Design Concepts

Book referred: Roger Pressman 7th Edition, Chapter 8

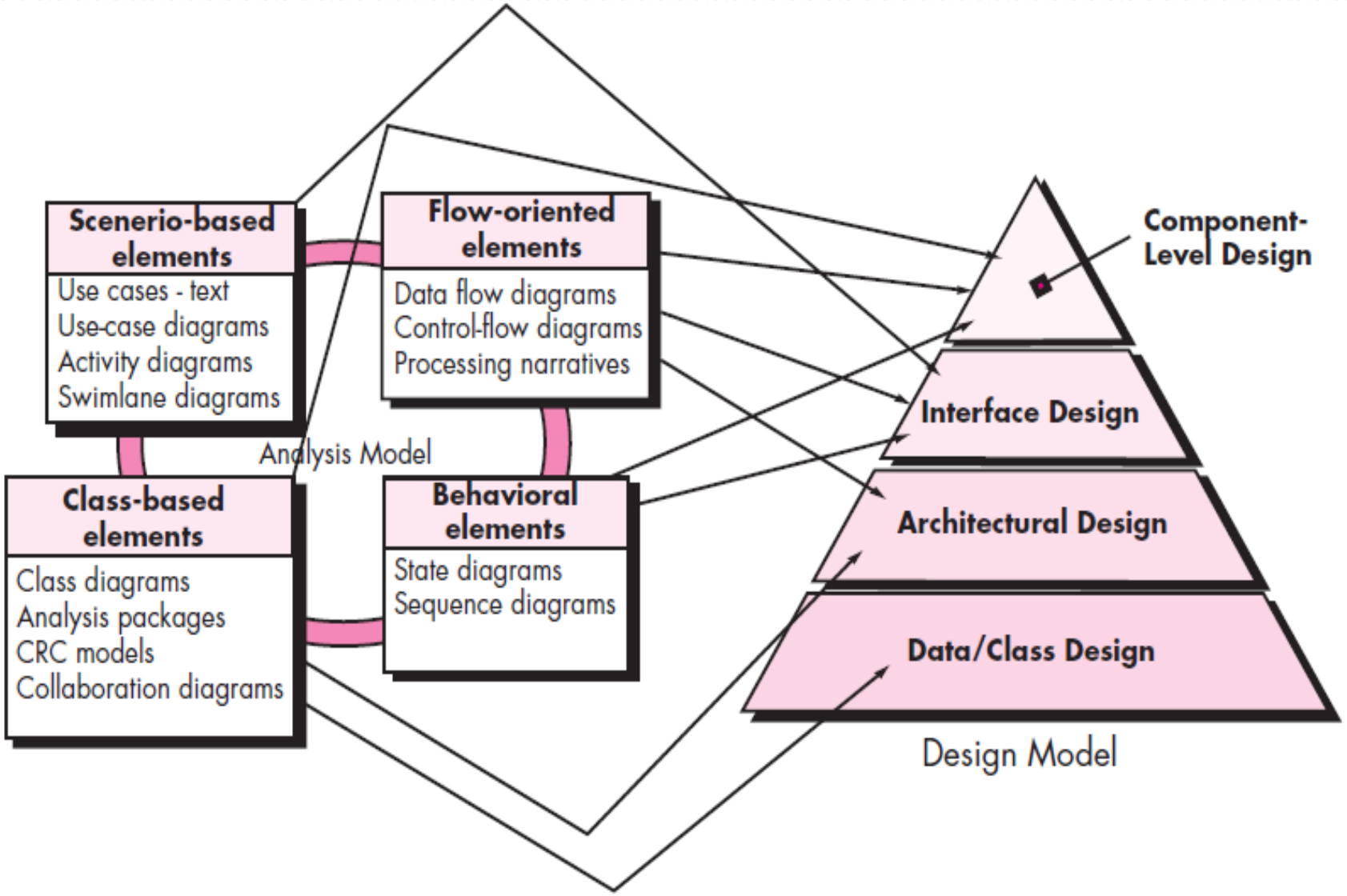
Introduction

- Design creates a representation or model of the software, but unlike the requirements model (that focuses on describing required data, function, and behavior), the design model provides detail about software architecture, data structures, interfaces, and components that are necessary to implement the system.
- Design allows you to model the system or product that is to be built.

Introduction

- Design depicts the software in a number of different ways.
- First, the architecture of the system or product must be represented.
- Then, the interfaces that connect the software to end users, to other systems and devices, and to its own constituent components are modeled.
- Finally, the software components that are used to construct the system are designed.
- Each of these views represents a different design action, but all must conform to a set of basic design concepts that guide software design work.

Analysis Model → Design Model



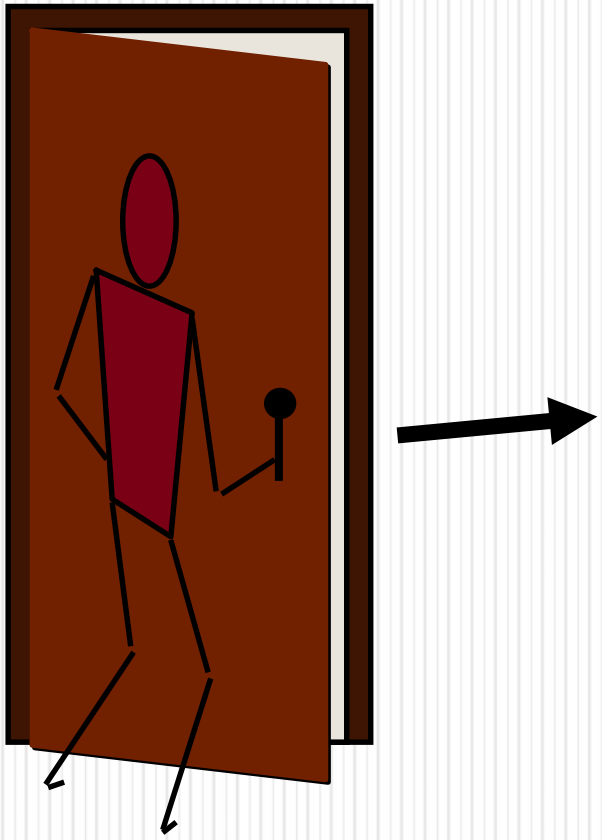
Design and Quality

- The design must implement all of the explicit requirements contained in the analysis model, and it must accommodate all of the implicit requirements desired by the customer.
- The design must be a readable, understandable guide for those who generate code and for those who test and subsequently support the software.
- The design should provide a complete picture of the software, addressing the data, functional, and behavioral domains from an implementation perspective.

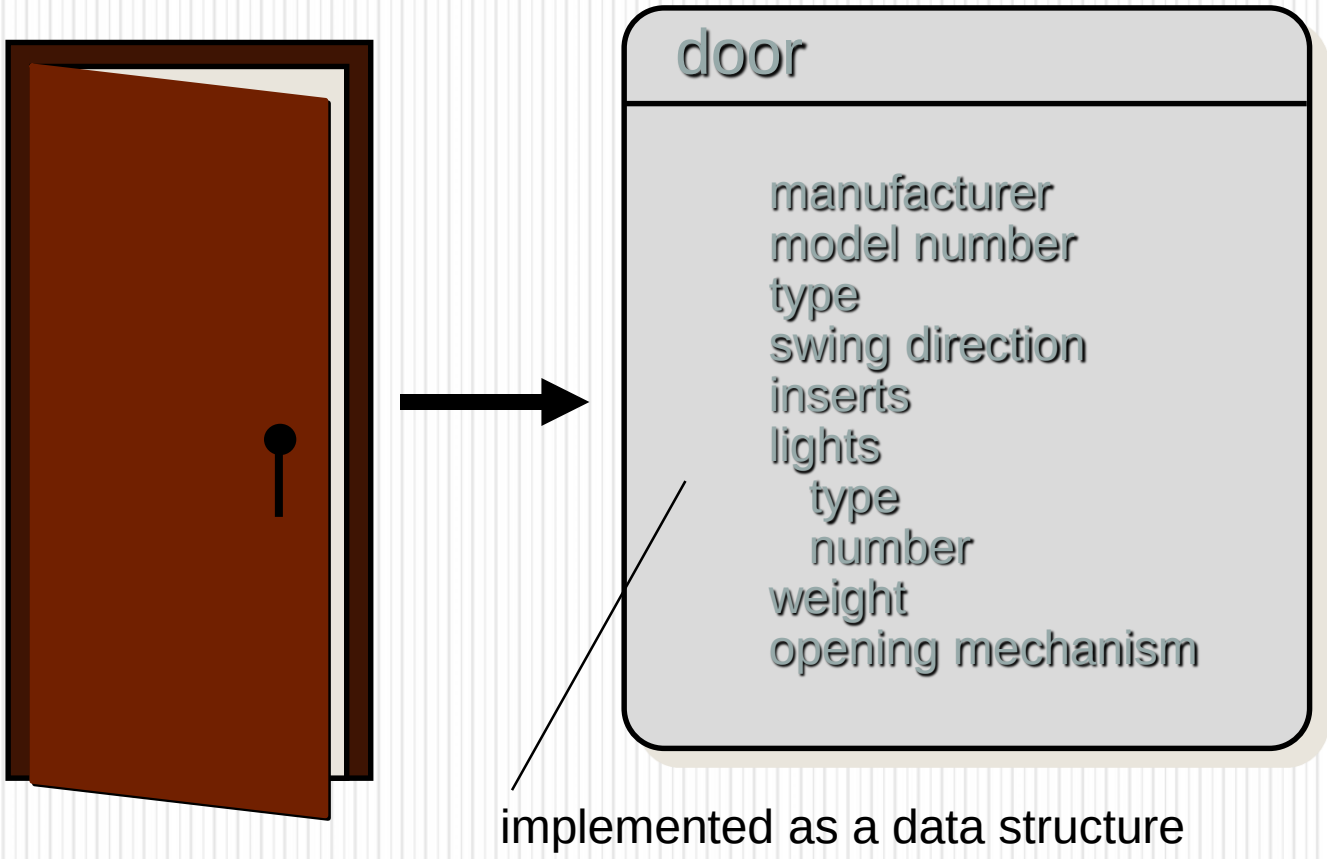
Fundamental Design Concepts

- Abstraction : data, procedure
- Architecture : the overall structure of the software
- Patterns : ”conveys the essence” of a proven design solution
- Separation of concerns : any complex problem can be more easily handled if it is subdivided into pieces
- Modularity : compartmentalization of data and function
- Hiding : controlled interfaces
- Functional independence : single-minded function and low coupling
- Refinement : elaboration of detail for all abstractions
- Aspects : a mechanism for understanding how global requirements affect design
- Refactoring : a reorganization technique that simplifies the design
- Design Classes : provide design detail that will enable analysis classes to be implemented

Procedural Abstraction



Data Abstraction



Architecture

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

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.

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.

Separation of Concerns

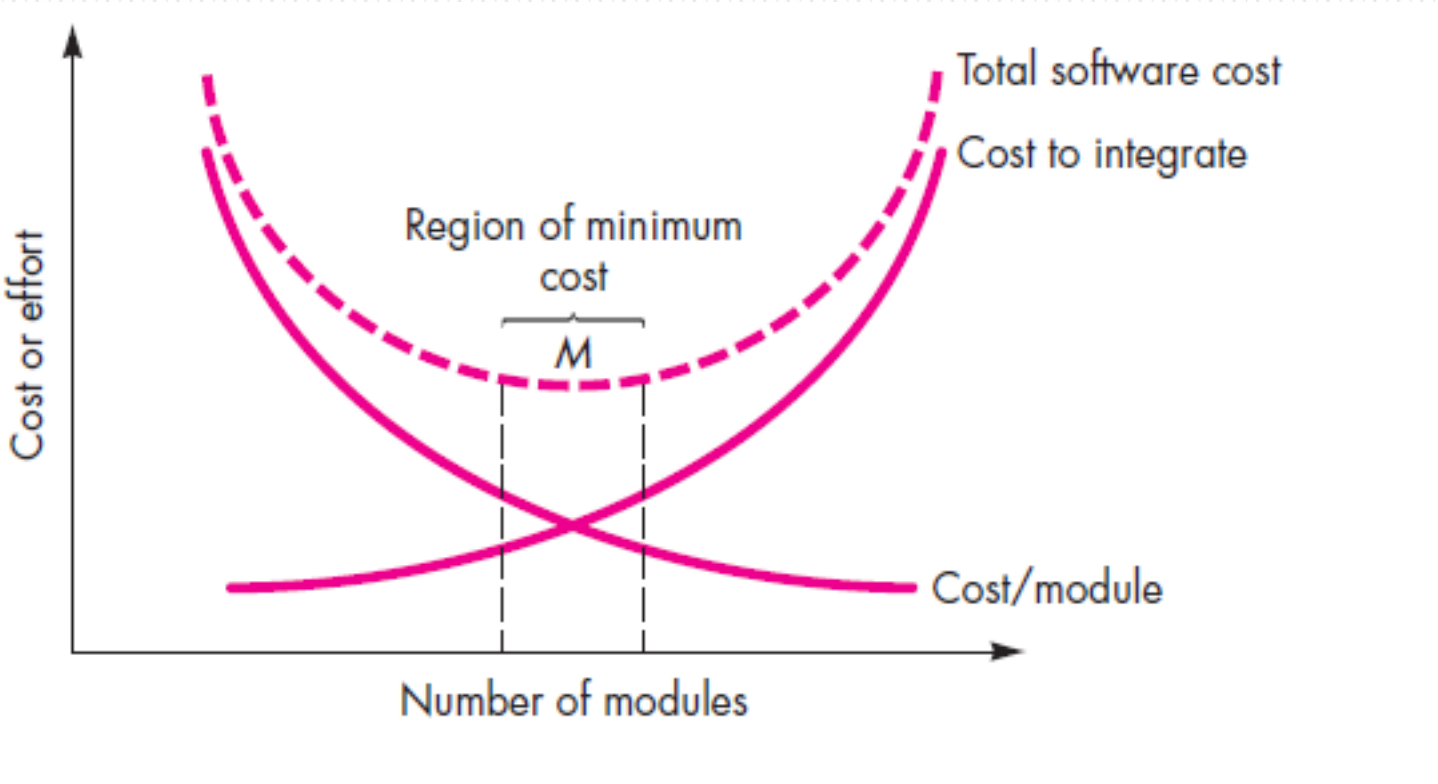
- Any complex problem can be more easily handled if it is subdivided into pieces that can each be solved and/or optimized independently
- A *concern* is a feature or behavior that is specified as part of the requirements model for the software
- By separating concerns into smaller, and therefore more manageable pieces, a problem takes less effort and time to solve.

Modularity

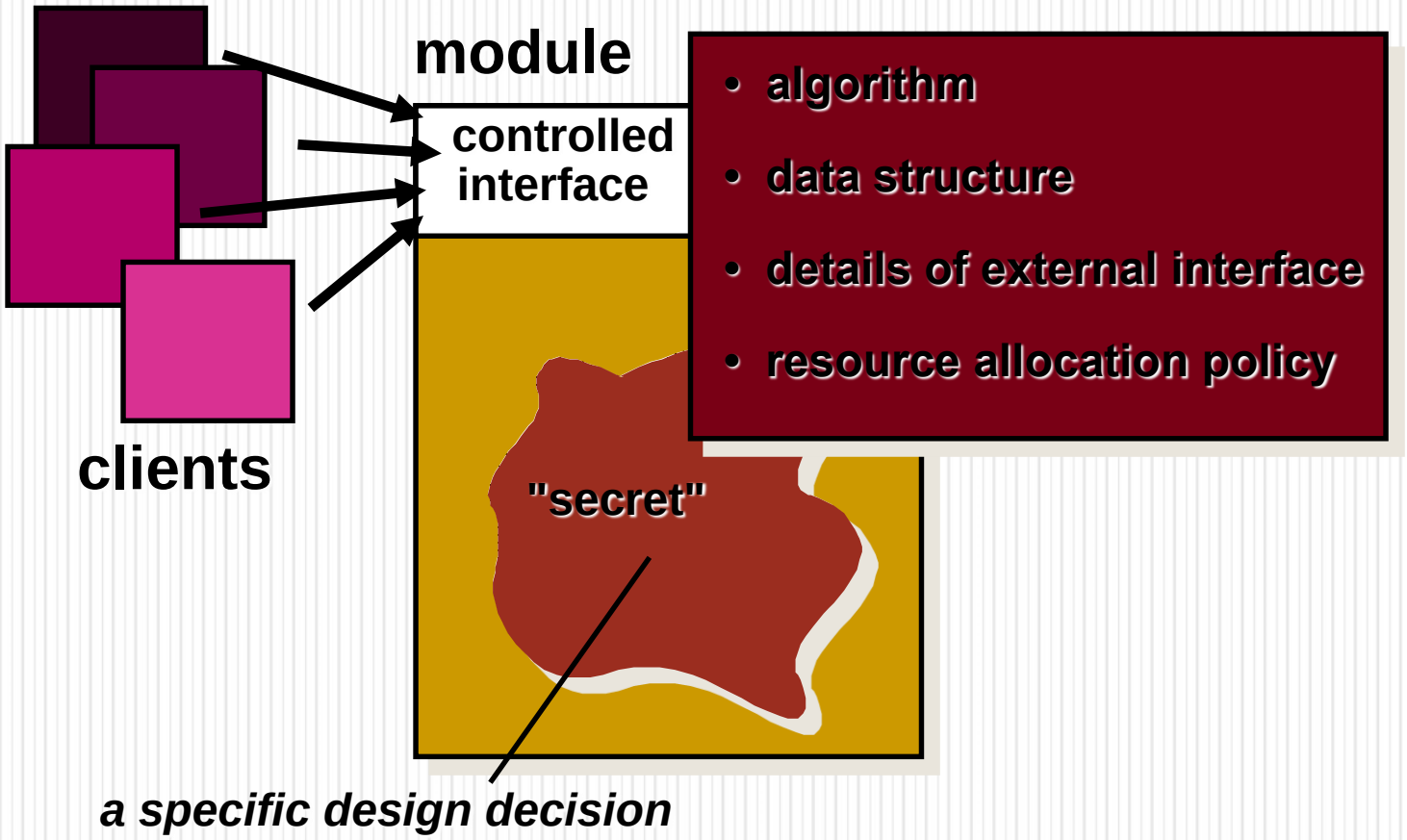
- “Modularity is the single attribute of software that allows a program to be intellectually manageable".
- Monolithic software (i.e., a large program composed of a single module) cannot be easily grasped by a software engineer.
  - The number of control paths, number of variables, and overall complexity would make understanding close to impossible.
- In almost all instances, you should break the design into many modules, hoping to make understanding easier and as a consequence, reduce the cost required to build the software.

Modularity: Trade-offs

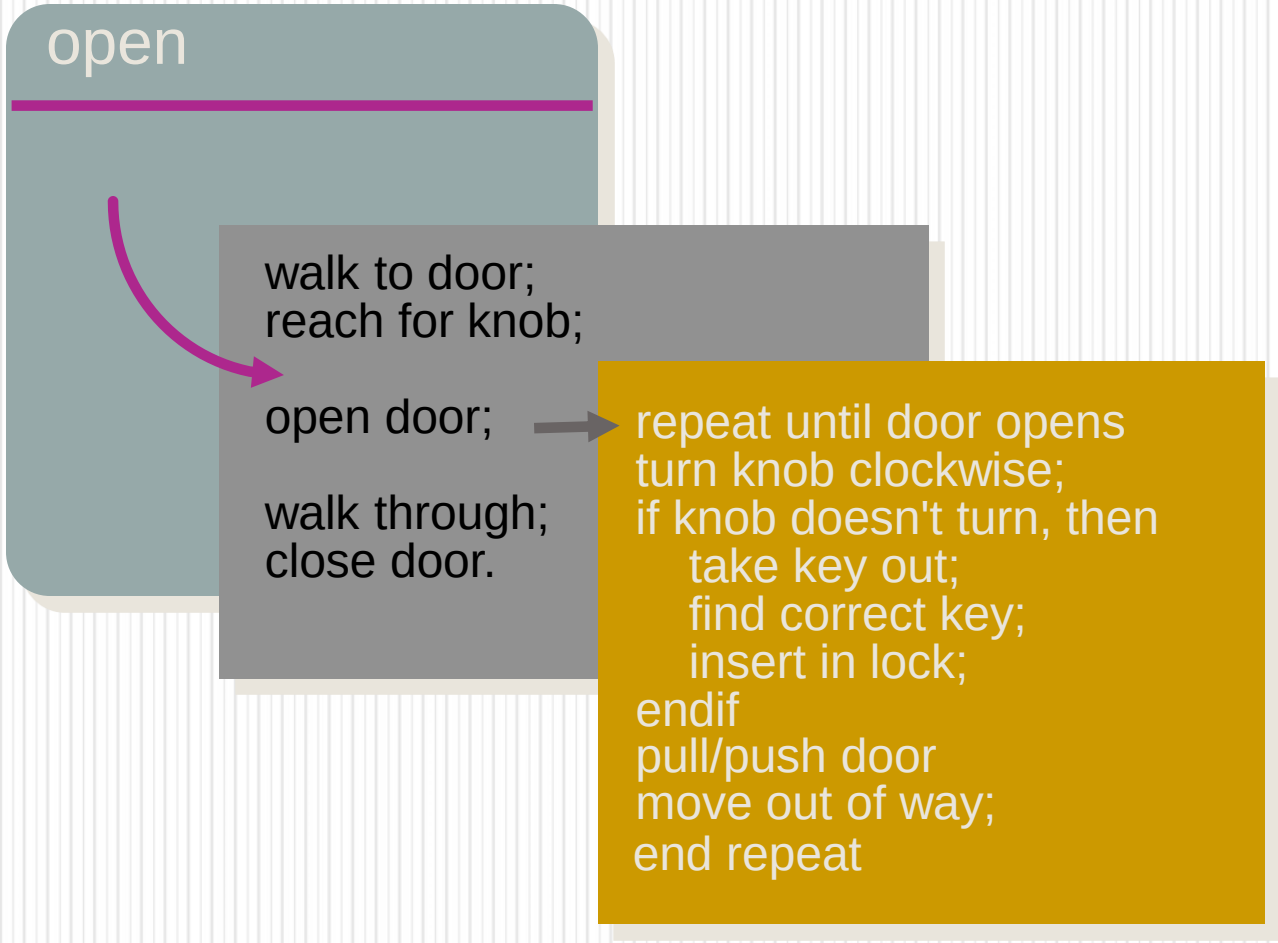
What is the "right" number of modules for a specific software design?



Information Hiding



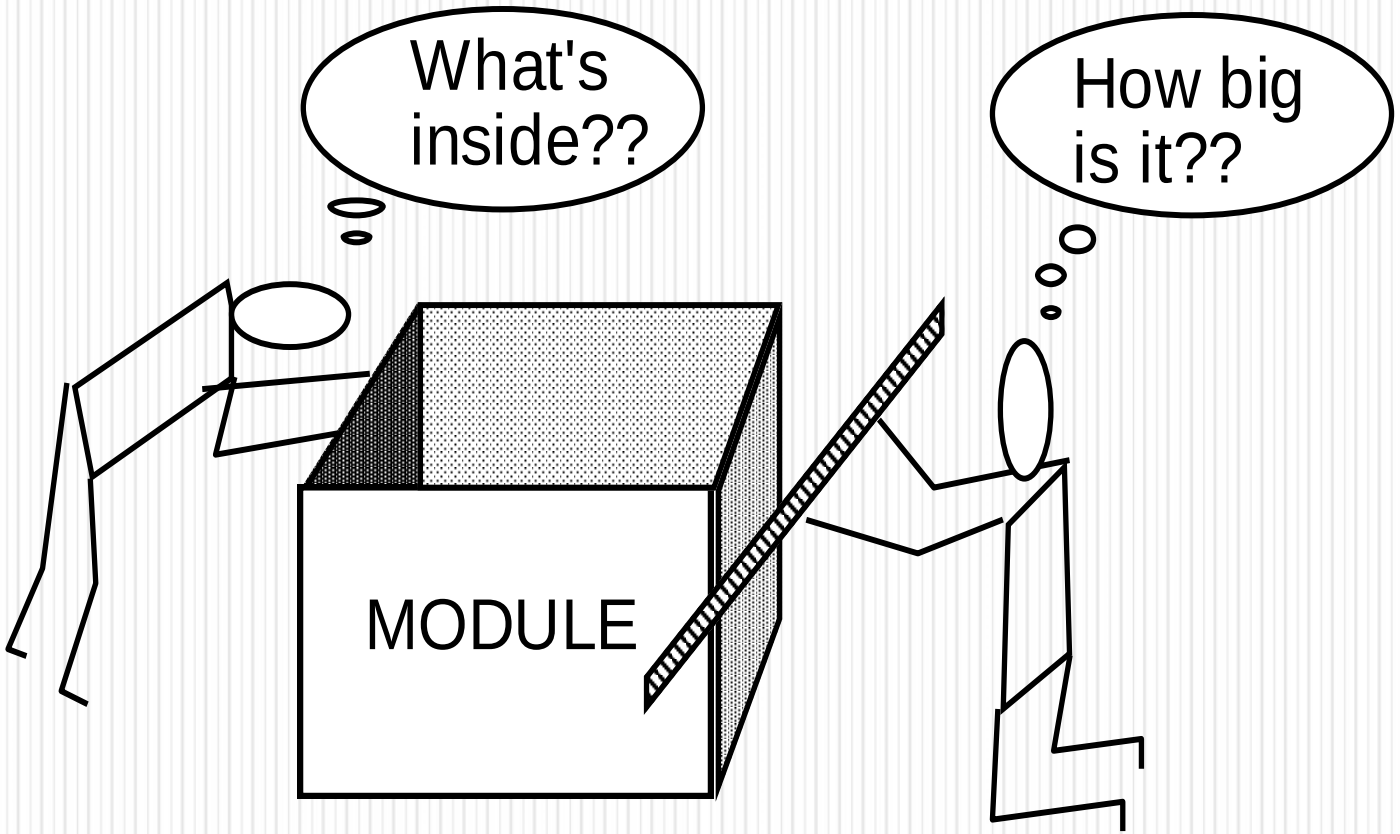
Stepwise Refinement



Abstraction & Refinement

- Abstraction and refinement are complementary concepts.
- Abstraction enables you to specify procedure and data internally but suppress the need for “outsiders” to have knowledge of low-level details.
- Refinement helps you to reveal low-level details as design progresses.
- Both concepts allow you to create a complete design model as the design evolves.

Sizing Modules: Two Views



Functional Independence

- Functional independence is achieved by developing modules with "single-minded" function and an "aversion" to excessive interaction with other modules.
- *Cohesion* is an indication of the relative functional strength of a module.
  - A cohesive module performs a single task, requiring little interaction with other components in other parts of a program. Stated simply, a cohesive module should (ideally) do just one thing.
- *Coupling* is an indication of the relative interdependence among modules.
  - Coupling depends on the interface complexity between modules, the point at which entry or reference is made to a module, and what data pass across the interface.

Refactoring

- Refactoring is defined in the following manner:
  - "Refactoring is the process of changing a software system in such a way that it does not alter the external behavior of the code [design] yet improves its internal structure."
- When software is refactored, the existing design is examined for
  - redundancy
  - unused design elements
  - inefficient or unnecessary algorithms
  - poorly constructed or inappropriate data structures
  - or any other design failure that can be corrected to yield a better design.

Thank You!!

ANY QUESTIONS???