Obeject Oriented Analysis And Design

Object Oriented Analysis and Design (OOAD) is a software development approach that uses objects and their interactions to design and develop software systems.

Object Oriented Analysis and Design is a way to design software by thinking of everything as objects similar to real-life things. In OOAD we first understood what the system needs to do, then identify key objects, and finally decide how these objects will work together. This approach helps make software easier to manage, reuse, and grow.

The concepts of the OOAD are based on the object-oriented programming (OOPS) and is an organized and systematic approach to designing and developing software systems. It is a software engineering paradigm

$$\begin{tikzcd} & \text{OOAD} \arrow[dl] \arrow[dr] & \\ \text{OOA} & & \text{OOD} \end{tikzcd}$$

Object Oriented Analysis (OOA)

Object Oriented Analysis is the first step in the Object Oriented Analysis and Design (OOAD) process. It is like the planning phase before building a house. In planning we focus on understanding the problem we want to solve and identifying the key objects involved.

Object Oriented Analysis is the process of understanding and analysing the system requirements by looking at the problem situation in terms of objects.

Analysis is like the planning phase. You figure out what kind of objects you need and how they work together.

Object Oriented Design (OOD)

Object Oriented Design (OOD) is a way of designing software systems that uses objects and their interactions to solve problems.

In OOD, we break down a big problem into smaller, manageable "objects." Each object has its own set of characteristics and actions it can perform.

Object Oriented Analysis and Design (OOAD) is a powerful tool for building complex software systems that are easier to understand, maintain, and adapt to changing needs.

Concept Of Object Oriented Programming System (OOPS)

OOPS stands for Object-Oriented Programming. OOPS is a programming paradigm based on the concept of "Objects".

Imagine you're building a house, you don't just know all the materials you use and throw them together. You have pre-made parts like doors and windows, and you assemble them following specific rules. OOPS is like that for software.

1.Object - These are like the "things" in your program. They can be anything, like a car, a person, a button on the screen. Each object has its own set of characteristics (like colour, size, speed for a car) and behaviours (like starting, stopping, turning).
Object is a real-time Entity(a) Physical Entity - An object is a real-time physical instance of a class.

Characteristics of an Object

Properties (Attributes)

Methods

State

Behaviour

1.Properties(Attributes): an attribute is a characteristic of a class or object, such as name, age, or address

2.Methods: A Method is an Action that a class or object can perform, such as calculate, validate, or update

3. State: State refers to the current state of the object, such as weather, "turned on" or "turned off", "open" or "close".

4. Behaviour: Behaviour refers to the how the object reacts to different situations.

2.class:- a class is a blue print or a template that defines the properties and behaviours of an object.it is a fundamental concept in object-oriented programming

3.Encapsulation:- Encapsulation is a programming concept that combines data and methods into a single line. Encapsulation is the concept of hiding the implementation details of an object from the outside

4.Abstraction: Abstraction is the concept of showing only the necessary information to the outside world. while hiding the implementation details

5.Inheritance:- Inheritance refers to the process of transmission of genus from parent to child.

Inheritance is nothing but process of the derived a new class (or) object from the exsiting class is called inheritance

6.Polymorphism: polymorphism means ”many forms”.it allows objects of different classes to be treated in a similar way.

Polymorphism can be achived through method overloading , ie, multiple methods with the same name but different parameters

Example : different vechicle’s like car, trucks, bike can all “move” even though they do it differently

OOPS Example Program

```
Public class Car

{

String colour;

Int speed;

Public void start()

{

System.out.println(“ car started “);

}

Public void accelerate()

{

Speed +=10;

}

}
```

```
Public class Main

{

Public  static void main(String[]args)

{

Car  mycar  =  new car ();

mycar.colour = “red”;

mycar.start();

mycar.accelerate();

}

}
```

Benefits of The OOPS

1.Modularity:- Represent this as a set of independent blocks or modules connected by lines. Each block represents a separate component of the system that can function independently.

2.Reusability:- code can be re used in multiple programs

3.Easier to maintenance :- code is easier to modify and update

4.Improved readability:- code is more organized and easier to understand

UNIT I

Introduction: The Structure of Complex systems, The Inherent Complexity of Software, Attributes of Complex System, Organized and Disorganized Complexity, Bringing Order to Chaos, Designing Complex Systems. Case Study: System Architecture: Satellite-Based Navigation.

What is a System?

In simple words, a system is a group of things that work together to achieve a common goal. It's like a team, where each member has a specific role to play, and they all cooperate to accomplish a task.

Example 1:

- ✓ A school is a system with teachers, students, classrooms, and a principal.
- ✓ Each person has a role, and they work together to provide education.

Example 2:

- ✓ A car is a system made up of many parts like the engine, wheels, brakes, and steering wheel.
- ✓ Each part has a specific function, and they all work together to make the car move.

What is a Software System?

A software system is a special kind of system that uses computer programs to achieve a goal. A software system is a set of instructions that a computer follows to perform a task or provide a service. It's like a recipe for a computer, telling it what to do step by step.

Examples of Software System:

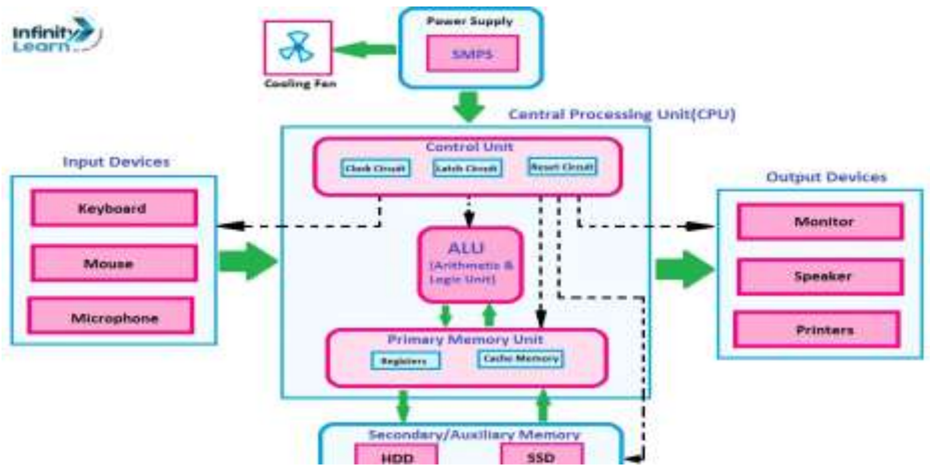
- ✓ A video game: it is a software system that creates an interactive world for you to play in.
- ✓ A website: it is a software system that displays information and lets you interact with it online.
- ✓ A word processor: it is a software system that creates an interactive world for you to play in.

The Structure of Complex systems

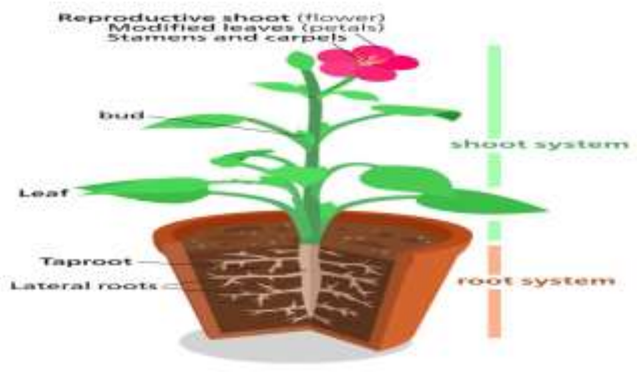
A Complex system is a System that Consists of many Interconnected Components that interact with each other in Complex way. These systems are difficult to understand, predict, and control because of their intricate relationships and behaviour's.

Imagine a jigsaw puzzle. Each piece is Simple. But when you put them together you get a beautiful picture. A Complex system like that. It's made up of many simple parts, but the way they interact creates something much bigger

.**The structure of a Personal Computer:** A personal computer is a device of moderate complexity. Major elements are CPU, monitor, keyboard and some secondary storage devices. CPU encompasses primary memory, an ALU, and a bus to which peripheral devices are attached. An ALU may be divided into registers which are constructed from NAND gates, inverters and so on. All are the hierarchical nature of a complex system.



The structure of Plants: Plants are complex multicellular organism which are composed of cells which is turn encompasses elements such as chloroplasts, nucleus, and so on. For example, at the highest level of abstraction, roots are responsible for absorbing water and minerals from the soil. Roots interact with stems, which transport these raw materials up to the leaves. The leaves in turn use water and minerals provided by stems to produce food through photosynthesis.



The structure of Animals: Animals exhibit a multicultural hierarchical structure in which collection of cells form tissues, tissues work together as organs, clusters of organs define systems (such as the digestive system) and so on.

The structure of Matter: Nuclear physicists are concerned with a structural hierarchy of matter. Atoms are made up of electrons, protons and neutrons. Elements and elementary particles but protons, neutrons and other particles are formed from more basic components called quarks, which eventually formed from pro-quarks.

The structure of Social institutions: In social institutions, group of people join together to accomplish tasks that can not be done by made of divisions which in turn contain branches which in turn encompass local offices and so on.

Key Characteristics of Complex Systems:

1. Many Parts
2. Interconnections
3. Non-linearity
4. Adaptation

1**. Many Parts**: Complex Systems have lots of different parts, like a city with its buildings, roads, people.

2**. Interconnections**: These parts are connected to each other, like the roads linking different parts of the city.

3. **Non-linearity**: A small change can have a big effect, like a single car accident causing a traffic jam.

4**.Adaptation**: Many complex systems can change and adapt over time to their environment.

Examples of the Complex System:

1. Human Brain
2. The Internet
3. The weather
4. An ecosystem

1. Human Brain

The Human Brain made up of Billions of neurons that communicate with each other.

2. The Internet

The Internet is a world wide network of computers and devices connected through the networks.

3.The weather

The weather is influenced by many factors like temperature, wind, and humidity.

4. An ecosystem

An ecosystem is made up of plants, animals, and their environment.

Why are Complex systems helps:

- ✓ **Climate Change**: Predictions how the Earth's Climate will change.
- ✓ **Disease out breaks**: Understanding how diseases spread and finding ways to stop them.
- ✓ **Social issues**: Understanding how societies function and how to improve them.

The Inherent Complexity of Software

Inherent complexity of software is the essential level of complexity in software that is necessary to solve a problem.

Inherent complexity can make software systems difficult to understand, modify, and maintain. However, it can be managed through careful design and implementation practices.

The Inherent Complexity of software stems from several key factors:

1. **Problem Domain Complexity**
2. **Software's Flexibility**
3. **Managing the development process.**
4. **Characterizing Discrete Systems**

1. Problem Domain Complexity:

- ➢ The problems software aims to solve often mirror the intricate nature of the real world.
- ➢ Consider financial systems, air traffic control, or medical simulations. These domains are inherently complex.
- ➢ Software designed for such domains must accurately model and handle this complexity.

2. Software's Flexibility:

- ➢ Software flexibility refers to the ability of a software system to adapt to changing requirements, needs, or environments without requiring significant modifications or rewrites.
- ➢ In software flexibility, easily modify and adapt software is a powerful advantage.

3. Managing the Development Process:

- ➢ Large software projects involve teams of developers working together.
- ➢ Coordinating effort, ensuring consistency, and managing dependencies add to the complexity.

- Miscommunication or lack of clear guidelines can lead to significant issues.

4. Characterizing Discrete Systems:

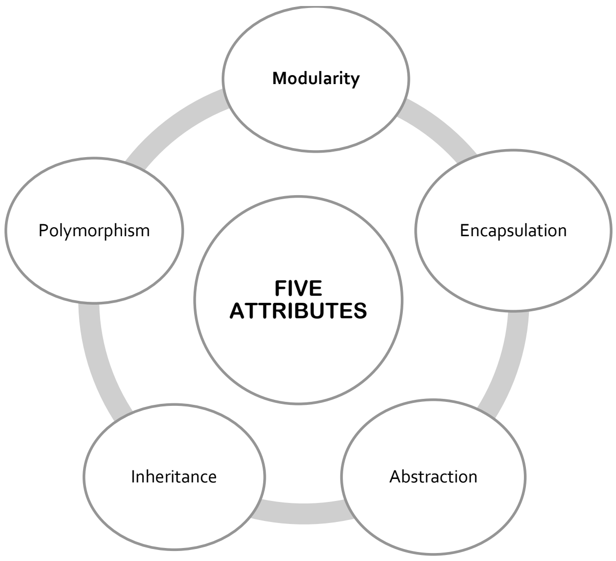
- Software discreteness can lead to unexpected behaviour and edge cases that are challenging to predict and handle.
- Software deals with discrete systems, unlike continuous physical systems.

Attributes of Complex System

In Object-Oriented Analysis and Design (OOAD), **Complex Systems** refer to systems that have many interconnected components and behaviours. These systems can be large, intricate, and dynamic, often involving various types of interactions and layers of functionality.

characteristics that make these systems complex:

1. Modularity
2. Encapsulation
3. Abstraction
4. Inheritance
5. Polymorphism



1. **Modularity:-**Represent this as a set of independent blocks or modules connected by lines. Each block represents a separate component of the system that can function independently.
2. **Encapsulation :-** Encapsulation is a programming concept that combines data and methods into a single line. Encapsulation is the concept of hiding the implementation details of an object from the outside
3. **Abstraction:** Abstraction is the concept of showing only the necessary information to the outside world. while hiding the implementation details

4.Inheritance :- Inheritance refers to the process of transmission of genus from parent to child.

Inheritance is nothing but process of the derived a new class (or) object from the exsiting class is called inheritance

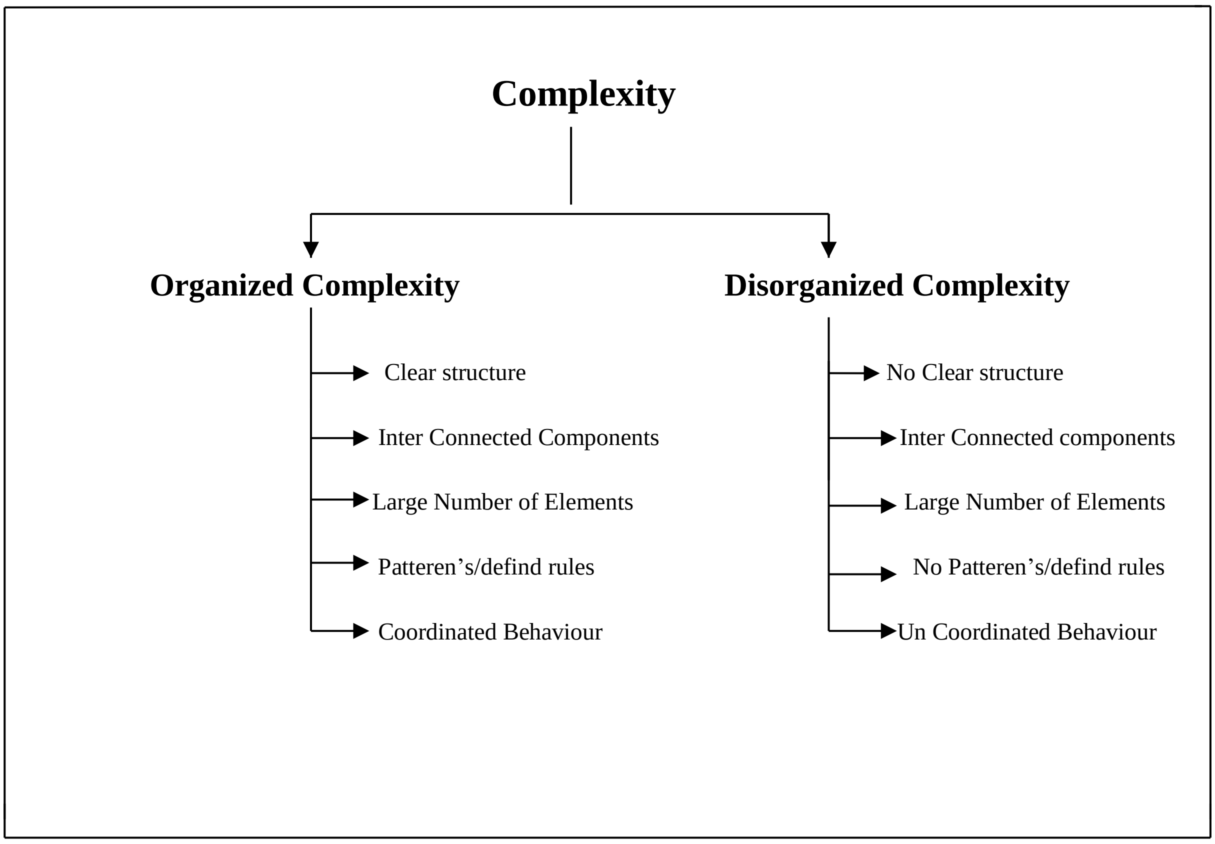
5.Polymorphism :- polymorphism means "many forms".it allows objects of different classes to be treated in a similar way.

Polymorphism can be achived through method overloading , I.e, multiple methods with the same name but different parameters

Example :- different vechicle's like car, trucks, bike can all "move" even though they do it differently

Organized and Disorganized Complexity

In object oriented analysis and design Complexity can broadly categerized into two types



1.Organized Complexity

Organized complexity refers to complex system that have a clear structure, patteren or organization. These systems are made up by many components that work together in coordinated way.

Characteristics of Organized Complexity:

1.Clear Structure :organized complex system have a well- defind structure (or)Archicture

2.Inter Connected Components :these systems made up by many inter connected components that work together

3.Large Number of Elements :In this organization complexity system involves many components

4.Patterens/ defind rules :these system exhibit patterns or organization that make them easier to understand

5. Coordinated Behaviour : the components in organized complex system behave in coordinated way

Examples of the Organized Complexity

1.Human Body

The human body is a complex system made up many interconnected organs and systems that work together in a coordinated way.

2.computer operating system

A computer operating system is a complex system with a clear structure and organization (or) pattren made up of many interconnected components

3.Social Network

A social network is complex system with a clear structure ,made up of many interconnected individuals

2.Disorganized Complexity

Disorganized Complexity refers to complex systems that lack a clear structure, pattern (or) organization. These systems are made up many components that do not work together in a coordinated way.

Characteristics of Disorganized Complexity

1.Lack of Clear Structure: Disorganized complex systems lack well-defined structure or architecture.

2.No Patterns (or) Organization: These systems do not exhibit patterns or organization that make them easier to understand.

3.Inter Connected Components

Consist of many inter-connected components that do not coordinate in an uncoordinated behaviour.

4.Un Coordinated Behaviour

The components in disorganized complex systems behave in an uncoordinated way.

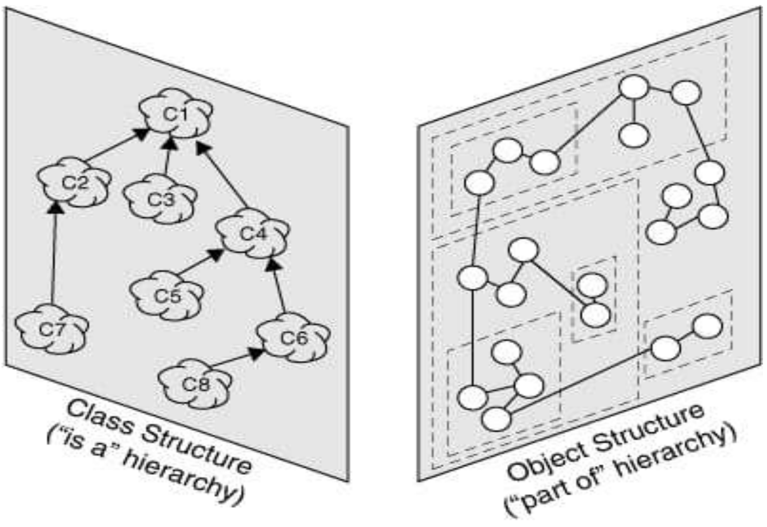
Example of Disorganized Complexity

1. Traffic Congestion: Traffic Congestion is a complex system with no clear structure or organization, made up of many inter-connected vehicles and roads.

2. Weather Patterns: Weather patterns are complex systems with no clear structure or organization, made up of many inter-connected components.

Canonical form of complex system

The diagram visually represent a relationship between class and objects in a system



Classe's and objects built on more primitive ones. What class or object is chosen as primitive is relative to the problem at hand. Looking inside any given level reveals yet another level of complexity. Especially among the parts of the object structure, there are close collaborations among objects at the same level of abstraction.

Combining the concept of the class and object structures together with the five attributes of a complex system (hierarchy, relative primitives [i.e., multiple levels of abstraction], separation of concerns, patterns, and stable intermediate forms), we find that virtually all complex systems take on the same (canonical) form, as we show in Figure 1–2. Collectively, we speak of the class and object structures of a system as its architecture.

Notice also that the class structure and the object structure are not completely independent; rather, each object in the object structure represents a specific instance of some class. (In Figure 1–2, note classes C3, C5, C7, and C8 and the number of the instances 03, 05, 07, and 08.) As the figure suggests, there are usually many more objects than classes of objects within a complex system. By showing the "part of" as well as the "is a" hierarchy, we explicitly expose the redundancy of the system under consideration. If we did not reveal a system's class structure, we would have to duplicate our knowledge about the properties of each individual part. With the inclusion of the class structure, we capture these common properties in one place

Organized vs. Disorganized Complexity

Aspect	**Organized Complexity**	**Disorganized Complexity**
Structure	Clear, well-defined structure	No clear structure or organization
Predictability	Behaviour is predictable	Behaviour is often unpredictable
Modularity	System is divided into modular components	No clear separation between components
Scalability	Easy to scale and add new features	Hard to scale, as each change can affect the whole system
Maintainability	Easy to maintain and update	Difficult to maintain or update without causing other issues
Example	Well-structured software system	Unorganized legacy codebase

In OOAD, the goal is often to reduce complexity by organizing systems in a way that makes them easier to understand, modify, and scale. OOAD uses **objects** and **classes** to break down complex systems into smaller, manageable parts.

1. **Modularity**: By breaking a system into classes and objects, OOAD makes it easier to understand each piece and how they fit together.
2. **Encapsulation**: This hides the internal details of each object and only exposes necessary information, making the system easier to understand and use.
3. **Inheritance**: Through inheritance, you can build new functionality on top of existing classes without having to rewrite everything, keeping the system organized.
4. **Polymorphism**: This allows different objects to respond to the same commands in different ways, keeping the system flexible and reducing unnecessary complexity.

Concept of Hieararchy :

Hieararchy is a system of organization where things are arranged in a series of level or ranks

Characteristics of Hierarchy

1. Levels: A hierarchy has multiple levels, with each level having a specific role or responsibility.

2. Ranking: Each level is ranked in order of importance, authority, or size.

3. Structure: A hierarchy has a clear structure, with each level connected to the ones above and below it.

Examples of Hierarchy

1. Company Organization: A company's hierarchy might include CEO, managers, team leaders, and employees.

2. Government: A government's hierarchy might include president, ministers, department heads, and officials.

3. Military: A military's hierarchy might include generals, colonels, captains, and soldiers.

4. File System: A computer's file system hierarchy might include folders, subfolders, and files**.**

Class Hierarchy

A class hierarchy is a way of organizing classes in a programming language, where one class is a specialized version of another class.

Characteristics of Class Hierarchy

1. Inheritance: A class can inherit properties and behavior from another class.

2. Parent-Child Relationship: A parent class (also called superclass or base class) has child classes (also called subclasses or derived classes) that inherit from it.

3. Levels: A class hierarchy has multiple levels, with each level representing a more specific class.

Example of Class Hierarchy

1. Vehicle (parent class)
   - Car (child class)
   - Truck (child class)
   - Motorcycle (child class)
2. Car (parent class)
   - Sedan (child class)
   - SUV (child class)

Object Hierarchy

An object hierarchy is a way of organizing objects in a programming language, where one object is a specialized version of another object.

Characteristics of Object Hierarchy

1. Inheritance: An object can inherit properties and behavior from another object.

2. Parent-Child Relationship: A parent object (also called superclass or base object) has child objects (also called subclasses or derived objects) that inherit from it.

3. Levels: An object hierarchy has multiple levels, with each level representing a more specific object.

Example of Object Hierarchy

1. Animal (parent object)
   - Mammal (child object)
   - Bird (child object)
   - Reptile (child object)
2. Mammal (parent object)
   - Dog (child object)
   - Cat (child object)
   - Human (child object)

Chaos in OOAD

In OOAD, chaos refers to a complex and disorganized system, process, or problem domain. It's like a big puzzle with many pieces that don't seem to fit together.

Bringing Order to Chaos

Bringing order to chaos means taking a complex and disorganized system, process, or problem domain and breaking it down into smaller, more manageable pieces. It's like solving the puzzle by identifying patterns, relationships, and structures.

Steps Bring Order to Chaos in OOAD

1. **Identify the Problem Domain**: Define the scope and boundaries of the problem or system.

2. **Gather Requirements**: Collect information about the problem or system through stakeholder interviews, surveys, and observations.

3. **Analysis the Problem Domain**: Break down the problem or system into smaller components and identify relationships between them.

4. **Identify Patterns and Structures**: Look for patterns, such as repetition, similarity, or hierarchy, and structures, such as classes, objects, or relationships.

5. **Create a Conceptual Model**: Develop a high-level model of the problem or system using concepts, such as classes, objects, and relationships.

6. **Refine the Model:** Iterate and refine the model based on feedback from stakeholders and further analysis.

7. **Implement the Solution**: Use the refined model to guide the implementation of the solution.

Bringing Order to Chaos Principals

1.Abstraction

2.Hierarchy

3.Decomposition

1.Role of the Abstraction

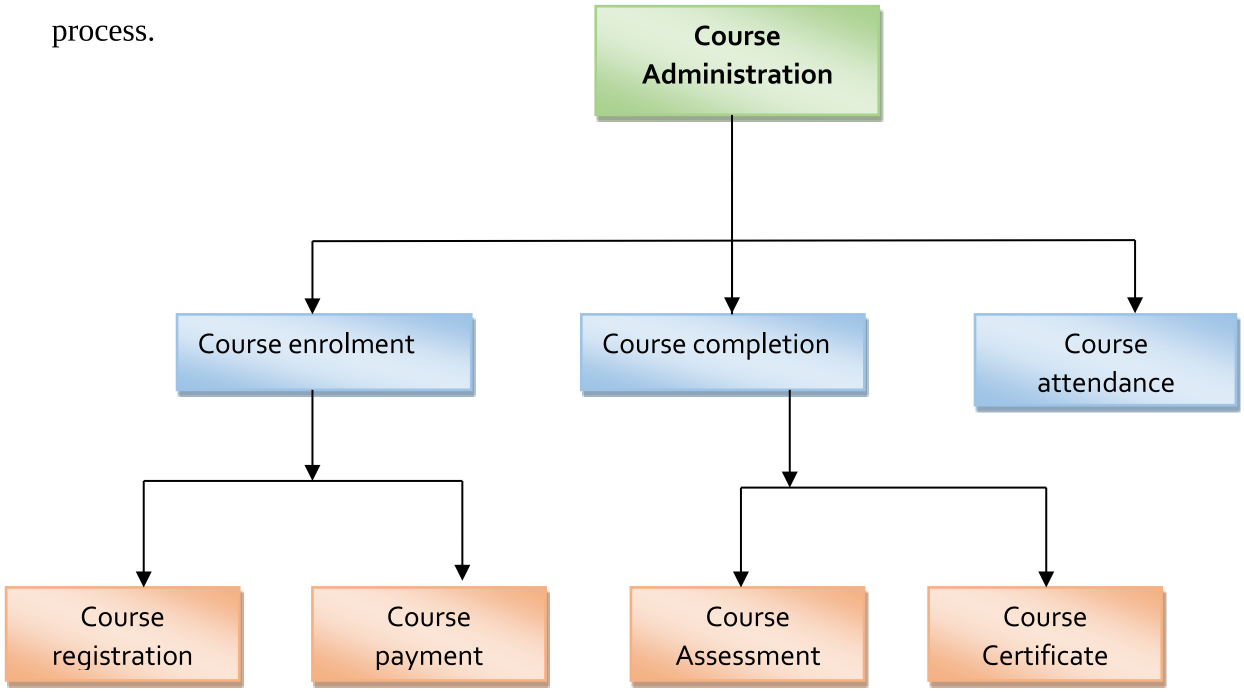
- ✓ Abstraction help us understand objects or things by simplifying them we group similar objects and things together and focus on the general idea.
- ✓ In computer programming we use classes to define groups of objects with similar properties this is form of a abstraction

2.The Role of The Hierarchy

- ✓ Hierarchy helps us see which see how different objects interact with each other
- ✓ Hierarchy helps us see which objects are more general and which are more specific

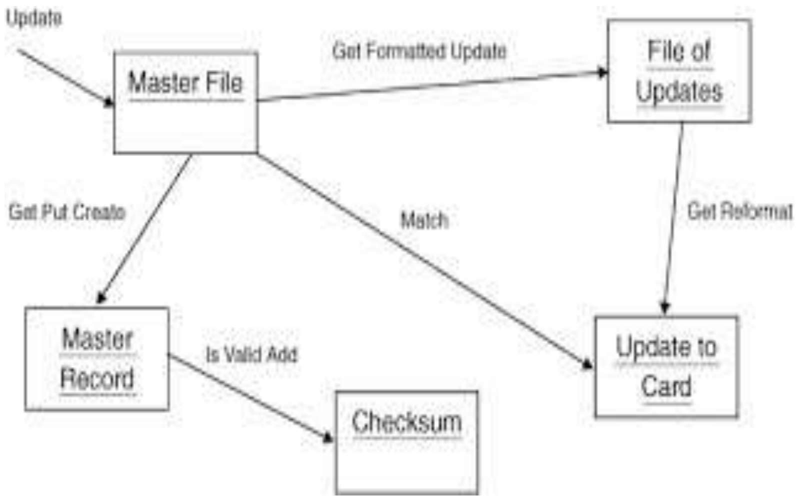
3.Decomposition

Decomposition helps us breaking down big problem into smaller easier to handle pieces Algoithmic Decomposition was each model in the system denotes a major step in overall process.



Object oriented Decomposition

The object oriented Decomposition focus on object and their interactions .each object in our solution represent it own unique behaviour



Decomposition means dividing a large complex system into a hierarchy of smaller components with lesser complexities, on the principles of divide–and–conquer. Each major component of the system is called a subsystem. Object-oriented decomposition identifies individual autonomous objects in a system and the communication among these objects

Benefits of Bringing Order to Chaos

1. **Improved Understanding**: Gain a deeper understanding of the problem domain and its complexities.
2. **Simplified Solution**: Develop a simpler, more elegant solution that addresses the core needs of the problem domain.
3. **Increased Maintainability**: Create a solution that is easier to maintain, modify, and extend over time.
4. **Better Communication**: Improve communication among stakeholders, developers, and users by providing a clear, concise, and consistent understanding of the problem domain and its solution.

Designing Complex Systems

Designing Complex Systems in Object-Oriented Analysis and Design (OOAD) refers to the process of planning, structuring, and building large and intricate software systems using object-oriented principles. These principles help break down complex systems into smaller, manageable pieces.

Using OOAD steps and principal's you can design complex systems in a structured and organized way, making them easier to understand ,maintain, and evolve over time

Engineering as a Science and an Art the role of the engineer as artist is particularly challenging when the task is to design an entirely new system. Especially in the case of reactive systems and systems for command and control, we are frequently asked to write software for an entirely unique set of requirements, often to be executed on a configuration of target processors constructed specifically for this system. In other cases, such as the creation of frameworks, tools for research in artificial intelligence, or information management systems, we may have a well-defined, stable target environment, but our requirements may stress the software technology in one or more dimensions. For example, we may be asked to craft systems that are faster, have greater capacity, or have radically improved functionality. In all these situations, we try to use proven abstractions and mechanisms (the "stable intermediate forms," in Simon's words) as a foundation on which to build new complex systems. In the presence of a large library of reusable software components, the software engineer must assemble these parts in innovative ways to satisfy the stated and implicit requirements, just as the painter or the musician must push the limits of his or her medium

Challenges of Designing Complex Systems

1. **Many Interconnected Components**: Complex systems have many components that interact with each other, making it hard to understand how the system works.
2. **Non-Linear Behaviour**: Small changes can have big, unexpected effects.
3. **Emergence**: The system exhibits behaviours that cannot be predicted from its individual components.
4. **Uncertainty**: It's hard to predict how the system will behave in different situations.

Principles of Designing Complex Systems in OOAD

1. **Modularity**: Break down the system into smaller, independent modules.
2**. Abstraction:** Focus on essential features and behaviours, while ignoring non-essential details.
3**. Encapsulation**: Hide internal details of modules from the outside world.
4**. Loose Coupling**: Minimize dependencies between modules.

5. **Separation of Concerns**: Separate different concerns, such as presentation, business logic, and data storage.

Steps to Design Complex Systems in OOAD

1. Identify the Problem Domain: Define the scope and boundaries of the problem or system.

2. Gather Requirements: Collect information about the problem or system through stakeholder interviews, surveys, and observations.

3. Analysis the Problem Domain: Break down the problem or system into smaller components and identify relationships between them.

4. Identify Patterns and Structures: Look for patterns, such as repetition, similarity, or hierarchy, and structures, such as classes, objects, or relationships.

5. Create a Conceptual Model: Develop a high-level model of the problem or system using concepts, such as classes, objects, and relationships.

6. Refine the Model: Iterate and refine the model based on feedback from stakeholders and further analysis.

7. Implement the Solution: Use the refined model to guide the implementation of the solution.

Techniques for Designing Complex Systems in OOAD

1. Object-Oriented Analysis (OOA): Use techniques, such as use cases, class diagrams, and sequence diagrams, to analysis and model the problem domain.

2. Domain-Driven Design (DDD): Focus on understanding the core business domain and modeling it using concepts, such as entities, value objects, and aggregates.

3. Pattern Recognition: Identify patterns, such as the Singleton pattern or the Factory pattern, to help organize and structure the solution.

4. Abstraction: Focus on essential features and behaviours, while ignoring non-essential details.

Benefits of Designing Complex Systems in OOAD

1**. Improved Understanding**: Gain a deeper understanding of the problem domain and its complexities.

2**.Simplified Solution**: D. Simplified evelop a simpler, more elegant solution that addresses the core needs of the problem domain.

3. **Increased Maintainability**: Create a solution that is easier to maintain, modify, and extend over time.

4. **Better Communication**: Improve communication among stakeholders, developers, and users by providing a clear, concise, and consistent understanding of the problem domain and its solution.

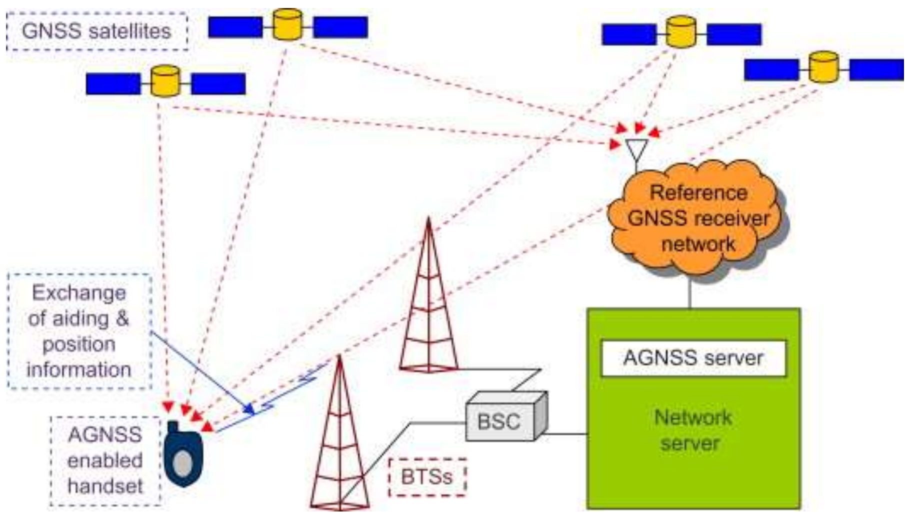
Case Study: System Architecture of Satellite-Based Navigation in OOAD

Satellite-based navigation systems, like GPS (Global Positioning System), help users figure out where they are and how to get to a specific place using satellites orbiting the Earth. These systems work by receiving signals from satellites, which tell the user's device (like a phone or car GPS) their exact location.

SYSTEM ARCHITECTURE

The **system architecture** refers to the structure of the system — how different parts of the system work together. For this case study, it describes the main components of a satellite navigation system, such as:

- **Satellites**: They send signals to the receivers (like GPS devices).
- **Ground Stations**: These monitor and manage the satellites.
- **User Devices**: These receive the satellite signals and help users navigate (e.g., GPS receivers, smartphones).


GNSS satellites
Reference GNSS receiver network
Exchange of aiding & position information
AGNSS server
Network server
BSC
AGNSS enabled handset
BTSs

How OOAD is Used

In OOAD, the system is broken down into different objects or components. Each object has:

1. **Attributes**: These are the data or properties that describe the object.
2. **Methods/Behaviours**: These are the actions the object can perform.

For Example:

- A **Satellite** object might have attributes like location and signal strength, and methods like sending a signal.
- A **GPS Device** object might have attributes like current location and destination, and methods like calculating the best route.

WHY USE OOAD FOR SATELLITE NAVIGATION?

By using OOAD, designers can:

- Break down the system into smaller, manageable parts (objects).
- Focus on real-world components (like satellites, devices, and signals) and their interactions.
- Make the system easier to maintain and improve in the future because the design is modular and flexible.

This case study shows how to use OOAD to design a satellite-based navigation system by breaking it down into objects with specific roles, attributes, and behaviours. This method helps in creating a clear, structured system that can be easily developed and updated.

******