CSE 411
Software Engineering and System Analysis and Design

Topic 6: Software Design

Software Design

Software design is a mechanism to transform user requirements into some suitable form, which helps the programmer in software coding and implementation.

The software design phase is the first step in SDLC (Software Design Life Cycle), which moves the concentration from the problem domain to the solution domain.

Software Design Levels

Software design process have two levels:

01 Focus on deciding which modules are needed, their specification, relationship and interface among them

02 Focus on planning and specifying the internal structure and processing detail

Objectives of Software Design

Correctness: Software design should be correct as per requirement.

Completeness: The design should have all components like data structures, modules, and external interfaces, etc.

Efficiency: Resources should be used efficiently by the program.

Flexibility: Able to modify on changing needs.

Consistency: There should not be any inconsistency in the design.

Maintainability: The design should be so simple so that it can be easily maintainable by other designers.

Software Design Principles

Software Design Principles

Software design principles are concerned with providing means to handle the complexity of the design process effectively. Effectively managing the complexity will not only reduce the effort needed for design but can also reduce the scope of introducing errors during design.

Following are the principles of Software Design:

- Problem Partitioning
- Abstraction
- Modularity
- Strategy of Design -
    1. Top-down Approach
    2. Bottom-up Approach

Problem Partitioning

For small problem, we can handle the entire problem at once but for the significant problem, divide the problems and conquer the problem it means to divide the problem into smaller pieces so that each piece can be captured separately.

Benefits of Problem Partitioning:

- Software is easy to understand
- Software becomes simple
- Software is easy to test
- Software is easy to modify
- Software is easy to maintain
- Software is easy to expand

Abstraction

An abstraction is a tool that enables a designer to consider a component at an abstract level without bothering about the internal details of the implementation. Abstraction can be used for existing element as well as the component being designed.

Here, there are two common abstraction mechanisms

- Functional Abstraction
- Data Abstraction

Abstraction

Functional Abstraction

A module is specified by the method it performs.

The details of the algorithm to accomplish the functions are not visible to the user of the function.

Functional abstraction forms the basis for Function oriented design approaches.

Data Abstraction

Details of the data elements are not visible to the users of data. Data Abstraction forms the basis for Object Oriented design approaches.

Modularity

Modularity specifies to the division of software into separate modules which are differently named and addressed and are integrated later on in to obtain the completely functional software. It is the only property that allows a program to be intellectually manageable.

The desirable properties of a modular system are:

- Each module is a well-defined system that can be used with other applications.
- Each module has single specified objectives.
- Modules can be separately compiled and saved in the library.
- Modules should be easier to use than to build.
- Modules are simpler from outside than inside.

Modularity

Modular design reduces the design complexity and results in easier and faster implementation by allowing parallel development of various parts of a system. We discuss a different section of modular design in detail in this section:

1. Functional Independence: Functional independence is achieved by developing functions that perform only one kind of task and do not excessively interact with other modules. Independence is important because it makes implementation more accessible and faster. The independent modules are easier to maintain, test, and reduce error propagation and can be reused in other programs as well. Thus, functional independence is a good design feature which ensures software quality.

It is measured using two criteria:

Cohesion: It measures the relative function strength of a module.

Coupling: It measures the relative interdependence among modules.

Modularity

2. Information hiding: The fundamental of Information hiding suggests that modules can be characterized by the design decisions that protect from the others, i.e., In other words, modules should be specified that data include within a module is inaccessible to other modules that do not need for such information.

Strategy of Design

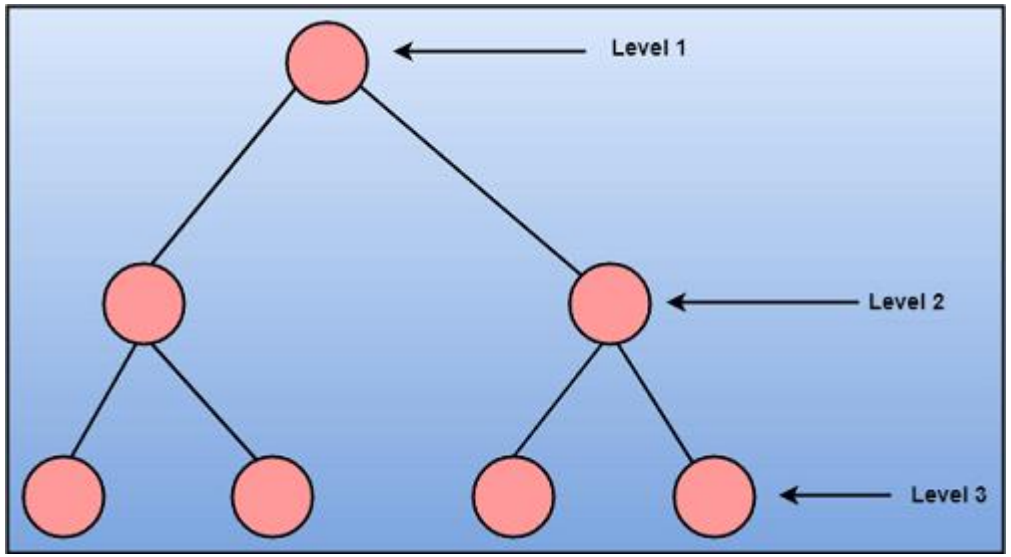
A good system design strategy is to organize the program modules in such a method that are easy to develop and latter too, change. Structured design methods help developers to deal with the size and complexity of programs.

To design a system, there are two possible approaches:

- Top-down Approach
- Bottom-up Approach

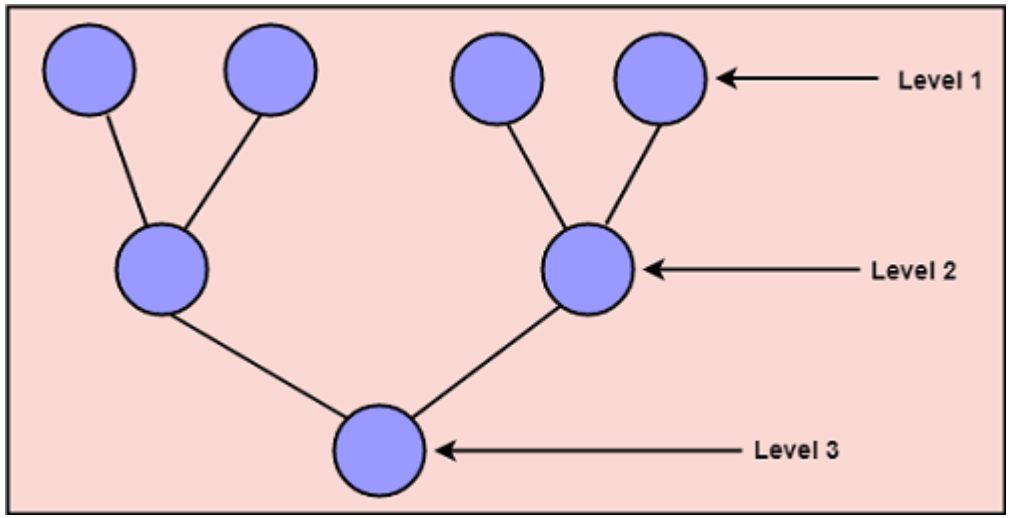
Strategy of Design

1. Top-down Approach: This approach starts with the identification of the main components and then decomposing them into their more detailed sub-components.



Strategy of Design

2. Bottom-up Approach: A bottom-up approach begins with the lower details and moves towards up the hierarchy, as shown in fig. This approach is suitable in case of an existing system.

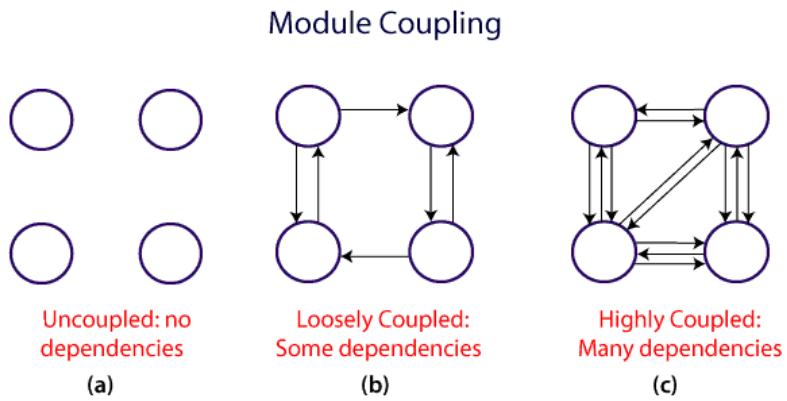


Coupling and Cohesion

Module Coupling

In software engineering, the coupling is the degree of interdependence between software modules. Two modules that are tightly coupled are strongly dependent on each other. However, two modules that are loosely coupled are not dependent on each other. Uncoupled modules have no interdependence at all within them.

The various types of coupling techniques are shown in fig:



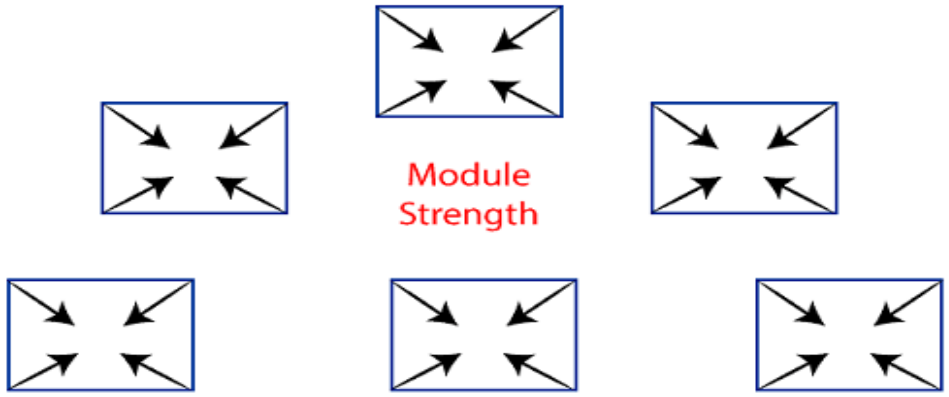
Module Coupling

A good design is the one that has low coupling.

A design with high coupling will have more errors.

Module Cohesion

- In computer programming, cohesion defines to the degree to which the elements of a module belong together.
- In highly cohesive systems, functionality is strongly related.
- Cohesion is an ordinal type of measurement and is generally described as "high cohesion" or "low cohesion."



Cohesion= Strength of relations within Modules

Coupling VS Cohesion

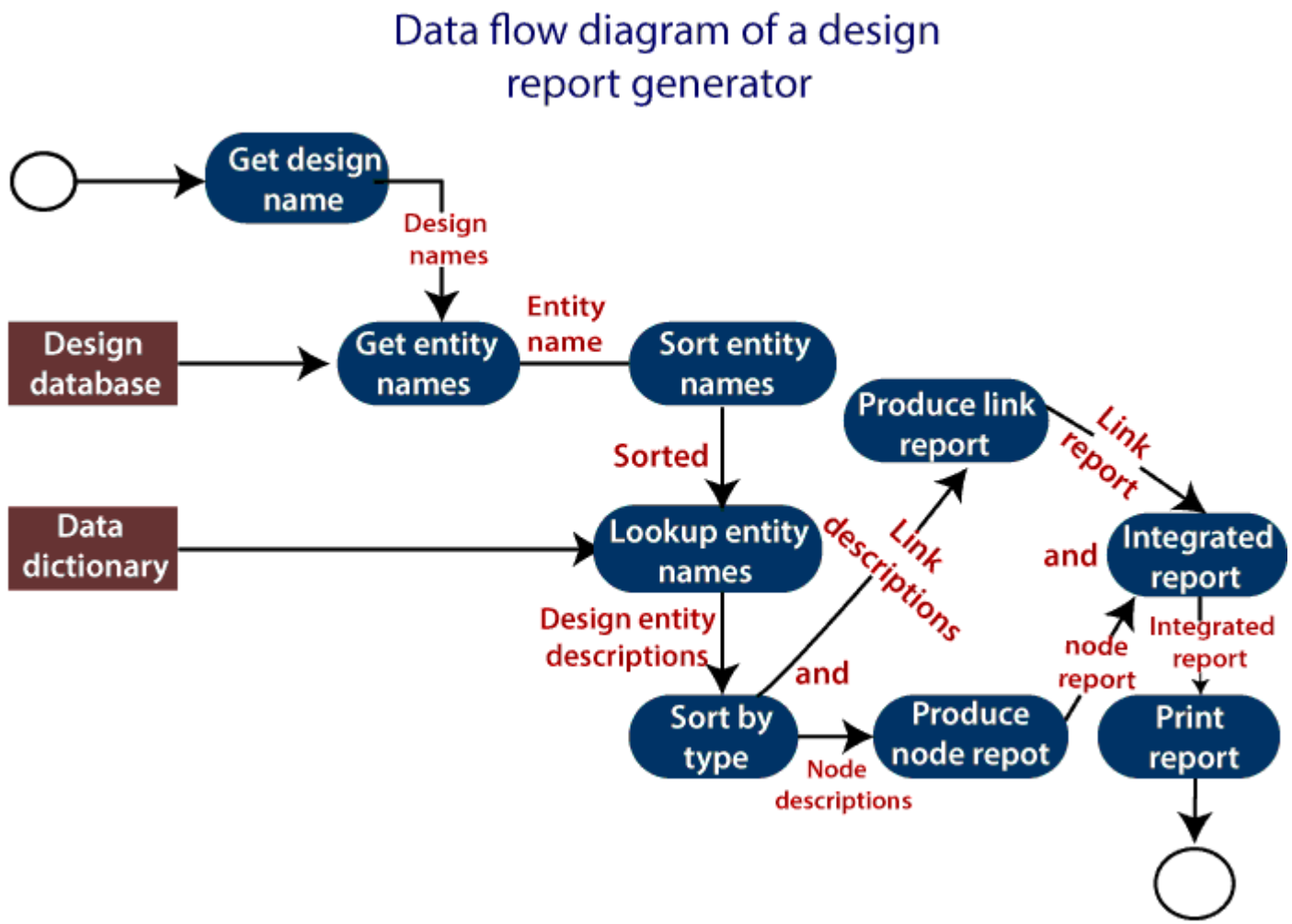
Coupling	Cohesion
Coupling is also called Inter-Module Binding.	Cohesion is also called Intra-Module Binding.
Coupling shows the relationships between modules.	Cohesion shows the relationship within the module.
Coupling shows the relative **independence** between the modules.	Cohesion shows the module's relative **functional** strength.
While creating, you should aim for low coupling, i.e., dependency among modules should be less.	While creating you should aim for high cohesion, i.e., a cohesive component/ module focuses on a single function (i.e., single-mindedness) with little interaction with other modules of the system.
In coupling, modules are linked to the other modules.	In cohesion, the module focuses on a single thing.

Design Methods

Function Oriented Design

Function Oriented design is a method to software design where the model is decomposed into a set of interacting units or modules where each unit or module has a clearly defined function. Thus, the system is designed from a functional viewpoint.

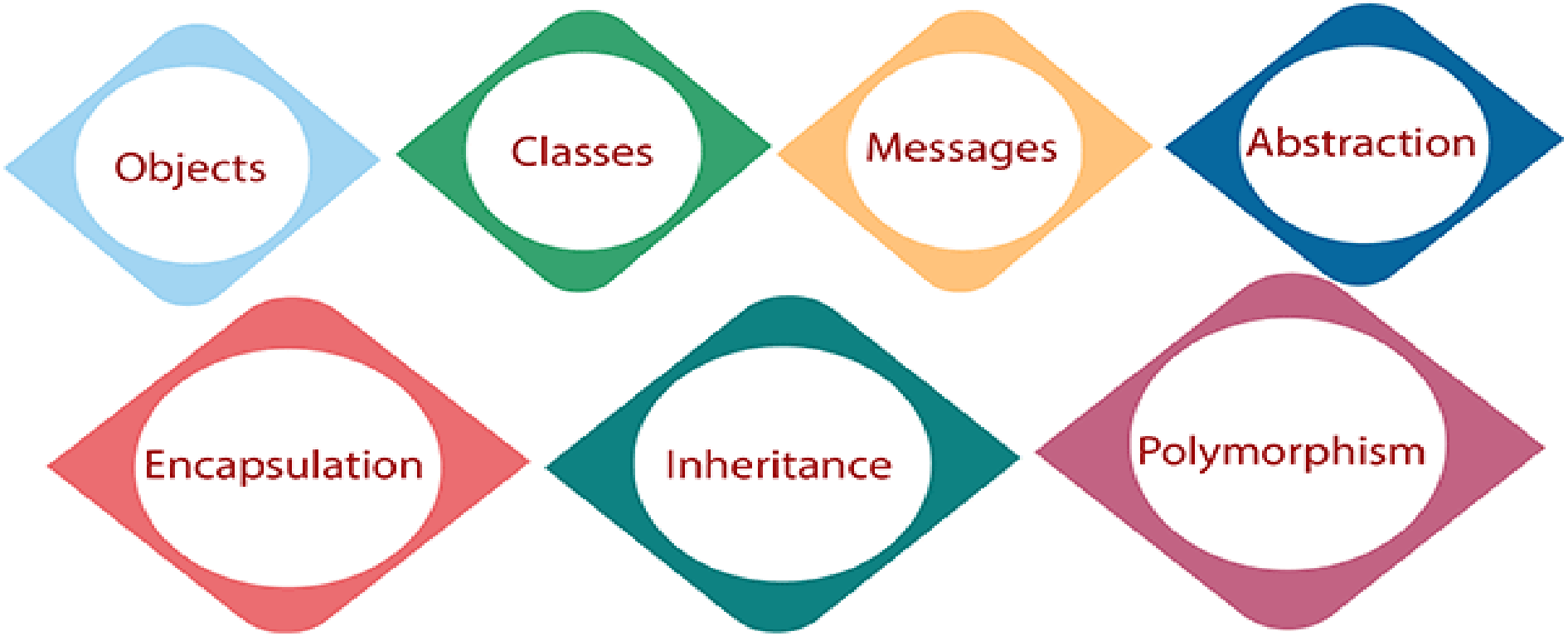
Function Oriented Design



Object-Oriented Design

In the object-oriented design method, the system is viewed as a collection of objects (i.e., entities). The state is distributed among the objects, and each object handles its state data. For example, in a Library Automation Software, each library representative may be a separate object with its data and functions to operate on these data. The tasks defined for one purpose cannot refer or change data of other objects. Objects have their internal data which represent their state. Similar objects create a class. In other words, each object is a member of some class. Classes may inherit features from the superclass.

Object-Oriented Design



Object-Oriented Design

Objects: All entities involved in the solution design are known as objects. For example, person, banks, company, and users are considered as objects. Every entity has some attributes associated with it and has some methods to perform on the attributes.

Classes: A class is a generalized description of an object. An object is an instance of a class. A class defines all the attributes, which an object can have and methods, which represents the functionality of the object.

Messages: Objects communicate by message passing. Messages consist of the integrity of the target object, the name of the requested operation, and any other action needed to perform the function. Messages are often implemented as procedure or function calls.

Object-Oriented Design

Inheritance: OOD allows similar classes to stack up in a hierarchical manner where the lower or sub-classes can import, implement, and re-use allowed variables and functions from their immediate superclasses. This property of OOD is called an inheritance. This makes it easier to define a specific class and to create generalized classes from specific ones.

Polymorphism: OOD languages provide a mechanism where methods performing similar tasks but vary in arguments, can be assigned the same name. This is known as polymorphism, which allows a single interface is performing functions for different types. Depending upon how the service is invoked, the respective portion of the code gets executed.

Object-Oriented Design

Abstraction In object-oriented design, complexity is handled using abstraction. Abstraction is the removal of the irrelevant and the amplification of the essentials.

Encapsulation: Encapsulation is also called an information hiding concept. The data and operations are linked to a single unit. Encapsulation not only bundles essential information of an object together but also restricts access to the data and methods from the outside world.

CODING

The coding is the process of transforming the design of a system into a computer language format.

Coding Standards

Indentation: Proper and consistent indentation is essential in producing easy to read and maintainable programs. Indentation should be used to:

- Emphasize the body of a control structure such as a loop or a select statement.
- Emphasize the body of a conditional statement
- Emphasize a new scope block

Coding Standards

Inline comments: Inline comments analyze the functioning of the subroutine, or key aspects of the algorithm shall be frequently used.

Rules for limiting the use of global: These rules file what types of data can be declared global and what cannot.

Coding Standards

Structured Programming: Structured (or Modular) Programming methods shall be used. "GOTO" statements shall not be used as they lead to "spaghetti" code, which is hard to read and maintain.

Naming conventions for global variables, local variables, and constant identifiers: A possible naming convention can be that global variable names always begin with a capital letter, local variable names are made of small letters, and constant names are always capital letters.

Coding Standards

Error return conventions and exception handling system: Different functions in a program report the way error conditions are handled should be standard within an organization. For example, different tasks while encountering an error condition should either return a 0 or 1 consistently.

End of Topic 6