Software Engineering

Software Design

Design phase

- Goal: transform SRS document into a form easily implementable using some programming language

- Items developed during this phase
  - Different module required to implement the design solution
  - Module structure
  - Control relationships among modules
  - Interface among modules
  - Data structures and algorithms of individual modules

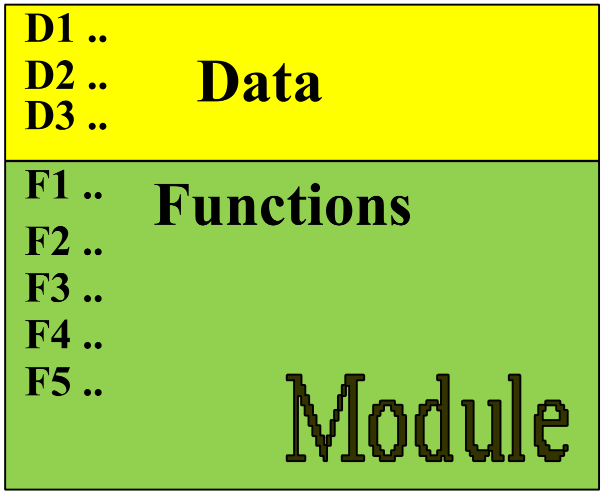
Two stages of design

- Preliminary or high-level design
  - Outcome is program structure (called software architecture)
  - Several notations available to represent program structure: **structure chart**, Jackson diagram, Warnier-Orr diagram
- Detailed design
  - Data structure & algorithm of each module
  - Outcome is called <u>module specification</u>

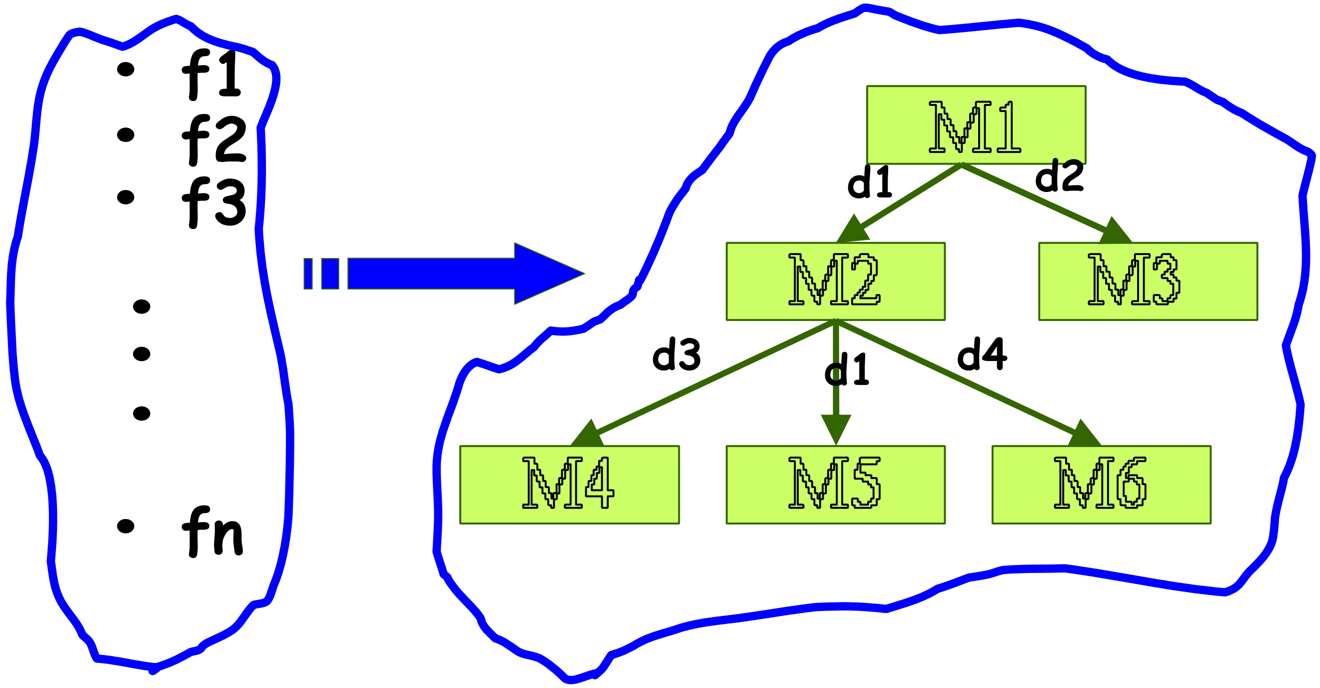
Module

A module consists of:

- Several functions
- Associated data structures.



High-level Design



High-level design maps functions into modules {fi} {Mj}

Good software design

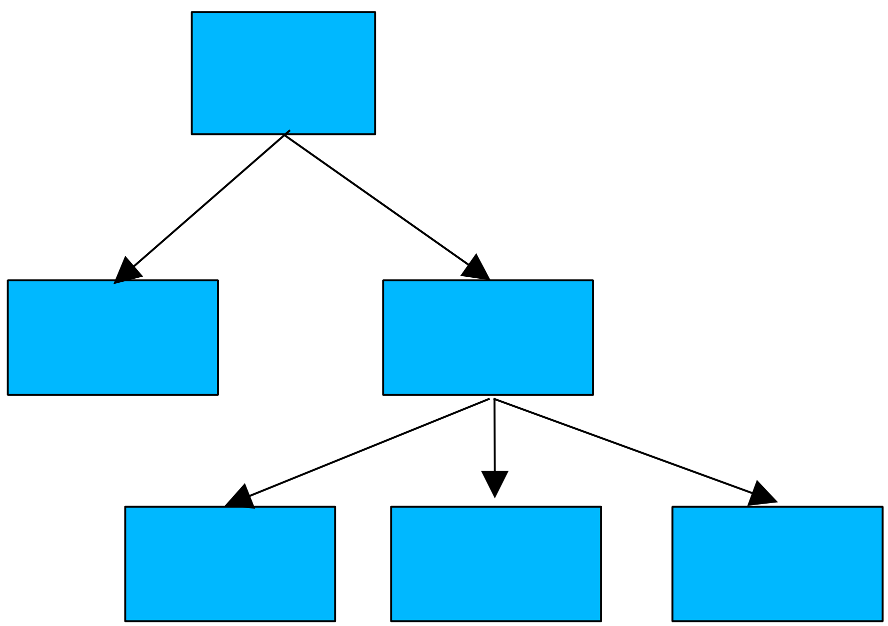
- No unique design for a system
  - Different designs possible for same system even in same design methodology is used
- Characteristics of good design
  - Implement all required functionalities correctly
  - Efficient
  - **Easily understandable**
    - Use consistent and meaningful names for various design components
    - Modularity
  - Easily amenable to change / maintainable

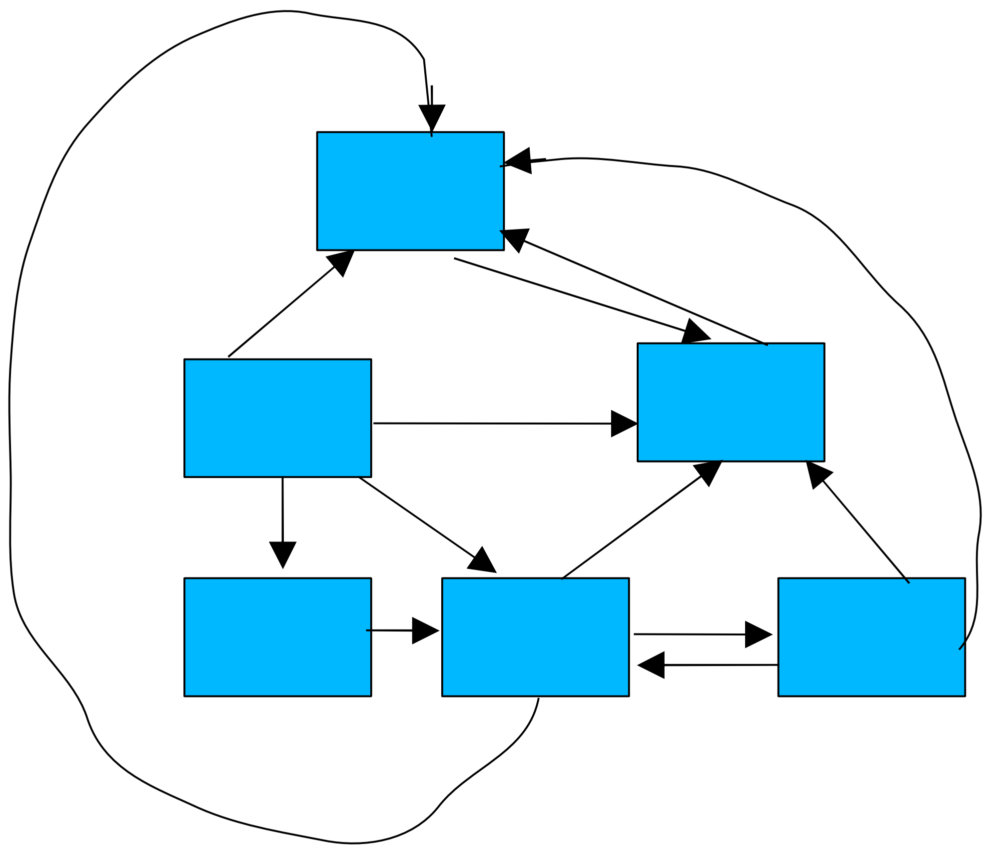
Modularity

- Design should consist of a cleanly decomposed set of modules
  - Modules should be almost independent of each other
  - Modules should be neatly arranged in a hierarchy

- In technical terms, modules should display
  - High cohesion
  - Low coupling
  - Low fan-out
  - Abstraction

Good and bad decomposition into modules

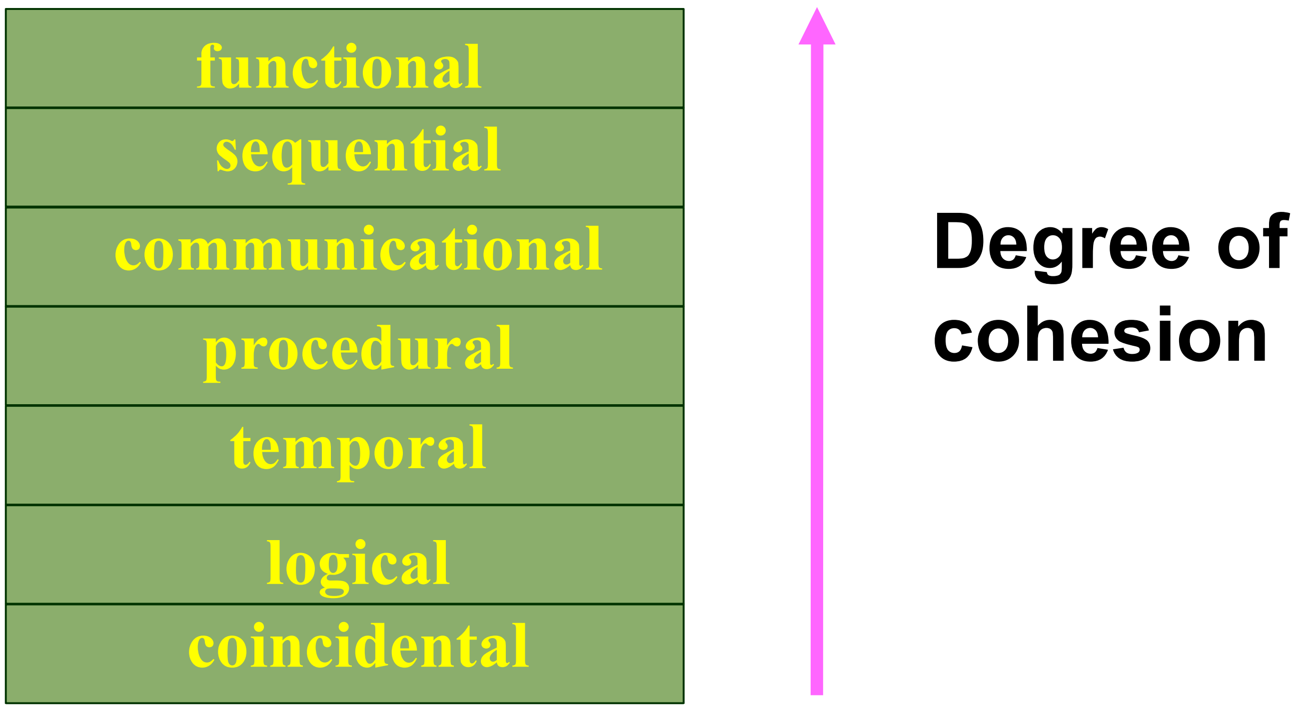




Cohesion & Coupling

- Cohesion
  - Measure of functional strength of a module
  - A cohesive module performs a single / few related tasks
- Coupling between two modules
  - measure of the degree of interdependence or interaction between the two modules
  - A module having low coupling is functionally independent of other modules
- No ways to quantitatively measure cohesion & coupling, only subjective classification

Classification of cohesiveness



Coincidental Cohesion

The module performs a set of tasks:

- Which relate to each other very loosely, if at all.
  - The module contains a random collection of functions.
  - Functions have been put in the module out of pure coincidence without any thought or design.

Logical Cohesion

All elements of the module perform similar operations:

- e.g. error handling, data input, data output, etc.

An example of logical cohesion:

- A set of print functions to generate an output report arranged into a single module.

Temporal Cohesion

The module contains tasks that are related by the fact:

- All the tasks must be executed in the same time span.

Example:

- The set of functions responsible for
  - initialization,
  - start-up, shut-down of some process, etc.

Procedural Cohesion

The set of functions of the module:

- All part of a procedure (algorithm)
- Certain sequence of steps have to be carried out in a certain order for achieving an objective,
  - e.g. the algorithm for encoding/decoding a message.

Communicational Cohesion

All functions of the module:

- Reference or update the same data structure,

Example:

- The set of functions defined on an array or a stack.

Sequential Cohesion

Elements of a module form different parts of a sequence,

- Output from one element of the sequence is input to the next.
- Example:

CreateOrder() → checkItemAvailability() → PlaceOrderOnVendor()

Functional Cohesion

Different elements of a module cooperate:

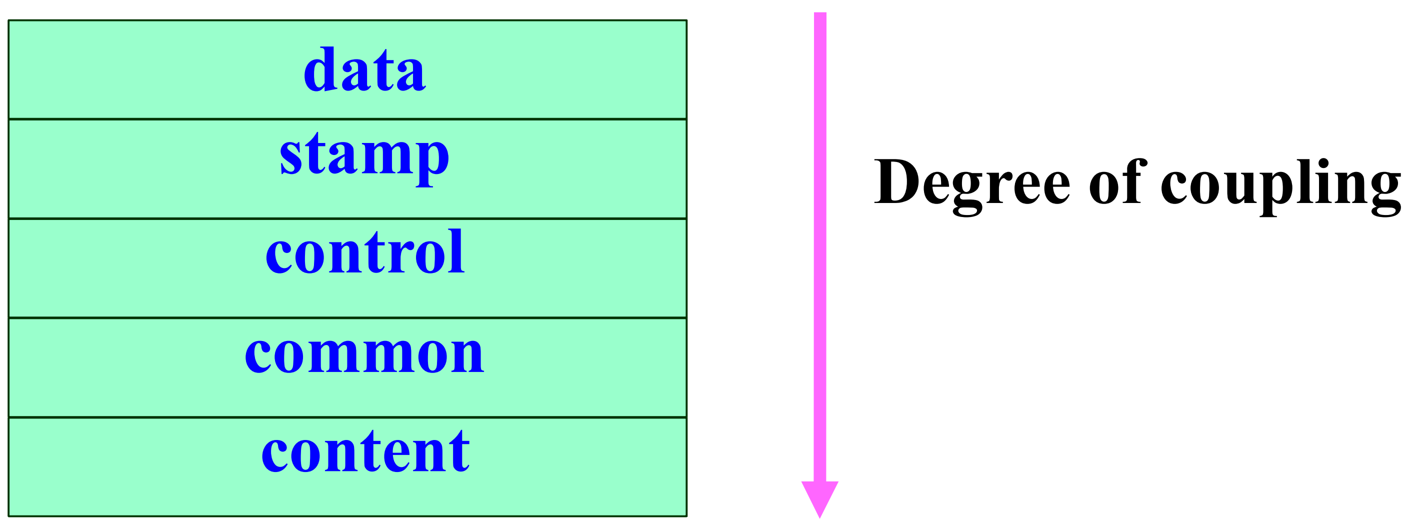
- To achieve a single function,
- e.g. managing an employee's pay-roll.
  - ComputeOvertime(), ComputeWorkhours(), compute deduction()

When a module displays functional cohesion,

- We can describe the function using a single sentence.

Coupling

- How closely two modules interact or how interdependent they are
  - Depends on the interface between the modules
  - No objective metrics, but a subjective classification



Data coupling

Two modules are data coupled,

- If they communicate via a parameter:
  - an elementary data item,
  - e.g an integer, a float, a character, etc.

- The data item should be problem related:
  - Not used for control purpose.

Stamp Coupling

Two modules are stamp coupled,

- If they communicate via a composite data item
  - such as a record in PASCAL
  - or a structure in C.

Control Coupling

Data from one module is used to direct:

- Order of instruction execution in another.

- Example of control coupling:
  - A control information (flag) is passed from one module to another. A flag set in one module and tested in another module.

Common Coupling

Two modules are common coupled,

- If they share some global data.

Content Coupling

Content coupling exists between two modules:

- If they share code,
- e.g, branching from one module into another module.

- The degree of coupling increases
  - from data coupling to content coupling.

Functional Independence

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

- It mean the functional independent module performs
  - A single task or function.
  - It has minimal interaction with other modules.

Advantages of functional independence

- Better understandability of design
  - Different modules can be understood in isolation

- Reduces error propagation

- Reuse of modules possible

Hierarchy of modules

- Control hierarchy
  - Organization and invocation relationships among modules
  - Also called program structure / software architecture

- Most common representation: a tree-like diagram called *structure chart*

- Properties of a neat hierarchy of modules
  - Low fan-out
  - Abstraction

Characteristics of module structure

- Layering: modules arranged in layers
  - Module M calls module N → M is in layer above N
  - M is super-ordinate to N, N is subordinate to M

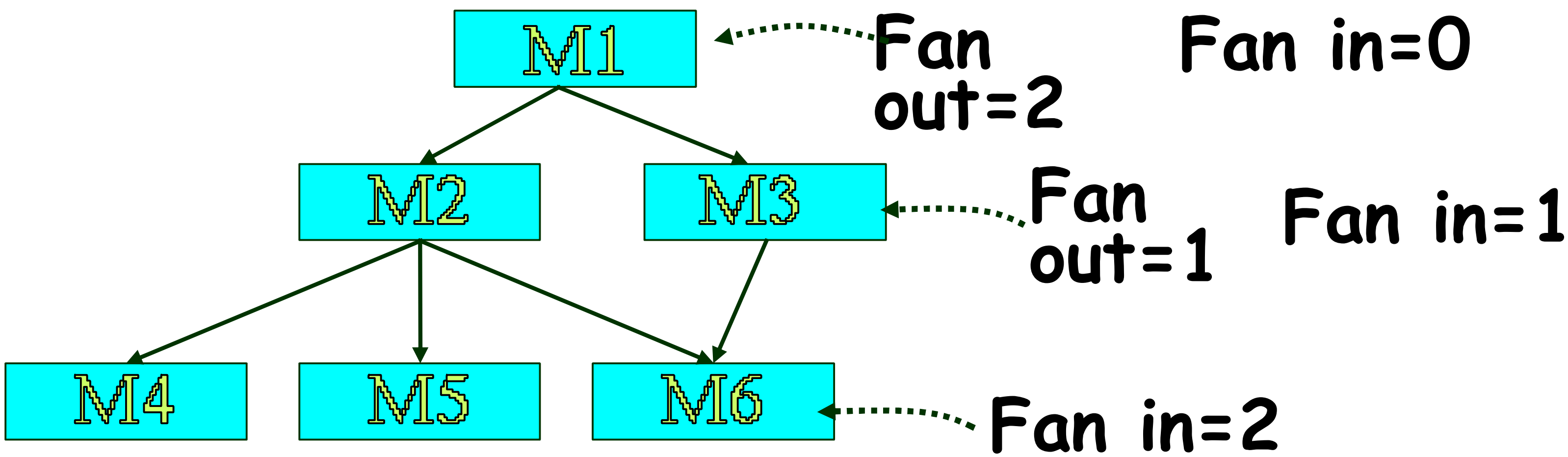
- Control abstraction: layered design
  - A module should invoke functions of modules only in the layer immediately below itself
  - Modules at lower layers should not invoke the modules above it

Characteristics of module structure

- Fan-out
  - A measure of the number of modules directly controlled by given module
  - In general, modules with high fan-out likely to lack cohesion, hence discouraged

- Fan-in
  - Indicates how many modules directly invoke a given module.
  - High fan-in represents code reuse and is in general encouraged

Module Structure



Design approaches

- High level design: mapping functional requirements (in SRS) into modules, such that
  - Each module has high cohesion
  - Coupling among modules is as low as possible
  - Modules are organized in a neat hierarchy

- Two fundamentally different software design approaches
  - Function-oriented design
  - Object-oriented design

Function-oriented design

- A system looked upon as something that performs a set of functions
  - Each function successively refined into more detailed functions
  - Functions are mapped to a module structure

- System state is centralized, accessible to different functions

Object-oriented design

- System is viewed as a collection of classes & their objects (e.g. real-world entities)
  - Objects have their own internal data
  - Objects communicate by message passing
- System state is decentralized among the objects

The two design approaches

- Gady Booch:
  - “Identify **verbs** if you are after procedural design and **nouns** if you are after object-oriented design.”

- The two approaches are complementary rather than competing
  - Both techniques applicable at different stages in the design process
  - Even in an object-oriented design, there is a hierarchy of function inside each class