CS-3004
SOFTWARE DESIGN AND ANALYSIS

RUBAB JAFFAR
RUBAB.JAFFAR@NU.EDU.PK

Introduction

Overview and Basics

Lecture # 1, 2, 3

TODAY'S OUTLINE

- Administrative Stuff
- Overview of CS-3004
- Basic Concepts---- Must have a grasp
- Software Design and its goal?
- Why software design is important?
- Summary of the lecture

ADMINISTRATIVE STUFF

OFFICE HOURS

- Office 6 (CS building)
- Office hours: 3:00 to 4:00 pm
  - (Monday, Thursday, Fridays)

OVERVIEW OF CS-3004
ABOUT THE COURSE

- Study application of a software engineering approach that models and designs a system as a group of interacting objects.
- Various cases studies will be used throughout the course to demonstrate the concepts learnt.
- A strong in class participation from the students will be encouraged and required during the discussion on these case studies.

OVERVIEW OF CS-3004

MAJOR TOPICS OF THE COURSE

- Course Introduction
- SDLC
- UML Relationships
  - • Association
  - • Aggregation
  - • Composition
- UML Packages
  - • Use Case
  - • Class Diagram
  - • Activity Diagram
  - • Collaboration Diagram
  - • Sequence Diagram
  - • State Transition Diagrams
  - • Timing Diagram
  - • Implementation Diagrams
- Design Patterns

OVERVIEW OF CS-3004

PRE-REQUISITES /KNOWLEDGE ASSUMED

- Programming language concepts
- Data structure concepts

OVERVIEW OF CS-3004

SKILLS ASSUMED

- We assume you have the skills to code in any programming language therefore you can design, implement, test, debug, read, understand and document the programs.

OVERVIEW OF CS-3004

TENTATIVE MARK DISTRIBUTION AND GRADING POLICY

- Assignments/Class Participation: 10 %
- Mid Exams: 30 %
- Final: 50%
- Presentations + Projects: 10%
- Absolute Grading Scheme

COURSE ETHICS

Projects/Assignments

- Deadlines are always final
- No credit for late submissions
- One student per assignment at maximum

Honesty

- All parties involved in any kind of cheating in any assessment will get zero in that assessment.

OVERVIEW OF CS-3004

PROJECT

- An advance project
  - Provides interesting problem, realistic pressures, unclear/changing
- 3 member teams
- Emphasis on good SE practice/methodology Homework's and deliverables tied to the project

Deliverables	Deadlines
Project Proposal	(4th week)
System Requirement Specifications (SRS)	(6th week)
Progress Report	(9th week)
System Design Specifications (SDS)	(11th week)
Final Report/User Manual	(13th week)
Presentations and Demo	(15th -16th week)

OVERVIEW OF CS-3004

COURSE MATERIAL

- You will have Presentations of each topic and ***reference books*** in PDF format will be available on GCR.

Text Books

1. Unified Modeling Language User Guide, The (Addison-Wesley Object Technology Series) by Grady Booch, 2nd Edition, 2017.
2. Software Modeling and Design: UML, Use Cases, Patterns, and SoftwareArchitectures, Hassan Gomaa, Cambridge University Press, 1st Edition, 2011.
3. Applying UML and Patterns 3rd Edition by Craig Larman, 2008.

Reference Books

1. Software Engineering by Ian Sommerville. Addison-Wesley Longman Publishing, 8th Edition, 2006

OVERVIEW OF SE-3002

COURSE GOALS/OBJECTIVES

- By the end of this course, you will
  - By the end of this course, you will be able to Perform analysis on a given domain and come up with a complete system and Design.
  - Practice various techniques which are commonly used in analysis and design phases in the software industry.
  - Use Unified Modeling Language (UML) as a tool to demonstrate the analysis and design ideas
  - Analyze, design and implement practical systems of up to average complexity within a team and develop a software engineering mindset

INTRODUCTION TO SOFTWARE DESIGN & ANALYSIS

WHY STUDY SDA?

WHAT IS A SYSTEM?

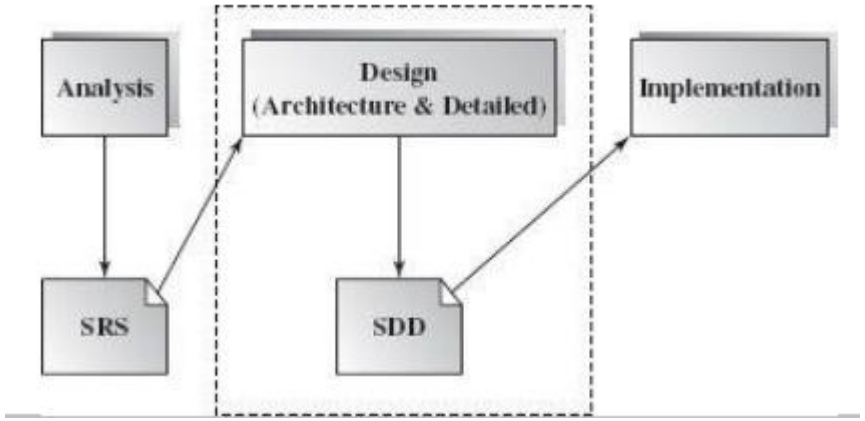
- Systems are created to solve problems
- It is an organized way of dealing with a problem
- Basically there are three major components in every system, namely input, processing and output.

SYSTEM TYPES

- A System can be a Application program or it can be an Information System.
- Computer App: is an application program (app for short) is a computer program designed to perform a group of coordinated functions, tasks, or activities for the benefit of the user.
- Information System: is software that helps you organize and analyze data. This makes it possible to answer questions and solve problems relevant to the mission of an organization.

WHAT IS SOFTWARE DESIGN?

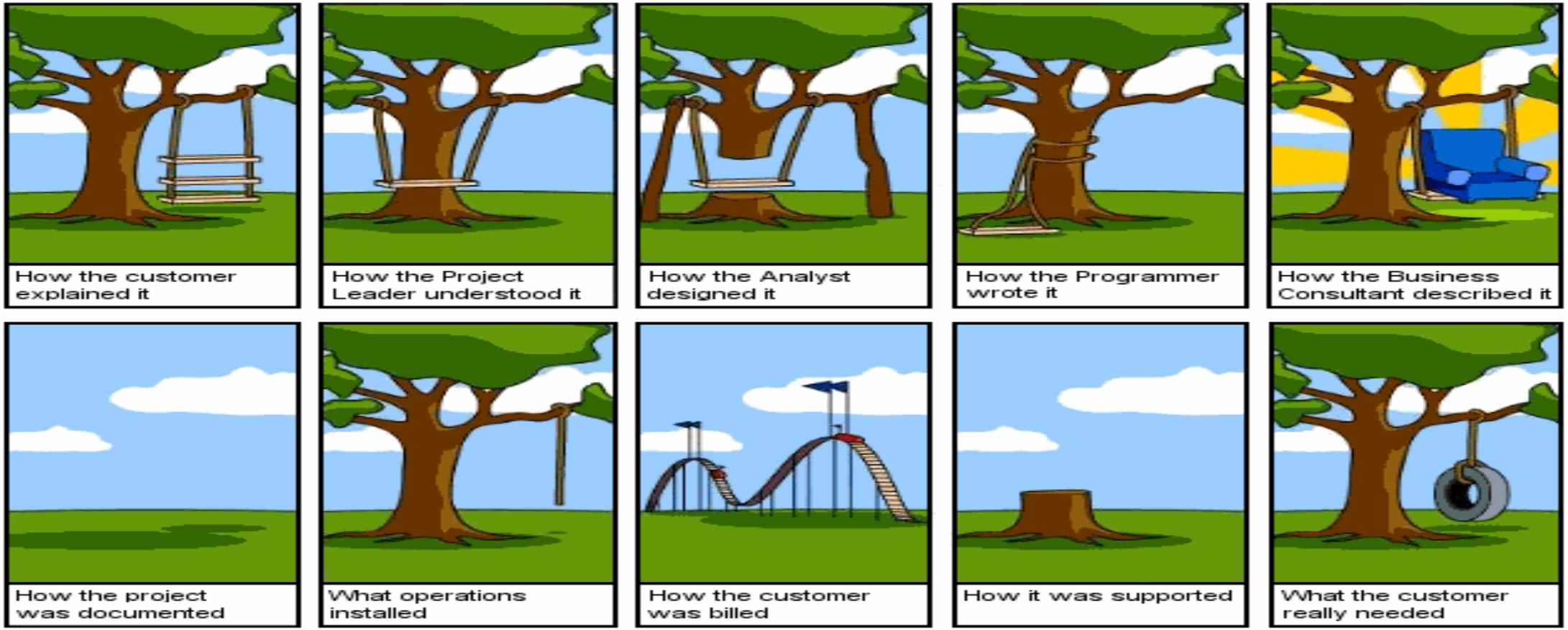
- What is software design and its goal?
  - Identifying the objects of a system.
  - Identifying their relationships.
- Making a design, which can be converted to executables using OO languages.
- The goal of software design is to build a model that meets all customer requirements and leads to successful implementation.



WHY SOFTWARE DESIGN IS IMPORTANT?

- As software systems continue to grow in scale, complexity, and distribution, their proper design becomes extremely important in software production. Any software, regardless of its application domain, should have an overall architecture design that guides its construction and development.

DEFINING THE PROBLEM IS THE PROBLEM



SOFTWARE CRISIS

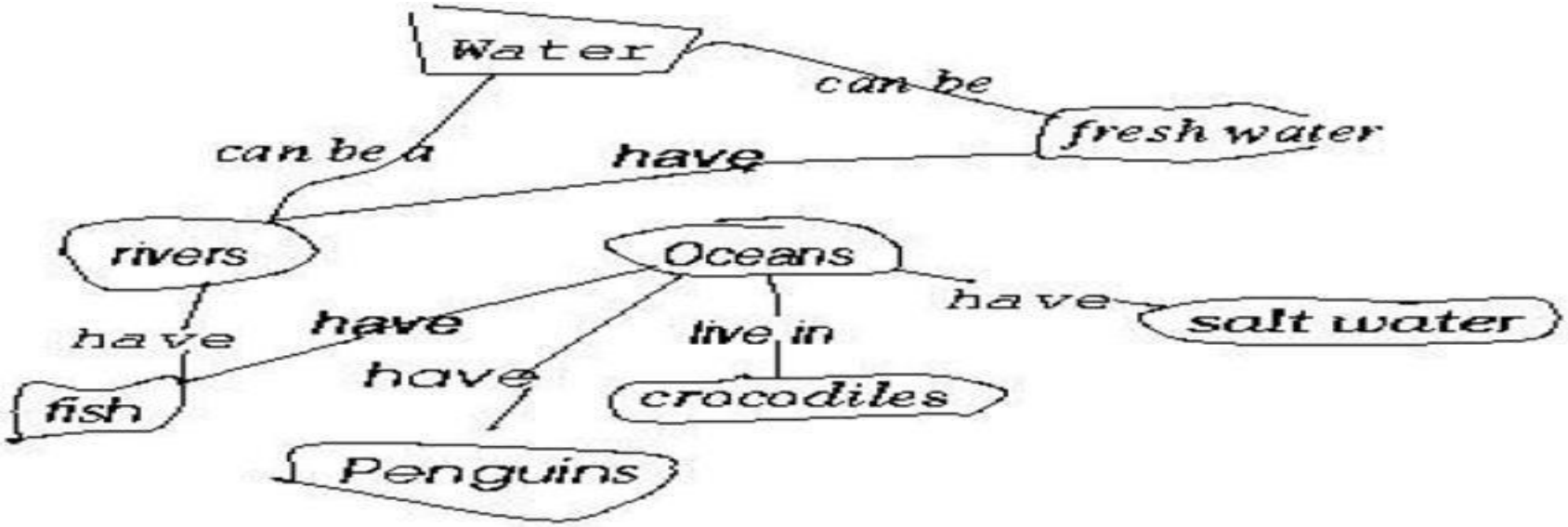
*"The "software crises" came about when people realized the major problems in software development were … caused by **communication** difficulties and the management of **complexity**" [Budd]*

What kind of language can alleviate difficulties with communication & complexity?

WHY OBJECT ORIENTED DESIGN?

Study of a first grade class [Martin & Odell] [Novak, 1984, Cambridge University Press]

When given a list of concepts (water, salt water, Oceans, Penguins,...),
Harry constructed a concept diagram through which he understands his world and communicates meaning



WHAT IS A MODEL AND WHY?

- A model is a simplification of reality.
- E.g., a miniature bridge for a real bridge to be built
- Well...sort of….but not quite
- A mental model is our simplification of our perception of reality
- A model is an abstraction of something for the purpose of understanding, be it the problem or a solution.

  - To understand why a software system is needed, what it should do, and how should do it.
  - To communicate our understanding of why, what and how.
  - To detect commonalities and differences in your perception, my perception, perception and her perception of reality.
  - To detect misunderstandings and miscommunications.

Basic OOP Concepts and Terms

WHAT IS OOP?

- OOP stands for **Object-Oriented Programming**.
- Procedural programming is about writing procedures or methods that perform operations on the data, while object-oriented programming is about creating objects that contain both data and methods.
- OOP is a paradigm that transforms code into organized, reusable, and efficient modules.

WHY DOES OOP MATTER?

- OOP isn't just another coding approach; it's a game-changer. Here's why it matters:
- **Modularity**: OOP divides complex systems into manageable chunks (objects), making development more structured and teamwork-friendly.
- **Reusability**: Objects can be reused across projects, saving time and effort, like having a toolbox that works for various DIY projects.
- **Flexibility**: OOP accommodates changes and updates smoothly. Alter an object's behavior without affecting the entire codebase.
- **Readability**: Code written in OOP is like a well-organized novel. It's easier to understand, maintain, and debug.

BASIC OO CONCEPTS

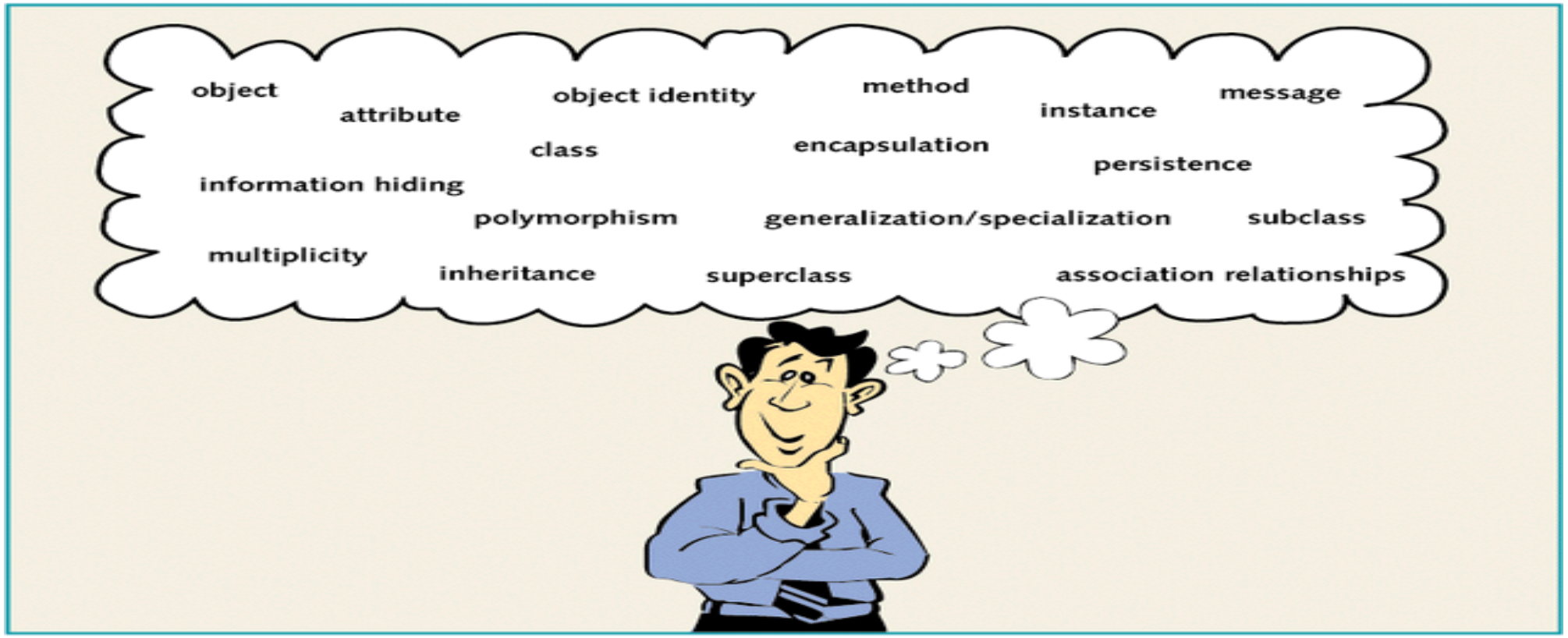


Figure 1-5 Key OO concepts

OBJECTS

- Most basic component of OO design. Objects are designed to do a small, specific piece of work.
- Objects represent the various components of a business system

EXAMPLES OF DIFFERENT OBJECT TYPES

- GUI objects
  - objects that make up the user interface
    - e.g. buttons, labels, windows, etc.
- Problem Domain objects
  - Objects that represent a business application
  - A problem domain is the scope of what the system to be built will solve. It is the business application. e.g.
    - An order-entry system
    - A payroll system
    - A student system

SAMPLE PROBLEM DOMAIN

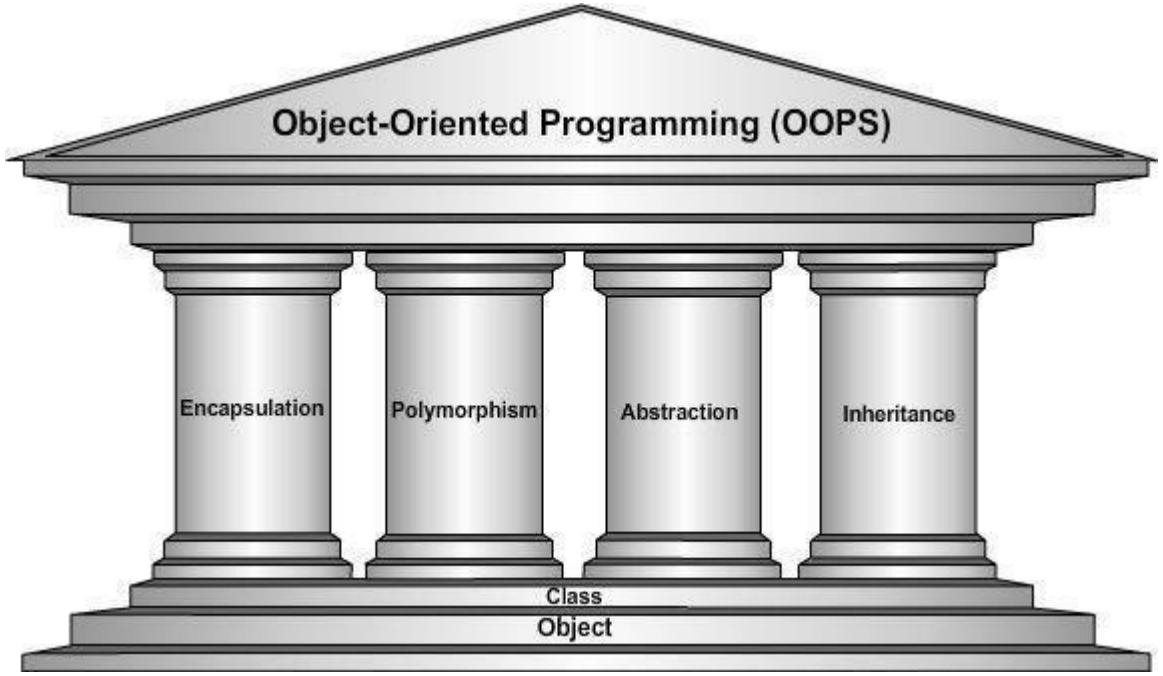
- Company ABC needs to implement an order-entry system that takes orders from customers for a variety of products.
- What are the objects in this problem domain?

CLASSES AND OBJECTS

- Classes
  - Define what all objects of the class represent
  - It is like a blueprint. It describes what the objects look like
  - They are a way for programs to model the real world
- Objects
  - Are the instances of the class
- So, a class is a template for objects, and an object is an instance of a class.
- When the individual objects are created, they inherit all the variables and methods from the class.

OOP PRINCIPLES

1) Abstraction
2) Encapsulation
3) Inheritance
4) Polymorphism



WHAT IS ABSTRACTION?

- **Abstraction** is a process of hiding implementation details and exposing only the functionality to the user. In abstraction, we deal with ideas and not events. This means the user will only know "what it does" rather than "how it does".
- Abstraction focuses on simplifying complex reality by modeling classes based on real-world objects. It hides the unnecessary details and exposes only the relevant aspects. Think of it as using a TV remote—you don't need to know the internal circuitry to change channels.
- **There are two ways to achieve abstraction in Java:**
- Abstract class
- Interface
- ***Real-Life Example:***
- ***Cars:*** *A driver will focus on the car functionality (Start/Stop -> Accelerate/ Break), he/she does not bother about how the Accelerate/ brake mechanism works internally. And this is how the abstraction works.*
- **Smartphones**: Smartphone users interact with a high-level interface, like tapping icons on the screen. The inner workings of the OS, hardware, and software are abstracted, offering a user-friendly experience.
- **Online Shopping**: When you add items to your cart and click "checkout," you interact with an abstracted system. It simplifies the shopping process without revealing the complexities of inventory management, payment processing, and shipping logistics.

ABSTRACTION

```
// Abstract class
abstract class Animal {
  // Abstract method (does not have a body)
  public abstract void animalSound();
  // Regular method
  public void sleep() {
    System.out.println("Zzz");
  }
}
```

```
// Subclass (inherit from Animal)
class Pig extends Animal {
  public void animalSound() {
    // The body of animalSound() is provided here
    System.out.println("The pig says: wee wee");
  }
}
```

```
class Main {
  public static void main(String[] args) {
    Pig myPig = new Pig(); // Create a Pig object
    myPig.animalSound();
    myPig.sleep();
  }
}
```

HOW ABSTRACTION ENHANCES CODE READABILITY?

- Abstraction enhances code readability by offering a clear and simplified view of complex systems.
- **Focus on Essentials**: Abstraction lets you concentrate on the core functionality of a class without being distracted by implementation details. For instance, in a car class, you're concerned with "start" and "stop" methods, not the intricacies of the engine.
- **User-Friendly**: It makes code more user-friendly. Other developers can understand and use your classes without comprehending every intricate detail. It's like driving a car without needing to be an automotive engineer.

WHAT IS ENCAPSULATION?

- **Encapsulation** is the process of wrapping code and data together into a single unit.
- Encapsulation is the foundation of data protection and code organization. It ensures your code is like a vault, keeping its secrets hidden while offering controlled access to its treasures.
- **Real-Life Examples of Encapsulation**
- **Banking System**: In a banking system, customer account details like balance and account number are encapsulated within a class. External access to these attributes is limited to specific methods, ensuring data integrity.
- **Smartphone**: Smartphone manufacturers encapsulate the hardware components (e.g., camera, screen) within classes. The user interacts with these components through well-defined methods (e.g., taking a photo).

ENCAPSULATION

```
public class Person {

    private String name;

    // Getter

    public String getName()
{

      return name;

    }

    // Setter

    public void
setName(String newName) {

      this.name = newName;

    }

}
```

```
public class Main {
  public static void main(String[]
args) {
     Person myObj = new Person();
     myObj.setName("John");

System.out.println(myObj.getName());
  }
}
```

WHY IS ENCAPSULATION IMPORTANT?

- **Data Security**: Encapsulation keeps sensitive data safe from unauthorized access. For instance, the password attribute is encapsulated in a user class, preventing direct access.
- **Modularity**: It promotes modular code, making it easy to maintain and understand. Changes to the internal structure don't ripple across the entire codebase.

WHAT IS INHERITANCE?

- **Inheritance** is the process of one class inheriting properties and methods from another class in Java. Inheritance is used when we have **is-a** relationship between objects. Inheritance in Java is implemented using **extends** keyword.
- **Real-life examples of Inheritance**
- **Animal Kingdom**: In a virtual zoo application, you might have a base class "Animal." From there, you can create subclasses like "Mammal," "Bird," and "Reptile," each inheriting common traits like "eat" and "sleep."
- **Software Interfaces**: In Java, interfaces enable multiple classes to inherit common methods, ensuring a consistent interaction with different objects. For example, the "Serializable" interface allows various classes to be serialized for storage or transmission.

INHERITANCE

```
class Vehicle {
  protected String brand = "Ford";
  public void honk() {
    System.out.println("Tuut, tuut!");
  }
}
class Car extends Vehicle {
  private String modelName = "Mustang";
  public static void main(String[] args) {
    Car myFastCar = new Car();
    myFastCar.honk();
    System.out.println(myFastCar.brand + " " + myFastCar.modelName);
  }
}
```

HOW INHERITANCE SIMPLIFIES CODE

- **Code Reusability**: You don't have to reinvent the wheel. If you have a class for "Vehicle," you can create subclasses like "Car" and "Bicycle" that inherit the standard features like "move" and "stop." There is no need to rewrite these functions for each type of vehicle.
- **Modularity**: Inheritance promotes a modular approach. Changes or updates to the base class automatically apply to all its subclasses. For example, enhancing the "Vehicle" class with a "fuelEfficiency" attribute will benefit all its subclasses.

WHAT IS POLYMORPHISM?

- Polymorphism is the ability to perform many things in many ways. The word Polymorphism is from two different Greek words- poly and morphs. "Poly" means many, and "Morphs" means forms. So polymorphism means many forms. The polymorphism can be present in the case of inheritance also. The functions behave differently based on the actual implementation.
- **Real-World Applications of Polymorphism**
- *A delivery person delivers items to the user. If it's a postman he will deliver the letters. If it's a food delivery boy he will deliver the foods to the user. Like this polymorphism implemented different ways for the delivery function.*
- **Shapes in Graphics**: Consider graphics software. You have shapes like circles, squares, and triangles, each with a "draw" method. Polymorphism allows you to treat all these shapes uniformly, simplifying code. When you call "draw" in any form, it behaves according to its specific class.
- **Payment Methods**: In an e-commerce system, you may have multiple payment methods like credit cards, PayPal, and bank transfers. Each method has a "processPayment" function. Polymorphism enables you to use a generic "processPayment" function that adapts to the chosen payment method at runtime.
- **Animal Sounds**: In a virtual pet game, you can have a variety of animals—dogs, cats, and birds. Each animal makes a sound when you interact with it. Polymorphism lets you call a "makeSound" function on any animal object, producing the appropriate sound based on the animal's class.

POLYMORPHISM

```
class Animal {
  public void animalSound() {
    System.out.println("The animal makes a sound");
  }
}

class Pig extends Animal {
  public void animalSound() {
    System.out.println("The pig says: wee wee");
  }
}

class Dog extends Animal {
  public void animalSound() {
    System.out.println("The dog says: bow wow");
  }
}
```

```
class Main {
  public static void main(String[] args) {
    Animal myAnimal = new Animal();
    Animal myPig = new Pig();
    Animal myDog = new Dog();

    myAnimal.animalSound();
    myPig.animalSound();
    myDog.animalSound();
  }
}
```

TYPES OF POLYMORPHISM

- There are two types of polymorphism as listed below:
  - Static or Compile-time Polymorphism
  - Dynamic or Run-time Polymorphism

COMPILE-TIME POLYMORPHISM

- Static or Compile-time Polymorphism when the compiler is able to determine the actual function, it's called **compile-time** polymorphism. Compile-time polymorphism can be achieved by **method overloading** in java. When different functions in a class have the same name but different signatures, it's called method overloading. A method signature contains the name and method arguments. So, overloaded methods have different arguments. The arguments might differ in the numbers or the type of arguments.

```
public class GFG {

    // First addition function
    public static int add(int a, int b)
    {
        return a + b;
    }

    // Second addition function
    public static double add(
        double a, double b)
    {
        return a + b;
    }

    // Driver code
    public static void main(String args[])
    {
        // Here, the first addition
        // function is called
        System.out.println(add(2, 3));

        // Here, the second addition
        // function is called
        System.out.println(add(2.0, 3.0));
    }
}
```

DYNAMIC OR RUN-TIME POLYMORPHISM

- Dynamic (or run-time) polymorphism occurs when the compiler is not able to determine at compile-time which method (superclass or subclass) will be called. This decision is made at run-time.
- Run-time polymorphism is achieved through method overriding, which happens when a method in a subclass has the same name, return type, and parameters as a method in its superclass. When the superclass method is overridden in the subclass, it is called method overriding.

```
class Test {

    // Implementing a method
    public void method()
    {
        System.out.println("Method 1");
    }
}

public class GFG extends Test {

    // Overriding the parent method
    public void method()
    {
        System.out.println("Method 2");
    }

    // Driver code
    public static void main(String args[])
    {
        Test test = new GFG();

        test.method();
    }
}
```

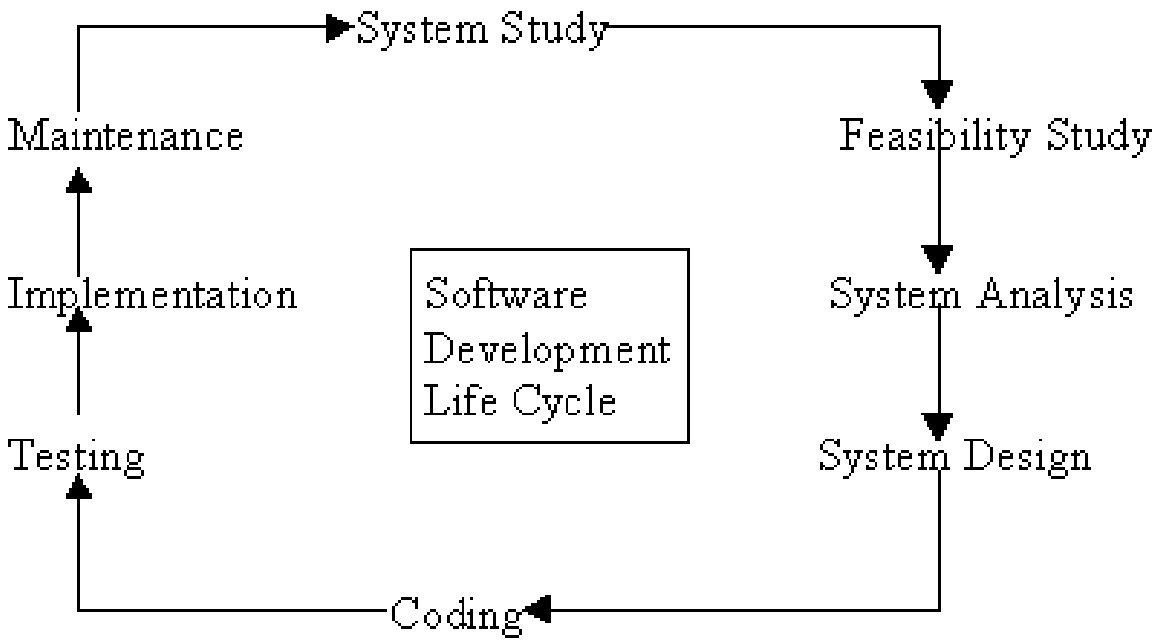
WHY IS POLYMORPHISM IMPORTANT?

- Polymorphism simplifies code by allowing you to work with objects at a higher level of abstraction, regardless of their specific type. It's like speaking a universal language that all things understand, making your code more flexible and adaptable.

SOFTWARE DEVELOPMENT LIFE CYCLE

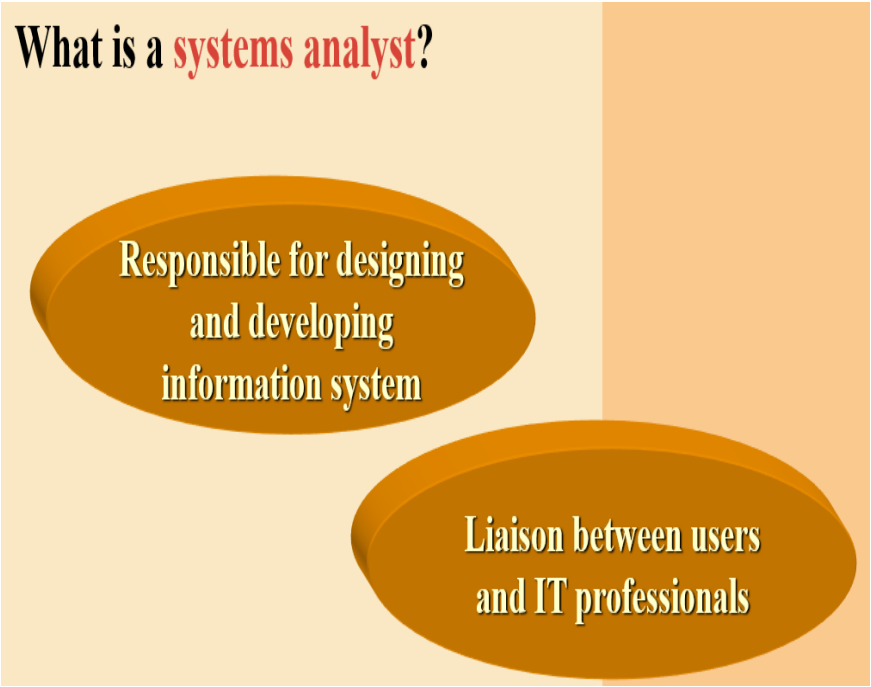
- Software Development Life Cycle is an organizational process of developing and maintaining systems.
- It is a mixture of various activities.
- Following are the phases for SDLC:
  - System study
  - Feasibility study
  - System analysis
  - System design
  - Coding
  - Testing
  - Implementation
  - Maintenance

SOFTWARE DEVELOPMENT LIFE CYCLE



SYSTEM STUDY

- System study - 1st stage of system development life cycle.
- Gives clear picture of what actually the physical system is.
- **System study phases (I & II):**
  - I: initial survey of the system - helps in identifying the scope.
  - II: in-depth study - requirement identification / limitations & issues of current system.
- **Proposal:**
  - Prepared after completing the system study,
  - prepared by the System Analyst.
  - Contains the findings of the current system
  - Recommendations to overcome the limitations / issues of the current system.
- **Steps of System Study phase:**
  - problem identification and project initiation.
  - background analysis.
  - inference or findings.



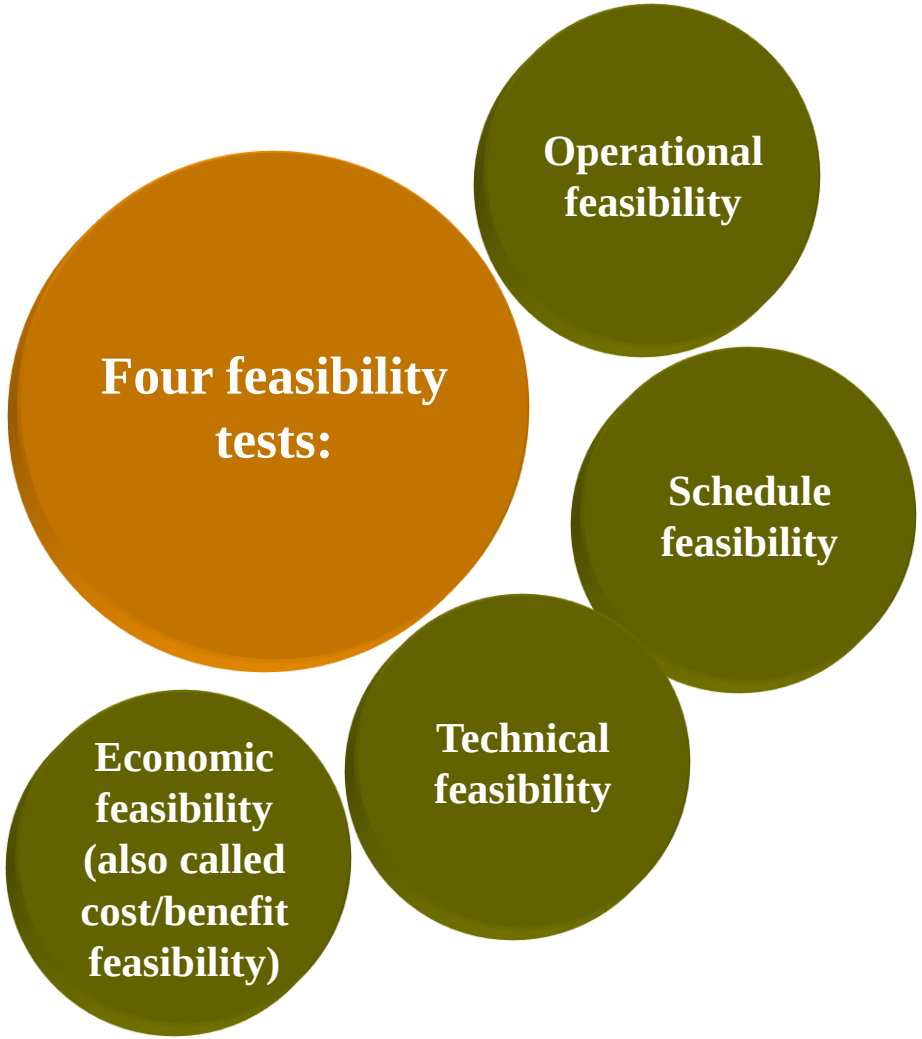
FEASIBILITY STUDY

- Done on the basis of initial study.
- It is the test of the proposed system in the light of its workability, user's requirements, effective use of resources and the cost effectiveness.
- Goal : to achieve the scope.(not to solve issues)
- Advantage: Cost and benefits are estimated with greater accuracy.

THE SYSTEM DEVELOPMENT LIFE CYCLE

What is feasibility?

Measure of how suitable system development will be to the company



SYSTEM ANALYSIS

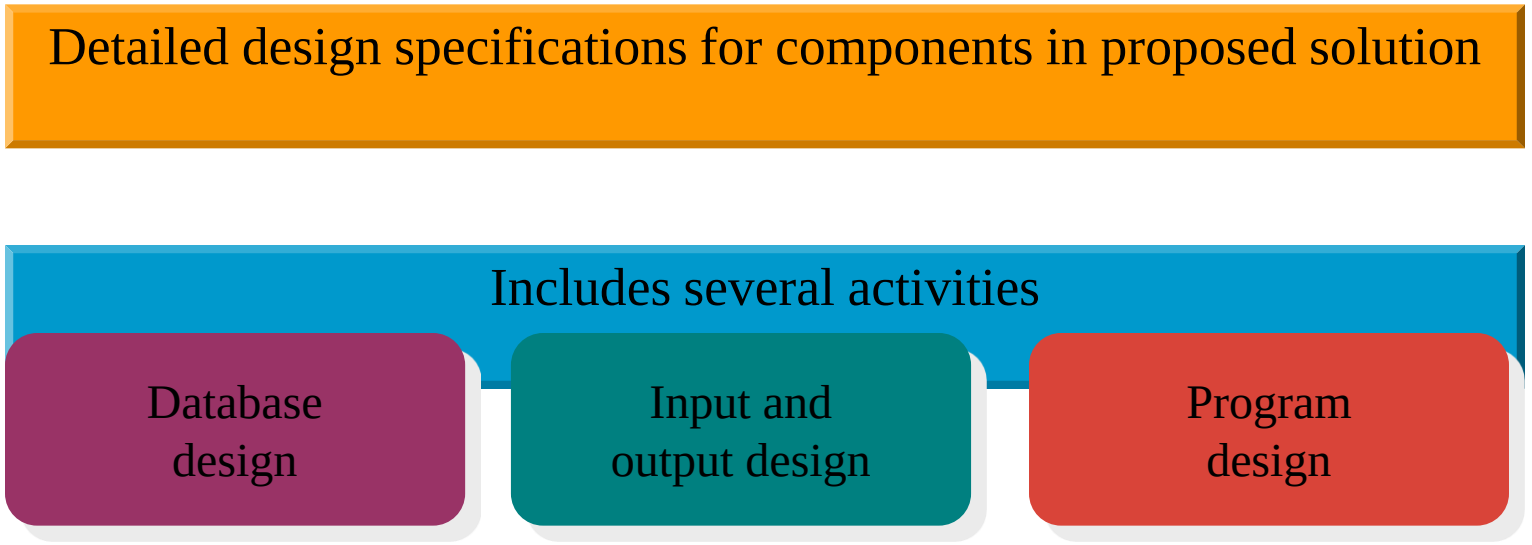
- Analysis is detailed study of :
  - Current system, leading to specifications of a new system.
  - System operations & its relationships within & outside system.
- Data collection for: files, decision points, & transactions of present system.
- Tools of system analysis: Interviews, on-site observation & questionnaire.
- Steps to define boundary of the new system:
  - Keeping in view the problems and new requirements
  - Work out pros & cons including new system
- Analysis is documented in: detailed DFDs, data dictionary, logical data structures & miniature specifications.
- Includes sub-dividing of complex process, data store identification & manual processes.

SYSTEM DESIGN

- The new system must be designed based on the user requirements and the detailed analysis of a new system.

- This phase has two stages:
  - Preliminary or General Design
  - Structure or Detailed Design

SYSTEM DESIGN



What is a detailed design?

CODING

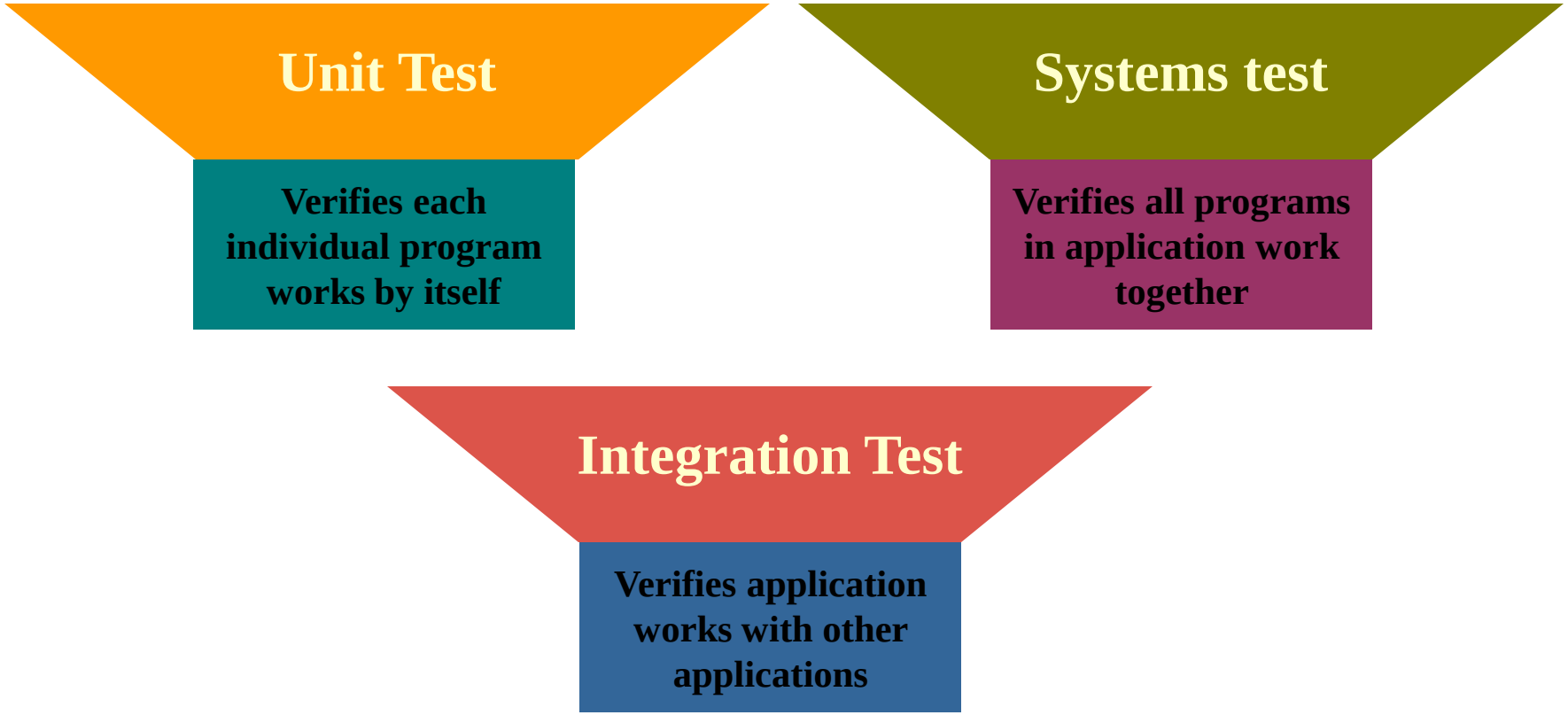
- Coding the new system into computer programming language converts human instructions into a format that computer understands.
- Coding is stage where defined procedure are transformed into control specifications by the help of a computer language.
- This is also called the programming phase in which the programmer converts the program specifications into computer instructions, which we refer as programs.
- The programs coordinate the data movements and control the entire process in a system.
- Generally, programs must be modular in nature. This helps in fast development, maintenance and future change, if required.

TESTING

- Removing all the bugs, if any - Before implementing
- Test plan is developed and run on given set of test data.
- Output of test run should match expected results.
- Using test data following test runs are carried out:

TESTING

What are the three types of tests performed by system developers?



IMPLEMENTATION

- After UAT, the implementation phase begins.
- It is the stage during which theory is turned into practice.
- All programs of the system are loaded onto the user's computer.
- After loading the system, training of the users starts.
- Main topics of such type of training are:
  - How to execute the package
  - How to enter the data
  - How to process the data (processing details)
  - How to take out the reports
- After the users are trained about the computerized system, manual working has to shift from manual to computerized working.

IMPLEMENTATION

What is training?

- **Showing users exactly how they will use new hardware and software in system**

SYSTEM RUN STRATEGIES

- **Parallel run:** Computerized & manual systems are executed in parallel. Advantages of Parallel run:
  - Manual results comparison with the computerized one.
  - Failure of the computerized system at the early stage, does not affect the working of the organization.

SYSTEM RUN STRATEGIES

- **Pilot run:** New system is installed in parts. Some part of the new system is installed first and executed successfully for considerable time period. Some advantages are:
  - When results are found satisfactory then only other parts are implemented.
  - This strategy builds the confidence and the errors are traced easily.

MAINTENANCE

- **System Review:** is necessary from time to time for:
- Knowing the full capabilities of the system
- Knowing the required changes or the additional requirements
- Studying the performance
- **Major change during the review:**
- If a major change to a system is needed, a new project may have to be set up to carry out the change.
- New project will then proceed through all above life cycle phases.

SYSTEM ENVIRONMENTS

- Development
- Test
- Staging
- Pre-Production
- Production
- Mirror

ROLES INVOLVED

- Developer
- Development Manager
- Project Manager
- Test Manager
- Configuartion Manager
- Deployment/Implementation Team

That is all