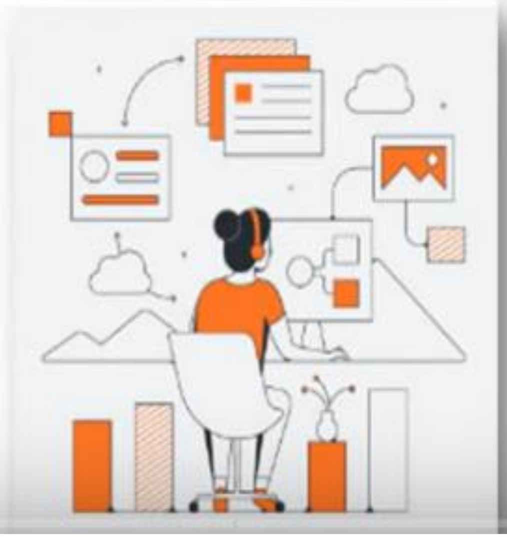
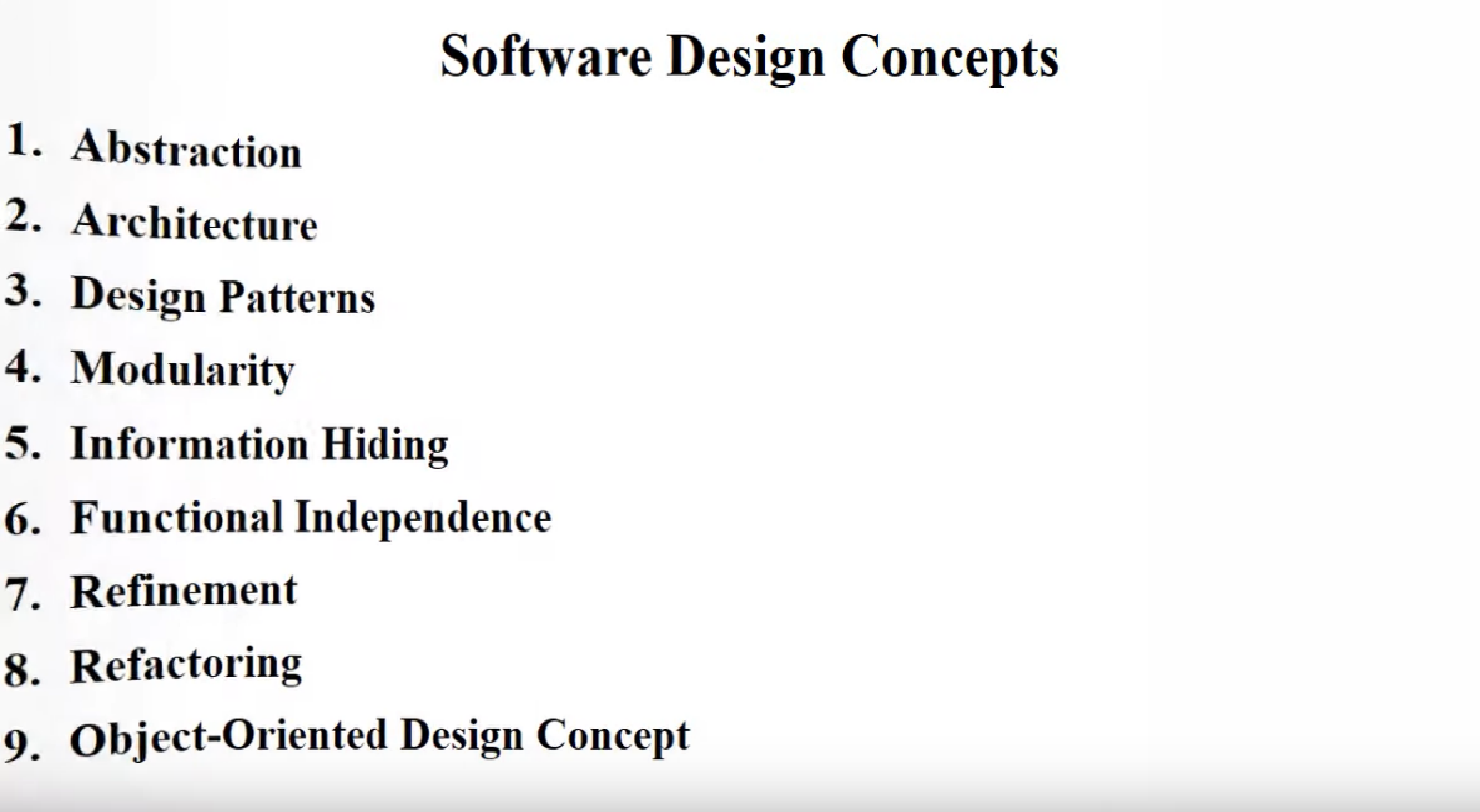
Software Design Concepts

About Software Design Concepts

- The software design concept simply means the idea or principle behind the design.
- It describes how you plan to solve the problem of designing software.
- It also shows the logic or thinking behind how you will design software.
- The software design concept for developing the right software provides a supporting and essential structure or model.





Software Design Concepts

1. Abstraction
2. Architecture
3. Design Patterns
4. Modularity
5. Information Hiding
6. Functional Independence
7. Refinement
8. Refactoring
9. Object-Oriented Design Concept

1. Abstraction

- Abstraction is used to hide background details or unnecessary implementation about the data.
- So that users see only required information.

Type 1: Procedural Abstraction:

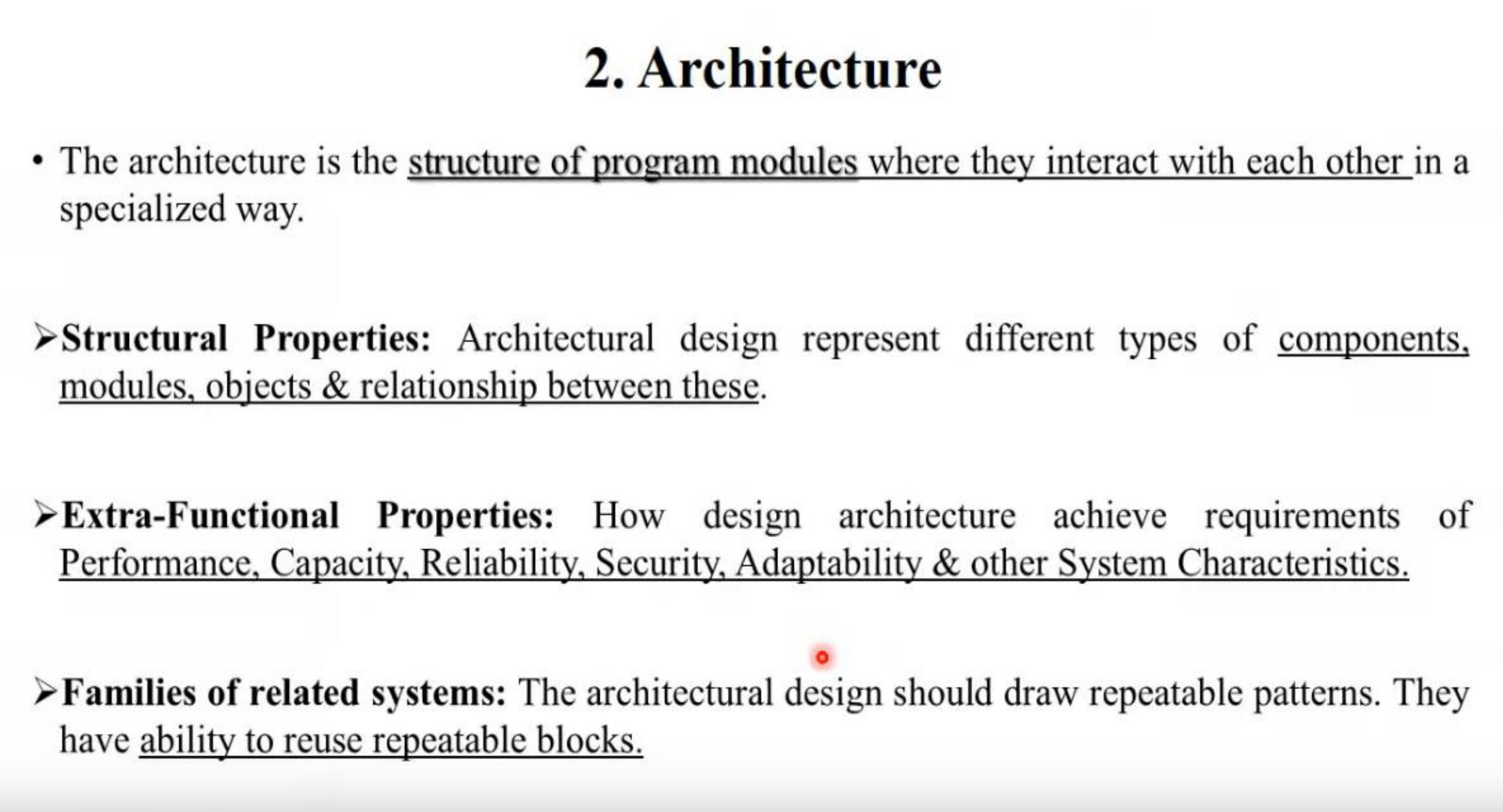
- There is collections of subprograms.
- One is hidden group another is visible group of functionalities. **Example:**

Private:
Fuel_machine()
Set_top_speed()
Develop_engine()

Public:
Turn_on() , Turn_off()
Accelerate(), Break()

Type 2: Data Abstraction:

- Collections of data that describe data objects.
- Show representation data & hide manipulation data.



2. Architecture

- The architecture is the structure of program modules where they interact with each other in a specialized way.

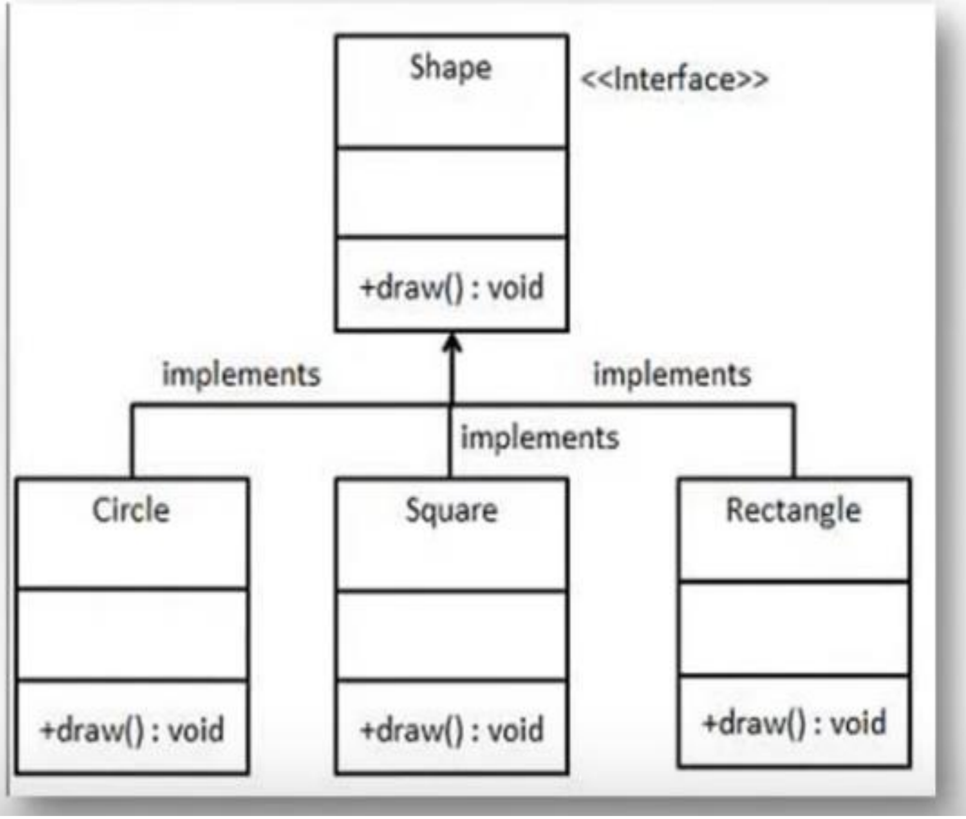
➢**Structural Properties:** Architectural design represent different types of components, modules, objects & relationship between these.

➢**Extra-Functional Properties:** How design architecture achieve requirements of Performance, Capacity, Reliability, Security, Adaptability & other System Characteristics.

➢**Families of related systems:** The architectural design should draw repeatable patterns. They have ability to reuse repeatable blocks.

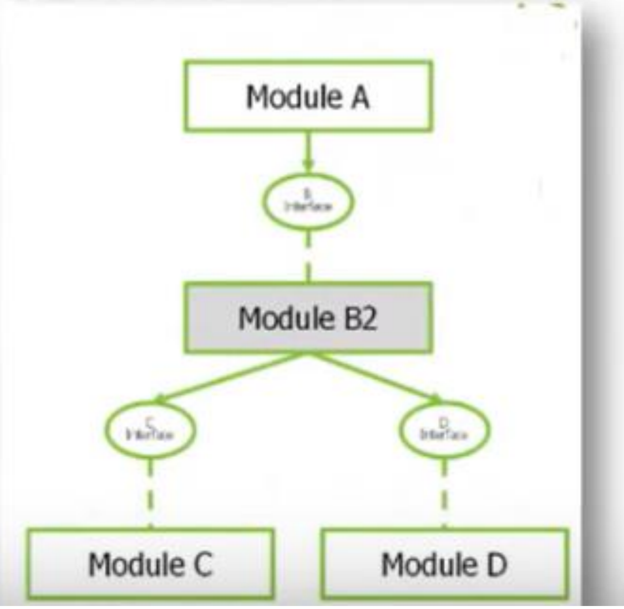
3. Design Patterns

- The pattern simply means a repeated form or design in which the same shape is repeated several times to form a pattern.
- **Example:**



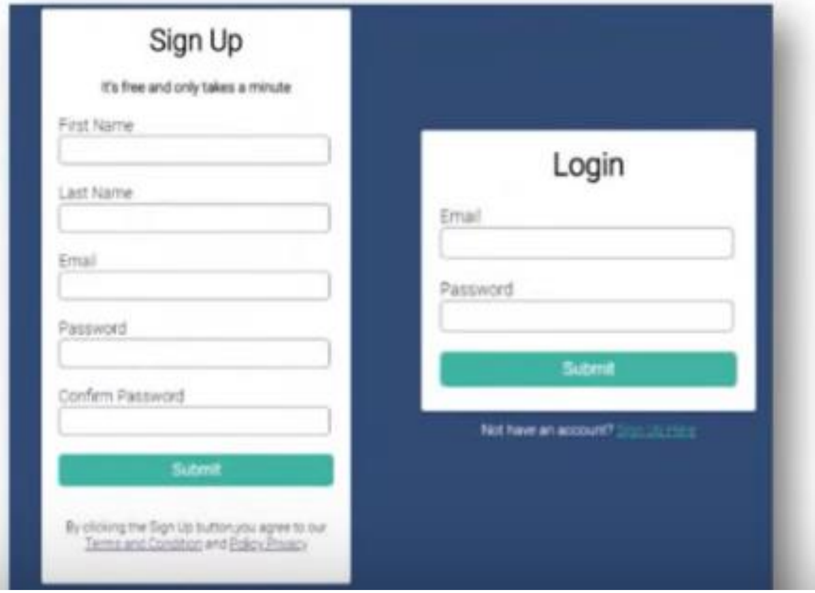
4. Modularity

- Modularity simply means dividing the system or project into smaller parts to reduce the complexity of the system or project.
- After developing the modules, they are integrated together to meet the software requirements.
- Modularizing a design helps to effective development, accommodate changes easily, conduct testing, debugging efficiently and conduct maintenance work easily.



5. Information Hiding

- Modules should be specified and designed in such a way that the data structures and algorithm details of one module are not accessible to other modules.
- They pass only that much information to each other, which is required to accomplish the software functions.
- The way of hiding unnecessary details in modules is referred to as information hiding.



Sign Up

It's free and only takes a minute

First Name

Last Name

Email

Password

Confirm Password

Submit

By clicking the Sign Up button,you agree to our Terms and Condition and Policy Privacy

Login

Email

Password

Submit

Not have an account? Sign Up Here

6. Functional Independence

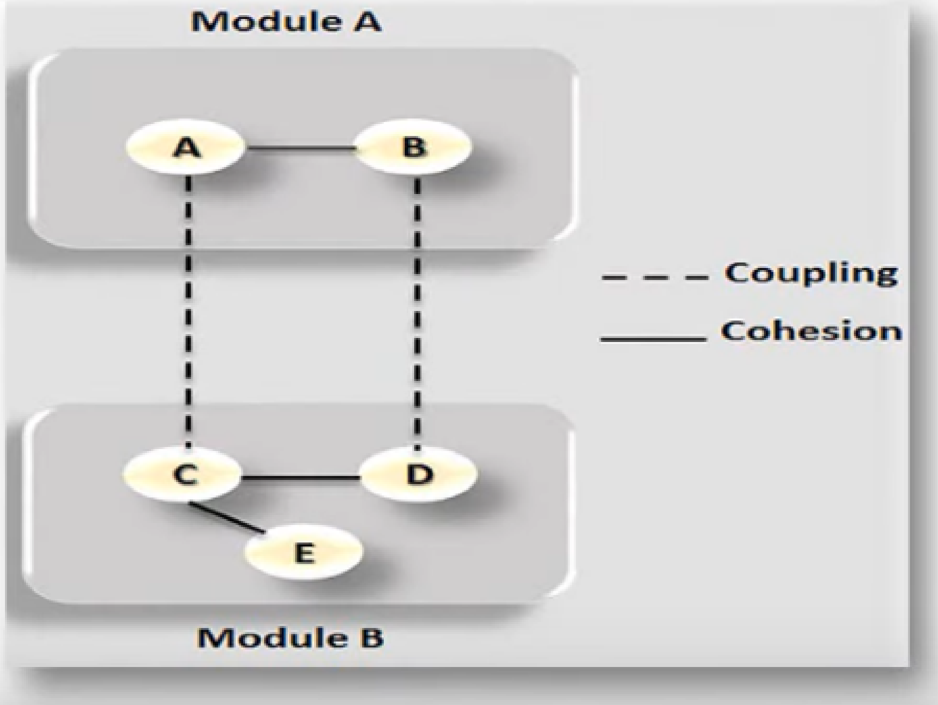
- The functional independence is the concept of separation and related to the concept of modularity, abstraction and information hiding.

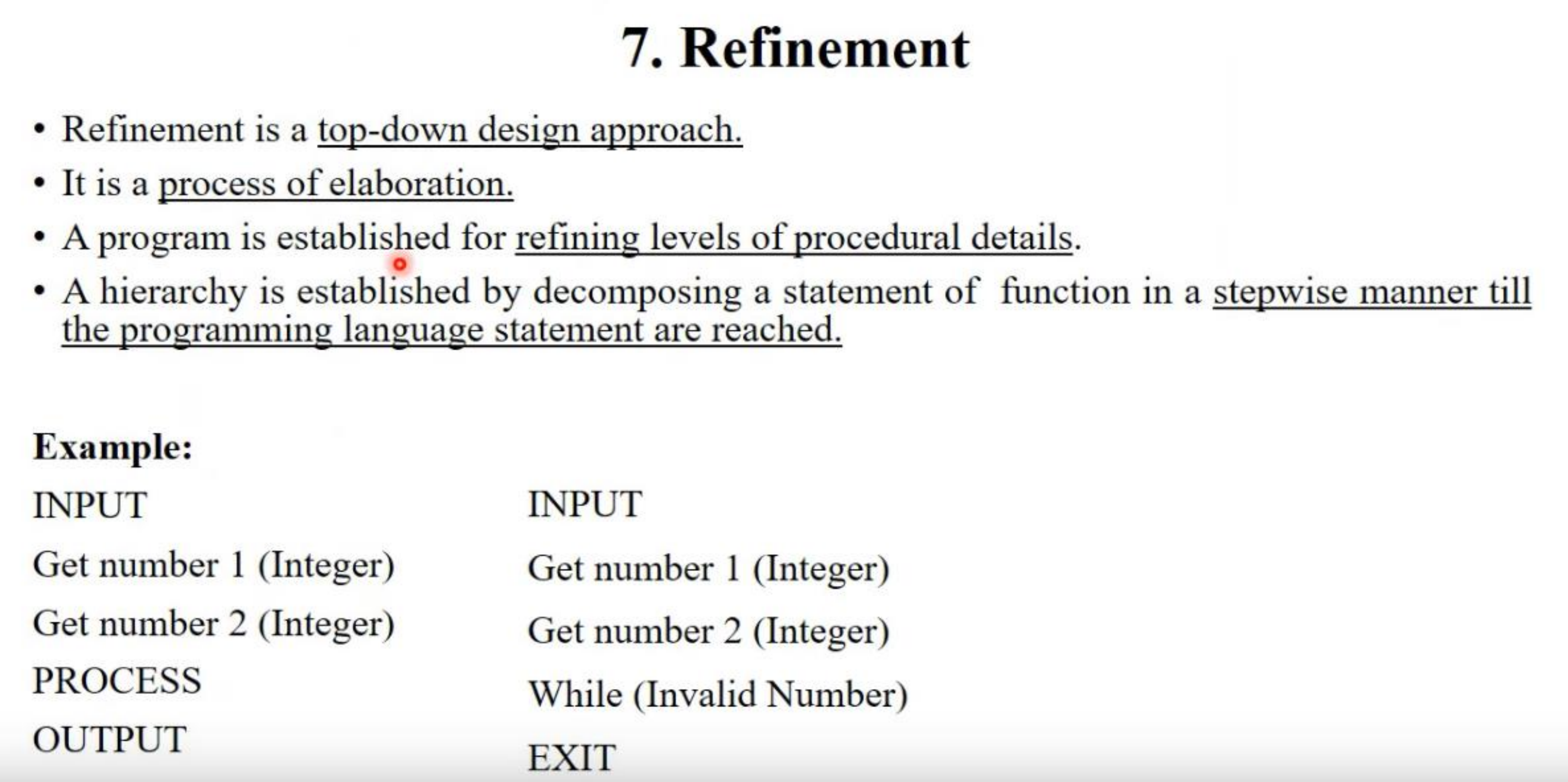
Criteria 1: Coupling

- The degree in which module is "connected" to other module in the system.
- Low Coupling necessary in good software.

Criteria 2: Cohesion

- The degree in which module perform functions in inner module in the system.
- High Cohesion necessary in good software.





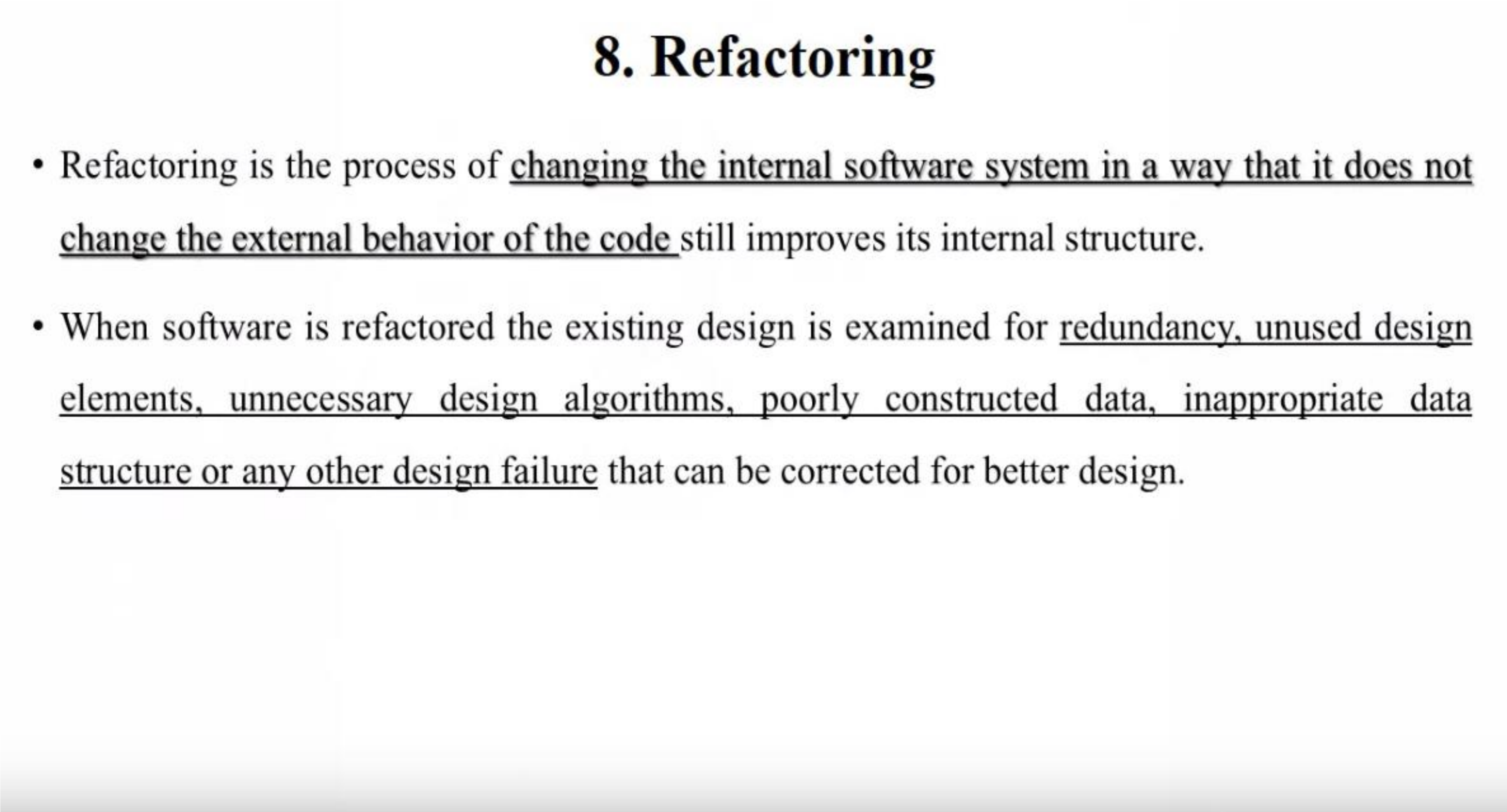
7. Refinement

- Refinement is a top-down design approach.
- It is a process of elaboration.
- A program is established for refining levels of procedural details.
- A hierarchy is established by decomposing a statement of function in a stepwise manner till the programming language statement are reached.

Example:

```
INPUT
Get number 1 (Integer)
Get number 2 (Integer)
PROCESS
OUTPUT
```

```
INPUT
Get number 1 (Integer)
Get number 2 (Integer)
While (Invalid Number)
EXIT
```

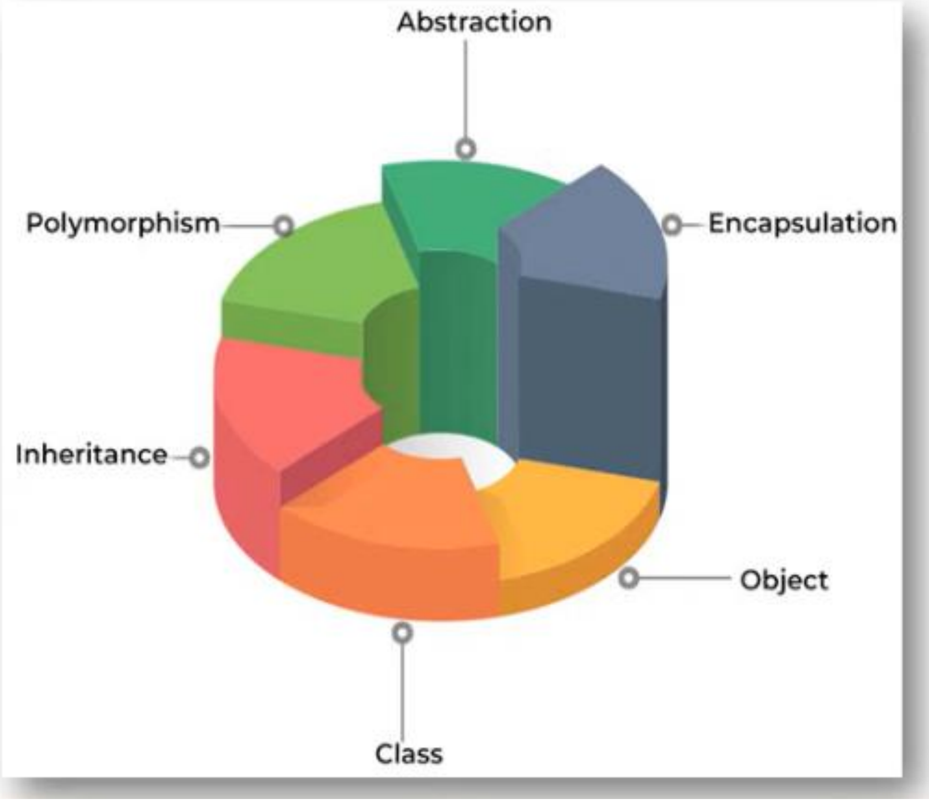


8. Refactoring

- Refactoring is the process of changing the internal software system in a way that it does not change the external behavior of the code still improves its internal structure.
- When software is refactored the existing design is examined for redundancy, unused design elements, unnecessary design algorithms, poorly constructed data, inappropriate data structure or any other design failure that can be corrected for better design.

9. Object Oriented Design Concepts

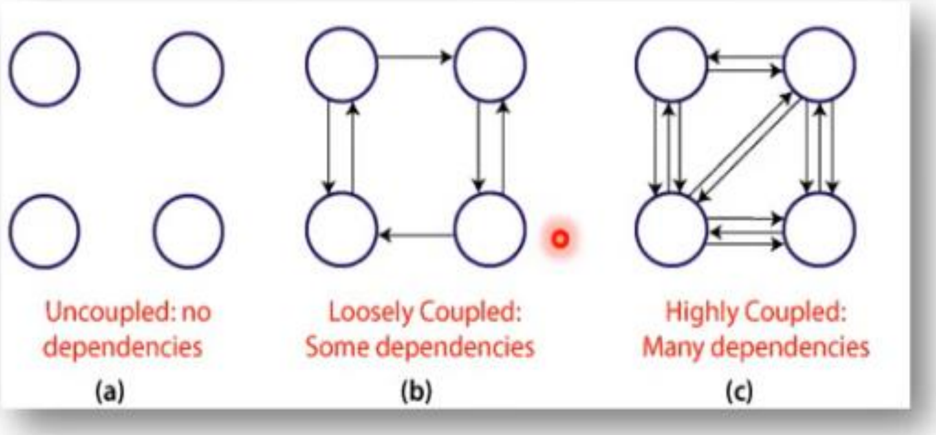
- Object Oriented is a popular design approach for analyzing and designing an application.
- Advantage is that faster, low cost development and creates a high quality software.



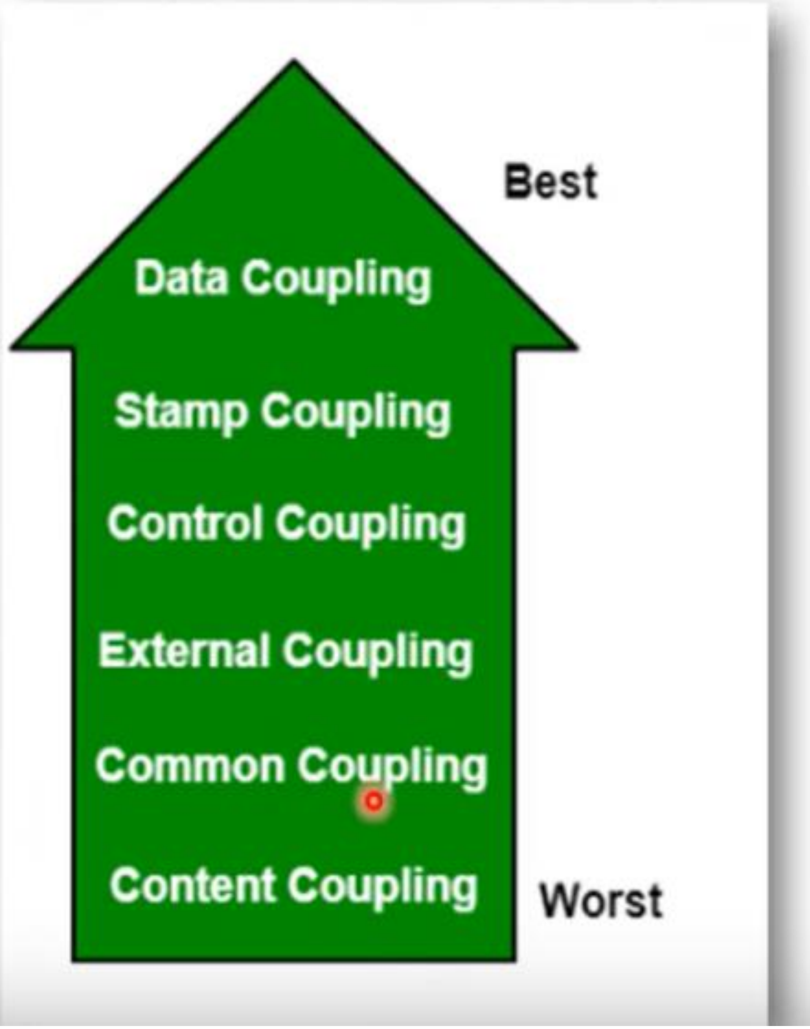
COUPLING

About Coupling

- The coupling is the degree of interdependence or number of relations between software modules.
- Two modules that are tightly coupled are strongly dependent on each other.
- However, two modules that are loosely coupled are not much dependent on each other.
- A **good design** is the one that has Low coupling.
- High coupling generates more errors because they shared large number of data.


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

Types of Coupling



Best

Data Coupling

Stamp Coupling

Control Coupling

External Coupling

Common Coupling

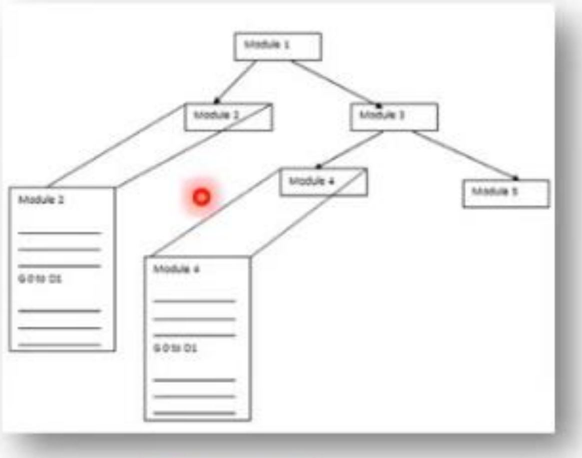
Content Coupling

Worst

Types of Coupling

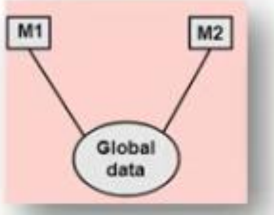
Type 1: Content Coupling

- Here, Two modules are connected as they share the same content like functions, methods.
- When a change is made in one module the other module needs to be updated as well.



Type 2: Common Coupling

- Two modules are common coupled if they share information through some global data items.



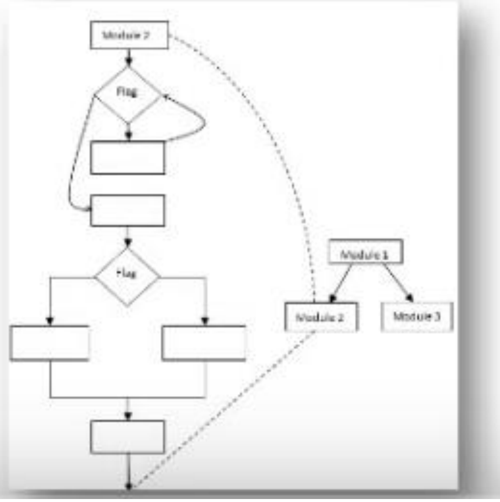
Types of Coupling

Type 3: External Coupling

- When two modules share an externally import data format, communication protocols or device interface.
- This is related to communication to external tools and devices.

Type 4: Control Coupling

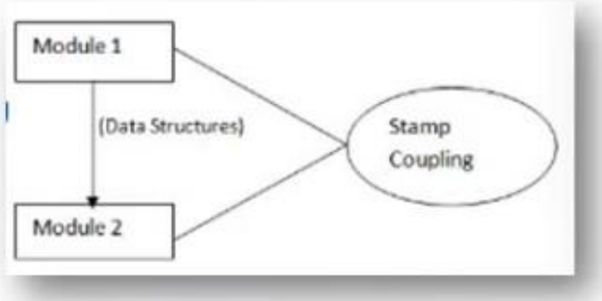
- Control coupling handle functional flow between software modules.
- **Example:** Module 1- Set Flag = 1 then only Module 2 perform action.



Types of Coupling

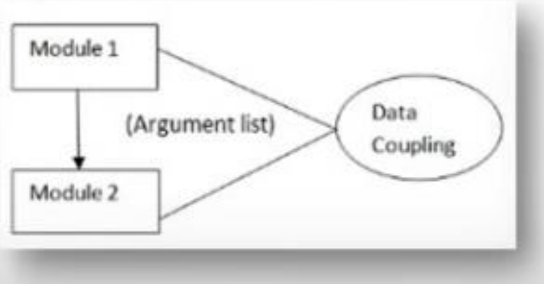
Type 5: Stamp Coupling

- Two modules are stamp coupled if they communicate using composite data items such as Complete Data Structure & objects.
- No junk or unused data shared between the two coupling modules.



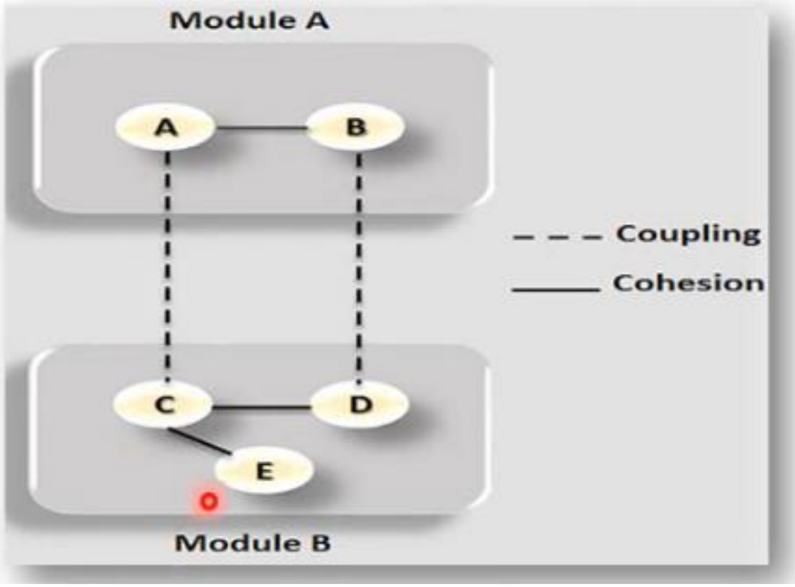
Type 6: Data Coupling

- When data are passed from one modules to another module via argument list or parameters through functional blocks.

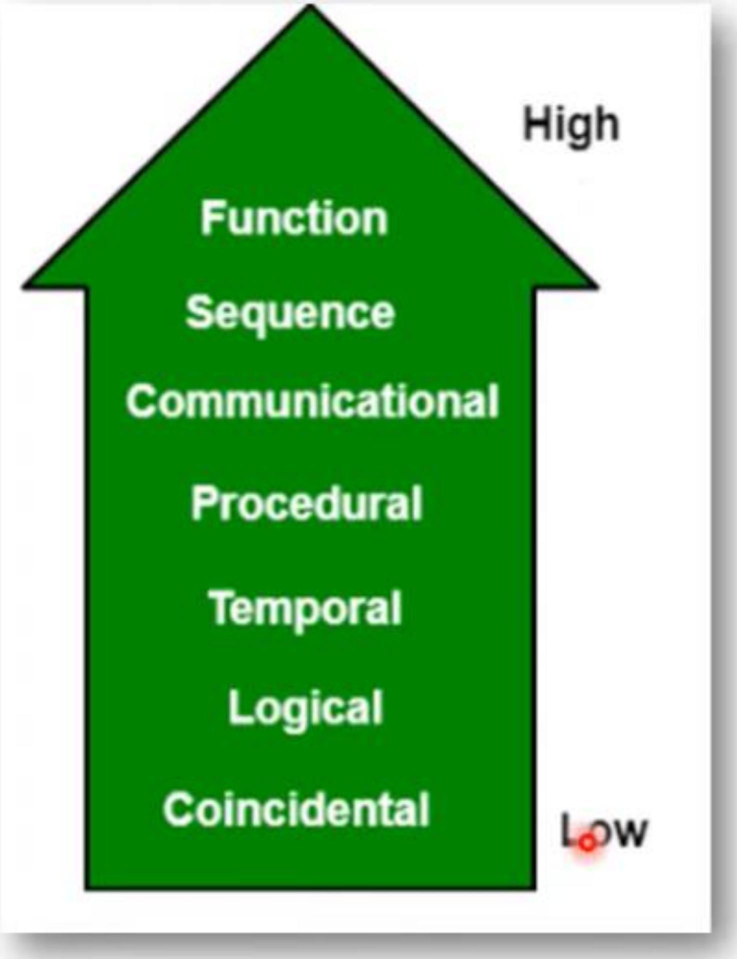


About Cohesion

- Cohesion defines to the degree to which the elements of a module belong together or interrelated.
- Thus, cohesion measures the strength of relationships between pieces of functionality within a given module.
- A **good software design** will have high cohesion.



Types of Cohesion



High

Function

Sequence

Communicational

Procedural

Temporal

Logical

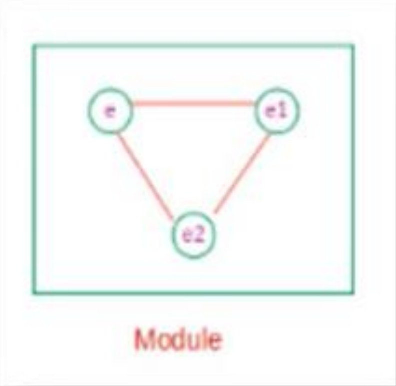
Coincidental

Low

Types of Cohesion

Type 1: Coincidental Cohesion

- It performs a set of tasks that are associated with each other very loosely.
- **Example:** Calculator : ADD, SUB, MUL, DIV



Type 2: Logical Cohesion

- If all the elements of the module perform a similar operation.
- **Example**: Error handling, Sorting, If Type of Record = Student then Display Student Record.

Type 3: Temporal Cohesion

- The activities related in time, Where all methods executed at same time.
- Temporal cohesion is found in the modules of initialization and termination.
- **Example:** Counter = 0, Open student file, Clear(), Initializing the array etc.

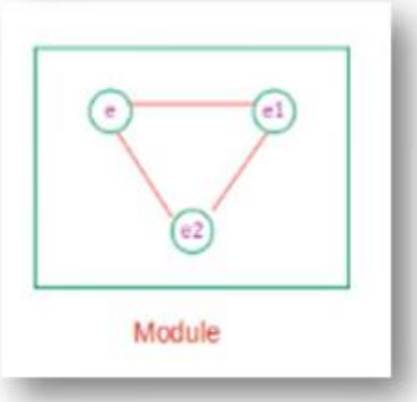
Types of Cohesion

Type 4: Procedural Cohesion

- All parts of a procedure execute in particular sequence of steps for achieving goal.
- **Example:** Calling one function to another function, Loop statements, Reading record etc.

Type 5: Communicational Cohesion

- If all the elements of a module are working on the same input & output data and are accessing that data through the same data structures.
- **Example:** Update record in the database and send it to the printer.



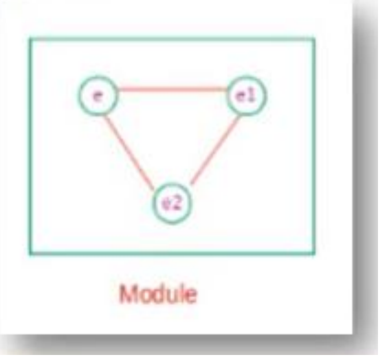
Types of Cohesion

Type 6: Sequence Cohesion

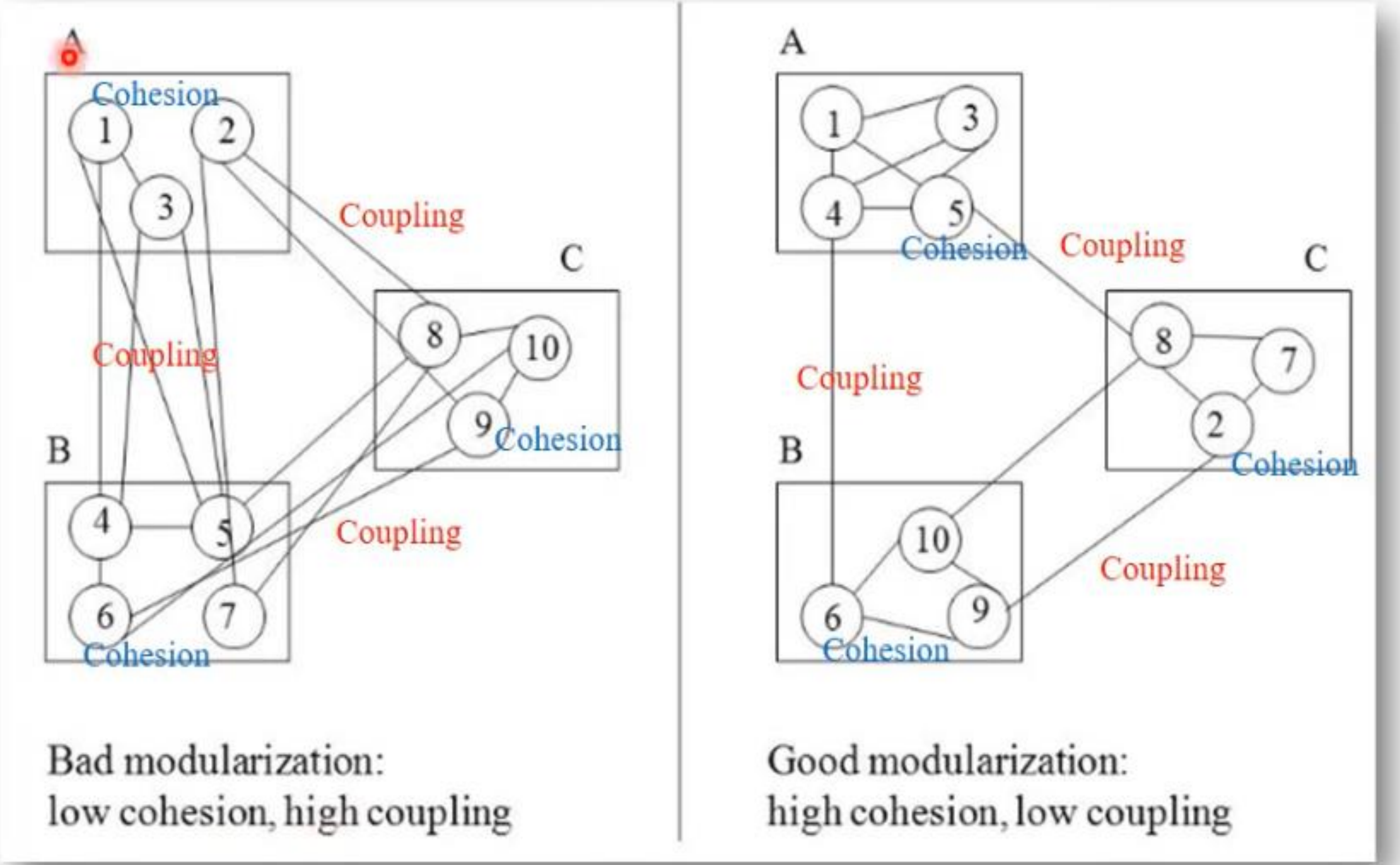
- Output of one element treats as an input to the other elements inside the same module.
- **Example:** Enter the numbers -> Perform Addition of that numbers -> Display Addition.

Type 7: Function Cohesion

- If a single module aims to perform all the similar types of functionalities through its different elements.
- The purpose of functional cohesion is single minded, high, strong and focused.
- **Example:** Railway Reservation System



Good & Bad Software System Design



Bad modularization:
low cohesion, high coupling

Good modularization:
high cohesion, low coupling

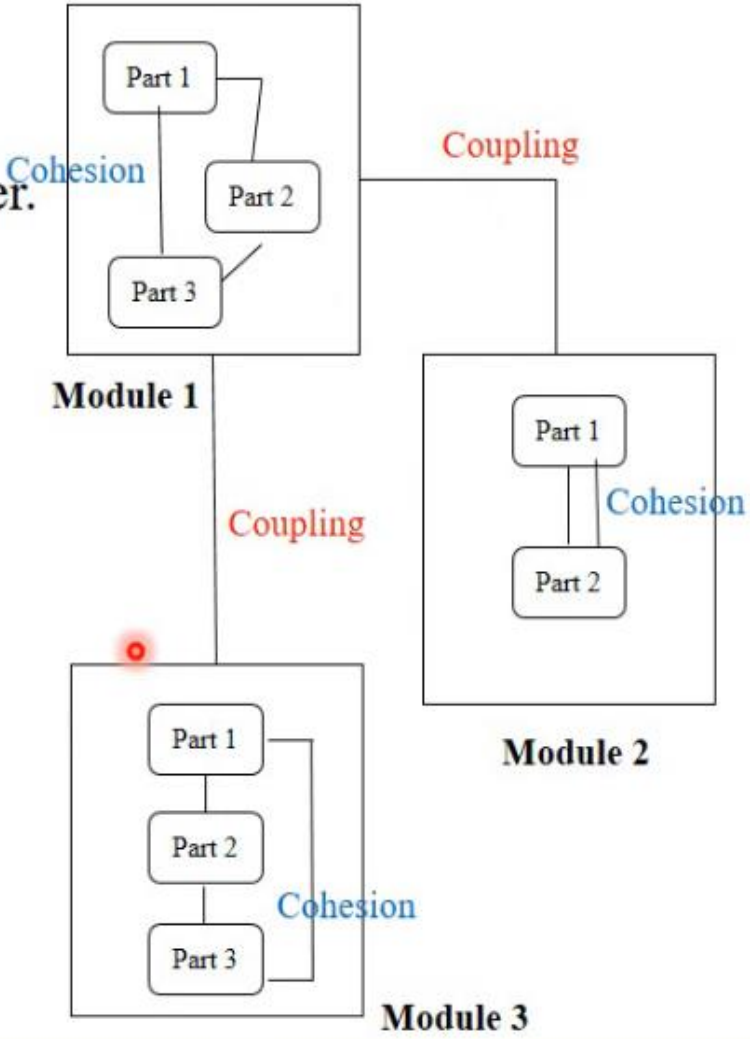
Why High Cohesive & Low Coupling generate good design?

➢Due to Low Coupling

- **Readability:** Modules are easy to understand not complex.
- **Maintainability:** Changes in one module little impact on other.
- **Modularity:** Enhance modules development.
- **Scalability:** Adding new module remove existing one easy.
- **Testability:** Modules are easy to test & debug.

➢Due to High Cohesion

- **Readability:** Related functions easy to understand.
- **Reusability:** Easily Reuse module in another system.
- **Reliability:** Generate overall improvement of system.
- **Testability:** Modules are easy to test & debug.


Part 1
Part 2
Part 3
Cohesion
Module 1
Coupling
Part 1
Part 2
Cohesion
Module 2
Coupling
Part 1
Part 2
Part 3
Cohesion
Module 3