checkbook
issue token
issue checkbook
check transaction cards
modify card limit
get new card

EXPERIMENT 7

Aim: Create a project schedule using Gantt chart

Theory:

Generalized Activity Normalization Time Table (GANTT) chart is type of chart in which series of horizontal lines are present that show the amount of work done or production completed in given period of time in relation to amount planned for those projects.
It is horizontal bar chart developed by Henry L. Gantt (American engineer and social scientist) in 1917 as production control tool. It is simply used for graphical representation of schedule that helps to plan in an efficient way, coordinate, and track some particular tasks in project.

The purpose of Gantt chart is to emphasize scope of individual tasks. Hence set of tasks is given as input to Gantt chart. It is also known as timeline chart. It can be developed for entire project or it can be developed for individual functions.
In most of projects, after generation of timeline chart, project tables are prepared. In project tables, all tasks are listed in proper manner along with start date and end date and information related to it.

Gantt chart represents following things :

- There are two halves to most Gantt chart tools. On the left is a grid that lists project tasks and important information about them
- On the right side, there's a Gantt timeline, which is represented with a stacked bar chart where each task has a corresponding bar that runs on the horizontal axis. The task bars start on the date that the work is scheduled to start. The longer the bar, the longer the task will take.
- When occurring of multiple horizontal bars takes place at same time on calendar, then that means concurrency can be applied for performing particular tasks.
- The diamonds indicate milestones.

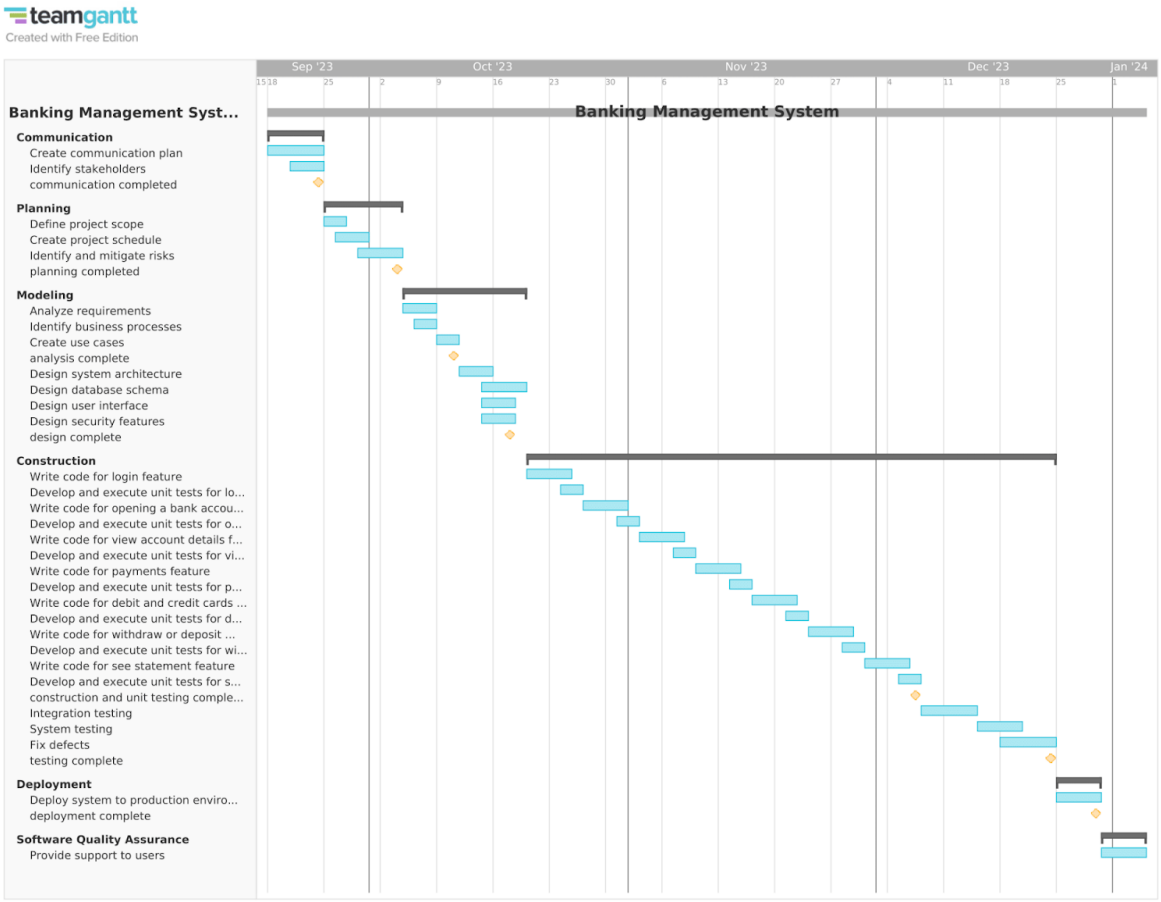
Advantages:

- Simplifies project: Gantt charts are generally used for simplifying complex projects.
- Establish schedule: It simply establishes initial project schedule in which it mentions who is going to do what, when, and how much time it will take to complete it.
- Provides efficiency: It brings efficiency in planning and allows team to better coordinate project activities.
- Emphasize on scope: It helps in emphasizing i.e., gives importance to scope of individual tasks.
- Ease to understand:It makes it easy for stakeholders to understand timeline and brings clarity of dates.
- It helps in clearly visualizing project management, project tasks involved.

- It makes the project planning practical and realistic as realistic planning generally helps to avoid any kind of delays and losses of many that can arise.

Disadvantages:

- Sometimes, using Gantt chart makes project more complex.
- The size of bar chart dost not necessarily indicate amount of work done in project.
- Gantt charts and projects are needed to be updated on regular basis.
- It is not possible or difficult to view this chart on one sheet of paper. The software products that produce Gantt chart needed to be viewed on computer screen so that whole project can be seen easily.


teamgantt
Created with Free Edition
Sep '23
Oct '23
Nov '23
Dec '23
Jan '24
15 18 25 2 9 16 23 30 6 13 20 27 4 11 18 25 1
Banking Management Syst...
Banking Management System
Communication
Create communication plan
Identify stakeholders
communication completed
Planning
Define project scope
Create project schedule
Identify and mitigate risks
planning completed
Modeling
Analyze requirements
Identify business processes
Create use cases
analysis complete
Design system architecture
Design database schema
Design user interface
Design security features
design complete
Construction
Write code for login feature
Develop and execute unit tests for lo...
Write code for opening a bank accou...
Develop and execute unit tests for o...
Write code for view account details f...
Develop and execute unit tests for vi...
Write code for payments feature
Develop and execute unit tests for p...
Write code for debit and credit cards ...
Develop and execute unit tests for d...
Write code for withdraw or deposit ...
Develop and execute unit tests for wi...
Write code for see statement feature
Develop and execute unit tests for s...
construction and unit testing comple...
Integration testing
System testing
Fix defects
testing complete
Deployment
Deploy system to production enviro...
deployment complete
Software Quality Assurance
Provide support to users

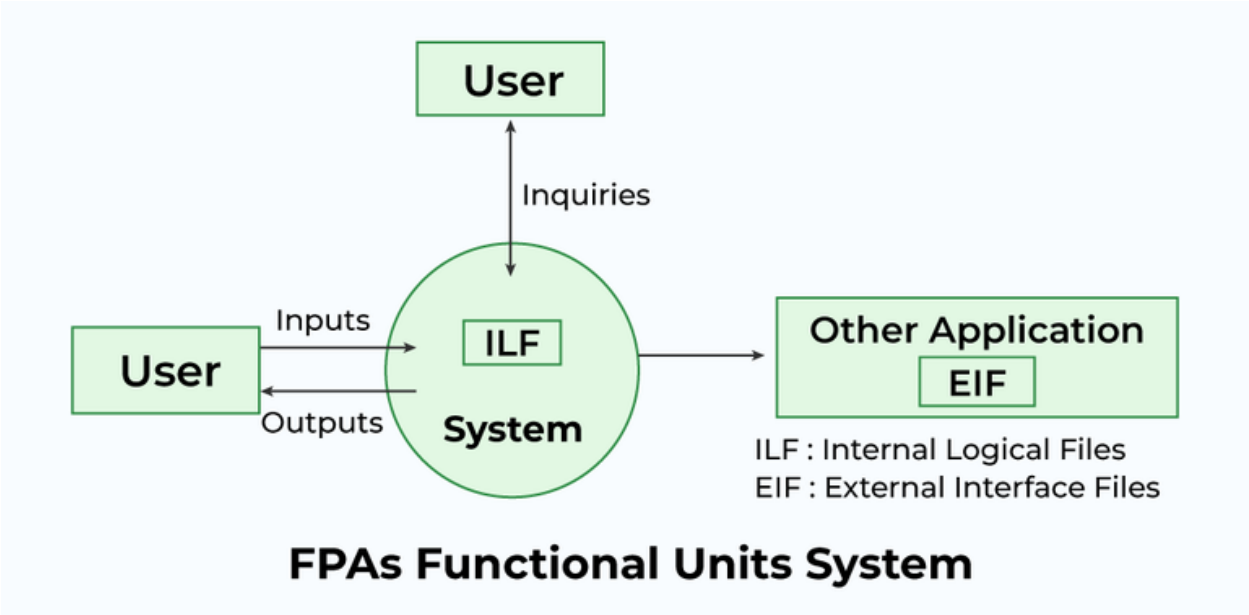
EXPERIMENT 8

Aim: Conduct Function Point Analysis for Banking Management System

Theory:
Functional Point Analysis gives a dimensionless number defined in function points which we have found to be an effective relative measure of function value delivered to our customer.

Objectives of Functional Point Analysis

- The objective of FPA is to measure the functionality that the user requests and receives.
- The objective of FPA is to measure software development and maintenance independently of the technology used for implementation.
- It should be simple enough to minimize the overhead of the measurement process.
- It should be a consistent measure among various projects and organizations.



Benefits of Functional Point Analysis

- FPA is a tool to determine the size of a purchased application package by counting all the functions included in the package.
- It is a tool to help users discover the benefit of an application package to their organization by counting functions that specifically match their requirements.
- It is a tool to measure the units of a software product to support quality and productivity analysis.
- It is a vehicle to estimate the cost and resources required for software development and maintenance.
- It is a normalization factor for software comparison.

To compute function points (FP), the following relationship is used:

$$FP = \text{count total} * [0.65 + 0.01 \sum(Fi)]$$

= UFP * CAF

where

UFP: Unadjusted Function Point

CAF: Complexity Adjustment Factor

We calculate the unadjusted functional point with the help of the number of functions and types of functions used in applications. These are classified into five types.

Measurement Parameters	**Description**	**Examples**
Number of External Inputs (EI)	Processes data or control information that comes from outside the application's boundary.	Input screen and tables
Number of External Output (EO)	Process that generates data or control information sent outside the application's boundary.	Output screens and reports
Number of external inquiries (EQ)	Process made up of an input-output combination that results in data retrieval.	Prompts and interrupts

Number of internal files (ILF)	User-identifiable group of logically related data or control information maintained within the boundary of the application.	Databases and directories
Number of external interfaces (EIF)	Group of users recognizable logically related data allusion to the software but maintained within the boundary of another software	Shared databases and shared routines

Weights of 5 Function Point Attributes

Measurement Parameter	**Weighing Factor**		
	Simple	**Average**	**Complex**
Number of external inputs (EI)	3	4	6
Number of external outputs (EO)	4	5	7

Number of external inquiries (EQ)	3	4	6
Number of internal files (ILF)	7	10	15
Number of External Interfaces (EIF)	5	7	10

The functional complexities are multiplied with the corresponding weights against each function and the values are added up to determine the UFP (Unadjusted Function Point) of the subsystem.

The CAF i.e. Complexity Adjustment Factor is given by [0.65 + 0.01 ∑(Fi)] where Fi (i = 1 to 14) are "complexity adjustment values" These are responses to the 14 questions and ∑(fi) ranges from 0 to 70.

Each of these questions is answered using a scale that ranges from 0 (not important or applicable) to 5 (absolutely essential).

0 – No Influence
1 – Incidental
2 – Moderate
3 – Average
4 – Significant
5 – Essential.

Sr. No.	Question	Value	Reason/Justification
1	Backup& Recovery	4	Need robust backup and recovery mechanisms to ensure the security and availability of customer data.

2	Data communications	3	Heavily relies on data communication for transactions and customer interactions.
3	Distributed Processing	3	Have distributed components like web servers, application servers, and databases
4	Performance Critical	5	Must provide high performance to ensure responsive user experiences and handle high loads
5	Existing Operating Environment	4	Utilizing existing operating environments can reduce complexity, but it can also introduce constraints.
6	Online Data entry	4	Online data entry is central to an online banking system
7	Input transactions over multiple screens	5	Input transactions over multiple screens, such as multi-step processes for opening accounts or making complex transactions impact the complexity of the system.
8	Master Files updated Online	3	Common in banking systems but not as complex as some other aspects
9	Information domain values Complex	3	While banking involves complex financial data, handling this complexity is a standard requirement
10	Internal Processing Complex	3	Necessary for security checks, transaction validation, and other banking operations, but it's not exceptionally complex
11	Code designed for reuse	2	Notas critical as other factors like performance and security.
12	Conversion/Installation in design	3	Depending on the existing infrastructure, converting or installing components may require some effort
13	Multiple Installations	2	Uncommon for an online banking website, so the impact is lower.
14	Application designed for change	3	Need to adapt to changing regulations and customer needs, making this factor moderately impactful.

∑(Fi)	47

Numerical:

3 External Inputs (EI)- Login, Fund Transfer, Bill Payment

2 External Inquiries (EQ)-Account Balance Inquiry, Transaction History Inquiry

2 External Outputs (EO)-Transaction Confirmation, Statement Generation

1 Internal Logical Files (ILF)- Customer Account Data

4 External Interface Files (EIF)- Payment Gateway Interface, User Profile Data, Transaction Logging, Security and Authentication

Measurement parameter	**Weighting factor**					
	Simple		**Average**		**Complex**	
Number of user inputs	3 x 3	**9**	4 x 3	**12**	6 x 3	**18**
Number of user outputs	4 x 2	**8**	5 x 2	**10**	7 x 2	**14**
Number of user inquiries	3 x 2	**6**	4 x 2	**8**	6 x 2	**12**
Number of files	7 x 1	**7**	10 x 1	**10**	15 x 1	**15**
Number of external interfaces	5 x 4	**20**	7 x 4	**28**	10 x 4	**40**
	Simple Total	**50**	**Average Total**	**68**	**Complex Total**	**99**

For Weighing Factor as **Simple:**

FP = count total * [0.65 + 0.01 ∑(Fi)]

=50* [0.65 + 0.01*47]

=50* [0.65 + 0.47]

=50*[1.12]

=56

For Weighing Factor as **Average:**

FP = count total * [0.65 + 0.01 ∑(Fi)]
=68* [0.65 + 0.01*47]
=68* [0.65 + 0.47]
=68*[1.12]
=76.16

For Weighing Factor as **Complex:**
FP = count total * [0.65 + 0.01*47]]
= 99*[0.65 + 0.01 ∑(Fi)]
=99* [0.65 + 0.47]
=99*[1.12]
=110.88

The computed value of the function point metric is *56 for Simple*, *76.16 for Average* and *110.88 for Complex*.

EXPERIMENT 9

Aim: Application of COCOMO Model for cost estimation of the project

Theory :

COCOMO 1

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. It was proposed by Barry Boehm in 1981 and is based on the study of 63 projects, which makes it one of the best-documented models.

The key parameters which define the quality of any software products, which are also an outcome of the Cocomo are primarily Effort & Schedule:

- **Effort:** Amount of labor that will be required to complete a task. It is measured in person-months units.
- **Schedule:** Simply means the amount of time required for the completion of the job, which is, of course, proportional to the effort put in. It is measured in the units of time such as weeks, and months.

COCOMO model has three software Project Types-

1. **Organic:** If the project deals with developing a well understood application program, small development team and team members are experienced in developing similar types of projects.
2. **Semi Detached:** If the development 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.
3. **Embedded:** If the software being developed is strongly coupled to complex hardware, or if the stringent regulations on the operational procedures exist.

Mode	Project-Size	Nature Of Project	Deadline of the project	Development Environment
Organic	2-50 KLOC	Small size project, small size team, Experienced developers in the familiar environment. Eg - Application Programs such as Payroll, Inventory projects.	No tight	Familiar & In House
Semi-detached	50-300 KLOC	Medium size comments, medium size team, Average previous experience on similar projects.Eg - Utility Programs : Compilers, linkers	Medium	Medium
Embedded	Over 300 KLOC	Large Projects, Real time systems, complex interfaces, very little previous experience.	Tight	Complex h/w, customer

1. Basic COCOMO

Primary cost driver is the number of Delivered Source Instructions (DSI) or Kilo Lines Of Code (KLOC) developed by the project.

The required formulas are -

- Effort=a1x(KLOC)^ a2 MM
- Development Time i.e Tdev=b1x(Effort)^b2 Months
- Productivity = KLOC/Effort KLOC/MM
- Average-Staffing = Effort / Tdev FSP

Where

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

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

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

FSP is Full-time-equivalent Software Personnel that shows average staff

a1,a2,b1,b2 are constants for each group of software products,

Software Project	a1	a2	b1	b2
Organic	2.4	1.05	2.5	0.38
Semi-Detached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

1.1. Organic

Q) We have determined our project fits the characteristics of Organic mode. We estimate our project will have 45,000 Delivered Source Instructions.Find Effort, development time, Productivity, and average staffing required for the project.
Solution:

Effort = 2.4x(45) ^1.05
= 130.64
= 131 man-months

Development Time = 2.5x(146)^0.38
=16.61
= 17 months

Productivity = 45 DSI / 131 MM
= 0.3435 DSI/MM

Average Staffing = 131 MM /17 months
=7.70
= 8 FSP

1.2. Semi - detached

Q) Developing a medium-sized Banking Management System with an estimated size of 70,000 Delivered Source Instructions (DSI). Determine the Effort, Development Time, Productivity, and Average Staffing required for the project in Semi-Detached mode.
Solution:

Effort = 3.0 x (70) ^ 1.12
= 349.64
= 350 man-months

Development Time = 2.5 x (350) ^ 0.35
= 19.42
= 20 months

Productivity = 70 DSI / 350 MM
= 0.200 DSI/MM

Average Staffing = 350 MM / 20 months
= 17.5
= 18 FSP

1.3. Embedded

Q) Developing a large-scale Banking Management System with an estimated size of 350,000 Delivered Source Instructions (DSI). Determine the Effort, Development Time, Productivity, and Average Staffing required for the project in Embedded mode.
Solution:

Effort = 3.6 x (350) ^ 1.20
= 4066.157
= 4067 man-months

Development Time = 2.5 x (4067) ^ 0.32
= 35.719
= 36 months

Productivity = 350 DSI / 4067 MM
= 0.0860 DSI/MM

Average Staffing = 4067MM / 36 months
= 112.97
= 113 FSP

2. Intermediate COCOMO

The basic Cocomo model considers that the effort is only a function of the number of lines of code and some constants calculated according to the various software systems. The intermediate COCOMO model recognizes these facts and refines the initial estimates obtained through the basic COCOMO model by using a set of 15 cost drivers based on various attributes of software engineering.Four areas for drivers

1. Product itself: inherent complexity of the product, reliability requirements etc.The Product attributes include-
    a. RELY --- Required Software Reliability
    The extent to which the software product must perform its intended functions satisfactorily over a period of time.
    b. DATA --- Data Base Size
    The degree of the total amount of data to be assembled for the data base.
    c. CPLX --- Software Product Complexity
    The level of complexity of the product to be developed.
2. Computer: execution speed required, storage space required, etc.The Computer attributes include-
    a. TIME --- Execution Time Constraint
    The degree of the execution constraint imposed upon a software product.
    b. STOR --- Main Storage Constraint
    The degree of main storage constraint imposed upon a software product.

c. VIRT --- Virtual Machine Volatility
The level of the virtual machine underlying the product to be developed.
d. TURN --- Computer Turnaround Time
The level of of computer response time experienced by the project team developing the product.

3. Personnel: experience level, programming capability, analysis capability, etc.The Personnel attributes include-
   a. ACAP --- Analyst Capability
   The level of capability of the analysts working on a software product.
   b. AEXP --- Applications Experience
   The level of applications experience of the project team developing the software product.
   c. PCAP --- Programmer Capability
   The level of capability of the programmers working on the software product.
   d. VEXP --- Virtual Machine Experience
   e. LEXP --- Programming Language Experience

4. Project itself or Development environment: development facilities available to developers.The Project Attributes include-
   a. MODP --- Modern Programming Practices
   The degree to which modern programming practices (MPPs) are used in developing software product.
   b. TOOL --- Use of Software Tools
   The degree to which software tools are used in developing the software product.
   c. SCED --- Required Development Schedule
   The level of schedule constraint imposed upon the project team developing the software product.

The Table for Cost Drivers and their Rating are as shown-

COCOMO - COST DRIVERS

	COST DRIVER	RATING V.LOW	LOW	NOMINAL	HIGH	V.HIGH	EX. HIGH
(PRODUCT)	RELY	0.75	0.88	1.00	1.15	1.40	.
..	DATA	.	0.94	1.00	1.08	1.16	.
..	CPLX	0.70	0.85	1.00	1.15	1.30	1.65
(COMPUTER)	TIME	.	.	1.00	1.11	1.30	1.66
..	STOR	.	.	1.00	1.06	1.21	1.56
..	VIRT	.	0.87	1.00	1.15	1.30	.
..	TURN	.	0.87	1.00	1.07	1.15	.
(PERSONNEL)	ACAP	1.46	1.19	1.00	0.86	0.71	.
..	AEXP	1.29	1.13	1.00	0.91	0.82	.
..	PCAP	1.42	1.17	1.00	0.86	0.70	.
..	VEXP	1.21	1.10	1.00	0.90	.	.
..	LEXP	1.14	1.07	1.00	0.95	.	.
(PROJECT)	MODP	1.24	1.10	1.00	0.91	0.82	.
..	TOOL	1.24	1.10	1.00	0.91	0.83	.
..	SCED	1.23	1.08	1.00	1.04	1.10	.

The values of the constants are -

Software Project	a1	a2	b1	b2
Organic	3.2	1.05	2.5	0.38
Semi-detached	3.0	1.12	2.5	0.35
Embedded	2.8	1.20	2.5	0.32

Intermediate COCOMO equations are:

- Efforts=a1x(KLOC)^a2x(cost-driver rating)
- Development Time i.e Tdev=b1x(Efforts)^b2
- Productivity = KLOC/Effort KLOC/MM
- Average-Staffing = Effort / Tdev FSP

2.1. Organic

Q) We have determined our project fits the characteristics of Organic mode. We estimate our project will have 56,000 Delivered Source Instructions (DSI).Find Effort, development time, Productivity, and average staffing required for the project.
Where the cost drivers are set as follows:
Product cost drivers- high
Computer cost drivers – nominal
Personnel cost drivers - low
Project cost drivers high
Solution:
Product cost drivers set high = 1.15 x 1.08 x 1.15 = 1.43
Computer cost drivers set nominal = 1.00
Personnel cost drivers set low = 1.19 x 1.13 x 1.17 x 1.10 x 1.07 = 1.85
Project cost drivers set high = 0.91 x 0.91 x 1.04 = 0.86
Total product of the cost drivers = 1.43 x 1.00 x 1.85 x 0.86 = 2.28

Effort = 3.2 x (56)^1.05 x 2.28
= 499.66
= 500 man-months

Development Time = 2.5 x (500) ^ 0.38
= 26.51
= 27 months

Productivity = 56 DSI / 500 MM
= 0.112 DSI/MM

Average Staffing = 500 MM / 27 months
= 18.51
= 19 FSP

2.2. Semi-detached

Q) Developing a medium-sized Banking Management System with an estimated size of 70,000 Delivered Source Instructions (DSI). Determine the Effort, Development Time, Productivity, and Average Staffing required for the project in Semi-Detached mode.
The cost drivers are set as follows:
Product cost drivers- low
Computer cost drivers – high
Personnel cost drivers - nominal
Project cost drivers -v.low

Solution:
Product cost drivers set low = 0.88 x 0.94 x 0.85 = 0.70
Computer cost drivers set high = 1.11 x 1.06 x 1.15 x 1.07 = 1.45
Personnel cost drivers set nominal = 1.00
Project cost drivers set v.low = 1.24 x 1.24 x 1.23 = 1.90
Total product of the cost drivers = 0.70 x 1.00 x 1.45 x 1.90 = 1.93
Effort = 3.0 x (70) ^ 1.12 x1.93
= 674.80
= 675 man-months

Development Time = 2.5 x (675) ^ 0.35
= 24.44
= 25 months

Productivity = 70 DSI / 675 MM
= 0.103 DSI/MM

Average Staffing = 675 MM / 20 months
= 33.75
= 34 FSP

2.3. Embedded

Q) Developing a large-scale Banking Management System with an estimated size of 350,000 Delivered Source Instructions (DSI). Determine the Effort, Development Time, Productivity, and Average Staffing required for the project in Embedded mode.
The cost drivers are set as follows:

Product cost drivers- high
Computer cost drivers – nominal
Personnel cost drivers - low
Project cost drivers high

Solution:
Product cost drivers set high = 1.15 x 1.08 x 1.15 = 1.43
Computer cost drivers set nominal = 1.00
Personnel cost drivers set low = 1.19 x 1.13 x 1.17 x 1.10 x 1.07 = 1.85
Project cost drivers set high = 0.91 x 0.91 x 1.04 = 0.86
Total product of the cost drivers = 1.43 x 1.00 x 1.85 x 0.86 = 2.28

Effort = 2.8 x (350) ^ 1.20x2.28
= 7210.65
= 7211 man-months

Development Time = 2.5 x (7211) ^ 0.32
= 42.90
= 43 months

Productivity = 350 DSI / 7211 MM
= 0.0485 DSI/MM

Average Staffing = 7211MM / 43 months
= 167.69
= 168 FSP

COCOMO 2

>COCOMO 2, short for Constructive Cost Model 2, is an extension and enhancement of the original COCOMO (Constructive Cost Model) software estimation model. COCOMO was initially developed by Dr. Barry Boehm in the 1980s, and COCOMO 2 builds upon the foundation of the original model to provide more accurate and flexible software cost estimation.

>COCOMO 2 is a valuable tool for project managers, software engineers, and stakeholders to make informed decisions regarding project planning, resource allocation, and budgeting in the software development process. It offers a systematic and data-driven approach to estimating the effort and cost required for successful software projects.Object count and classifying complexity level:

Number of Screens	· Authentication Screen · Dashboard screen	Simple Medium
Number of reports	· Result report	Medium
Number of 3GL components	· Python · OpenCV	Difficult Difficult

Complexity Weights and Object Point:

Object Type	Complexity Weights			Given Values			Total
	Simple	Medium	Difficult	Simple	Medium	Difficult	
Screens	1	2	3	1	1	0	3
Reports	2	5	8	0	1	0	5
3GL components	-	-	10	0	0	2	20
Object Point							28

Object Point = 28

New Object Point (NOP): Reusability = 20% (assumed)

NOP = Object Point * [(100 – reuse)/100]

= 28 * 80 / 100

= 22.4

New Object Point = 22.4

Productivity Rate (PROD): Developer experience – high Environment Maturity – low

PROD = (high + low)/2

= (10 + 6)/2= 8

Productivity Rate = 8

Efforts:

Efforts = NOP / PROD

= 22.4/8

= 2.8 person months

Efforts for project development is 2.8 person months.

Size Estimation:

Total FP = 365.19

3GL components 1FP = 50 LOC Estimated size =

*365.19 * 50*

= 18259.5 LOC

= 18.259 KLOC

Estimated project size is 18.259 KLOC.

EXPERIMENT 10

Aim: Develop a Risk Mitigation, Monitoring and Management Plan (RMMM) for a banking management system

Theory:

- Risk always involves two characteristics:
    1. Uncertainty - the risk may or may not happen; that is, there are no 100% probable risks.
    2. Loss - if the risk becomes a reality, unwanted consequences or losses will occur
- When risks are analyzed, it is important to quantify the level of uncertainty and the degree of loss associated with each risk.
- To accomplish this, different categories or types of risks are considered.
    1. Project Risks
    2. Technical Risks
    3. Business Risks
    4. Known Risks.
    5. Predictable Risks
    6. Unpredictable Risks

- A risk management technique is usually seen in the software Project plan. This can be divided into Risk Mitigation, Monitoring, and Management Plan (RMMM).
- In this plan, all works are done as part of risk analysis. As part of the overall project plan project manager generally uses this RMMM plan.
- In some software teams, risk is documented with the help of a Risk Information Sheet (RIS).
- This RIS is controlled by using a database system for easier management of information i.e creation, priority ordering, searching, and other analysis.
- After documentation of RMMM and start of a project, risk mitigation and monitoring steps will start.
- Drawbacks of RMMM:
    1. It incurs additional project costs.
    2. It takes additional time.
    3. For larger projects, implementing an RMMM may itself turn out to be another tedious project.
    4. RMMM does not guarantee a risk-free project, infact, risks may also come up after the project is delivered.

Risk information sheet			
Risk ID: P02-4-32	Date: 5/9/02	Prob: 80%	Impact: high
Description: Only 70 percent of the software components scheduled for reuse will, in fact, be integrated into the application. The remaining functionality will have to be custom developed.			
Refinement/context: Subcondition 1: Certain reusable components were developed by a third party with no knowledge of internal design standards. Subcondition 2: The design standard for component interfaces has not been solidified and may not conform to certain existing reusable components. Subcondition 3: Certain reusable components have been implemented in a language that is not supported on the target environment.			
Mitigation/monitoring: 1. Contact third party to determine conformance with design standards. 2. Press for interface standards completion; consider component structure when deciding on interface protocol. 3. Check to determine number of components in subcondition 3 category; check to determine if language support can be acquired.			
Management/contingency plan/trigger: *RE* computed to be $20,200. Allocate this amount within project contingency cost. Develop revised schedule assuming that 18 additional components will have to be custom built; allocate staff accordingly. Trigger: Mitigation steps unproductive as of 7/1/02			
Current status: 5/12/02: Mitigation steps initiated.			
Originator: D. Gagne		Assigned: B. Laster	

- Risk Mitigation :
  -It is an activity used to avoid problems (Risk Avoidance).
  -Steps for mitigating the risks as follows:
    1. Finding out the risk.
    2. Removing causes that are the reason for risk creation.
    3. Controlling the corresponding documents from time to time.
    4. Conducting timely reviews to speed up the work.
- Risk Monitoring :
  -It is an activity used for project tracking.
  -It has the following primary objectives as follows.:
    1. To check if predicted risks occur or not.
    2. To ensure proper application of risk aversion steps defined for risk.
    3. To collect data for future risk analysis.
    4. To allocate what problems are caused by which risks throughout the project.
- Risk Management and planning :
  -It assumes that the mitigation activity failed and the risk is a reality.
  -This task is done by Project manager when risk becomes reality and causes severe problems.
  -If the project manager effectively uses project mitigation to remove risks successfully then it is easier to manage the risks.
  -This shows that the response that will be taken for each risk by a manager.

Risk Information Sheet for banking management system:

RISK INFORMATION SHEET			
Risk ID: 01	**Date:** 5/10/23	**Probability:** Medium	**Impact:** High
Description: Data breach due to a security vulnerability in the online banking system.			
Mitigation: ○ Regular security audits and penetration testing. ○ Encryption of sensitive user data. ○ Implementing multi-factor authentication. ○ Employee security training.			
Monitoring: ○ Security log analysis. ○ Real-time monitoring of user activities. ○ Intrusion detection systems.			
Management: Regularly update security protocols and promptly address vulnerabilities.			
Current Status: Not Yet Occurred			
Originator: Rittika		**Assigned:** Rittika	

RISK INFORMATION SHEET			
Risk ID: 02	**Date:** 5/10/23	**Probability:** Low	**Impact:** Medium
Description: System downtime during peak usage, affecting online transactions.			
Mitigation: ○ Redundant server setup. ○ Regular maintenance and backups during off-peak hours. ○ Quick server replacement policy.			
Monitoring: ○ Server health monitoring. ○ Automated alerts for downtime.			
Management: Establish a disaster recovery plan for minimal service disruption.			
Current Status: Not Yet Occurred			
Originator: Rittika		**Assigned:** Rittika	

RISK INFORMATION SHEET			
Risk ID: 03	**Date:** 5/10/23	**Probability:** High	**Impact:** High
Description: Third-party payment gateway unavailability affecting online payments.			
Mitigation: ○ Multiple payment gateway providers for redundancy. ○ Graceful degradation of payment services. ○ Regular testing of payment gateways. ○ Contractual agreements with payment service providers.			
Monitoring: ○ Real-time payment gateway status checks. ○ Alerting for payment gateway downtime.			
Management: Develop a fallback mechanism and notify users during payment gateway downtime.			
Current Status: Not Yet Occurred			
Originator: Rittika		**Assigned:** Rittika	

RISK INFORMATION SHEET			
Risk ID: 04	**Date:** 5/10/23	**Probability:** Medium	**Impact:** Low
Description: Inadequate user education on secure online banking practices.			
Mitigation: ○ User manuals and online security guides. ○ Online tutorials on secure online banking practices. ○ Regular communication on security measures.			
Monitoring: ○ User feedback and support requests related to security. ○ Assessment of user awareness.			
Management: Regularly update educational materials and provide ongoing user support.			
Current Status: Not Yet Occurred			
Originator: Rittika		**Assigned:** Rittika	

RISK INFORMATION SHEET			
Risk ID: 05	**Date:** 5/10/23	**Probability:** Low	**Impact:** High
Description: Unauthorized access to customer accounts due to weak authentication.			
Mitigation: ○ Implementation of multi-factor authentication. ○ Regular password policy updates. ○ Real-time monitoring of login attempts.			
Monitoring: ○ Real-time access logs. ○ Anomaly detection for unusual login activity.			
Management: Strict access control policies and periodic security audits.			
Current Status: Not Yet Occurred			
Originator: Rittika		**Assigned:** Rittika	

RISK INFORMATION SHEET			
Risk ID: 06	**Date:** 5/10/23	**Probability:** Medium	**Impact:** Low
Description: Inaccurate online account statements generation.			
Mitigation: ○ Automated statement generation with thorough validation checks. ○ Regular reconciliation of online transactions. ○ User feedback for discrepancies.			
Monitoring: ○ Automated reconciliation reports.			
Management: Implement a robust online statement generation algorithm and address discrepancies promptly.			
Current Status: Not Yet Occurred			
Originator: Niyati		**Assigned:** Niyati	

RISK INFORMATION SHEET			
Risk ID: 07	**Date:** 5/10/23	**Probability:** High	**Impact:** Medium
Description: Software bugs affecting online transaction accuracy.			
Mitigation: ○ Thorough testing in a sandbox environment. ○ Continuous integration and automated testing. ○ Prompt bug fixes with version control. ○ User feedback for bug identification.			
Monitoring: ○ Bug tracking system. ○ User-reported issues.			
Management: Regularly update the software with bug fixes and improvements.			
Current Status: Not Yet Occurred			
Originator: Niyati		**Assigned:** Niyati	

RISK INFORMATION SHEET			
Risk ID: 08	**Date:** 5/10/23	**Probability:** High	**Impact:** Low
Description: Service interruptions due to online network issues.			
Mitigation: ○ Redundant online network connections. ○ Collaboration with multiple online service providers. ○ Regular online network health checks.			
Monitoring: ○ Real-time online network monitoring. ○ Automated alerts for online network downtime.			
Management: Swiftly address online network issues to minimize online service disruptions.			
Current Status: Not Yet Occurred			
Originator: Niyati		**Assigned:** Niyati	

RISK INFORMATION SHEET			
Risk ID: 09	**Date:** 5/10/23	**Probability:** Low	**Impact:** High
Description: Loss of customer online data during system migration.			
Mitigation: ○ Thorough online data backup before migration. ○ Gradual online data migration with rollback options. ○ Extensive online testing in a controlled environment.			
Monitoring: ○ Real-time online monitoring during migration. ○ User feedback on online data accuracy.			
Management: Have a rollback plan and swift response to online data integrity issues.			
Current Status: Not Yet Occurred			
Originator: Niyati		**Assigned:** Niyati	

RISK INFORMATION SHEET			
Risk ID: 10	**Date:** 5/10/23	**Probability:** Medium	**Impact:** Low
Description: Inadequate integration with online external systems.			
Mitigation: ○ Thorough online API documentation and testing. ○ Collaboration with online external system providers. ○ Continuous monitoring of online integration points.			
Monitoring: ○ Regular checks on online external system interfaces. ○ Automated alerts for online integration failures.			
Management: Maintain open communication channels with online external system providers.			
Current Status: Not Yet Occurred			
Originator: Niyati		**Assigned:** Niyati	

RISK INFORMATION SHEET			
Risk ID: 11	**Date:** 5/10/23	**Probability:** Medium	**Impact:** Medium
Description: Inadequate monitoring of online system scalability.			
Mitigation: ○ Regularly review and update scalability plans. ○ Implement automated alerts for potential scalability issues. ○ Conduct periodic load testing to simulate various online usage scenarios.			
Monitoring: ○ Continuous monitoring of server load and response times. ○ Periodic analysis of user growth patterns.			
Management: Regularly assess the effectiveness of scalability plans and make adjustments accordingly.			
Current Status: Not Yet Occurred			
Originator: Ujjwal		**Assigned:** Ujjwal	

RISK INFORMATION SHEET			
Risk ID: 12	**Date:** 5/10/23	**Probability:** Medium	**Impact:** Medium
Description: Inadequate testing of online disaster recovery procedures.			
Mitigation: ○ Regularly conduct comprehensive drills for online disaster recovery. ○ Document and update recovery procedures based on lessons learned. ○ Ensure off-site backups are regularly tested for reliability.			
Monitoring: ○ Periodic testing of online disaster recovery procedures. ○ Monitoring the status of online backups.			
Management: Regularly update and enhance online disaster recovery plans based on testing outcomes.			
Current Status: Not Yet Occurred			
Originator: Ujjwal		**Assigned:** Ujjwal	

RISK INFORMATION SHEET			
Risk ID: 13	**Date:** 5/10/23	**Probability:** Low	**Impact:** High
Description: Inadequate adaptation of online system features to comply with new regulations.			
Mitigation: ○ Establish a dedicated team for monitoring and implementing regulatory changes. ○ Regularly review and update online system features to align with the latest regulations.			
Monitoring: ○ Continuous monitoring of online regulatory updates. ○ Periodic online compliance checks for system features.			
Management: Implement a proactive approach to adapt online system features in response to regulatory changes.			
Current Status: Not Yet Occurred			
Originator: Ujjwal		**Assigned:** Ujjwal	

RISK INFORMATION SHEET			
Risk ID: 14	**Date:** 5/10/23	**Probability:** Medium	**Impact:** Medium
Description: Inadequate monitoring of economic indicators affecting online transactions.			
Mitigation: ○ Collaborate with financial analysts to identify potential economic downturns. ○ Establish a real-time monitoring system for key economic indicators.			
Monitoring: Regular analysis of economic indicators affecting online transactions. ○ Continuous monitoring of customer transaction patterns during economic changes.			
Management: Swiftly respond to economic indicators and adjust online financial strategies as needed.			
Current Status: Not Yet Occurred			
Originator: Ujjwal		**Assigned:** Ujjwal	

RISK INFORMATION SHEET			
Risk ID: 15	**Date:** 5/10/23	**Probability:** High	**Impact:** Low
Description: Lack of user engagement with new online features			
Mitigation: ○ Conduct user surveys and focus groups during the development of new online features. ○ Implement targeted marketing and communication strategies for new features.			
Monitoring: ○ Monitor user feedback on online forums and support channels. ○ Analyze usage metrics for new online features.			
Management: Continuously assess user satisfaction and adjust marketing strategies to improve online feature adoption rates.			
Current Status: Not Yet Occurred			
Originator: Ujjwal		**Assigned:** Ujjwal	

EXPERIMENT 11

Aim: Case Study - GitHub Version Control

Theory:

1. What is GitHub?
   a. GitHub is a Git repository hosting service that provides a web-based graphical interface.
   b. It is the world's largest coding community.
   c. Putting a code or a project into GitHub brings it increased, widespread exposure.
   d. Programmers can find source codes in many different languages and use the command-line interface, Git, to make and keep track of any changes.
   e. GitHub helps every team member work together on a project from any location while facilitating collaboration.
   f. It makes it easy for developers to share code files and collaborate with fellow developers on open-source projects.
   g. GitHub also serves as a social networking site where developers can openly network, collaborate, and pitch their work.
   h. We can also review previous versions created at an earlier point in time.

   i. Some of the features of GitHub are as follows:
      i. Easy Project Management:

      GitHub is a place where project managers and developers come together to coordinate, track, and update their work so that projects are transparent and stay on schedule.

      ii. Increased Safety With Packages:

      Packages can be published privately, within the team, or publicly to the open-source community. The packages can be used or reused by downloading them from GitHub.

      iii. Effective Team Management:

      GitHub helps all the team members stay on the same page and organized. Moderation tools like Issue and Pull Request Locking help the team to focus on the code.

      iv. Improved Code Writing:

      Pull requests help the organizations to review, develop, and propose new code. Team members can discuss any implementations and proposals through these before changing the source code.

      v. Increased Code Safety:

      GitHub uses dedicated tools to identify and analyze vulnerabilities to the code that other tools tend to miss. Development teams everywhere work together to secure the software supply chain, from start to finish.

      vi. Easy Code Hosting:

      All the code and documentation are in one place. There are millions of repositories on GitHub, and each repository has its own tools to help you host and release code.

2. Version control
   a. Version control, also known as source control, is the practice of tracking and managing changes to software code.
   b. Version control systems are software tools that help software teams manage changes to source code over time.
   c. As development environments have accelerated, version control systems help software teams work faster and smarter.
   d. Version control software keeps track of every modification to the code in a special kind of database.
   e. 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.
   f. 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.
   g. Version control protects source code from both catastrophe and the casual degradation of human error and unintended consequences.
   h. Software developers working in teams are continually writing new source code and changing existing source code.
   i. The code for a project, app or software component is typically organized in a folder structure or "file tree".
   j. 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.
   k. Version control helps teams solve these kinds of problems, tracking every individual change by each contributor and helping prevent concurrent work from conflicting.

   l. The benefits of Version Control are as follows:
      i. <u>Long-term change history of every file:</u>
         This means that every change made by many individuals over the years is stored safely. Changes include the creation and deletion of files as well as edits to their contents.

      ii. <u>Branching and merging</u>
         Version Control tools keep 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.

      iii. <u>Traceability</u>
         Features such as 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 are made possible using Version Control.

3. Version control using GitHub
   a. The core functionality of Git works in mainly three file states: modified **state**, **staged state**, or **committed state**. If we include a remote repository then we can divide its core functionality into four states.
   b. Let’s consider an example: Suppose we are asked to create a photo album with some caption or message

along with the pictures.

c. The following steps will be followed by most of the users:
    i. The user will take some pictures to include in the photo album. The user has the freedom to filter out the pictures to be included.
    ii. If the user hasn't clicked a good picture, then they also have the choice to take the same picture again if it's necessary.
    iii. In the next step, the user will filter out the pictures to be included and print them. Basically, the user creates a staging area for the photo album.
    iv. Once the user is ready with the picture they can fix them in the photo album with some message or title describing the picture.
d. Let's relate these steps to Git stages for better understanding:
    i. Modified State: In this state, the files are modified, but changes are not saved. Taking the picture is just like making changes in the files. We create files, we write code to add some features, or we delete the files.
    ii. Staged State: In this state, the files are prepared to be committed in the .git repository. This is like clicking some new pictures and then deciding to paste them into the photo album.
    iii. Committed State: In this final stage, we successfully save the files in the .git repository, and we create a commit for them. Creating the commit is just like fixing the pictures in the photo album.

4. Synchronous feature of Github
    a. GitHub does not have a native synchronous feature. However, there are a few ways to achieve synchronous behavior on GitHub.
    b. One way is to use GitHub Actions. GitHub Actions allows you to automate workflows using scripts. You can create a GitHub Action that will sync your repository with another repository on a schedule or when a certain event occurs, such as a push or pull request.
    c. GitHub Actions:
        i. actions-template-sync: This GitHub Action can be used to sync a repository with a GitHub template.
        ii. repo-sync/github-sync: This GitHub Action can be used to sync branches between two GitHub repositories.
    d. Another way to achieve synchronous behavior on GitHub is to use a third-party tool. There are a number of third-party tools that can be used to sync GitHub repositories. Some of these tools are free and open source, while others are paid.
    e. Third-party tools:
        i. GitSync: This is a free and open source tool that can be used to sync GitHub repositories.
        ii. CloudBees CodeSync: This is a paid tool that can be used to sync GitHub repositories, GitLab repositories, and Bitbucket repositories.

5. Change control
    a. Change control refers to the software that helps a system transition from it's older to it's newer version. It controls, supports, and manages changes to artifacts, any code change, process change, documentation change etc.
    b. It is used in the Software Development Life Cycle, and although it is not a part of SDLC, it still plays an important role in software development.
    c. Processes of Change Control:

i. Creating a request for change
ii. Reviewing and assessing the request for change
iii. Planning the change
iv. Testing the change
v. Creating a change proposal
vi. Implementing change
vii. Review the change performance
viii. Close the process

d. Importance of Change Control:
i. Improve Performance
ii. Enhance innovation
iii. Increase engagement
iv. For reducing cost
v. For improving the system

e. Points to be considered during Change Control:
i. Reason of change
ii. result of change
iii. Risks involved in change
iv. Resources required for change

EXPERIMENT 12

Aim: Develop a test case for project using White Box Testing(Junit)

Theory: White Box Testing, also known as structural or glass box testing, is a software testing technique that focuses on the internal logic and structure of a software application. In white box testing, testers examine the code, design, and internal components of the software to ensure that it functions correctly according to its specifications. This approach is primarily used by developers and quality assurance engineers to identify and fix issues in the code, as well as to verify that the code is executed as intended. These are some key points related to white box testing:

1.) Internal Perspective: White box testing takes an internal perspective of the software, meaning that it requires knowledge of the internal structure of the application, such as source code, algorithms, and data structures.

2.) Code Coverage: It aims to achieve thorough code coverage, which means that testers strive to execute all possible code paths and statements within the software to uncover any logical errors, defects, or vulnerabilities.

3.) Test Criteria: Test cases in white box testing are designed based on specific criteria, such as statement coverage (ensuring every code statement is executed at least once), branch coverage (testing all possible decision branches), and path coverage (exercising all possible paths through the code).

4.) Types of White Box Testing: White box testing encompasses various techniques, including unit testing, integration testing, and code review. Unit testing, often done using tools like JUnit in Java, focuses on testing individual functions or methods within the code.

5.) Benefits: White box testing can help identify issues early in the development process, improve code quality, and ensure that the software meets its design specifications. It is essential for maintaining the reliability and security of software applications.

to develop a test case using White Box Testing with JUnit for a Java project:

1. Identify the Unit to Test:

- Determine which specific unit or component of your Java project you want to test. This could be a class, a method, or a set of closely related functions.
- Ensure that the unit is well-defined and isolated from other parts of the code, as unit tests should focus on a single, self-contained piece of functionality.

2. Set Up the Test Environment:

- Create a test environment to provide the necessary context for your unit test. This may involve:

  - Initialising mock objects: If your unit depends on external services, databases, or APIs, you can use mock objects or stubs to simulate their behavior in a controlled manner.
  - Preparing test data: Set up the initial state of the unit by providing suitable test data or configurations.

3. Create a Test Class:

- In your test project or test package, create a Java class that will contain your test cases. This class should be separate from the production code and typically follows a naming convention like `MyClassTest` for testing `MyClass`.

4. Write Test Methods:

- Within your test class, create test methods to cover different scenarios and functionalities of the unit under test. Use the `@Test` annotation provided by JUnit to mark these methods as test cases.
- Ensure that each test method has a clear purpose and focuses on a specific aspect of the unit's behavior.

Example : -

```
import org.junit.Test;
import static org.junit.Assert.*;

public class MyClassTest {

   @Test
   public void testMethod1() {
      // Test logic and assertions for scenario 1
   }

   @Test
   public void testMethod2() {
      // Test logic and assertions for scenario 2
   }

   // More test methods...
}
```

5. Define Test Assertions:

- In each test method, define assertions using JUnit's assertion methods. These assertions validate whether the actual results match the expected outcomes, helping determine if the unit behaves as expected.
- Common assertion methods include `assertEquals`, `assertTrue`, `assertFalse`, `assertNull`, and others.

Example:-

```
import org.junit.Test;
import static org.junit.Assert.*;

public class MyClassTest {
    @Test
    public void testMethod1() {
        // Arrange
        MyClass myObject = new MyClass();

        // Act
        int result = myObject.myMethod(2, 3);

        // Assert
        assertEquals(5, result);
    }

    // More test methods...
}
```

6. Execute the Tests:

- Use a test runner to execute the test cases. Many integrated development environments (IDEs) and build tools like Maven and Gradle offer built-in support for running JUnit tests.
- The test runner will execute each test method and report the results, indicating whether each test passed or failed.

7. Analyse Test Results:

- Review the test results to identify any failures or errors. When a test fails, it indicates a problem in the unit's functionality.
- Debug and fix the code associated with the failing test to make it pass.

8. Refactor and Repeat:

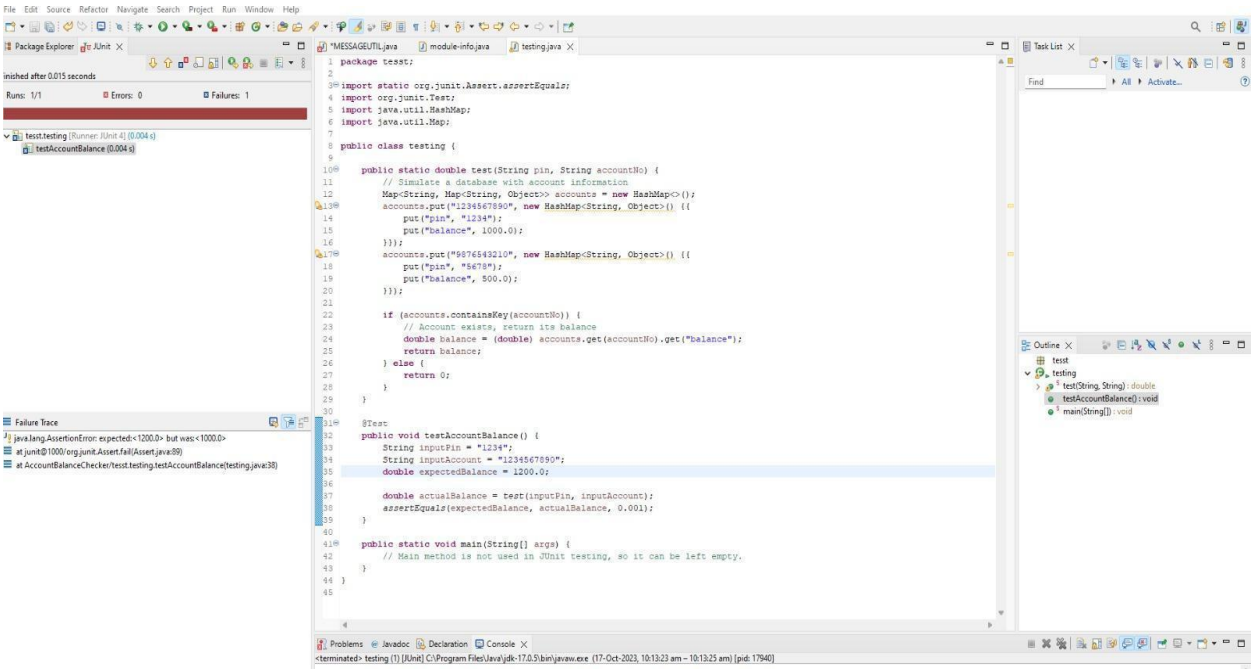
- If necessary, refactor your code and repeat the testing process until all test cases pass.

- Ensure that your unit tests provide comprehensive coverage of the unit's behaviour, including edge cases and boundary conditions.

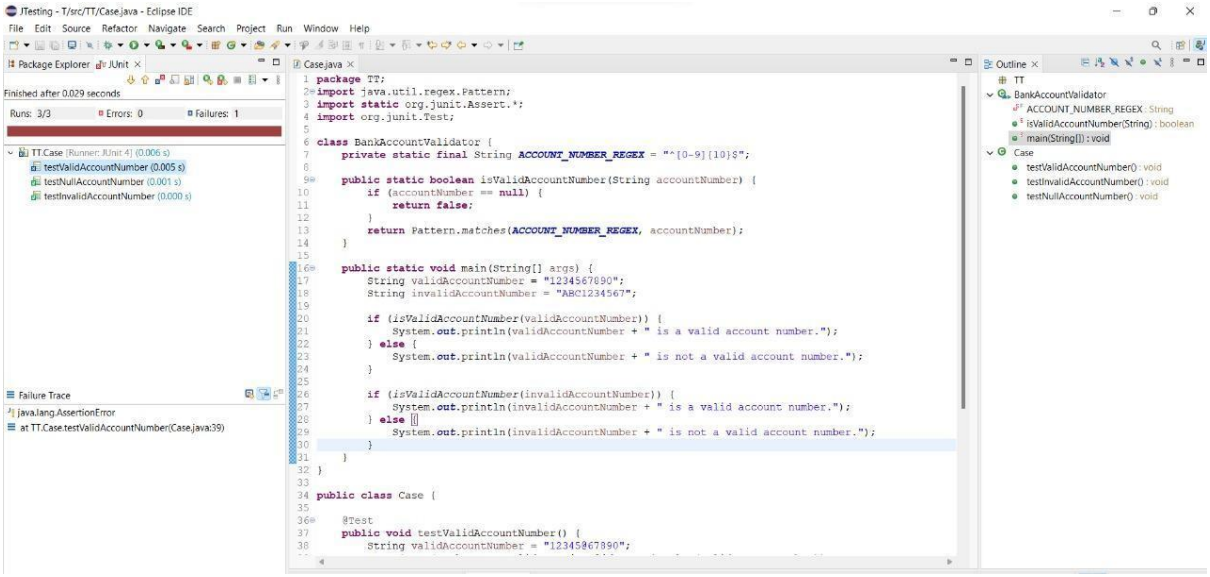
JUnit simplifies the process of white box testing by providing a framework for organising and automating tests. By following these steps, you can thoroughly test your Java code, improve code quality, and ensure that your software components work as intended.

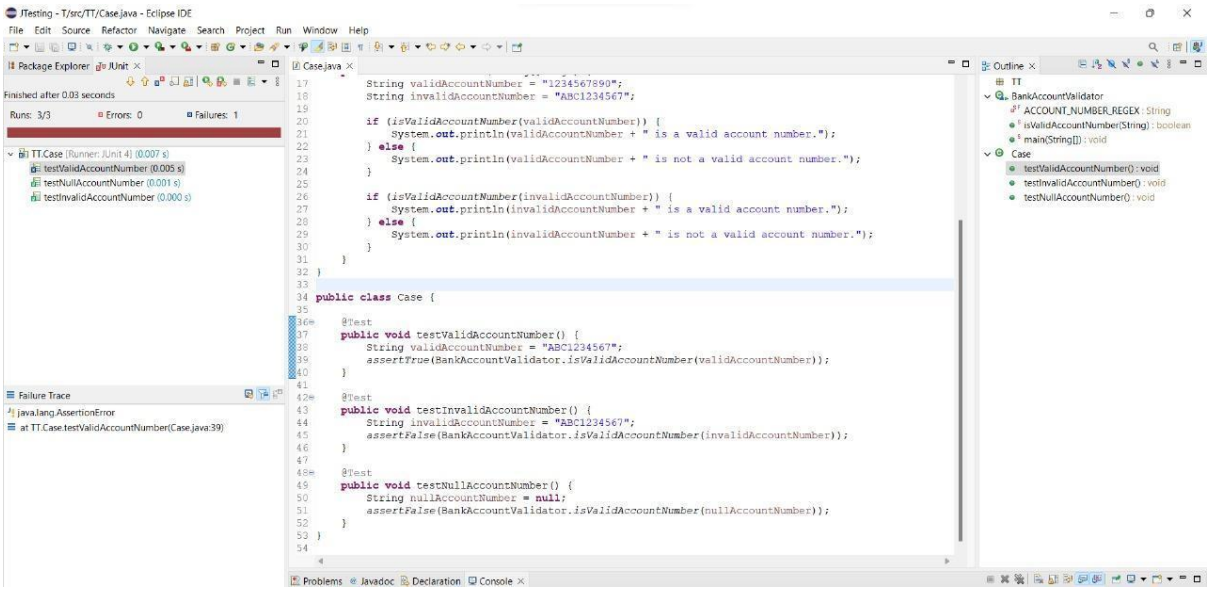
Output:

Test 1: Checking balance .



Test 2: validating account number.





Niyati Savant
2103156
C31

Pursuer 5/10/23 A+

SE
Written Assignment-01

Q Architectural Design: Explain in detail each type with examples

→ A software needs the architectural design to represent the design. As per IEEE, architectural design is the process of defining a collection of hardware and software components and their interfaces to establish the framework for the development of a computer system. A software built for computer-based systems can exhibit one of these many architectural styles. Each style will describe a system category that consists of:

i) A set of components (eg: a database, computational modules) that will perform a function required by the system

ii) The set of connectors will help in co-ordination, communication and co-operation

iii) Conditions that show how components can be integrated to form the system.

iv) Semantic models that help the designer to understand the overall properties of the system.

Types of Architectural designs:

i) Data Centered Architectures

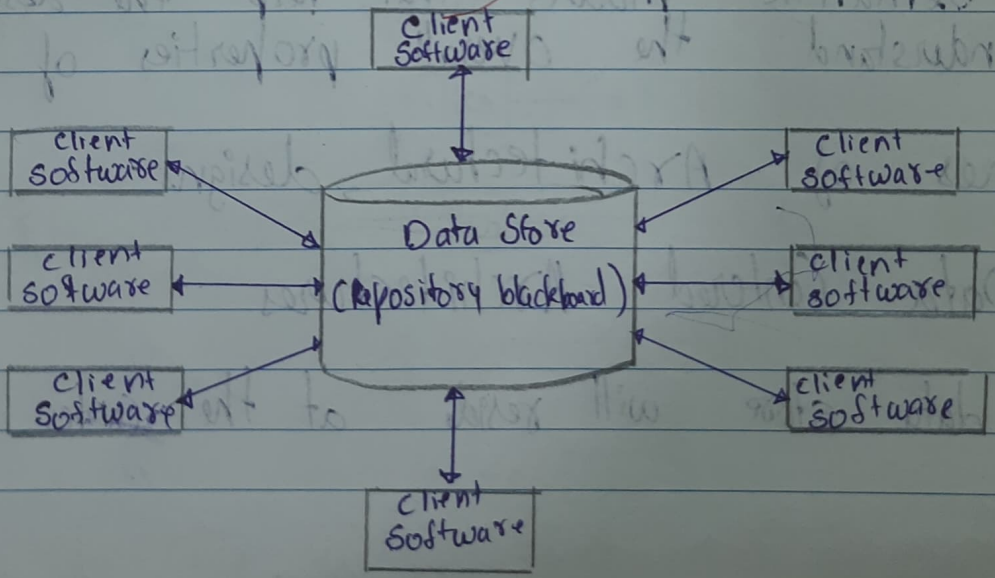
→ A data store will reside at the center and is

Niyati Savant
2103156
C31

accessed frequently by other components that update, add, delete or modify the data present within the store. The following figure illustrates a typical data centered style. The client software access a central repository. Variation of this approach are used to transform the repository into a blackboard when data related to client or data of interest for the client change the notifications to client software. This architecture promotes integrability ie existing components can be changed and new client components can be added to the architecture without the permission or concern of other clients. Data can be passed among clients using blackboard mechanism

Advantage of Data Centered Architecture -

1) Repository of data is independent of clients
2) Client work independent of each other
3) It may be simple to add clients
4) Modification can be easy



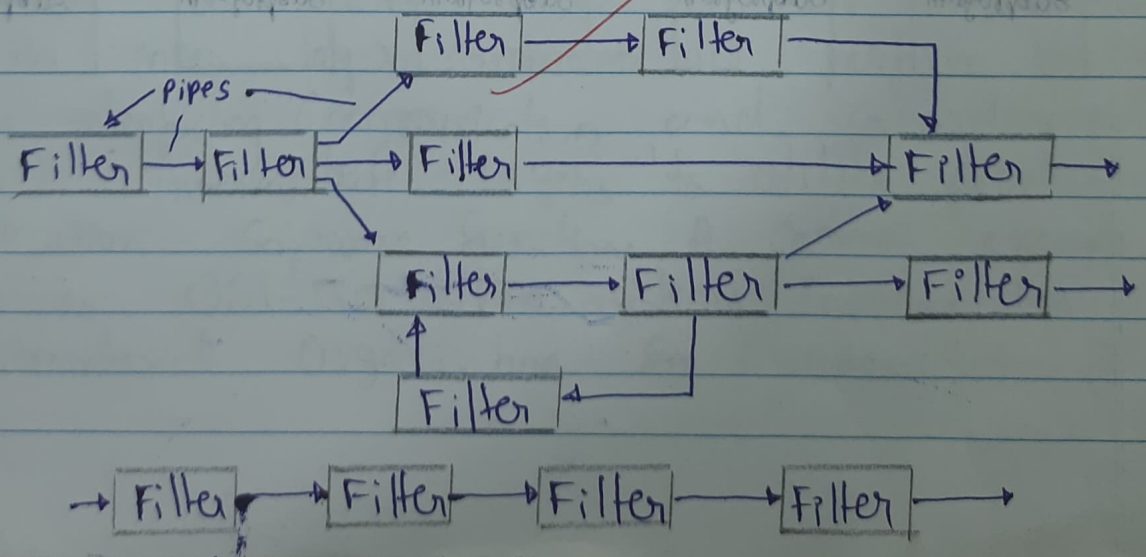
Niyati Savant
210315 6
C31

2) Data flow architectures

→ It is used when input data is transformed into output data through a series of computational manipulative components. The figure represents pipe-and-filter architecture since it uses both pipe and filter and it has a set of components called filters connected by lines. Pipes are used to transmit data from one component to the next. Each filter will work independently and is designed to take data input of a certain form and produces data output to the next filter. If the data flow degenerates into a single line of transforms, then it is termed as batch sequential as it accepts a batch of data and then applies a series of sequential components to transform it.

Advantages of Data Flow architecture:-

1) Encourages upkeep, repurposing and modification
2) Supports concurrent execution

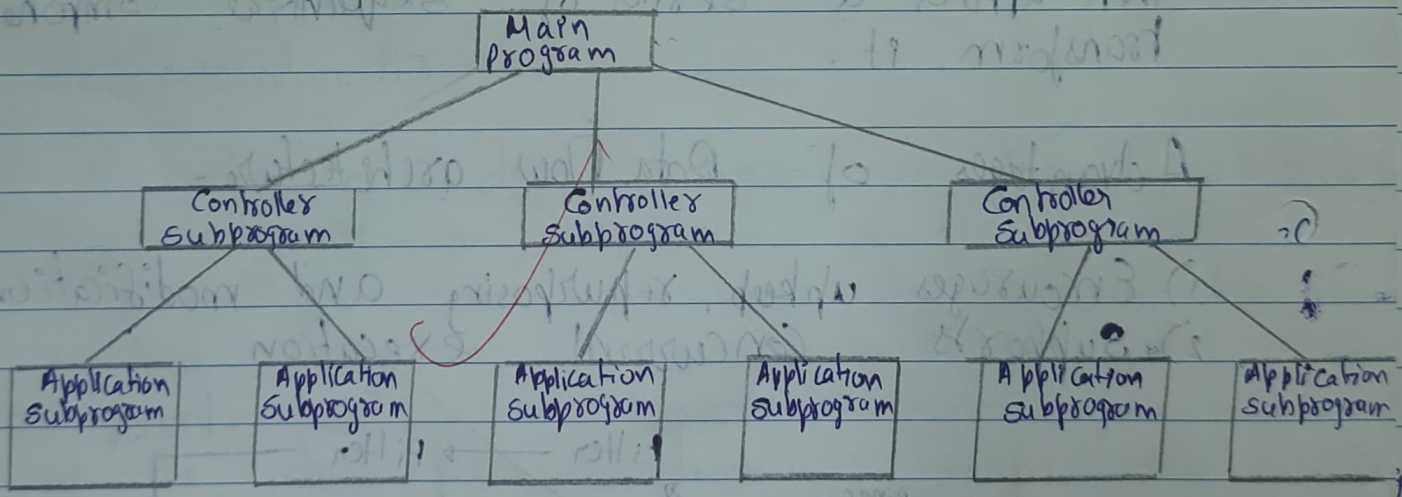


Niyati Savant
210356
C31

3) Call and Return Architectures

→ It is used to create a program that is easy to scale and modify. Two of the many sub-styles are -

- Remote procedure call architecture
  This components is used to present in a main program or sub program; architecture distributed among multiple computers on a network.
- Main program or Subprogram architectures
  The main program structure decomposes into number of subprograms or function into a control hierarchy. Main program contains number of subprograms that can invoke other components.



Niyati Savant
210315 6
C31

4) Object Oriented Architecture

→ The components of a system encapsulate data and the operations that must be applied to manipulate the data. The coordination and communication between the components are established via the message passing. It protects system's integrity and is unaware of the depictions of other items.

Advantages
1) Enables designer to separate a challenge into a collection of autonomous objects.
2) Other objects are aware of object's implementation details allowing changes to be made

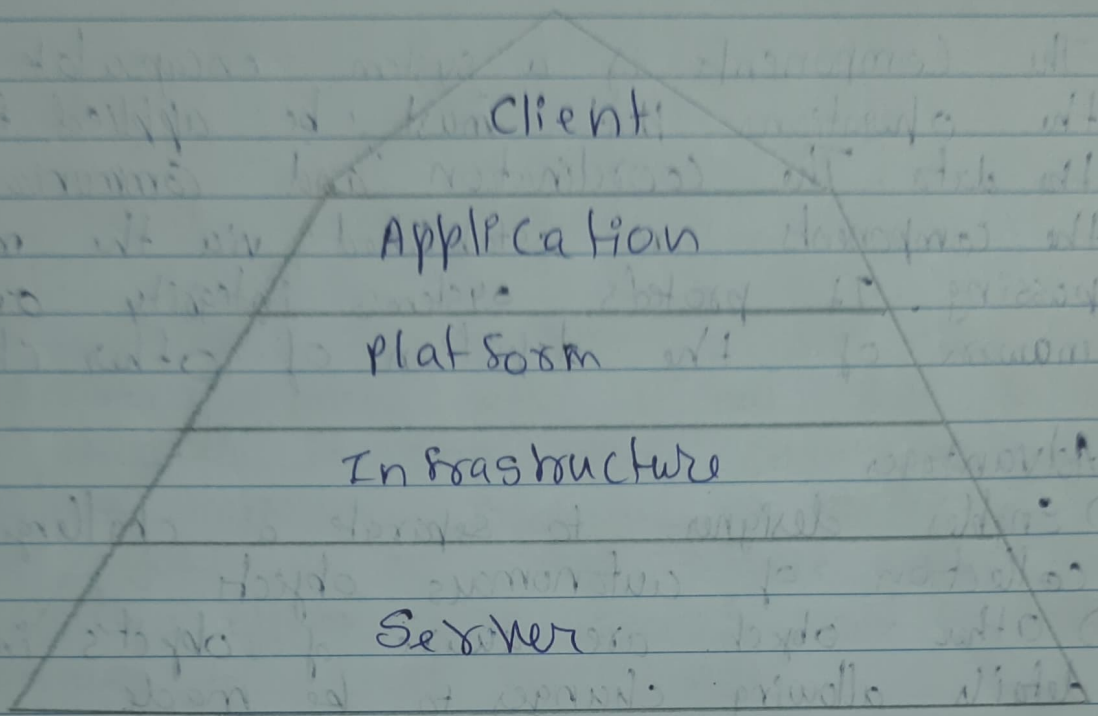
5) Layered Architecture

→ A number of different layers are defined with each layer performing a well-defined set of operations. Each layer will do some operations that becomes closer to machine instruction set progressively. At the outer layer components will receive user interface operations and at the inner layers, components will perform the operating system interfacing (communication and co-ordination with OS). Intermediate layers to utility services and application software functions. A common example of this is OSI-ISO (Open systems Interconnection - International Organisation for standardisation)

Niyati Savant
2103156
C31

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

Communication system



Viyati Savant
2103156
C31 / SE

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

Written Assignment

Viyati 5/10/23 (A+)

Q Explain in detail

a) Software Maintenance

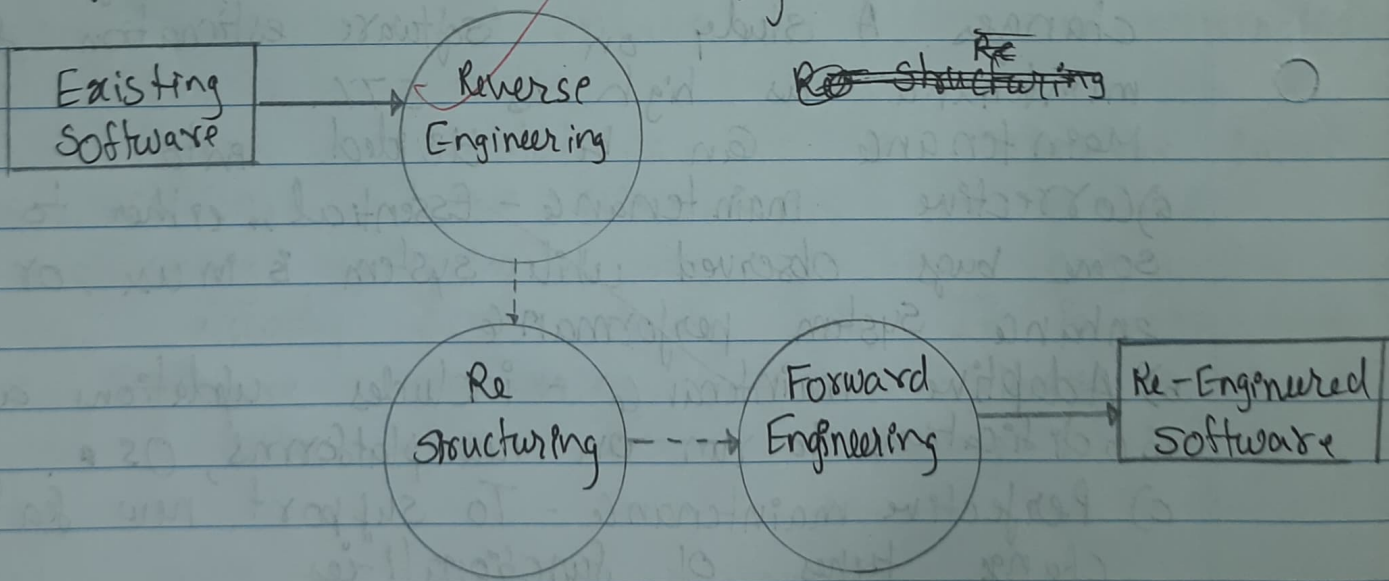
→ The process of modifying and updating a software system after it has been delivered to the customer. It can include fixing bugs, adding new features, improving performance, or updating software. Its goal is to keep software system working correctly, efficiently and securely and to ensure it continues to meet user needs.
They must be performed to correct faults, improve design, implement enhancements refine and the reasons include market conditions, client requirements, host modifications, Organization changes. A study on software estimation found maintenance as high as 67%.
Maintenance can be divided into -
a) Corrective maintenance - Essential, either to rectify some bugs observed while system is in use, or to enhance system performance
b) Adaptive maintenance - includes updations and modifications to run on new platforms, OS.
c) Perfective maintenance - To support new features or change types of functionalities
d) Preventive maintenance - Modifications and updations

Niyati Savant
210315 6
C31

to prevent future problems which though insignificant at present may cause serious issues.

b) Re-Engineering

→ It is a process of software development done to improve maintainability of software system. It is the examination and alteration of system and encompasses a combination of subprocesses like reverse engineering, forward engineering, reconstructing etc.
It is a thorough process where software design is changed and programs are rewritten. Legacy software can't keep tuning with latest technology as hardware becomes obsolete. software updating becomes difficult though functionality remains same.



Niyati Savant
2103156
C31

Its primary goal is to improve quality and maintainability of software system while minimizing risks and costs associated with redevelopment from scratch.
The 3 processes for a re-engineered software are.

1) Reverse Engineering
Process to achieve system specification by thoroughly analyzing, understanding existing system a.k.a a reverse SDLC model. It is the process of recovering design and requirement specification of a product from an analysis of its code.

2) Program Restructuring
Process to restructure and re-construct existing software. It is about re-arranging source code, either in same programming language or from a different one. It can be a source code-restructuring, data-restructuring or both. It does not impact functionality but enhance reliability and maintainability.

3) Forward Engineering
Process of obtaining desired software from specifications that were brought down by reverse engineering assuming some software engineering was already done in the past. It is the same as software engineering and is always carried out only after reverse engineering

Niyati Savant
2103156
C31

c) Reverse Engineering

→ It is the process of extracting knowledge or design information from anything man-made and reproducing it based on extracted information. Its main objective is to check out how the system works and to recreate the object by adding some enhancements.
Since several existing software products, lack proper documentation, are highly unstructured or their structure has degraded through a series of maintenance efforts. As it is the process of recovering design and the requirements specification from code analysis, it provides proper system documentation, assists with maintenance and helps with recovery of lost information. Further, it facilitates software reuse, helps discover unexpected flaws or faults. It also implements innovative processes for specific use. It is helpful in adding new features to existing software, in software testing and recover lost or inaccessible source code and analyze system behavior for security or compliance purposes. It is also used in malware analysis, legacy systems, IP protection, security, compliance.