Project Title: UrbanGarden

Mohib Abbas Sayed – 2103158

Hamza Sayyed – 2103159

Om Shete – 2103163

Experiment No 1

Aim: Write a detailed problem statement for any case study. Justify which process model would be best suited to apply on it.

Theory:

1. Title: UrbanGarden

2. Problem Statement:
   Despite the increasing demand for plants and gardening products, many plant nurseries **struggle to attract** and **retain customers** through their websites.
   Customers often face difficulty **navigating websites**, finding **detailed plant information**, and completing purchases, resulting in **lost sales** and decreased customer satisfaction.
   As a result, there is a need to develop and optimise plant nursery websites to provide-
   a. User-Friendly shopping experience
   b. Wide Variety to select the desired products to shop
   c. Enhancing online experience that encourages customer loyalty and drives sales

3. Traditional Model:
   The Evolutionary Model, also known as the Prototyping Model, can be justified for developing the UrbanGarden platform due to its focus on rapid prototyping, user feedback, and iterative development. Here's why the Evolutionary Model would be a suitable choice:

   Rapid Prototyping:
   - In the Evolutionary Model, a basic prototype of the platform is developed quickly, allowing the team to gather early feedback from stakeholders and potential users.
   - This means that the core functionalities of the platform can be demonstrated to nursery owners and users early in the development process.

   User-Centric Approach:
   - The Evolutionary Model emphasizes involving end-users from the beginning.
   - As UrbanGarden is a platform for nursery owners and users, their early feedback is crucial to ensuring the platform meets their specific needs and expectations.

   Iterative Development:
   - The Evolutionary Model supports an iterative development approach. The initial prototype can be further refined and enhanced based on user feedback and changing requirements.

- Within a 3-month timeframe, multiple iterations can be conducted to continuously improve the platform.

Flexibility and Adaptability:

- The Evolutionary Model is well-suited for projects where requirements are not entirely clear or may evolve during development.
- In the case of UrbanGarden, the platform's functionalities might evolve as nursery owners and users provide feedback and new use cases emerge.

Time-Effective:

- The Evolutionary Model allows the development team to quickly build and demonstrate a functional prototype.
- This enables stakeholders to get a clear idea of what the final product will look like and provide feedback early in the process.

Small Development Team:

- With a team of three developers, the Evolutionary Model is practical as it encourages close collaboration and communication among team members.

Risk Mitigation:

- By receiving early feedback through prototyping, the team can identify potential risks and challenges sooner, allowing for timely adjustments and risk mitigation strategies.

However, it's essential to consider the potential challenges and mitigate them:

Scope Management:

- The Evolutionary Model might be more challenging to manage in terms of defining and controlling the scope, especially if new features and functionalities are continuously added based on user feedback.

Balancing Speed and Quality:

- Rapid prototyping might prioritize speed over perfecting each feature, potentially leading to some limitations in the initial prototypes
- Balancing speed with maintaining quality is crucial.

Documentation:

- In an evolutionary development approach, the focus is on building prototypes and iterating.
- Careful documentation is required to ensure that knowledge is preserved, especially if team members change during the project.

4. Agile Model:

The Agile model can be well-justified for developing the UrbanGarden platform due to its adaptability, collaborative nature, and ability to deliver incremental value within a time-sensitive project. Here's why Agile would be a suitable choice:

Iterative and Incremental Development:

- With Agile, the development process is broken down into smaller iterations (sprints) that typically last 1-4 weeks. This allows the development team to deliver functional increments of the platform within each sprint.
- In a 3-month time frame, multiple iterations can be completed, and valuable features can be delivered to users throughout the development process.

Flexibility and Adaptability:

- Agile embraces changes in requirements, allowing the team to adjust and respond to evolving needs.
- As the project progresses, feedback from nursery owners and users can be incorporated, ensuring the platform aligns better with their expectations.

Regular User Feedback:

- Agile prioritizes constant user feedback through iterative cycles. This enables the team to validate assumptions, understand user needs better, and make improvements based on real-world usage, leading to a more user-friendly and customer-centric platform.

Collaborative Teamwork:

- Agile encourages close collaboration among team members, including developers, designers, and stakeholders.
- With a small team of three developers, communication and coordination are more streamlined, leading to faster decision-making and issue resolution.

Early Delivery of Core Features:

- Agile prioritizes delivering the most valuable features first. Core functionalities, such as nursery registration, product listings, and user registration, can be developed and deployed early to start gathering feedback and attracting users to the platform.

Risk Management:

- Agile allows risks to be identified and addressed early in the development process.
- Frequent testing and continuous integration help detect and resolve issues promptly, reducing the chances of major setbacks or delays.

Time-Boxed Sprints:

- The time-boxed nature of Agile sprints ensures that the team focuses on delivering specific, achievable goals within a fixed timeframe.
- This helps keep the project on track and maintain momentum throughout the development process.

Continuous Improvement:

- Agile encourages retrospectives after each sprint, allowing the team to reflect on their performance, identify areas for improvement, and implement changes in subsequent iterations.

However, it's important to consider the potential challenges:
Scope Management:

- In a 3-month timeframe, the scope of the project needs to be well-defined and prioritized.
- To avoid scope creep, clear communication with stakeholders is essential.

Resource Allocation:

- With a small team, it's crucial to ensure that developers have the necessary skills and expertise to deliver the required functionalities effectively.

Managing Expectations:

- Agile requires close collaboration with stakeholders, and they need to understand the iterative nature of development and the potential for changes during the process.

In conclusion, the Agile model is well-suited for developing the UrbanGarden platform, given the project's time constraints, need for flexibility, and desire for early user feedback. By embracing Agile principles and practices, the development team can deliver a functional and user-friendly platform that meets the needs of nursery owners and users while adapting to changing requirements within the 3-month timeframe.

EXPERIMENT – 02

Aim: Application of Agile Process Model on the project (JIRA)

Theory:

JIRA is a popular project management and issue-tracking tool developed by Atlassian. Kanban, on the other hand, is a visual workflow management methodology that has gained widespread adoption in software development and various other industries. This theory will explore the concepts, uses, and conventions associated with JIRA and the Kanban board.

Overview

JIRA is a versatile project management tool designed to help teams plan, track, and manage their work effectively. It offers a wide range of features that facilitate issue tracking, project management, and collaboration. JIRA is highly customizable, making it suitable for various industries and team sizes.

Key Features of JIRA

a. Issue Tracking: JIRA allows users to create, categorise, and track issues, tasks, and user stories.
b. Custom Workflows: It supports the creation of custom workflows to match your team's unique processes.
c. Agile and Scrum Support: JIRA integrates seamlessly with Agile and Scrum methodologies, enabling sprint planning, backlog management, and more.
d. Reporting and Dashboards: Teams can generate reports and create dashboards to monitor project progress and key metrics.
e. Integration: JIRA integrates with various third-party tools, making it a central hub for project-related activities.

Kanban Board

Overview of Kanban

Kanban is a visual workflow management system that originated in the manufacturing sector but has since been adapted for various industries, including

software development. The Kanban method focuses on visualizing work, limiting work in progress (WIP), and continuously improving the process.

Key Concepts of Kanban

a. Visual Board: A Kanban board is a visual representation of the workflow, consisting of columns and cards that represent tasks or work items.
b. Work in Progress (WIP) Limits: Kanban enforces WIP limits on each column to prevent overloading and maintain a smooth flow of work.
c. Pull System: Work is pulled from one column to the next based on available capacity rather than being pushed onto team members.
d. Continuous Improvement: Kanban promotes a culture of continuous improvement by regularly reviewing and optimizing the workflow.

Uses and Conventions

1. Combining JIRA and Kanban
   Many teams use JIRA and Kanban together to benefit from the strengths of both tools. JIRA can be configured to support Kanban boards, allowing for seamless integration of project management and visual workflow management.

2. Conventions in Kanban Boards:

a. Columns: Typical columns on a Kanban board include "To Do," "In Progress," "Review," and "Done."
b. Card Details: Each card should include task details, such as a title, description, assignee, and due date.
c. WIP Limits: Enforce WIP limits on columns to optimize flow and maintain efficiency.
d. Daily Stand-ups: Teams often hold daily stand-up meetings to discuss progress, impediments, and potential improvements.

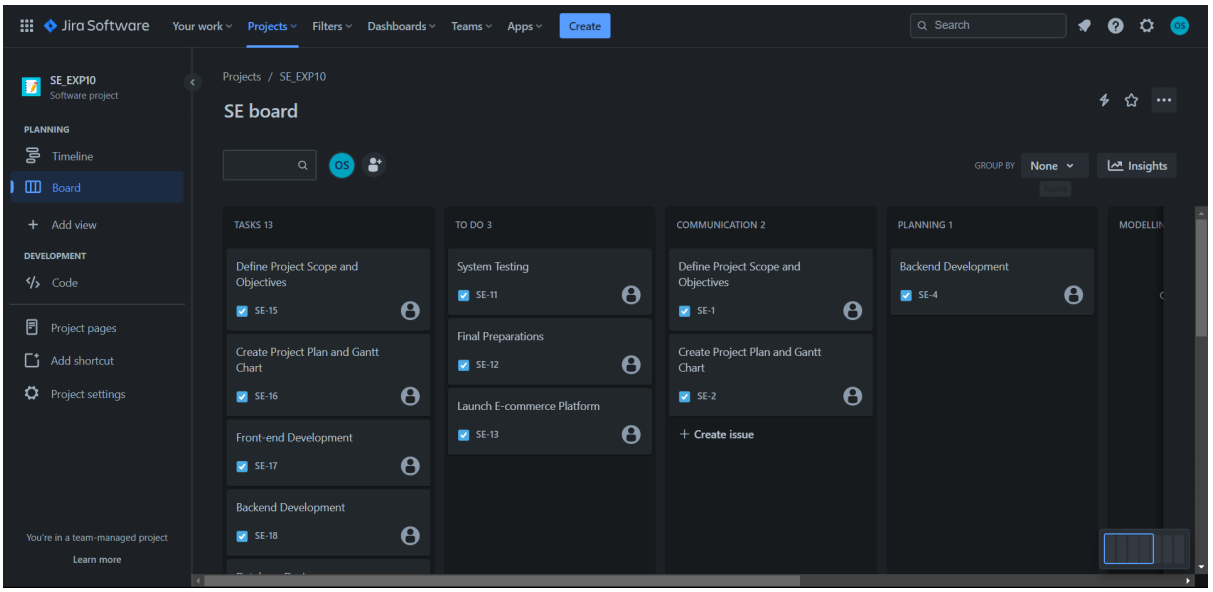
3. Uses of JIRA and Kanban:

a. Bug Tracking: JIRA is commonly used for tracking and managing software bugs.
b. Project Management: JIRA helps teams plan and oversee projects, including tasks and milestones.
c. Continuous Delivery: Kanban boards are instrumental in managing the flow of tasks in continuous delivery pipelines.

Conclusion

JIRA and the Kanban board offer powerful tools for project and visual workflow management, respectively. Combining these two tools can enhance productivity and facilitate effective project planning, tracking, and execution. Adhering to conventions and best practices is crucial for realizing the full potential of JIRA and Kanban in a team's workflow.

Output:



Experiment No. 3

Aim: Develop SRS document in IEEE Format for the project.

Project Title: UrbanGarden

Mohib Abbas Sayed - 2103158

Hamza Sayyed - 2103159

Om Shete - 2103163

Software Requirements Specification

For

Urban Garden

Prepared by

Mohib Abbas Sayed

Hamza Sayyed

Om Shete

Thadomal Shahani Engineering College

31st July 2023

Table of Contents

Revision History

Name	Date	Reason For Changes	Version

1. Introduction

1.1 Purpose

The purpose of this report is to provide an in-depth analysis of the key elements needed to build a successful nursery plant e-commerce website. With the global online plant delivery market experiencing rapid growth, it is essential for nurseries to establish an online presence and provide customers with a convenient and efficient way to purchase plants.

1.2 Document Conventions

Key points have been underscored to highlight significance. Main headings and subheadings are emphasised in bold for added distinction. All points within sections are structured by priority, ensuring essential elements are not overlooked. Abbreviations, familiar to application developers, are employed in specific instances to enhance communication efficiency.

1.3 Intended Audience and Reading Suggestions

The target readership comprises the group of developers tasked with conceptualising and executing the Urban Garden project. Additionally, the content is intended for utilisation by the testing team responsible for scrutinising and assessing the application's performance and design. This document encompasses comprehensive essential details tailored for the software engineering team engaged in the project's execution.

1.4 Product Scope

UrbanGarden encompasses an e-commerce platform enabling nursery vendors to seamlessly register, manage shops, and showcase plant varieties and gardening products. Customers benefit from an intuitive interface, shopping cart, secure checkout, and user profiles. Interactive forums foster community engagement. Admin tools ensure smooth platform management. UrbanGarden also recommends a variety of products based on the location of accessing the website. Prioritised for security, and performance. Thorough testing, documentation, and customer support ensure a seamless experience. UrbanGarden aims to create a vibrant online gardening hub, connecting vendors and customers in a user-friendly ecosystem.

1.5 References

Websites:

- MaterialUI: https://mui.com/
- Redux: https://redux.js.org/
- w3schools: https://www.w3schools.com/

2. Overall Description

2.1 Product Perspective

The traditional method of buying and selling nursery plants involves physical visits to local stores, limited choices, and minimal information. In contrast, the Urban Garden platform revolutionises this process by providing vendors with an easy way to register and showcase their products online, offering customers a wide selection, rich product details, secure transactions, and a vibrant community for knowledge sharing. It transforms nursery shopping into a convenient, diverse, and interactive experience that transcends geographical limitations.

2.2 Product Functions

• Streamlined Transaction Process: UrbanGarden eliminates the requirement for intermediaries in the buying and selling of nursery products, fostering a direct interaction between vendors and customers, resulting in efficient and transparent transactions.

• Effortless Product Search: Users can effortlessly explore a diverse range of nursery items from various locations, saving them the need to physically travel, and thereby enhancing convenience.

• Enhanced Communication: The platform facilitates direct communication between buyers and nursery vendors, enabling easy clarification of queries and concerns, thus promoting effective engagement.

• Time Efficiency: UrbanGarden optimises the buying and selling process, conserving time for both parties involved by offering a seamless online platform for transactions.

2.3 User Classes and Characteristics

UrbanGarden accommodates two distinct user classes: Vendors and Customers. Vendors, comprising nursery owners, engage through the Vendor interface, enabling seamless product showcasing, inventory management, setting up their profiles and direct interaction with customers. Customers, utilising the Customer interface, enjoy convenient browsing, purchase options, and engagement with vendors, fostering a vibrant gardening community.

2.4 Operating Environment (OE)

Since the application is a web application it can work in any browser.

- Device: Computers, Laptops, Tablets.
- Operating System: Windows, Linux distributions, Mac OS, Android
- RAM: 3GB or more
- Disk Space: 20 MB or more.
- Browsers: Mozilla Firefox 30+, Google Chrome 27.0+, Microsoft Edge.
- Internet connection: Strong internet connection with a speed of at least 1 Mbps for the best experience.

2.5 Design and Implementation Constraints

CO-1:

The time allotted for this project is at most 3 months.

CO-2:

The front end of the application is made using ReactJs, and MaterialUI.

CO-3:

NodeJs and ExpressJs will be used as the language for the backend of the application and MongoDB will be used for the database of the application.

CO-4:

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

2.6 User Documentation

Comprehensive guidance will be seamlessly integrated throughout the application's user journey to ensure a user-friendly experience. Future enhancements include the incorporation of a user-assisting chatbot. Clear instructions will be available during form completion, photo uploads, and location inputs. Inaccurate or erroneous entries will trigger informative error messages, preventing user frustration and enhancing application usability.

2.7 Assumptions and Dependencies

AS-1:

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

AS-2:

The users of the application should have basic knowledge of uploading images and location.

DE-1:

The application will require the MERN framework as a dependency since we have used ExpressJs as the backend language.

DE-2:

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

DE-3:

*For maps and geolocation, we will be using Mapbox APIs.

3. External Interface Requirements

3.1 User Interfaces

UI-1:

The website will start with a landing page. The landing page will have all information about the web application. It will include instructions for maintaining the smooth functioning of the web application.

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. This will also include a search bar to provide a smooth search of desired/required products.

UI-3:

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

UI-4:

There will be alerts and pop-ups which appear in case the user makes a mistake while using the application.

UI-5:

The interface will be responsive for all screen sizes as much as possible to provide the users with a seamless experience.

3.2 Hardware Interfaces

N/A

3.3 Software Interfaces

• Browsers: Mozilla Firefox 30+, and Google Chrome 27.0+ are the preferred browsers.

• Operating System: Android, Windows 7, 8, 10, Mac OS, Linux distributions.

3.4 Communication Interfaces

The application will be using HTTPS protocol.

4. System Features

4.1 Authentication and Authorization

4.1.1 Description and Priority:

The application will have multiple users and so authentication becomes a high-priority system feature. The application will be using the JWT authentication module in order to implement this functionality. When the user creates a new account on the application, they will have to provide their email address and password. The password must be at least 8 characters long and must have at least one uppercase character, one digit and one special character. The passwords in the system will be hashed and stored so that no other person can get to know the password.

4.1.2 Response Sequences:

Once the user registers in the application, they will be guided to a Home page where they can start using the application. There will also be a logout button on the navigation bar. On clicking the logout button, the user will be logged out.

4.1.3 Functional Requirements:

REQ-1: We use the JWT authentication module for authentication and authorization functionality. The authentication will be session-based authentication.

4.2 Add Nursery Feature

4.2.1 Description and Priority:

Features to add nurseries on the website will be provided to the user (owner in this case). The owner will have to provide all specifications about the nursery which he intends to put on the lease. The feature is of high priority as the application is based on owners adding nurseries on the website to get more potential buyers.

4.2.2 Response Sequences:

The user will have to provide all the necessary information about the nursery. On filling out all the information the user will be redirected to a page where he will have the option to add photographs of the nursery. If there are any errors in the description of the nursery, they will be pointed out using alerts to notify the user.

4.2.3 Functional Requirements:

REQ-1: The information provided will be validated using JavaScript on the client side of the application.

4.3 Add Photographs Feature:

4.3.1 Description and Priority:

The user gets to add photographs of the nursery products which he intends to put on the lease. This is again a high-priority feature. The user gets the option to choose images using the file system or simply drag and drop the pictures into the provided area.

4.3.2 Response Sequences:

The user can add images using the drag-and-drop functionality or simply choose images from the files that he wants to upload. In case any image exceeds the maximum file size the user will be notified using alerts. On adding the images, the user is redirected to the add location page.

4.3.3 Functional Requirements:

REQ-1: For the drag and drop functionality we use the Dropzone JavaScript library.

4.4 Add Product Feature

4.4.1 Description and Priority:

The user gets to add the product of the nursery. The user will simply have to add details of the nursery product in order to let the user know more about their product if they want to sell it faster. The priority of the feature is moderate.

4.4.2 Response Sequences:

On adding the product, the nursery product is successfully added to the database and the user is redirected to the Add new product page of the website.

4.4.3 Functional Requirements:

REQ-1: The information provided will be validated using JavaScript on the client side of the application.

4.5 Search Nurseries Feature:

4.5.1 Description and Priority:

The user can search different nurseries using the search functionality. This feature is of high priority and is primarily for experienced users. The user can search based on his requirements such as desired location, area, type etc.

4.5.2 Response Sequence:

On filling in the required information based on desired features, the results will be displayed on the webpage.

4.5.3 Functional Requirements:

REQ-1: The information provided will be validated using JavaScript on the client side of the application.

4.6 Shortlisting Nurseries Feature

4.6.1 Description and Priority:

This feature will allow users to shortlist or save nursery products that they are interested in, for future reference. This is a moderate priority feature.

4.6.2 Response Sequence:

Once the user clicks on the shortlist button the nursery product will be added to the list of nursery products that have been shortlisted by that particular user.

4.6.3 Functional Requirements:

REQ-1: For shortlisting nurseries, AJAX calls will be used. Therefore, JQuery will be used on the client side.

4.7 Delete Nurseries Feature

4.7.1 Description and Priority:

This feature will allow the owner of the nursery to remove the nursery from the system. This is a moderate priority feature.

4.7.2 Response Sequence:

Once the user clicks on the delete nursery button, a popup will appear asking the user if he is sure if he wants to delete the nursery.
If the user responds if Yes, then the nursery will be removed from the system.

4.7.3 Functional Requirements:

REQ-1: For deleting nurseries, AJAX calls will be used. Therefore, JQuery will be used on the client side.

4.8 Deleting Shortlisted Nursery Product Feature

4.8.1 Description and Priority:

This feature will allow users to remove nursery products shortlisted by them. This feature is of moderate priority.

4.8.2 Response Sequence:

Once the user clicks on the remove button the nursery product will be removed from the list that has been shortlisted by that particular user.

4.8.3 Functional Requirements:

REQ-1: For removing shortlisting nurseries, AJAX calls will be used.

Therefore, JQuery will be used on the client side.

5. Other Nonfunctional Requirements

5.1 Performance Requirements

5.1.1 Scalability:

The application should be scalable and should perform without any interruption for all users.

5.2 Safety Requirements

• A backup power supply should be present for the server so that it does not stop functioning in case of power failure.

• API keys of the APIs used should not be made open source.

• The credentials of the Database connection need to be hidden in the env file and made to be ignored.

• Code backup should be taken at regular time intervals.

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 use the HTTPS protocol for security.

• POST requests are used for transferring information regarding authentication, adding nurseries, adding advertisements etc. through forms.

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 Availability:

• The system should be available at all times.

• The system should be reliable and there should be no loss of data in case the server breaks down when operations are going on.

5.4.3 Maintainability:

The code for the application should be written cleanly and should be well documented. The code should contain comments to help new programmers and developers make changes in the application.

5.4.4 Testability:

The code should be written with proper test cases to be tested upon so that no errors during production take place.

5.5 Business Rules

The administrator of the application has full permission to control the system.

6. Other Requirements

Appendix A: Glossary

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

Appendix B: Analysis Models

ER Diagram of the Application

Use Case Diagram for the application

EXPERIMENT. NO: 04

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

Theory:

What is a data flow diagram?

DFD is the abbreviation for Data Flow Diagram. The flow of data of a system or a process is represented by DFD. It also gives insight into the inputs and outputs of each entity and the process itself. DFD does not have a control flow and no loops or decision rules are present. Specific operations depending on the type of data can be explained by a flowchart.

It is a graphical tool, useful for communicating with users,managers and other personnel. itis useful for analyzing existing as well as proposed system.

It provides an overview of

- What data is system processes.
- What transformation are performed.
- What data are stored.
- What results are produced , etc.

Data Flow Diagram can be represented in several ways. The DFD belongs to structured-analysis modeling tools. Data Flow diagrams are very popular because they help us to visualize the major steps and data involved in software-system processes.

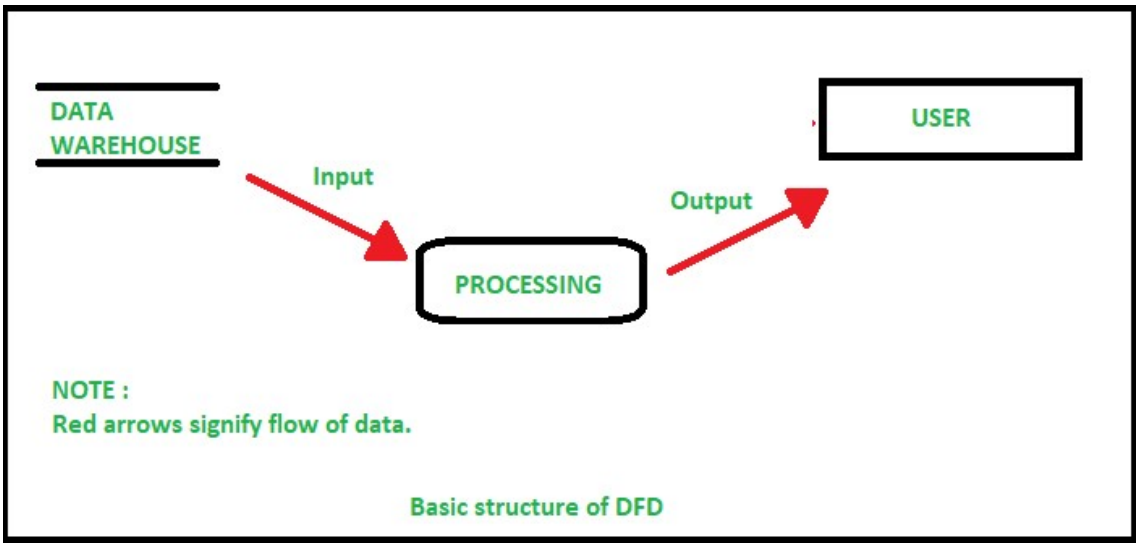
Components of DFD

The Data Flow Diagram has 4 components:

- Process Input to output transformation in a system takes place because of process function. The symbols of a process are rectangular with rounded corners, oval, rectangle or a circle. The process is named a short sentence, in one word or a phrase to express its essence
- Data Flow Data flow describes the information transferring between different parts of the systems. The arrow symbol is the symbol of data flow. A relatable name should be given to the flow to determine the information which is being moved. Data flow also represents material along with information that is being moved. Material shifts are modeled in systems that are not merely informative. A given flow should only

transfer a single type of information. The direction of flow is represented by the arrow which can also be bi-directional.

- Warehouse The data is stored in the warehouse for later use. Two horizontal lines represent the symbol of the store. The warehouse is simply not restricted to being a data file rather it can be anything like a folder with documents, an optical disc, a filing cabinet. The data warehouse can be viewed independent of its implementation. When the data flow from the warehouse it is considered as data reading and when data flows to the warehouse it is called data entry or data updating.
- Terminator The Terminator is an external entity that stands outside of the system and communicates with the system. It can be, for example, organizations like banks, groups of people like customers or different departments of the same organization, which is not a part of the model system and is an external entity. Modeled systems also communicate with terminator.



Basic structure of DFD

Rules for creating DFD

- The name of the entity should be easy and understandable without any extra assistance(like comments).
- The processes should be numbered or put in ordered list to be referred easily.
- The DFD should maintain consistency across all the DFD levels.
- A single DFD can have a maximum of nine processes and a minimum of three processes.

Symbols Used in DFD

- Square Box: A square box defines source or destination of the system. It is also called entity. It is represented by rectangle.
- Arrow or Line: An arrow identifies the data flow i.e. it gives information to the data that is in motion.
- Circle or bubble chart: It represents as a process that gives us information. It is also called processing box.
- Open Rectangle: An open rectangle is a data store. In this data is store either temporary or permanently.

Levels of DFD

DFD uses hierarchy to maintain transparency thus multilevel DFD's can be created. Levels of DFD are as follows:

- 0-level DFD: It represents the entire system as a single bubble and provides an overall picture of the system.
- 1-level DFD: It represents the main functions of the system and how they interact with each other.
- 2-level DFD: It represents the processes within each function of the system and how they interact with each other.
- 3-level DFD: It represents the data flow within each process and how the data is transformed and stored.

Advantages of DFD

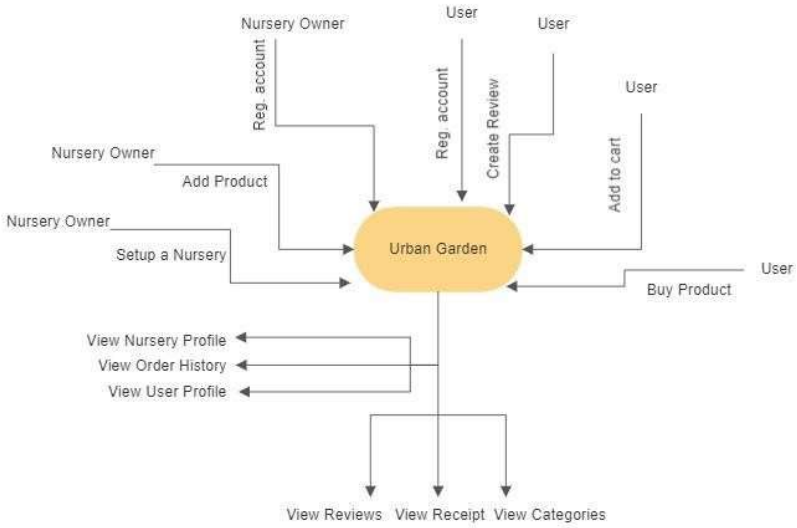
- It helps us to understand the functioning and the limits of a system.
- It is a graphical representation which is very easy to understand as it helps visualize contents.
- Data Flow Diagram represent detailed and well explained diagram of system components.
- It is used as the part of system documentation file.
- Data Flow Diagrams can be understood by both technical or nontechnical person because they are very easy to understand.

Disadvantages of DFD

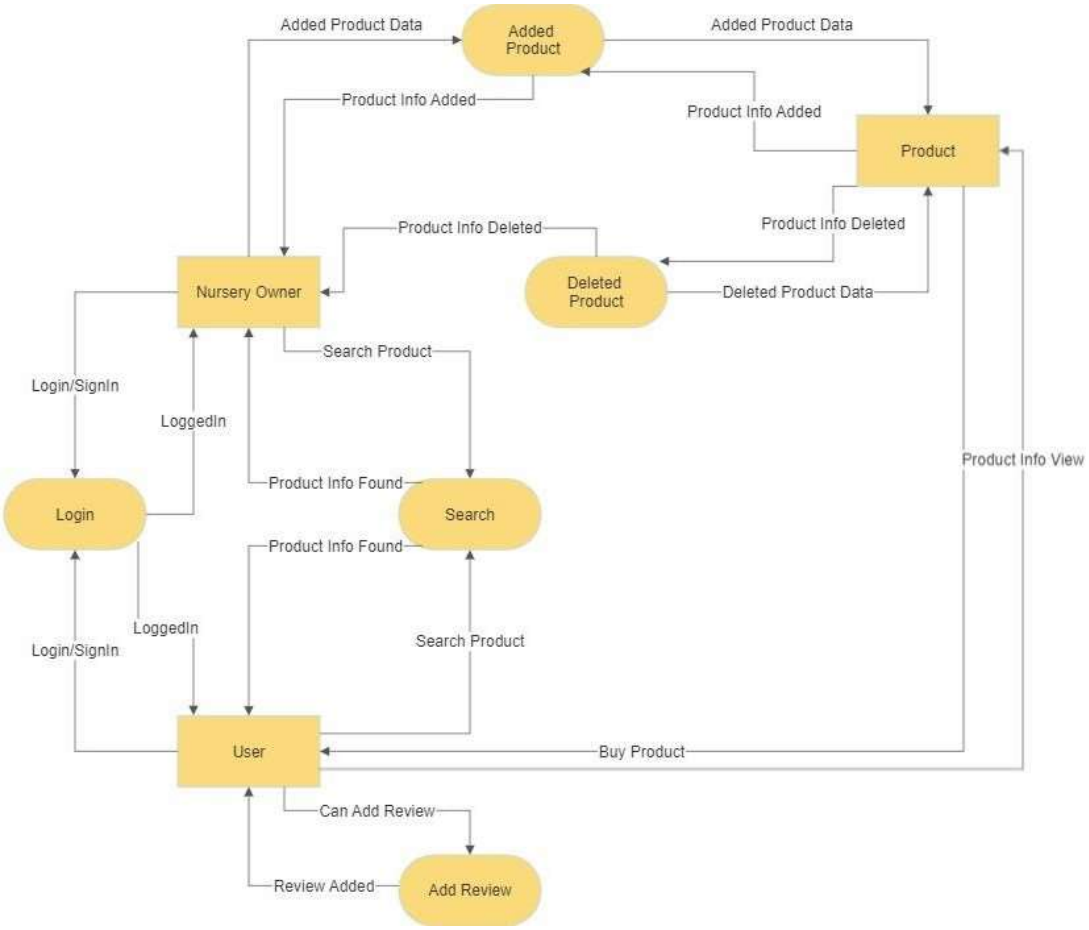
- At times DFD can confuse the programmers regarding the system.
- Data Flow Diagram takes long time to be generated, and many times due to this reasons analysts are denied permission to work on it.

Diagrams:

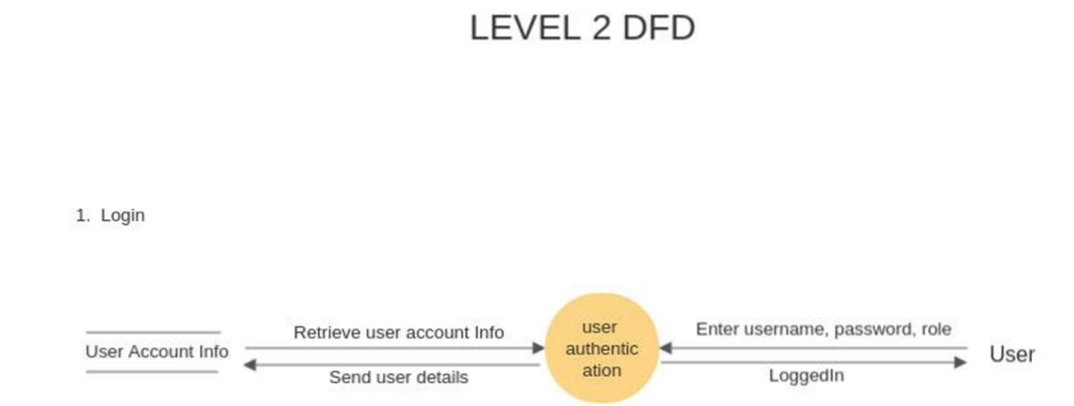
Level 0:

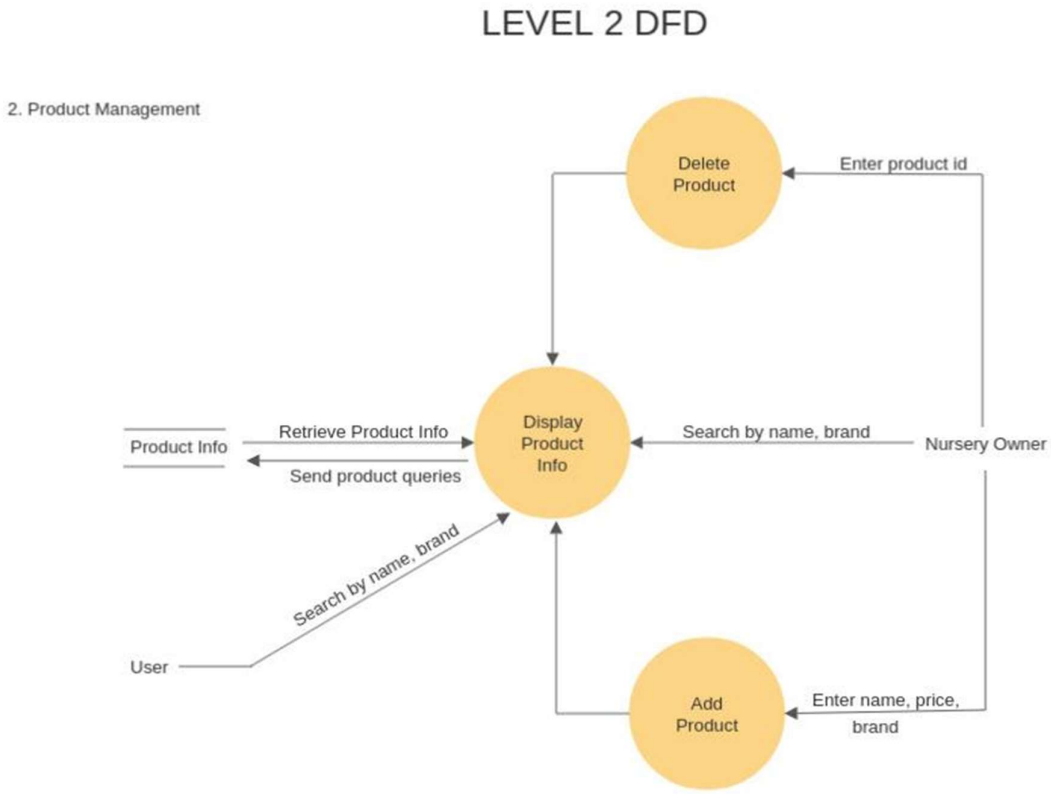


Level 1:


Added Product Data
Added Product
Added Product Data
Product Info Added
Product Info Added
Product
Product Info Deleted
Product Info Deleted
Nursery Owner
Deleted Product
Deleted Product Data
Search Product
Login/SignIn
LoggedIn
Product Info View
Product Info Found
Login
Search
Product Info Found
LoggedIn
Login/SignIn
Search Product
User
Buy Product
Can Add Review
Review Added
Add Review

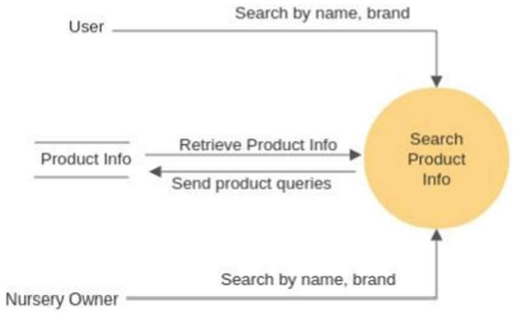
Level 2:


LEVEL 2 DFD
1. Login
User Account Info
Retrieve user account Info
Send user details
user authentic ation
Enter username, password, role
LoggedIn
User


LEVEL 2 DFD
2. Product Management
Delete Product
Enter product id
Product Info
Retrieve Product Info
Send product queries
Display Product Info
Search by name, brand
Nursery Owner
Search by name, brand
User
Add Product
Enter name, price, brand

3. Search Management

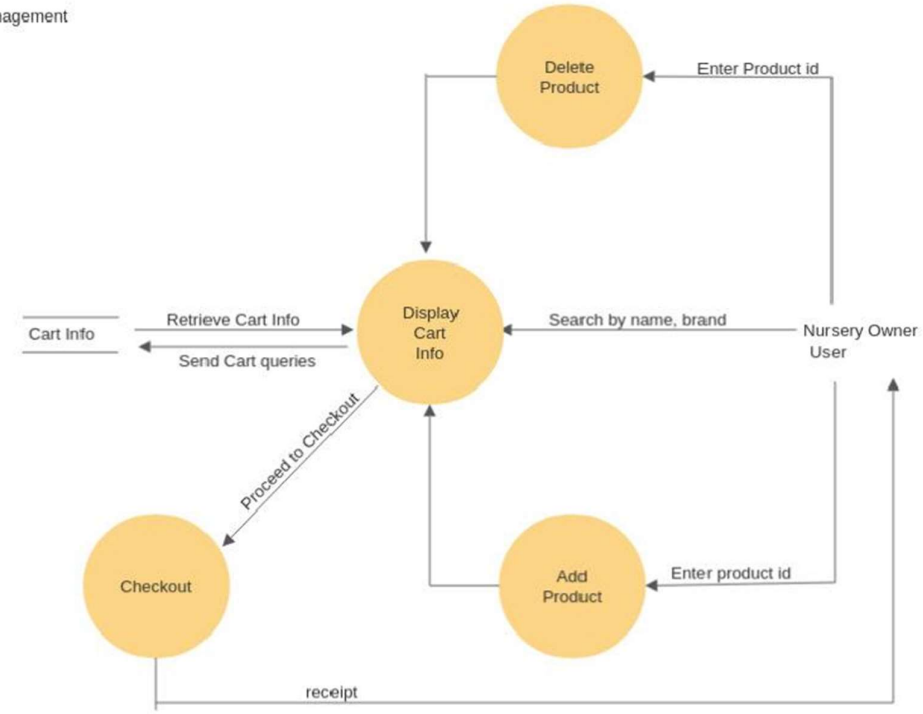
LEVEL 2 DFD



4. Review

LEVEL 2 DFD

5. Cart Management



EXPERIMENT. NO: 05

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

Theory:

Activity diagram is another important diagram in UML to describe the dynamic aspects of the system.

An activity diagram is basically a flowchart to represent the flow from one activity to another activity. The activity can be described as an operation of the system.

The control flow is drawn from one operation to another. This flow can be sequential, branched, or concurrent. Activity diagrams deal with all types of flow control by using different elements such as fork, join, etc

Activity is a particular operation of the system. Activity diagrams are not only used for visualizing the dynamic nature of a system but they are also used to construct the executable system by using forward and reverse engineering techniques. The only missing thing in the activity diagram is the message part.

It does not show any message flow from one activity to another. An activity diagram is sometimes considered a flowchart. Although the diagrams look like a flowchart, they are not. It shows different flows such as parallel, branched, concurrent, and single.

The purpose of an activity diagram can be described as –

- Draw the activity flow of a system.
- Describe the sequence from one activity to another.

- Describe the parallel, branched and concurrent flow of the system.

Activity Diagram Notations –

1. **Initial State –** The starting state before an activity takes place is depicted using the initial state.
   Figure – notation for initial state or start state A process can have only one initial state unless we are depicting nested activities. We use a black-filled circle to depict the initial state of a system. For objects, this is the state when they are instantiated. The Initial State from the UML Activity Diagram marks the entry point and the initial Activity State. For example – Here the initial state is the state of the system before the application is opened.
   Figure – initial state symbol being used

2. **Action or Activity State –** An activity represents the execution of an action on objects or by objects. We represent an activity using a rectangle with rounded corners. Basically, any action or event that takes place is represented using an activity.
   Figure – notation for an activity state For example – Consider the previous example of opening an application opening the application is an activity state in the activity diagram.
   Figure – activity state symbol being used

3. **Action Flow or Control flows –** Action flows or Control flows are also referred to as paths and edges. They are used to show the transition from one activity state to another.
   Figure – notation for Control Flow An activity state can have multiple incoming and outgoing action flows. We use a line with an arrowhead to depict a Control Flow. If there is a constraint to be adhered to while making the transition it is mentioned on the arrow. Consider the example – Here both the states transit into one final state using action flow symbols i.e. arrows.

Figure – using action flows for transitions

4. **Decision node and Branching** – When we need to make a decision before deciding the flow of control, we use the decision node.
   Figure – notation for decision node The outgoing arrows from the decision node can be labelled with conditions or guard expressions. It always includes two or more output arrows.
   Figure – an activity diagram using decision node

5. **Guards** – A Guard refers to a statement written next to a decision node on an arrow sometimes within square brackets.

   Figure – guards being used next to a decision node The statement must be true for the control to shift along a particular direction. Guards help us know the constraints and conditions which determine the flow of a process.

6. **Fork** – Fork nodes are used to support concurrent activities.
   Figure – fork notation When we use a fork node when both the activities get executed concurrently i.e. no decision is made before splitting the activity into two parts. Both parts need to be executed in case of a fork statement. We use a rounded solid rectangular bar to represent a Fork notation with an incoming arrow from the parent activity state and outgoing arrows towards the newly created activities. For example: In the example below, the activity of making coffee can be split into two concurrent activities and hence we use the fork notation.
   Figure – a diagram using a fork

7. **Join** – Join nodes are used to support concurrent activities converging into one. For join notations we have two or more incoming edges and one outgoing edge.

Figure – join notation for example – When both activities i.e. steaming the milk and adding coffee get completed, we converge them into one final activity.
Figure – a diagram using join notation

8. **Merge or Merge Event –** Scenarios arise when activities which are not being executed concurrently have to be merged. We use the merge notation for such scenarios. We can merge two or more activities into one if the control proceeds onto the next activity irrespective of the path chosen.
   Figure – merge notation For example – In the diagram below: we can't have both sides executing concurrently, but they finally merge into one. A number can't be both odd and even at the same time.
   Figure – an activity diagram using merge notation

9. **Swimlanes –** We use swimlanes for grouping related activities in one column. Swimlanes group related activities into one column or one row. Swimlanes can be vertical and horizontal. Swimlanes are used to add modularity to the activity diagram. It is not mandatory to use swimlanes. They usually give more clarity to the activity diagram. It's similar to creating a function in a program. It's not mandatory to do so, but, it is a recommended practice.
   Figure – swimlanes notation We use a rectangular column to represent a swimlane as shown in the figure above. For example – Here different sets of activities are executed based on whether the number is odd or even. These activities are grouped into a swim lane.
   Figure – an activity diagram making use of swimlanes

10. **Time Event – Figure –** time event notation We can have a scenario where an event takes some time to complete. We use an hourglass to represent a time event. For example – Let us assume that the

processing of an image takes a lot of time. Then it can be represented as shown below.
Figure – an activity diagram using time event

11. **Final State or End State** – The state that the system reaches when a particular process or activity ends is known as a Final State or End State. We use a filled circle within a circle notation to represent the final state in a state machine diagram. A system or a process can have multiple final states.

    Figure – notation for final state

Uses of an Activity Diagram –

- Dynamic modelling of the system or a process.
- Illustrate the various steps involved in a UML use case.
- Model software elements like methods, operations and functions.
- We can use Activity diagrams to depict concurrent activities easily.
- Show the constraints, conditions and logic behind algorithms.

State Diagram:

Basic components of a state chart diagram –

1. **Initial state** – We use a black-filled circle to represent the initial state of a System or a class.
   Figure – initial state notation

2. **Transition** – We use a solid arrow to represent the transition or change of control from one state to another. The arrow is labelled with the event which causes the change in state.
   Figure – transition

3. **State** – We use a rounded rectangle to represent a state. A state represents the conditions or circumstances of an object of a class at an instant in time.
   Figure – state notation

4. **Fork** – We use a rounded solid rectangular bar to represent a Fork notation with an incoming arrow from the parent state and outgoing arrows towards the newly created states. We use the fork notation to represent a state splitting into two or more concurrent states.
   Figure – a diagram using the fork notation

5. **Join** – We use a rounded solid rectangular bar to represent a Join notation with incoming arrows from the joining states and outgoing arrows towards the common goal state. We use the join notation when two or more states concurrently converge into one on the occurrence of an event or events.
   Figure – a diagram using join notation

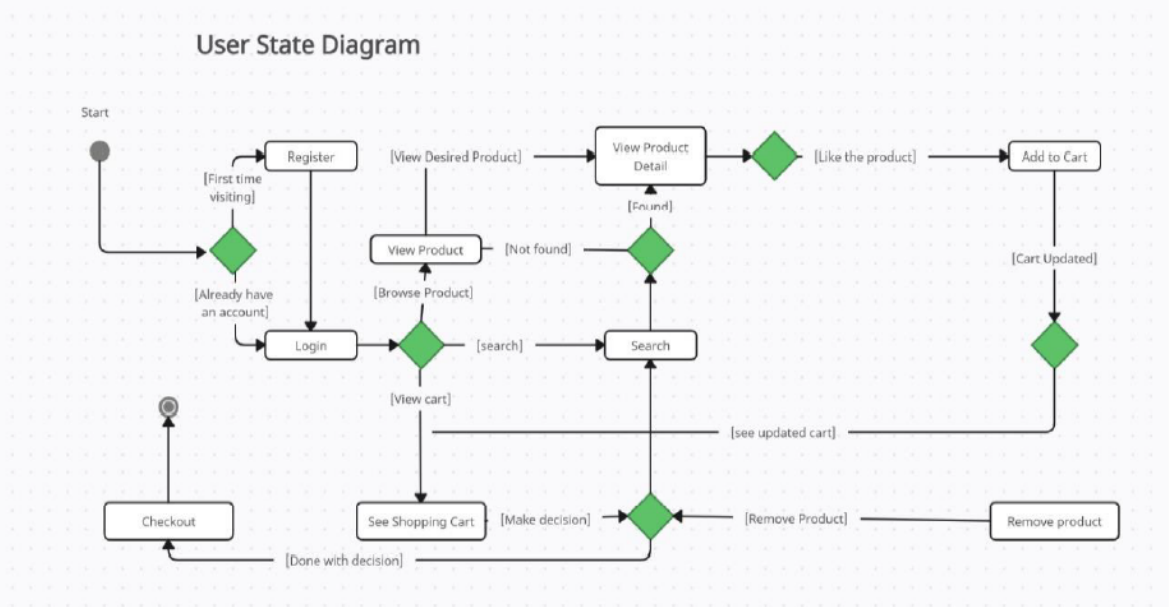
6. **Self-transition** – We use a solid arrow pointing back to the state itself to represent a self-transition. There might be scenarios when the state of the object does not change upon the occurrence of an event. We use self-transitions to represent such cases.
   Figure – self transition notation

7. **Composite state** – We use a rounded rectangle to represent a composite state. We represent a state with internal activities using a composite state.
   Figure – a state with internal activities

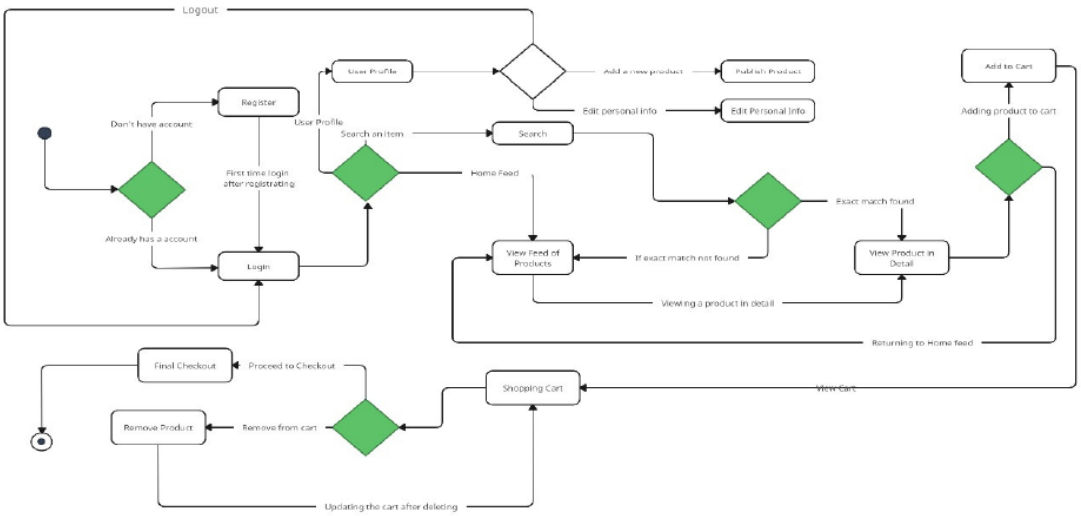
8. **Final state** – We use a filled circle within a circle notation to represent the final state in a state machine diagram.
   Figure – final state notation

Uses of statechart diagram –

- We use it to state the events responsible for the change in state (we do not show what processes cause those events).
- We use it to model the dynamic behaviour of the system.
- To understand the reaction of objects/classes to internal or external stimuli.



USE STATE DIAGRAM FOR NURSERY OWNER



EXPERIMENT. NO: 06

Aim: Develop Use Case Diagram for the project (Smart Draw, Lucid Chart).

Theory:

A use case diagram is used to represent the dynamic behavior of a system. It encapsulates the system's functionality by incorporating use cases, actors, and their relationships. It models the tasks, services, and functions required by a system/subsystem of an application. It depicts the high-level functionality of a system and also tells how the user handles a system.

The main purpose of a use case diagram is to portray the dynamic aspect of a system. It accumulates the system's requirement, which includes both internal as well as external influences. It invokes persons, use cases, and several things that invoke the actors and elements accountable for the implementation of use case diagrams. It represents how an entity from the external environment can interact with a part of the system.

Following are the purposes of a use case diagram given below:

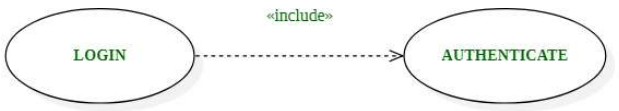
1. It gathers the system's needs.
2. It depicts the external view of the system.
3. It recognizes the internal as well as external factors that influence the system.
4. It represents the interaction between the actors.

Difference between **<<include>>** and **<<extend>>** in **Use Case Diagram:**

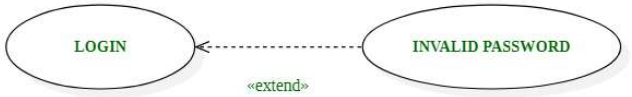
- ***<<include>>*** extends **Base Use Case** and it specifies that an **Included Use Case** must run successfully to complete Base Use Case. The Base Use Case is incomplete in the absence of an Included Use Case. The Included Use Case can be Base Use Case itself or it might be shared by a number of distinct Base Use Cases.

- ***<<extend>>*** on the other end , is used to add an **Extended Use Case** which extends the **Base Use Case.** Base Use Case can run successfully even without invoking/calling extended use case called Optional Use Case. The Base Use Case is complete in itself but under certain conditions it would require to refer to extension condition.

The representation of <<include>> and <<extend>> is as below :-



LOGIN is the BASE USE CASE and AUTHENTICATE is the INCLUDED USE CASE.

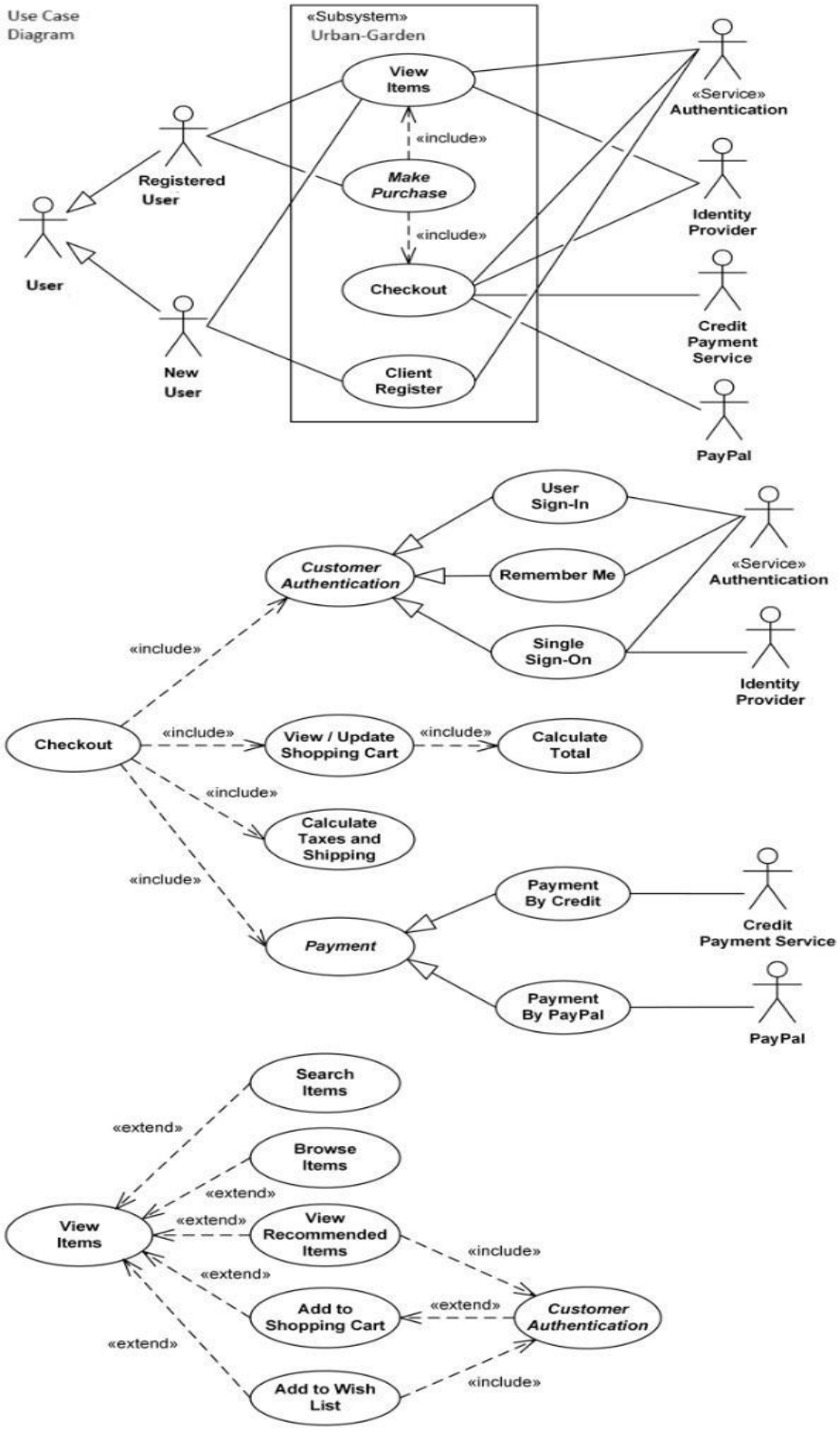


ILOGIN is the BASE USE CASE and INVALID PASSWORD is the EXTENDED USE CASE.

How to plan use case?
Following example will illustrate on how to plan use cases:

Use Case: What is the main objective of this use case. For eg. Adding a software component, adding certain functionality etc.
Primary Actor: Who will have the access to this use case. In the above examples, administrators will have the access.
Scope: Scope of the use case
Level: At what level the implementation of the use case be.
Flow: What will be the flow of the functionality that needs to be there. More precisely, the work flow of the use case.
Some other things that can be included in the use cases are:

- **Preconditions**
- **Postconditions**
- **Brief course of action**
- **Time Period**


Use Case Diagram
«Subsystem»
Urban-Garden
View Items
«include»
Make Purchase
«include»
Checkout
Client Register
Registered User
User
New User
«Service» Authentication
Identity Provider
Credit Payment Service
PayPal
User Sign-In
Customer Authentication
Remember Me
Single Sign-On
«include»
Checkout
«include»
View / Update Shopping Cart
«include»
Calculate Total
«include»
Calculate Taxes and Shipping
«include»
Payment
Payment By Credit
Payment By PayPal
Search Items
«extend»
Browse Items
«extend»
View Items
«extend»
View Recommended Items
«include»
«extend»
Add to Shopping Cart
«extend»
Customer Authentication
«extend»
Add to Wish List
«include»

EXPERIMENT. NO: 07

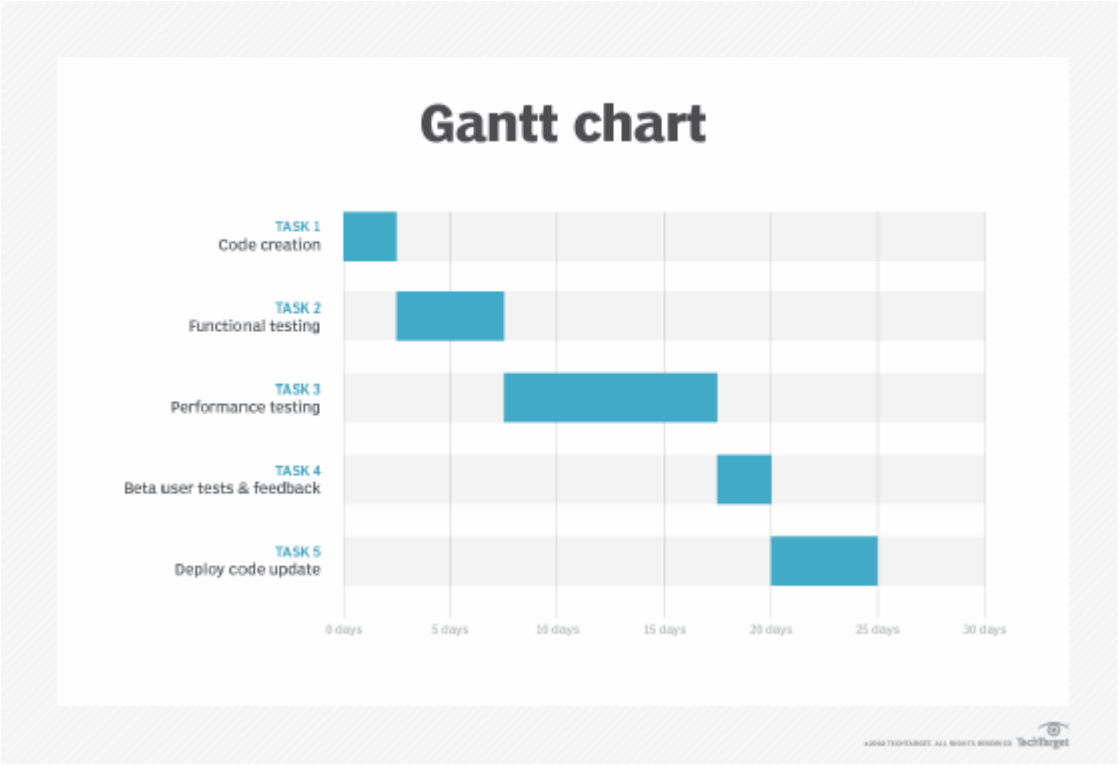
Aim: Create a project schedule using Gantt Chart

Theory:

What is a Gantt chart?

A Gantt chart is a horizontal bar chart developed as a production control tool in 1917 by Henry L. Gantt, an American engineer and social scientist. Frequently used in project management, a Gantt chart provides a graphical illustration of a schedule that can be used to plan, coordinate and track tasks in a project.

Gantt charts can be simple versions created on graph paper or more complex automated versions created using project management applications such as Microsoft Project or Excel.



History of the Gantt chart

Karol Adamiecki, a Polish mechanical engineer and professor, is credited with devising what would become the first Gantt chart in 1896. He developed the law of harmony in management. It emphasizes the importance of good teams, scheduling and using compatible production tools.

Adamiecki created the harmonogram, a precursor of the Gantt chart. It was not published until 1931 and was only published in Polish.

In 1917, American engineer Henry L. Gantt used Adamiecki's harmonogram as inspiration for his own chart. The Gantt chart became popular as a way to describe resource loading and production planning in factories. It was also useful for gauging employee productivity.

Early Gantt charts were made of physical materials like paper and blocks. Some of the earliest ones were used in World War I at munitions manufacturing plants to help manage the unprecedented production requirements to support war efforts.

Until the 1980s, Gantt charts were limited in complexity and were hand-drawn. Computers enabled more complex Gantt charts. They also made the charts easier to change and suitable for more applications. Today, Gantt charts are widely used as part of project management software.

What are Gantt charts used for?

Gantt charts are one of the many project management tools. They present in one chart all the tasks in a project. They show the order in which they tasks should be done and the time needed to complete them. This is valuable information to have when managing projects for the following reasons:

- Progress monitoring. Project managers can see if individual tasks are completed on time and adjust the project schedule. The charts also show which goals were met on schedule, helping managers gauge employee productivity.
- Project planning. Managers can set deadlines, milestones and schedules for various project components.
- Resource management. Project planners can coordinate resource allocation with the project schedule. Managers can see the amount of time each process takes and designate resources accordingly.

Both the Waterfall and Agile project management methodologies make use of Gantt charts. Because they display project information linearly, they work particularly well with Waterfall, where customer expectations are collected at the beginning of a project, and a linear plan is devised to meet them.

Project teams using the Agile approach set their own goals and use continuous customer feedback to update their plan in real time. Gantt charts can be useful in Agile to compare an old plan to a proposed change and to see what effect the change has on the overall plan.

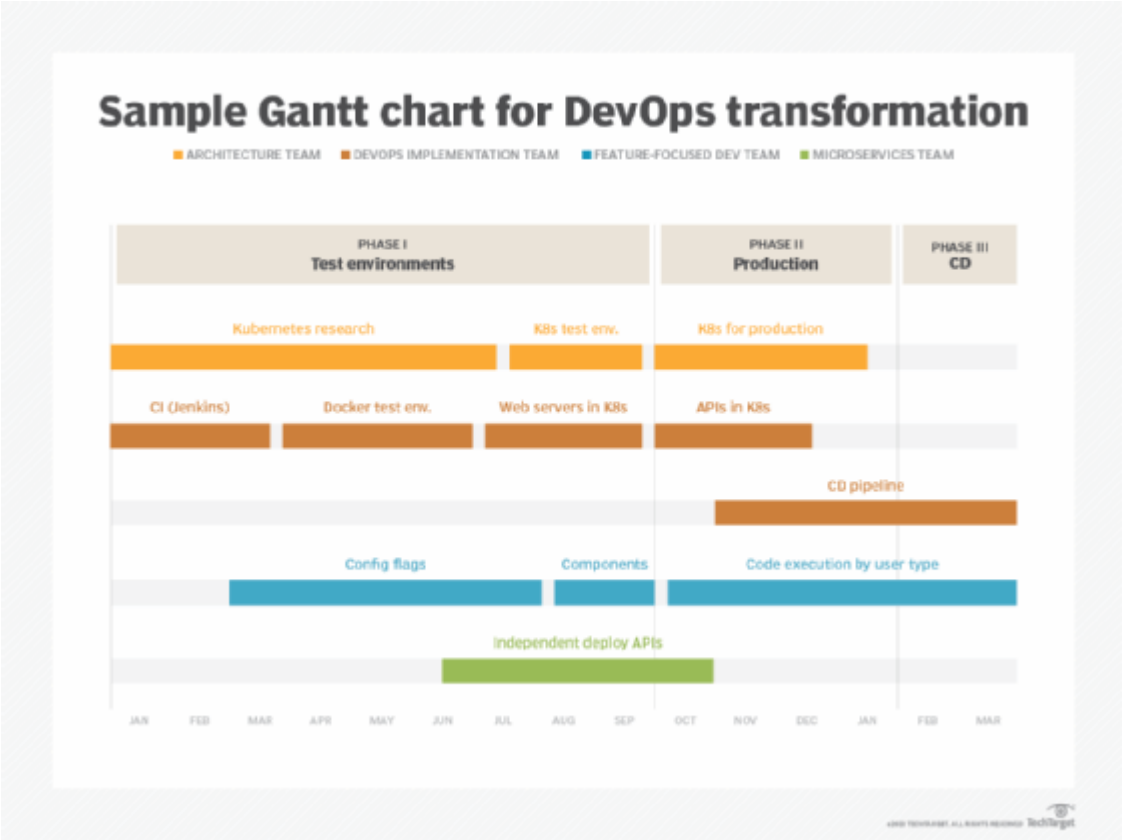
How to build a Gantt chart

A Gantt chart is constructed with a horizontal axis representing the total time span of the project, broken down into increments -- days, weeks or months. It has a vertical axis representing the project tasks. For example, if the project is choosing new HR software, major tasks might be: conduct research, choose software and install software.

Horizontal bars of varying lengths represent the sequences, timing, and time span for each task. Using the same example, put "conduct research" at the top of the vertical axis and draw a bar on the graph that represents the amount of time anticipated for the research, then enter the other tasks below the first one with representative bars at the points in time when they'll be undertaken.

The bar spans may overlap. For example, conducting research and choosing software may happen during the same time span. As the project progresses, secondary bars, arrowheads or darkened bars may be added to indicate completed tasks, or the portions of tasks completed. A vertical line is used to represent the report date.

Gantt charts can also be created using software like Microsoft Excel. With some programming knowledge, JavaScript can be used. It has several chart libraries that work well for data visualization.



What are the benefits of using a Gantt chart?

Some benefits of using a Gantt chart include the following:

- **Efficiency.** Gantt charts help project managers calculate realistic project completion times and set goals based on available resources. Both of these advantages increase productivity.
- **Teamwork**. Team members have access to the same information, keeping everyone informed of a project's progress. Because of this, it's possible to hold all team members, including remote workers, accountable for their tasks. Team members can reference the chart and establish roles and responsibilities.
- **Tracking**. Gantt charts allow project planners and team members to analyze workflows for constraints and adjust their work accordingly. Gantt charts help project managers track benchmarks and tasks throughout the project process. Team members can easily visualize which elements may be missing from the Gantt chart.
- **Versatility**. Gantt charts help teams balance multiple projects at once. They prompt managers to estimate which resources are needed when. They are also easy to change.
- **Visualization**. Gantt charts provide a project roadmap at a glance, allowing for easier management, monitoring and organization of project components. The chart gives a holistic view of the project timeline and tasks. It also provides high-level visibility into

the who, what, when and where of a project. Employees can quickly see the status of the project or a project phase.

Gantt chart limitations

Gantt charts are useful project management tools, but they have flaws, such as the following:

- **Lack of dependencies**. Gantt charts don't indicate task dependencies or critical paths of projects in detail. Users cannot tell how one task falling behind schedule affects other tasks. The PERT chart, another popular project management method, is designed to do this.
- **Lack of specificity**. Gantt charts are good for providing a high-level view of project workflows. However, they can leave some things to interpretation. This is especially true for large-scale IT projects, such as DevOps Certain details about priorities, deadlines and expectations may be left out of the high-level view of complex projects.
- **Quantity of work**. A third issue with Gantt charts is that the task bar doesn't show the quantity of work each task requires. It may show a specific length of time, which roughly implies a certain amount of resources. However, it does not designate the resources. That creates ambiguity and can lead to resource contention.

Gantt chart software

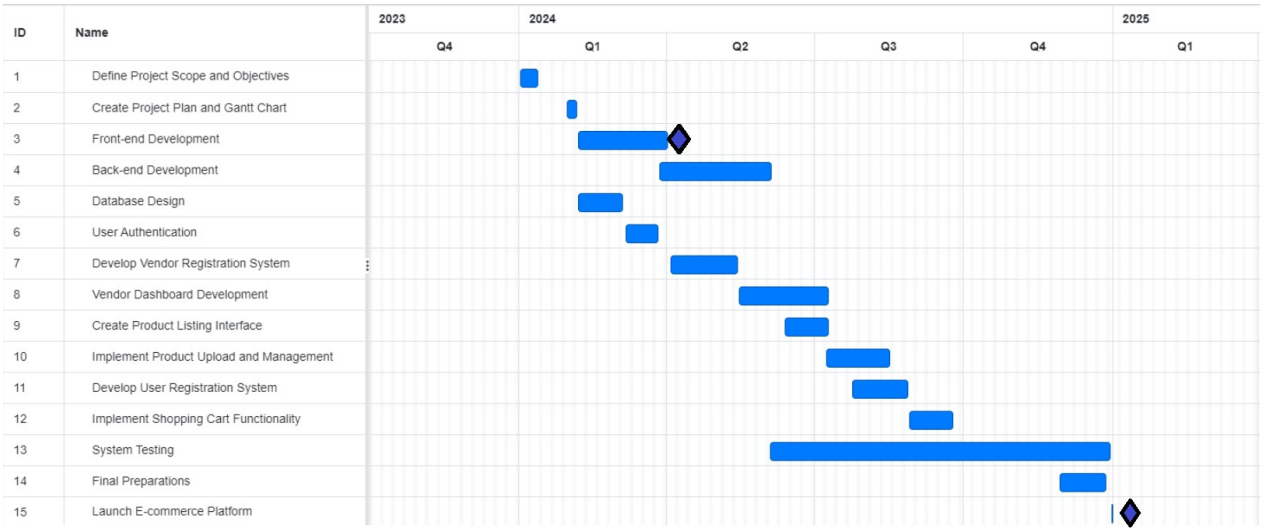
A variety of project management tools can function as Gantt chart software. Examples include:

- Asana
- Atlassian Jira
- Citrix Systems Wrike
- Gantto
- Microsoft Excel
- Microsoft Project
- Microsoft SharePoint
- Microsoft Visio
- Matchware
- Smartsheet
- Workzone

Not all of these tools are exclusively for creating Gantt charts. Some of them support other functions and contain features that let users create Gantt charts. For example, Jira is a

project management and application lifecycle management tool that contains a Gantt chart feature. On the other hand, Gantto is largely dedicated to creating and sharing Gantt charts

Diagram:



EXPERIMENT NO. 8

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

Theory:

Function Point Analysis was initially developed by Allan J. Albrecht in 1979 at IBM and has been further modified by the International Function Point Users Group (IFPUG). The initial definition is given by Allan J. Albrecht.

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.

Characteristics of Functional Point Analysis:

1. **External Input (EI):** EI processes data or control information that comes from outside the application's boundary. The EI is an elementary process.
2. **External Output (EO):** EO is an elementary process that generates data or controls information sent outside the application's boundary.
3. **External Inquiries (EQ):** EQ is an elementary process of input-output combination that results in data retrieval.
4. **Internal Logical File (ILF):** A user-identifiable group of logically related data or control information maintained within the boundary of the application.
5. **External Interface File (EIF):** A group of users recognisable logically related data allusion to the software but maintained within the boundary of another software.

Calculating the Count based on all three factors

Weighing Factor is assumed to be **Simple**

Information Domain Value	Count		Weighing Factor				Total
			Simple	Average	Complex		
No. of External Inputs (EIs)	**4**	*	**3**	4	6	=	***12***
No. of External Outputs (EOs)	**7**	*	**4**	5	7	=	***28***
No. of External Inquiries (EQs)	**2**	*	**3**	4	6	=	***6***
No. of Files (IFs)	**2**	*	**7**	10	15	=	***14***
No. of External Interfaces (EIFs)	**2**	*	**5**	7	10	=	***10***
					Count	=	***70***

Weighing Factor is assumed to be **Average**

Information Domain Value	Count		Weighing Factor				Total
			Simple	Average	Complex		
No. of External Inputs (EIs)	**4**	*	3	**4**	6	=	***16***
No. of External Outputs (EOs)	**7**	*	4	**5**	7	=	***35***
No. of External Inquiries (EQs)	**2**	*	3	**4**	6	=	***8***
No. of Files (IFs)	**2**	*	7	**10**	15	=	***20***
No. of External Interfaces (EIFs)	**2**	*	5	**7**	10	=	***14***
					Count	=	***93***

Weighing Factor is assumed to be **Complex**

Information Domain Value	Count		Weighing Factor				Total
			Simple	Average	Complex		
No. of External Inputs (EIs)	**4**	*	3	4	**6**	=	***24***
No. of External Outputs (EOs)	**7**	*	4	5	**7**	=	***49***
No. of External Inquiries (EQs)	**2**	*	3	4	**6**	=	***12***
No. of Files (IFs)	**2**	*	7	10	**15**	=	***30***
No. of External Interfaces (EIFs)	**2**	*	5	7	**10**	=	***20***
					Count	=	***135***

Convention for measuring Adjustment factors

0 – No Influence

1 – Incidental

2 – Moderate

3 – Average

4 – Significant

5 – Essential

14 Adjustment Factors:

1. ***Backup & Recovery = 5***

   Definition: Operational Ease measures the complexity of operating and managing the software. It encompasses factors like backup, recovery, and system monitoring.

   Justification: In UrbanGarden, the need for robust backup and recovery systems is critical.UrbanGarden stores sensitive customer data, order history, and product information. A high rating in this factor reflects the importance of ensuring data integrity and availability, especially in the event of system failures or data breaches.

2. ***Data Communications = 4***

   Definition: It measures the complexity of data exchange between the application and external entities.

   Justification: UrbanGarden often requires extensive data exchange with external entities, including payment gateways, shipping providers, and third-party integrations. This complexity in data communications arises from the need to securely transmit and receive data between various systems, which can affect the overall complexity of the project.

3. ***Distributed Data Processing = 4***

   Definition: It evaluates the complexity resulting from processing data across multiple locations.

   Justification: UrbanGarden involves distributed data processing to handle user requests, inventory management, and real-time order updates. Distributing these processes across multiple locations or servers can introduce complexities related to data consistency, latency, and synchronisation.

4. ***Performance Critical = 4***

   Definition: It assesses the performance requirements and constraints of the system.

   Justification: In any e-commerce platform, performance is crucial. Customers expect fast page load times and quick responses during the shopping and checkout process. Ensuring the application meets these performance criteria can require additional development effort and complexity, such as optimising code and database queries.

5. ***Existing Operating Environment = 3***

   Definition: The "Existing Operating Environment" in software development refers to the pre-existing technological infrastructure, including hardware, software systems, and network components, that serves as the foundation for a new software application.

   Justification: Applications like UrbanGarden are often built upon existing operating environments, including web servers, databases, and hosting infrastructure. While this can provide a foundation for development, it may also introduce some constraints or dependencies that need to be considered, impacting the complexity to a moderate extent.

6. ***Online Data Entry = 3***

   Definition: It evaluates the complexity of user interfaces for data entry and updates.

   Justification: Online data entry refers to user interfaces for entering data, such as customer registration forms and address input during checkout. In any e-commerce app, these forms need to be user-friendly, efficient, and capable of handling various data inputs. While they are important, their complexity is moderate compared to other aspects of the application.

7. ***Input transaction over multiple screens = 4***

   Definition: It measures the number of business transactions or operations the system must support.

   Justification: In UrbanGarden, complex transactions often involve multiple steps and screens. For example, the checkout process may include steps for selecting products, entering shipping details, choosing payment methods, and confirming orders. Handling these multi-screen transactions requires careful design and implementation, contributing to the complexity.

8. ***Master Files updated online = 3***

   Definition: It measures the complexity of real-time data updates and maintenance.

   Justification: Updating master files online relates to managing product catalogues, inventory levels, and other critical information in real-time. While important, this complexity is moderate as it involves data management and synchronisation but doesn't typically require highly complex processing.

9. ***Information domain values Complex = 2***

   Definition: "Information Domain Values (IDV) Complex" in Function Point Analysis (FPA) refers to the degree of complexity associated with intricate calculations, transformations, or business rules that the software performs on data or information within the application.

   Justification: Information domain values (IDV) complexity may be relatively lower in e-commerce applications compared to other domains. IDV complexity typically pertains to intricate calculations or data processing, which are less prominent in the core functionality of e-commerce apps.

10. ***Internal Processing Complex = 3***

    Definition: It evaluates the complexity of business logic and algorithms.

    Justification: Internal processing complexity in any e-commerce application can arise from complex business rules, pricing calculations, tax calculations, and inventory management algorithms. These complexities, while essential, are generally of moderate magnitude.

11. ***Code Designed for Reuse = 4***

    Definition: It measures the extent to which existing software components can be reused.

    Justification: Designing code for reuse in UrbanGarden can reduce development time and effort. Reusable components can include payment integrations, user authentication modules, and shopping cart functionality, contributing to the overall project complexity.

12. ***Conversion/Installation in design = 5***

    Definition: It assesses the complexity of installing the software in various environments.

    Justification: In UrbanGarden, migrating or setting up a new platform often requires data conversion, especially if you are transitioning from an old system. The complexity of handling data migration and system installation can be significant, hence the higher rating.

13. ***Multiple Installations = 4***

    Definition: It considers the complexity introduced when the system operates in multiple geographical locations.

Justification: E-commerce platforms may require installations across multiple geographical locations or for various clients. Managing and configuring these installations while ensuring consistency can add complexity to the project.

14. *Application designed for change* = 3

Definition: It measures the ease with which the software can be modified or adapted.

Justification: The E-commerce sector is a dynamic field with changing customer preferences and industry trends. Designing the application to accommodate future changes or enhancements is important but generally of moderate complexity. It involves maintaining a balance between flexibility and stability.

Calculating Functional Point Analysis for each of the three weighing factors

1. **Function Point (FP) for Simple:**

   (FP) = Total Count * [0.65 + 0.01 * ΣFi]
   = 70 * [0.65 + 0.01 * 51]
   = 81.2

2. **Function Point (FP) for Average:**

   (FP) = Total Count * [0.65 + 0.01 * ΣFi]
   = 93 * [0.65 + 0.01 * 51]
   = 107.88

3. **Function Point (FP) for Complex:**

   (FP) = Total Count * [0.65 + 0.01 * ΣFi]
   = 135 * [0.65 + 0.01 * 51]
   = 156.6

Calculate the FOUR parameters

(Effort, Productivity, Cost per Function Point, and Cost)

Let's calculate the four parameters (Effort, Productivity, Cost per Function Point, and Cost) based on the given formulae and the provided values:

1. **Effort**:
   - Estimated Cost = Let's assume the estimated cost is 10,000/-.
   - Labour Rate = Let's assume the labour rate is 50/- per hour.

   Effort = Estimated Cost / Labour Rate
   Effort = 10,000 / 50
   Effort = 200 hours

2. **Productivity**:
   - Function Points (FP) for each type (Simple, Average, Complex) are already calculated.
   Productivity = FP / Effort

   For Simple:
   Productivity (Simple) = 81.2 / 200
   Productivity (Simple) = 0.406 FP/hour

   For Average:
   Productivity (Average) = 107.88 / 200
   Productivity (Average) = 0.5394 FP/hour

   For Complex:
   Productivity (Complex) = 156.6 / 200
   Productivity (Complex) = 0.783 FP/hour

3. **Cost per Function Point**:
   - Cost per Function Point (for each type) can be calculated *as the reciprocal of Productivity.*

   Cost per Function Point (Simple) = 1 / Productivity (Simple)
   Cost per Function Point (Simple) = 1 / 0.406
   Cost per Function Point (Simple) ≈ 2.46/FP

   Cost per Function Point (Average) = 1 / Productivity (Average)
   Cost per Function Point (Average) = 1 / 0.5394
   Cost per Function Point (Average) ≈ 1.85/FP

   Cost per Function Point (Complex) = 1 / Productivity (Complex)

Cost per Function Point (Complex) = 1 / 0.783
Cost per Function Point (Complex) ≈ 1.28/FP

4. **Cost**:
- Function Points (FP) for each type are already calculated.

*Cost = FP * Cost per Function Point*

For Simple:
Cost (Simple) = 81.2 * 2.46/FP
Cost (Simple) ≈ 199.57

For Average:
Cost (Average) = 107.88 * 1.85/FP
Cost (Average) ≈ 199.69

For Complex:
Cost (Complex) = 156.6 * $1.28/FP
Cost (Complex) ≈ 200.45

So, based on the provided values and calculations:

1. **Effort**:
   - Simple: 200 hours
   - Average: 200 hours
   - Complex: 200 hours

2. **Productivity**:
   - Simple: 0.406 FP/hour
   - Average: 0.5394 FP/hour
   - Complex: 0.783 FP/hour

3. **Cost per Function Point**:
   - Simple: 2.46/FP
   - Average: 1.85/FP
   - Complex: 1.28/FP

4. **Cost**:
   - Simple: 199.57/-
   - Average: 199.69/-
   - Complex: 200.45/-

EXPERIMENT NO: 09

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

Theory:

Cocomo (Constructive Cost Model) is a regression model based on LOC, i.e. number of Lines of Code. It is a procedural cost estimate model for software projects and is often used as a process of reliably predicting the various parameters associated with making a project such as size, effort, cost, time, and quality.
The key parameters which define the quality of any software products, which are also an outcome of the Cocomo are primarily Effort and schedule:

- Effort: Amount of labour 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 1:

Software development projects can be classified into the following categories based on the development complexity:

1. **Organic**: A software project is said to be organic if the team size required is adequately small, the problem is well understood and has been solved in the past, and the team members have a nominal experience regarding the problem.

2. **Semi-Detached**: A software project is said to be a Semi-detached type if the vital characteristics such as team size, experience, and knowledge of the various programming environments lie in between organic and Embedded. The projects classified as Semi-Detached are comparatively less familiar and challenging to develop than organic ones and require more experience better guidance and creativity. Eg: Compilers or different Embedded Systems can be considered Semi-Detached types.

3. **Embedded**: A software project requiring the highest complexity, creativity, and experience requirement falls under this category. Such software requires a larger team size than the other two models and also the developers need to be sufficiently experienced and creative to develop such complex models.

Estimation of Effort:

Organic: Effort = $2.4(KLOC)^{1.05}$ PM

Semi-detached: Effort = $3.0(KLOC)^{1.12}$ PM

Embedded: Effort = $3.6(KLOC)^{1.20}$ PM

Estimation of Development Time:

Organic: Tdev= $2.5(Effort)^{0.38}$ Months

Semi-detached: Tdev= $2.5(Effort)^{0.35}$ Months

Embedded: Tdev= $2.5(Effort)^{0.32}$ Months

Question:

We have determined our project fits the characteristics of Basic, Semi-Detached mode. We estimate our project will have 83,000 Delivered Source Instructions.

Find the Effort, Schedule, Productivity, and average staffing required for the project.

Answer:

BASIC MODE:

Using the formulas, we can estimate:

1. **Organic -**

   Effort = $2.4 * (KLOC)^{1.05}$ PM
   = $2.4 * (83000)^{1.05}$ PM
   = 350948 PM

   Schedule = $2.5 * (Effort)^{0.38}$ Months
   = $2.5 * (350948)^{0.38}$ Months
   = 320 Months

   Productivity = DSI / Effort
   = 83000 DSI / 350948 PM
   = 0.24 DSI/PM

Average. Staffing = Effort / Schedule
= 350948 / 320 FSP
= 1097 FSP

2. Semi-Detached -

Effort = 3.0 * $(KLOC)^{1.12}$ PM
= 3.0 * $(83000)^{1.12}$ PM
= 969368 PM

Schedule = 2.5 * $(Effort)^{0.35}$ Months
= 2.5 * $(969368)^{0.35}$ Months
= 311 Months

Productivity = DSI / Effort
= 83000 DSI / 969368 PM
= 0.086 DSI/PM

Average. Staffing = Effort / Schedule
= 969368 / 311 FSP
= 3117 FSP

3. Embedded -

Effort = 3.6 * $(KLOC)^{1.20}$ PM
= 3.6 * $(83000)^{1.20}$ PM
= 2878699 PM

Schedule = 2.5 * $(Effort)^{0.32}$ Months
= 2.5 * $(2878699)^{0.32}$ Months
= 292 Months

Productivity = DSI / Effort
= 83000 DSI / 2878699 PM
= 0.029 DSI/PM

Average. Staffing = Effort / Schedule
= 2878699 / 292 FSP
= 9858 FSP

INTERMEDIATE MODE:

COST DRIVER	DESCRIPTION
RELY	Required software reliability
DATA	Database size
CPLX	Product complexity
TIME	Execution time constraints
STOR	Main storage constraints
VIRT	Virtual machine volatility - degree to which the operating system changes
TURN	Computer turn around time
ACAP	Analyst capability
AEXP	Application experience
PCAP	Programmer capability
VEXP	Virtual machine (i.e. operating system) experience
LEXP	Programming language experience
MODP	Use of modern programming practices
TOOL	Use of software tools
SCED	Required development schedule

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	.

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

Question:

Project A is to be an **83,000 DSI organic, semi-detached, embedded** software. It is in a mission-critical area, so the **reliability** is high (RELY=high=1.15).

Find the Effort, Schedule, Productivity, and average staffing required for the project.

Answer:

Using the formulas, we can estimate:

1. **Organic -**

Effort = $2.4 * (KLOC)^{1.05} * 1.15$ PM
= $2.4 * (83000)^{1.05}$ PM
= 403590 PM

Schedule = $2.5 * (Effort)^{0.38}$ Months
= $2.5 * (403590)^{0.38}$ Months
= 337 Months

Productivity = DSI / Effort
= 83000 DSI / 403590 PM
= 0.21 DSI/PM

Average. Staffing = Effort / Schedule
= 403590 / 337 FSP
= 1198 FSP

2. **Semi-Detached -**

Effort = $3.0*(KLOC)^{1.12} * 1.15$ PM
= $3.0*(83000)^{1.12}$ PM
= 1114773 PM

Schedule = $2.5*(Effort)^{0.35}$ Months
= $2.5*(1114773)^{0.35}$ Months

= 327 Months

Productivity = DSI / Effort
= 83000 DSI / 1114773 PM
= 0.075 DSI/PM

Average. Staffing = Effort / Schedule
= 1114773 / 327 FSP
= 3409 FSP

3. **Embedded -**

Effort = $3.6*(KLOC)^{1.20}$ * 1.15 PM
= $3.6*(83000)^{1.20}$ PM
= 3310504 PM

Schedule = $2.5*(Effort)^{0.32}$ Months
= $2.5*(3310504)^{0.32}$ Months
= 305 Months

Productivity = DSI / Effort
= 83000 DSI / 3310504 PM
= 0.025 DSI/PM

Average. Staffing = Effort / Schedule
= 3310504 / 305 FSP
= 10854 FSP

Question:

As an example of how the intermediate COCOMO model works, the following is a calculation of the estimated effort for an organic, semi-detached, embedded project of 83 KLOC. The cost drivers are set as follows:

Product cost drivers- **very high**

Computer cost drivers – **nominal**

Personnel cost drivers - **high**

Project cost drivers - **low**

Solution:

Product cost drivers (from the table) set **Very High** = 1.40 x 1.16 x 1.30

= 2.11

Computer cost drivers (from the table) set **nominal** = 1.00

Personnel cost drivers (from the table) set **high** = 0.86 x 0.91 x 0.86 x 0.95 x 0.90

= 0.57

Project cost drivers (from the table) set **low** = 1.10 x 1.10 x 1.08

= 1.30

product(cost drivers) = 2.11 x 1.00 x 0.57 x 1.30

= 1.56

For an **organic** project of 83 KLOC: **a** = 3.2 **b** = 1.05 **S** = 83

E = a(S)b x product(cost drivers)

E = 3.2 x (83)^1.05 x 1.56

E = 156.78 person-months

For a **semi-detached** project of 83 KLOC: **a** = 3.0 **b** = 1.12 **S** = 56

E = a(S)b x product(cost drivers)

E = 3.0 x (83)^1.12 x 1.56

E = 660.11 person-months

For a **semi-detached** project of 83 KLOC: **a** = 2.8 **b** = 1.20 **S** = 56

E = a(S)b x product(cost drivers)

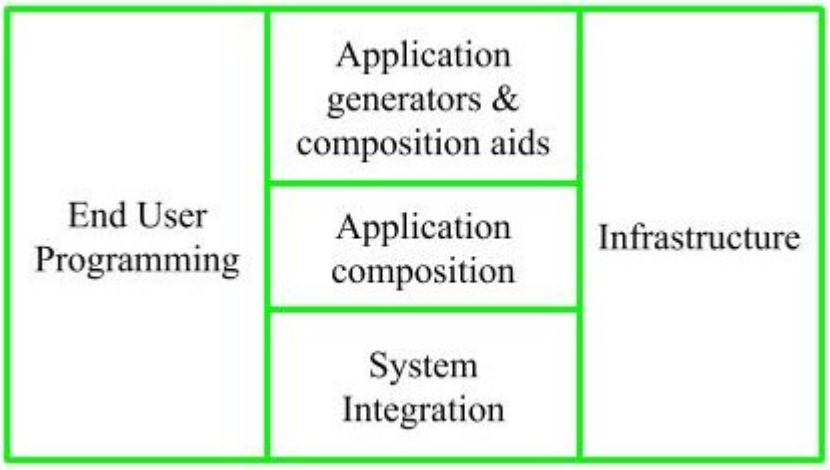
E = 2.8 x (83)^1.20 x 1.56

E = 877.36 person-months

COCOMO 2:

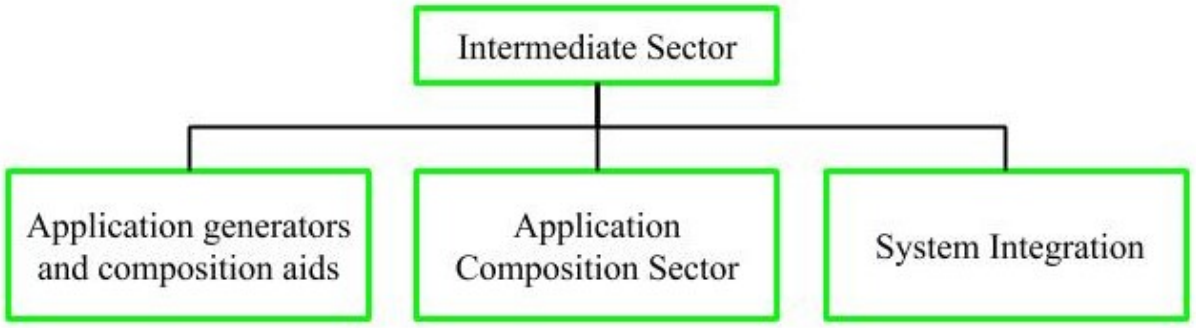
COCOMO-II is the revised version of the original Cocomo (Constructive Cost Model) and was developed at the University of Southern California. It is the model that allows one to estimate the cost, effort and schedule when planning a new software development activity.

It consists of three sub-models:



1. End-User Programming:
Application generators are used in this sub-model. End users write the code by using these application generators.
Examples are spreadsheets, report generators, etc.

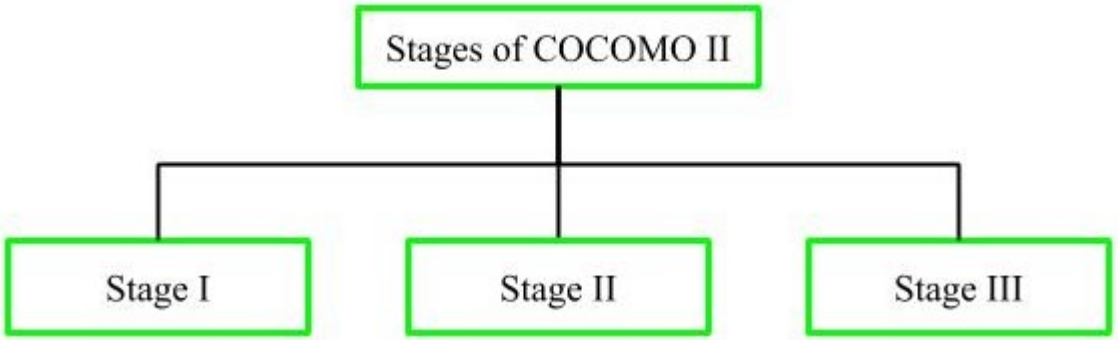
2. Intermediate Sector:



(a). Application Generators and Composition Aids –
This category will create largely prepackaged capabilities for user programming. Their product will have many reusable components. Typical firms operating in this sector are Microsoft, Lotus,
Oracle, IBM, Borland, Novell.

(b). Application Composition Sector –
This category is too diversified and to be handled by prepackaged solutions. It includes GUI, Databases, and domain-specific components such as financial, medical or industrial process control packages.
(c). System Integration –
This category deals with large-scale and highly embedded systems.

3. Infrastructure Sector:



This category provides infrastructure for software development like Operating Systems, Database Management Systems, User Interface Management Systems, Networking Systems, etc.

Stages of COCOMO II:

Stage-I:
It supports the estimation of prototyping. For this, it uses the Application Composition Estimation Model. This model is used for the prototyping stage of application generator and system integration.
Stage-II:
It supports estimation in the early design stage of the project when we less know about it. For this, it uses the Early Design Estimation Model. This model is used in the early design stage of application generators, infrastructure, and system integration.
Stage-III:
It supports estimation in the post-architecture stage of a project. For this, it uses the Post Architecture Estimation Model. This model is used after the completion of the detailed architecture of the application generator, infrastructure, and system integration.

Certainly,

QUESTION: let's adapt the question for an e-commerce website project. In this scenario, we'll consider the development of an e-commerce website with 2 main pages, each containing 4 different views, and 10 data tables for managing products, users, orders, and other related information. The website is designed to handle traffic from 1 server and 5 clients. Additionally, the project includes the generation of 3 different reports, each consisting of 7 sections, all of which are derived from the data stored in the 10 data tables. We'll also assume a 20% reuse of object points, and that the developer's experience and capability in a similar environment is Nominal.

Step 1: Determine the number of screens and reports.

- Number of screens = 2 main pages * 4 different views = 8 screens
- Number of reports = 3 reports

Step 2: Determine the factors for screens and reports.

For Screens:

- Number of views per screen = 4
- Number of data tables per screen = 10
- Number of servers = 1
- Number of clients = 5
- Complexity level for each screen = Nominal (since developer experience is considered Nominal)

For Reports:

- Number of sections per report = 7
- Number of data tables per report = 10
- Number of servers = 1
- Number of clients = 5
- Complexity level for each report = Nominal

Step 3: Assign complexity weights.

Now, you can assign complexity weights based on the provided information. Complexity weights are typically assigned on a scale from Low to High. In this case, since you've mentioned that the developer's experience is Nominal, we can assign weights as follows:

- Complexity weight for each screen = 5 (Nominal)
- Complexity weight for each report = 5 (Nominal)

Step 4: Calculate Object Point Count

First, let's calculate the Object Point Count for your e-commerce website project using the complexity weights determined earlier:

For Screens:

Object Point Count for Screens = (Number of screens * Complexity weight for each screen)

Object Point Count for Screens = (8 screens * 5) = 40 Object Points

For Reports:

Object Point Count for Reports = (Number of reports * Complexity weight for each report)

Object Point Count for Reports = (3 reports * 5) = 15 Object Points

Now, we can calculate the total Object Point Count:

Total Object Point Count = Object Point Count for Screens + Object Point Count for Reports

Total Object Point Count = 40 + 15 = 55 Object Points

Given that there is a 20% reuse of object points, we can calculate the Non-Reused Object Points (NOP):

Non-Reused Object Points (NOP) = [Total Object Points * (100 - % Reuse)] / 100

Non-Reused Object Points (NOP) = [55 * (100 - 20)] / 100

Non-Reused Object Points (NOP) = [55 * 80] / 100

Non-Reused Object Points (NOP) = 44

Step 6: Developer’s Experience and Capability

The developer’s experience and capability are Nominal

Using the information given about the developer and productivity rate table

Productivity rate (PROD) of given project = 13

Step 7: Effort Calculation

Effort = NOP / PROD

Effort = 44 / 13

Effort ≈ 3.38 person-days (rounded to two decimal places)

Experiment No. 10

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

Objective:

To be able to:

- Identify Risks
- Analyse the risks and prepare the RMMM plan

Resources Needed:

MS-office / Open office

Theory:

- Planning the Risk Management - The proactive strategy for risk estimation is used which helps in identifying the possible threats that can occur during the project well in advance. Accordingly, steps to avoid, monitor and manage the risk are to be carried out and noted down in the form of RMMM plan.
- THE RMMM PLAN - A risk management strategy can be included in the software project plan or the risk management steps can be organized into a separate Risk Mitigation, Monitoring and Management Plan. The RMMM plan documents all work performed as part of risk analysis and is used by the project manager as part of the overall project plan.
- Some software teams do not develop a formal RMMM document. Rather, each risk is documented individually using a risk information sheet (RIS). In most cases, the RIS is maintained using a database system, so that creation and information entry, priority ordering, searches, and other analysis may be accomplished easily.
- Once RMMM has been documented and the project has begun, risk mitigation and monitoring steps commence. Risk mitigation is a problem avoidance activity. Risk monitoring is a project tracking activity with three primary objectives:

  1) to assess whether predicted risks do, in fact, occur

  2) to ensure that risk aversion steps defined for the risk are being properly applied

  3) to collect information that can be used for future risk analysis.

- In many cases, the problems that occur during a project can be traced to more than one risk. Another job of risk monitoring is to attempt to allocate the origin (which risk(s) caused which problems throughout the project)

- Risk management is the identification, assessment, and prioritization of risks (defined in ISO 31000 as the effect of uncertainty on objectives, whether positive or negative) followed by coordinated and economical application of resources to minimize, monitor, and control the probability and/or impact of unfortunate events or to maximize the realization of opportunities.
- In ideal risk management, a prioritization process is followed whereby the risks with the greatest loss (or impact) and the greatest probability of occurring are handled first, and risks with lower probability of occurrence and lower loss are handled in descending order.
- In practice the process of assessing overall risk can be difficult, and balancing resources used to mitigate between risks with a high probability of occurrence but lower loss versus a risk with high loss but lower probability of occurrence can often be mishandled.
- Risk Identification and Management Process comprises of following steps:

  1) Risk Identification

  2) Risk Analysis

  3) Risk Assessment

  4) RMMM (Risk Mitigation, Monitoring, Management) plan

<table>
<tr><th colspan="4">Risk Information Sheet</th></tr>
<tr><td>Risk ID: R001</td><td>Date: 05/09/23</td><td>Prob: Medium</td><td>Impact: High</td></tr>
<tr><td colspan="4">Description:
Payment Processing Failure - There is a risk that the payment processing system on the Urban Garden platform may experience failures, causing disruptions in transactions and customer dissatisfaction.</td></tr>
<tr><td colspan="4">Mitigation:
● Implement redundant payment gateways to ensure backup options are available.
● Regularly update and test payment processing software for reliability.
● Establish a dedicated technical team responsible for resolving payment issues promptly.</td></tr>
<tr><td colspan="4">Monitoring:
● Monitor transaction success rates in real-time and set thresholds for acceptable failure rates.
● Implement automated alerts to notify the technical team immediately upon detecting payment processing issues.
● Regularly review payment processing logs to identify any recurring patterns of failure.</td></tr>
<tr><td colspan="4">Management:
1. Activate the contingency plan.
2. Notify the technical team to diagnose and resolve the issue promptly.
3. Notify affected customers of the issue and assure them of resolution.
4. Use backup payment gateways if necessary to minimise disruption.
5. Continuously update affected parties on the progress of issue resolution.</td></tr>
<tr><td colspan="4">Current Status:
This risk has been identified and measures are in place for both mitigation and monitoring.</td></tr>
<tr><td colspan="2">Originator: Om Shete</td><td colspan="2">Assigned: Om Shete, Mohib Abbas Sayed, Hamza Sayyed</td></tr>
</table>

Risk Information Sheet			
Risk ID: R003	Date: 05/09/23	Prob: Medium	Impact: High

Description:
Data Security Breach - There is a risk of a data security breach where unauthorised individuals could gain access to sensitive customer data stored on the Urban Garden platform, including personal and payment information.

Mitigation:
- Implement robust encryption for all customer data, both at rest and during transmission.
- Enforce strict access controls, limiting data access to authorised personnel only.
- Conduct regular security audits and vulnerability assessments.

Monitoring:
- Utilise intrusion detection systems to monitor network traffic for suspicious activity.
- Implement log analysis tools to review access logs for unusual or unauthorised behaviour.
- Regularly update and patch software to address security vulnerabilities.

Management:
1. Activate the incident response plan immediately.
2. Notify affected customers and regulatory authorities as required by data protection laws.
3. Conduct a forensic analysis to determine the extent of the breach and identify vulnerabilities.
4. Enhance security measures to prevent future breaches, including updating security protocols.
5. Collaborate with legal and public relations teams to manage the crisis and communicate with affected parties.

Current Status:
This risk has been identified, and comprehensive measures for both mitigation and monitoring are in place.

Originator: Om Shete	Assigned: Om Shete, Mohib Abbas Sayed, Hamza Sayyed

Risk Information Sheet			
Risk ID: R004	Date: 05/09/23	Prob: Medium	Impact: Medium

Description:
Platform Performance Degradation - There is a risk that the Urban Garden platform may experience performance issues, resulting in slow response times and reduced user satisfaction.

Mitigation:
- Regularly conduct performance testing to identify bottlenecks and areas of improvement.
- Invest in scalable infrastructure to accommodate increasing user loads.
- Optimise code and database queries for efficiency.

Monitoring:
- Implement performance monitoring tools to track response times, server load, and database performance.
- Set performance thresholds and automate alerts for immediate issue identification.
- Monitor user feedback and complaints related to platform performance.

Management:
1. Activate the contingency plan.
2. Notify the technical team to investigate and address the performance issues.
3. Communicate with users, acknowledging the issue and providing estimated resolution times.
4. Implement temporary measures such as caching or load balancing to mitigate performance issues.
5. Continuously update affected parties on the progress of issue resolution.

Current Status:
This risk has been identified, and measures for both mitigation and monitoring are in place.

Originator: Om Shete	Assigned: Om Shete, Mohib Abbas Sayed, Hamza Sayyed

Risk Information Sheet			
Risk ID: R005	Date: 05/09/23	Prob: Medium	Impact: High

Description:
Vendor Reliability - There is a risk that a key vendor or supplier, responsible for providing essential services or products to the Urban Garden platform, may become unreliable or face operational issues.

Mitigation:
- Diversify vendor relationships to reduce dependency on a single supplier.
- Maintain open communication with vendors to stay informed about their operational status.
- Establish contingency agreements with backup vendors for critical services.

Monitoring:
- Regularly assess vendor performance and reliability through key performance indicators (KPIs).
- Monitor vendor news, financial stability, and industry trends that may affect their reliability.
- Maintain a process for immediate communication with vendors in case of issues.

Management:
1. Activate the contingency plan.
2. Initiate communication with the vendor to resolve the issue or transition to backup vendors.
3. Assess the impact on platform operations and customer experience.
4. Implement contingency measures, such as switching to backup suppliers or services.
5. Update affected stakeholders, including customers, on any service disruptions and expected resolution timelines.

Current Status:
This risk has been identified, and measures for both mitigation and monitoring are in place.

Originator: Om Shete	Assigned: Om Shete, Mohib Abbas Sayed, Hamza Sayyed

Risk Information Sheet			
Risk ID: R006	Date: 05/09/23	Prob: Medium	Impact: Medium

Description:
Insufficient Marketing Reach - There is a risk that the Urban Garden platform may struggle to reach a wide audience and potential customers due to limitations in marketing efforts.

Mitigation:
- Develop a comprehensive marketing strategy that includes online advertising, social media, email campaigns, and content marketing.
- Explore partnerships with gardening influencers or relevant online communities.
- Allocate a budget for marketing and promotions to ensure consistent outreach efforts.

Monitoring:
- Track key performance metrics for marketing campaigns, such as click-through rates, conversion rates, and customer acquisition costs.
- Monitor the effectiveness of different marketing channels and adjust strategies based on performance.
- Gather and analyse user feedback related to the effectiveness of marketing efforts.

Management:
1. Review and update the marketing strategy to identify areas for improvement.
2. Consider reallocating marketing budget to more effective channels.
3. Collaborate with marketing experts or agencies to optimize campaigns.
4. Explore additional marketing partnerships or collaborations to expand reach.
5. Maintain a proactive approach to user engagement and retention.

Current Status:
This risk has been identified, and measures for both mitigation and monitoring are in place.

Originator: Om Shete	Assigned: Om Shete, Mohib Abbas Sayed, Hamza Sayyed

Experiment No 11

Aim: Case Study: GitHub For Version Control.

Theory:

Configuration Management:

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

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

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

Some examples of software configuration metadata are:

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

It's easy for these configuration values to become an afterthought, leading to the configuration becoming disorganized and scattered. Imagine numerous Post-it notes

with passwords and URLs blowing around an office. Configuration management solves this challenge by creating a “sourceof truth” with a central location for configuration.

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

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

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

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

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

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

For almost all software projects, the source code is like the crown jewels - a precious asset whose value must be protected. For most software teams, the source code is a repository of the invaluable knowledge and understanding about the problem domain that the developers have collected and refined through careful effort. Version control protects source code from both catastrophe and the casual degradation of human error and unintended consequences. Software developers working in teams are

continually writing new source code and changing existing source code. The code for a project, app or software component is typically organized in a folder structure or "file tree". One developer on the team may be working on a new feature while another developer fixes an unrelated bug by changing code, each developer may make their changes in several parts of the file tree.

Version control helps teams solve these kinds of problems, tracking every individual change by each contributor and helping prevent concurrent work from conflicting. Changes made in one part of the software can be incompatible with those made by another developer working at the same time. This problem should be discovered and solved in an orderly manner without blocking the work of the rest of the team. Further, in all software development, any change can introduce new bugs on its own and new software can't be trusted until it's tested. So testing and development proceed together until a new version is ready.

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

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

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

Benefits of version control systems:
Using version control software is a best practice for high-performing software and DevOps teams. Version control also helps developers move faster and allows software teams to preserve efficiency and agility as the team scales to include more developers.

Version Control Systems (VCS) have seen great improvements over the past few decades and some are better than others. VCS are sometimes known as SCM (Source Code Management) tools or RCS (Revision Control System). One of the most popular VCS tools in use today is called Git. Git is a Distributed VCS, a category known as DVCS, more on that later. Like many of the most popular VCS systems available today, Git is free and open source. Regardless of what they are called, or which system is used, the primary benefits you should expect from version control are as follows.

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

2. Branching and merging. Having team members work concurrently is a no-brainer, but even individuals working on their own can benefit from the ability to work on independent streams of change. Creating a "branch" in VCS tools keeps multiple streams of work independent from each other while also providing the facility to merge that work back together, enabling developers to verify that the changes on each branch do not conflict. Many software teams adopt a practice of branching for each feature perhaps branching for each release, or both. There are many different workflows that teams can choose from when they decide how to make use of branching and merging facilities in VCS.

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

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

GitHub commands

Getting & Creating Projects

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

Basic Snapshotting

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

Branching & Merging

Command	Description
git branch	List branches (the asterisk denotes the current branch)
git branch -a	List all branches (local and remote)
git branch [branch name]	Create a new branch
git branch -d [branch name]	Delete a branch
git push origin --delete [branch name]	Delete a remote branch
git checkout -b [branch name]	Create a new branch and switch to it
git checkout -b [branch name] origin/[branch name]	Clone a remote branch and switch to it

git branch -m [old branch name] [new branch name]	Rename a local branch
git checkout [branch name]	Switch to a branch
git checkout -	Switch to the branch last checked out
git checkout -- [file-name.txt]	Discard changes to a file
git merge [branch name]	Merge a branch into the active branch
git merge [source branch] [target branch]	Merge a branch into a target branch
git stash	Stash changes in a dirty working directory
git stash clear	Remove all stashed entries

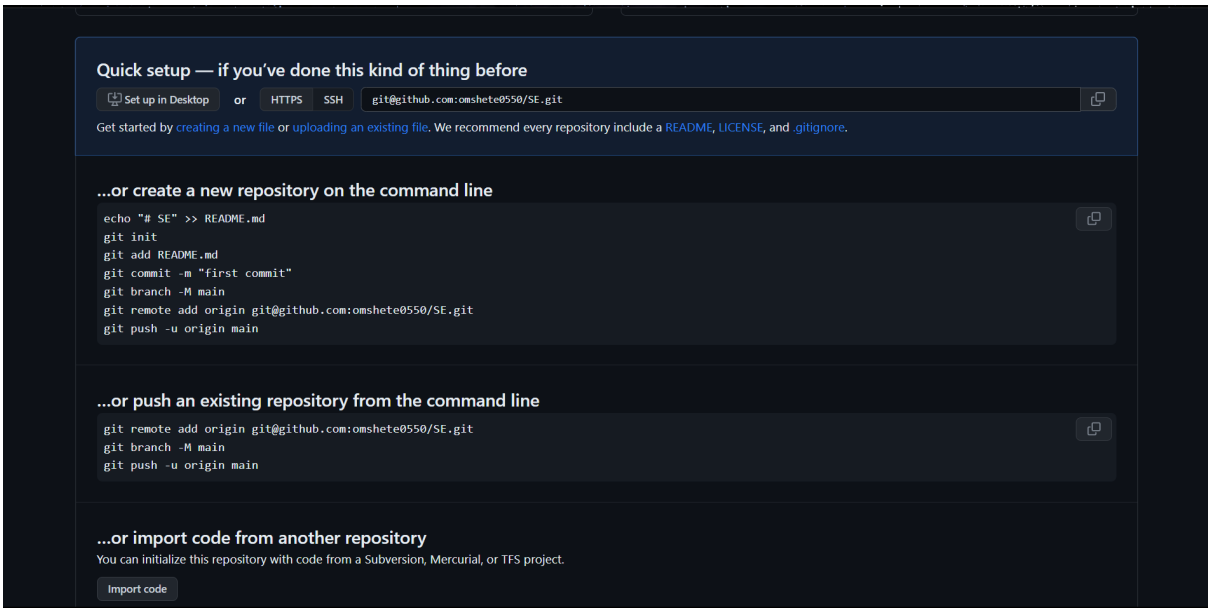
Sharing & Updating Projects

Command	Description
git push origin [branch name]	Push a branch to your remote repository
git push -u origin [branch name]	Push changes to the remote repository (and remember the branch)
git push	Push changes to the remote repository (remembered branch)
git push origin --delete [branch name]	Delete a remote branch
git pull	Update the local repository to the newest commit
git pull origin [branch name]	Pull changes from the remote repository
git remote add origin ssh://git@github.com/[username]/[repository-name].git	Add a remote repository
git remote set-url origin ssh://git@github.com/[username]/[repository-name].git	Set a repository's origin branch to SSH

Inspection & Comparison

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

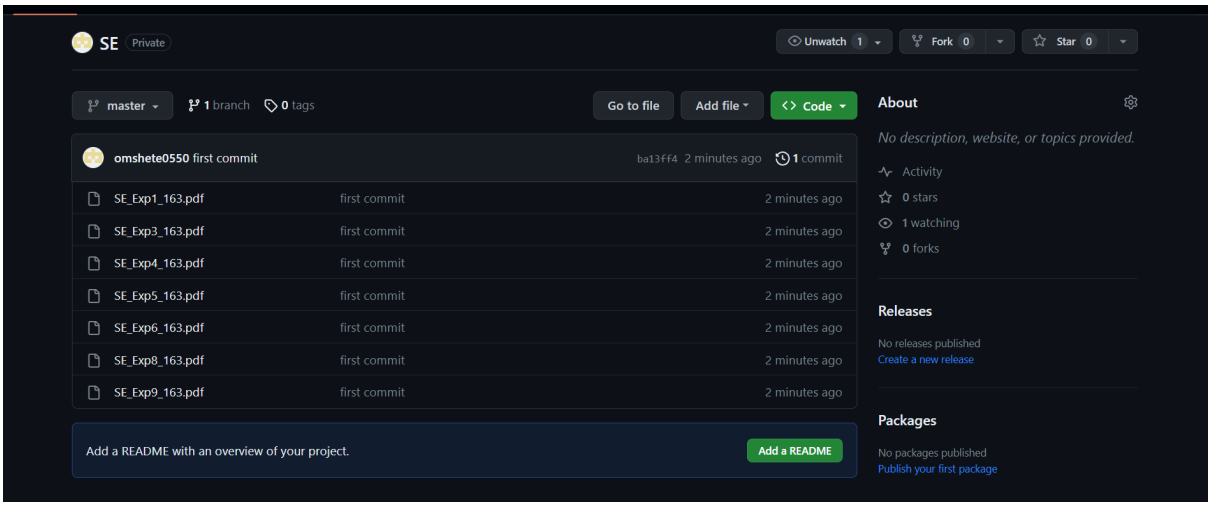
1. Creating a Repository:



2. Initial Commit

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS E:\College\Sem-V\SE_EXP> git init
Initialized empty Git repository in E:/College/Sem-V/SE_EXP/.git/
PS E:\College\Sem-V\SE_EXP> git add -A
PS E:\College\Sem-V\SE_EXP> git commit -m "first commit"
[master (root-commit) ba13ff4] first commit
 7 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 SE_Exp1_163.pdf
 create mode 100644 SE_Exp3_163.pdf
 create mode 100644 SE_Exp4_163.pdf
 create mode 100644 SE_Exp5_163.pdf
 create mode 100644 SE_Exp6_163.pdf
 create mode 100644 SE_Exp8_163.pdf
 create mode 100644 SE_Exp9_163.pdf
PS E:\College\Sem-V\SE_EXP> git remote add origin git@github.com:omshete0550/SE.git
PS E:\College\Sem-V\SE_EXP> git push -u origin master
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 20 threads
Compressing objects: 100% (9/9), done.
Writing objects: 100% (9/9), 3.08 MiB | 1.84 MiB/s, done.
Total 9 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:omshete0550/SE.git
 * [new branch]      master -> master
branch 'master' set up to track 'origin/master'.
PS E:\College\Sem-V\SE_EXP> git log
commit ba13ff4461fde1fa486850b12b08ee7558d5741a (HEAD -> master, origin/master)
Author: Om <omshete0550@gmail.com>
Date:   Fri Oct 13 21:16:19 2023 +0530

    first commit
PS E:\College\Sem-V\SE_EXP>
```

3. Commit History



4. Git Commands

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS E:\College\Sem-V\SE_EXP> git --help
usage: git [-v | --version] [-h | --help] [-C <path>] [-c <name>=<value>]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p | --paginate | -P | --no-pager] [--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           [--super-prefix=<path>] [--config-env=<name>=<envvar>]
           <command> [<args>]

These are common Git commands used in various situations:

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

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

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

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

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

'git help -a' and 'git help -g' list available subcommands and some
concept guides. See 'git help <command>' or 'git help <concept>'
to read about a specific subcommand or concept.
See 'git help git' for an overview of the system.
PS E:\College\Sem-V\SE_EXP>
```

Experiment No 12

Aim: Develop test cases for the projects using White Box Testing (JUnit)

Theory:

White Box Testing:

White box testing techniques analyze the internal structures the used data structures, internal design, code structure, and the working of the software rather than just the functionality as in black box testing. It is also called glass box testing or clear box testing or structural testing. White Box Testing is also known as transparent testing or open box testing.

White box testing is a software testing technique that involves testing the internal structure and workings of a software application. The tester has access to the source code and uses this knowledge to design test cases that can verify the correctness of the software at the code level.

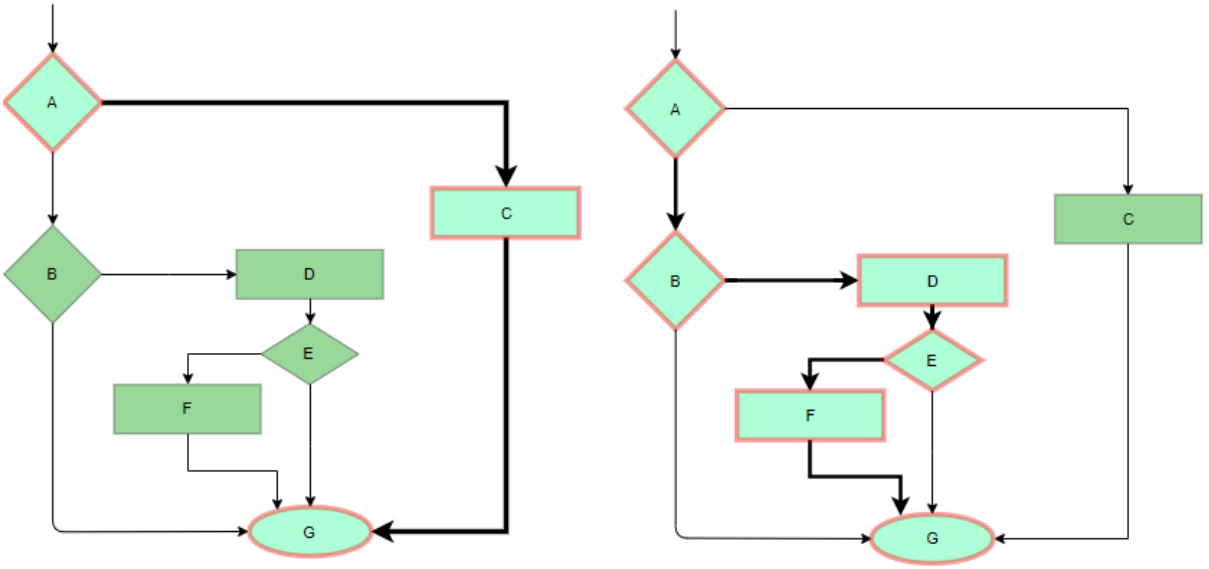
White box testing is also known as structural testing or code-based testing, and it is used to test the software's internal logic, flow, and structure. The tester creates test cases to examine the code paths and logic flows to ensure they meet the specified requirements.

Working process of white box testing:

- Input: Requirements, Functional specifications, design documents, source code.
- Processing: Performing risk analysis to guide through the entire process.
- Proper test planning: Designing test cases so as to cover the entire code. Execute rinse-repeat until error-free software is reached. Also, the results are communicated.
- Output: Preparing final report of the entire testing process.

Testing techniques:

Statement coverage: In this technique, the aim is to traverse all statements at least once. Hence, each line of code is tested. In the case of a flowchart, every node must be traversed at least once. Since all lines of code are covered, helps in pointing out faulty code.



Branch Coverage: In this technique, test cases are designed so that each branch from all decision points is traversed at least once. In a flowchart, all edges must be traversed at least once.

Condition Coverage: In this technique, all individual conditions must be covered as shown in the following example:

1. READ X, Y
2. IF(X == 0 || Y == 0)
3. PRINT '0'
4. #TC1 – X = 0, Y = 55
5. #TC2 – X = 5, Y = 0

Multiple Condition Coverage: In this technique, all the possible combinations of the possible outcomes of conditions are tested at least once. Let's consider the following example:

1. READ X, Y
2. IF(X == 0 || Y == 0)
3. PRINT '0'
4. #TC1: X = 0, Y = 0
5. #TC2: X = 0, Y = 5
6. #TC3: X = 55, Y = 0
7. #TC4: X = 55, Y = 5

Basis Path Testing: In this technique, control flow graphs are made from code or flowchart and then Cyclomatic complexity is calculated which defines the number of independent paths so that the minimal number of test cases can be designed for each independent path.

Steps:

1. Make the corresponding control flow graph
2. Calculate the cyclomatic complexity
3. Find the independent paths
4. Design test cases corresponding to each independent path
5. V(G) = P + 1, where P is the number of predicate nodes in the flow graph
6. V(G) = E – N + 2, where E is the number of edges and N is the total number of nodes
7. V(G) = Number of non-overlapping regions in the graph
8. #P1: 1 – 2 – 4 – 7 – 8
9. #P2: 1 – 2 – 3 – 5 – 7 – 8
10. #P3: 1 – 2 – 3 – 6 – 7 – 8
11. #P4: 1 – 2 – 4 – 7 – 1 – . . . – 7 – 8

Loop Testing: Loops are widely used and these are fundamental to many algorithms hence, their testing is very important. Errors often occur at the beginnings and ends of loops.

1. Simple loops: For simple loops of size n, test cases are designed that:

- Skip the loop entirely
- Only one pass through the loop
- 2 passes
- m passes, where m < n
- n-1 ans n+1 passes

2. Nested loops: For nested loops, all the loops are set to their minimum count and we start from the innermost loop. Simple loop tests are conducted for the innermost loop and this is worked outwards till all the loops have been tested.

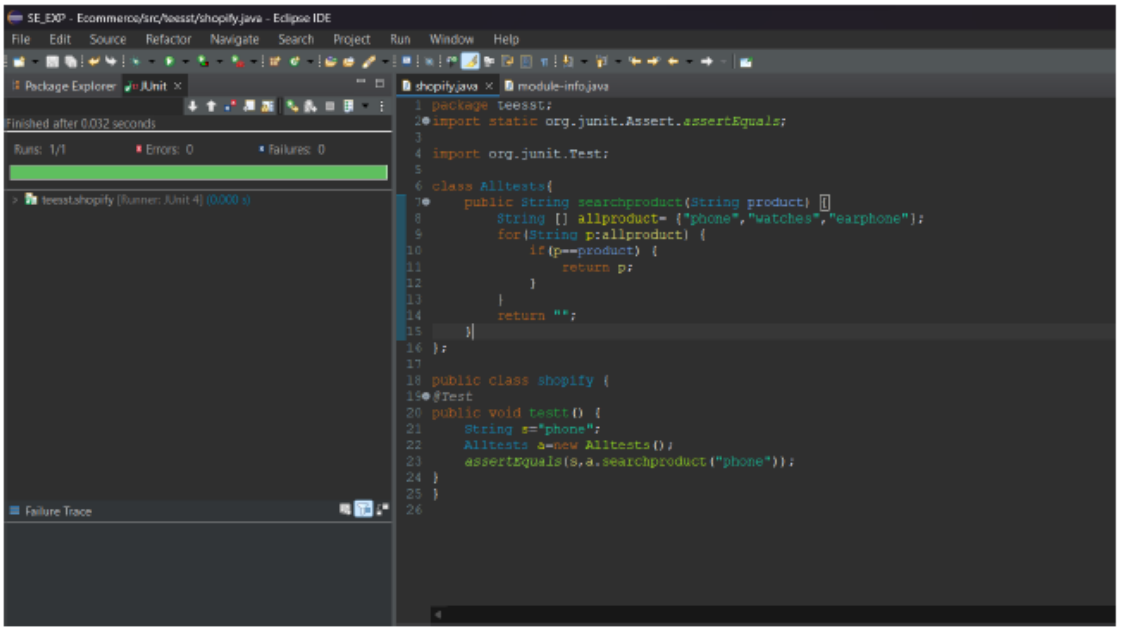
3. Concatenated loops: Independent loops, one after another. Simple loop tests are applied for each. If they’re not independent, treat them like nesting.

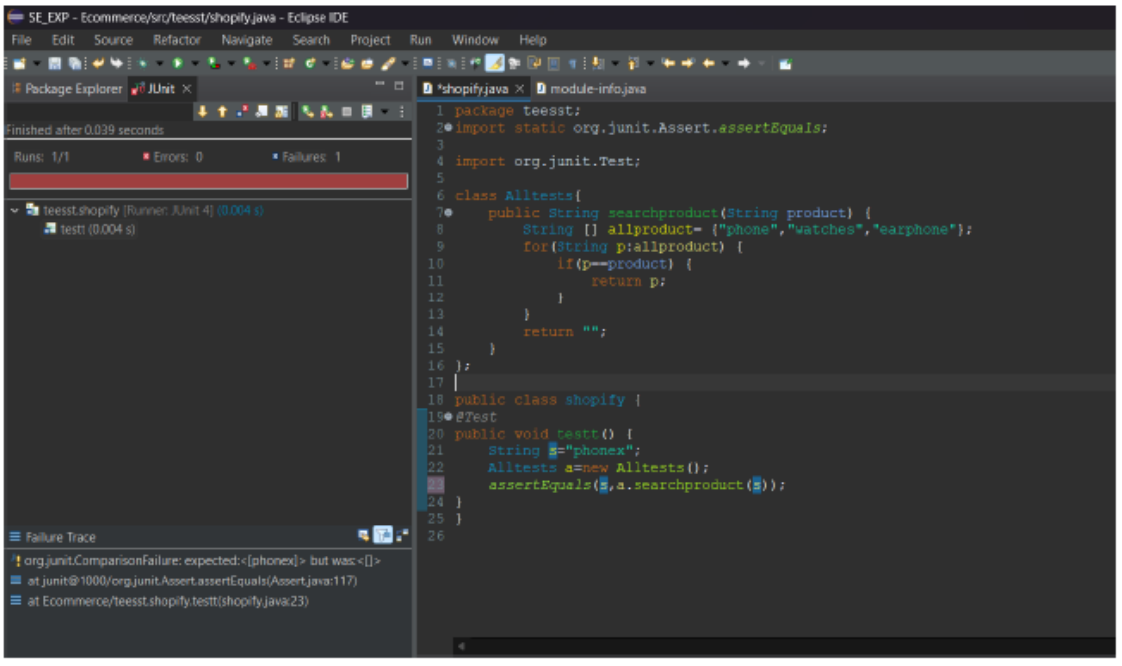
Code:

```java
package teesst;
import static org.junit.Assert.assertEquals;
import java.util.HashMap;
import java.util.Map;
import org.junit.Test;

class Alltests{
    public String searchproduct(String product) {
        String [] allproduct= {"phone","watches","earphone"};
        for(String p:allproduct) {
            if(p==product) {
                return p;
            }
        }
        return "";
    }
      public String testSearchProductFoundbyorderno(Integer orderno) {
            Map<Integer, String> orderMap = new HashMap<>();
            orderMap.put(1, "phone");
            orderMap.put(2, "watches");
            orderMap.put(3, "earphone");
                for (Integer key : orderMap.keySet()) {
                    String value = orderMap.get(key);
                    if(key==orderno) {
                        return value;
                    }
                }
                return "";
      }
};

public class shopify {
@Test
public void testt() {
    String s="phone";
    Alltests a=new Alltests();

    assertEquals(s,a.searchproduct(s));
}
@Test
public void test2() {
    String temp="aaaa";
    String [] allproduct= {"phone","watches","earphone"};
    Integer orderno=3;
    Alltests a=new Alltests();
    String result = (orderno <= allproduct.length) ? allproduct[orderno - 1] : temp;
    assertEquals(result,a.testSearchProductFoundbyorderno(orderno));
}
}
```

Output:

Test 1: Search a Product.





Om Shede
THADOMAL SHAHANI TE C32
TSEC ENGINEERING COLLEGE 2103163

Assignment 1

Rusari 03/10/23 A+

Q.1 Architectural Design : Explain in detail each type with example

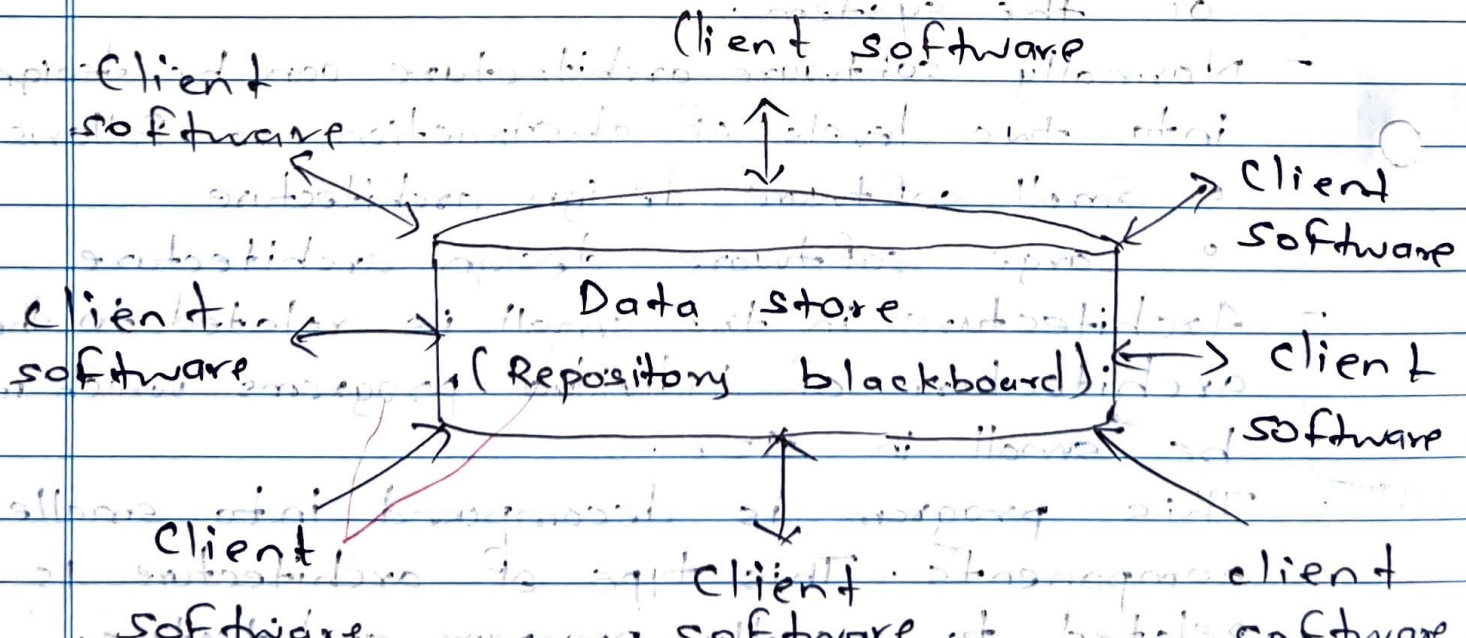
⇒
- Architectural design is backbone of any software system design and it is responsible for the overall system of structure.
- In nearly all the models of software process, architectural design is considered as the first stage in the software design and developement process.
- The output of the architectural design process is an architectural model that explains the overall structure of the system and also explains the working of the system and the communication among various components of the system.
- Normally software architecture can be designed into two levels of abstraction as follows:
  - Small software design architecture
  - Large software design architecture
- Architecture in the small is related to the architecture of individual programs which may be small in size.
- This program is decomposed into smaller components. This type of architecture is related to mostly program architecture.
- Architecture in the large is related to complex systems that includes overall system, the entire program and the

components of the program.
- The non-functional system requirements are totally depend on the system architecture whereas functional system requirements are dependent on the individual components
- Types of Architecture Design :-

1) Data Centered architecture

⇒ A data store will reside at the center of this architecture and is accessed frequently by the other components that update, add, delete or modify the data present within the store.

- The figure illustrates a typical data centered style



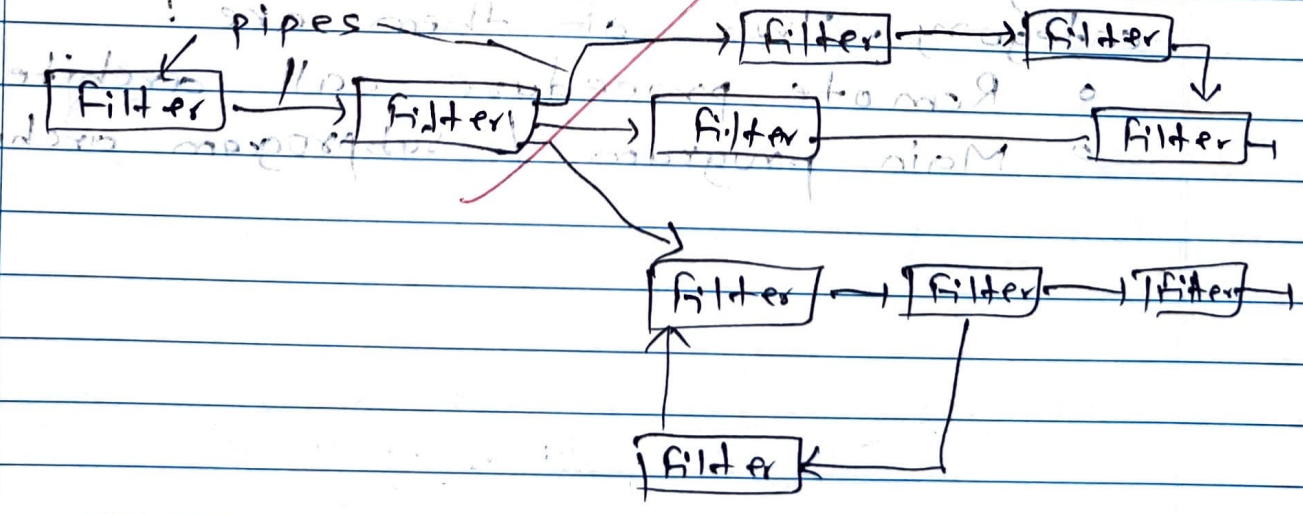
- The client software access a central repository.

- 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 notification to client software.
- This data-centered architecture will promote integrability.

2) Data Flow Architecture

⇒ This kind of architecture is used when input data is transformed into output data through a series of computational manipulative components.

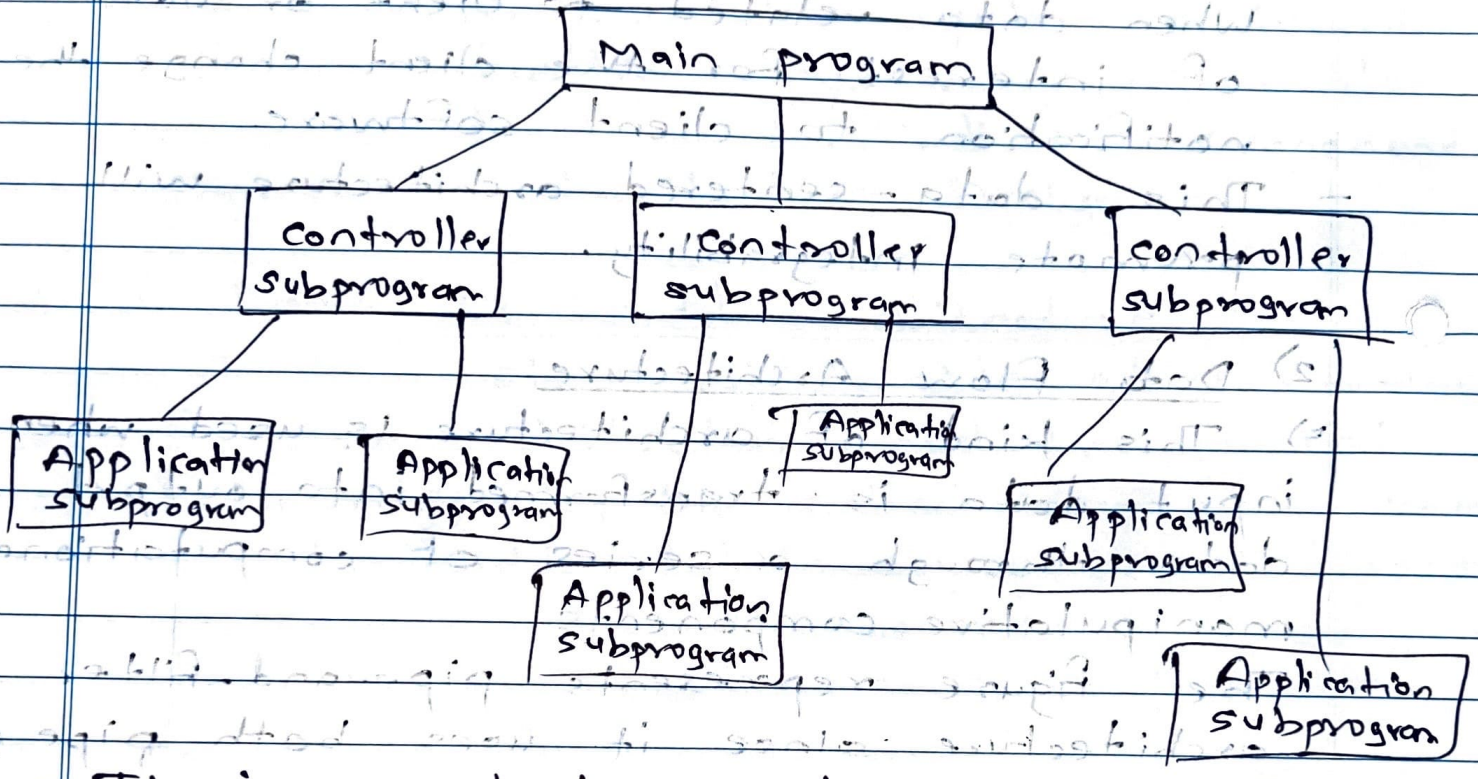
The figure represents pipe-and-filter architecture since it uses both pipe and filter and it has a set of components called filters connected by lines.



THADOMAL SHAHANI
TSEC
ENGINEERING COLLEGE

3) Call and return architecture

=>



- It is used to create a program that is easy to scale and modify.
- Many sub-style exists within this category. Two of them are :
  - Remote procedure call architecture
  - Main program / Subprogram architecture

4) Object-oriented architecture

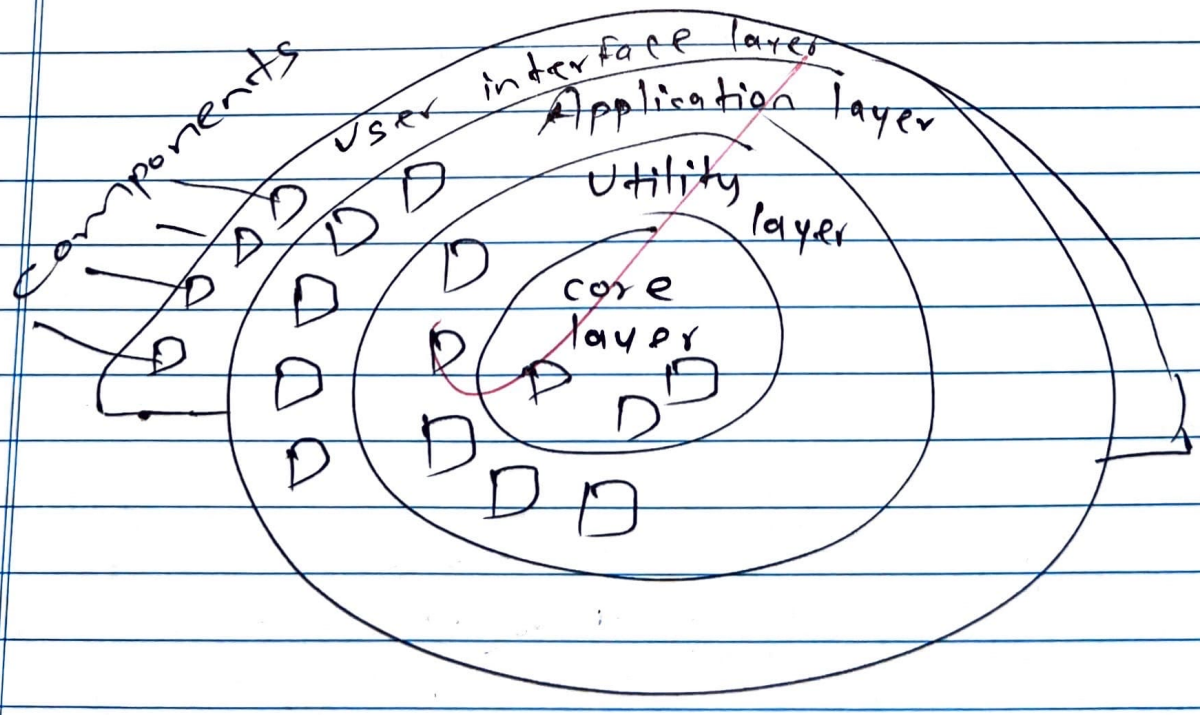
⇒ The components of a system encapsulates data and the operations that must be applied to manipulate the data.

- The co-ordination and communication between the components are establish via the message passing.

5) Layered architecture

⇒ A number of different layers are defined with each layer performing a well defined set of operation.

- Each layer will do some operations that becomes closer to machine instruction set progressively.



Om Shete
THADOMAL SHAHANI TSEC ENGINEERING COLLEGE
TE C32
2103163

Assignment 2

Q.1 Explain in detail

Bhusari 8/10/23 A+

a) Software Maintainance

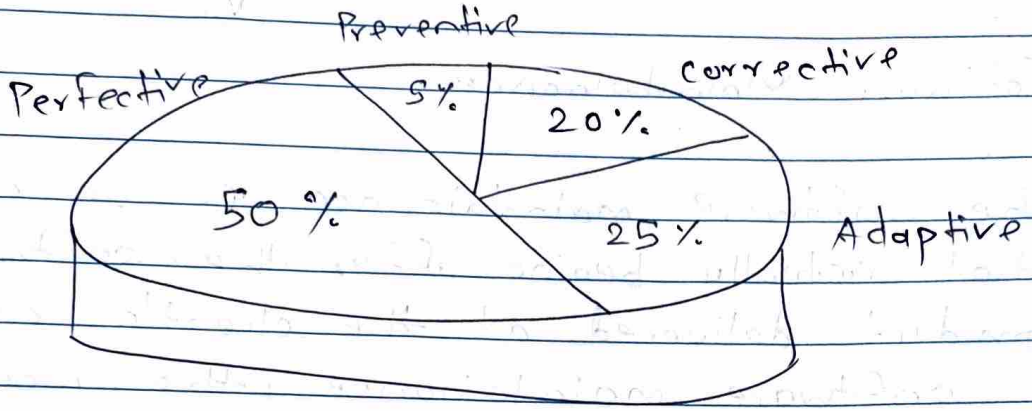
⇒

- The software maintainance is an activity that actually begins after the software product delivered at the client's end.
- In software maintainance, the modifications are carried out or the updates in the software are taken place.
- In software maintainance phase no major changes are implemented.
- In software maintainance, the changes are done in the existing program or the some small new functionality is added.
- The modifiability means the ability of the software to be modified. The large number of definition of quality exists
- Maintainability is defined as the ability of the software system to be modified.
- These modifications include: improvements, corrections, adapatability in changing environments and in changing requirements and the functional specifications.
- There are four types of maintainance

⇒

1) Corrective maintainance
2) Adaptive maintainance
3) Perfective maintainance

4) Preventive maintainance.



1) Corrective Maintainance

⇒ The corrective maintainance is the type of maintainance in which the errors are fixed when it is observed during the use of the software.

- In this the errors may be caused due to faculty software design, incorrect logic and improper coding.
- All these errors are called as residual errors and they prevent the final specification.

2) Adaptive Maintainance

⇒ The adaptive maintainance means the implementation of the modification in the system.

- The changes may be of hardware or the operating system environments.
- For e.g., business rules, work patterns.

3) Perfective Maintainance

⇒ Perfective maintainance deals with the modified and changed user requirements.

- The functional enhancements are taken into consideration in perfective maintainance.
- In this, the function and efficiency of the code is continuously improved.
- For e.g.,
  Modifying the payroll program to add new union settlement, Adding a new report in the sales analysis system.

4) Preventive Maintainance

⇒ Preventive maintainance is used to prevent the possible errors to occur.

- Thus in this activity, the complexity is minimised and the quality of the program is enhanced.
- Adaptive maintainance normally used for 5% of all the maintainance activities and it is the smallest among all.

b) Re-Engineering

=>

- The business level reengineering is done by business experts whereas the process of reengineering software level is done by software engineers and software developers.
- The demands on business functions and the information technology depends that support them are changing at rapid pace that puts huge amount of pressure on every organization.
- Both the business and software that supports the business must be reengineered to keep pace.
- Business process reengineering (BPR) defines business goals, identifies and evaluates existing business processes, and creates revised business processes that better meet current goals.
- A software reengineering process encompasses inventory analysis, document restructing, reverse engineer, program and data restructing and forward engineering.
- The intent of these activities is to create versions of existing programs that exhibit higher quality and better maintainability.

- A variety of reengineering work products are produced. The final output is the reengineered business process and/or the reengineered software that supports it.
- In order to ensure the correct approaches, use the same SQA practices that are applied in every software engineering process, format technical reviews access the analysis and design models, specialized reviews consider business applicability and compatibility, testing is applied to uncover errors in content, functionality and interoperability.

c) Reverse Engineering.

=>

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

- Enabling additional feature
- Used as learning tools
- Developing compatible products cheaper that that are currently available in the market.

- Following are the three important parameters to be considered for of a reverse engineering process :

1) Abstraction level

⇒ In the abstraction level of a reverse engineering process, the design information is extracted from the source code.
- It is the highest level in the reverse engineering process.
- It is always expected that the abstraction level for any reverse engineering process must be high.
- When the level of abstraction is high, then it becomes easy for the software developer to understand the program.

2) Completeness

⇒ The completeness is nothing but the details available through the abstraction level of reverse engineering process.
- For example, from the given source code it is very easy to develop the complete procedural design.

3) Directionality

=> The directionality of a reverse engineering process is a method of extracting information from the source code and making it available to the software developer.

- The software developer can use this information during the maintenance activity.
- Following diagram exhibits the reverse engineering process.