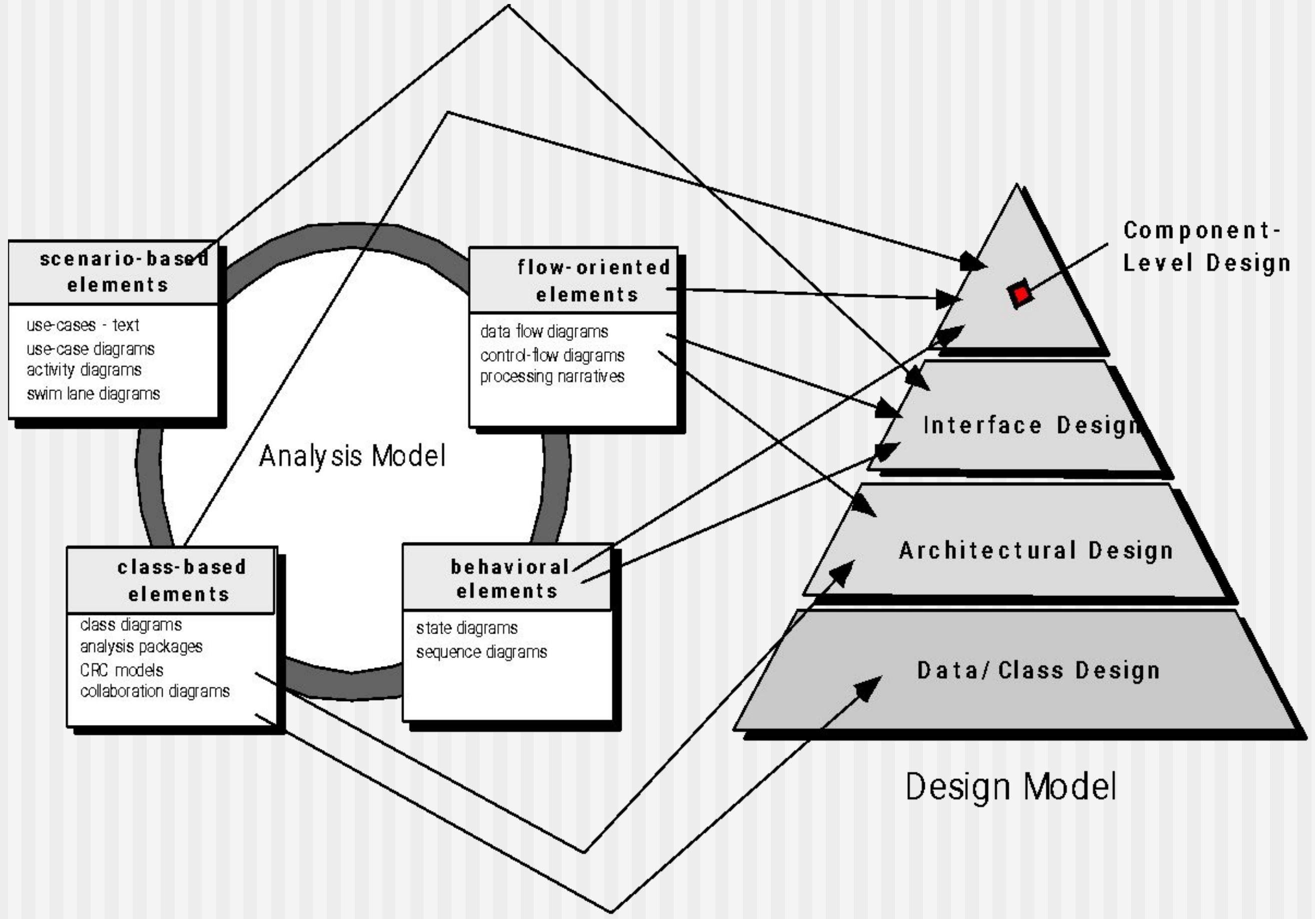
Chapter 8

- **Design Concepts**

Design

- A software design strategy:
  - Good software design should exhibit:
  - *Firmness:* A program should not have any bugs that inhibit its function.
  - *Commodity:* A program should be suitable for the purposes for which it was intended.
  - *Delight:* The experience of using the program should be pleasurable one.

Analysis Model -> Design Model



Characteristics as a guide for good design

- 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.

Quality Guidelines

- A design should exhibit an architecture that (1) has been created using recognizable architectural styles or patterns, (2) is composed of components that exhibit good design characteristics and (3) can be implemented in an evolutionary fashion
    - For smaller systems, design can sometimes be developed linearly.
- A design should be modular; that is, the software should be logically partitioned into elements or subsystems
- A design should contain distinct representations of data, architecture, interfaces, and components.
- A design should lead to data structures that are appropriate for the classes to be implemented and are drawn from recognizable data patterns.
- A design should lead to components that exhibit independent functional characteristics.
- A design should lead to interfaces that reduce the complexity of connections between components and with the external environment.
- A design should be derived using a repeatable method that is driven by information obtained during software requirements analysis.
- A design should be represented using a notation that effectively communicates its meaning.

Quality Attributes

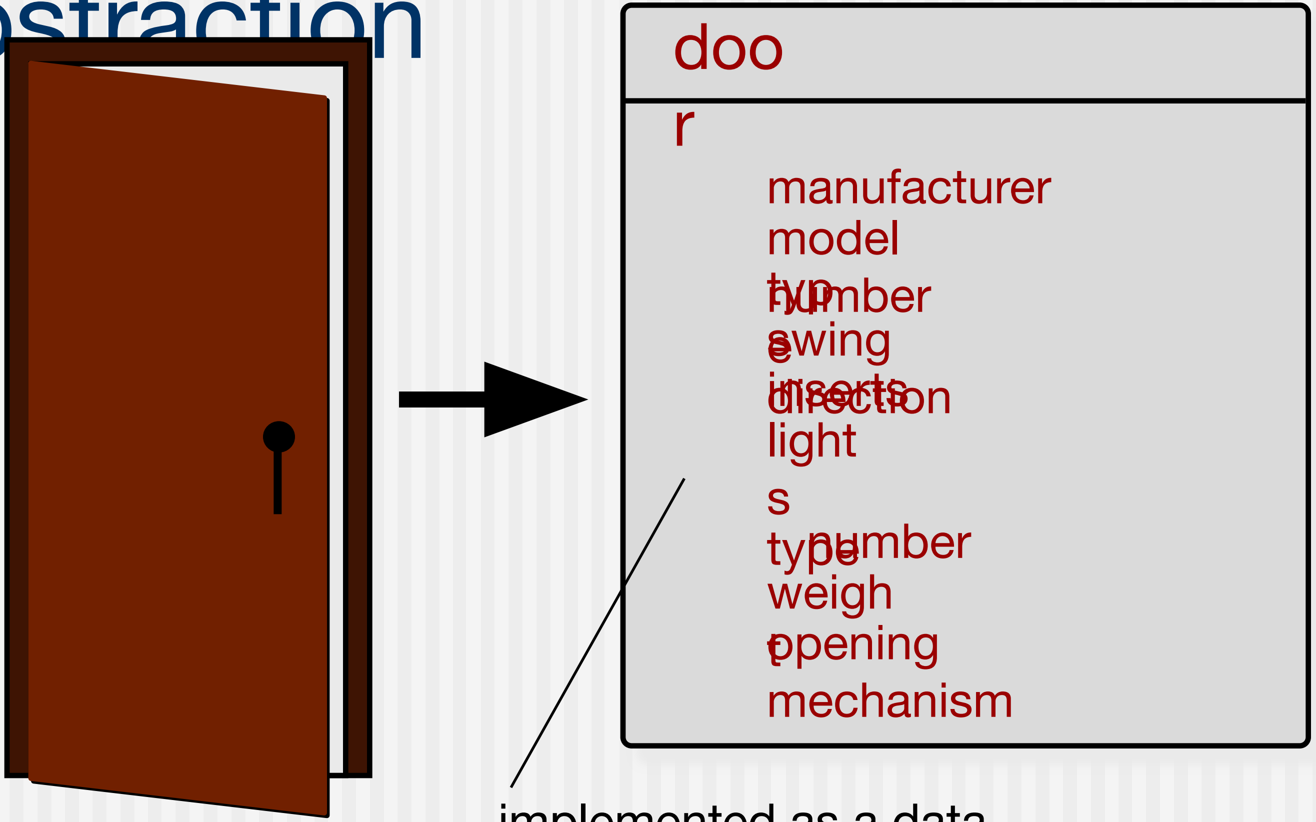
FURPS Quality Attributes:

- Functionality
  - Assessed by evaluating the feature set and capabilities of the program(i.e Functions, Security)
- Usability
  - Assessed by considering human factors (aesthetic, consistency, documentation)
- Reliability
  - Assessed by measuring accuracy of output
- Performance
  - Assessed by processing speed, response time, resource consumption, throughput, efficiency
- Supportability : assess based on maintainability

Fundamental Concepts

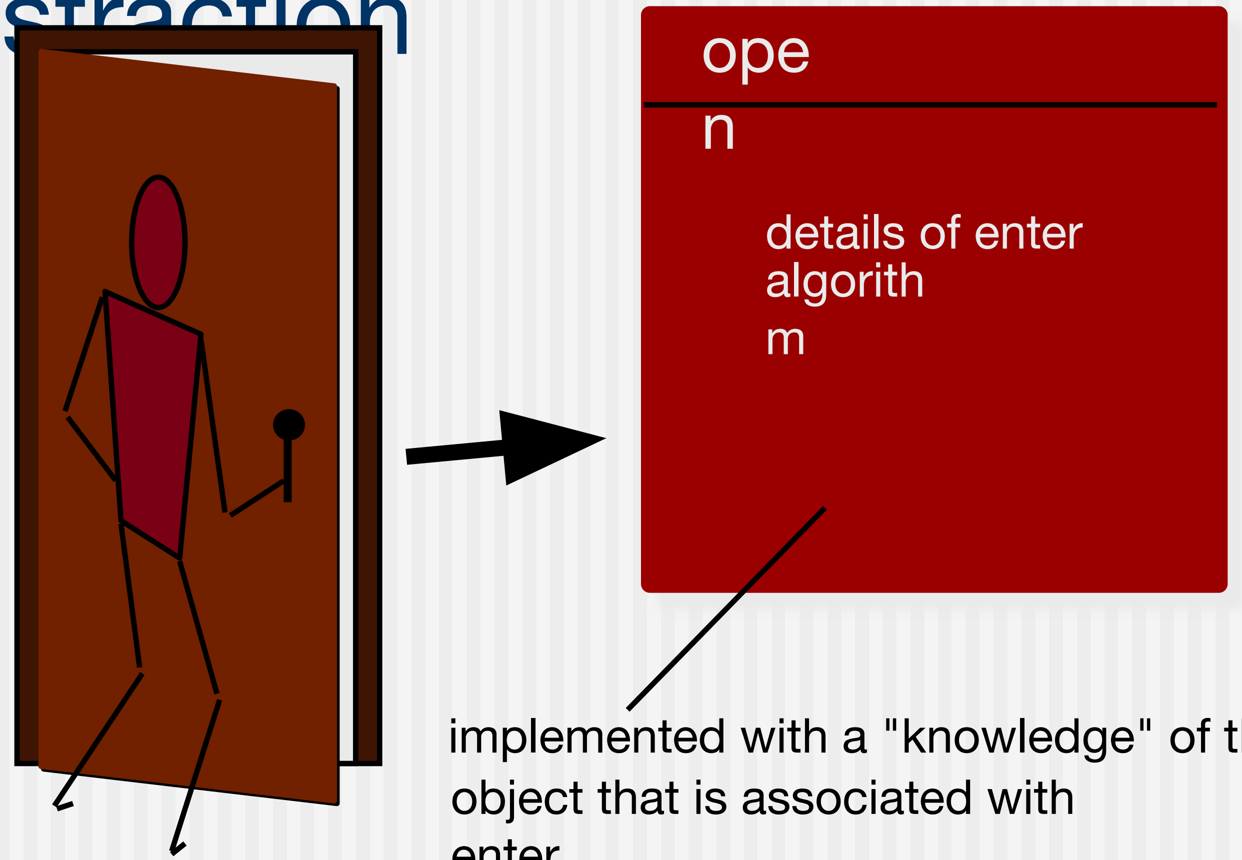
- Abstraction—data, procedure
- Architecture—the overall structure of the software
- Patterns—”conveys the essence” of a proven design solution
- Separation of concerns—any complex problem can be more easily handled if it is subdivided into pieces
- Modularity—compartmentalization of data and function
- 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
- OO design concepts-data hiding, polymorphism, inheritance….
- Design Classes—provide design detail that will enable analysis classes to be implemented , characteristics of well-formed design classes are : Complete and Sufficient, Primitive (method should focus on one service for the class) High cohesive, low coupling

Data Abstraction



implemented as a data structure

Procedural Abstraction



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

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. 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.

Patterns

Design Pattern Template

Pattern name—describes the essence of the pattern in a short but expressive name

Intent—describes the pattern and what it does

Also-known-as—lists any synonyms for the pattern

Motivation—provides an example of the problem

Applicability—notes specific design situations in which the pattern is applicable

Structure—describes the classes that are required to implement the pattern

Participants—describes the responsibilities of the classes that are required to implement the pattern

Collaborations—describes how the participants collaborate to carry out their responsibilities

Consequences—describes the "design forces" that affect the pattern and the potential trade-offs that must be considered when the pattern is implemented

Related patterns—cross-references related design patterns

Separation of Concerns

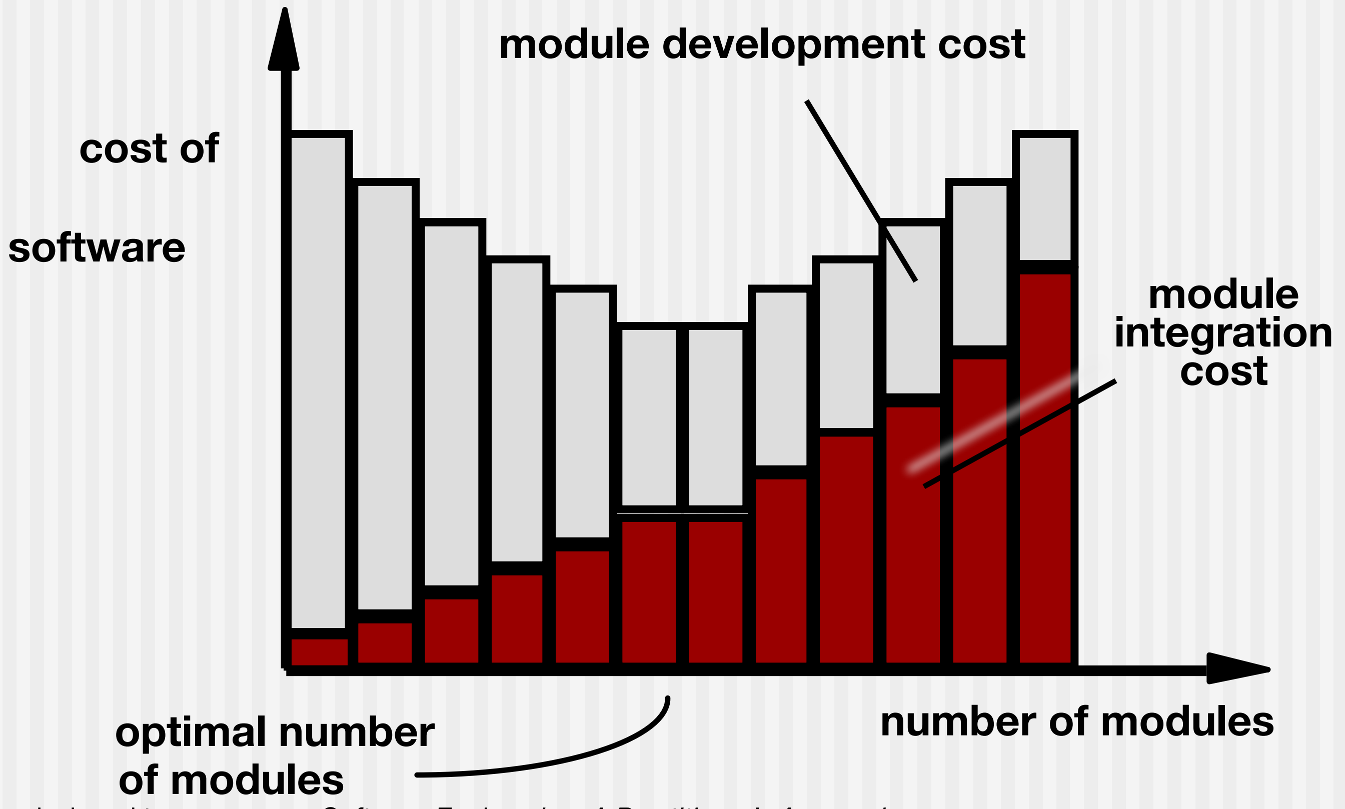
- Any complex problem can be more easily handled if it is subdivided into pieces that can each be solved and/or optimized independently
- A *concern* is a feature or behavior that is specified as part of the requirements model for the software
- By separating concerns into smaller, and therefore more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

Modularity

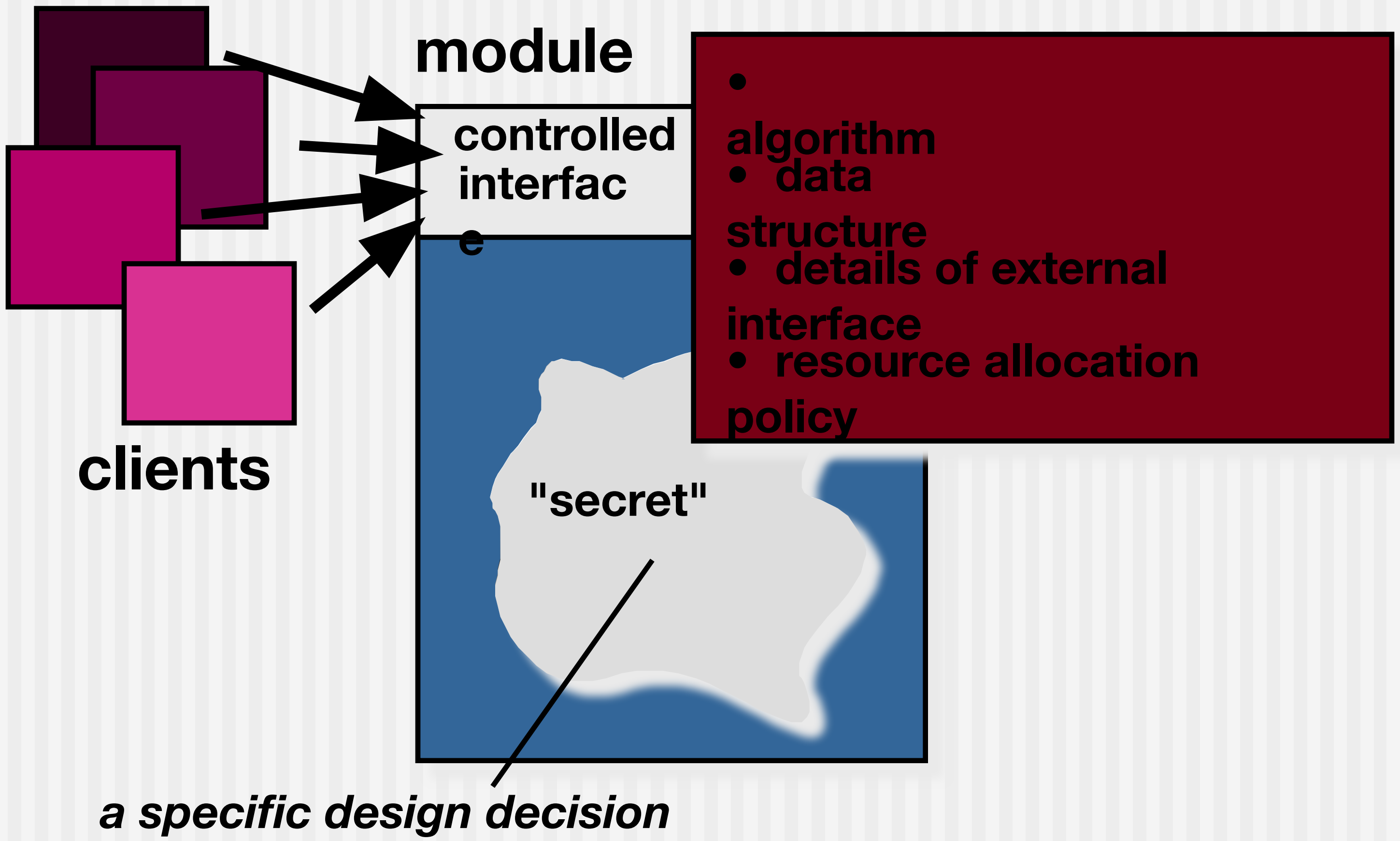
- "modularity is the single attribute of software that allows a program to be intellectually manageable" [Mye78].
- Monolithic software (i.e., a large program composed of a single module) cannot be easily grasped by a software engineer.
  - The number of control paths, span of reference, number of variables, 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?



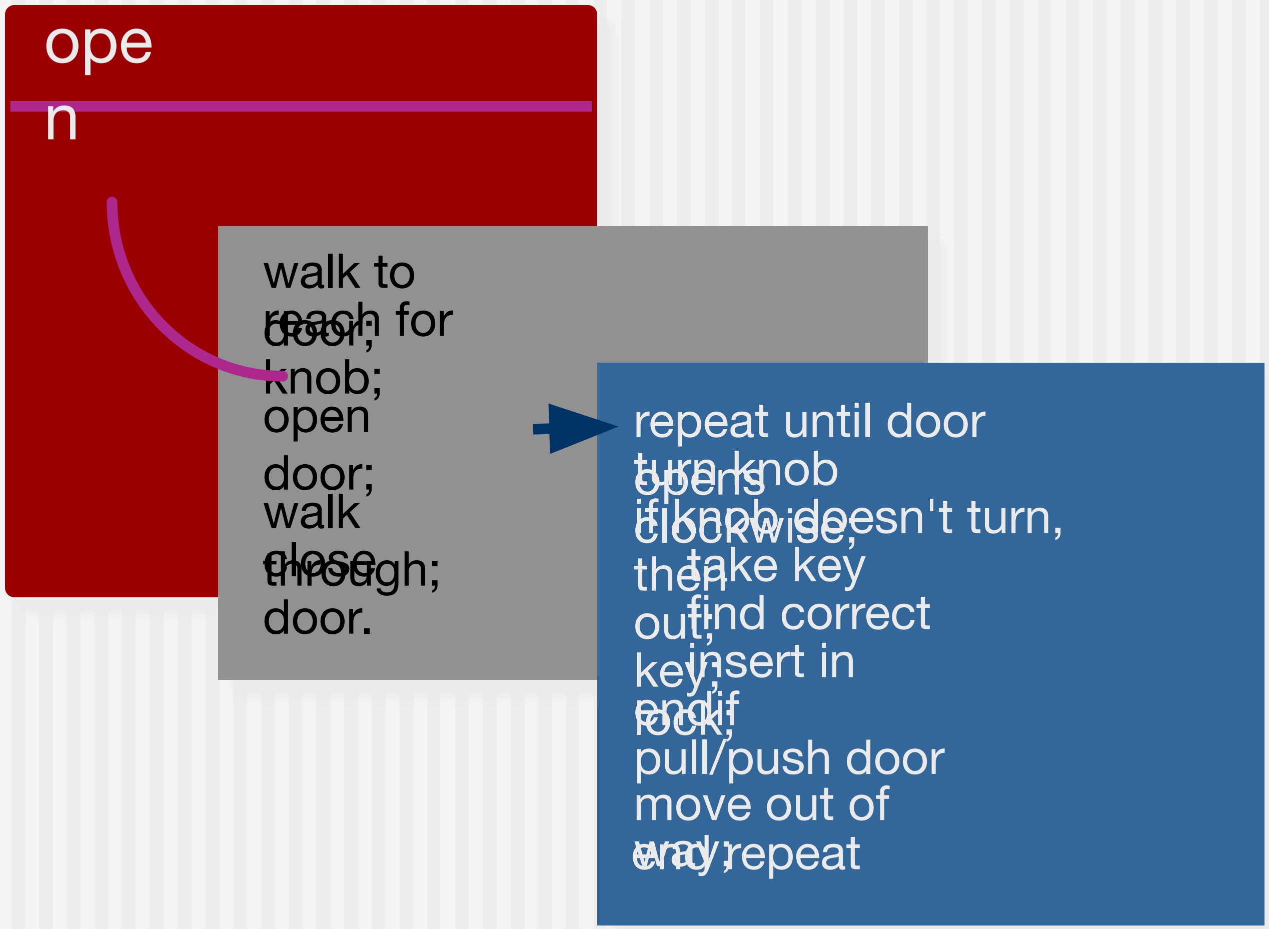
Information Hiding


module
controlled interfac e
• algorithm
• data structure
• details of external interface
• resource allocation policy
clients
"secret"
a specific design decision

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
- leads to encapsulation—an attribute of high quality design
- results in higher quality software

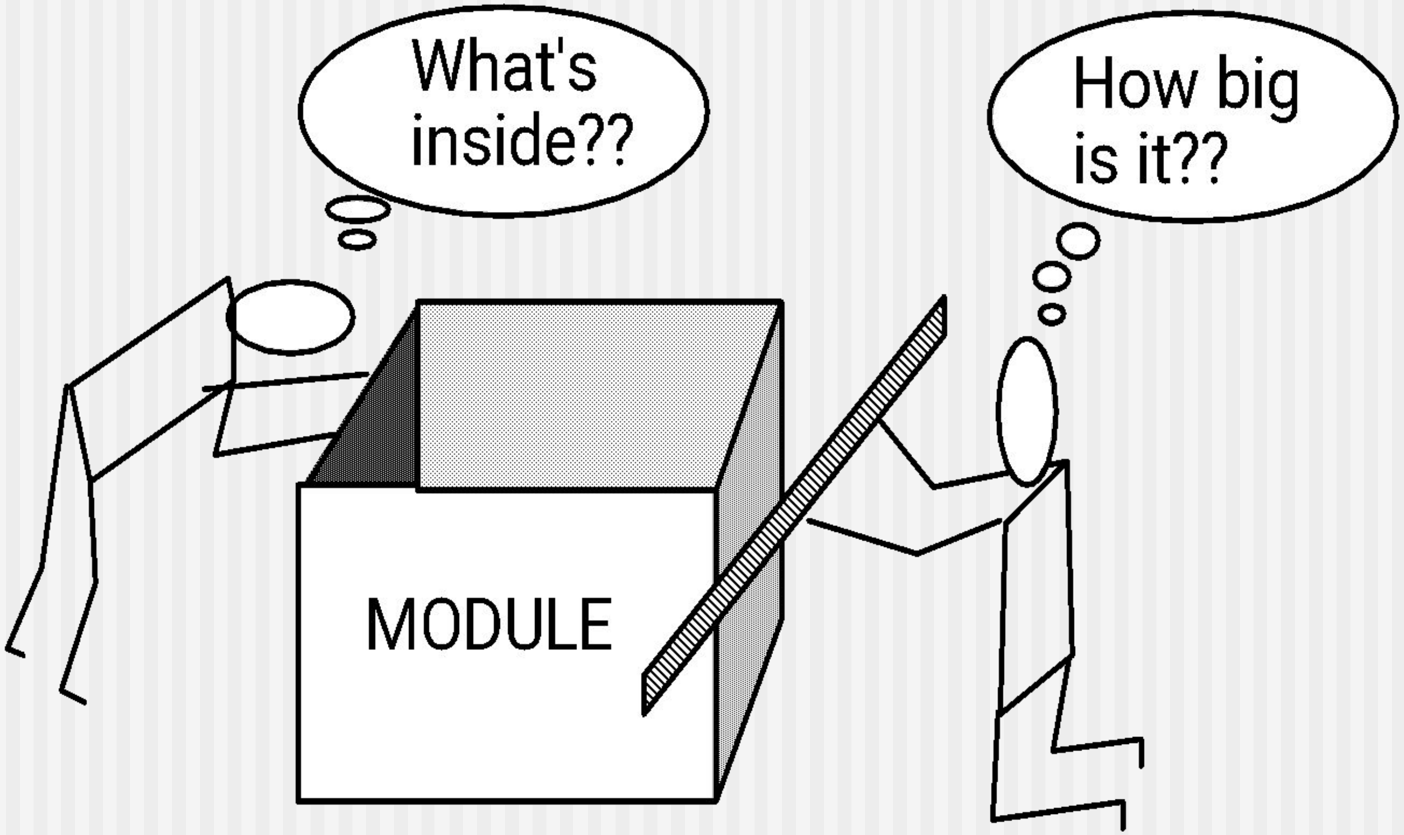
Stepwise Refinement



```
walk to door;
reach for knob;
open door;
walk through;
close door.
```

```
repeat until door opens
   turn knob clockwise;
   if knob doesn't turn, then
      take key out;
      find correct key;
      insert in lock;
   endif
   pull/push door
   move out of way;
end repeat
```

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.

Aspects

A mechanism for understanding how global requirements affect design

- Consider two requirements, *A* and *B*. Requirement *A* *crosscuts* requirement *B* "if a software decomposition [refinement] has been chosen in which *B* cannot be satisfied without taking *A* into account.
- An *aspect* is a representation of a cross-cutting concern.
- [Cross-cutting concerns are aspects of a program that affect other concerns
- Cross-cutting concerns are parts of a program that rely on or must affect many other part of the system.

Refactoring

- refactoring :
  - "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.

Design Model Elements

- Data elements is derived
  - At architectural level data design which focuses on database
  - At component level data design which considers the data structures that are required to implement local data objects
- Architectural elements derived from
  - Information about Application domain
  - specific requirements model elements such as data flow diagrams or analysis classes, their relationships and collaborations for the problem at hand
  - the availability of architectural patterns and styles
- Interface elements
  - the user interface (UI)
  - external interfaces to other systems, devices, networks or other producers or consumers of information
  - internal interfaces between various design components.
- Component elements define each of the modules that populate the architecture
- Deployment elements allocate physical configuration

As each of these element is developed a more complete view of the design evolves

Exercise

- Explain in brief good s/w design strategy.
- Explain in brief fundamental concepts of design
- Explain design triangle elements and its mapping with elements of requirement analysis