OOSE Unit -1

Evolution of Object Orientation

- The idea of object-oriented programming gained momentum in the 1970s and in the early 1980s.
- • Bjorn Stroustrup integrated object-oriented programming into the C language. The resulting language was called C++ and it became the first object-oriented language to be widely used commercially.
- • In the early 1990s a group at Sun led by James Gosling developed a simpler version of C++ called Java that was meant to be a programming language for video-on-demand applications.
- • This project was going nowhere until the group re-oriented its focus and marketed Java as a language for programming Internet applications.
- • The language has gained widespread popularity as the Internet has boomed, although its market penetration has been limited by its inefficiency

Object Oriented Programming Approach

- The basic principal of the OOP approach is to combine both data and functions so that both can operate into a single unit. Such a unit is called an Object. • This approach secures data also. Now a days this approach is used mostly in applications. The programming languages: C++ and JAVA follow this approach. Using this approach we can write any lengthy code.

Object Orientation Paradigm

- An approach to the solution of problems in which all computations are performed in context of objects.
- • The objects are instances of programming constructs, normally called as classes which are data abstractions with procedural abstractions that operate on objects.
- • A software system is a set of mechanism for performing certain action on certain data Algorithm + Data structure = Program
- • Data Abstraction + Procedural Abstraction

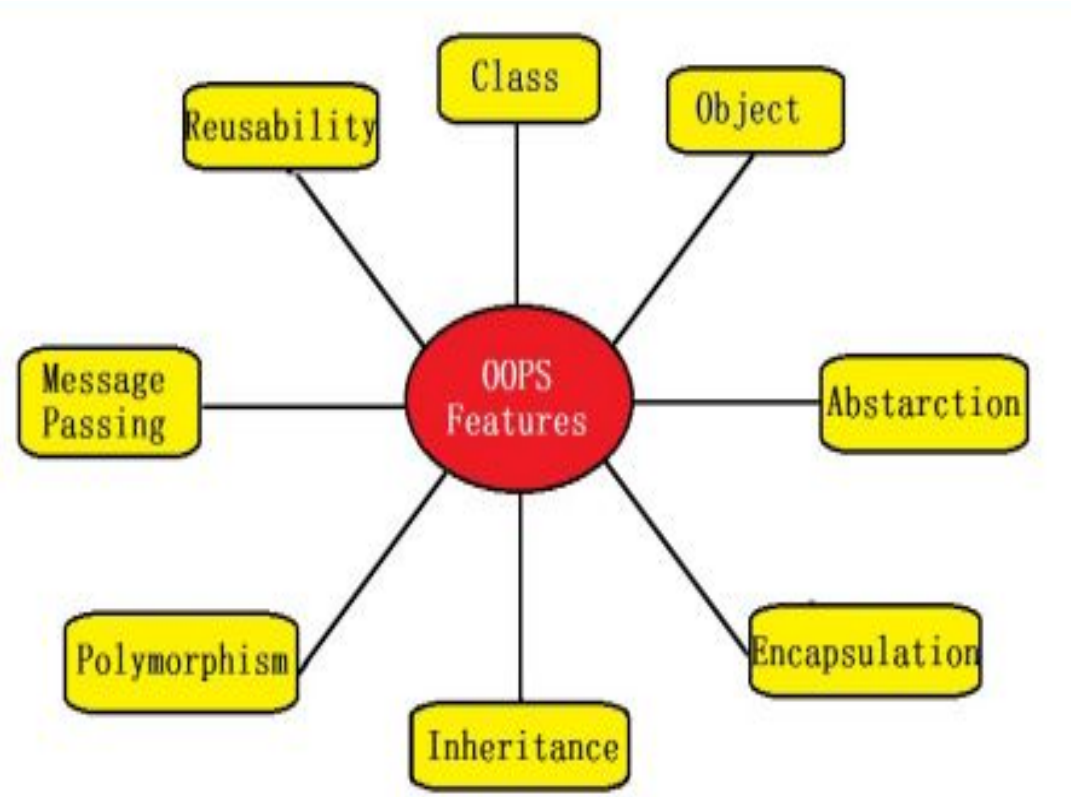
Object Orientation

- Object orientation refers to a special type of programming paradigm that combines data structures with functions to create re-usable objects.
- • The object-oriented (OO) paradigm is a development strategy based on the concept that systems should be built from a collection of reusable components called objects.
- • Instead of separating data and functionality as is done in the structured paradigm, objects encompass both.
- • Why object orientation? To create sets of objects that work together concurrently to produce s/w that better, model their problem domain that similarly system produced by traditional techniques.

Object Orientation Adaptation

- Object orientation adapts to the following criteria’s
- 1. Changing requirements
- 2. Easier to maintain
- 3. More robust
- 4. Promote greater design
- 5. Code reuse
- 6. Higher level of abstraction
- 7. Encouragement of good programming techniques
- 8. Promotion of reusability

Object Orientated Features



Object Orientated Features

- . OBJECT - Object is a collection of number of entities. Objects take up space in the memory. Objects are instances of classes. When a program is executed , the objects interact by sending messages to one another. Each object contain data and code to manipulate the data. Objects can interact without having know details of each others data or code. Each instance of an object can hold its own relevant data.
- 2. CLASS - Class is a collection of objects of similar type. Objects are variables of the type class. Once a class has been defined, we can create any number of objects belonging to that class. Classes are user define data types. A class is a blueprint for any functional entity which defines its properties and its functions.

Object Orientated Features

- 3. DATA ENCAPSULATION – Combining data and functions into a single unit called class and the process is known as Encapsulation. Class variables are used for storing data and functions to specify various operations that can be performed on data. This process of wrapping up of data and functions that operate on data as a single unit is called as data encapsulation. Data is not accessible from the outside world and only those function which are present in the class can access the data.
- 4. DATA ABSTRACTION- Abstraction (from the Latinn abs means away from and trahere means to draw) is the process of taking away or removing characteristics from something in order to reduce it to a set of essential characteristics. Advantage of data abstraction is security.

Object Orientated Features

- 5. INHERITANCE- It is the process by which object of one class acquire the properties or features of objects of another class. The concept of inheritance provide the idea of reusability means we can add additional features to an existing class without modifying it. This is possible by driving a new class from the existing one. Advantage of inheritance is reusability of the code.
- 6. MESSAGE PASSING - The process by which one object can interact with other object is called message passing.
- 7. POLYMORPHISM - A greek term means ability to take more than one form. An operation may exhibit different behaviours in different instances. The behaviour depends upon the types of data used in the operation.

Object Orientated Features

- 8. PERSISTENCE - The process that allows the state of an object to be saved to non-volatile storage such as a file or a database and later restored even though the original creator of the object no longer exists

Benefits of OOPs

- Code Reuse and Recycling: Objects created for Object Oriented Programs can easily be reused in other programs.The code and designs in object-oriented software development are reusable because they are modeled directly out of the real-world problem-domain.
- • Design Benefits: Large programs are very difficult to write. Object Oriented Programs force designers to go through an extensive planning phase, which makes for better designs with less flaws. • Ease out development: In addition, once a program reaches a certain size, Object Oriented Programs are actually easier to program than non-Object Oriented ones.

Benefits of OOPs

- • Object orientation works at a higher level of abstraction One of our most powerful techniques is the form of selective amnesia called 'Abstraction'. Abstraction allows us to ignore the details of a problem and concentrate on the whole picture.
- • Software life cycle requires no vaulting The object-oriented approach uses essentially the same language to talk about analysis, design, programming and (if using an Object-oriented DBMS) database design. This streamlines the entire software development process, reduces the level of complexity and redundancy, and makes for a cleaner system architecture and design.

Benefits of OOPs

- • Data is more stable than functions Functions are not the most stable part of a system, the data is. Over a period of time, the requirements of a system undergo radical change. New uses and needs for the software are discovered; new features are added and old features are removed. During the course of all this change, the underlying heart- data of the system remains comparatively constant.
- • Software Maintenance: Legacy code must be dealt with on a daily basis, either to be improved upon or made to work with newer computers and software. An Object Oriented Program is much easier to modify and maintain. So although a lot of work is spent before the program is written, less work is needed to maintain it over time.

Object- The CRUX of the matter!!

- “An object is an entity which has a state and a defined set of operations which operate on that state.”
- o The state is represented as a set of object attributes. The operations associated with the object provide services to other objects (clients) which request these services when some computation is required
- o Objects are created according to some object class definition. An object class definition serves as a template for objects. It includes declarations of all the attributes and services which should be associated with an object of that class.
- o An Object is anything, real or abstract, about which we store data and those methods that manipulate the data.
- o An object is a component of a program that knows how to perform certain actions and how to interact with other elements of the program.

Object- The CRUX of the matter!!

- Each object is an instance of a particular class or subclass with the class's own methods or procedures and data variables. An object is what actually runs in the computer.
- • Objects are the basic run time entities in an object oriented system.
- • They match closely with real time objects.
- • Objects take up space in memory and have an associated address like a Record in Pascal and a Structure in C.
- • Objects interact by sending Message to one other. E.g. If “Customer” and “Account” are two objects in a program then the customer object may send a message to the account object requesting for bank balance without divulging the details of each other’s data or code.
- • Code in object-oriented programming is organized around objects.

What is software engineering

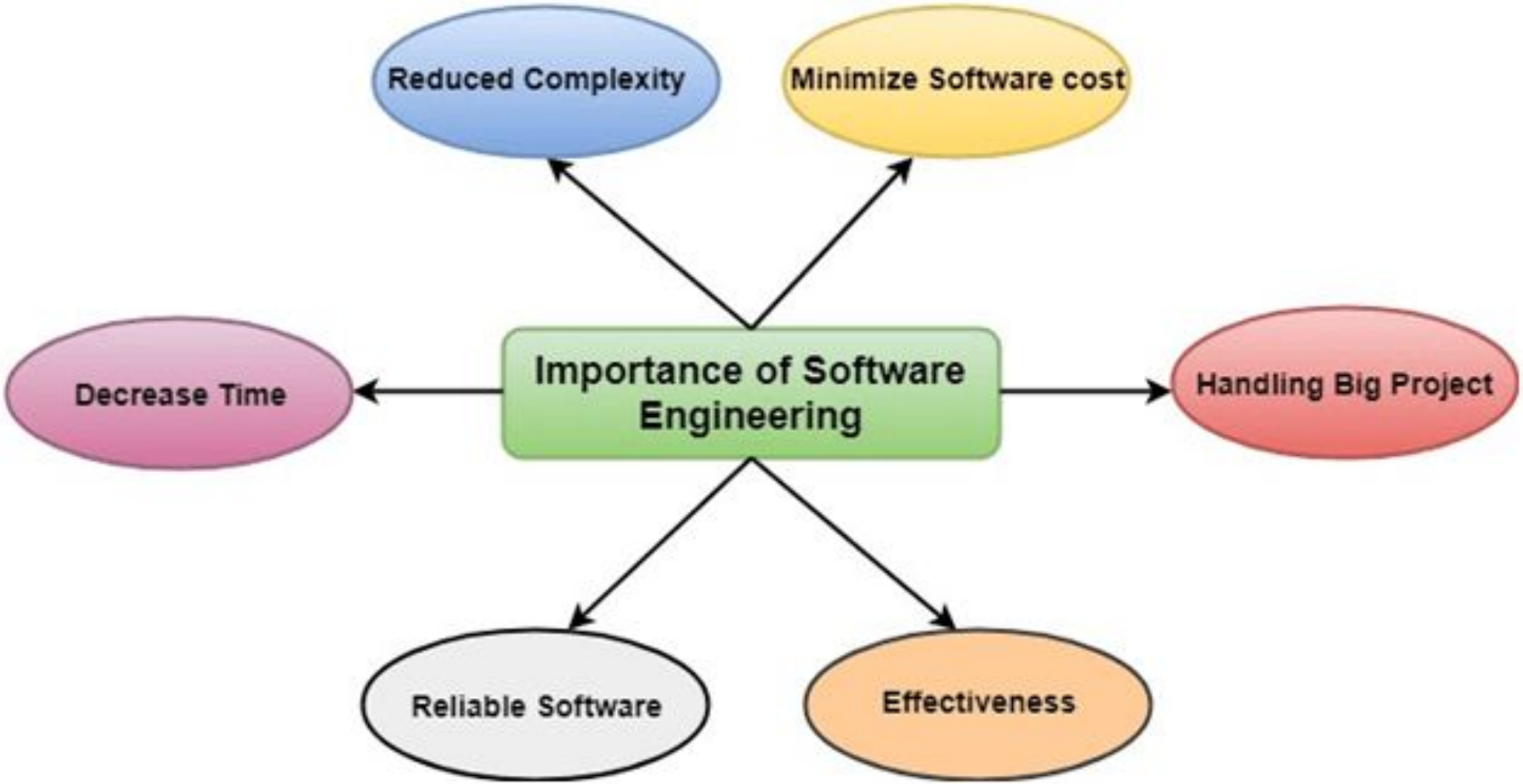
- The term **software engineering** is the product of two words, **software**, and **engineering**.
- The **software** is a collection of integrated programs.
- Software subsists of carefully-organized instructions and code written by developers on any of various particular computer languages.
- Computer programs and related documentation such as requirements, design models and user manuals.
- **Engineering** is the application of **scientific** and **practical** knowledge to **invent, design, build, maintain**, and **improve frameworks, processes, etc**.
- **Software Engineering** is an engineering branch related to the evolution of software product using well-defined scientific principles, techniques, and procedures. The result of software engineering is an effective and reliable software product.


Deployment
Updates
Testing
Maintenance
Software Product
Code Design
System Design
System Analysis
Requirements

Need of Software Engineering

- The necessity of software engineering appears because of a higher rate of progress in user requirements and the environment on which the program is working.

- **Huge Programming:** It is simpler to manufacture a wall than to a house or building, similarly, as the measure of programming become extensive engineering has to step to give it a scientific process.
- **Adaptability:** If the software procedure were not based on scientific and engineering ideas, it would be simpler to re-create new software than to scale an existing one.
- **Cost:** As the hardware industry has demonstrated its skills and huge manufacturing has let down the cost of computer and electronic hardware. But the cost of programming remains high if the proper process is not adapted.
- **Dynamic Nature:** The continually growing and adapting nature of programming hugely depends upon the environment in which the client works. If the quality of the software is continually changing, new upgrades need to be done in the existing one.
- **Quality Management:** Better procedure of software development provides a better and quality software product.


Reduced Complexity
Minimize Software cost
Decrease Time
Importance of Software Engineering
Handling Big Project
Reliable Software
Effectiveness

Importance of Software engineering

1. **Reduces complexity:** Big software is always complicated and challenging to progress. Software engineering has a great solution to reduce the complication of any project. Software engineering divides big problems into various small issues. And then start solving each small issue one by one. All these small problems are solved independently to each other.
2. **To minimize software cost:** Software needs a lot of hardwork and software engineers are highly paid experts. A lot of manpower is required to develop software with a large number of codes. But in software engineering, programmers project everything and decrease all those things that are not needed. In turn, the cost for software productions becomes less as compared to any software that does not use software engineering method.
3. **To decrease time:** Anything that is not made according to the project always wastes time. And if you are making great software, then you may need to run many codes to get the definitive running code. This is a very time-consuming procedure, and if it is not well handled, then this can take a lot of time. So if you are making your software according to the software engineering method, then it will decrease a lot of time.

Importance of Software engineering

- **Handling big projects:** Big projects are not done in a couple of days, and they need lots of patience, planning, and management. And to invest six and seven months of any company, it requires heaps of planning, direction, testing, and maintenance. No one can say that he has given four months of a company to the task, and the project is still in its first stage. Because the company has provided many resources to the plan and it should be completed. So to handle a big project without any problem, the company has to go for a software engineering method.
- **Reliable software:** Software should be secure, means if you have delivered the software, then it should work for at least its given time or subscription. And if any bugs come in the software, the company is responsible for solving all these bugs. Because in software engineering, testing and maintenance are given, so there is no worry of its reliability.
- **Effectiveness:** Effectiveness comes if anything has made according to the standards. Software standards are the big target of companies to make it more effective. So Software becomes more effective in the act with the help of software engineering.

Principles of Software Engineering

1. **Keep It Simple, Stupid**

- The principle of simplicity states that codes should be as simple as possible without a complicated structure, otherwise debugging and maintenance might be more difficult. Moreover, another programmer will have a harder time understanding the code logic, which will entail more time and effort. Do you want to add that complexity? When coding your next big project, make sure you write simple and easily comprehensible code.

- It’s best if your methods are small, not exceeding 40-50 lines.
- All the crucial/critical methods should have a commented doc for better understanding for other dev.
- Methods should only address one problem at a time.
- Your project has a lot of conditions, right? Break up the codes into smaller blocks as you go.
- If possible, use simple constructions that solve the problem without a lot of branching, deep nesting, or complex class structures.

Principles of Software Engineering

2. DRY (Don’t Repeat Yourself)

- In a nutshell, the principle of DRY states that we shouldn’t repeat the same thing too many times in too many places. In software systems, it aims to reduce repetitive code and effort. Developers unknowingly re-write code repeatedly. When writing your code, don’t copy-paste the same code repeatedly. If you don’t, then you’ll need to keep them in sync; any change in code at one place will need to be done at other places. It will take extra time, effort, and attention (which isn’t always easy).

- Make sure not only that your code is error-free, but that it is free of duplicate lines.
- If a piece of code appears more than twice in the codebase, it should be moved to a separate function.
- You should create a separate method even if you find that it is repeated a second time.
- As a bonus, automate any manual processes you can in order to keep the code lean.

Principles of Software Engineering

- **S – SRP (Single Responsibility Principle):** According to the Single Responsibility Principle, a class, function, module, or service must have only one reason to change, i.e., it must have only one responsibility. Yet why is this so important?

- When you write classes or functions that are dedicated to a single functionality, it becomes easier to understand, maintain, and modify your code.
- If you want to modify the functionality of the system, you would know the exact DRY location where you have to modify the code.
- It makes the code more organized and readable. It also makes reusing the code easier.

Principles of Software Engineering

- **O – OCP (Open Closed Principle):** In software development, we work in phases. As a team, we implement a bunch of functionalities, test them, and then deliver them to the users. We then move on to implementing the next set of functionalities. When it comes to developing new functionality, the last thing we want to do is to change the existing functionality, which has been tested and is working. Therefore, we try to add new functionality on top of existing ones.

- This idea is facilitated by the Open-Closed principle. According to it, our functions, classes, and modules should be designed in such a way that they're open for extension, but closed for modification.

- **Open for Extension:** New functionality can be added to classes and modules without breaking the existing code. Composition and inheritance can be used to accomplish this.

- **Closed for Modification:** It's ideal not to make changes that break current functionality, as doing so would require refactoring a lot of existing code and writing several tests to ensure the changes work.

Principles of Software Engineering

- **L – LSP (Liskov Substitution Principle):** According to the Liskov Substitution Principle, all child/derived classes should be replaceable for their parent/base classes without affecting or breaking the program's correctness. Thus, objects in your subclass (derived/child class) should behave similarly to objects in your superclass (parent/base class). Therefore, you should use inheritance carefully in your projects. While inheritance can be beneficial, it is advisable to use it moderately and contextually. Before you can perform inheritance, you have to consider the postconditions and preconditions of the class.

- **I – ISP (Interface Segregation Principle):** According to the Interface Segregation Principle, clients shouldn't be forced to depend or rely on methods that they don't use. How can this be achieved? Simple: make your interfaces short or small and focused. An interface with a lot of behaviors is hard to maintain and evolve. Therefore, Separate large interfaces into smaller ones, each focused on a specific set of functions, so that choose to depend or rely only on the functionalities that they require

Principles of Software Engineering

- **D – DIP (Dependency Inversion Principle):** This Principle seeks to eliminate tight coupling between software modules. According to this principle, high-level modules should not depend on lower-level modules, but rather on their abstractions. We can break it down into two parts:

  - A high-level module must be independent of a low-level module. Both should rely on abstractions
  - The abstraction should be independent of the details, while the details should be dependent upon the abstractions.

Object oriented development

- In Object-Oriented Development, we apply object orientation across every system development activity such as requirement analysis, design, programming, testing, and maintenance. For instance, an object oriented analysis (OOA) will usually have the following steps:
- Identifying the system functionality
- Identifying the involved objects
- Recognizing the object classes
- Analysing the objects to fulfil the system functionality

Object-Oriented Programming (OOP) is based on object oriented features like Encapsulation, Polymorphism, Inheritance and Abstraction. These features are usually referred to as the OOPs concepts. We will see more about analysis, design, testing and maintenance later.

Encapsulation

- **Encapsulation** is the process of wrapping up of data (as properties) and behavior (as methods) of an object into a single unit (a class). Encapsulate in plain English means *to enclose or be enclosed in or as if in a capsule*.
- Encapsulation enables **data hiding,** hiding irrelevant information from the users of a class and exposing only the relevant details required by the user. We can expose our operations to the outside world and hide the details of what is needed to perform that operation. We can thus protect the internal state of an object by hiding its attributes from the outside world, and then exposing them through setter and getter methods. Now modifications to the object internals are only controlled through these methods.
- **Example from real world:** Consider the smart phone you are using now. You are not worried about the internal operations of the smart phone. You only know and care about the operations or functions it exposes to you such as making call, sending SMS or using your apps.

Inheritance

- Inheritance describes the relationship between two classes. A class can get some of its characteristics from a parent class and then add unique features of its own. For example consider a Vehicle parent class and its child class Car. Vehicle class will have all common properties and functionalities for all vehicles in common and Car will inherit those common properties from the Vehicle class and then add those properties which are specific to a car. Here, Vehicle is known as base class, parent class, or superclass. Car is known as derived class, Child class or subclass.
- In multiple inheritance a class can inherit from more than one parent, but due to its complexity some languages like Java doesn't fully support it. In multi-level inheritance there will be many levels of inheritence like A inheriting from B and B inheriting from C and so on.

Polymormism

- Polymormism means that one operation can be used for different purposes. Poly means many and morph means form. For instance, Java supports different kinds of polymorphism like oveloading, overriding, parametric etc. In overloading, the multiple methods having same name can appear in a class, but with different signature. Overriding is defining a method in a subclass with the same name and type signature as a method in its superclass and the overriden method is called at runtime based on the object at runtime.

Abstraction

- In plain English, abstract means a concept or idea not associated with any specific instance and does not have a concrete existence. Abstraction in OOP emphasizes what is important and what is not important at a particular level of detail. Abstraction refers to the ability to make a class abstract at a level and define some of the behaviour at each level, relevant to the current perspective. Abstraction allows the programmer to focus on few concepts at a time. Java provides interfaces and abstract classes for describing abstract types.

Coupling vs Cohesion

- Coupling and cohesion are two important concepts in the Object Oriented design and hence we will briefly discuss it here.
- Coupling is the degree to which one class knows about another class. If the knowledge is only through exposed interfaces (data hiding), it is called loosely coupled. If the knowledge is more like accessing data members directly, it is called tightly coupled. We should try to make our code as loosely coupled as possible. Even though you make some change in a class adhering strictly to the class's API, tight coupling can make other classes that use this class not working properly after the change.

Coupling vs Cohesion

- The term cohesion is used to indicate the degree to which a class has a single, well-focused purpose. The more focused a class is, the higher its cohesiveness, which is a good thing. The key benefit of high cohesion is that such classes are typically much easier to maintain than classes with low cohesion. Another benefit of high cohesion is that classes with a well-focused purpose tend to be more reusable than other classes.
- **In summary, loose coupling and high cohesion are desirable, whereas tight coupling and less cohesion can lead you to problems in the long run.**

Advantages of object oriented software development

- Some of the advantages of object oriented software development are:
- Less maintenance cost mostly because it is modular.
- Better code reusability due to features such as inheritance and hence faster development.
- Improved code reliability and flexibility
- Easy to understand due to realworld modelling.
- Better abstraction at object level.
- Reduced complexity during the transitions from one development phase to another.

What is Software Component Reuse?

- Software component reuse is the software engineering practice of creating new software applications from existing components, rather than designing and building them from scratch. Reusable components can be requirements specifications, design documents, source code, user interfaces, user documentation, or any other items associated with software. All products resulting from the software development life cycle have the potential for reuse.
-

System Development Methodology

A system development methodology refers to the framework that is used to structure, plan, and control the process of developing an information system.

A wide variety of such frameworks have evolved over the years, each with its own recognized strengths and weaknesses.

One system development methodology is not necessarily suitable for use by all projects. Each of the available methodologies is best suited to specific kinds of projects, based on various technical, organizational, project and team considerations.

Types of System Development Methodologies

1 – Waterfall

2 – Prototyping

3 – Incremental

4 – Spiral

5 – Rapid Application Development (RAD)

Object Oriented Software Estimation

- Cost Estimation
- Time Estimation
- Effort Estimation
- Size Estimation