Function Oriented System Design

Software Design

- Deriving a solution which satisfies software requirements.

- The activities carried out during the design phase called as design process, transform the SRS document into the design document.

Stages of Design

- **Problem understanding**
  - Look at the problem from different angles to discover the design requirements.
- **Identify one or more solutions**
  - Evaluate possible solutions and choose the most appropriate depending on the designer's experience and available resources.
- **Describe solution abstractions**
  - Use graphical, formal or other descriptive notations to describe the components of the design.
- Repeat process for each identified abstraction until the design is expressed in primitive terms.

The Design Process

- Any design may be modelled as a directed graph made up of entities with attributes which participate in relationships.
- The system should be described at several different levels of abstraction.
- Design takes place in overlapping stages. It is artificial to separate it into distinct phases but some separation is usually necessary.

Software design and its activities

Software design : Transforming the customer requirements, described in the SRS document, into a form (a set of documents) that is suitable for implementation in a programming language.

A good software design is seldom arrived by using a single step procedure but rather through several iterations through a series of steps.

Design activities can be broadly classified into two important parts:

- **Preliminary (or high-level) design**

and

- **Detailed design.**

Preliminary (or high-level) design

- ***Through high-level*** design, a problem is decomposed into a set of **modules**.
- The control relationships among the modules are identified and the interfaces among the various modules are identified.
- The outcome of high-level design is called program structure or software architecture.
- Many different types of notations have been used to represent a high-level design.
- High-level design is a crucial step in the overall design of a software.
- Once the high-level design is complete, detailed design is undertaken.

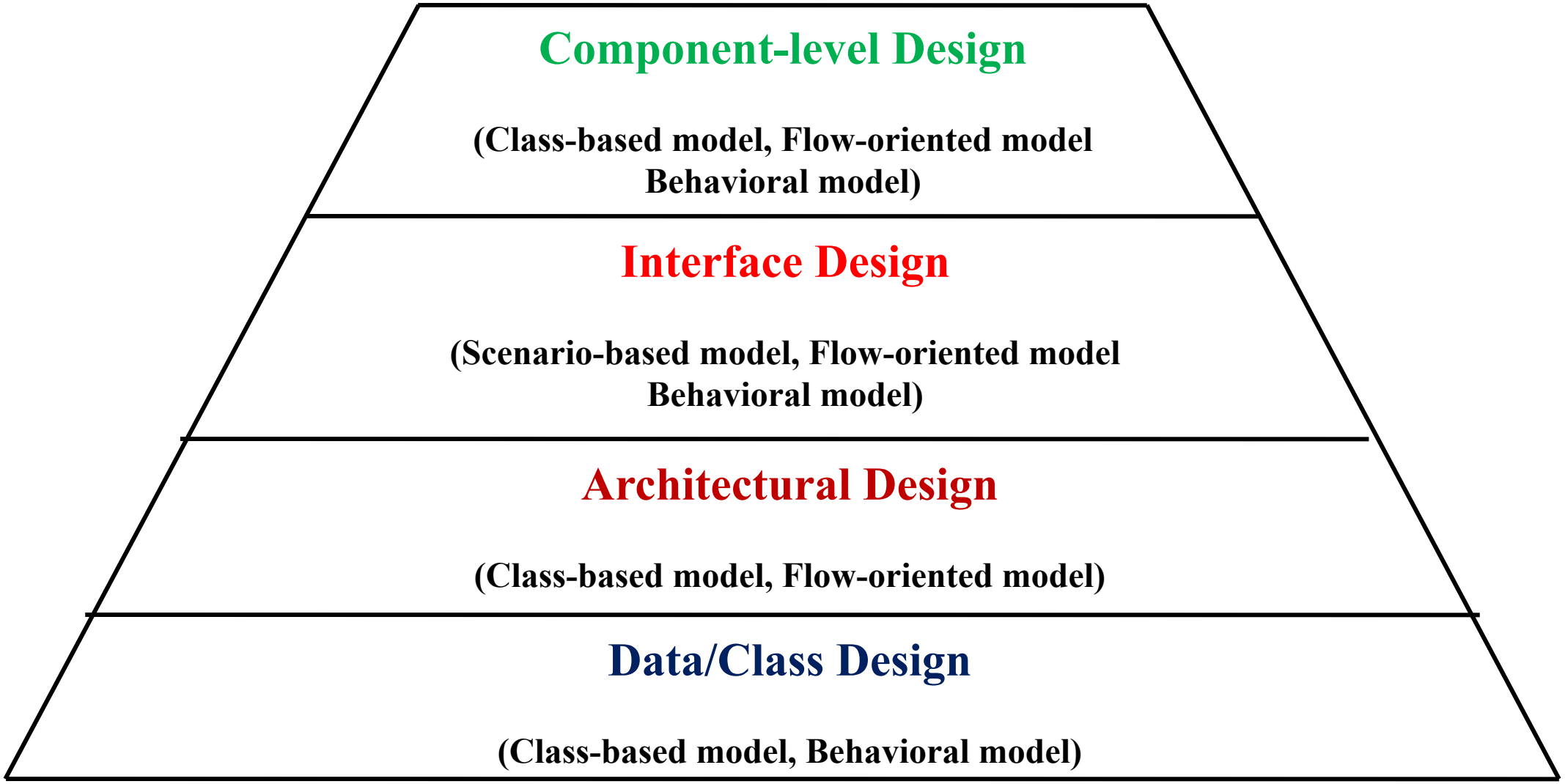
Detailed Design

During detailed design each module is examined carefully to design its data structure and the algorithms.

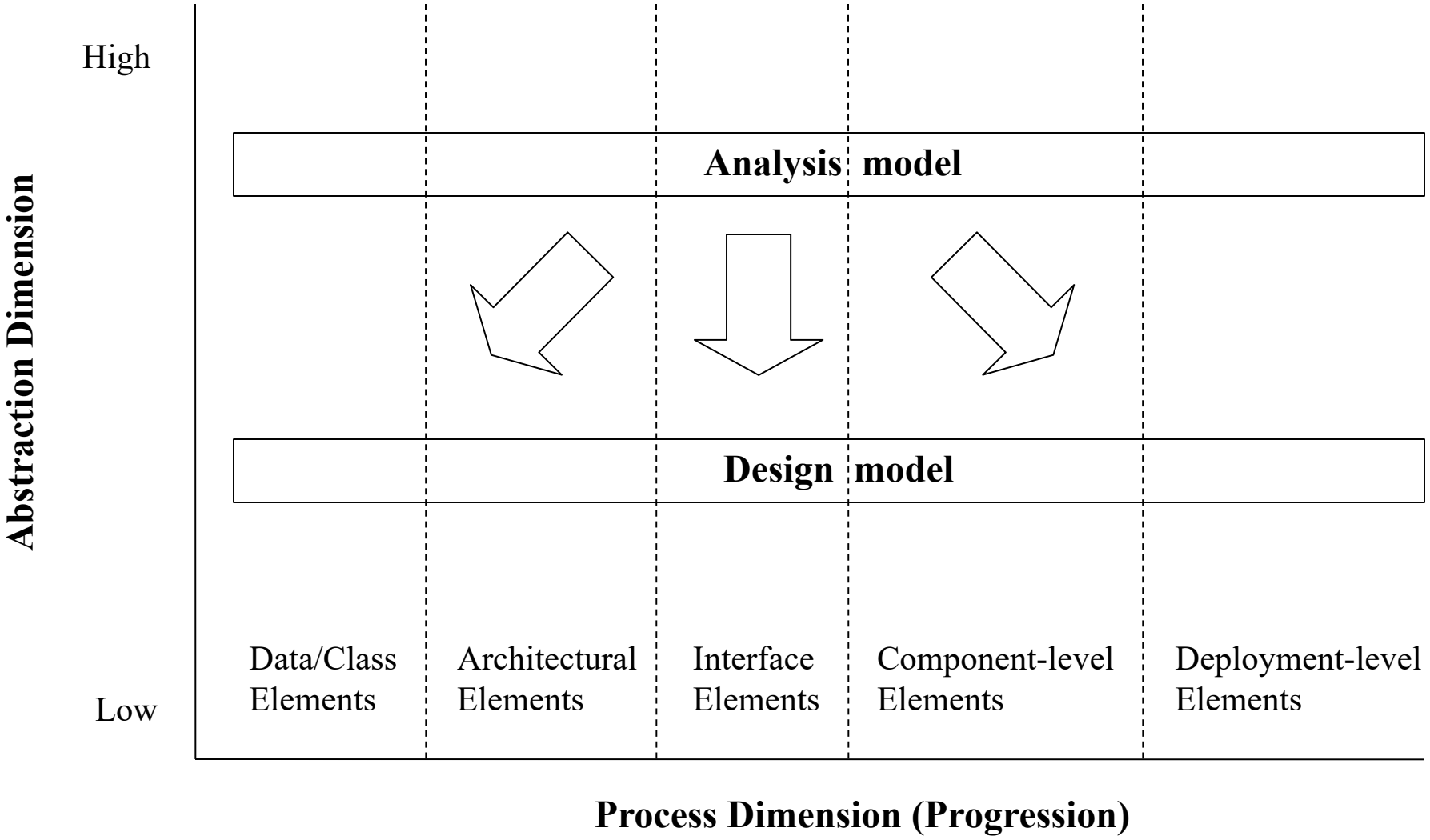
The outcome of the detailed design stage is usually known as the **module-specification (MSPEC)** document.

The data structure and the algorithms described using MSPEC, can be easily grasped by programmers for initiating the coding.

From Analysis Model to Design Model (continued)



Dimensions of the Design Model



Difference between analysis and design

The aim of **analysis** is to understand the problem with a view to eliminate any deficiencies in the requirement specification such as

incompleteness,
inconsistencies, etc.

The model which we are trying to build may be or may not be ready.

The aim of **design** is to produce a model that will provide a seamless transition to the coding phase.

Once the requirements are analyzed and found to be satisfactory, a design model is created which can be easily implemented.

Items developed during the software design phase

The following items must be designed during the design phase.

- Different modules required to implement the design solution.
- Control relationship among the identified modules.
- The relationship is also known as the call relationship or invocation relationship among modules.
- Interface among different modules. (The interface among different modules identifies the exact data items exchanged among the modules.)
- Data structures of the individual modules.
- Algorithms required to implement each individual module.

Characteristics of a good software design

- The definition of “a good software design” can vary depending on the application being designed.

Every good software design for general application must possess. The characteristics are listed below:

- **Correctness:** A good design should correctly implement all the functionalities identified in the SRS document.
- **Understandability:** A good design is easily understandable.
- **Efficiency:** It should be efficient.
- **Maintainability:** It should be easily amenable to change.

Features of a design document

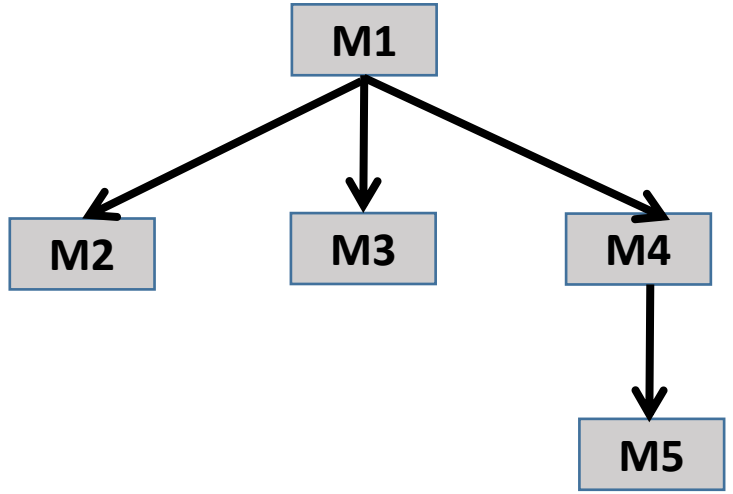
In order to facilitate understandability, the design should have the following features:

- It should use consistent and meaningful names for various design components.
- The design should be modular. The term modularity means that it should use a cleanly decomposed set of modules.
- It should neatly arrange the modules in a hierarchy, e.g. in a tree-like diagram.

Modularity & Layered Design

A designed solution is said to be highly modular,

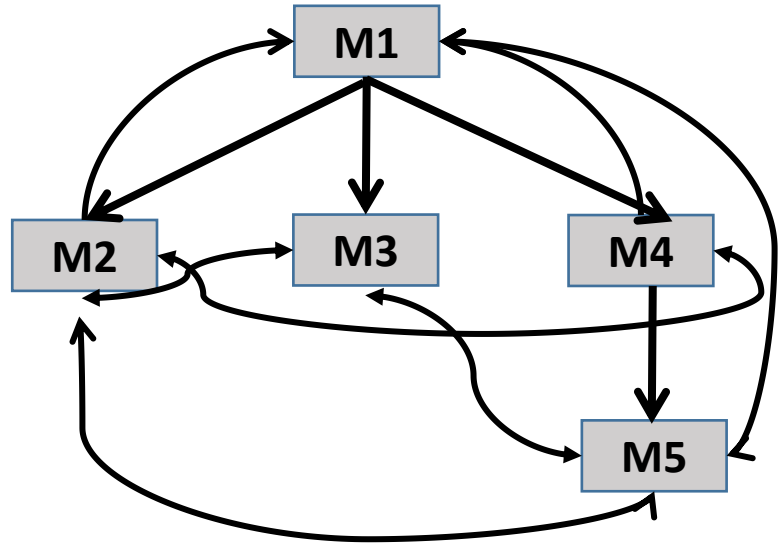
- If the different modules in the solution have high cohesion and their inter-module coupling are low



A modular and hierarchical design

A layered design can make the design solution easily understandable,

- Since to understand the working of a module, one would at best have to understand how the immediately lower layer modules work without having to worry about the functioning of the upper layer modules.



A design of poor modularity and hierarchy

Coupling and Cohesion

- When a software program is modularised, its tasks are divided into several modules based on some characteristics.
- As we know, modules are set of instructions put togeather in order to achive some tasks.
- They are though, considered as single entity but may refer to each other to work to geather.
- There are measures by which the quality of a design of modules and their interaction among them can be measured.

These measures are called coupling and cohesion.

Coupling and Cohesion

- **COUPLING** between two modules is a measure of the degree of interaction (or interdependence) between the two modules.

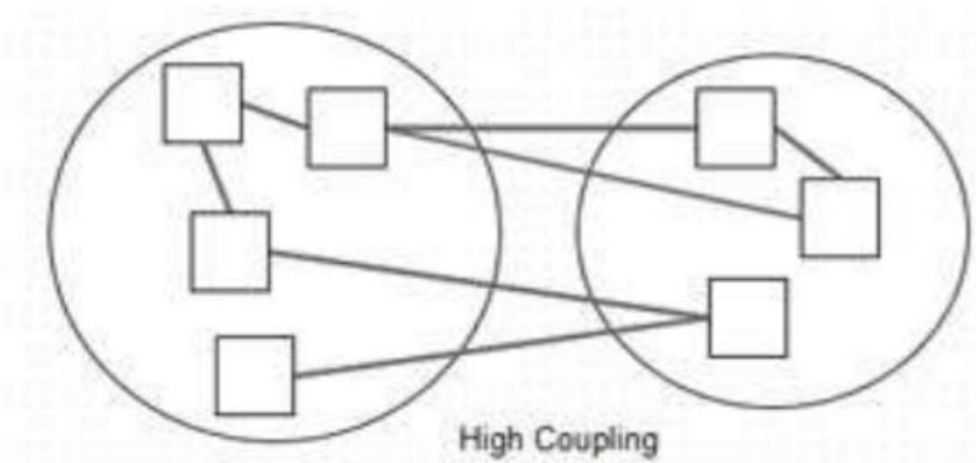
- Whereas the **COHESION** is a measure of functional strength of a module.

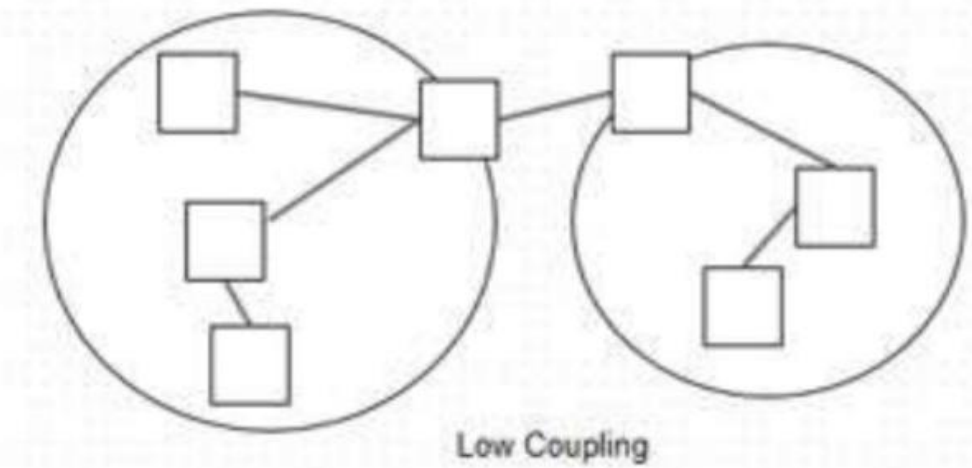
A module having high cohesion and low coupling is said to be functionally independent of other modules.

i.e, A module performs a single task and needs very little interaction with other modules.

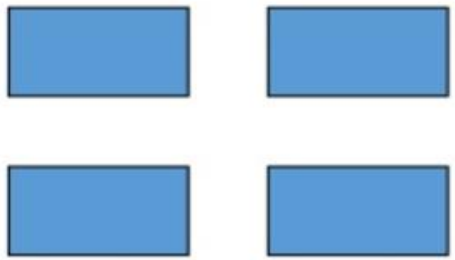
Characteristics of Good Design

- Component independence
  - High cohesion
  - Low coupling
- Exception identification and handling
- Fault prevention and fault tolerance
- Design for change

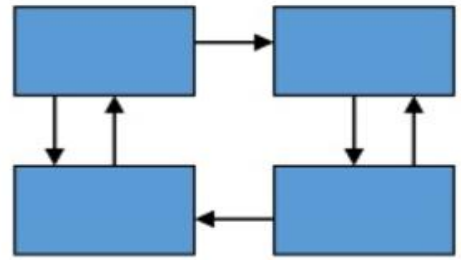




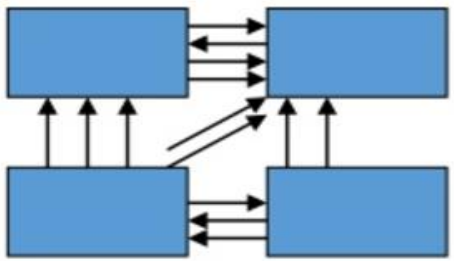
Coupling: Degree of Dependence Among Components



No dependencies



Loosely coupled-some dependencies

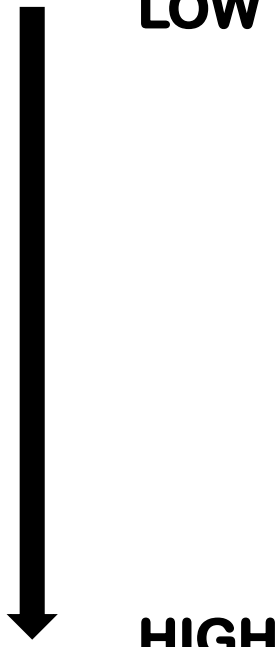


Highly coupled-many dependencies

High coupling makes modifying parts of the system difficult, e.g., modifying a component affects all the components to which the component is connected.

Classification of Coupling

Different classes of Coupling
Data
Stamp
Control
Common
Content

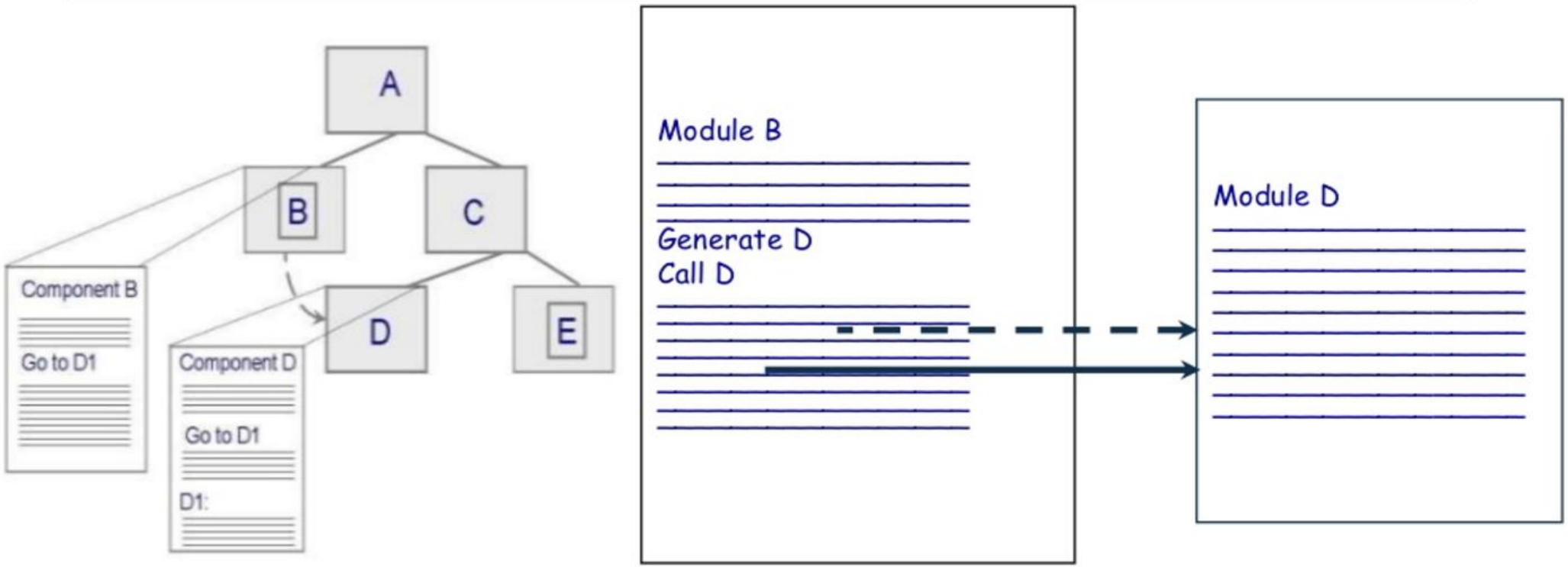


Content Coupling

- **Content coupling** - When a module can directly access or modify or refer to the content of another module, it is called content level coupling.
- One module directly references contents of the other
- **Example**
  - module **a** modifies statements of module **b**
  - module **a** refers to local data of module **b** in terms of some numerical displacement within **b**
  - module **a** branches into local label of module **b**
- **Why is this bad?**
  - almost any change to **b** requires changes to **a**

Example of Content Coupling

- Occurs when one component modifies an internal data item in another component, or when one component branches into the middle of another component



Common Coupling

- **Common coupling-** When multiple modules have read and write access to some global data, it is called common or global coupling.
- **Example**
  - two modules have access to same database, and can both read and write same record
  - Modules `a` and `b` can access and change the value of global variable
  - use of Java **`public`** statement
- **Why is this bad?**
  - resulting code is unreadable
    - modules can have side-effects
    - must read entire module to understand
  - difficult to reuse
  - module exposed to more data than necessary

Example of Common Coupling

Process control component maintains current data about state of operation. Gets data from multiple sources. Supplies data to multiple sinks. Each source process writes directly to global data store. Each sink process reads directly from global data store.

Improvement?

Data manager component is responsible for data in data store. Processes send data to and request data from data manager.

Control Coupling

- **Control coupling-** Two modules are called control-coupled if one of them decides the function of the other module or changes its flow of execution.
- one module passes an element of control to the other
- **Example**

  -If module `p` calls module `q` and `q` passes back

  – "I am unable to complete my task", then `q` is passing data

  – "I am unable to complete my task; write error message

  • ABC123", then `p` and `q` are control coupled
- **Why is control coupling bad?**
  – Modules are not independent; module `q` (the called module) must know internal structure and logic of module `p`. This affects reusability
  – Usually is associated with modules of logical cohesion

Example of Control Coupling

- **Acceptable:** Module p calls module q and q passes back flag that says it cannot complete the task, then q is passing data

- **Not Acceptable:** Module p calls module q and q passes back flag that says it cannot complete the task and, as a result, writes a specific message.

Stamp Coupling

- **Stamp coupling-** When multiple modules share common data structure and work on different part of it, it is called stamp coupling.
- data structure is passed as parameter, but called module operates on only some of individual components
- **Example**

```
calculate withholding (employee record)
```

- **Why is this bad?**
  - affects understanding
    - not clear, without reading entire module, which fields of record are accessed or changed
  - unlikely to be reusable
    - other products have to use the same higher level data structures
  - passes more data than necessary
    - e.g., uncontrolled data access can lead to computer crime

Example of Stamp Coupling

Customer Billing System

The print routine of the customer billing accepts customer data structure as an argument, parses it, and prints the name, address, and billing information.

Improvement?

The print routine takes the customer name, address, and billing information as an argument.

Data Coupling

- Def: Component passes data (not data structures) to another component.
- Every argument is simple argument or data structure in which all elements are used
- Good, if it can be achieved.
- Example: Customer billing system
  - The print routine takes the customer name, address, and billing information as arguments.

Cohesion

- **Cohesion** is a measure that defines the degree of intra-dependability within elements of a module. The greater the cohesion, the better is the program design.
- "The measure of the strength of functional relatedness of elements within a module".
- **Cohesion** refers to the dependence within and among a module's internal elements (e.g., data, functions, internal modules)

Cohesion

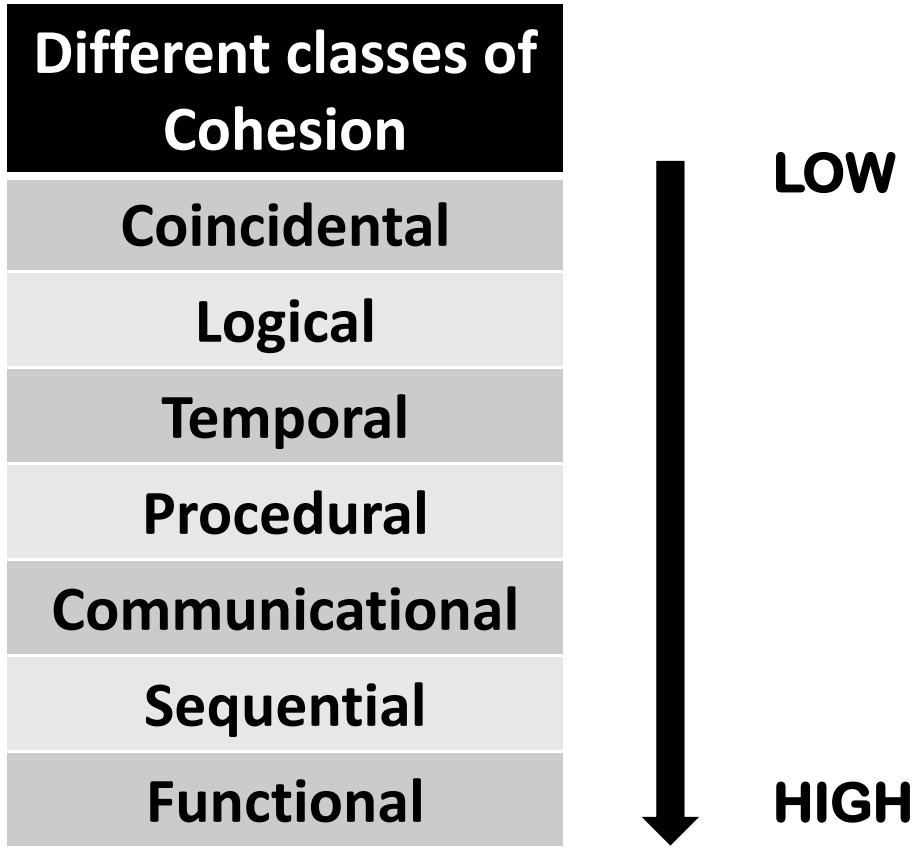
- **Cohesion** is a natural extension of the information
  - hiding concept
- A cohesive module performs a single task within a software procedure, requiring little interaction with procedures being performed in other parts of a program
- Simply state, a cohesive module should (ideally) do just one thing
- We always strive for high cohesion, although the
  - mid-range of the spectrum is often acceptable

Cohesion

- High cohesion makes it easier to:
  - Understand what a class or method does
  - Use descriptive names
  - Reuse classes or methods

Classification of cohesion

The different classes of cohesion that a module may possess are;



Coincidental Cohesion

- A module has coincidental cohesion if it performs
  - multiple, completely unrelated actions
    - print next line, reverse string of characters comprising second parameter, add 7 to fifth parameter, convert fourth parameter to floating point
- Arise from rules like “Every module will consist of
  - between 35 and 50 statements”
- Why is coincidental cohesion so bad?
  - Degrades maintainability
  - Modules are not reusable
- How to fix
  - Break into separate modules each performing one task

Logical Cohesion

- A module has logical cohesion when it performs a series of related actions, one of which is selected by the calling module
- **Example**
  - Module performing all input and output
- Why is logical cohesion so bad?
  - The interface is difficult to understand
  - Code for more than one action may be intertwined
  - Difficult to reuse

Example of Logical Cohesion

- A component reads inputs from tape, disk, and network. All the code for these functions are in the same component.
- Operations are related, but the functions are significantly different.

Improvement

- A device component has a read operation that is overridden by sub-class components. The tape sub-class reads from tape. The disk sub-class reads from disk. The network sub-class reads from the network.

Temporal Cohesion

- A module has **temporal cohesion** when it
  - performs a series of actions related in time
- **Example**
  - open old master file, new master file, transaction file, print file, initialize sales district table, read first transaction record, read first old master record
- Why is temporal cohesion so bad?
  - Actions of this module are weakly related to one another, but strongly related to actions in other modules
  - Not reusable

Example of Temporal Cohesion

- A system initialization routine: this routine contains all of the code for initializing all of the parts of the system. Lots of different activities occur, all at in it time.

Improvement

- A system initialization routine sends an initialization message to each component.
- Each component initializes itself at component instantiation time.

Procedural Cohesion

- A module has procedural cohesion if it performs a series of actions related by the procedure to be followed by the product
- Example
  - read part number and update repair record on
    - master file
- Why is procedural cohesion bad?
  - Actions are still weakly connected
  - Module is unlikely to be reusable

Communicational Cohesion

- A module has communicational cohesion if it performs a series of actions related by the procedure to be followed by the product, but in addition all the actions operate on the same data
- **Examples**
  - update record in database and write it to audit trail
  - calculate new coordinates and send them to terminal
- Why is communicational cohesion bad?
  - Still lack of reusability

Example of Communicational Cohesion

- Update record in data base and send it to the printer.

- database.Update (record).
- record.Print().

Functional Cohesion

- It is considered to be the highest degree of cohesion, and it is highly expected. Elements of module in functional cohesion are grouped because they all contribute to a single well-defined function. It can also be reused.
- Module with functional cohesion performs exactly one action
- **Examples**
  - calculate sales commission
- Why is functional cohesion good?
  - More reusable
  - Corrective maintenance easier
  - Easier to extend product

Informational Cohesion

- A module has informational cohesion if it performs a number of actions, each with its own entry point, with independent code for each action, all performed on the same data structure
- Major difference between logical and information cohesion
  - Actions in the module with logical cohesion are intertwined
  - Actions in the module with information cohesion are completely independent
- Why is informational cohesion good?
  - All the advantages of using abstract data type are gained

Cohesion vs Coupling

Cohesion	Coupling
Cohesion is the indication of the relationship within **module**.	**Coupling** is the indication of the relationships between modules.
Cohesion shows the module's relative **functional** strength.	Coupling shows the relative **independence** among the modules.
Cohesion is a degree (quality) to which a component / module focuses on the **single** thing	Coupling is a degree to which a component / module is connected to the **other** modules.