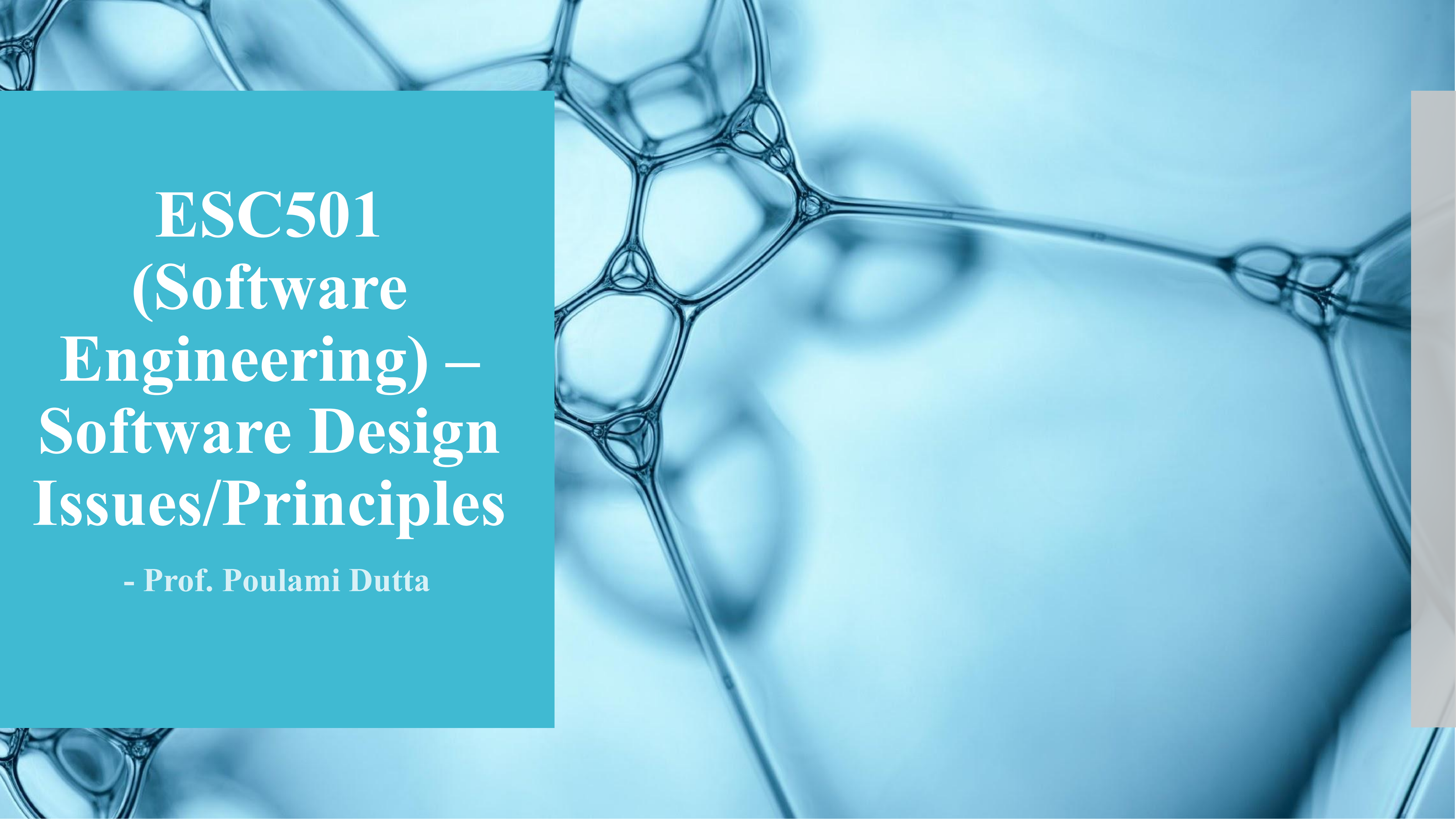


ESC501 (Software Engineering) – Software Design Issues/Principles

- Prof. Poulami Dutta

Intended Learning Outcomes (ILOs)

- Identify the software design activities.
- Identify the items to be designed during the preliminary and detailed design activities.
- Identify the primary differences between analysis and design activities.
- Identify the important items developed during the software design phase.
- State the important desirable characteristics of a good software design.

Intended Learning Outcomes (ILOs)

- State what cohesion means.
- Classify the different types of cohesion that a module may possess.
- State what coupling means.
- Classify the different types of coupling between modules.
- State when a module can be called functionally independent of other modules.
- State why functional independence is the key factor for a good software design.
- State the salient features of a function-oriented design approach.
- State the salient features of an object-oriented design approach.
- Differentiate between function-oriented and object-oriented design approach.

Software Design Principles

- Problem Partitioning
- Abstraction
- Modularity
  - Cohesion
  - Coupling
- Strategy of Design

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.
- Goal is to divide the problem into manageable pieces.
- These pieces cannot be entirely independent of each other as they together form the system. They have to cooperate and communicate to solve the problem. This communication adds complexity.
- As the number of pieces increase, the cost and complexity of partition also increases.

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.
- 2 common abstraction mechanisms:
- Functional Abstraction
- Data 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**.

Function -Oriented Design

- Salient Features:
- A system is viewed as something that performs a set of functions.
- Starting at this high-level view of the system, each function is successively refined into more detailed functions.
- For example, consider a function create-new-library-member which essentially creates the record for a new member, assigns a unique membership number to him, and prints a bill towards his membership charge. This function may consist of the following sub-functions:
  - assign-membership-number
  - create-member-record
  - print-bill
- Each of these sub-functions may be split into more detailed sub-functions and so on.

Function -Oriented Design (Contd.)

- The system state is centralized and shared among different functions, e. g. data such as member-records is available for reference and updation to several functions such as:
- • create-new-member
  • delete-member
  • update-member-record

Object -Oriented Design

- In the object-oriented design approach, the system is viewed as collection of objects (i.e. entities). The state is decentralized among the objects and each object manages its own state information.
- For example, in a Library Automation Software, each library member may be a separate object with its own data and functions to operate on these data.
- In fact, the functions defined for one object cannot refer or change data of other objects. Objects have their own internal data which define their state. Similar objects constitute a class.
- In other words, each object is a member of some class. Classes may inherit features from super class. Conceptually, objects communicate by message passing.

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. Single large programs are difficult to understand and read due to a large number of reference variables, control paths, global variables, etc.

Desirable Properties of a Modular System

- 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.

Advantages of Modularity

- It allows large programs to be written by several or different people.
- It encourages the creation of commonly used routines to be placed in the library and used by other programs.
- It simplifies the overlay procedure of loading a large program into main storage.
- It provides more checkpoints to measure progress.
- It provides a framework for complete testing, more accessible to test.
- It produced the well designed and more readable program.

Disadvantages of Modularity

- Execution time maybe, but not certainly, longer.
- Storage size perhaps, but is not certainly, increased.
- Compilation and loading time may be longer.
- Inter-module communication problems may be increased.
- More linkage required, run-time may be longer, more source lines must be written, and more documentation has to be done.

Modular Design

- Modular design reduces the design complexity and results in easier and faster implementation by allowing parallel development of various parts of a system.

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.
- A module having high cohesion and low coupling is said to be functionally independent of other modules.

Need for Functional Independence

- Functional independence is a key to any good design.
- **Error isolation:** Functional independence reduces error propagation. The reason behind this is that if a module is functionally independent, its degree of interaction with the other modules is less.
- Therefore, any error existing in a module would not directly effect the other modules.
- **Scope of reuse:** Reuse of a module becomes possible because each module does some well-defined and precise function, and the interaction of the module with the other modules is simple and minimal.
- Therefore, a cohesive module can be easily taken out and reused in a different program.

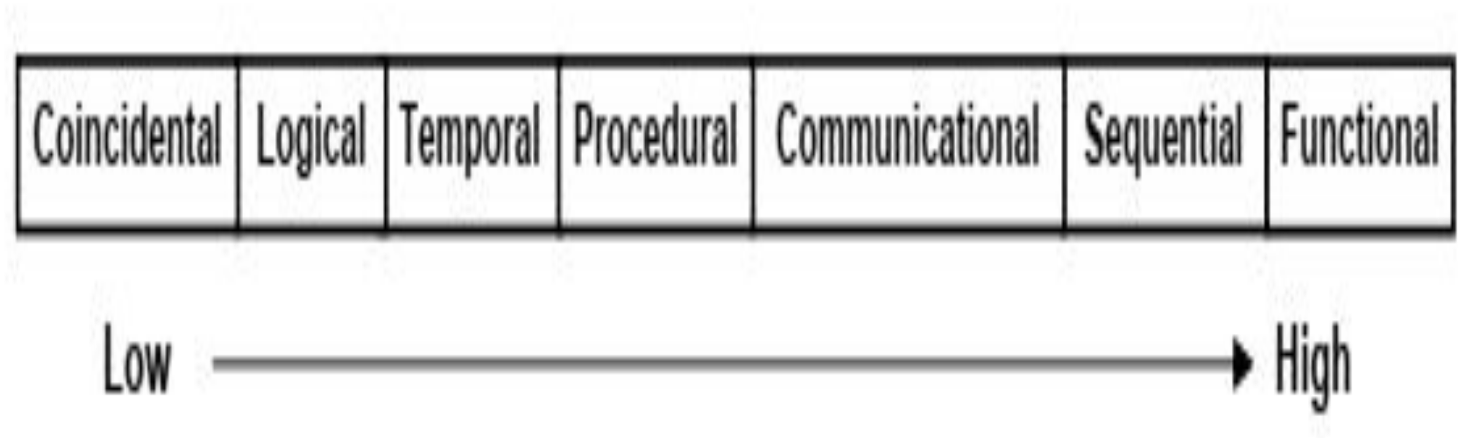
Need for Functional Independence (Contd.)

- Functional independence is a key to any good design.
- **Understandability:** Complexity of the design is reduced, because different modules can be understood in isolation as modules are more or less independent of each other.

COHESION

- A good software design implies clean decomposition of the problem into modules, and the neat arrangement of these modules in a hierarchy.
- Primary characteristics of neat module decomposition are high cohesion and low coupling.
- Cohesion is a measure of functional strength of a module.
- A module having high cohesion and low coupling is said to be functionally independent of other modules.
- A functionally independent module has minimal interaction with other modules.

Classification of Cohesion



Coincidental and Logical Cohesion

- **Coincidental cohesion:** A module is said to have coincidental cohesion, if it performs a set of tasks that relate to each other very loosely, if at all.
- In this case, the module contains a random collection of functions. It is likely that the functions have been put in the module out of pure coincidence without any thought or design.
- **Logical cohesion:** A module is said to be logically cohesive, if all elements of the module perform similar operations, e.g. error handling, data input, data output, etc.

Temporal and Procedural Cohesion

- **Temporal cohesion:** When a module contains functions that are related by the fact that all the functions must be executed in the same time span, the module is said to exhibit temporal cohesion.
- The set of functions responsible for initialization, start-up, shutdown of some process, etc. exhibit temporal cohesion.
- **Procedural cohesion:** A module is said to possess procedural cohesion, if the set of functions of the module are all part of a procedure (algorithm) in which certain sequence of steps have to be carried out for achieving an objective, e.g. the algorithm for decoding a message.

Communicational and Sequential Cohesion

- **Communicational cohesion:** A module is said to have communicational cohesion, if all functions of the module refer to or update the same data structure, e.g. the set of functions defined on an array or a stack.
- **Sequential cohesion:** A module is said to possess sequential cohesion, if the elements of a module form the parts of sequence, where the output from one element of the sequence is input to the next.

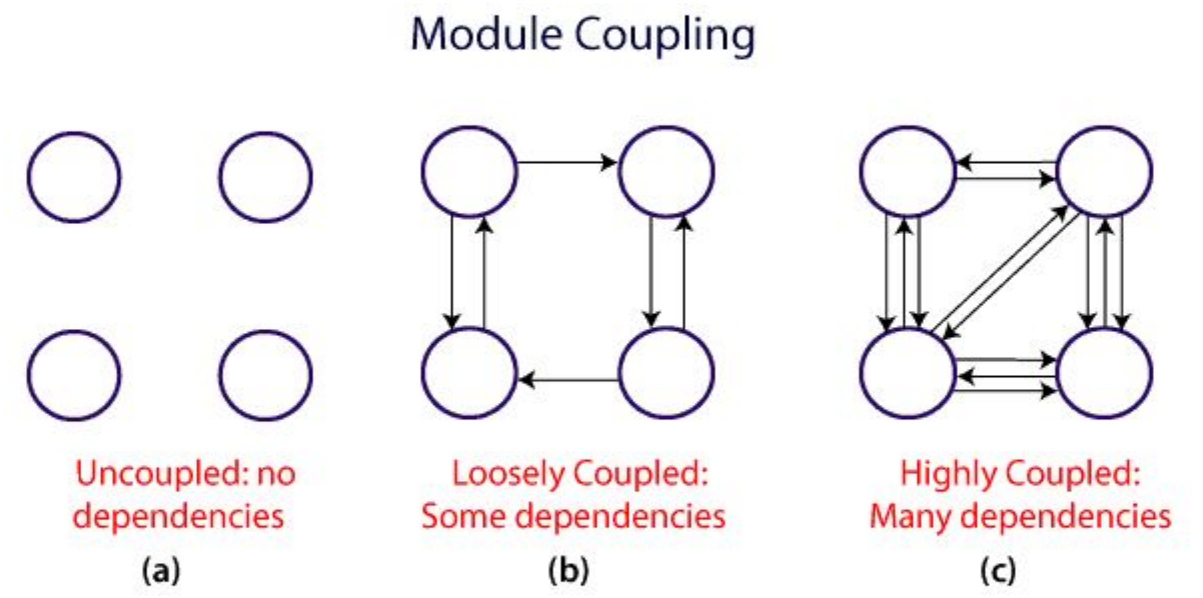
Functional Cohesion

- **Functional cohesion:** Functional cohesion is said to exist, if different elements of a module cooperate to achieve a single function.
- For example, a module containing all the functions required to manage employees' pay-roll exhibits functional cohesion.

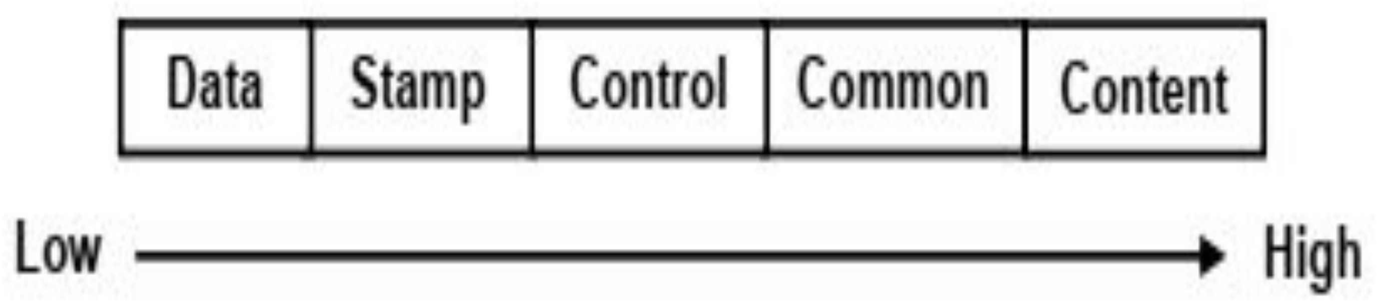
COUPLING

- Coupling between two modules is a measure of the degree of interdependence or interaction between the two modules.
- A module having high cohesion and low coupling is said to be functionally independent of other modules.
- If two modules interchange large amounts of data, then they are highly interdependent. The degree of coupling between two modules depends on their interface complexity.

Module Coupling


Module Coupling
Uncoupled: no dependencies
(a)
Loosely Coupled: Some dependencies
(b)
Highly Coupled: Many dependencies
(c)

Classification of Coupling



Data and Stamp Coupling

- **Data coupling:** Two modules are data coupled, if they communicate through a parameter.
- An example is an elementary data item passed as a parameter between two modules, e.g. an integer, a float, a character, etc.
- This data item should be problem related and not used for the control purpose.
- **Stamp coupling:** Two modules are stamp coupled, if they communicate using a composite data item such as a record in PASCAL or a structure in C.

Control, Common and Content Coupling

- **Control coupling:** Control coupling exists between two modules, if data from one module is used to direct the order of instructions execution in another.
- **Common coupling:** Two modules are common coupled, if they share data through some global data items.
- **Content coupling:** Content coupling exists between two modules, if they share code, e.g. a branch from one module into another module.

Information Hiding

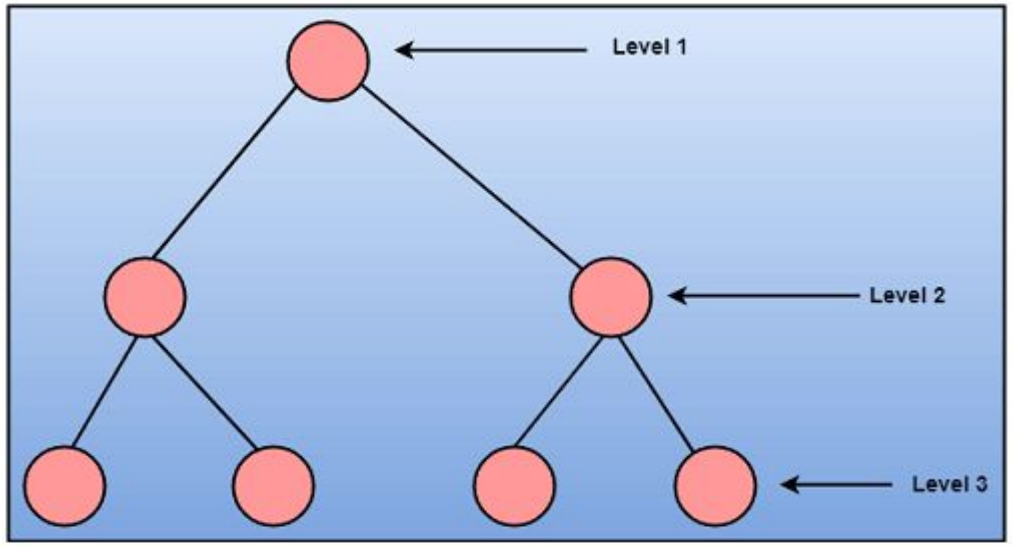
- The fundamental of Information hiding suggests that modules can be characterized by the design decisions that protect from the others, i.e., in other words, modules should be specified that data include within a module is inaccessible to other modules that do not need for such information.
- The use of information hiding as design criteria for modular system provides the most significant benefits when modifications are required during testing and later during software maintenance.
- This is because as most data and procedures are hidden from other parts of the software, inadvertent errors introduced during modifications are less likely to propagate to different locations within the software.

Strategy of Design

- A good system design strategy is to organize the program modules in such a method that are easy to develop and change.
- Structured design methods help developers to deal with the size and complexity of programs. Analysts generate instructions for the developers about how code should be composed and how pieces of code should fit together to form a program.
- To design a system, there are two possible approaches:
- Top-down Approach
- Bottom-up Approach

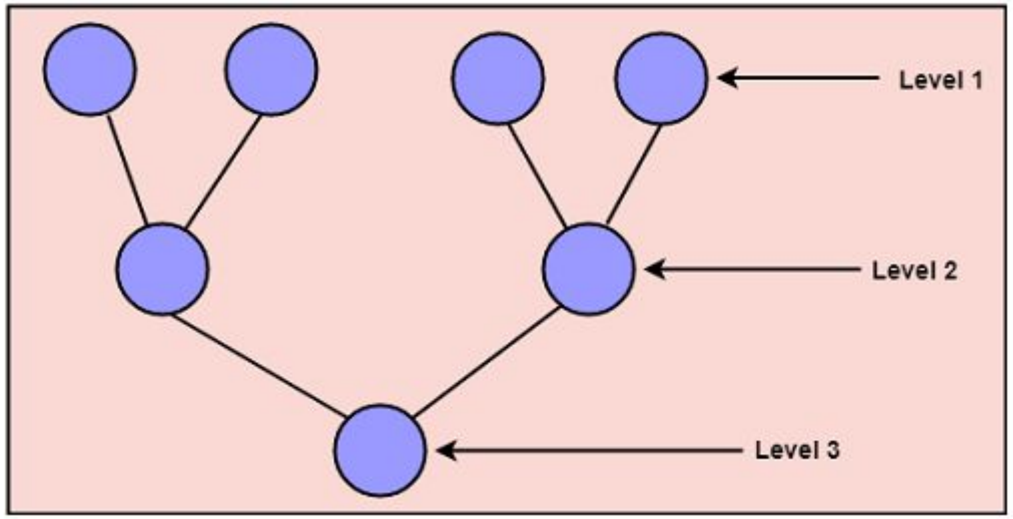
Top-down Approach

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



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.



Structured Analysis

- Structured analysis is used to carry out the top-down decomposition of a set of high-level functions depicted in the problem description and to represent them graphically.
- During structured analysis, functional decomposition of the system is achieved. That is, each function that the system performs is analyzed and hierarchically decomposed into more detailed functions.
- Structured analysis technique is based on the following essential underlying principles:
  • Top-down decomposition approach.
  • Divide and conquer principle. Each function is decomposed independently.
  • Graphical representation of the analysis results using Data Flow Diagrams (DFDs).

Importance of DFDs in a Good Software Design

- The main reason why the DFD technique is so popular is probably because of the fact that DFD is a very simple formalism – it is simple to understand and use.
- Starting with a set of high-level functions that a system performs, a DFD model hierarchically represents all sub-functions.
- The data flow diagramming technique also follows a very simple set of intuitive concepts and rules. DFD is an elegant modeling technique that turns out to be useful not only to represent the results of structured analysis of a software problem, but also for several other applications such as showing the flow of documents or items in an organization.

Thank You