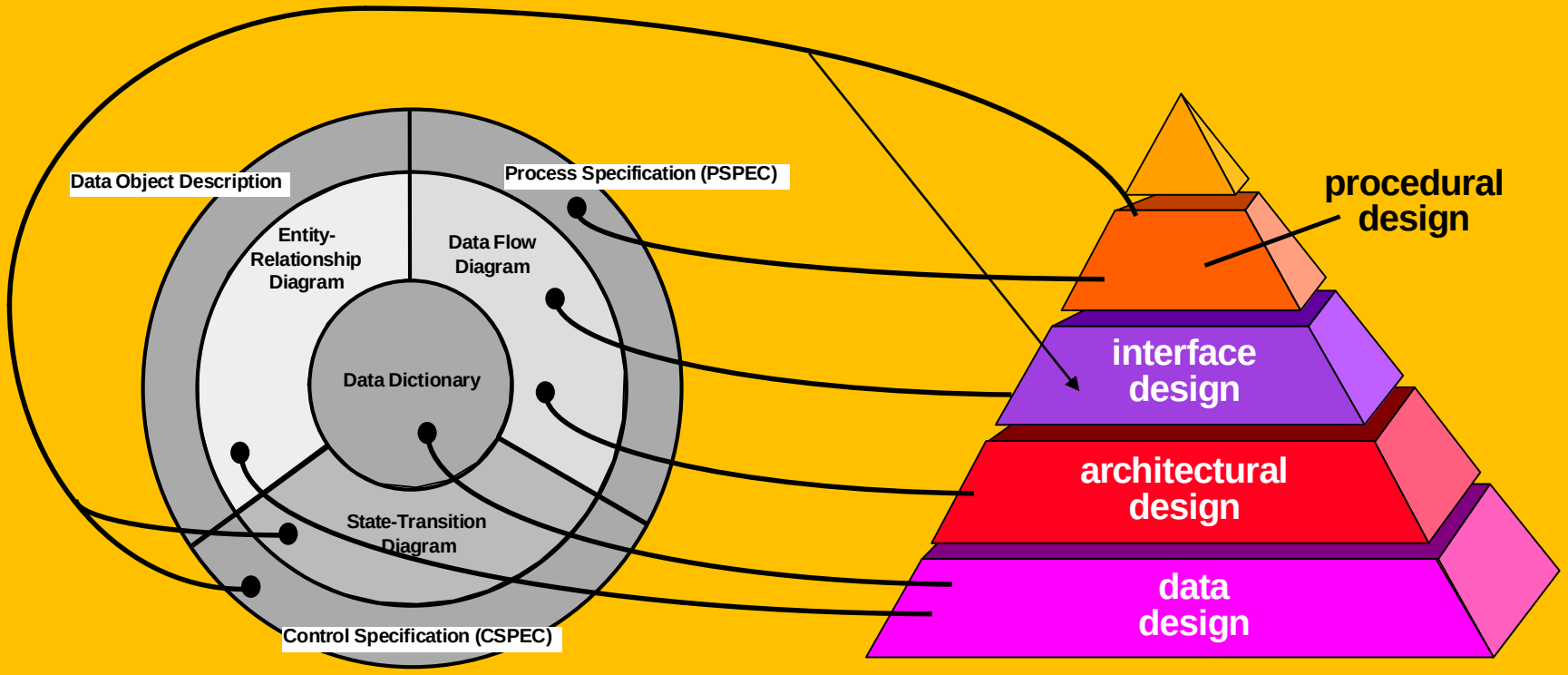
Software Design

Analysis to Design



THE ANALYSIS MODEL

THE DESIGN MODEL

Design Principles

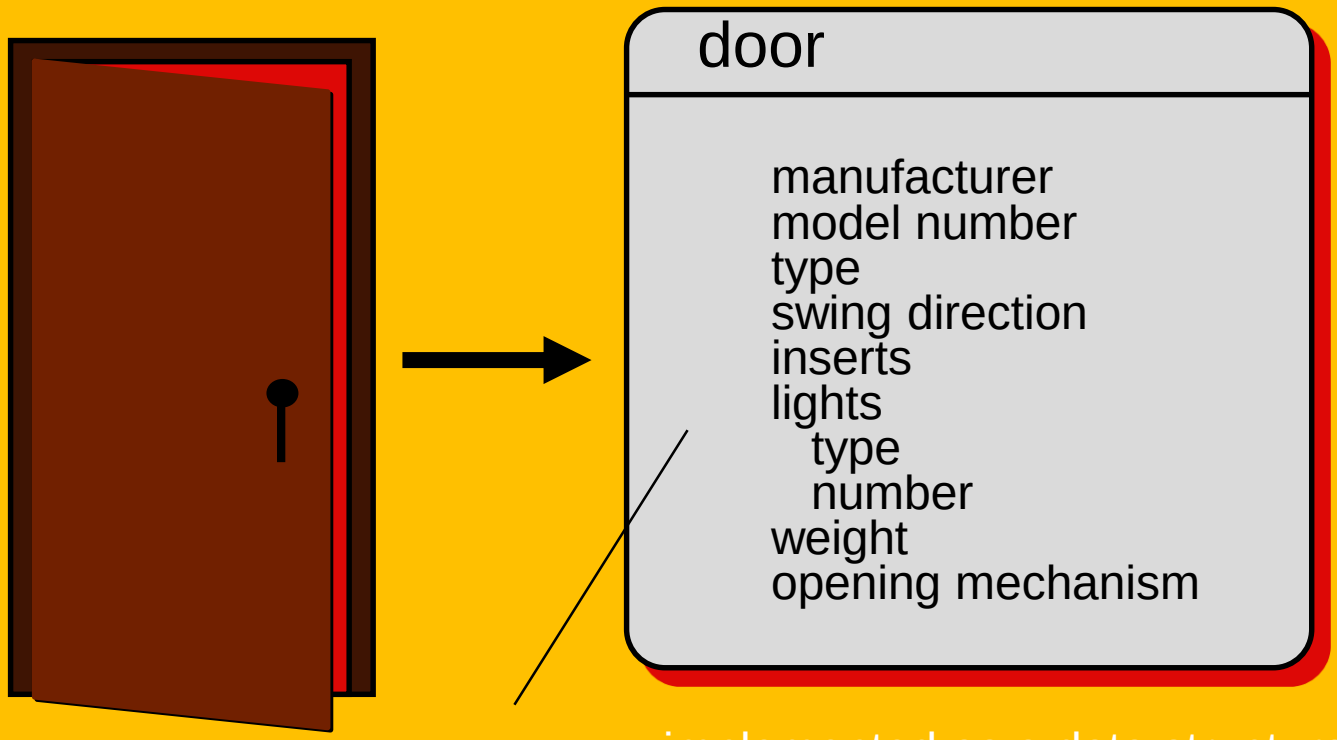
- The design process should not suffer from ‘tunnel vision.’
- The design should be traceable to the analysis model.
- The design should not reinvent the wheel.
- The design should “minimize the intellectual distance” [DAV95] between the software and the problem as it exists in the real world.
- The design should exhibit uniformity and integration.
- The design should be structured to accommodate change.
- The design should be structured to degrade gently, even when aberrant data, events, or operating conditions are encountered.
- Design is not coding, coding is not design.
- The design should be assessed for quality as it is being created, not after the fact.
- The design should be reviewed to minimize conceptual (semantic) errors.

From Davis [DAV95]

Fundamental Concepts

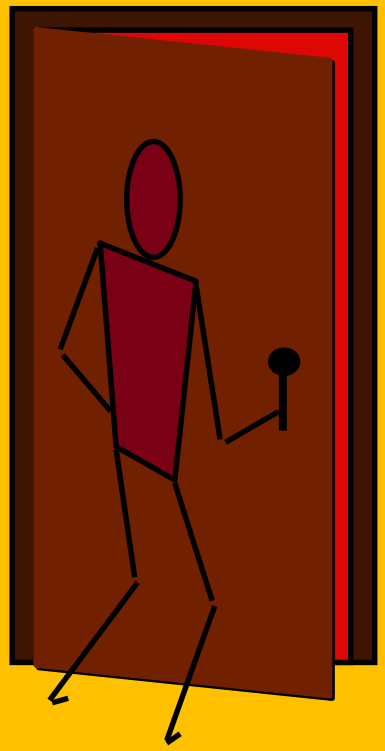
- Abstraction—**data, procedure, control**
- Refinement—**elaboration of detail for all abstractions**
- Modularity—**compartmentalization of data and function**
- Architecture—**overall structure of the software**
  - **Structural properties**
  - **Extra-structural properties**
  - **Styles and patterns**
- Procedure—**the algorithms that achieve function**
- Hiding—**controlled interfaces**

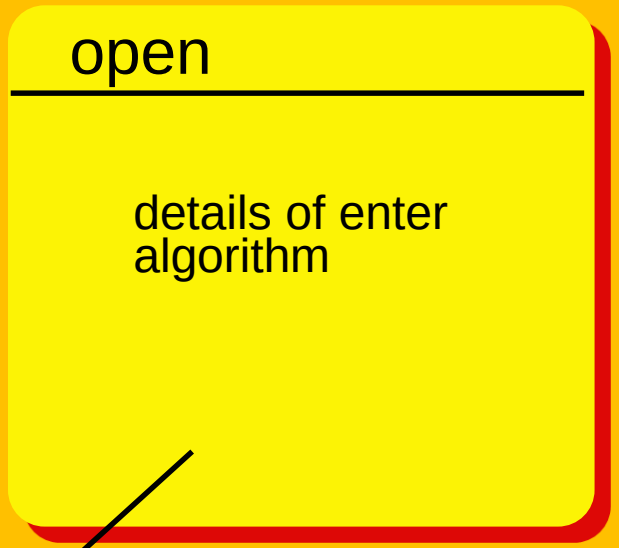
Data Abstraction



implemented as a data structure

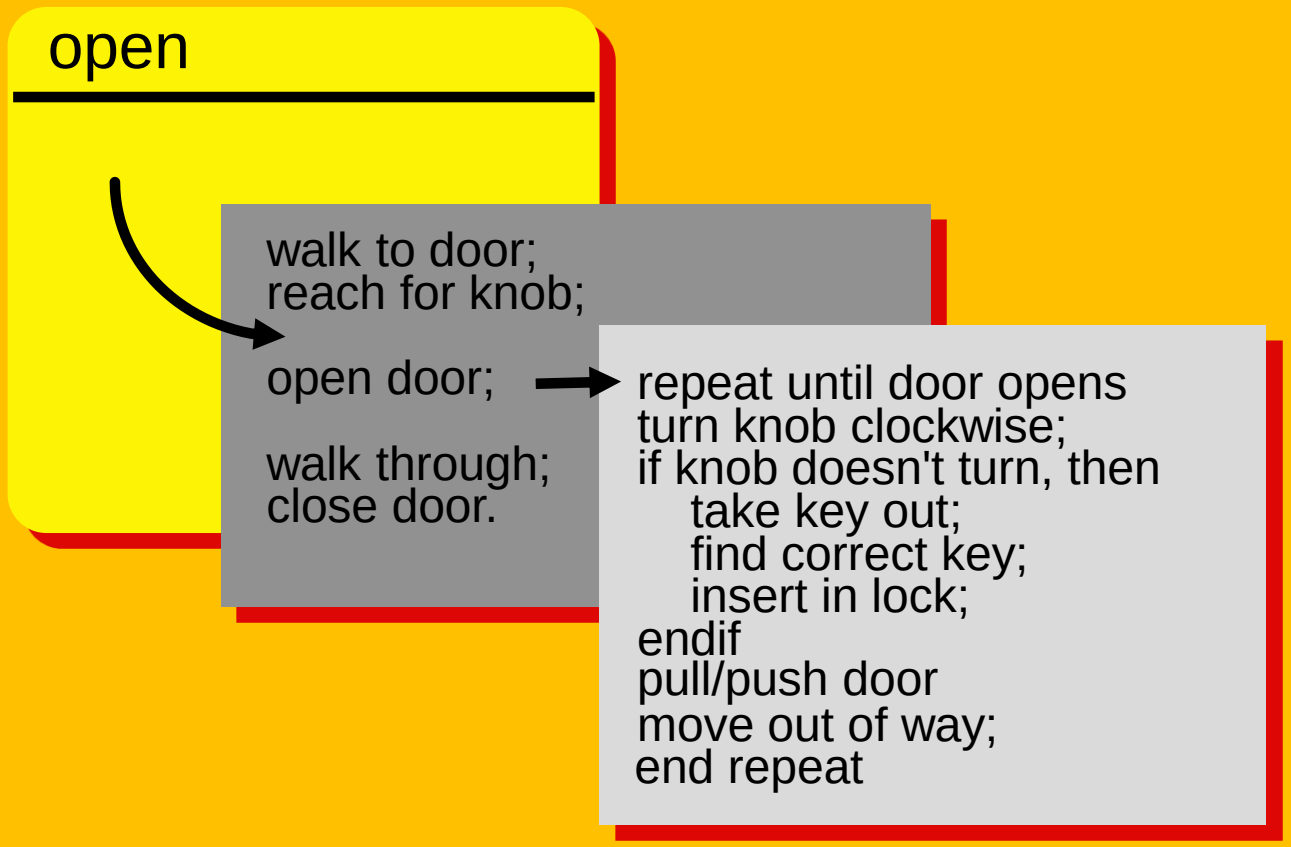
Procedural Abstraction





implemented with a "knowledge" of the object that is associated with enter

Stepwise Refinement

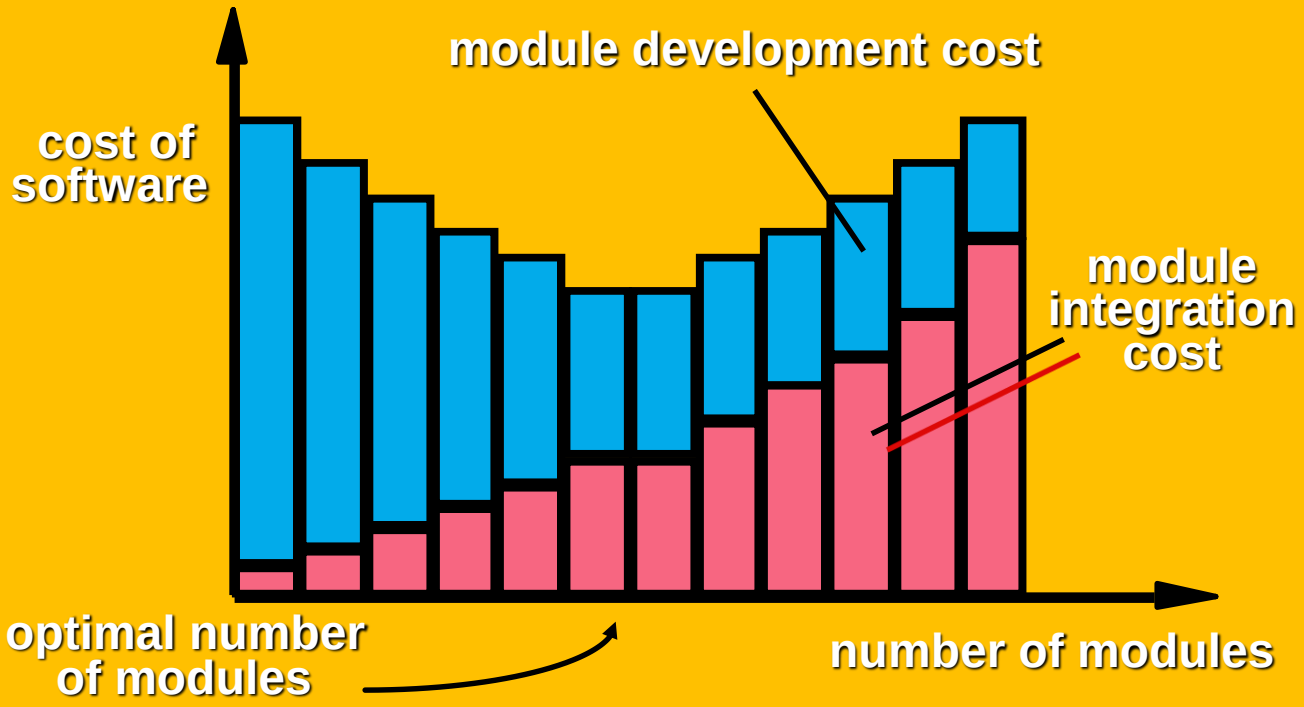


Modular Design

easier to build, easier to change, easier to fix ...

Modularity: Trade-offs

What is the "right" number of modules for a specific software design?



Architecture

"The overall structure of the software and the ways in which that structure provides conceptual integrity for a system." [SHA95a]

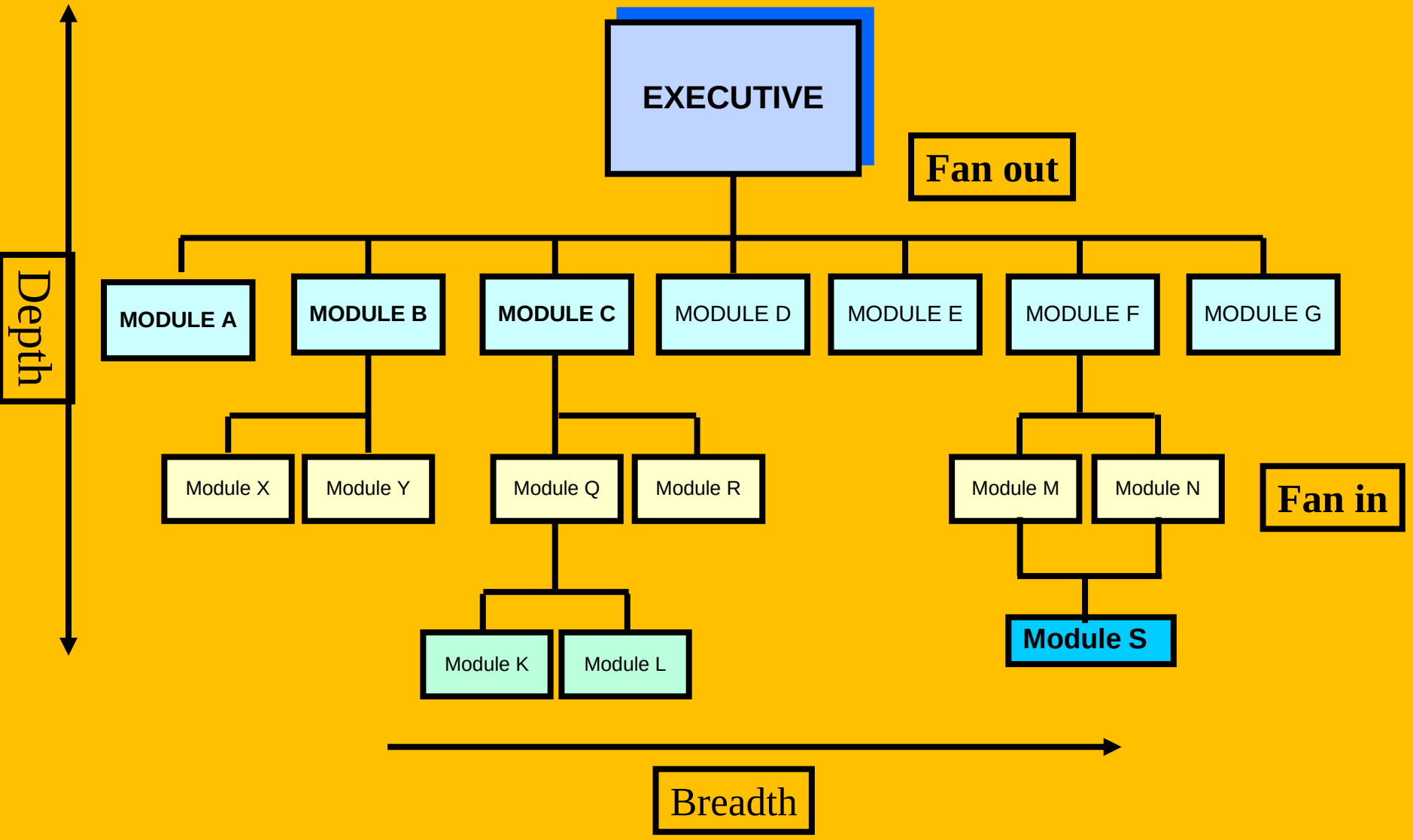
Structural properties. This aspect of the architectural design representation defines the components of a system (e.g., modules, objects, filters) and the manner in which those components are packaged and interact with one another. For example, objects are packaged to encapsulate both data and the processing that manipulates the data and interact via the invocation of methods .

Extra-functional properties. The architectural design description should address how the design architecture achieves requirements for performance, capacity, reliability, security, adaptability, and other system characteristics.

Families of related systems. The architectural design should draw upon repeatable patterns that are commonly encountered in the design of families of similar systems. In essence, the design should have the ability to reuse architectural building blocks.

Control Hierarchy

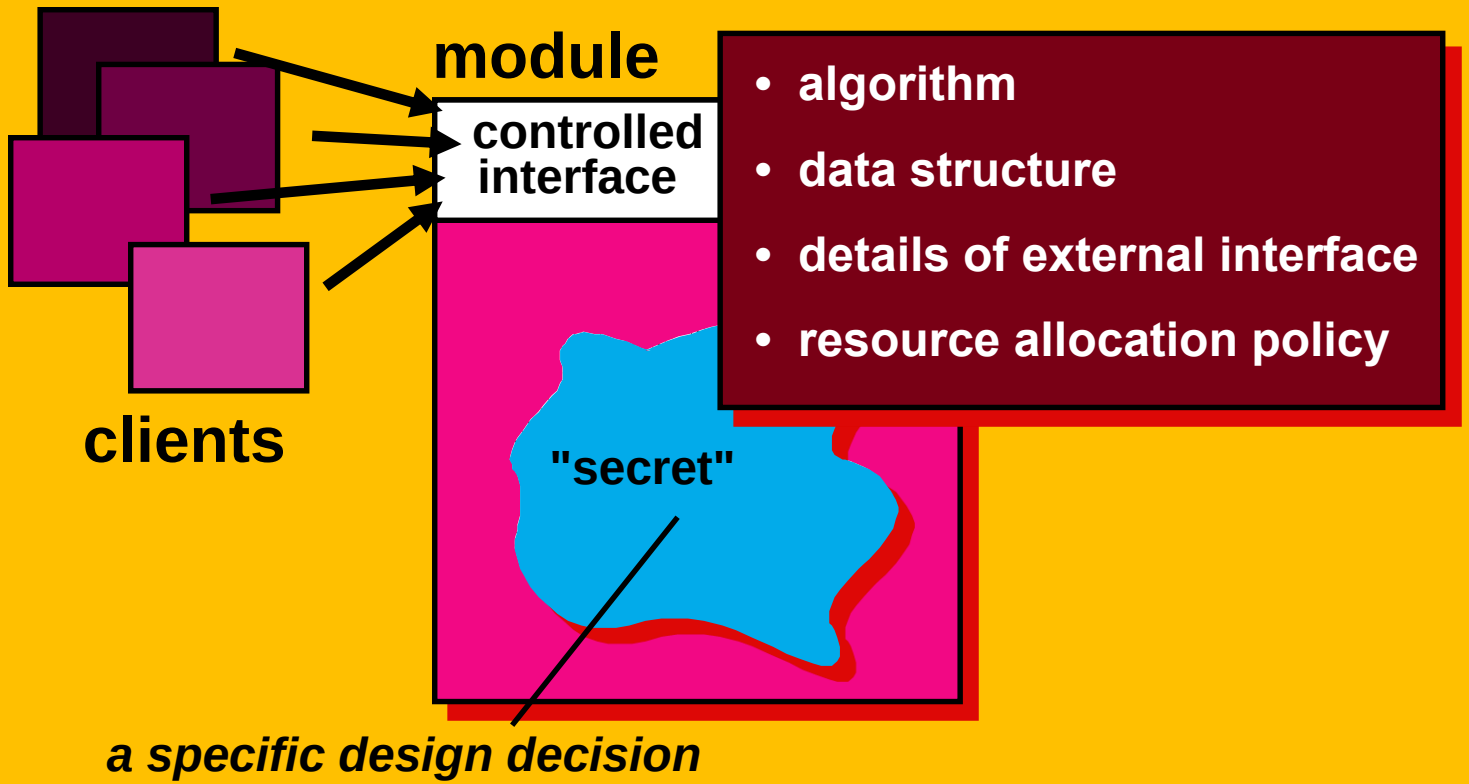
- Program Structure, represents the organization of program components(modules) and implies a hierarchy of control.
- NO concern to procedural aspects of software


EXECUTIVE
Fan out
Depth
MODULE A
MODULE B
MODULE C
MODULE D
MODULE E
MODULE F
MODULE G
Module X
Module Y
Module Q
Module R
Module M
Module N
Fan in
Module K
Module L
Module S
Breadth

Major Concepts

- **Depth**
- **Width**
- **Fan-in**
- **Fan-out**
- **Super ordinate**
- **Sub ordinate**
- **Visibility**
- **Connectivity**

Information Hiding



Why Information Hiding?

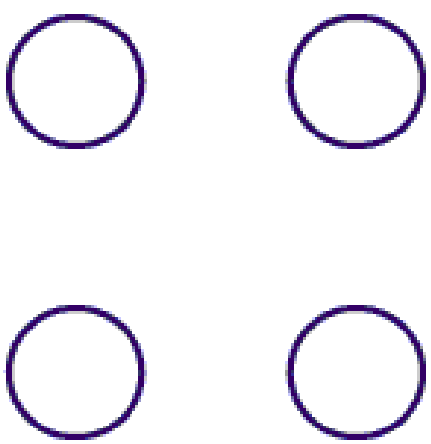
- reduces the likelihood of “side effects”
- limits the global impact of local design decisions
- emphasizes communication through controlled interfaces
- discourages the use of global data
- results in higher quality software

Functional Independence

COHESION - the degree to which a module performs one and only one function.

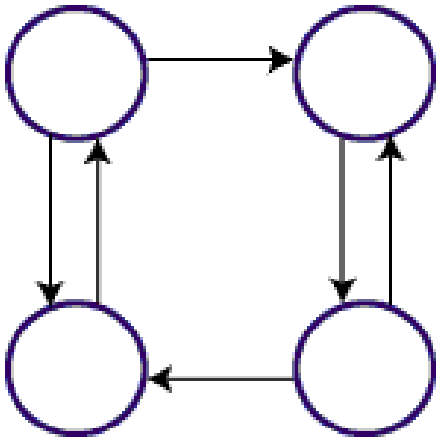
COUPLING - the degree to which a module is "connected" to other modules in the system.

Module Coupling



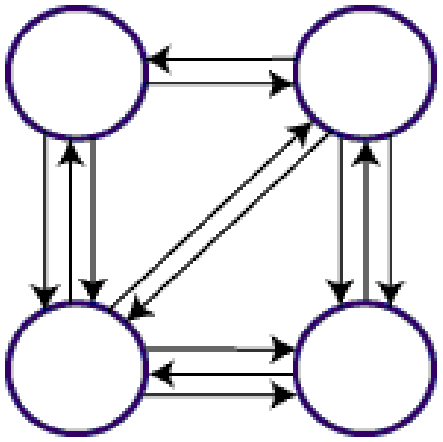
Uncoupled: no dependencies

(a)



Loosely Coupled: Some dependencies

(b)



Highly Coupled: Many dependencies

(c)

Cohesion

- **A natural extension to information hiding**
- **A cohesive module should (ideally) do just one thing**
- **High cohesion *should* be achieved and low cohesion should be avoided**

Low Cohesion

- **Coincidently Cohesive**

  A module having set of tasks that relate to each other loosely

- **Logical Cohesive**

  A module performs tasks that are logically cohesive

- **Temporal Cohesive**

  A module having set of tasks that must be executed with the same span of time.

Moderate Cohesion

- **Procedural Cohesion**

  Processing elements of a module are related and must be executed in a specific order.

- **Communicational Cohesion**

  All processing elements concentrate on one area of a data structure

High Cohesion

It is characterized by a module that performs one distinct procedural task

Nutshell:

Most Importantly, it is unnecessary to determine the precise level of cohesion, Rather it is important to strive for high cohesion and recognize low cohesion so that software design can be modified to achieve greater functional independence.

Coupling

- A measure of interconnection among modules in a software structure

- Coupling depends on:
  - Interface Complexity between modules
  - Entry Point
  - Data that is passed across the interface

- We strive for low coupling ideally

Loose Coupling

- Data Coupling
  - No direct relationship between modules
  - Simple data is passed
- Stamp Coupling
  - A portion of data structure is passed via a module interface

Moderate Coupling

- Characterized by passage of controls between modules
- A control flag is passed between modules

High Coupling

- Occur when modules are tied to an environment external to software
- Content Coupling
  - Occurs when one module makes use of data or control information maintained within the boundary of another module