**6.a) Distinguishing Between Plan-Driven Development Approach and Agile Development Approach:**

1. **Plan-Driven Development (Traditional/Waterfall Approach):**
   * **Process:** Follows a linear, sequential process with defined phases such as requirements gathering, design, implementation, testing, and maintenance.
   * **Documentation:** Heavy documentation is required at each phase before moving to the next, ensuring all requirements and designs are fully detailed upfront.
   * **Flexibility:** Low flexibility. Changes to requirements after the project starts can be expensive and difficult to implement.
   * **Customer Involvement:** Limited customer involvement after the initial phase, with client feedback typically collected at the end of the project.
   * **Delivery:** The product is usually delivered at the end of the development cycle.
   * **Best for:** Projects with well-understood and stable requirements, such as government projects or safety-critical systems.
2. **Agile Development:**
   * **Process:** Iterative and incremental approach where the project is divided into small sprints or iterations. Each sprint results in a working part of the system.
   * **Documentation:** Documentation is minimal and focuses more on code and collaboration. Agile values "working software over comprehensive documentation."
   * **Flexibility:** High flexibility. Agile embraces changing requirements even late in the development cycle, allowing for continuous adaptation.
   * **Customer Involvement:** High involvement, with regular feedback from customers or stakeholders throughout the process.
   * **Delivery:** Working software is delivered incrementally, often in a few weeks.
   * **Best for:** Projects with evolving requirements, such as startups, web applications, or software with a fast-changing market.

The **Agile method** is a software development approach that emphasizes **flexibility, collaboration, and iterative progress**. It focuses on delivering small, working pieces of software frequently, allowing for continuous feedback and adaptation to changing requirements throughout the development process.

**Key Characteristics of the Agile Method:**

1. **Iterative and Incremental Development:**
   * Work is broken into small, manageable units called **sprints** (typically lasting 1-4 weeks), each delivering a functional part of the product.
2. **Continuous Feedback:**
   * Frequent interactions with stakeholders or customers are encouraged to gather feedback and ensure the product aligns with their needs.
3. **Adaptability to Change:**
   * Agile embraces changes in project requirements, even late in the development cycle, allowing for ongoing adjustments based on evolving conditions.
4. **Collaboration and Communication:**
   * Agile relies heavily on team collaboration, with daily meetings (e.g., **stand-ups**) to track progress, identify obstacles, and ensure alignment within the team.
5. **Minimal Documentation:**
   * Agile focuses more on delivering working software than on extensive documentation. It values simplicity and direct communication.
6. **Self-Organizing Teams:**
   * Teams are empowered to make decisions and self-organize, fostering a sense of responsibility and ownership over the product.

**Benefits of Agile:**

* **Faster Time to Market:** Frequent releases ensure that usable versions of the software are delivered more quickly.
* **Customer-Centric:** Continuous feedback ensures that the product remains aligned with customer needs and expectations.
* **Reduced Risk:** Early detection of issues allows for quick adjustments, reducing the risk of project failure.

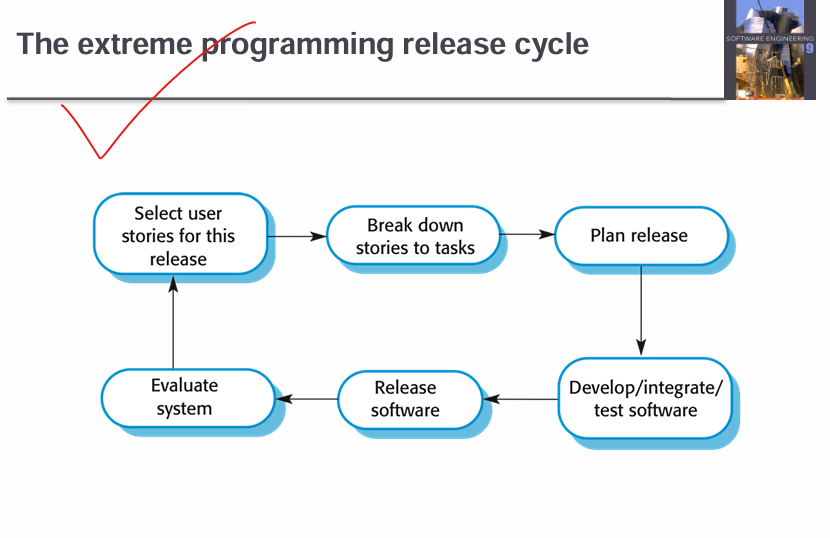
**Common Agile Methodologies:**

* **Scrum:** Focuses on sprints, with defined roles like Scrum Master, Product Owner, and Development Team.
* **Kanban:** Emphasizes continuous delivery and visual workflow management.
* **Extreme Programming (XP):** Focuses on technical excellence and practices like Test-Driven Development (TDD) and Pair Programming.

**b) Agile Method Specific Problems and Applicable Areas:**

1. **Agile Method Specific Problems:**
   * **Unclear Requirements:** If the project lacks clear vision or the customer is not sure of their requirements, it may lead to scope creep and confusion in Agile teams.
   * **Documentation Issues:** Minimal documentation can lead to problems during maintenance or when a new team takes over the project.
   * **Lack of Predictability:** Due to its iterative nature, Agile may struggle with accurate cost estimation and time planning for long-term projects.
   * **Teamwork Dependency:** Agile relies heavily on strong teamwork, communication, and collaboration. If the team lacks maturity or discipline, productivity can suffer.
   * **Scaling Issues:** Scaling Agile to large teams or organizations can be challenging, leading to issues in coordinating efforts across teams.
2. **Applicable Areas:**
   * **Small to Medium-Sized Projects:** Agile is well-suited for projects with small teams where communication is frequent and manageable.
   * **Rapidly Changing Markets:** Projects that need to adapt to changing customer needs, technologies, or market conditions benefit from Agile's flexibility.
   * **Customer-Focused Products:** Projects where continuous customer feedback and involvement are critical, such as mobile apps or web development.
   * **Startups:** Agile works well for startups due to its focus on iteration, early delivery, and adaptability.

**c) Extreme Programming (XP) and Its Release Cycle and Principles of Practice:**

1. **Definition of Extreme Programming (XP):** Extreme Programming (XP) is an Agile software development methodology that emphasizes frequent releases in short development cycles, which improves productivity and introduces checkpoints for new customer requirements. XP focuses on technical excellence and good programming practices.
2. **XP Release Cycle:**
   * **Iterations:** XP uses short iterations that typically last 1-2 weeks. Each iteration delivers a small, functional part of the software.
   * **Releases:** After several iterations (often 1-3 months), a release is made available to the customer. This release is considered a usable version of the product.
   * **Frequent Releases:** XP advocates for frequent small releases that allow for quick feedback and adaptation based on user input or changes in the business environment.
   * 
3. **Principles of XP Practice:**
   * **Continuous Integration:** Code is integrated and tested frequently, sometimes several times a day, to avoid integration issues and bugs.
   * **Test-Driven Development (TDD):** Developers write tests before writing the actual code. Code is considered complete only when all tests pass.
   * **Pair Programming:** Two developers work together at one computer. One writes the code, while the other reviews each line of code as it’s written.
   * **Collective Code Ownership:** Everyone in the team is responsible for the entire codebase, and anyone can make changes to any part of the code at any time.
   * **Simplicity:** Developers focus on writing simple, clean code that meets current requirements without over-engineering or anticipating future changes.
   * **Sustainable Pace:** XP promotes a work-life balance by encouraging a steady, sustainable pace, avoiding long hours or "crunch" periods.
   * **Customer Involvement:** A customer representative is part of the development team, providing continuous feedback and clarifying requirements as needed.
   * **Small releases**
   * **Simple design**

**5.a) Briefly describe about modified waterfall model.4**

**b) Show the data centered software architecture, data flow architecture and call-return architecture design.3**

**a) Modified Waterfall Model:**

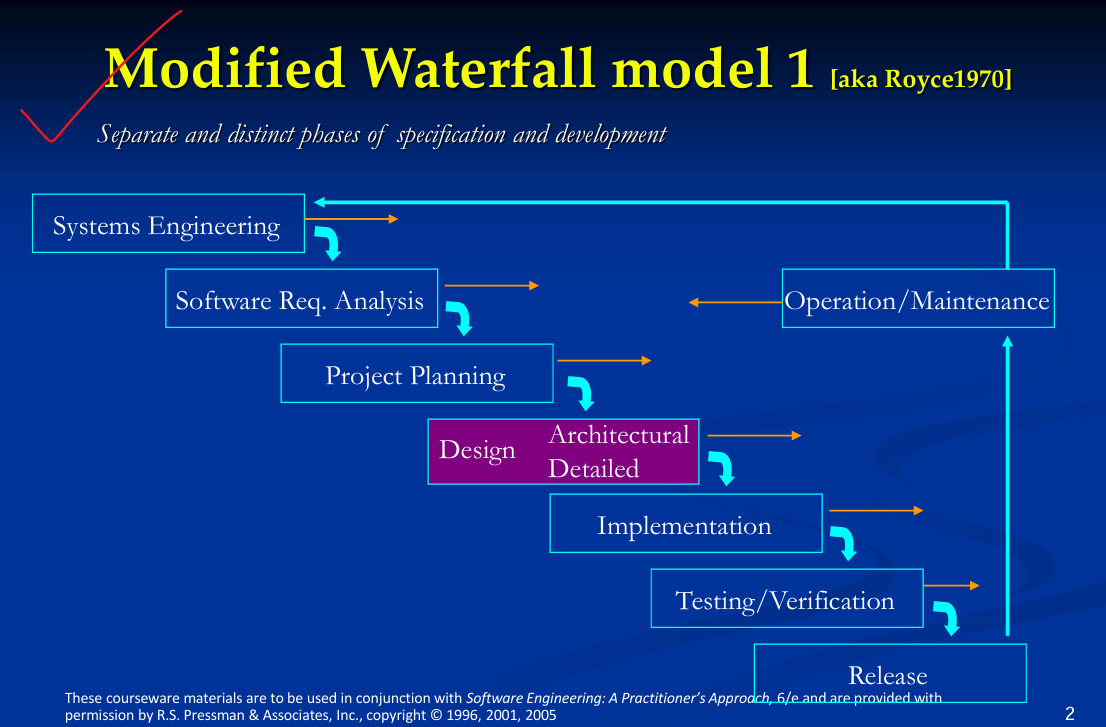
The **Modified Waterfall Model** (also known as the **V-Model**) is an adaptation of the traditional Waterfall model, designed to address some of its rigid limitations. While it follows the same sequential stages (such as requirements, design, implementation, testing, and maintenance), it allows for slight overlapping between phases or feedback loops.

**Key Features:**

1. **Flexibility for Feedback:** Unlike the strict linear flow in the traditional Waterfall, the modified version allows for revisiting previous stages (e.g., going back to requirements if issues arise during design).
2. **Validation and Verification:** Testing is integrated at every phase of the development process, improving the chances of detecting defects early.
3. **Documentation Focused:** Like the original model, it emphasizes extensive documentation at each phase but adapts better to changes.

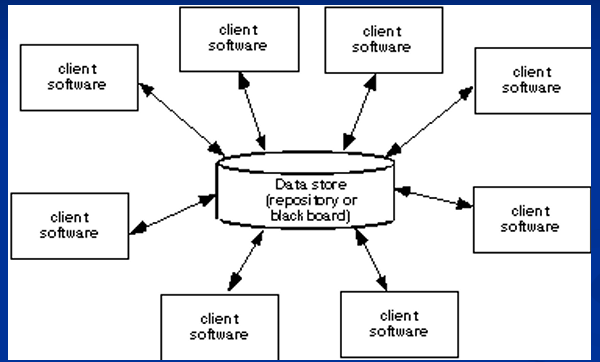
**Applications:**

This model is suitable for projects with clear requirements but where slight flexibility is required to handle feedback during development, like in some government or large enterprise systems.

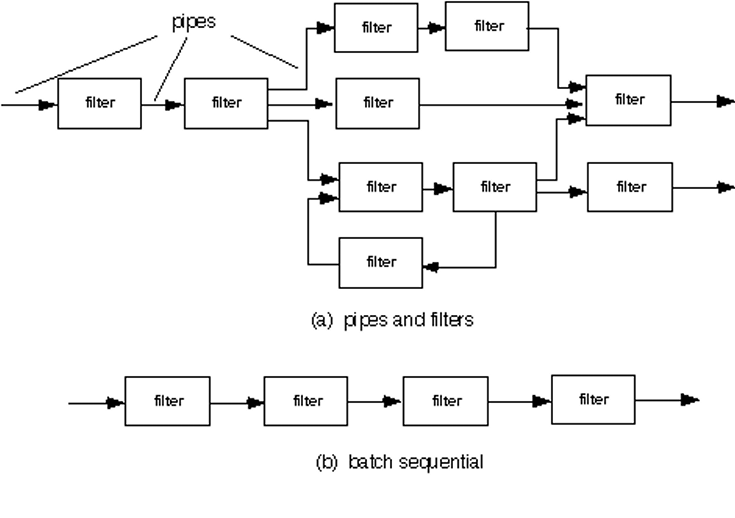
****

**b) Software Architecture Designs:**

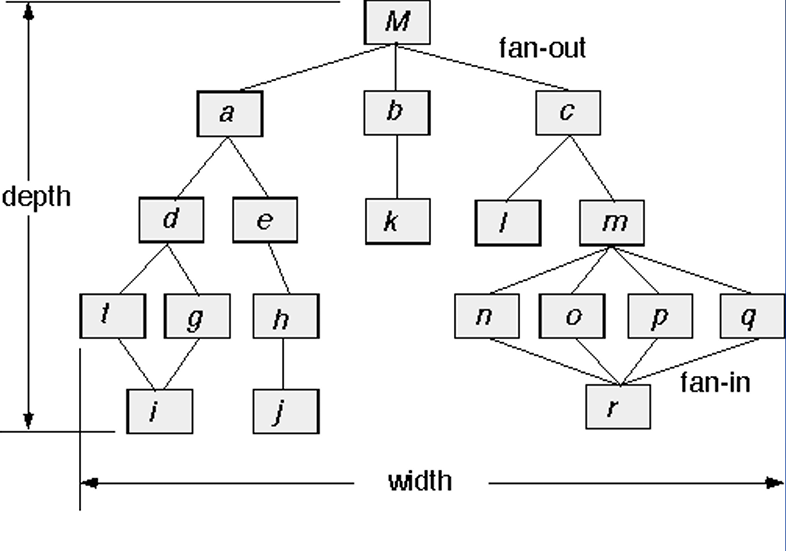
1. **Data-Centered Software Architecture:**
   * In this architecture, the system revolves around a central data repository (like a database), with various components or clients accessing and modifying the data.
   * Example: **Repository Architecture**, where components work with a shared central database. The system evolves around data storage, retrieval, and maintenance.
   * **Diagram:**



1. **Data Flow Architecture:**
   * This architecture is designed around the flow of data through the system in the form of input, transformation, and output.
   * Example: **Pipe-and-Filter Architecture**, where the system is made up of data transformation components (filters) connected by data conduits (pipes).
   * **Diagram:**



1. **Call-Return Architecture (Hierarchical Architecture):**
   * A call-return architecture organizes the system as a hierarchy of modules, where higher-level modules call lower-level ones.
   * Example: **Main-Subroutine Design**, where the system has a main control module and other subordinate modules that handle specific tasks.
   * **Diagram:**



**c) c) What is UI/UX design in software engineering? Write down the steps for interface analysis and user analysis. 3**

**UI/UX Design in Software Engineering:**

**UI (User Interface) Design** refers to the graphical layout and presentation of a software interface that users interact with, including buttons, icons, text, and other visual elements.

**UX (User Experience) Design** focuses on the overall experience of the user when interacting with the software, ensuring usability, efficiency, and satisfaction.

**Steps for Interface Analysis:**

1. **User Research:** Understand user needs, behaviors, and goals by conducting surveys, interviews, or focus groups.
2. **Task Analysis:** Analyze the tasks users need to accomplish with the system and determine what actions the interface should facilitate.
3. **Environment Analysis:** Assess the physical and social environment in which the system will be used.
4. **Prototyping:** Create mockups or wireframes that depict the layout and functionality of the user interface.
5. **Usability Testing:** Evaluate the interface with real users to test its effectiveness and gather feedback.

**Steps for User Analysis:**

1. **Identify User Roles:** Define the different types of users who will interact with the system (e.g., admin, end-user).
2. **Create User Personas:** Develop user profiles to represent typical users based on research.
3. **User Journey Mapping:** Visualize the steps users take when interacting with the software to identify pain points and improvement opportunities.
4. **Behavioral Patterns:** Observe how users interact with similar systems to anticipate their expectations and preferences.

**d) Illustrate evolutionary development in software engineering process with its problem and applications. Show incremental development process.4**

**d) Evolutionary Development in Software Engineering Process:**

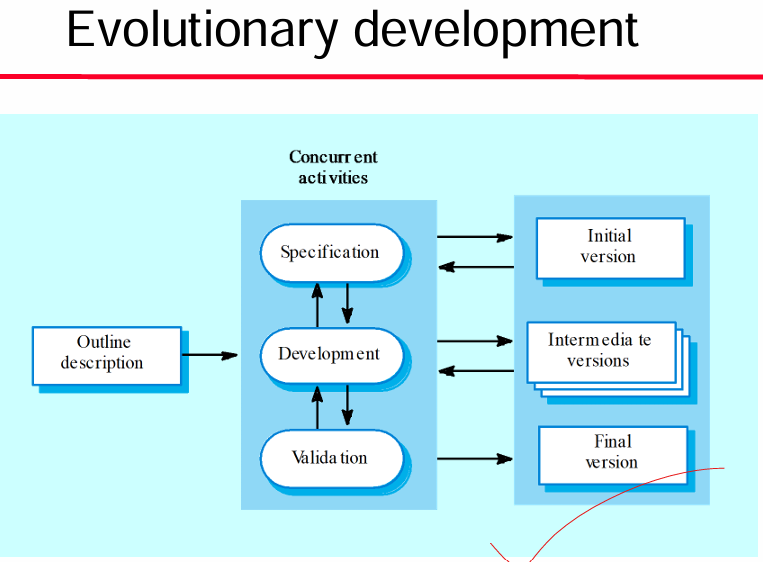
**Evolutionary Development** is a software engineering process where the system is developed incrementally over time through iterative process. It allows continuous delivery of a working system that can evolve based on user feedback and changing requirements.

**Key Features: (Extra)**

1. **Iterative Cycles:** The system is built in successive versions or prototypes, which are refined in each iteration.
2. **Continuous Feedback:** Feedback is continuously incorporated into each iteration, ensuring the system stays aligned with user needs.
3. **Flexibility:** It supports flexibility to accommodate requirement changes during development.

**Problems:**

* **Scope Creep:** Continuous changes can lead to uncontrolled expansion of project scope.
* **Difficulty in Long-Term Planning:** It's challenging to estimate the total effort, cost, and timeline upfront due to frequent changes.
* **Maintenance Complexity:** Managing and maintaining evolving prototypes can become complex over time.

****

**Applications:**

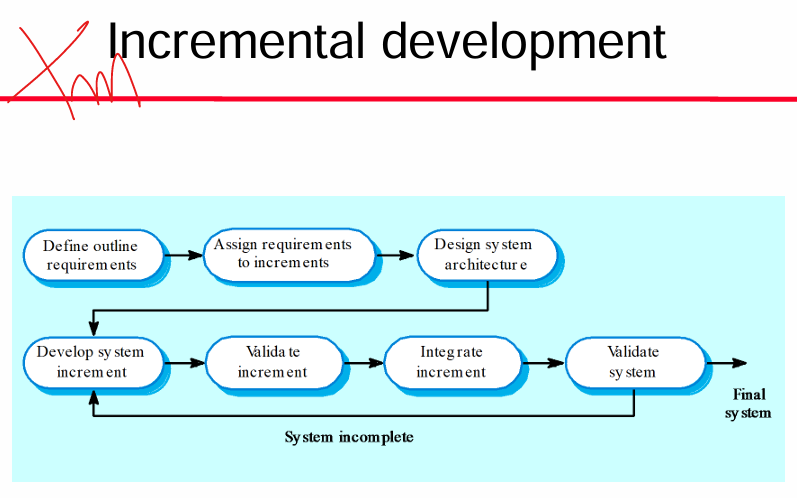
* **Rapid Prototyping:** For projects where initial requirements are unclear and need to evolve, such as innovative product development.
* **Long-Term Projects:** For systems that require long-term flexibility, like web applications or complex enterprise software.

**Incremental Development Process:**

In **Incremental Development**, the system is built piece by piece, with each part delivering functionality that can be independently tested and used. Each increment adds to the previous releases until the full system is complete.

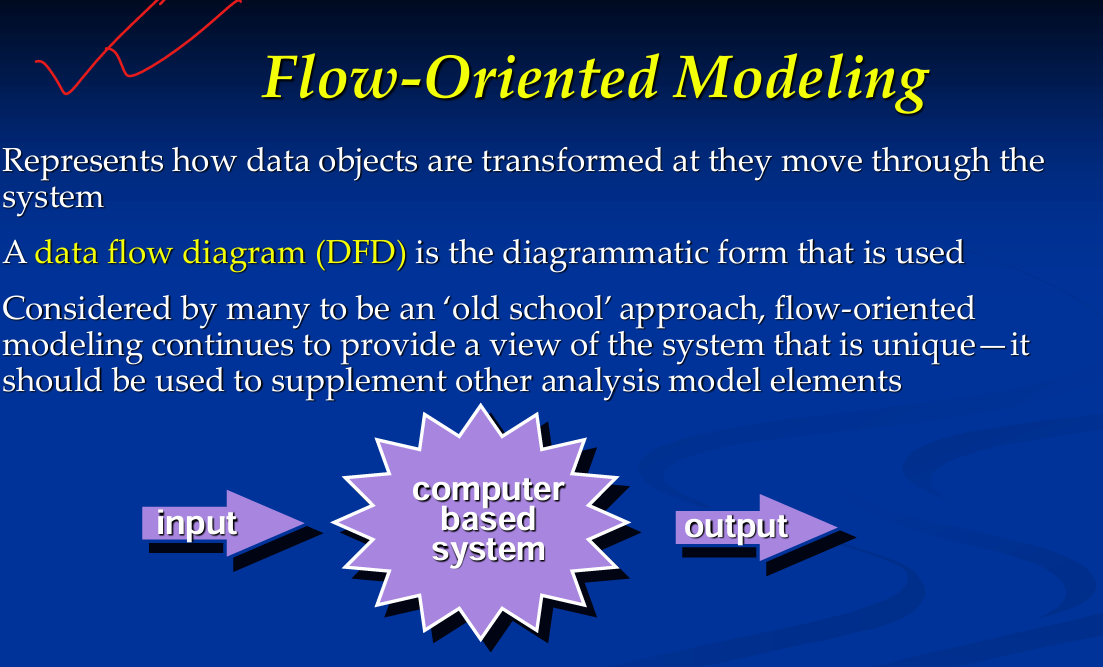
**Steps of Incremental Development:**

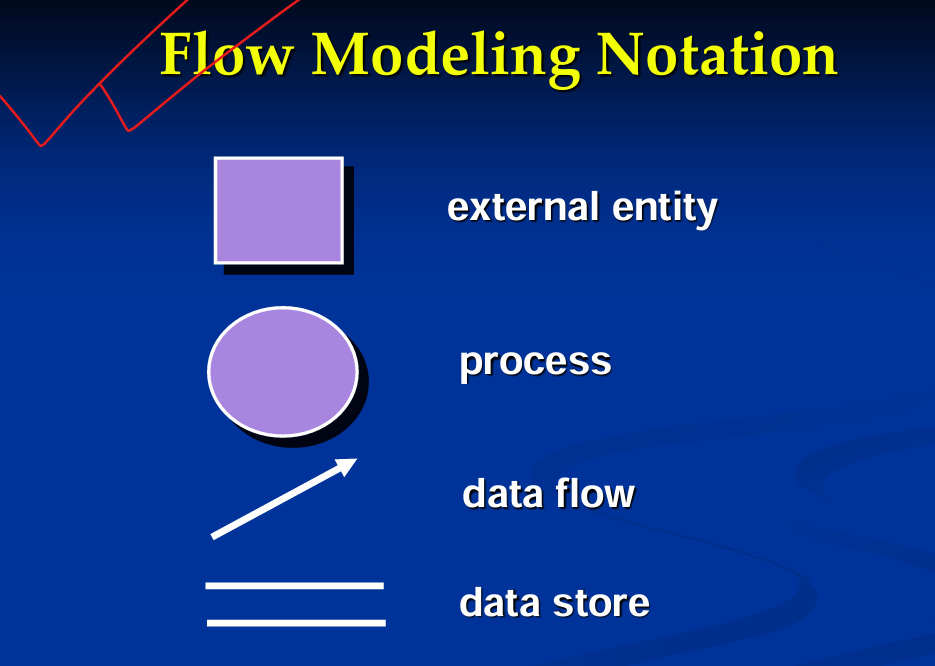
1. **Initial Planning:** Identify the system's overall functionality and divide it into smaller, manageable modules or increments.
2. **Increment 1 (Core):** Develop the core, essential features first. Deliver this part to the user for feedback.
3. **Increment 2 (Extension):** Add more features in the next increment based on user feedback from the first release.
4. **Subsequent Increments:** Continue adding features, testing each increment independently, and incorporating feedback.
5. **Final Integration:** After several increments, integrate all parts into the final product.



4.

a) What is flow oriented modeling in software engineering? Show the Flow Modeling Notation.3





b) Write down the steps of Data Flow diagramming guidelines. Give a level 0 DFD example.3

Data Flow Diagramming: Guidelines

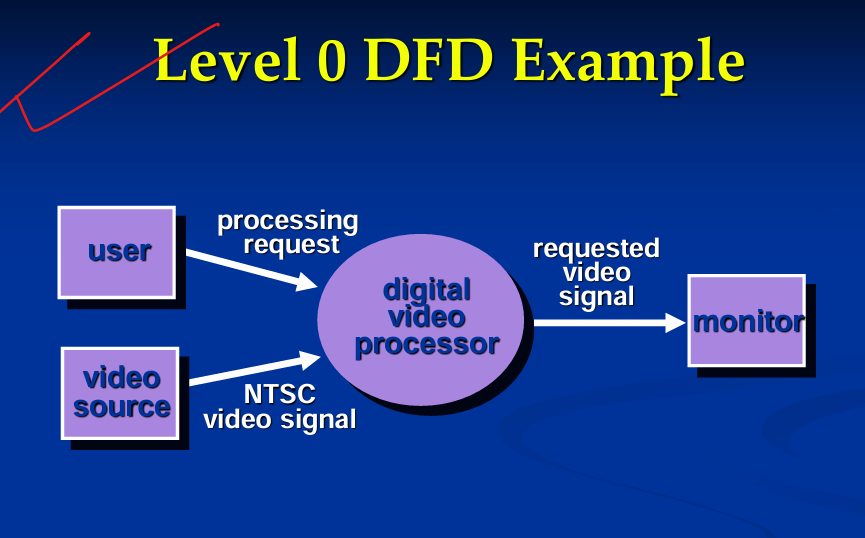
◼ all icons must be labeled with meaningful names

◼ the DFD evolves through a number of levels of detail always begin with a context level diagram (also called level 0)

◼ always show external entities at level 0

◼ always label data flow arrows

◼do not represent procedural logic



c) **Why control flow diagram (CFD) is a very helpful tool for both systems developers and stakeholders? Give an example of CFD.3**

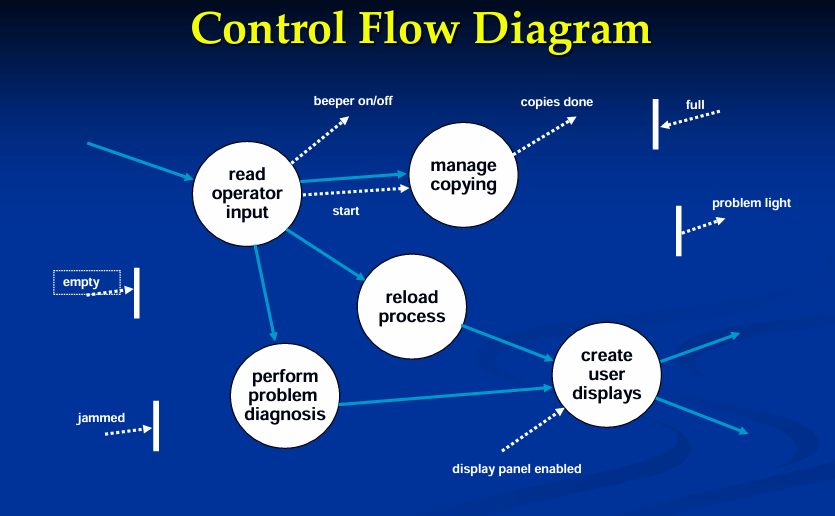
A **Control Flow Diagram (CFD)** is a visual representation that shows the flow of control or decision-making within a system or process. It illustrates how different components of the system interact, the sequence of operations, and how information or control passes through various stages of the process.

**1. For System Developers:**

* **Clear Representation of Logic:** CFD provides a structured, visual way to see how control flows through the system, making it easier to understand complex decision-making processes.
* **Helps Debugging and Error Detection:** By breaking down the system into manageable control flow segments, developers can identify potential bottlenecks, points of failure, or bugs.
* **System Optimization:** Developers can see inefficient loops or redundant steps, making it easier to optimize the performance of the system.
* **Simplifies Code Mapping:** Developers can use the CFD to directly map out and organize code structures, facilitating the writing and maintenance of code.

**2. For Stakeholders:**

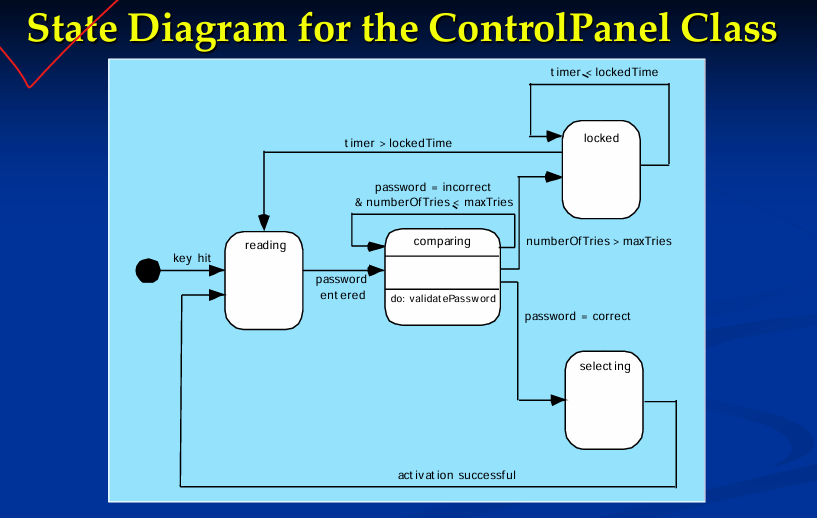
* **Non-Technical Understanding:** A CFD offers stakeholders a simplified, visual understanding of how the system operates without needing detailed technical knowledge.
* **Communication Tool:** It acts as a communication bridge between developers and non-technical stakeholders, aligning everyone on the operational aspects of the system.
* **Identifying Business Impacts:** Stakeholders can understand how different decisions or system components impact the flow and output, helping them make informed decisions about system changes.
* **Requirement Clarification:** Helps stakeholders provide feedback on whether the system meets their needs and aligns with their business processes.



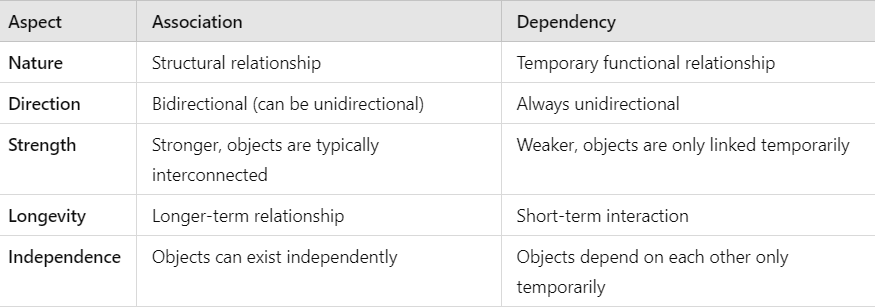
d) **Define the states of a system. Show the State Diagram for the Control Panel Class.**3

A **system state** refers to the condition or status of a system at a specific point in time.

* **State Transitions** occur when an event triggers a change from one state to another. These transitions are governed by predefined rules and conditions.
* **Events** or **inputs** drive the system to move from one state to another.
* **Actions** can occur during transitions or as a result of being in a particular state.



e)Differentiate between association and dependency. Define package analysis with example.2



**Package Analysis** involves organizing and grouping related classes, interfaces, or components into logical units called **packages**. Packages help manage the complexity of large systems by dividing them into smaller, manageable parts~~. A~~ **~~package~~** ~~in UML can contain classes, sub-packages, interfaces, or even other packages, making it easier to handle large-scale system design~~.

Consider a **Banking System** with different functional areas such as **Accounts**, **Transactions**, and **Users**.



**3.**

**a) Suppose you are open a new startup business agency. Now you want to develop a customized ERP solution for your business venture. So how to write a software requirement specification as per analysis of your business solution?3**

**b) Write down the advantages of domain analysis in software engineering. Define data modeling.3**

**Domain Analysis** is the process of studying and understanding a specific domain (a particular area of interest, industry, or business) to identify the common structures, functions, and rules that can be reused across different projects within that domain. The goal is to create a reusable foundation of knowledge and components for developing systems in a particular field.

**Advantages:**

1. **Reusability:** By identifying common patterns and features across multiple projects, domain analysis helps in creating reusable components and models, which speeds up development for future projects.
2. **Consistency:** It ensures that systems developed for the same domain follow consistent design principles, ensuring uniformity across projects.
3. **Improved Communication:** Domain analysis provides a shared vocabulary and understanding between developers, stakeholders, and domain experts, leading to better communication and fewer misunderstandings.
4. **Reduced Development Time and Cost:** Since domain knowledge is reused, the effort and time spent on analysis and design for new projects within the same domain are reduced, leading to lower costs.
5. **Enhanced Quality:** Reusing proven domain-specific components and architectures reduces the risk of errors and increases the overall quality of the software.
6. **Better Requirements Understanding:** Domain analysis helps in identifying the core requirements of a system within a domain, making it easier to meet user needs and avoid missing essential features.

**Definition of Data Modeling :**

**Data Modeling** is the process of creating a visual representation of how data is organized, stored, and related within a system.

**Key Components of Data Modeling:**

1. **Entities:** These are the objects about which data is collected (e.g., a **Customer**, **Order**, or **Product**).
2. **Attributes:** These are the characteristics of an entity (e.g., **Customer Name**, **Order Date**, **Product Price**).
3. **Relationships:** The associations between entities (e.g., a **Customer** places an **Order**).

**Example:**

A simple data model for an e-commerce system might include entities like **Customer**, **Order**, and **Product**, where:

* A **Customer** can place multiple **Orders**.
* Each **Order** contains multiple **Products**.

**c) What are data objects and data attributes in software engineering? Show the ERD notation with an example.3**

A **data object** represents a collection of information that is **treated as a single unit** by a software system. It could be anything that holds data and has attributes that describe it. Data objects are typically used to model real-world entities or concepts in a system.

**Examples of Data Objects:**

* In a **Banking System**, a data object could be a **Customer**, **Account**, or **Transaction**.
* In an **E-Commerce System**, data objects could include **Product**, **Order**, or **Shopping Cart**.

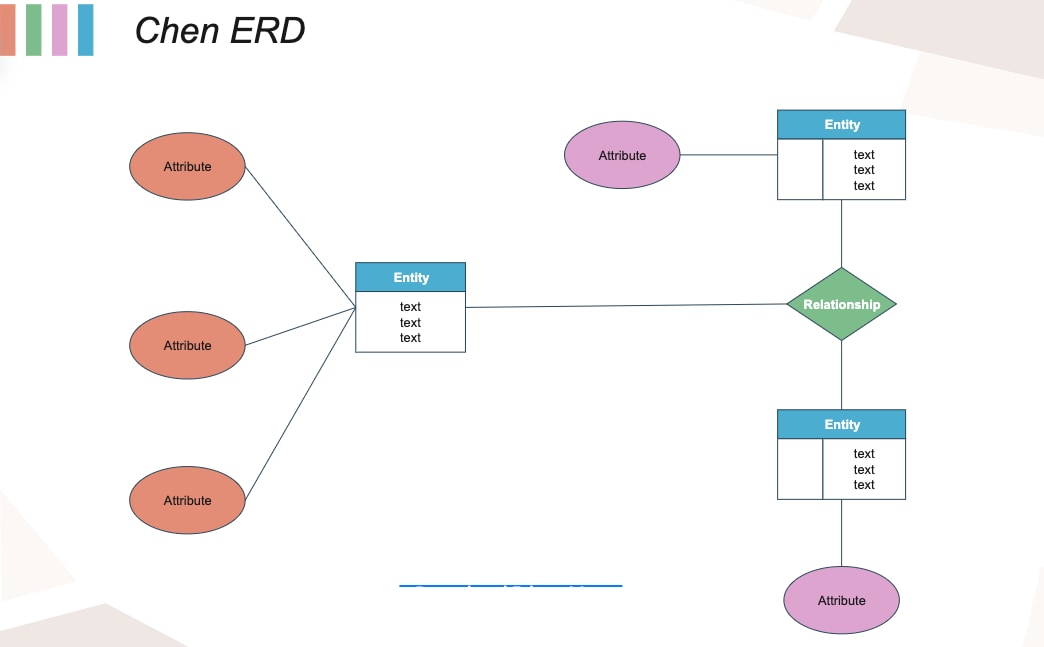
**Data Attributes in Software Engineering:**

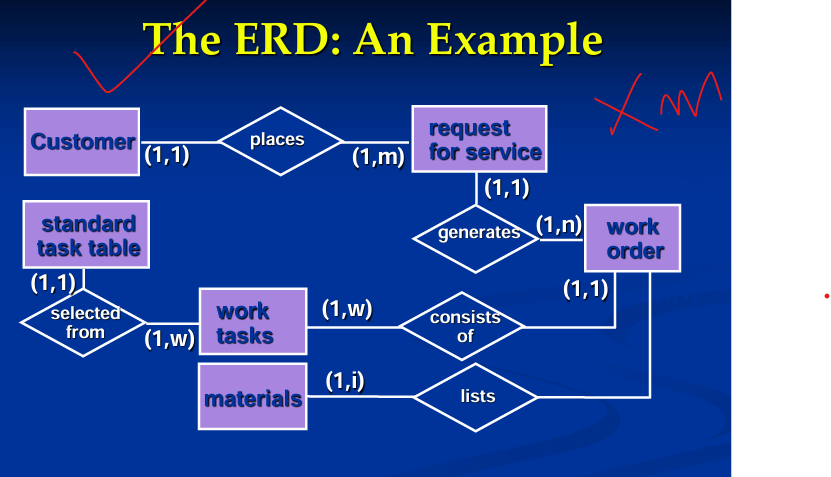
A **data attribute** is a property or characteristic that describes a data object. Each data object consists of several attributes, which define the state or values of that object. Attributes can be simple (like a name or age) or complex (like a collection of items).

**Characteristics of Data Attributes: (NTK)**

* **Descriptive Information:** Attributes provide specific details about a data object. For example, the **Customer** object might have attributes such as **Name**, **Email**, or **Address**.
* **Defines Object Properties:** They help define the characteristics and behavior of the data object.
* **Can Be of Different Types:** Attributes can have different data types such as string, integer, date, or boolean.

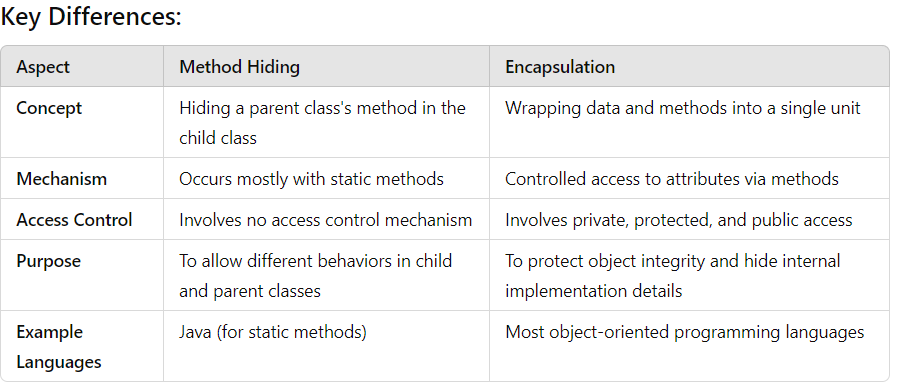
**Entity Relationship Diagram:**



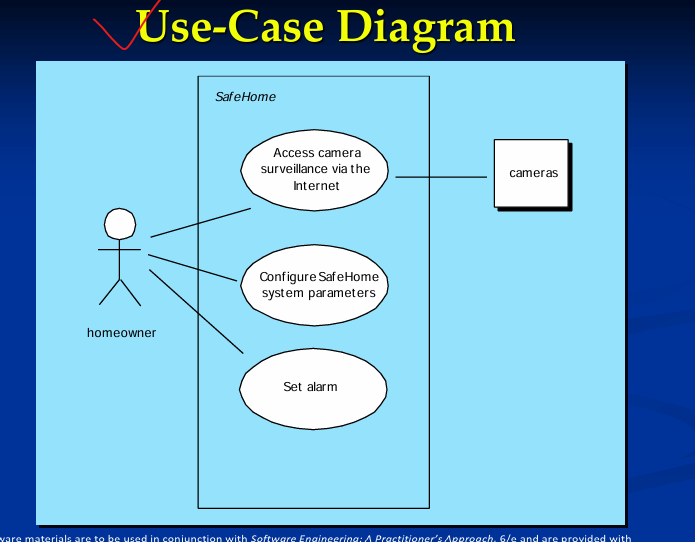
****

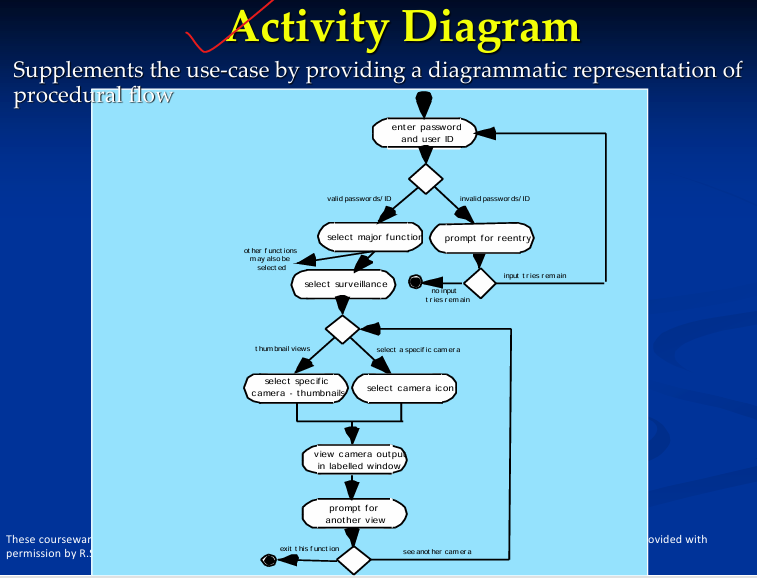
**d) Define class in software engineering. Difference between method hiding and encapsulation.2**

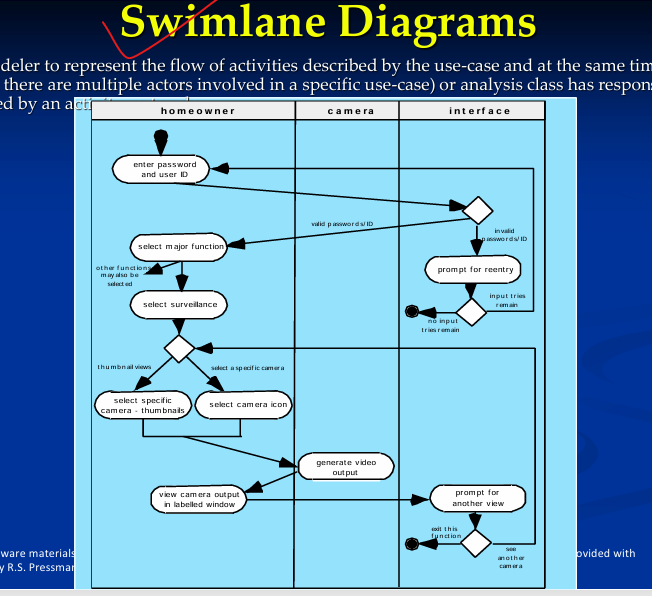
In software engineering, a **class** is a blueprint or template for creating objects. It defines a set of attributes (data) and methods (functions) that an object created from the class will have.

****

**e) Show the use case diagram, activity diagram and swim lane diagrams in software engineering.3**

****

****

****

**2.**

**a) What is software scope in software engineering? List out the some key steps to understand a project scope statement.3**

**Software scope** refers to the **detailed description** of all the functionalities, features, and deliverables of a software project. It defines the boundaries of the project, specifying what will be included in the final product and what will be excluded.

◼ Understand the customers needs .

◼ understand the business context .

◼ understand the project boundaries .

◼ understand the customer’s motivation .

◼ understand the likely paths for change.

**b) Show the tasks required for project scheduling and planning in software engineering.2**

For scheduling a project, it is necessary to -

* Break down the project tasks into smaller, manageable form
* Find out various tasks and correlate them
* Estimate time frame required for each task
* Divide time into work-units
* Assign adequate number of work-units for each task

Calculate total time required for the project from start to finish

**c) Define project estimation in software engineering? Give an example of conventional LOC based estimation.3**

**Project Estimation**

For an effective management accurate estimation of various measures is a must. With correct estimation managers can manage and control the project more efficiently and effectively.

Project estimation may involve the following:

* **Software size estimation**

Software size may be estimated either in terms of KLOC (Kilo Line of Code) or by calculating number of function points in the software. Lines of code depend upon coding practices and Function points vary according to the user or software requirement.

* **Effort estimation**

The managers estimate efforts in terms of personnel requirement and man-hour required to produce the software. For effort estimation software size should be known. This can either be derived by managers’ experience, organization’s historical data or software size can be converted into efforts by using some standard formulae.

* **Time estimation**

Once size and efforts are estimated, the time required to produce the software can be estimated. Efforts required is segregated into sub categories as per the requirement specifications and interdependency of various components of software. Software tasks are divided into smaller tasks, activities or events by Work Breakthrough Structure (WBS). The tasks are scheduled on day-to-day basis or in calendar months.

The sum of time required to complete all tasks in hours or days is the total time invested to complete the project.

* **Cost estimation**

This might be considered as the most difficult of all because it depends on more elements than any of the previous ones. For estimating project cost, it is required to consider -

* + Size of software
  + Software quality
  + Hardware
  + Additional software or tools, licenses etc.
  + Skilled personnel with task-specific skills
  + Travel involved
  + Communication
  + Training and support

**LOC-based estimation** (Lines of Code estimation) is a traditional software estimation technique where the cost, effort, and time required to develop a system are estimated based on the number of lines of code (LOC) the project will have.

Here's an example of **conventional LOC-based estimation**:

**Example Scenario:**

Suppose you are developing a **Customer Management System (CMS)**. Based on prior experience and domain knowledge, you estimate that the project will require around 25,000 lines of code (LOC). You need to estimate the effort, time, and cost required to complete the project using the LOC-based estimation technique.

**d) How function point (FP) analysis is used in estimation of software project? Give proper example.2**

Function Point

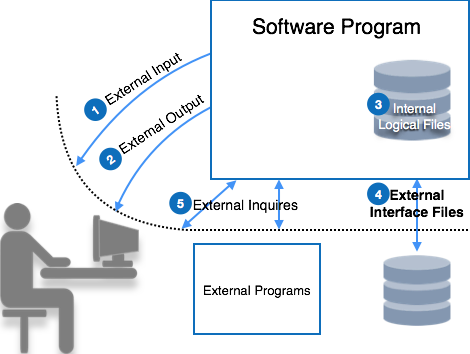
It is widely used to measure the size of software. Function Point concentrates on functionality provided by the system. Features and functionality of the system are used to measure the software complexity.

Function point counts on five parameters, named as External Input, External Output, Logical Internal Files, External Interface Files, and External Inquiry. To consider the complexity of software each parameter is further categorized as simple, average or complex.

External Input

Every unique input to the system, from outside, is considered as external input. Uniqueness of input is measured, as no two inputs should have same formats. These inputs can either be data or control parameters.

* **Simple** - if input count is low and affects less internal files
* **Complex** - if input count is high and affects more internal files
* **Average** - in-between simple and complex.



Let us see parameters of function point:

External Output

All output types provided by the system are counted in this category. Output is considered unique if their output format and/or processing are unique.

* **Simple** - if output count is low
* **Complex** - if output count is high
* **Average** - in between simple and complex.

Logical Internal Files

Every software system maintains internal files in order to maintain its functional information and to function properly. These files hold logical data of the system. This logical data may contain both functional data and control data.

* **Simple** - if number of record types are low
* **Complex** - if number of record types are high
* **Average** - in between simple and complex.

External Interface Files

Software system may need to share its files with some external software or it may need to pass the file for processing or as parameter to some function. All these files are counted as external interface files.

* **Simple** - if number of record types in shared file are low
* **Complex** - if number of record types in shared file are high
* **Average** - in between simple and complex.

External Inquiry

An inquiry is a combination of input and output, where user sends some data to inquire about as input and the system responds to the user with the output of inquiry processed. The complexity of a query is more than External Input and External Output. Query is said to be unique if its input and output are unique in terms of format and data.

* **Simple** - if query needs low processing and yields/give small amount of output data
* **Complex** - if query needs high processing and yields/give large amount of output data
* **Average** - in between simple and complex.

Each of these parameters in the system is given weightage according to their class and complexity. The table below mentions the weightage given to each parameter:

|  |  |  |  |
| --- | --- | --- | --- |
| **Parameter** | **Simple** | **Average** | **Complex** |
| Inputs | 3 | 4 | 6 |
| Outputs | 4 | 5 | 7 |
| Enquiry | 3 | 4 | 6 |
| Files | 7 | 10 | 15 |
| Interfaces | 5 | 7 | 10 |

The table above yields raw Function Points. These function points are adjusted according to the environment complexity. System is described using fourteen different characteristics:

* Data communications
* Distributed processing
* Performance objectives
* Operation configuration load
* Transaction rate
* Online data entry,
* End user efficiency
* Online update
* Complex processing logic
* Re-usability
* Installation ease
* Operational ease
* Multiple sites
* Desire to facilitate changes

These characteristics factors are then rated from 0 to 5, as mentioned below:

* No influence
* Incidental
* Moderate
* Average
* Significant
* Essential

All ratings are then summed up as N. The value of N ranges from 0 to 70 (14 types of characteristics x 5 types of ratings). It is used to calculate Complexity Adjustment Factors (CAF), using the following formulae:

CAF = 0.65 + 0.01N

Then,

Delivered Function Points (FP)= CAF x Raw FP

This FP can then be used in various metrics, such as:

**Cost** = $ / FP

**Quality** = Errors / FP

**Productivity** = FP / person-month

**e) Differentiate between estimation for OO projects and estimation for agile projects.4** Estimation methods for **Object-Oriented (OO) projects** and **Agile projects** differ due to the underlying principles and approaches of each software development methodology. Below are key distinctions between the two:

**1. Basis of Estimation**

**OO Projects:**

* **Components-Based Estimation**: Estimation in OO projects is typically done based on identifying and sizing the **classes**, **objects**, and **relationships** within the system. Components like methods, attributes, and inheritance hierarchies are considered.
* **Use Case and Function Point**: Common techniques include **Use Case Points (UCP)** and **Function Point (FP) Analysis**, where estimation revolves around functional requirements and the expected complexity of the objects.

**Agile Projects:**

* **Story-Based Estimation**: Agile projects are broken down into **user stories**, which are small, independent tasks or requirements that deliver value to the user.
* **Story Points**: Agile estimation is often done using **story points**, which measure the effort required to complete a user story based on factors such as complexity, risk, and size. These estimates are relative, not absolute.

**2. Granularity of Estimation**

**OO Projects:**

* **Coarse-Grained Estimation**: Estimations tend to focus on the overall system structure, such as how many classes or objects will be needed, how they interact, and the complexity of methods. The process can cover the entire project or large phases of the development cycle.

**Agile Projects:**

* **Fine-Grained Estimation**: Agile estimation is more iterative and incremental. Each sprint focuses on a small, specific set of user stories. Estimation happens frequently (before each sprint or iteration), with a focus on delivering small pieces of functionality in shorter timeframes.

**3. Estimation Techniques**

**OO Projects:**

* **Use Case Points (UCP)**: Estimation is based on analyzing use cases, with complexity ratings assigned to each use case and actor. It is a detailed technique used early in the software lifecycle.
* **Function Point Analysis (FP)**: Measures the size of the software by quantifying its functionality, such as external inputs, outputs, and internal data structures.

**Agile Projects:**

* **Planning Poker**: A collaborative estimation technique where team members assign story points to user stories, using cards with numbers based on the **Fibonacci sequence** (1, 2, 3, 5, 8, 13, etc.) to indicate effort.
* **T-Shirt Sizing**: Another estimation method where user stories are classified as small, medium, large, or extra-large to quickly gauge the effort required.
* **Velocity**: Agile teams estimate future sprints based on their velocity (how many story points they completed in previous sprints), ensuring more accurate, experience-based predictions.

**4. Time Estimation**

**OO Projects:**

* **Effort in Hours or Days**: The effort is typically estimated in **man-hours** or **man-days**, focusing on how long it will take to complete certain classes, methods, or object interactions.

**Agile Projects:**

* **Effort in Story Points**: Agile estimation does not directly use hours or days. Instead, it uses **story points** to represent relative effort. Once the team has an established velocity, it can predict how long a sprint will take based on the number of story points they can handle.

**5. Adjusting for Change**

**OO Projects:**

* **Less Frequent Updates**: Estimations in traditional OO projects tend to be more rigid and are typically updated less frequently. Once the project plan is set, changes to estimates happen only when significant changes in the requirements occur, which may lead to costly and time-consuming re-estimation.

**Agile Projects:**

* **Frequent Re-Estimation**: Agile estimation is flexible, and estimates are updated regularly, usually at the start of each sprint during the **backlog grooming** and **sprint planning** sessions. Agile projects are designed to handle changes more easily, with re-estimation happening as the project evolves.

**6. Focus of Estimation**

**OO Projects:**

* **Focus on Technical Design**: Estimation is focused more on the **technical aspects** of the system, such as the design of objects, relationships between classes, and inheritance hierarchies.

**Agile Projects:**

* **Focus on User Value**: Estimation focuses more on the **value delivered** to the user through the completion of specific user stories. The priority is on creating working software quickly and continuously improving it over time.

**7. Risk Consideration**

**OO Projects:**

* **Detailed Risk Analysis**: In OO projects, risk analysis is often performed upfront, and estimations typically account for risks related to system design, implementation, and technology choices.

**Agile Projects:**

* **Adaptive Risk Management**: Agile projects handle risk by breaking the project into short sprints. Each sprint allows for reassessment of risks and challenges. Story points are adjusted based on risk factors like complexity and uncertainty.

**8. Team Collaboration in Estimation**

**OO Projects:**

* **Centralized Estimation**: In many OO projects, estimation is performed by a small group of technical leads or architects, who review the requirements and plan based on their expertise.

**Agile Projects:**

* **Collaborative Estimation**: Estimation is a team activity. All team members, including developers, testers, and product owners, participate in estimation exercises like **Planning Poker**, ensuring a shared understanding of the complexity and effort.

**Summary of Differences:**

| **Aspect** | **OO Projects** | **Agile Projects** |
| --- | --- | --- |
| **Estimation Basis** | Classes, objects, methods, use cases | User stories, features |
| **Granularity** | Coarse-grained (system-wide) | Fine-grained (per sprint/user story) |
| **Estimation Techniques** | Use Case Points, Function Points | Story Points, Planning Poker, T-shirt sizing |
| **Effort Estimation** | Man-hours or man-days | Story points (relative effort) |
| **Re-Estimation** | Rare, usually at major changes | Frequent, at the start of each sprint |
| **Risk Consideration** | Upfront, based on detailed technical design | Adaptive, risk re-assessed in each sprint |
| **Collaboration** | Centralized, led by managers | Collaborative, involves the whole team |

**1.a)How do you define MOI Model? Define the five Common-Sense Approach to software projects. 3**

◼ ***The MOI Model***

◼Motivation. The ability to encourage (by “push or pull”) technical people to produce to their best ability.

◼Organization. The ability to invent new ones that will enable the initial concept to be translated into a final product.

◼ Ideas or innovation. The ability to encourage people to create and feel creative even when they must work within bounds established for a particular software product or application.

**5 Common-Sense Approach to Projects**

1. Start on the right foot. This is accomplished by working hard (very hard) to understand the problem that is to be solved and then setting realistic objectives and expectations.
2. Maintain momentum. The project manager must provide incentives to keep turnover of personnel to an absolute minimum, the team should emphasize quality in every task it performs, and senior management should do everything possible to stay out of the team’s way.
3. Track progress. For a software project, progress is tracked as work products (e.g., models, source code, sets of test cases) are produced and approved (using formal technical reviews) as part of a quality assurance activity.
4. Make smart decisions. In essence, the decisions of the project manager and the software team should be to “keep it simple.”
5. Conduct a postmortem analysis. Establish a consistent mechanism for extracting lessons learned for each project.

**b) What are the four organizational paradigms for software engineering teams?3**

* closed paradigm authority — The leader has complete control.
* random paradigm — The team members work independently.
* open paradigm — (A mix of close and random paradigm (control and independence.) attempts to structure a team in a manner that achieves some of the controls associated with the closed paradigm but also much of the innovation that occurs when using the random paradigm
* synchronous paradigm — The team members work on different parts of the problem but don't communicate much.

**c) Why is teamwork important in software engineering? How can you be a good team member in software development?3**

Team Coordination & Communication

◼ Formal, impersonal approaches products include software engineering documents and work (including source code), technical memos, project milestones, schedules, and project control tools (Chapter 23), change requests and related documentation, error tracking reports, and repository data (see Chapter 26).

◼ Formal, interpersonal procedures focus on quality assurance activities (Chapter 25) applied to software engineering work products. These include status meetings and design and code inspections

◼ Informal, interpersonal procedures include . group meetings review for information dissemination and problem solving and “collocation of requirements and development staff.”

◼ Electronic communication encompasses electronic mail, electronic bulletin boards, and by extension, video based conferencing systems.

◼ Interpersonal networking includes informal discussions with team members and those outside the project who may have experience or insight that can assist team

**d) List out the factors you must be considered when selecting a software project team structure.3**

**The following factors must be considered when selecting a software project team structure ...**

**1. Project Size and Complexity**

**2. Team Size**

**3. Project Methodology**

**4. Skills and Expertise Required**

**5. Geographical Distribution**

**6. Communication and Collaboration Needs**

**7. Project Duration**

**8. Decision-Making Process**

**9. Client or Stakeholder Involvement**

**10. Team Member Experience and Seniority**

**11. Leadership and Ownership**

**12. Budget Constraints**

**14. Risk and Complexity Management**

**15. Cultural Fit and Team Dynamics**

**e) What are the four P's of an effective software project management? Who are stakeholders in software engineering?2**

The 4 P’s

◼ People—the most important element of a successful project

◼ Product—the software to be built

◼ Process—the set of framework activities and software engineering tasks to get the job done

◼ Project—all work required to make the product a reality

Stakeholders

◼ **Senior managers** who define the business issues that often have significant influence on the project.

◼ **Project (technical) managers** who must plan, motivate, organize, and control the practitioners who do software work.

◼ **Practitioners** who deliver the technical skills needed to build a product. **Developers** (e.g., software engineers, programmers),**Testers/QA Engineers** (ensuring software quality),**Designers** (e.g., UI/UX designers for user interfaces)

◼ **Customers** who specify the requirements for the software to be engineered and other stakeholders who have a peripheral interest in the outcome.

◼ **End-users** who interact with the software once it is released for production use.