* Software Engineering is a field of study and practice that deals with the systematic design, development, maintenance and documentation of software systems.

It encompasses various principles, methods and tools to ensure the effective and efficient creation of high-quality software products.

* Object-Oriented Software Engineering (OOSE) is a software development methodology that applies the principles of object-oriented programming (OOP) to the entire software development process.

OOSE focuses on creating modular, flexible and maintainable software systems by organizing code into objects and classes, which encapsulate data and behavior.

This approach promotes the reusability and scalability of software components.

* Key Concepts in object-oriented Software Engineering:-

1. Objects: Objects are instances of classes that represent real-coorld entities, concepts, or abstractions within the software system.

Each object contains data (attributes or properties) and behavior (methods or functions) that operate on that data.

2. Classes: Classes are blueprints or templates for creating objects. They define the structure and behavior of objects that belong to them. A class specifies what data the objects will have and what actions they can perform.

3. Encapsulation: Encapsulation is the process of hiding the internal details of an object and exposing only the necessary interfaces for interacting with it. This ensures that the object's state remains consistent and can only be modified through defined methods.

4. Inheritance: Inheritance is a mechanism that allows a class (called a subclass or derived class) to inherit properties and behavior from another class (called a superclass or base class). This promotes code reuse and enables the creation of hierarchies of related classes.

5. Polymorphism: Polymorphism allows objects to take on multiple forms and exhibit different behaviors based on the context. It enables the use of a single interface to represent different data types or classes, enhancing flexibility and extensibility.

6. Abstraction: Abstraction involves simplifying complex systems by breaking them down into manageable components. In OOSE, classes represent abstractions of real-world entities, and the details of how those classes work internally are hidden from other parts of the system.

* Benefits of OOSE:

a) Modularity: → dividing a large system into smaller, manageable modules (classes/ objects).

b) Reusability: → The use of classes and objects allows code reuse, as objects can be instantiated and reused in various parts of the application.

c) Maintainability: → The encapsulation of data and behavior in objects makes it easier to maintain and update code.

d) Scalability: → Object oriented systems are generally easier to scale because new classes and objects can be added without affecting the existing codebase.

e) Flexibility: → Polymorphism and inheritance provide flexibility, allowing developers to extend and modify existing functionality without altering the original code.

f) Team Collaboration: → Different team members can work on separate modules independently, leading to a more organized development process and efficient collaborat

* Software development life cycle models:
(Software process models)

A Software life cycle model (also termed process model) is a pictorial and diagrammatic representation of the software life cycle. A life cycle model represents all the methods required to make a software product transit through its life cycle stages.

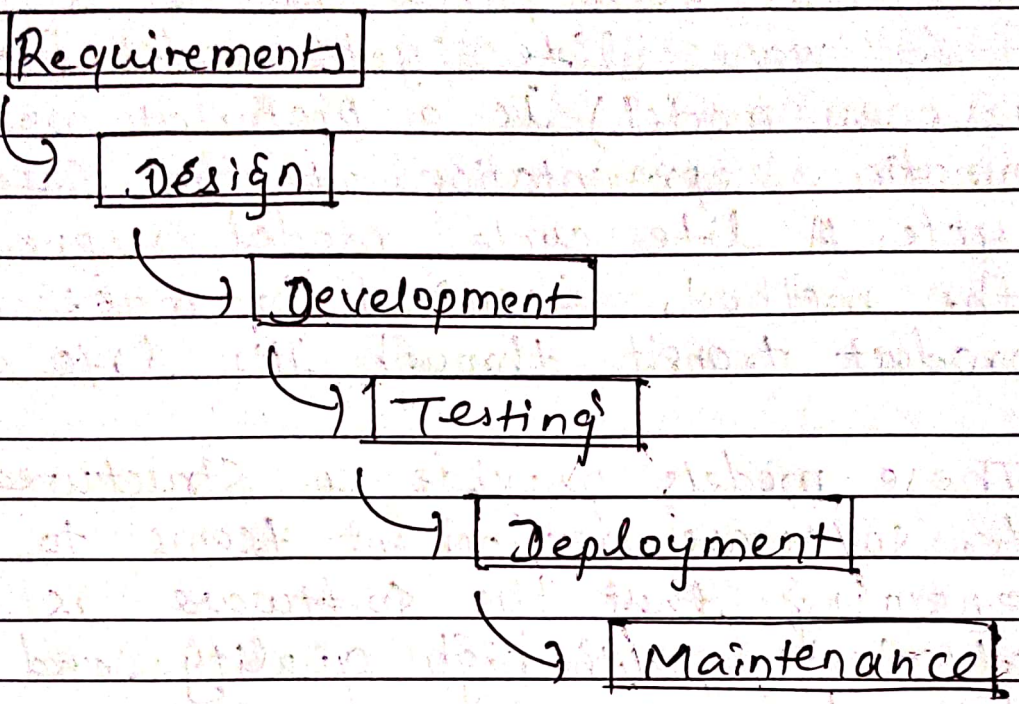
These models provide a structured framework for software development teams to follow, ensuring that the software is developed efficiently, with high quality, and meets the desired requirements.

Phases:

a) Requirement Analysis
b) Defining (SRS)
c) Designing
d) Coding
e) Testing
f) Deployment
g) Maintenance

- It acts as a guide to the project & meet clients objectives.
- It ensures correct and timely delivery to the client.
- It helps in evaluating, scheduling and estimating deliverables.

1. Waterfall Model :→



The Waterfall model is a linear and sequential approach to software development. It Consists of a series of phases (Requirements, Design, implementation, testing, deployment, and Maintenance) that must be Completed in a strict order.

Once a phase is completed, the development team team moves on to the next phase, and there is no turning back. This model is straight-forward and easy to manage but may lack flexibility for changing requirements.

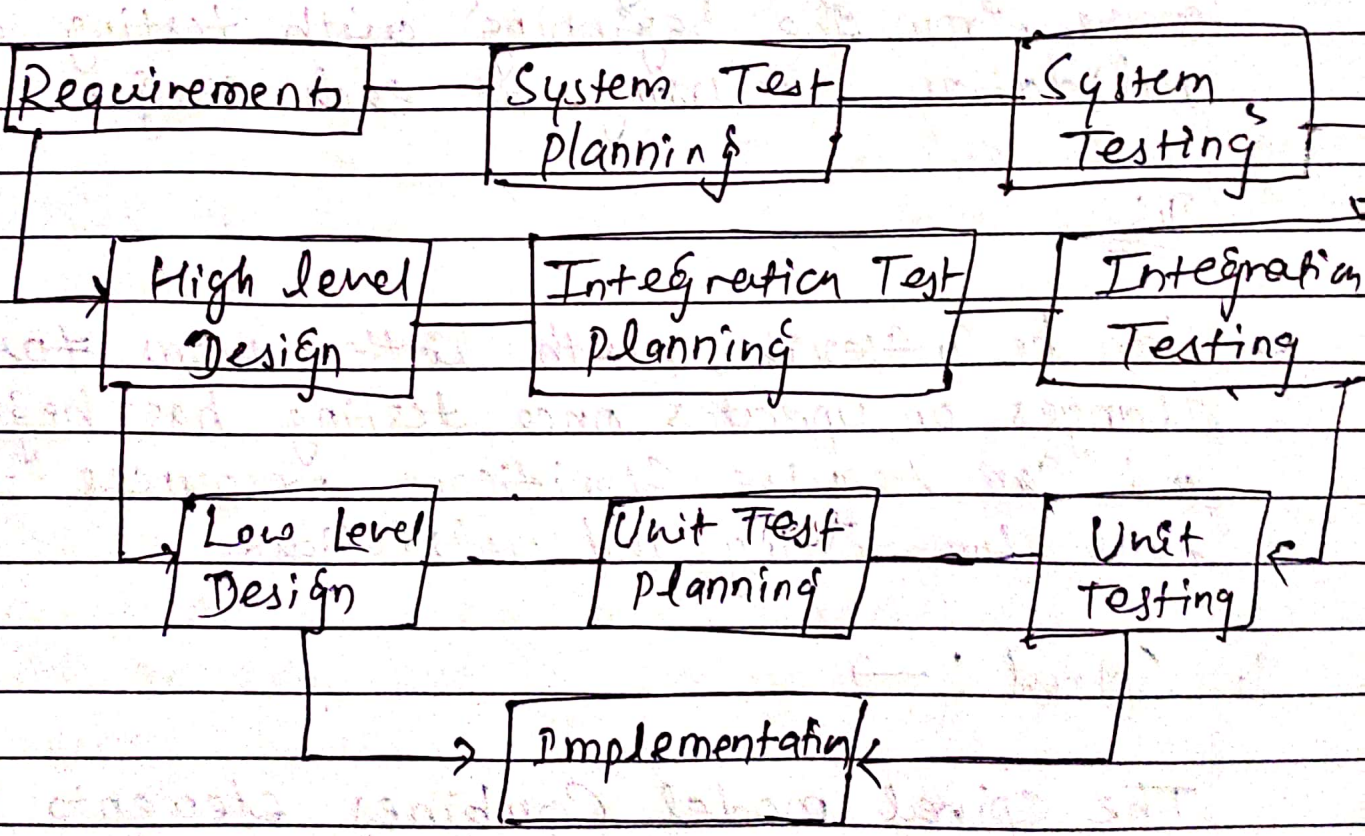
Adv

- Clear structure and defined milestones make it easier to plan & estimate.

Dis

- Little room for changes or updates once development has started. Can lead to delays and cost overruns.

Q. V-shaped model :→



V-Model is also known as Verification and Validation model. It is an extension of the Waterfall model that emphasizes the relationship between each development phase and its corresponding testing phase. It helps to ensure that the software is tested thoroughly and aligns testing activities with the development process.

Adv

- Ensures that quality is built into the software from the beginning with testing and validation occurring at each stage.

Dis

- Can be inflexible, with little room for changes or updates once testing has begun and can require significant resources and time for testing and validation.

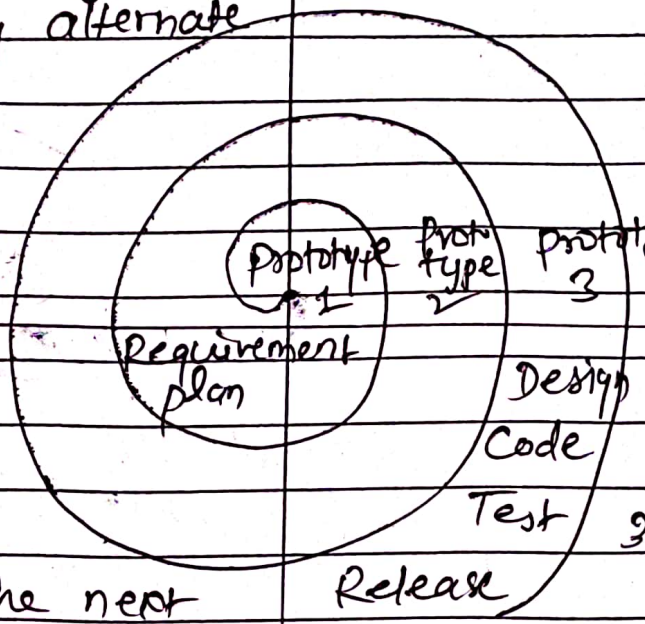
3. Spiral :→

The spiral model combines elements of the waterfall model and iterative development. It involves a series of iterative cycles, each including planning, risk analysis, engineering and testing.

The model emphasizes risk assessment and mitigation, making it suitable for large and Complex projects with changing requirements

1. Planning objective or identify alternate Solutions.

2. Risk analysis and resolving.



3. Develop the next level of product

4. Plan the next phase

Adv

- Flexible and adaptable approach that can incorporate changes and updates all needed, with an emphasis on mitigating risk and ensuring quality.

Dis

- Can be Complex and difficult to manage, with multiple iterations and stages that require Careful planning and Coordination.

* Requirement Analysis and Specification:

Requirement analysis and specification are crucial steps in the software development lifecycle. They involve understanding, capturing and documenting the needs and expectations of stakeholders to build a successful software solution.

Key activities involved in requirement analysis include:

a. Stakeholder Identification: Identify all the individuals, groups or entities that will interact with the software. This could include end-users, clients, managers, developers and other relevant parties.

b. Elicitation: Interact with stakeholders to collect their requirements through interviews, surveys, workshops or other communication methods.

c. Documentation: Record and document the gathered requirements in a clear and organized manner. This documentation may include use cases, user stories, functional & non-functional requirements, constraints and assumptions.

d. Analysis: Analyze the collected requirements to ensure they are feasible, consistent and complete. Identify any potential conflicts or ambiguities.

e. Prioritization: Determine the relative importance of each requirement to guide the development process.

Requirement specification is the process of documenting the analyzed requirements in a formal and structured manner. It involves translating the gathered information into clear and unambiguous specifications that can be understood by developers, designers and other team members.

The main components of requirement specification are:

a. Functional Requirements: Describe the specific functions and capabilities the software should provide. These are often described as use cases, user stories or detailed feature descriptions.

b. Non-functional Requirements: Specify the qualities and constraints of the software, such as performance, security, scalability,

usability and Compatibility.

c. User Interface (UI) Design: Provide visual representations, wireframes or mockups of the user interface to show how the software will look and feel.

d. Data Requirements: Define the data models and database specifications necessary to support the software's functionality.

e. System Architecture: Outline the high-level system architecture, including components, modules, and their interactions.

f. Testability and Validation: Define criteria for testing and validation to ensure that the software meets the specified requirements.

* Object-oriented Software engineering:

- OOSE is the first object-oriented design methodology that employs use cases in software design.

- The use case scenario begins with a user of the system initiating a sequence of interrelated events.

- Objectory is built around several different models:

(i) Use-Case model:

It defines outside (actors) and inside (use case) of the system's behavior.

(ii) Domain object model:

The objects of the real world are mapped into the domain object model.

(iii) Analysis object model:

It presents how the source code (implementation) should be carried out.

(iv) Implementation model:

It represents the implementation of the system.

(v) Test model:

It constitutes the test plans, specifications and reports.

It includes a requirements, an analysis, a design, an implementation and a testing model.

Interaction diagrams are similar to UML's sequence diagrams. State transition diagrams are like UML state chart diagrams

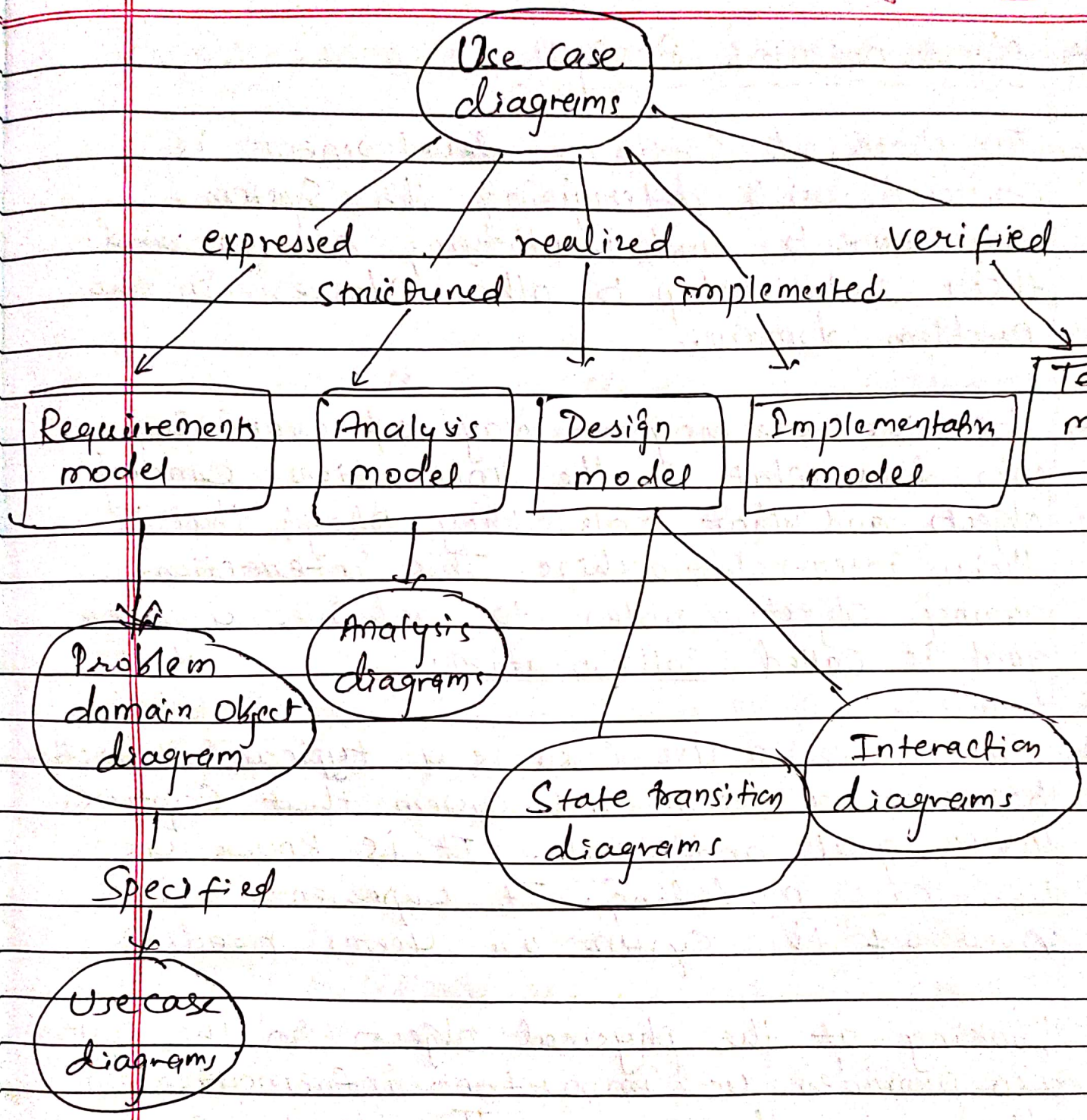


Fig: Object-Oriented Software engineering

* Object oriented Analysis:

- The phase of Software development is concerned with determining the System requirements and identifying classes and their relationship to other classes in the problem domain.

- Scenarios are a great way of examining who does what in the interactions among objects and what role they play; that is, their inter-relationships. This interaction among object's roles to achieve a given goal is called Collaboration.

- In essence, a use case is a typical interaction between a user and System that Captures user's goals and needs. It is known as use-case modeling. It represents user's view of the system or user's needs

- Looking at the physical objects in the system also provides us important information on objects in the systems. The objects could be individuals, organizations, machines, units of information, pictures, or whatever else makes sense in the Context of real world systems.

Object-Oriented Analysis: Unified Approach:-

- The goal is first to find domain of the problem and System's responsibilities by knowing all user needs which is accomplished by models.
- These models Concentrate in describing what the System does rather than how does it

OOA has the following steps:

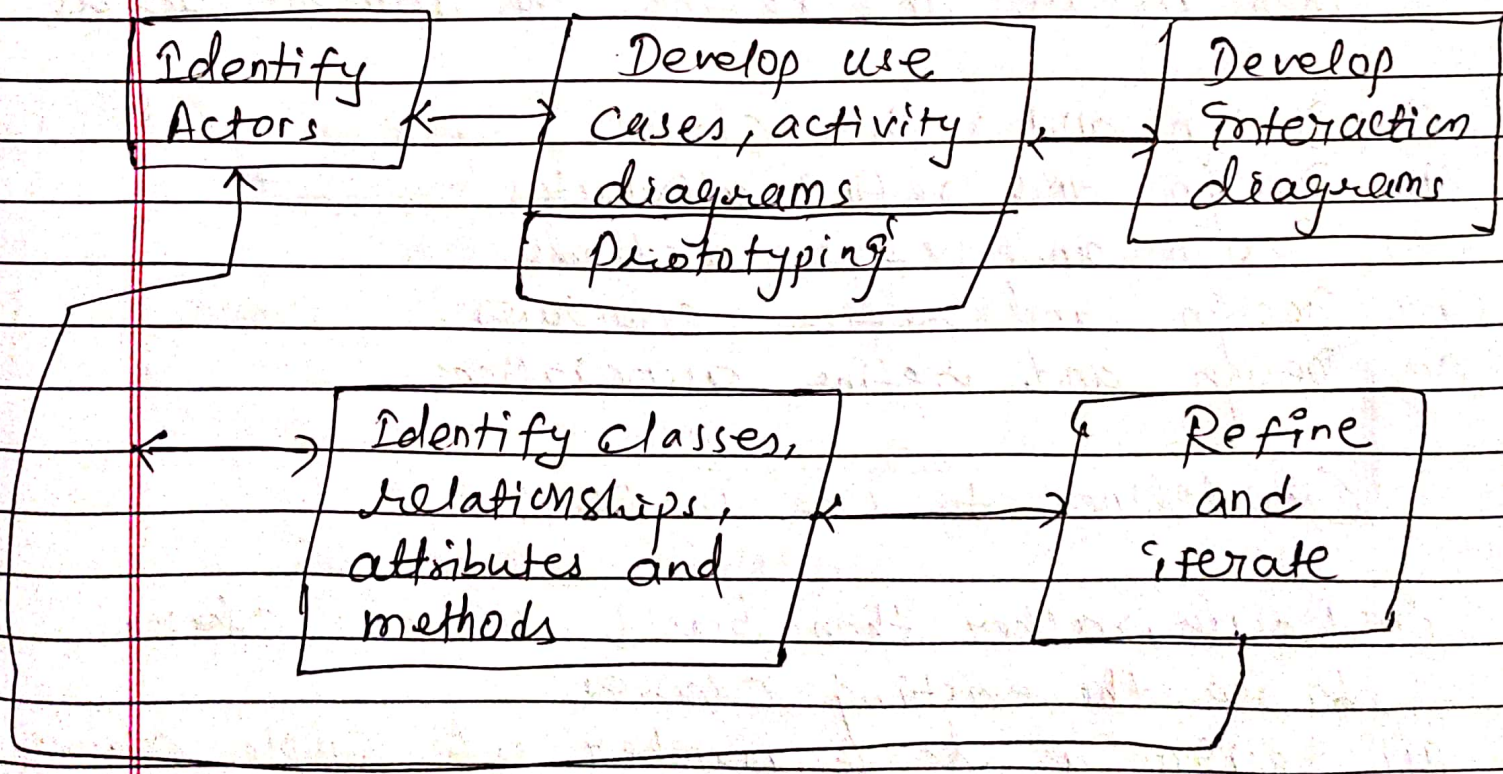


Fig: O-O-Analysis

* Object-oriented Design (O-O program design):

- The goal of OOD is to design the classes identified during the analysis phase and the user interface.

- Here, we identify and define additional objects, classes that support implementation of requirements.

- Here, we first, build the object model based on objects and their relationships, then iterate and refine the model such as,

(i) Design and refine classes
(ii) Design and refine attributes
(iii) Design and refine methods
(iv) Design and refine structures
(v) Design and refine associations

Guidelines to use in OOD:

(i) Reuse, rather than build, a new class. Know the existing classes
(ii) Design a large number of simple classes, rather than a small number of complex classes.
(iii) Design methods

(iv) Critique what you have proposed. If possible, go back and refine the classes.

Object oriented Design: Unified Approach:-

OOD has the following steps:

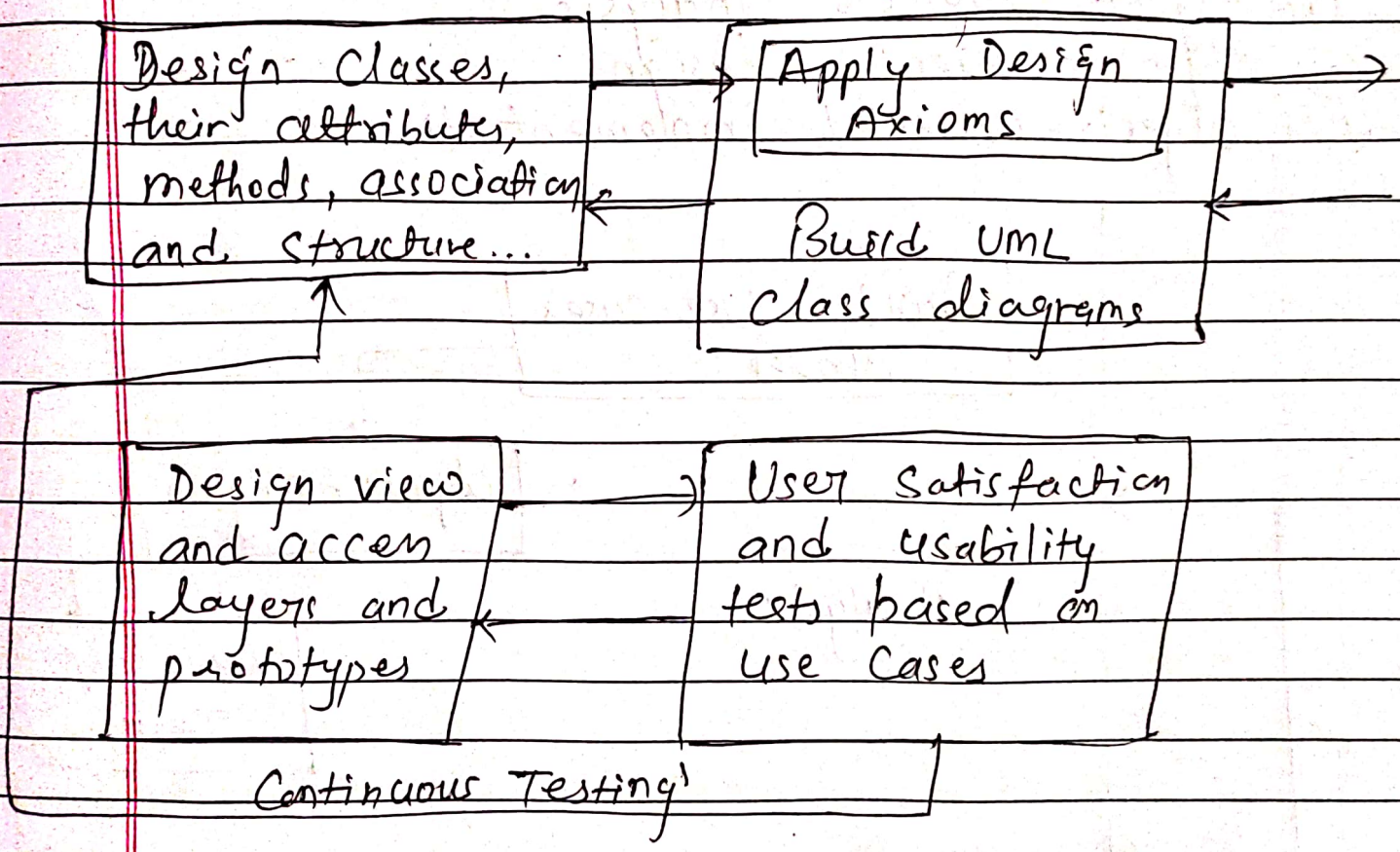
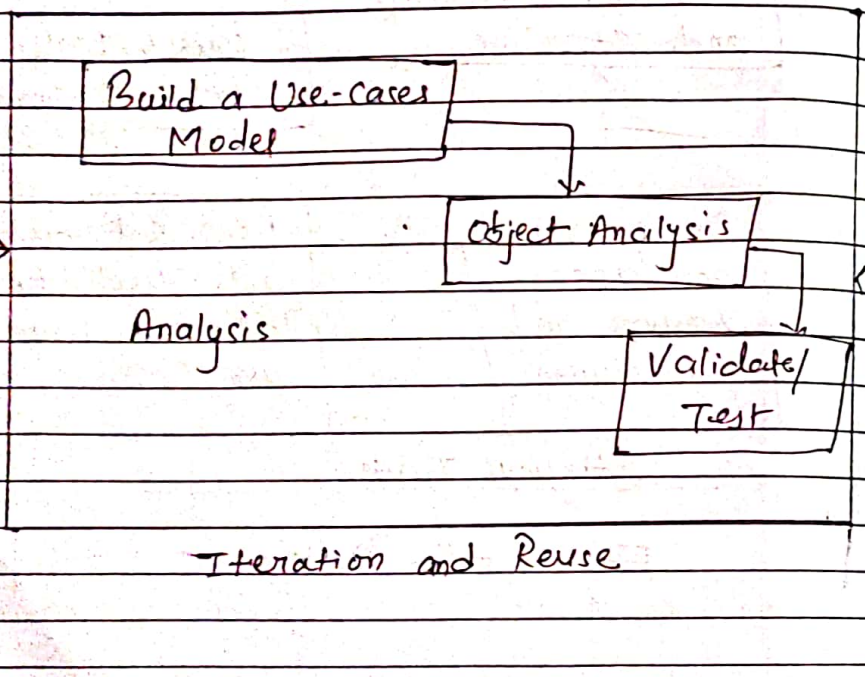


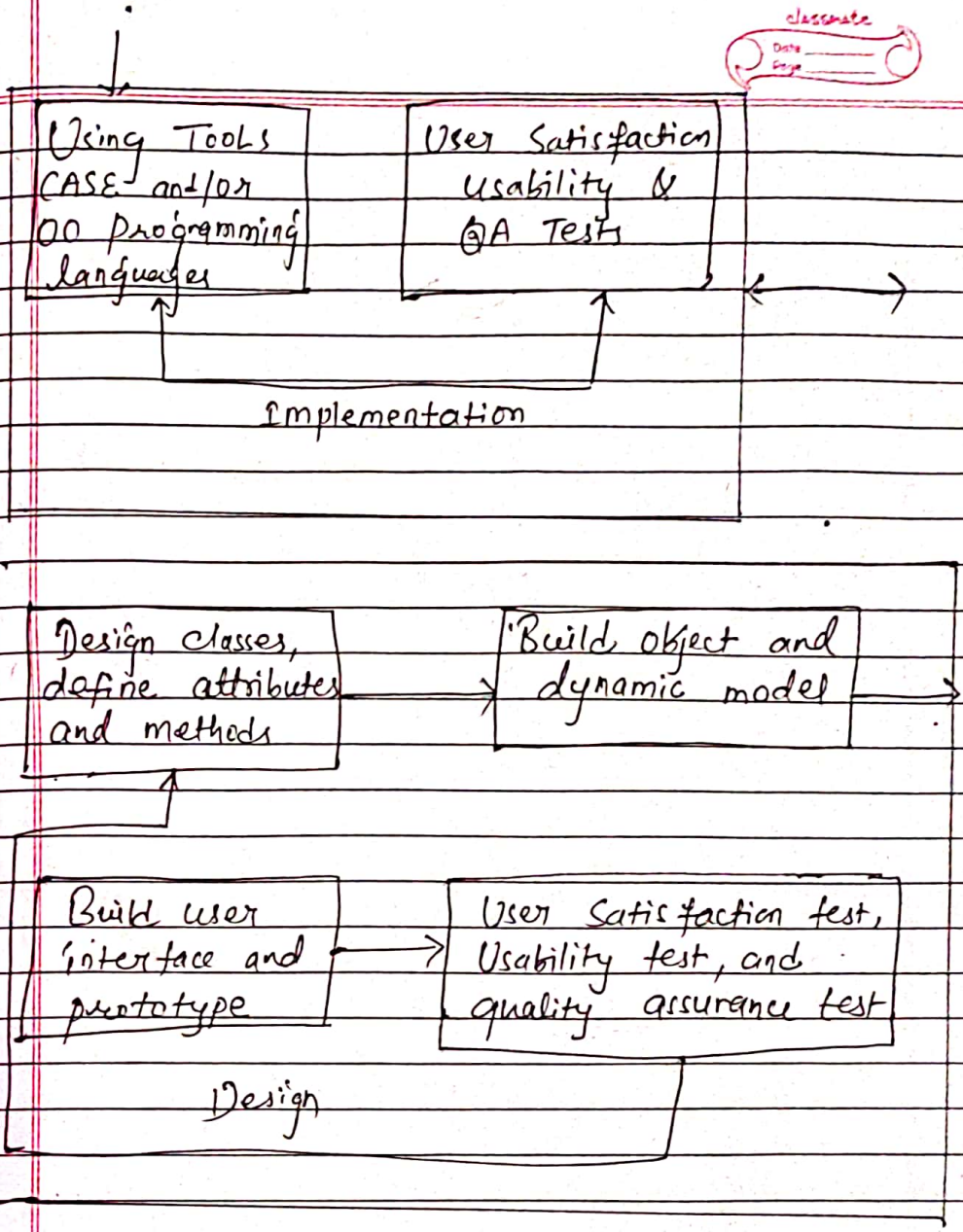
Fig: O-O- Design

* Object Oriented Software development:-

The object oriented Software development life cycle (SDLC) Consists of three macro processes:

(i) Object-oriented analysis
(ii) Object-oriented design
(iii) Object-oriented implementation





- Produce designs are traceable across requirements, analysis, design, implementation and testing and it can be traced back directly to user requirements.

*

Function/Data Methods	object oriented method
• In function data methods function is active and have behavior, and data is passive information holder which is affected by the function.	• In object oriented method, function and data is Combined as integrated objects.
• The system is typically broken down into functions whereas data is sent between those functions.	• The system is broken down into objects, function and data will be method and properties of that object.
• The system developed using function/data method often becomes difficult to maintain.	• This method is easier to understand and thus easier to maintain than the result of function/data method.
• In function/data method approach all function must know how the data must be stored, i.e. data structure	• Each object should be defined internally, which includes defining the information that each object must hold.

• System generated using this methods often quite unstable, a slight modification will generate major Consequences.	• Items with low modification probability are naturally identified and it is possible to isolate at an early stage.
• The requirement specification is normally formulated in normal human language not in function/data structure, which creates large semantic gap between the external and internal views of the system.	• object is a naturally occurring entities in the application domain. So this method will model the application domain more efficiently and reduces the semantic gap between them.
•	•

* Requirement Models:

- Requirement models defines the system boundaries and functionality that should offer.

- It functions as Contact between developer and the ordered of the system, specially from developers view of what customer wants.

- This model govern the development of all other model so this model is central one throughout the whole system.

- The requirement model will be structured by analysis model, realized by design model, implemented by implementation model and tested in testing model.

The requirements model consists of three parts:

(i) Use case model:

- A use case is a specific way of using the system by performing some part of functionalities.

- A use case constitutes complete course of event initiate by the actor and it specifies the interaction between the actor and the system.

- The collected use case specify all the existing way of using the system.

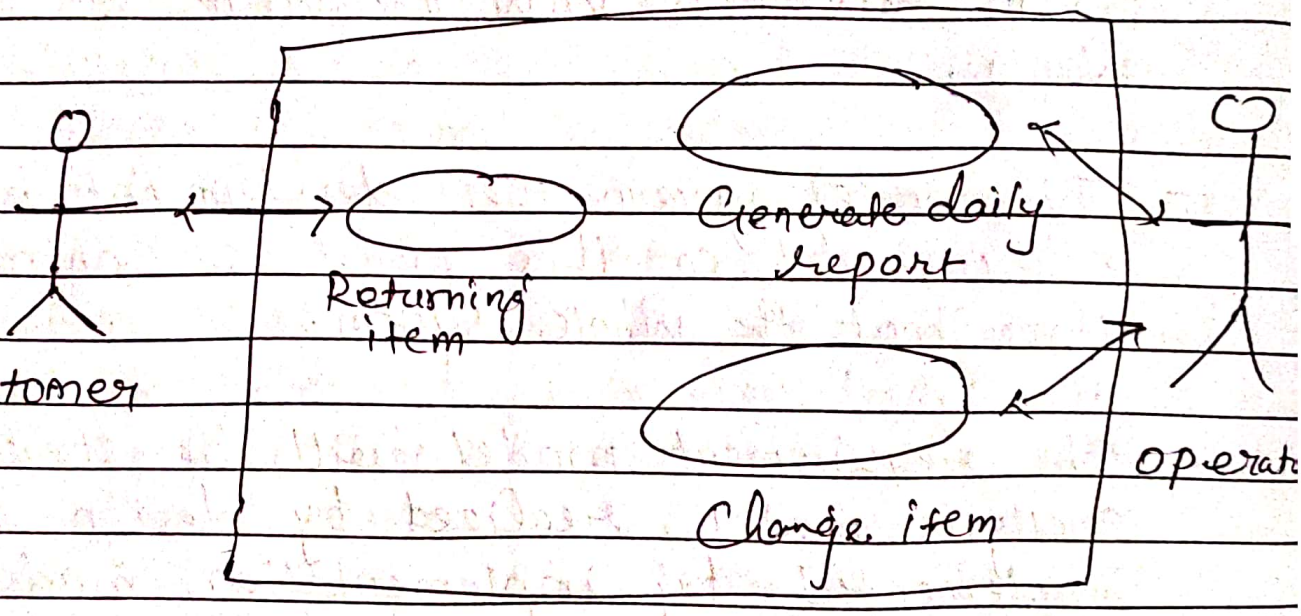


Fig: The first attempt of the use case model for the recycling machine.

(ii) Interface Description:

- We should define the interface in details while we describes the use cases and communicating to potential users.

- We can simulate the use cases with sketches of what user will see in the screen and for sophisticated simulation use UIMS (user interface management system)

- This model guarentee that the user interface will be consistent with the user's logical system perspective.

- In the recycling machine the user interface is quite trivial (being mainly a push-button machine)

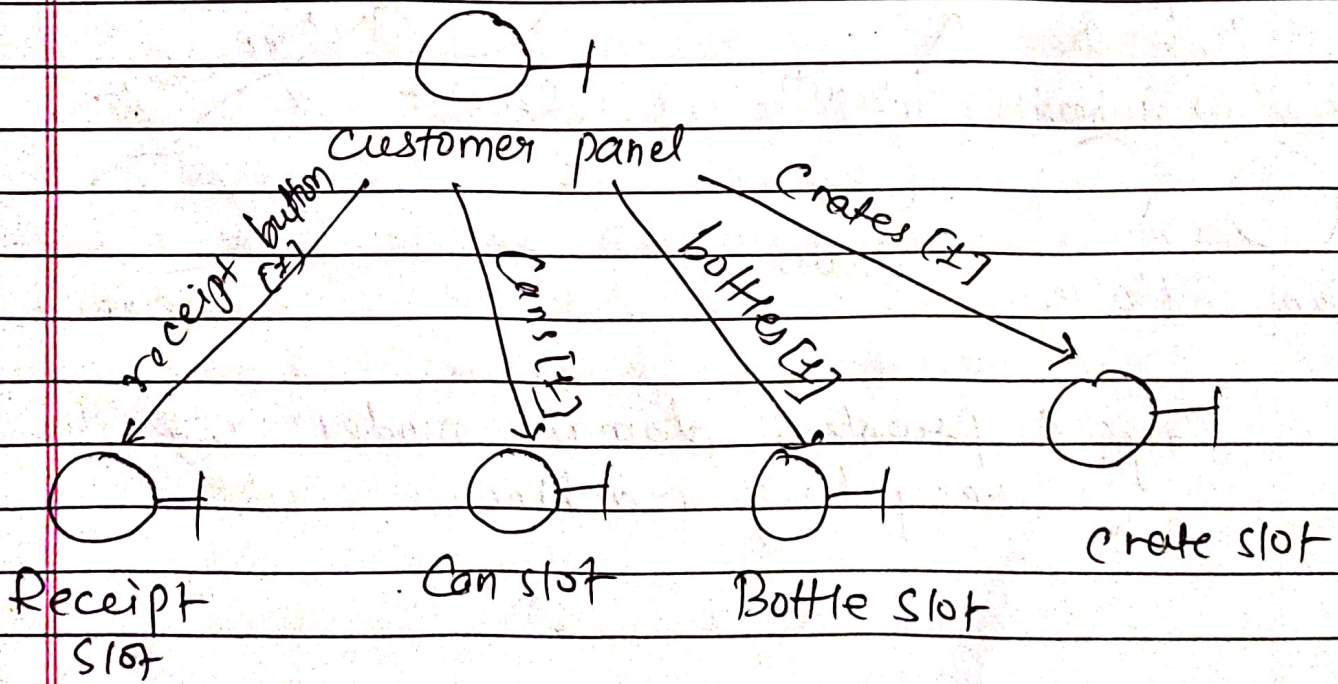


Fig: Example of interface design

(iii) Problem domain object:

- When requirements specification exists in very vague form, it is difficult to define task and boundary of a system.
- So, it is good to develop a logical view of the system using problem domain objects.

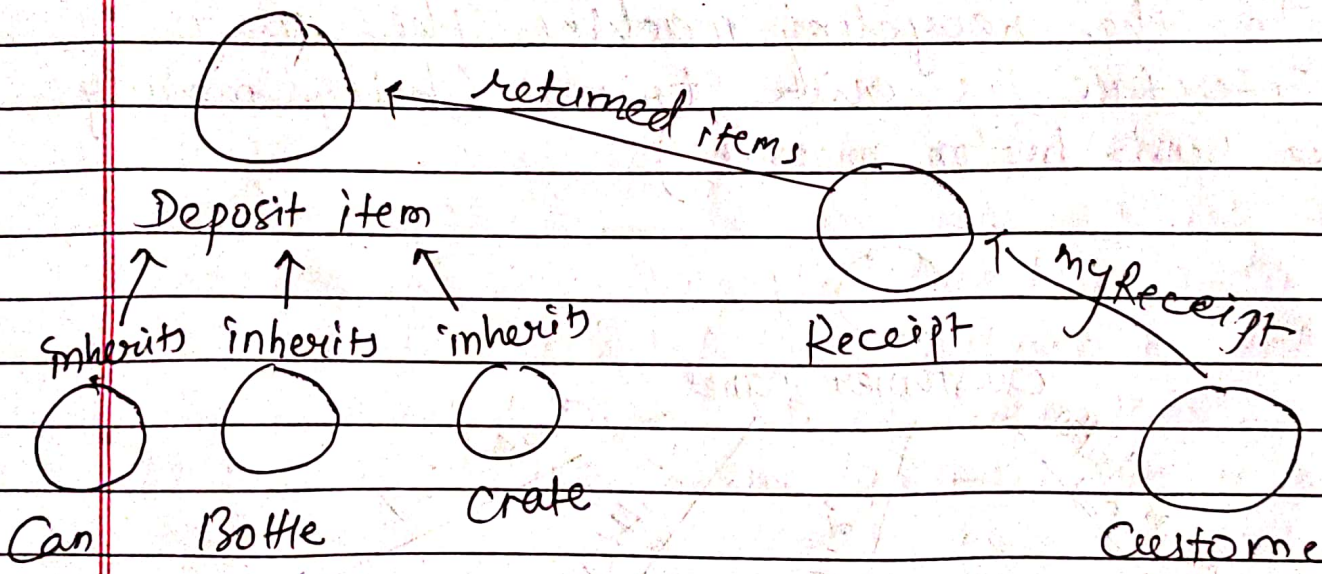


Fig: A problem domain model of the recycling machine.

* Construction phase:

There are three main reasons for having Construction phase.

(i) Analysis model is not sufficiently formal:

- To change the source code we must refine the object; Which operation should we offer, exactly what should the communication between different objects look like etc.

(ii) The actual System must be adopted to implementation environment:

- In analysis we assumed an ideal world for our system. Actually there is no ideal world and we must adopt to the environment in which system the system is to be implemented.

(iii) To validate the analysis results:

- In Construction we see the results of analysis.

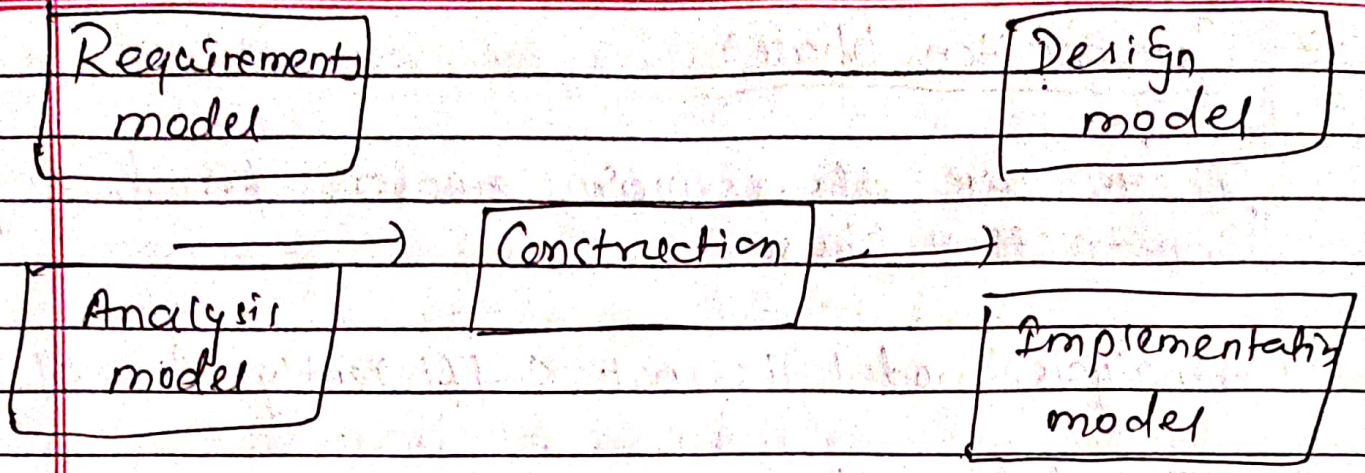


Fig: The input and output models of Construction

Thus, Construction model is further divided into two phases; design and implementation. The design model is a further refinement and formalization of the analysis model where consequences of implementation environment have been taken into account. The implementation model is actual implementation (Code) of the system.

* Key factors involved in managing OOSE:-

1. Project selection and preparation
2. Product development organization
3. Project organization and management
4. Project Staffing
5. Software quality assurance
6. Software metrics

1. Project Selection and preparation:

Selection:

- Select a real project that is important, but not with a tight time schedule or any other hard Constraint.
- Select a problem domain that is well known and well defined.
- Select people experienced in system development who have positive view of changes. The management should have confidence in them.
- Select a project manager with high degree of interest in the task.
- The staff should work full time in the projects and not be distracted by other projects.
- Base your work on a detailed plan developed in advance.

Preparation:

- All personnel involved in the new order of work need education and trainings.
- Give strict method procen definition, more emphasis can be put on formal education and training.
- A new development procen involves a lots of changes which brings potential risks.
  These risk can be managed by three simple steps:
(i) Risk identification
(ii) Risk valuation
(iii) Managing the risks.

2. Product development organization:

Product development is to develop different models in sequence.

- The first model is to be developed is requirement model.
- The next model is analysis procen which forms the well defined result procen model.
- Analysis model is input for the construction procen.
- The next is testing procen.

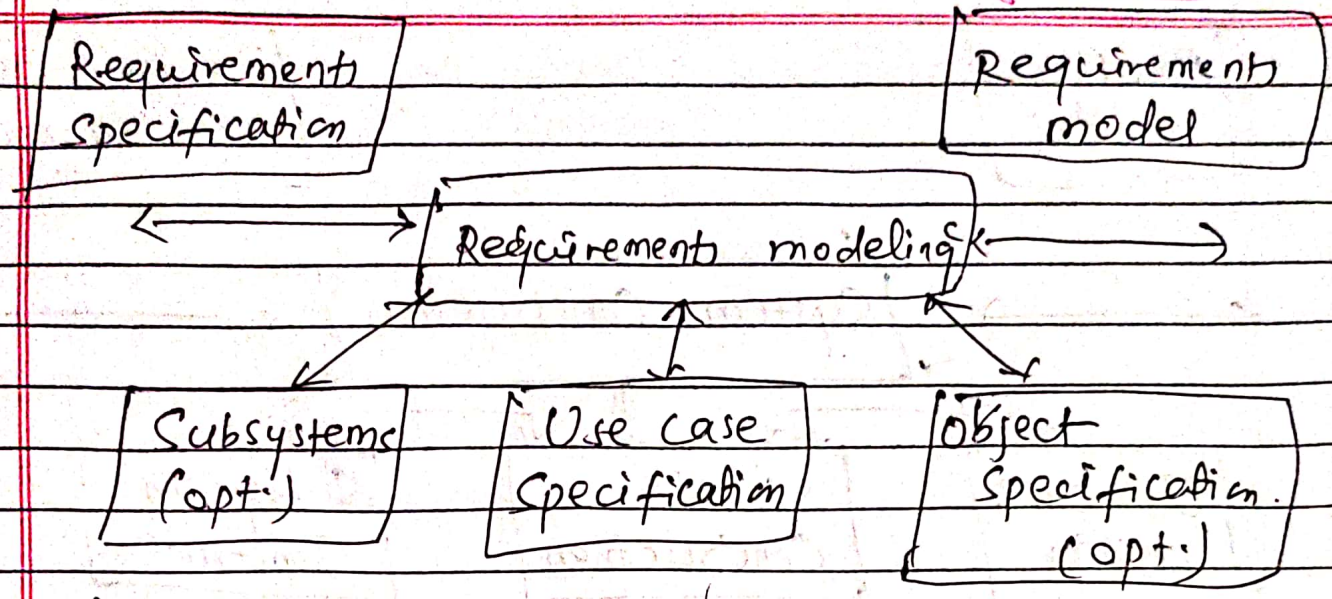


Fig. The procen of requirements analysis

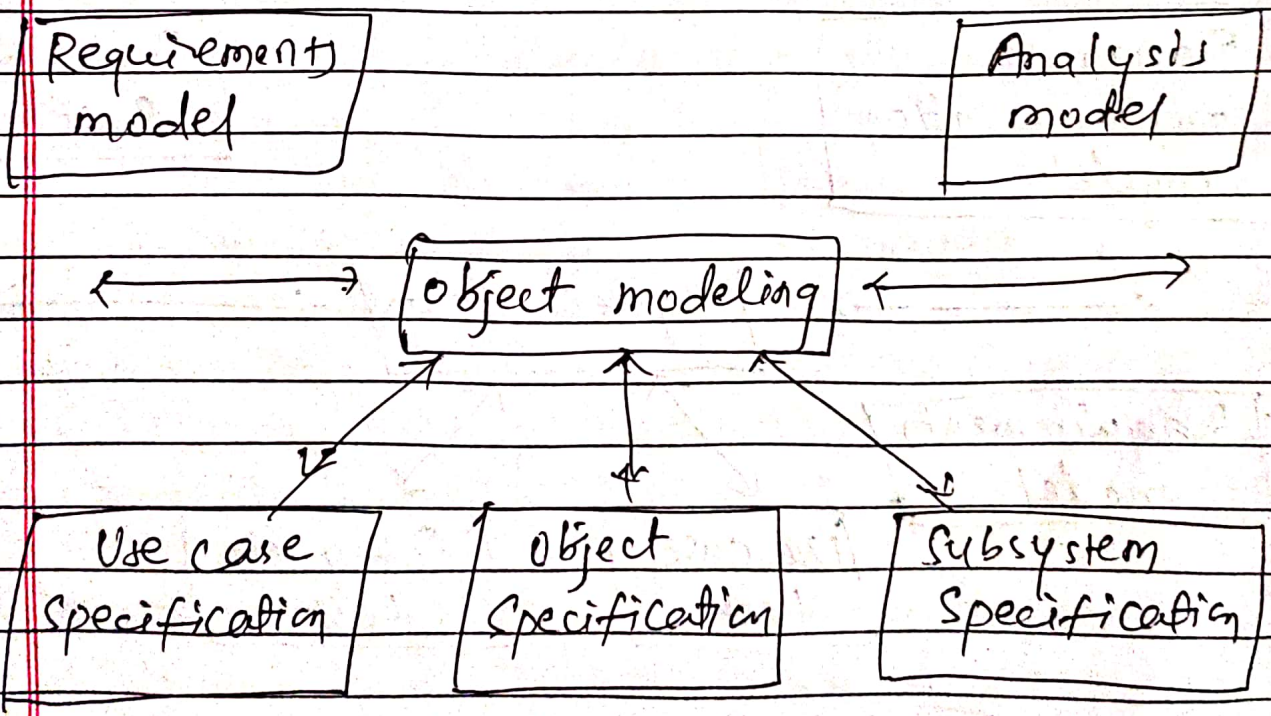


Fig: The procen of robustness analysis

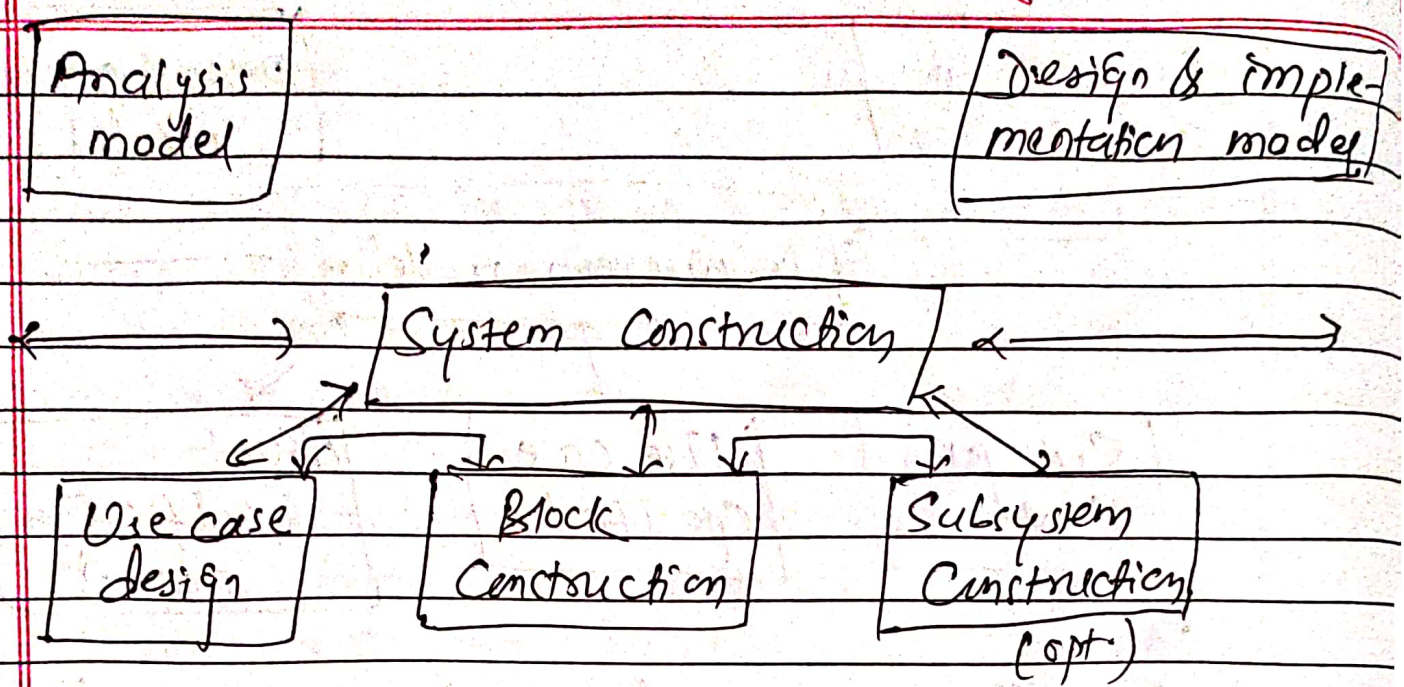


Fig: The process of Construction

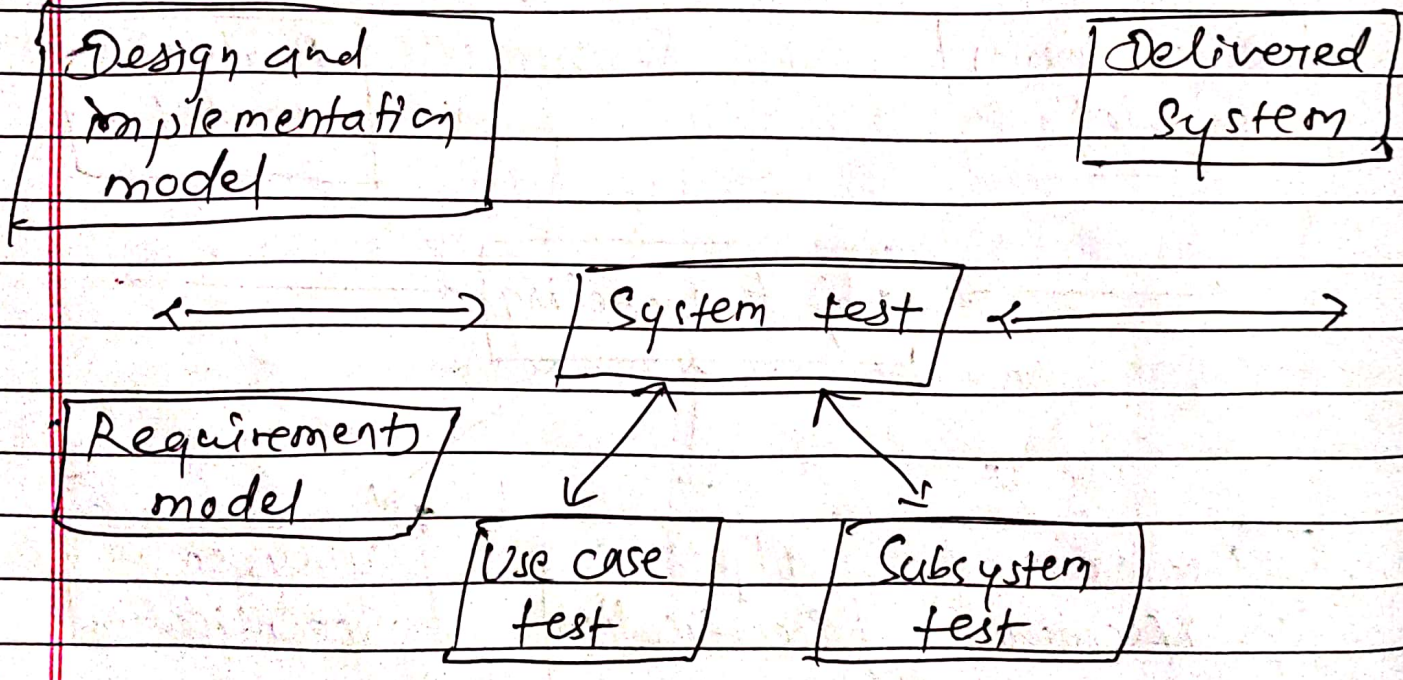


Fig: Testing process

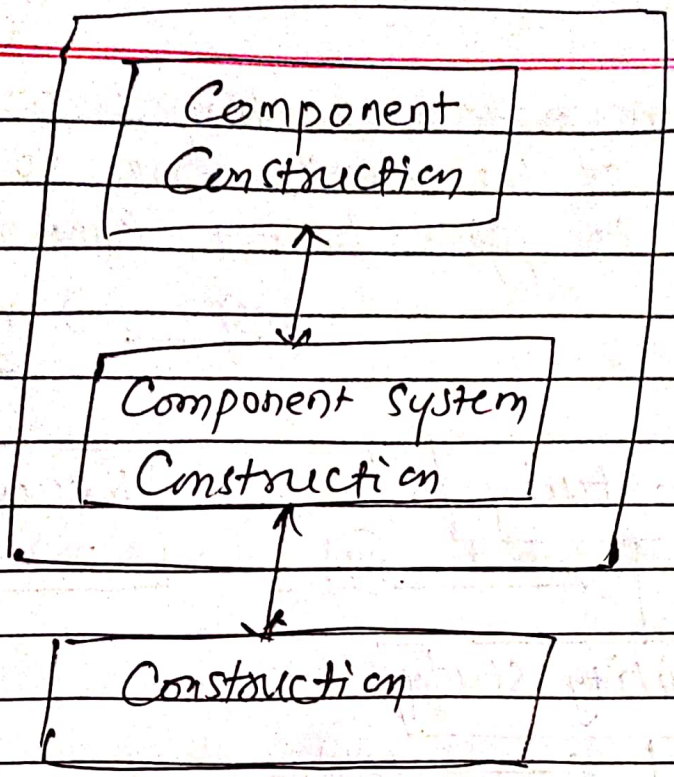


Fig: The Component process interacts with Several Construction process.

3- Project organization and management:

A necenary but not Sufficient, Condition for successful software development is good project management.

- A project is divided into numbers of milestones and the managerial and technical aspects must fit together to acheive this milestones.

- Milestones are concrete, objectively defined deliverables.

- Milestones are often combined with reviews with audits of the work done so far. This division aims to give better control of the projects.

(ms) → milestone

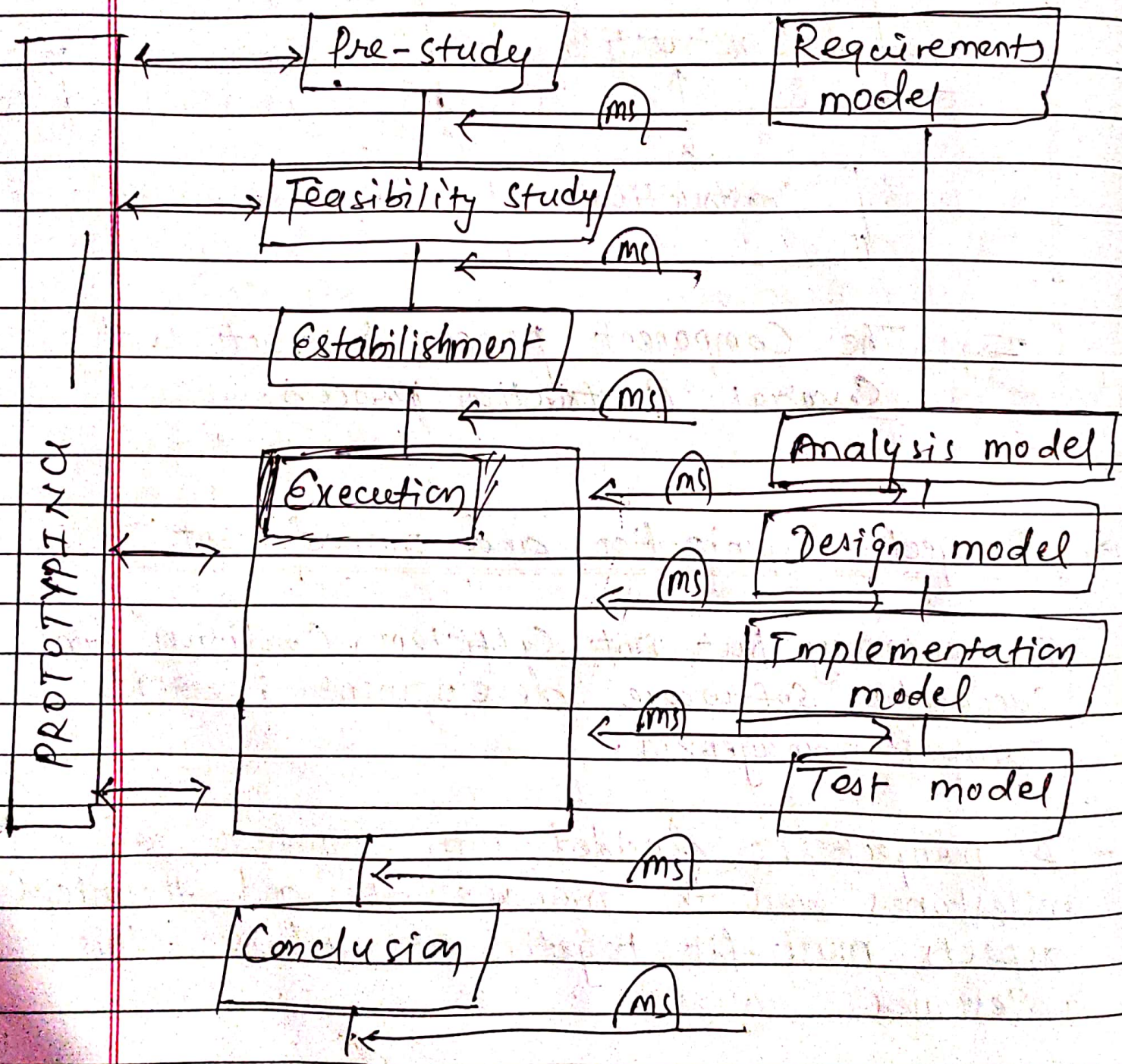


Fig: Management part of the project

Project management phases are :-

(i) Pre Study:
It studies about if the project is practicable or not.

(ii) Feasibility Study:
It studies different technical alternatives and their consequences.
Types: Technical, Economical & operational feasibility.

(iii) Establishment:
Detailed plans and resource plans are developed.

(iv) Execution:
The project is developed in accordance with the plans previously prepared.

(v) Conclusion:
The project is completed and proposals to improve the project & development methods are summerized.

4. Project staffing:

One of the difficulties in software development lies in the staffing problems.

A group of people with different knowledge and skills, which we call a software project team, work together to develop software.

Accordingly, the project team influences the outcome of software development. Therefore, project staffing that is, how to form software project teams, has persistently been a key question of software organizations.

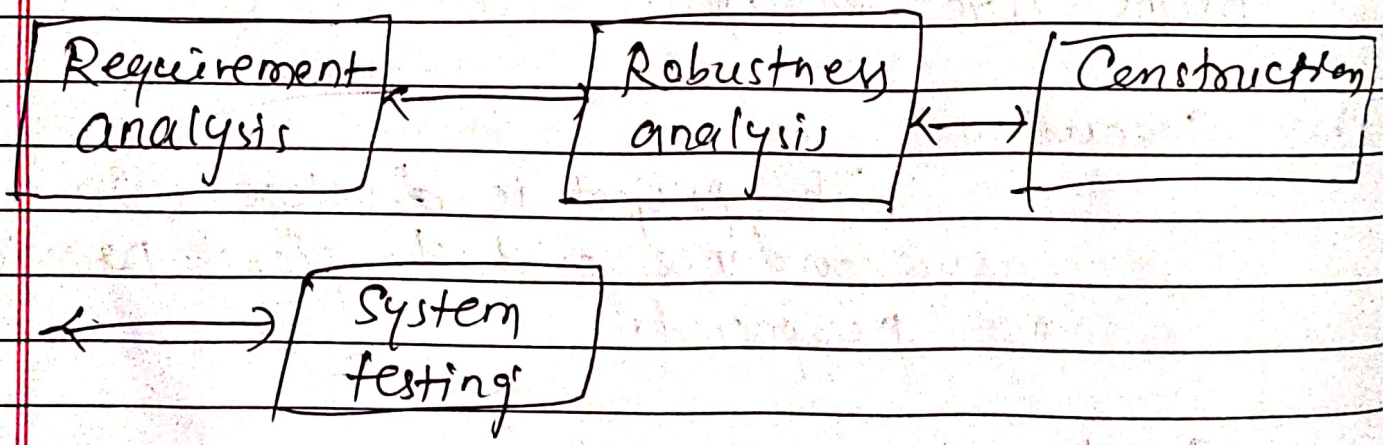


Fig: The coordination process in the OOSE.

Different development groups can form for a projects:

(i) System architecture Group:

- These people are responsible for making system architecture and delivering coherent idea to the projects. They are core of development and they same for at least first three processes i-e. requirement analysis, robustness analysis, Construction.

- Project manager belongs to this groups.

(ii) Requirement analysis group:

- Initial requirement analysis is done by a small group.

(iii) Development personnel Group:

- More detail work should be done by development personnel who are skilled for their activity.

- During Construction phase more people can add to existing group or add new group to manage more people needed in this phase.

(iv) Testing group:

- Testing is done in a separate phase often by separate group.

Besides the actual development groups, there are other roles and groups in the projects:

- Methodologist
- Quality Assurance
- Documentation, manuals & training
- Reuse Co-ordinator
- Staff.
- Help system Co-ordinator.

* Component and Components management:

A Component is a standard building unit in an organization that is used to develop application.

Eg: For a car manufacture, steering wheel, gear box, doors, seats, engine, gas tank can be the components which may or may not be further divided into smaller components.

- Component must be designed to be reused. To make a component reusable in various applications, it must be independent of the application for which it was designed (but not necessarily always.)

- Components need to be well tested and well documented so that it gets accepted by developers.

- Components that one can modify according to one's need or component that needs to be modified before reusing it is called **White box Component**. E.g. Web framework like Laravel / Symfony for PHP, Spring for Java.

- Component that doesn't need to modify for use or Components which are designed to solve specific domain problem are called **black box Component**. E.g. Moment js (Data manipulation plugin)

Use of Components:-

General Entity object:

- An entity object that are used to develop other entity object and whose information should be stored in a database. E.g. ORM (object relational mapping)

Interface object:

- Interface objects can be implemented using Components. E.g. For windows systems, windows button and scroll bars are typical Components.

Some Control objects:

- General function such as logging activities, data collection for statistics, online-help etc. used in multiple places which could be a reusable framework.

Different kinds of types:

- E.g. attributes types, types of parameters, & local types. When implementing the blocks some of this types are general and can be implemented as Component.

Component Management:

The development of Component is often more expensive that the development of ordinary software that is why the Component system is normally not profitable for only one project. The real benefits Come when a Component could be use on multiple projects. Thus, Component management should be based on multiple projects.

- A Special Component management department or group is therefore necessary, that builds Component library for the organization and also obtains Components from the market.

- There are two types of activities for Component management.

- One for the design of Complete Component System.
- One for design of individual Components.

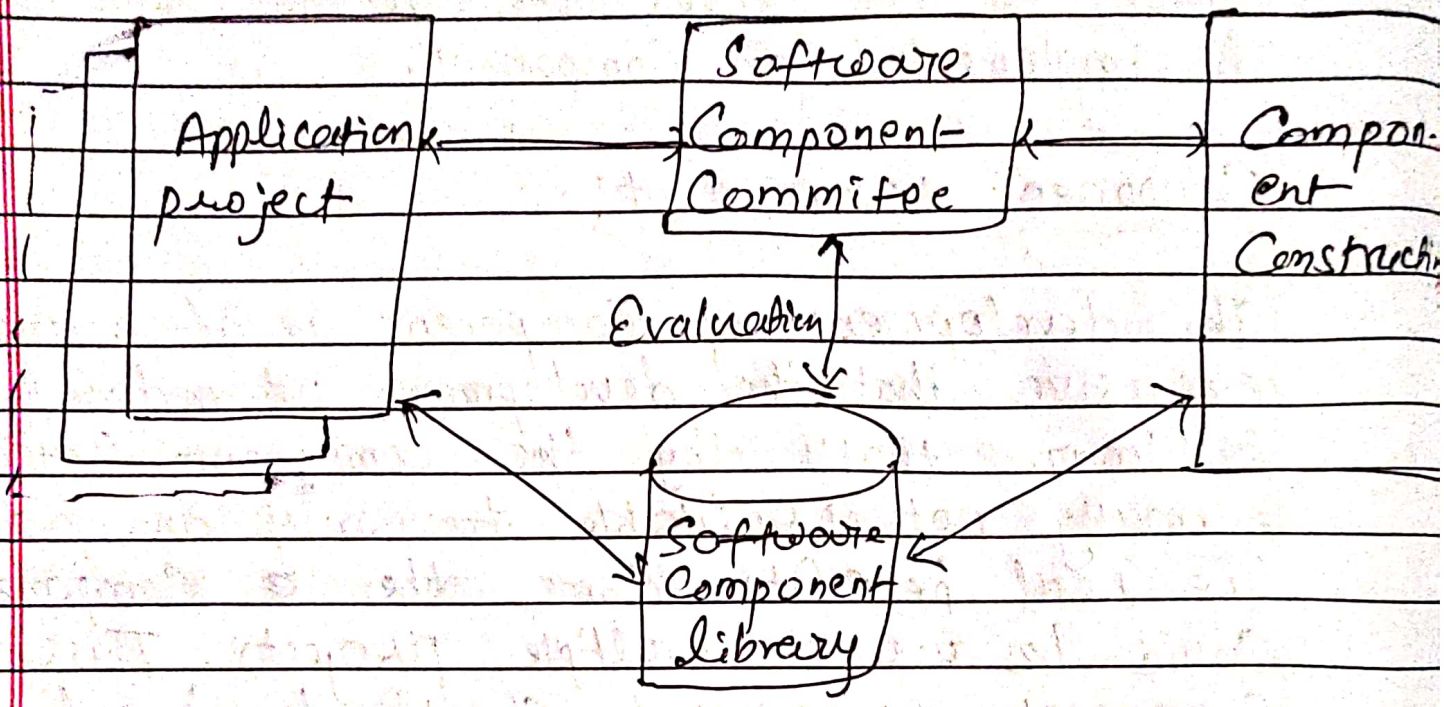


Fig: An organization of Component management

* Software Quality Assurance and Software metrices:

Software quality assurance:

- Software quality assurance is a systematic procen used to ensure the quality of software products and to identify and rectify defects or issues during the software development lifecycle.

- Primary goal → Meet the specified requirements, deliver a reliable product, and enhance customer satisfaction

- Cost and time are often tracked in the early stages, rather than quality, but quality problems appears later in development.

- Quality assurance focuses on both the product and procen.

- The main tools for quality assurance are development procen, reviews and adults testing and metrices.

- To achieve good quality disciplines and high quality awareness an independent quality group responsible for quality assurance in

the development department may be needed.

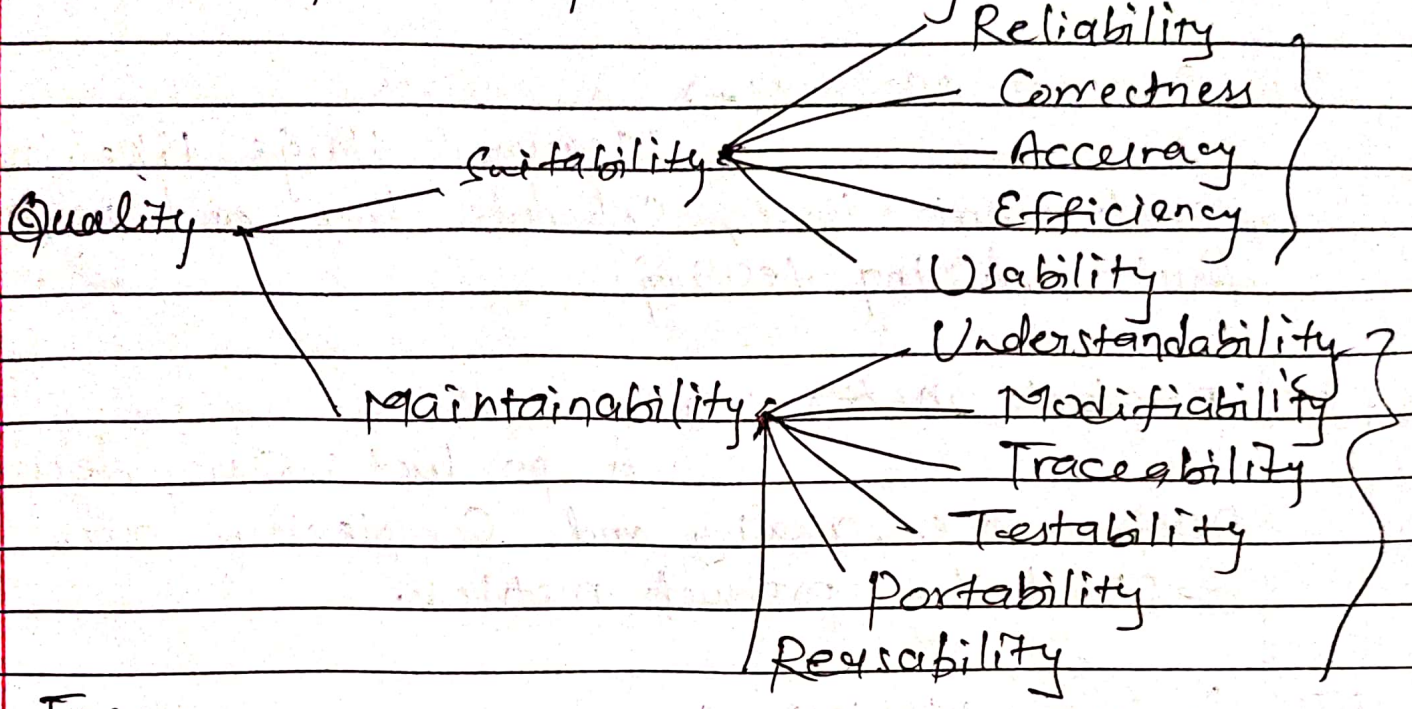


Fig:
Characteristics of Software product quality

Software metrics:

- Software metrics are quantitative measurements used to assess various aspects of software development, maintenance and quality.

- These metrics help software team and stakeholders understand the software's complexity, size, performance and other important attributes.

- E.g. Lines of Code (LOC), Cyclomatic Complexity, Code Coverage, Defect density (no. of defects per unit code), Lead time etc.

Types:

(i) Procen Metrics: →
Measures things like total development time, Schedule time and no. of faults during testing.

(ii) Product metrics: →
A product's size, design, performance, quality and Complexity are defined by product metrics.

(iii) Project Metrics: →
It is used to estimate a project's resources and deliverables as well as to determine costs, productivity and flaws.

For instance, Complexity measures (using McCabe Cyclomatic Complexity)

graph N = Connection − Nodes + 2

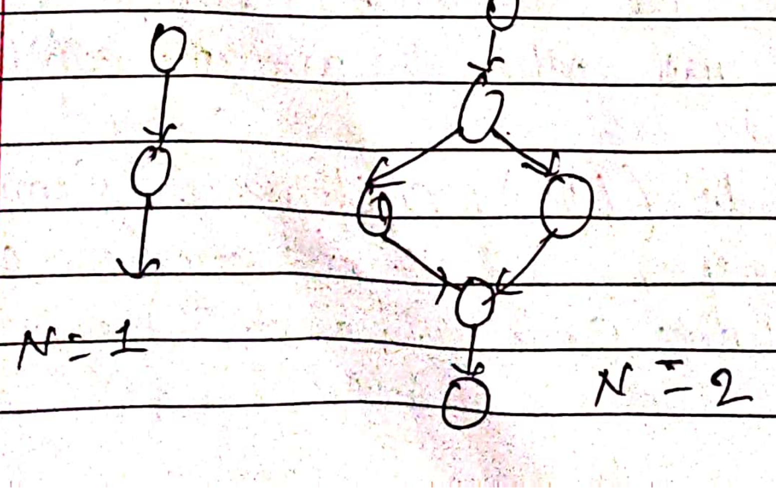


Fig: McCabe Complexity metrics

* Database - RDBMS and Object DBMS:

RDBMS (Relational Database Management System) and object DBMS (Object Database Management System) are both types of database management systems, but they differ in their data models and approaches to organizing and accessing data.

1. RDBMS (Relational Database Management System)

- RDBMS is the most widely used database management system today. It is based on the relational data model, which represents data as tables with rows and columns. Each table corresponds to an entity and each row in the table represents an instance of that entity, while columns represent attributes of the entity. The relationships between tables are established through primary keys and foreign keys.

Key features of RDBMS include:

- Data is stored in structured tables with fixed columns.
- ACID (Atomicity, Consistency, Isolation, Durability) properties are enforced to ensure data integrity.

- SQL (Structured Query Language) is used to manipulate and query data.
- RDBMS is best suited for structured and tabular data, making it suitable for applications that require a well-defined schema, such as accounting systems, HR systems and e-commerce platforms.

Popular examples of RDBMS include MySQL, Oracle database, Microsoft SQL Server etc.

2. Object DBMS / Object Database Management System

- Object DBMS is a database management system that stores data in the form of objects, similar to how object-oriented programming languages represent data. Instead of using tables, data is stored as objects with their attributes and methods. Object-oriented concepts like inheritance, encapsulation and polymorphism can be directly mapped to the database structure.

Key features of object DBMS include:

- Data is represented as objects with attributes and methods.

- Complex data structures and relationships can be easily represented.
- Some object DBMS support native query languages that support object-oriented queries.
- It is well suited for applications that deal with complex, hierarchical and interconnected data, such as CAD/CAM systems, multimedia applications and simulations.

Examples of object DBMS include Versant Object database and Objectstore.

Despite the advantages of object DBMS, it has not gained as much popularity as RDBMS.

	Key	RDBMS	OODBMS
1.	Definition	RDBMS stands for Relational Database Management System.	OODBMS stands for Object oriented Database Management System.
2.	Data Management	Data is stored as entities defined in tabular format.	Data is stored as objects.
3.	Data Complexity	RDBMS handles simple data.	OODBMS handles large & complex data.
4.	Term	An entity refers to collection of similar items having same definition.	A class refers to group of objects having common relationships, behavior and properties.
5.	Data Handling	RDBMS handles only data.	OODBMS handles both data & functions operating on that data
6.	Objective	To keep data independent form application program.	To implement data encapsulation.
7.	Key	A primary key identifies an object in a table uniquely.	Object id, OID represents an object uniquely.

* Testing - on testing, Unit, integration, System and the testing process:

* Hierarchical object oriented design (HOOD):

HOOD is a method of hierarchical decomposition of the design into software units based on identification of objects, classes and operations reflecting problem domain entities.

- It is a detailed design method which starts after analysis & extends down to coding and testing.
- It has object-oriented paradigms.

The hierarchy described in HOOD takes two forms:

(i) Uses: dependence of one object on another's services.
(ii) Functional decomposition: object split into child objects, to give functionality of the parent object.

This method has four phases:

(i) Problem definition
(ii) Development of informal solution strategy.
(iii) Formalization of the strategy
(iv) Formalization of the solution
- ↳ Identification of objects, identification of operations, grouping objects & operations, graphical description, justification of design decision.

HOOD design
Root object

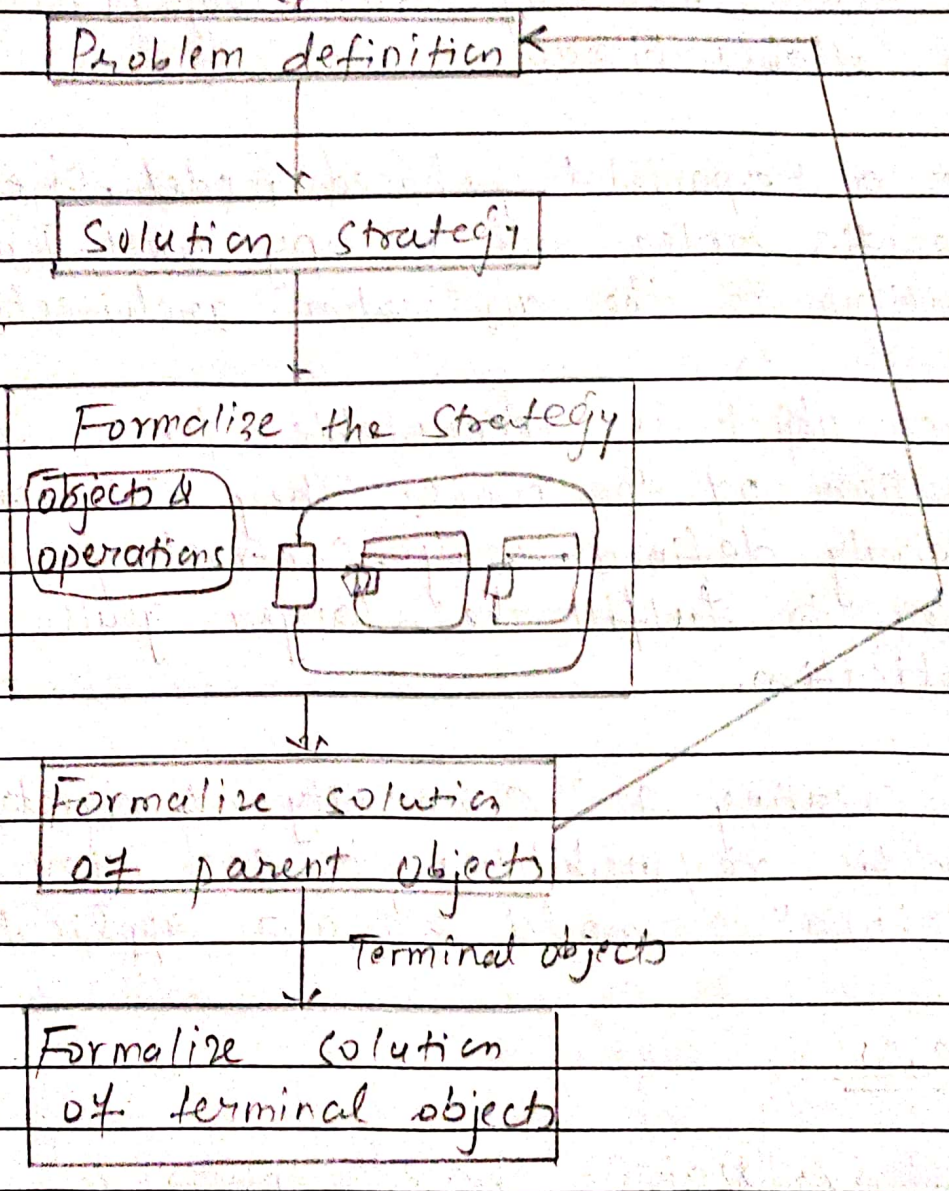


Fig: Basic design steps in HOOD

* Responsibility Driven Design (RDD):-

RDD is a way to design that emphasizes behavioral modeling using objects, responsibilities and collaborations.

- In a responsibility-based model, objects play specific roles and occupy well-known positions in the application architecture.
- Each object is accountable for a specific portion of the work. They collaborate in clearly defined way, contracting with each other to fulfill the larger goals of the application.
- By creating a "community of objects", assigning specific responsibilities to each, you build a collaborative model of our application.

Phases:

The exploratory phase consists of

(i) Classes: They are found by reading the specification and extracting the essential nouns.

(ii) Responsibilities of each class: By looking verbs in the requirements specification we can find

the action of objects in the system.

(iii) Collaboration between objects: By asking like "with what does this class need to collaborate to fulfill it's responsibilities?"

The refining phase consists of:

(i) Hierarchical between classes: Inheritance hierarchies between classes are further refined.
(ii) Subsystems: Groups of classes collaborate to fulfill their responsibilities.
(iii) Protocol: Shows the specific signature of each operations.

The output of RDD consist of:

(i) A graph of each class hierarchy
(ii) A graph of the collaboration for each subsystem.
(iii) A specification of each class
(iv) A specification of each subsystem.
(v) A specification of the contracts supported by each class and subsystem.

* Object Modeling Technique (OMT):

The object modeling technique covers analysis design and implementation of a system using an object oriented technique.

- OMT is a fast, intuitive approach for identifying and modeling all the objects making up a system.

- Details such as class, attributes, method, inheritance and association also can be expressed easily.

OMT consists of four phases, which can be performed iteratively.

(i) Analysis: The results are object and dynamic and functional models.

(ii) System Design: The results are structure of basis architecture of system along with high-level strategy decisions.

(iii) Object Design: Produces a design document, consisting of detailed object static, dynamic and functional models.

(iv) Implementation: Produces reusable, extensible

and robust code.

OMT separates modeling into three different parts:

(i) An object model → describes the static structure of the system with class and relationship.

(ii) A dynamic model → Captures aspect of the object model with events and state of the objects.
→ presented by the state diagram and event flow diagrams.

(iii) A functional model → Describes the computation in terms of how output values are derived from input values.
→ Presented by data flow and constraints.

* Object Oriented Analysis / Coad-yourdon (OOA) :-

In OOA, an analysis model is developed to describe the functionality of the system.

The idea in Coad-yourdon design is to extend this model with respect to process (tasks), human interfaces and DBMS.

This method consists of five steps.

(i) Finding class and objects:

(ii) Identifying structure: (generalization-specification)

(iii) Defining subjects: (Subjects are group of class & objects)

(iv) Defining attributes: (identifying information & association that should associated with each & every instance.)

(v) Defining services: (defining operations of classes).

Figure:

* Object-oriented design (OOD/BOOCH):

- It is a widely used object oriented method that helps us design our system using the object paradigm.

- It covers the analysis and design phases of an object oriented system.

- The Booch method consists of the following diagrams:

(i) Class diagrams,
(ii) Object diagrams,
(iii) State Transition diagrams
(iv) Module diagrams
(v) Process diagrams
(vi) Interaction programs.

Method involves following steps:

(i) Identify classes and object:

(ii) Identify class and object semantics:

(Establishing relationship bet^n the classes & objects)

(iii) Identify class and object relationships:
(How class & object interact with each other)

(IV) Implementing classes and objects:
(How to use particular programming language to implement)

Figure