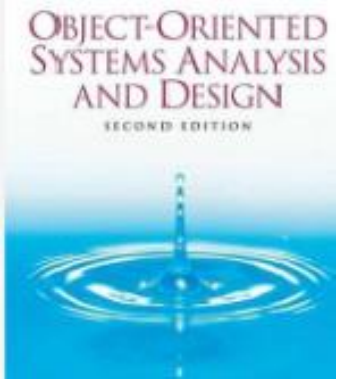
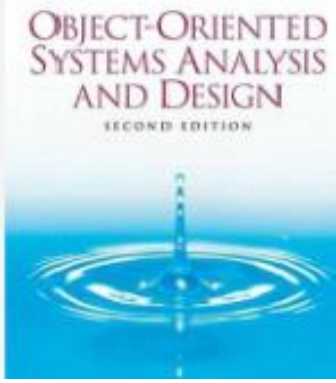
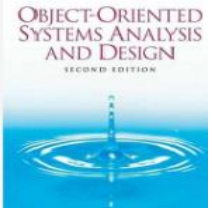
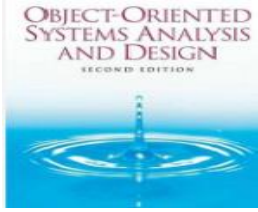




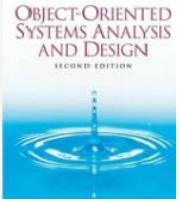
Object Oriented System Analysis and Design (OOSAD) INSY3063



CHAPTER 1: OBJECT ORIENTATION – THE NEW SOFTWARE DEVELOPMENT PARADIGM

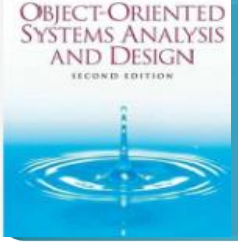


What is System, System thinking and Software?

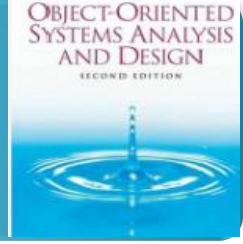


CHAPTER 1: OBJECT ORIENTATION – THE NEW SOFTWARE DEVELOPMENT PARADIGM

- Review of Systems and System thinking, Software
- Structured VS Object Oriented Paradigm
- The Potential Benefits and Drawbacks of Structured System Analysis and Design
- The Potential Benefits and Drawbacks of Object Orientation System Analysis and Design
- Object Technology
- The Object Orientation Software Process Models
- Software Process Patterns



Objectives



- To revise what, system, system thinking and software means.
- Differentiate structured approach from object-oriented approach.
- Define what object technology is and examine its basic concepts
- Explain the benefits and drawback of structured and object orientation
- Explain Object orientation software process models and software process patterns

What is System and Systems thinking?

- **A system** is an interrelated set of business procedures (or components) used within one business unit, working together to achieve some purpose.
- **System** is an organized collection of parts (or subsystems) that are highly integrated to accomplish an overall goal.

(Read about the important characteristics of a system.)

- **Systems thinking** is the ability to see organizations and information systems as systems.
- **Systems thinking** provides a framework from which to see the important relationships among ***information systems, the organizations they exist in, and the environment in which the organizations themselves exist***

What is System and Systems thinking?

- **Systems thinking** is a holistic way to investigate factors and interactions that could contribute to a possible outcome.
- **Systems thinking** provides an understanding of how individuals can work together in different types of teams and through that understanding, create the best possible processes to accomplish just about anything.
- **Systems thinking** is the philosophy that all things are connected and is often the missing piece in our attempts to solve today's problems.
- It is basic not Optional.

  Systems thinking answer the following

  - ✓ What is the purpose?
  - ✓ How do we assemble the pees and parts?
  - ✓ How will we build and maintain the system?

- **Important systems concepts**
  - **Decomposition-** Breaking a system into smaller subsystems (Components).
  - **Modularity-** This is a direct result of decomposition. It refers to dividing a system into **chunks** or **modules** of a relatively uniform size
  - **Coupling**: Subsystems should be as independent as possible
  - **Cohesion**: This is the extent to which a subsystem performs a single function- ***separation of concern (Highly Cohesiveness)***

- **The product**
  - Information system
  - Software
- **The process**
  - System development
  - Software engineering

- **System development ultimate objectives**
  - Problem solving
  - Supporting business processes
- **Participants and roles in system/software production**
  - Users
  - Mangers (owners)
  - Analysts
  - Designers
  - Programmers
  - Consultants

Factors Initiating Software Development

1. **Scenarios** initiating system/software development
   - **Real problem on the day-to-day task**
     - ✓ By end users
   - **Looking for effectiveness and efficiency**
     - ✓ By mangers
   - **Sensing new opportunities and technologies**
     - ✓ By technical personnel

2. The Inherent Nature of Software Complexity

- Some software systems are **not complex**.
- These are the largely forgettable applications that are specified, constructed, maintained, and used by the same person(Amateur or Professional).
- This is not to say that all such systems are crude and inelegant, nor do we mean to belittle their creators.
- Such systems tend to have a very limited purpose and a very short life span.

The Inherent Nature of Software Complexity

- More interested in the challenges of developing what we will call industrial-strength software.
  - Applications that exhibit a very rich set of behaviors
  - Applications that maintain the integrity of hundreds of thousands of records of information while allowing concurrent updates and queries.
  - Systems for the command and control of real-world entities, such as the routing of air or railway traffic.
- Such **Complex Software systems** tend to have a ***long life span***, *and* ***over time***, ***many users come to depend on their proper functioning***.

3. Distinguishing characteristic of industrial-strength software – Complex Systems

- More interested in the challenges of developing what we will call industrial-strength software.
- Complex systems are Intensely difficult, if not impossible for the individual developer to comprehend all the subtleties of its design.
- The complexity of such systems exceeds the human intellectual capacity.

Distinguishing characteristic of industrial-strength software – Complex Systems

- Unfortunately, this complexity we speak of seems to be an essential property of all large software systems.
- We may master how to solve this complexity, but we cannot avoid.
- Hence, the complexity of large software is an essential property, not an accidental one.

4. **Inherent complexity derives from four elements:**

- The **complexity** of the problem domain
  - Changing Requirement and Competing needs of diverse stakeholders
- The **difficulty** of managing the development process.
  - Team management and coordination
- The **flexibility** possible through software.
  - Flexibility to develop software the way you want it. labor-intensive. No strict standards like in engineering artifacts
- The **problems** of characterizing the behavior of discrete systems
  - Uncontrollable **inter-state phase** in using digital discrete systems which could corrupt our systems accidentally (if proper event interaction between the different objects is not carefully crafted)

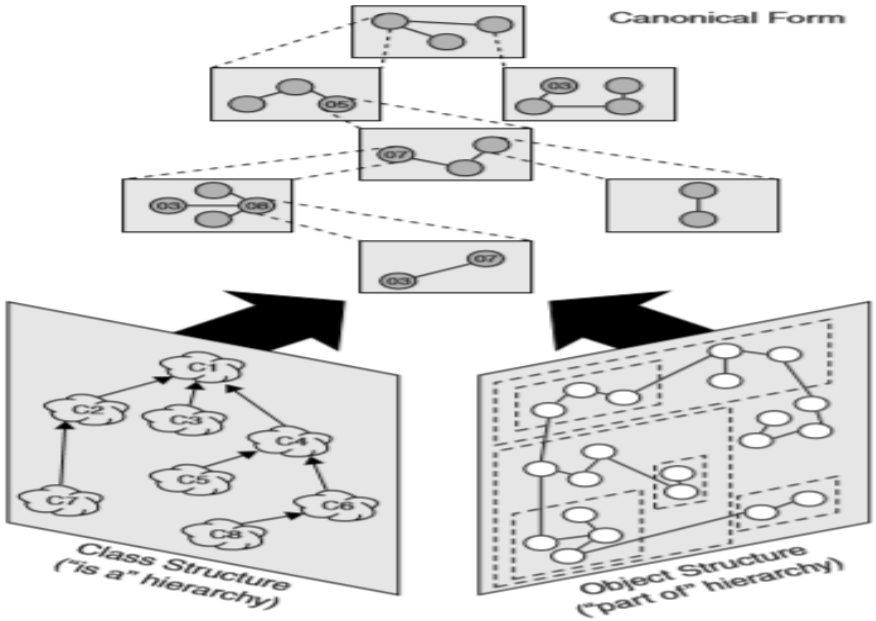
5. Attributes of complex systems

❖ **There are five common characteristics of Complex Systems.**

i. Hierarchic Structure

- Frequently, complexity takes the form of a hierarchy, whereby a complex system is composed of interrelated subsystems that have in turn their own subsystems, and so on, until some lowest level of elementary components is reached.(**System thinking)**
- The **architecture of a complex system** is a function of its components as well as the **hierarchic relationships** among these components.

Review of Systems, System thinking and Software



- **Hierarchical(Canonical) form of complex system**

- **Attributes of complex systems**

  ii. Relative Primitives

  - The nature of the primitive components of a complex system is *relative(subjective).*
  - The *choice of what components* in a system are primitive is relatively arbitrary and is largely up to the discretion of the observer of the system.
  - What is *primitive for one observer* may be at a *much higher level of abstraction* for another.

- **Attributes of complex systems**

  iii. Separation of Concerns

  - Hierarchic systems are decomposable because they can be divided into identifiable parts (*nearly decomposable because their parts are not completely independent*).
  - ***Intra-component linkages*** are generally stronger than ***inter-component linkages***.(Loose Coupling and High Cohesion)
  - This difference between *intra- and inter-component interactions* provides a *clear separation of concerns among the various parts of a system*, making it possible to study each part in *relative isolation*.

- **Attributes of complex systems**

  iv. Common Patterns

  - Hierarchic systems are usually composed of **only a few different kinds of subsystems in various combinations and arrangements**.
  - In other words, complex systems have **common patterns**.
  - These **patterns may involve the reuse of small components**.

- **Attributes of complex systems**
  - **v. Stable Intermediate Forms .**
    - Complex systems tend to *evolve over time*
    - A **complex system that works** is **consistently** found to have evolved from **a simple system that worked**
    - As systems evolve, **objects** that were once considered complex become the primitive objects on which more complex systems are built.

Solution for Software Complexity

- The complexity of the software systems we are asked to develop is increasing, yet there are basic limits on our ability to cope with this complexity.
  - Hence we need a mechanism to cope with this complexity.
  - The technique of mastering complexity has been known since ancient times:
- Hence, the solution is **Decomposition**

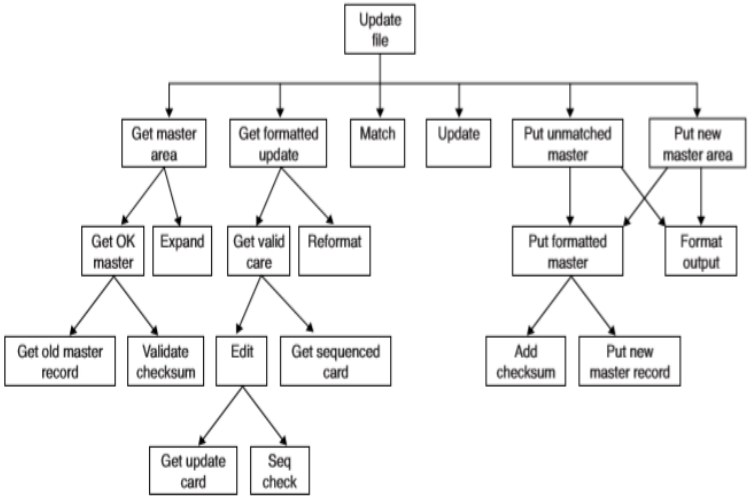
Solution for Software Complexity

- Two types of Decomposition
  - **Algorithmic** -top-down structured design, and so we approach decomposition – as a simple algorithmic decomposition where in each module in the system denotes a major step in some overall process.
  - **Object-Oriented** -according to the key abstractions (**concepts**) in the problem domain. Rather than decomposing the problem into steps such as *Get formatted update* and *Add checksum*,
  - we can identify objects such as **Master File(**the main that contains relatively permanent records about particular items or entries**)** and **Checksum(**digital fingerprint**),** which derive directly from the vocabulary of the problem domain

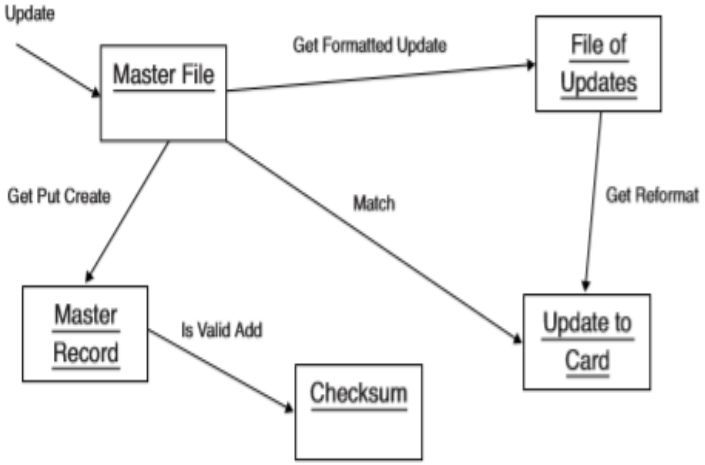
- **Solution for Software Complexity**
  - In summary, **algorithmic decomposition** is focused on the *processes involved in solving a problem*,
  - while **object-oriented decomposition** is focused on the *objects and data involved in the problem.*
  - Both approaches can be effective in solving complex problems, but they may be more or less appropriate depending on the specific problem being solved.

➢ OO Decomposition vs Algorithmic Decomposition

- Algorithmic Decomposition



OO Decomposition



❖**The algorithmic view** highlights the ordering of events.

❖**The object-oriented view** emphasizes the agents that either cause action or are the subjects on which these operations act.

Characteristics of Good Software

Product characteristics	Description
Maintainability	Software should be written in such a way so that it can evolve to meet the changing needs of customers. This is a critical attribute because software change is an inevitable requirement of a changing business environment.
Dependability and security	Software dependability includes a range of characteristics including reliability, security, and safety. Dependable software should not cause physical or economic damage in the event of system failure. Malicious users should not be able to access or damage the system.
Efficiency	Software should not make wasteful use of system resources such as memory and processor cycles. Efficiency therefore includes responsiveness, processing time, memory utilization, etc.
Acceptability	Software must be acceptable to the type of users for which it is designed. This means that it must be understandable, usable, and compatible with other systems that they use.

Software Failure Crisis

- *"The "software crises" came about when people realized the major problems in software development were … caused by communication difficulties and the management of complexity"*
- *Evidence over the years has shown that some of the most common reasons software projects fail center around poor or nonexistent communication between the key stakeholders.*
- It is a failure that occurs when the user perceives that the software has *ceased to deliver the expected result* with respect to *the specification input values.*
- The user may need to identify the severity of the levels of failures such as **catastrophic**, **critical**, **major** or **minor**, depending on their impact on the systems.

Software Quality and the Stakeholders View

Customer:

solves problems at

an acceptable cost in

terms of money paid and

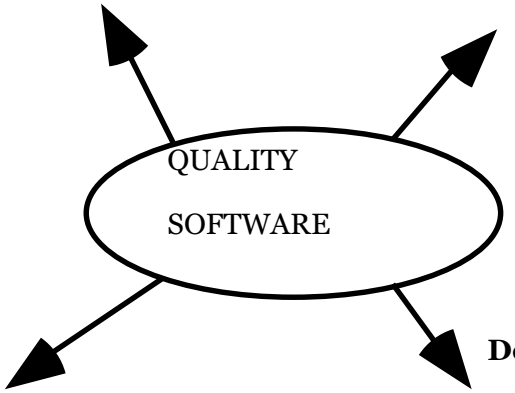
resources used

User:

easy to learn;

efficient to use;

helps get the work done



Developer:

easy to design;

easy to maintain;

easy to reuse its parts

Development manager:

sells more and

pleases customers

while costing less

to develop and maintain

Structured VS object oriented paradigm

Structured Paradigm:

- Structured methods for design were developed in the 1970s and 1980s and were the precursor to the UML and object-oriented design.
- Modeling process and data separately
- Suitable for small sized software

Object Oriented Paradigm

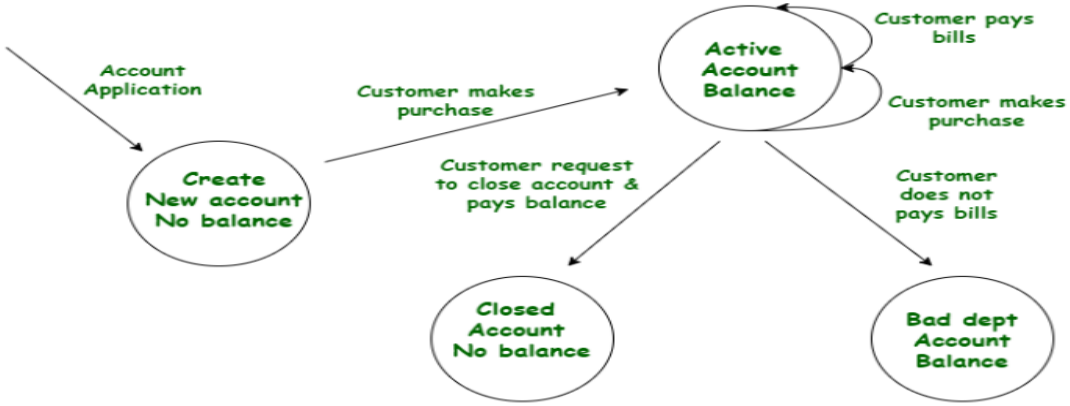
- Things are made up of objects
- Objects are identified as having data and function
- Is a software development strategy based on the idea of building systems/software from reusable components called objects

➢ Structured VS object oriented paradigm

❑ Structured Analysis and Design:

- ❖ **Focus on processes**: Structured Analysis focuses on the processes involved in a software system, modeling them as a series of connected steps.
- ❖ **Top-down approach**: Structured Analysis follows a top-down approach, breaking down complex systems into smaller, simpler parts that can be more easily understood.
- ❖ **Data-centered**: Structured Analysis focuses on the data that a software system manipulates, modeling it as data flows between processes.
- ❖ **Emphasis on functional decomposition**: Structured Analysis emphasizes the functional decomposition of a software system into smaller, independent functions.

- **Goals of SAD**
  - Improve quality and reduce the risk of system failure
  - Establish **concrete requirements specifications and complete requirements documentation**
  - Focus on **Reliability, Flexibility, and Maintainability** of **system**
    - **Example:**



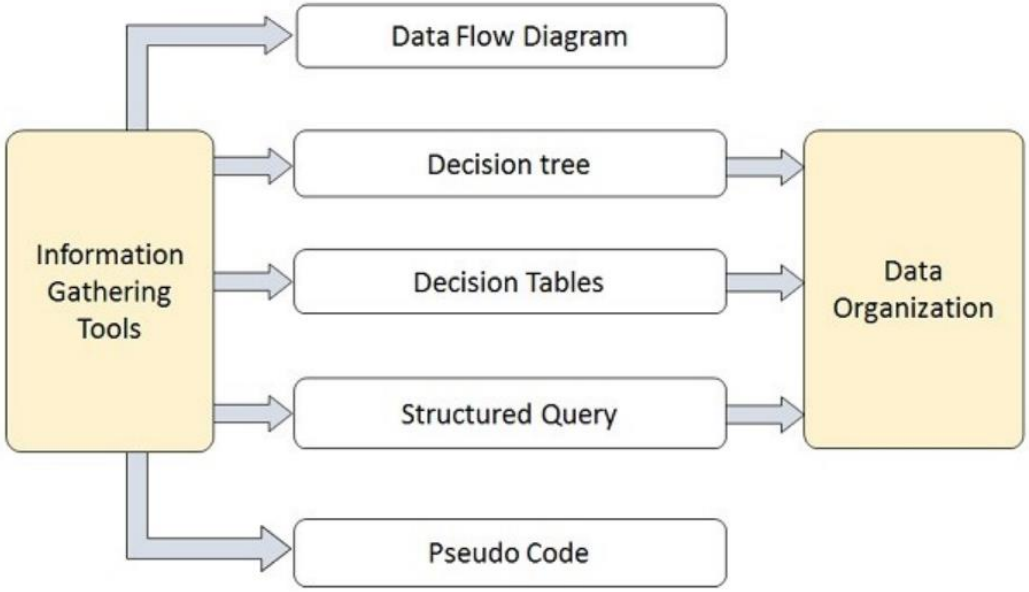
State Transition Diagram

- **Elements of Structured Analysis and Design**
  1. **Essential Model:** Model of what the system must do.
     - Does not define how the system will accomplish its purpose.
     - Is a combination of the **environmental and behavioral model**
       - i. **Environmental Model**
         - » Defines the scope of the proposed system.
         - » Defines the boundary and interaction between the system and the outside world.
         - » Composed of: **Statement of Purpose, Context Diagram, and Event List.**

- **Elements of Structured Analysis and Design**
  1. **Essential Model:** Model of what the system must do.
     - ii. **Behavioral Model**
       - Model of the internal behavior and data entities of the system.
       - Models the functional requirements.
       - Composed of **Data Dictionary, Data Flow Diagram, Entity Relationship Diagram, Process Specification, and State Transition Diagram**

- **Elements of Structured Analysis and Design**
  2. **Implementation Model:** Model of what the system must do.
     - Maps the functional requirements to the hardware and software.
     - Determines which functions should be manual and which should be automated.
     - Defines the Human-Computer Interface.
     - Defines non-functional requirements.
     - Tool: Structure Charts

Modern Development Technologies



Structured Analysis Tools

Modern Development Technologies

OBJECT-ORIENTED SYSTEMS ANALYSIS AND DESIGN
SECOND EDITION

- **Advantages of structured analysis and design**
  - **Visual**, so it is easier for users/programmers to understand
  - Makes good use of **graphical tools**
  - A **mature technique**
  - **Simple and easy** to understand and implement

Disadvantages of Structured analysis and design

- Not enough **user-analyst** interaction
- It depends on **dividing system to sub systems** but it is hard to decide when to stop decomposing.
- **Limited scalability:** Structured Analysis is limited in its scalability, and may become cumbersome when dealing with complex, large-scale systems.
- **Lack of object orientation:** Structured Analysis does not provide the object orientation and
- **Encapsulation benefits of OOA,** making it more difficult **to manage and maintain large systems** over time.
- **Limited ability to model complex relationships:** Structured Analysis has a limited ability to model complex relationships between objects, making it less suitable for modeling large, complex systems.

➢ Structured VS object oriented paradigm

❑ Object-Oriented Analysis (OOA):

❖ The **main concepts** behind the object-oriented paradigm is that

- Instead of defining systems as two separate parts (**data and functionality**), **you now define systems as a collection of interacting objects.**
- **Objects do things** (that is, they have functionality) and they **know things**(they have data).
- It requires a different way of thinking about **decomposition** –**OO Decomposition** as opposed to algorithmic(functional) decomposition.

➢ Structured VS object oriented paradigm

❑ Object-Oriented Analysis (OOA):

- **Focus on objects**: OOA focuses on the objects involved in a software system, modeling them as instances of classes that encapsulate both **data and behavior**.
- **Bottom-up approach**: OOA follows a bottom-up approach, building complex systems from smaller, simpler objects that can be more easily understood.
- **Object-centered**: OOA focuses on the ***objects*** that make up a software system, modeling their relationships and interactions.
- Emphasis ***on object-oriented design patterns***: OOA emphasizes the **reuse** of **objects** and **object-oriented design patterns**, **reducing the amount of code** that needs to be written and **improving the quality and consistency of the software.**

Structured VS object oriented paradigm

Object-Oriented Analysis (OOA):.

Advantages of Object-Oriented Analysis (OOA):

- **Reusable code:** OOA enables the creation of reusable objects and design patterns, reducing the amount of code that needs to be written and improving the quality and consistency of the software.
- **Scalability:** OOA is more scalable than Structured Analysis, making it better suited for large, complex systems.
- **Object orientation:** OOA provides the benefits of object orientation, including **encapsulation, inheritance, and polymorphism, making it easier to manage and maintain large systems over time.**
- **Better modeling of complex relationships:** OOA enables better modeling of complex relationships between objects, making it better suited for modeling large, complex systems.

Structured VS object oriented paradigm

Object-Oriented Analysis (OOA):.

Disadvantages of Object-Oriented Analysis (OOA):

- **Steep learning curve:** OOA can be more difficult to understand and implement than Structured Analysis, especially for those who are not familiar with object-oriented programming.
- **Bottom-up approach:** The bottom-up approach of OOA can make it difficult to understand the high-level functions and processes involved in a software system, and to break them down into smaller, more manageable components.
- **Emphasis on objects:** OOA places a strong emphasis on objects, making it more difficult to understand the relationships between data and processes.

➢ Structured VS object oriented paradigm

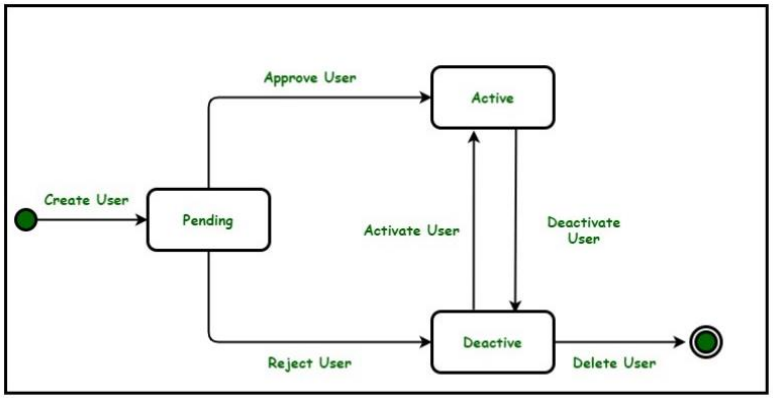
❖ Object-Oriented Analysis (OOA):

- ❖ In conclusion, ***Structured Analysis and OOA are both valuable software development methodologies***, each with its ***own strengths and weaknesses***.
- ❖ The choice of which to use depends on the particular requirements and constraints of a software project.

➢ Structured VS object oriented paradigm

❑ Object-Oriented Analysis (OOA):

❖ **Example**



State Chart Diagram

➢ Structured VS object oriented paradigm

➢Structured Analysis	➢Object-Oriented Analysis
➢ The main focus is on the process and procedures of the system.	➢ The main focus is on data structure and real-world objects that are important.
➢ It uses System Development Life Cycle (SDLC) methodology for different purposes like planning, analyzing, designing, implementing, and supporting an information system.	➢ It uses Incremental or Iterative methodology to refine and extend our design.
➢ It is suitable for well-defined projects with stable user requirements.	➢ It is suitable for large projects with changing user requirements.
➢ Risk while using this analysis technique is high and reusability is also low.	➢ Risk while using this analysis technique is low and reusability is also high.
➢ Structuring requirements include DFDs (Data Flow Diagram), Structured Analysis, ER (Entity Relationship) diagram, CFD (Control Flow Diagram), Data Dictionary, Decision table/tree, and the State transition diagram.	➢ Requirement engineering includes the Use case model (find Use cases, Flow of events, Activity Diagram), the Object model (find Classes and class relations, Object interaction, Object to ER mapping), Statechart Diagram, and deployment diagram.
➢ This technique is old and is not preferred usually.	➢ This technique is new and is mostly preferred.

Similarities

- Both SAD and OOSAD had started off from **programming techniques**.
- Both techniques use **graphical design and graphical tools to analyze and model the requirements.**
- Both techniques provide a systematic step-by-step process for developers
- Both techniques focus on documentation of the requirements.

Object Technology

- **Object Technology (OT)** has been a part of IS application development strategies for over twenty years.
- **OT** has been incorporated into software development processes with greater benefits experienced as a function of its more pragmatic application.
- It is an **umbrella** term for **object-oriented programming, object-oriented databases and object-oriented design methodologies**

Object Technology

- Object-oriented technology (OOT) is **a software design model in which objects contain both data and the instructions that work on the data**.
- Object Basics:

Objects:

- The term Object means a combination of data and logic that represents some real world entity.
- When developing an object – oriented application, two basic questions always arise:
  1. What objects does the application need?
  2. What functionality should those objects have?

Object Technology

- There are **string objects, output objects, input objects, graphics objects, date objects, time objects, audio objects, video objects**, etc.
- Almost any **noun** can be reasonably represented as a software object in terms of **attributes** (e.g., **name, color and size**) and **behaviors** (e.g., **calculating, moving and communicating**).

Object Technology

- In the following, we explain a set of object-oriented programming concepts by drawing analogy to a car.
  - ✓ **Attributes and Instance Variables**
  - ✓ **Methods and Classes**
  - ✓ **Instantiation**
  - ✓ **Reuse**
  - ✓ **Method Calls**
  - ✓ **Encapsulation**
  - ✓ **Inheritance**
  - ✓ **Polymorphism**

Object Technology

✓ Attributes and Instance Variables

- A car has many attributes, such as color, its number of doors, the amount of gas in its tank, its current speed and its record of total miles driven (i.e., its odometer reading), etc.
- These include the physical parts as well as the information about the parts.
- The car's attributes are specified in its blueprint.
- Every car has its own attributes. Each car knows how much gas is in its own gas tank, but not how much is in the tanks of other cars.

Object Technology

✓ Attributes and Instance Variables

- A Java object also has **attributes**.
- These attributes are information items that the object keeps while it is used in a program.
- The object keeps these attributes in its assigned memory which is organized like a C struct.
- Attributes are specified in the class of the object as **instance variables**.
- **Each object knows its own attributes, but not the attributes of other objects**.

Object Technology

- Using a **modular, object-oriented design and implementation** approach can make software-development **more productive.**
- **Object-oriented programs** are often easier to understand, correct and modify.

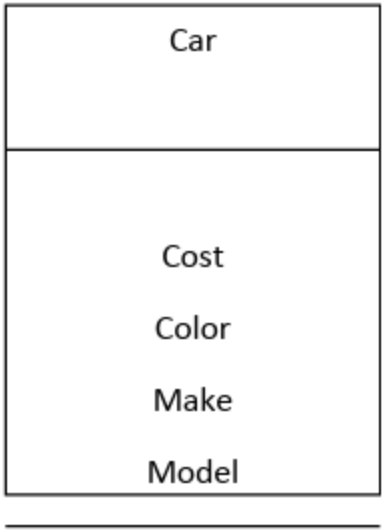
Example:

- In the real-world, there are many analogies of objects.
- A car is an object that can be used to transport people and goods.

Object Technology

Object state and properties (attributes):

- **Properties** represent the state of an object.
- Example: the properties of a car, such as **color**, **manufacturer**, **Model** and **cost**, are abstract descriptions.
- **The attributes of a Car object**



- For **color**, we could choose to use a sequence of characters such as **red**, etc

Object Technology

Object state and properties (attributes):

- A car has an engine, four wheels, a number of doors, and a lot of other mechanical and electrical parts.
- These parts are hidden under the hood(cover).
- A driver controls the car using a steering wheel, the gas and brake pedals, and many buttons and switches on the dashboard.
- If you are driving a car and want to make it go faster, you press its **gas pedal** with your foot.

Object Technology

- The specific car you drive is one of many cars of a particular model that were made by a car factory according to a specific **design or blueprint.**
- The **blueprint** of the car may contain a set of engineering drawings and documents that specify the details of building the car.
- Specifically, **the blueprint** will include the steering wheel, gas and brake pedals, etc.

Object Technology

- The gas pedal hides from the driver, or the user of the car,
- the complex mechanisms that actually make the car go faster,
- the brake pedal hides the mechanisms that slow the car, and
- the steering wheel hides the mechanisms that turn the car.
- This enables people with little or no knowledge of how engines, braking and steering mechanisms work to drive a car easily and safely.

Object Technology

- To be able to use a car to travel, the car has to be made by the factory.
- In computer software, an **object** is used to keep and modify information.
- It is also made according to a **blueprint** and used by a user (which is typically another program).

Object Technology

OBJECTS ARE GROUPED IN CLASSES

- **A class** is a specification of structures (instance variables), behavior (methods), and inheritance for objects.
- **Classes** are important mechanisms for classifying objects.
- In an **object- oriented system**, a method or behavior of an object is defined by its class.
- Each object is an instance of a class.
- There may be many different classes in a system

Object Technology

✓ Methods and Classes

- In Java, a class is a blueprint of a set of objects.
- It describes how the object is made of and what tasks the object is able to perform.
- The performing of a task by a Java object is **a method**.
- The **method** contains a sequence of Java statements that actually perform the task.
- A **method** hides these statements from its user, just as the gas pedal of a car hides from the driver the mechanisms of making the car go faster.

Object Technology

Methods and Classes

- In Java, **a class** is a programming module containing the methods that perform the class's tasks.
- Thus, the class is similar in concept to a **car's design**, which specifies the "**methods**" **to accelerate, steer, brake, and so on**

Object Technology

Methods and Classes

- **Behavior refers the Object Behavior or methods)**
- We can **drive** a car, we can **ride** an elephant, or the elephant can **eat** a peanut. Each of these statements is a description of the **object's behavior**.
- In the object model, **object behavior** is described in **methods or procedures**.
- A **method** implements the behavior of an object

Object Technology

Messages (Objects respond to messages):

- An object's capabilities are determined by the methods defined for it. Methods conceptually are equivalent to the functions definitions used in procedural languages.
- Objects perform **operations** in response to **messages**.
- For example, when you press on the brake pedal of a car, you send a stop message to the car object.

Object Technology

Messages (Objects respond to messages):

- **Messages** essentially are nonspecific function calls
- A **message** is different from a **subroutine call**, since different objects can respond to the same message in different ways.
- For example, **cars, motorcycles, and bicycles** will all respond to a **stop message**, but the actual operations performed are **object specific**.

Object Technology

Information Hiding (and Encapsulations):

- **Information hiding** is the principle of concealing the **internal data and procedures** of an object and providing an interface to each object in such a way to reveal as little as possible about its inner workings.
- On object is said to **encapsulate** the data and a program. This means that **user cannot see the inside of the object "capsule,"** but can use the object by **calling the object's methods**.

Object Technology

Information Hiding (and Encapsulations):

- **Encapsulation** or **information hiding** is a design goal of an object-oriented system.
- Rather than allowing an object direct access to another object's data, a message is sent to the target object requesting information.
- Another issue is **per-object or per-class protection.**
- In **per-class protection**, the most common form class methods can access any object of that class and **not just the receiver**.
- In **per-object protection**, **methods can access only the receiver**.
- **A car engine** is an example of **encapsulation**.

Object Technology

✓ Encapsulation

- Just as a car hides its parts under the hood, an object encapsulates (i.e., hides) its instance variables and the statements of its methods.
- Ideally, objects are not allowed to directly modify or access each other's variables except through method calls.
- The implementation details should be hidden within the objects themselves.
- **Information hiding** is crucial to good software engineering..

Object Technology

Class Hierarchy:

- An object-oriented system organizes **classes into a subclass- super class hierarchy.**
- Different behaviors are used as the basics for making distinctions between classes and subclasses.
- At the top of the Class hierarchy are the most general classes and at the bottom are the most specific.
- A **subclass inherits all of the properties and methods defined in its superclass**

Object Technology

✓ Instantiation

- Just as a factory has to construct a car from its blueprint before you can actually drive a car,
- a Java program must ask the Java compiler to **construct an object** of a class before it can make method calls to the object.
- This is called the **instantiation** of an object, and
- hence an **object is referred to as an instance of its class**.

Object Technology

✓ Instantiation

- Technically, the instantiation of a Java object will allocate memory for the object **to hold information and identify code sections that implement the methods.**

✓ Reuse

- Just as a car's blueprint can be reused many times to build many cars, a Java class can be reused many times to build many objects.

Object Technology

Reuse

- Reusable classes save programming time and effort and helps you build more reliable and effective systems.
- This is because existing classes often have gone through extensive testing, debugging and performance tuning.
- Just as the notion of interchangeable parts was crucial to the Industrial Revolution**, reusable classes are crucial to the software revolution** that has been spurred by object technology.

Object Technology

✓ Reuse

- The reuse of a class is not limited to the original design.
- A class can be extended to add new constructs or modified to perform new tasks.

Object Technology

Method Calls

- When you drive a car, pressing its gas pedal sends a message to the car and makes it to perform a task, namely, to go faster.
- Similarly, when using a Java object,
- you make method calls to an object.
- A method call **sends a message to the object to perform a task by executing the method's statements**.
- The message may include additional information that is needed to perform the task.

Object Technology

✓ Inheritance

- A new class of objects can be programmed quickly and conveniently by **inheritance**, that is, extending existing classes.
- The new class created in this way will have **the variables and methods** of an existing class, possibly customizing them and adding unique ones of its own.
- In our car analogy, a convertible car of a model can be an modification of the model so that the roof can be raised or lowered, but the other parts are the same.
- So the blueprint of the convertible is an extension of the more general blueprint of that model.

Object Technology

Polymorphism

- Often, a method does not apply just to a specific class of objects, but to a more general class.
- For example, a speed limit usually applies to all vehicles on the road, whether they are cars, trucks, motorcycles, etc.
- Thus, drivers of convertibles can determine whether they are driving within the speed limit because a convertible is a type of car, which is a type of vehicle.
- Code that is written for a general class is polymorphic and will usually depend on methods that are common to all the specific classes. In Java, an interface is often used to specify the names, parameters and return types of these methods.

The OO Software Process Models

Software(Information system) Process:

- A structured set of activities required to develop a software system(product).
- The steps we go through to develop a software product is called **a software process.**

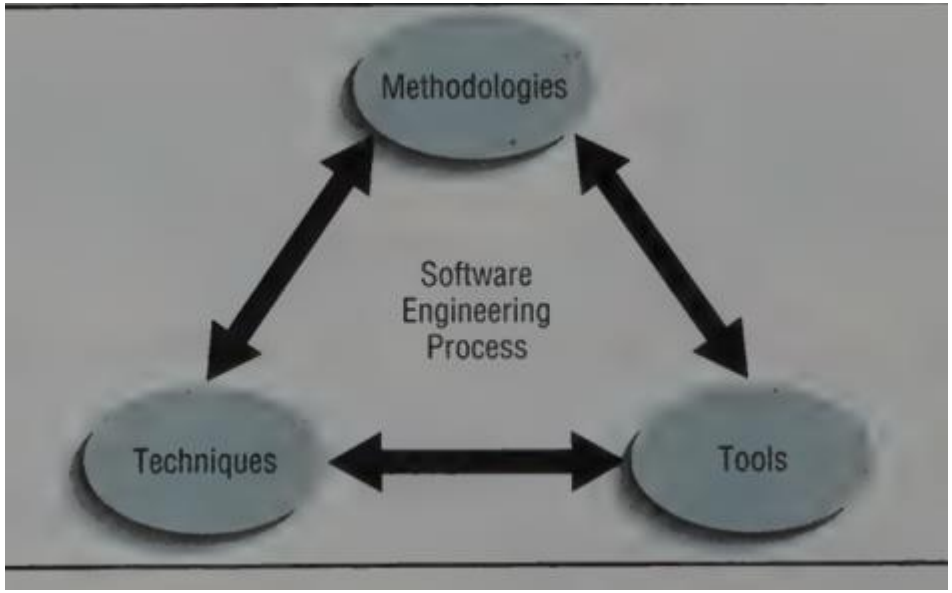
The OO Software Process Models

Information system(Software) development Process:

- Our goal is to help you understand and follow the systems development process that leads to **the creation of information systems**.
- The Software Development Process Uses **Methodologies**, **Techniques**, and **Tools**
- **Methodologies**, **techniques**, and **tools** are **central to systems development processes.**

The OO Software Process Models

Information system(Software) development Process:



The OO Software Process Models

Information system(Software) development Process:

- Methodologies are a sequence of step-by-step approaches that help in the development of the final product: **the information system**.
- Most methodologies incorporate several development techniques, such as **direct observations and interviews with users of the current system**.
- Techniques are processes that analysts follow to help ensure that their work is well thought out, complete, and comprehensible.

The OO Software Process Models

Information system(Software) development Process:

- **Techniques** provide support for a wide range of tasks including conducting thorough interviews with current and future users of the information systems to determine what the system should do, planning and managing the activities in a systems development project, diagramming
- How the system will function, and designing the reports, such as invoices, that the system will generate for its users to perform their jobs.

The OO Software Process Models

Information system(Software) development Process:

- **Tools** are computer programs, such as computer-aided software engineering (CASE) tools, that make it easy to use specific techniques. These three elements— **methodologies, techniques, and tools—work together to form an organizational approach to systems analysis and design.**

The OO Software Process Models

What is a model?

- **A model** is an abstract representation of something real or imaginary.
- Like maps, models represent something else.
- They are useful in several different ways, precisely because they differ from the things that they represent.
- **A model** is quicker and easier to build.

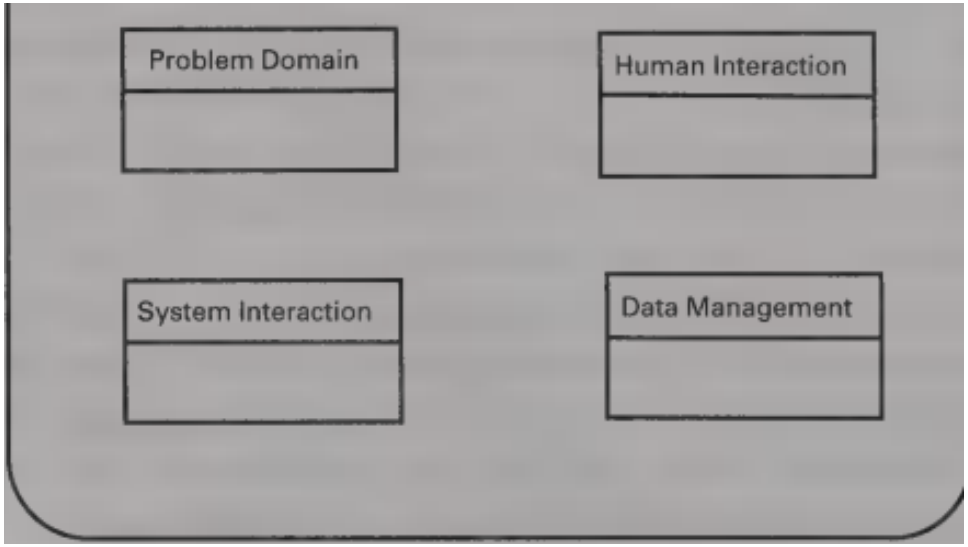
The OO Software Process Models

What is a model?

- **A model** can be used in simulations, to learn more about the thing it represents.
- **A model** can evolve more about **a task or problem**. We can choose which details to represent in a model, and which to ignore.
- **A model** is **an abstraction**
- **A model** can represent **real or imaginary things from any domain**.

The OO Software Process Models

- Information System Object Model



- The three components of the object **model—human interaction, data management, and system interaction**

The OO Software Process Models

- **Modeling, as a methodology** is the systematic representation of the relevant features of a product or a system from particular perspectives.
- **Modeling** uses a set of **techniques, methods and a language** of specialized notations to produce **artifacts** in a variety of media.
- Each product needs a certain amount of modeling before the development can start.
- The real world consists of objects, and development tools have now become object-oriented as well.

The OO Software Process Models

- But what about the process that takes us from the requirements of the real world to the features of the information system?
- This process consists of two activities:
  - **analysis**, which discovers the concepts of the real world and builds a conceptual model of the product, and
  - **design**, which develops the results of analysis into a concrete model for building the system.

The OO Software Process Models

- The main motive for adopting an object-oriented approach to analysis and design is the **challenge of complexity**, both the **complexity of requirements and the complexity of the product** that must satisfy those requirements
- **Complex products (Information system),** regardless of the field to which they belong, **need modeling**
- What **object-oriented analysis and design requires** is a **language** that can **model objects, classes, encapsulation, inheritance, and other object-oriented concepts**.

➢ The OO Software Process Models

❖ Unified Modeling Language (UML)

- ❖ **UML** is a modeling language for object-oriented system analysis, design, and deployment. UML is not a product, nor is it a process or a methodology.
- ❖ **UML** used as a modeling language for analysis and design.
- ❖ **UML** is a language for object-oriented modeling. UML's notation—its system of figures and symbols—is designed to represent object-oriented concepts

The OO Software Process Models

- **Unified Modeling Language (UML)**
- **UML provides the "primitives" (or the basic elements) for building object-oriented concept (analysis) and concrete (design) models.**
- UML is not a **specific product, process, or methodology,**
- but **a notational system** that allows **analysts, designers, and developers to adopt its diagrams to the task at hand, detailed or general, conceptual or concrete.**
-

➢ The OO Software Process Models

➢ Unified Modeling Language (UML)

- ❖ **UML** supports multiple views of same system, with varying degrees of detail or generalization as needed

1. **Owner's View**: what the owner (or business) wants, or the conceptual view of the system.
2. **Architect's View:** how the architect conceives the solution, or the logical view of the system.
3. **Builder's View:** the **blueprints** for building the product, or the **physical view of the system.**

➢ The OO Software Process Models

➢ Unified Modeling Language (UML)

❖ To achieve these tasks, UML embodies four properties

i. Visualization

✓ **UML** diagrams visualize system **components, their relationships, and their interactions.**

✓ **UML** provides of **a set of graphical elements** that are combined to form **diagrams**.

✓ Each diagram is **a visual presentation or view of the system** and **satisfies one or more broad but overlapping types of modeling:**

- **Behavioral modeling** represents the interaction of the system with the outside world.
- **Structural modeling** represents the components of the system and their interrelationships.
- **Dynamic modeling** represents how the components of the system interact with the outside world and with each other to satisfy the behavioral requirements of the system.

The OO Software Process Models

Unified Modeling Language (UML)

- To achieve these tasks, UML embodies four properties

ii. Specification

- **UML** provides precise and complete models for the **three major activities of system development**: **analysis, design, and implementation**

iii. Construction

- **UML** models are compatible with object-oriented language
- While UML is not a programming language, its models can be mapped to object oriented programming languages. With varying degrees of effectiveness and ease, UML-based modeling and development tools allow both **forward-engineering (generating code from models) and backward-engineering (generating or modifying models from code).** The result is that **any changes to the visual model can be traced back to code, and vice versa**

➢ The OO Software Process Models

➢ Unified Modeling Language (UML)

- ❖ To achieve these tasks, UML embodies four properties

iv. Documentation

- ❖ UML modeling tracks major development activities throughout the system life cycle.
- ❖ The advantage of UML for documenting system development is that UML modeling is **not a separate activity**, but **an organic part of the development project.**

The OO Software Process Models

Information system(Software) development Process:

Methodology

- Development of a product must follow a set of **practices, procedures, rules, and techniques.**
- **Product development** needs a **methodology**, one that is relevant to the task at hand, is flexible, and can accommodate new experiences.
- Information system development is also a development process, must follow a methodology,
- **Methodology** is a variable blend of two sources:
  - **a systematic generalization** and
  - **An abstraction of lessons learned from the past and ideas for improving on the past.**

The OO Software Process Models

- **Information system(Software) development Process:**
  - **Methodology**
    - **Methodology** is both the **most abstract** and **the most systematic guide to action**.
    - **Complexity** is the primary source behind the emergence or the change of methodology
    - **Methodology** is needed **not only in creating the solution but also in understanding the problem, organizing the production, assuring quality, and managing the consequences of the solution**

The OO Software Process Models

Information system(Software) development Process:

Methodology

- Methodology is:-
  - a set of **methods, rules, practices, procedures, techniques, and tools** used to achieve a goal; or build a product.
  - **the theoretical understating of the principles** that **determine how such methods, practices, tools,** etc., are used.
  - we first need to know certain things about the essence of methodology itself:
    - ✓ what it is,
    - ✓ where it comes from,
    - ✓ what it addresses,
    - ✓ what its benefits are, and
    - ✓ what risks it involves.

The OO Software Process Models

Information system(Software) development Process:

Methodology

- **Methodology** results from abstracting and organizing experience within a theoretical framework.
- In other words, "**method**" defines a tactic, whereas "**methodology**" defines the strategy.
- **Complexity** is the primary source behind the emergence or the change of methodology.
- **Methodology** is needed not only in creating the solution but also in **understanding the problem, organizing the production, assuring quality, and managing the consequences of the solution.**

The OO Software Process Models

Information system(Software) development Process:

Methodology

- **Methodology** results from abstracting and organizing experience within a theoretical framework.
- The methodology to assure the quality of the product must be integrated with the methodology for its development.
- The size of a project is often a deciding factor in selecting or shaping the methodology.
- The process of building a solution may require more than one methodology to succeed.

The OO Software Process Models

Information system(Software) development Process:

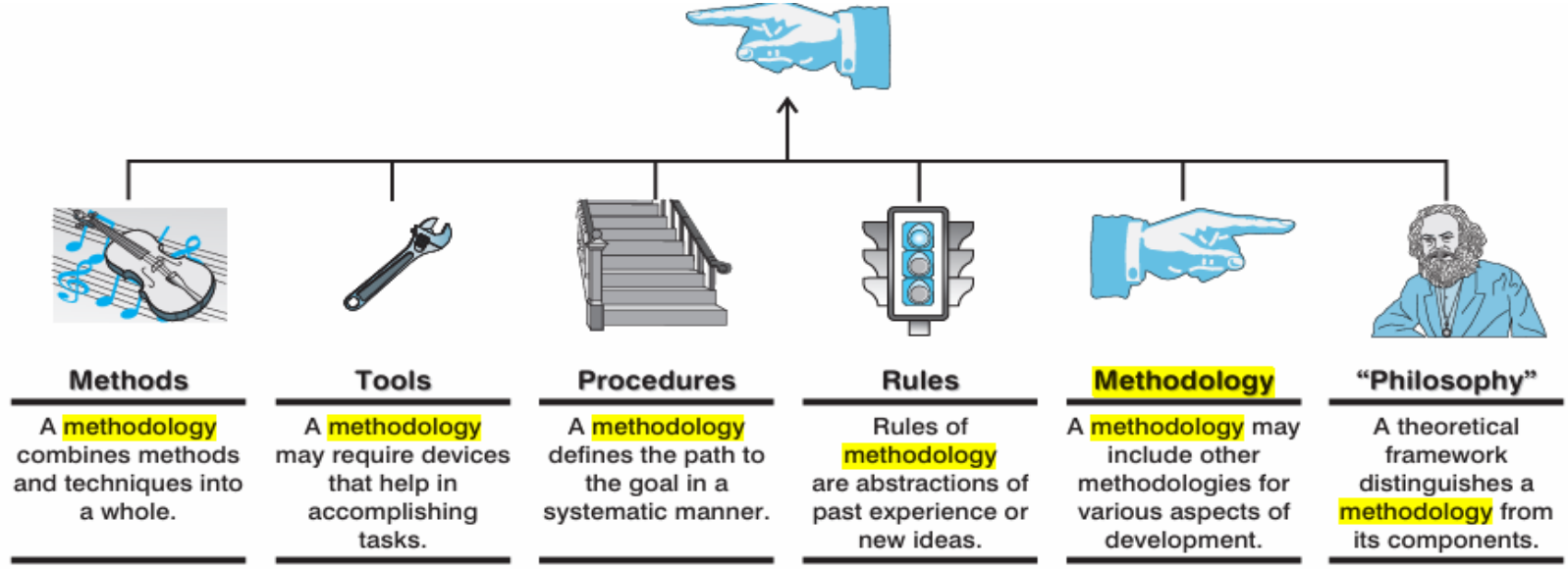
Methodology

- **The golden rule** that any methodology should be applied prudently, the fact is that one methodology **cannot handle every aspect of a complex undertaking, even if that methodology is the best and the most appropriate one.**
- **A good methodology** allows project management to verify the quality of the product at critical milestones during the development process, not just at the completion of the project

The OO Software Process Models

Information system(Software) development Process:

- **Methodology is the "How" of Building Solutions**



- **No methodology is perfect, but "no methodology means any strategy."**

- **The OO Software Process Models**
  - **Information system(Software) development Process:**
    - **Methodology**
      - **Methodology** is then a systematized interpretation of the past experience, and often new ideas for improving on the past, that provides a set of methods, rules, procedures, and tools to achieve a goal.

The OO Software Process Models

Information system(Software) development Process:

What Do Methodologies Address?

- Many activities must be carried out to turn the idea of software into actual software.
- These activities can be broadly classified as follows:

1. Gathering requirements	6. Implementation,
2. Feasibility study	7. Quality assurance and testing
3. Domain analysis,	8. Deployment and training, and
4. System analysis,	9. Maintenance
5. Design,	

The OO Software Process Models

Information system(Software) development Process:

Software Process Patterns

A pattern is the core of the solution to a problem that occurs repeatedly in an environment.

Patterns are old. **A systematic way to describe patterns and their consequences is new**

An object-oriented pattern is an abstraction of a **doublet, triplet, or other small grouping of classes** that is likely to be helpful again and again in object-oriented development.

The OO Software Process Models

Information system(Software) development Process:

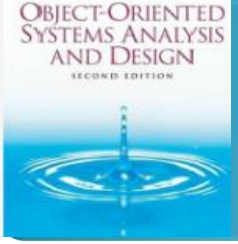
Software Process Patterns

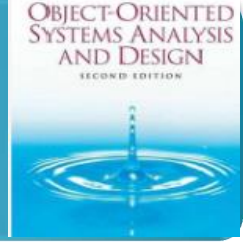
An information system is an aggregation of several subsystems, no single pattern or guideline is likely to apply to all subsystems.

Usually, when software development and architectural patterns are discussed together, the term should be taken to mean **"software patterns" or "application patterns."**

Object-Oriented Analysis and Design

- How will you create the code for your programs?
- You should follow a detailed analysis process for determining your project's requirements (i.e., defining what the system is supposed to do).
- You should develop a design that satisfies them (i.e., deciding **how the system should do it).**





Object-Oriented Analysis and Design

- Object-oriented analysis and design (OOAD) views a systems as a **group of interacting objects.**
- Object-oriented programming (OOP) allows you to **implement an object-oriented design as a working system.**
- It provides a compiler that allows a programmer to define **classes**.
- **The Object-Oriented development life cycle** consists of progressively developing an object representation through three phases ***analysis, design, and implementation.***

Object-Oriented Analysis and Design

Analysis Phase: At this stage

- The *problem is formulated*, *user requirements are identified*, and then a *model is built* based upon real–world objects.
- The analysis *produces models* on how the desired system should function and how it must be developed.
- The models do not include any implementation details so that it can be understood and examined by any non–technical application expert.
  - Model of the real-world application is developed showing its important properties.
  - Model specifies the functional behavior of the system independent of implementation details

- **Object Technology**
  - ✓ **Object-Oriented Analysis and Design**
- **Design Phase**
  - ❖ **Analysis model** is refined and adapted to the environment
  - ❖ Can be separated into two stages
    1. **System design**
       - done according to both the system analysis model and the proposed system architecture.
       - the emphasis is on the objects comprising the system rather than the processes in the system.
       - Concerned with overall system architecture

- **Object Technology**
  - ✓ **Object-Oriented Analysis and Design**
- **Design Phase**

2. Object design

- **Object design model** is developed based on both the models developed in the system analysis phase and the architecture designed in the system design phase.
  - All the classes required are identified.
  - The designer decides whether: new classes are to be created from scratch,
  - any existing classes can be used in their original form, or new classes should be inherited from the existing classes.

- **Object Technology**
  - ✓ **Object-Oriented Analysis and Design**
- **Design Phase**

2. Object design

- The associations between the identified classes are established and the hierarchies of classes are identified.
- Besides, the developer designs
  - **the internal details of the classes and their associations**,
  - **the data structure for each attribute and the algorithms for the operations.**

- **Object Technology**
  - ✓ **Object-Oriented Analysis and Design**
- **Implementation and Testing Phase**
  - In this stage, the design model developed in the object design is **translated into code in an appropriate programming language or software tool**.
  - The databases are created and the specific hardware requirements are ascertained.
  - Once the code is in shape, it is tested using specialized techniques to identify and remove the errors in the code.
  - **Design** is implemented **using a programming language or database management system**

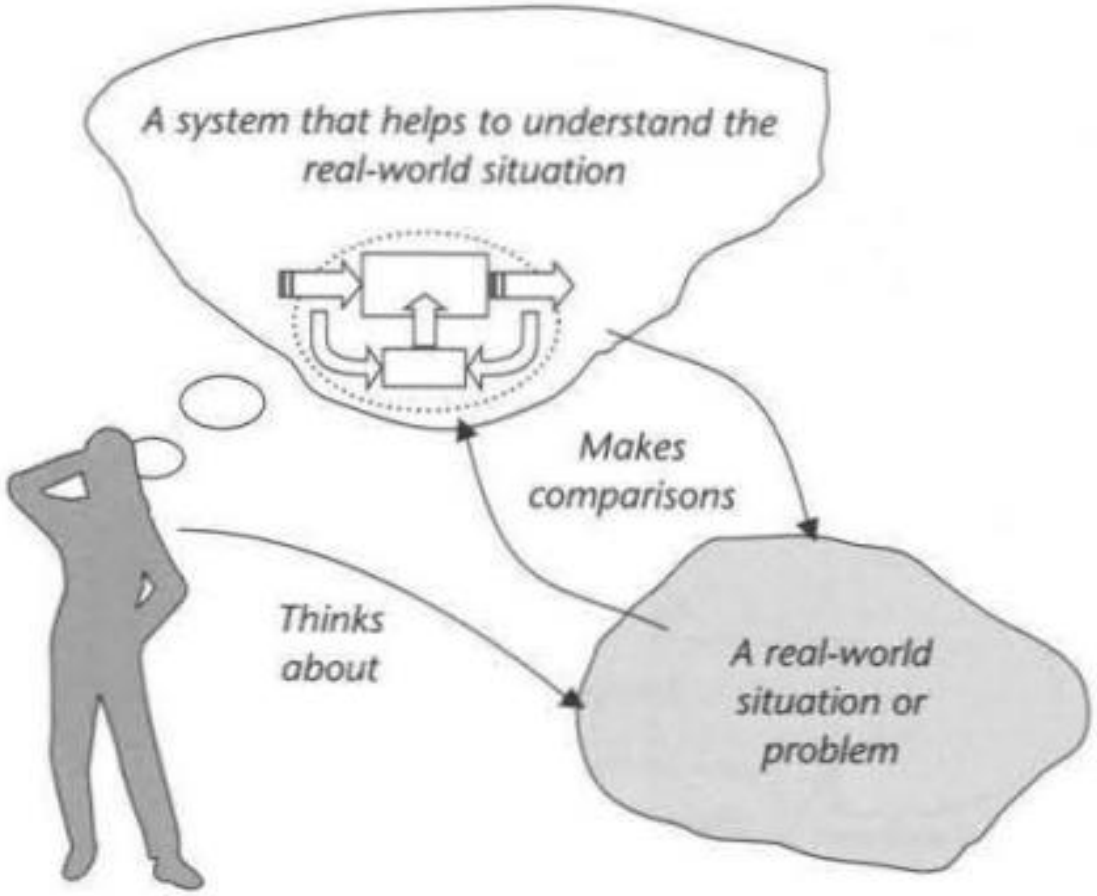
Summary

- Software is inherently complex; the complexity of software systems often exceeds the human intellectual capacity.
- The task of the software development team is to engineer the illusion of simplicity.
- Complexity often takes the form of a hierarchy; it is useful to model both the "is a" and the "part of" hierarchies of a complex system.
- Complex systems generally evolve from stable intermediate forms.
- There are fundamental limiting factors of human cognition; we can address these constraints through the use of decomposition, abstraction, and hierarchy.

Summary

- Complex systems can be viewed by focusing on either things or processes; there are compelling reasons for applying object-oriented decomposition, in which we view the world as a meaningful collection of objects that collaborate to achieve some higher-level behavior.
- Object-oriented analysis and design is the method that leads us to an object oriented decomposition; object-oriented design uses a notation and process for constructing complex software systems and offers a rich set of models with which we may reason about different aspects of the system under consideration

Modern Development Technologies



The relationship between system and reality


End of
Unit One