Design Concepts

Software Design

- **Iterative process**
- Convert user requirements to *blueprints* which helps the programmer in software implementation.
- The design model is different from requirements ; it shows the software architecture, interfaces, components & data structures used.
- Initially, the represented design is the high level abstract view of system functionality and architectural details.
- Later on, more iterations lead to much refined and detailed design.

These slides are designed to accompany *Software Engineering: A Practitioner's Approach, 7/e* (McGraw-Hill, 2009) Slides copyright 2009 by Roger Pressman.

Software Design

- Who does it?
- Why its important? Software quality established
- What are the design steps?
  - Design depicts the software as:
    - **Architecture of the systems**
    - **Interfaces** that connects end users, others systems or components of software
    - **Software components** themselves
- Design verification?
  - Design assess by software team to check for errors, omissions, inconsistencies, other possible alternatives, constrained budgets and costs.

These slides are designed to accompany *Software Engineering: A Practitioner's Approach, 7/e* (McGraw-Hill, 2009) Slides copyright 2009 by Roger Pressman.

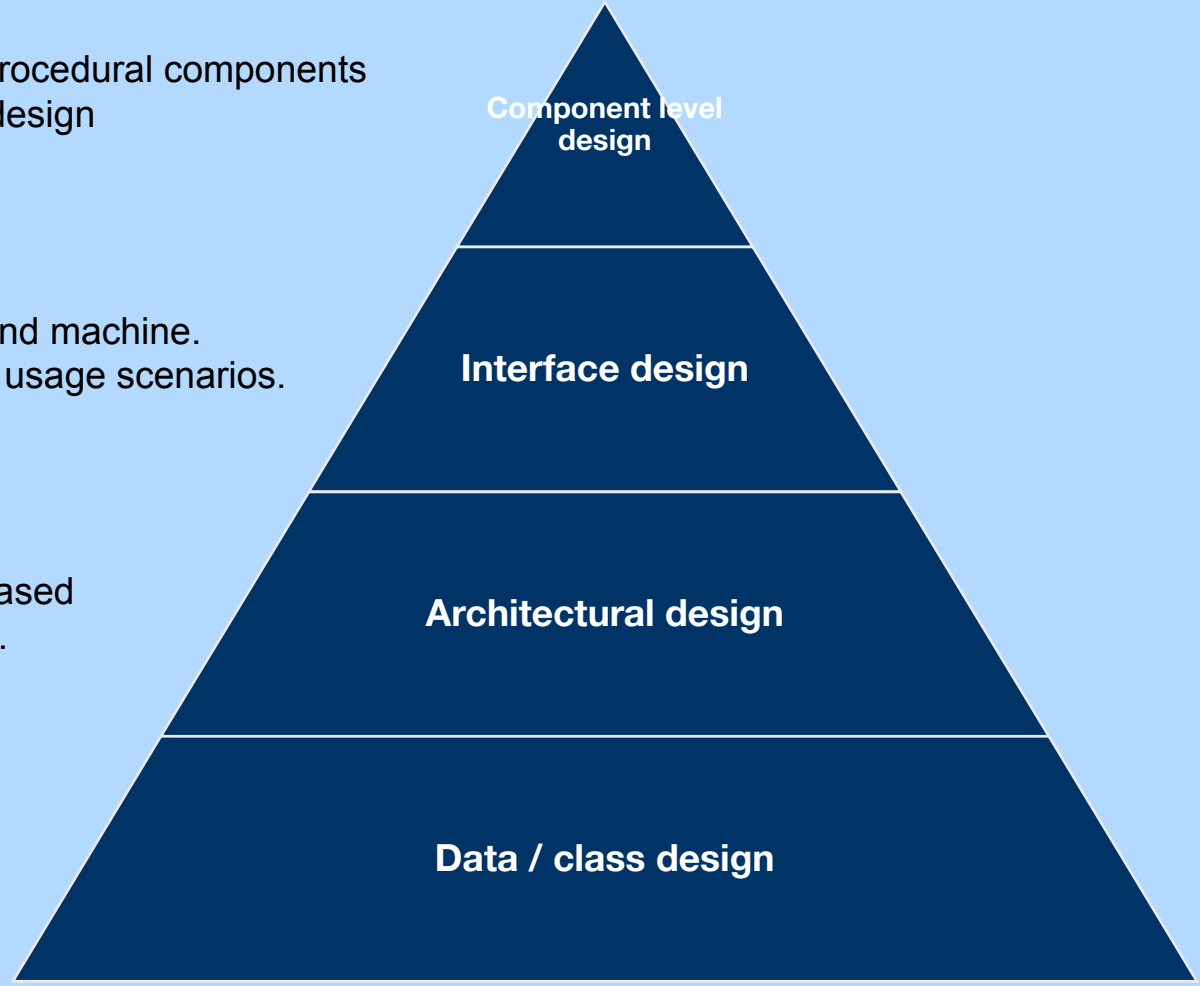
Software Design

Defines the structural elements in procedural components that can be developed using class design

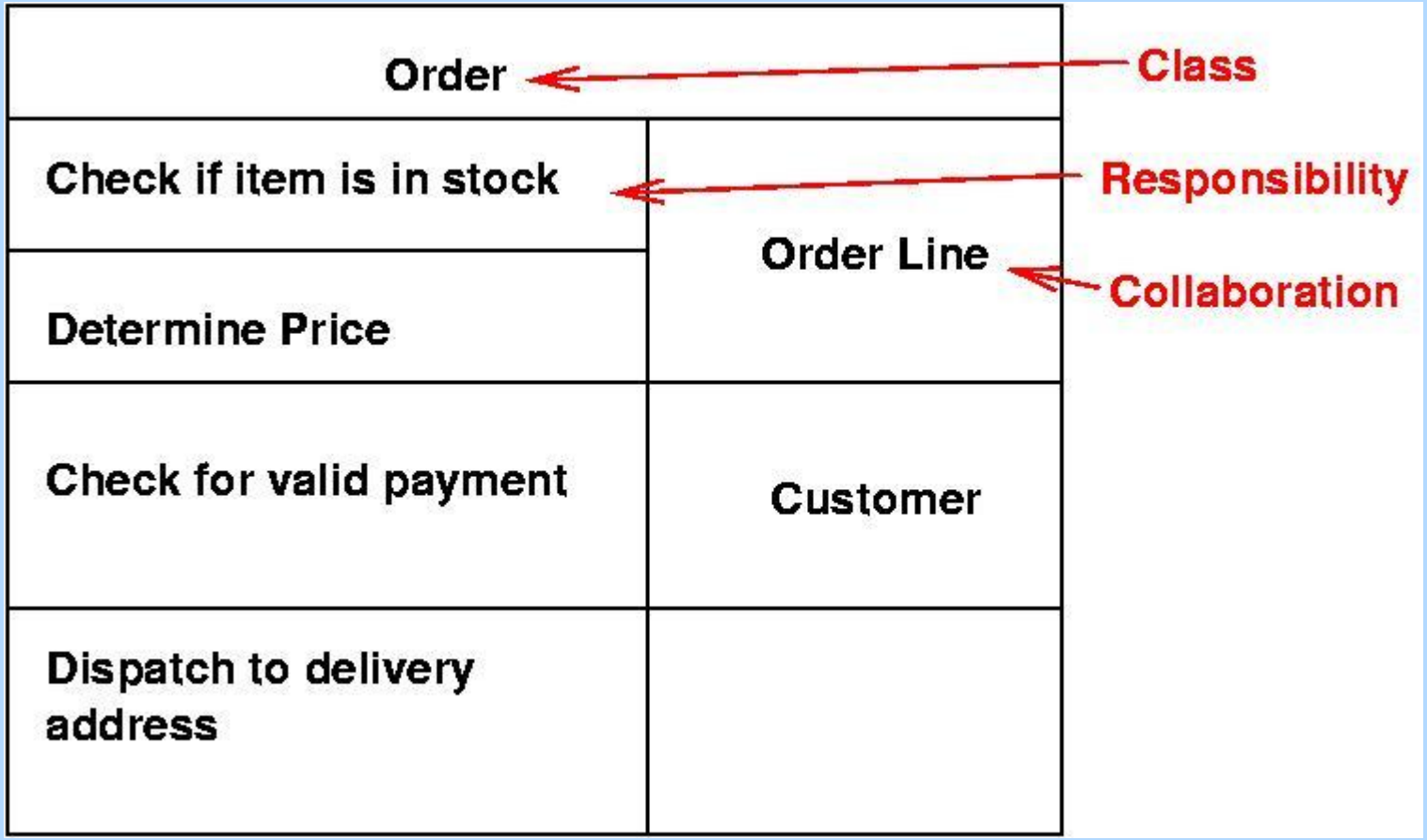
Defines data flow between human and machine. Done using behavioral analysis and usage scenarios.

Defines framework of a computer based system; standalone / distributed etc.

Define class attributes, objects and relationship defined in CRC (class responsibility collaborator) models



These slides are designed to accompany *Software Engineering: A Practitioner's Approach, 7/e* (McGraw-Hill, 2009) Slides copyright 2009 by Roger Pressman.



These slides are designed to accompany *Software Engineering: A Practitioner's Approach, 7/e* (McGraw-Hill, 2009) Slides copyright 2009 by Roger Pressman.

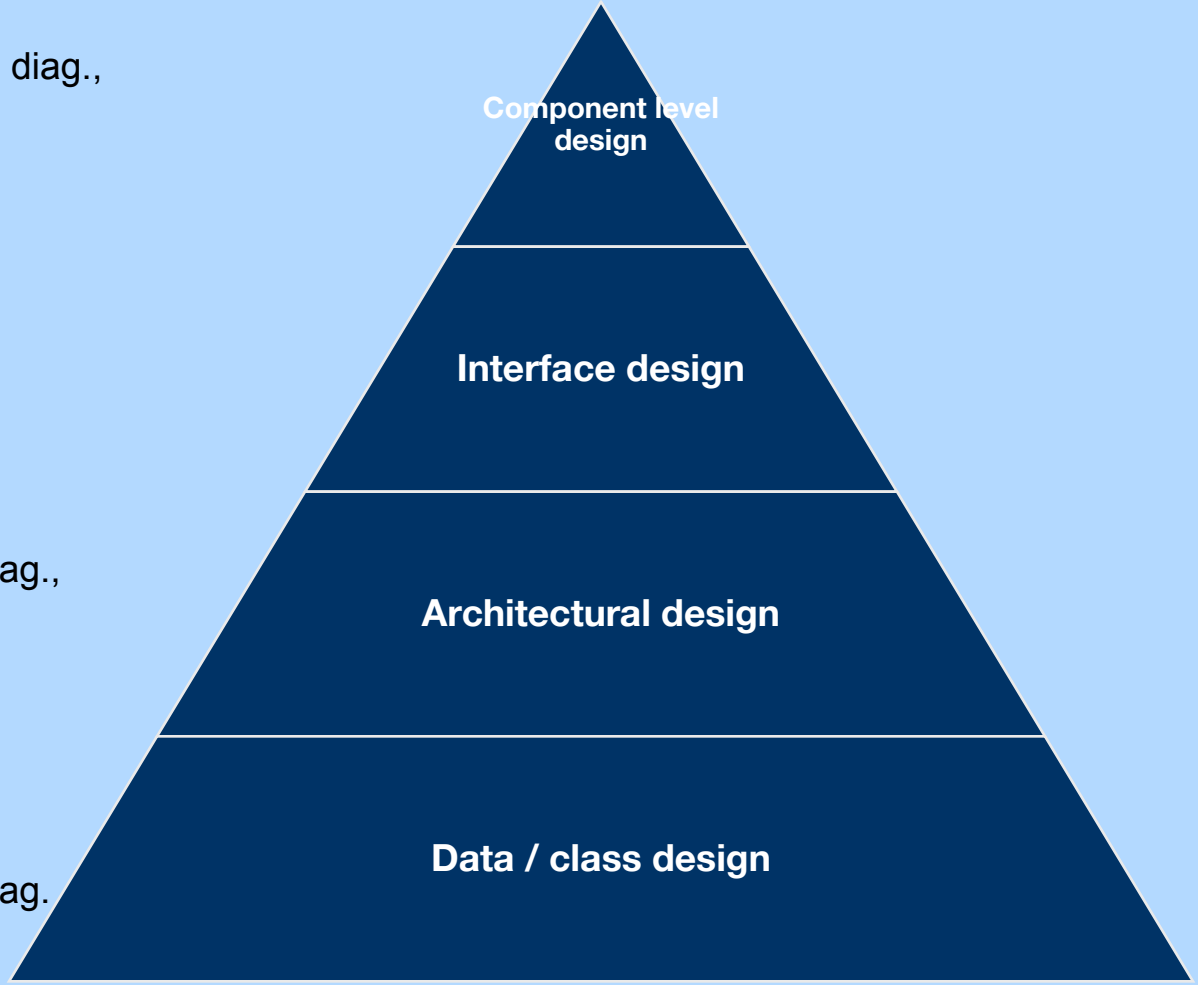
Software Design

DFD, sequence diag., state transition diag., class diagram

Use case diag., state transition diag. ,sequence diag. , DFDs

Class diagram, CRC, collaboration diag., DFD, control flow diag.

Class diagram, CRC, collaboration diag.



These slides are designed to accompany *Software Engineering: A Practitioner's Approach, 7/e* (McGraw-Hill, 2009) Slides copyright 2009 by Roger Pressman.

Good Software Design characteristics

Firmness:

- program should not have any bugs that inhibit its function.

Commodity:

- A program must serve the purpose for which it is designed.

Delight:

- The user experience should be pleasurable one.

Slides copyright 2009 by Roger Pressman.

Characteristics of a 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.
- **All are the ultimate goal of design process**

These slides are designed to accompany *Software Engineering: A Practitioner's Approach, 7/e* (McGraw-Hill, 2009) Slides copyright 2009 by Roger Pressman.

Quality Guidelines

- **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 must use appropriate data structures** 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 an understandable notation**

These slides are designed to accompany *Software Engineering: A Practitioner's Approach, 7/e* (McGraw-Hill, 2009) Slides copyright 2009 by Roger Pressman.

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.

These slides are designed to accompany *Software Engineering: A Practitioner’s Approach, 7/e* (McGraw-Hill, 2009) Slides copyright 2009 by Roger Pressman.

Attributes for assessment of design Quality

- **Functionality** ~ assess the capabilities of system & overall security
- **Usability** ~ assess by considering human factors like aesthetics, consistency, documentation
- **Reliability** ~ suggest possible severity of failures, no single point failures are appreciated
- **Performance** ~ assessed using resource consumptions, throughput, process speed, response time
- **Supportability** ~ ability to extend a system, lesser system limitations are good
- **Maintainability** ~ system installation and maintenance should be done easily

- **Suggested by HP**

These slides are designed to accompany *Software Engineering: A Practitioner's Approach, 7/e* (McGraw-Hill, 2009) Slides copyright 2009 by Roger Pressman.

software design Evolution

- Initially modular approach used ~ **structured programming**
  - Develop using top-down manner
- Later advances and involvement of data structures and reusability and dataflow reshaped it to **object oriented programming** paradigm.
- Now the design approaches are moving for **test driven developments**

These slides are designed to accompany *Software Engineering: A Practitioner's Approach, 7/e* (McGraw-Hill, 2009) Slides copyright 2009 by Roger Pressman.

How to make an effective design

- Problem partitioning
  - Define functions, modules and multiple products that could operate separatley
- Abstraction
  - Show only the upper level details of the requirements, hide implementation details
  - As in level 0 of DFD
- Top down / bottom up approach used

  - Top down used for reverse engineering problems -> you have a defined product and you need to customize it as per your needs

  - Bottom up -> you know modules / subsystems, you integrate them and make up final product

These slides are designed to accompany *Software Engineering: A Practitioner's Approach, 7/e* (McGraw-Hill, 2009) Slides copyright 2009 by Roger Pressman.

Fundamental Concepts

- Abstraction—data, procedure, control
- 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—Appendix II
- Design Classes—provide design detail that will enable analysis classes to be implemented

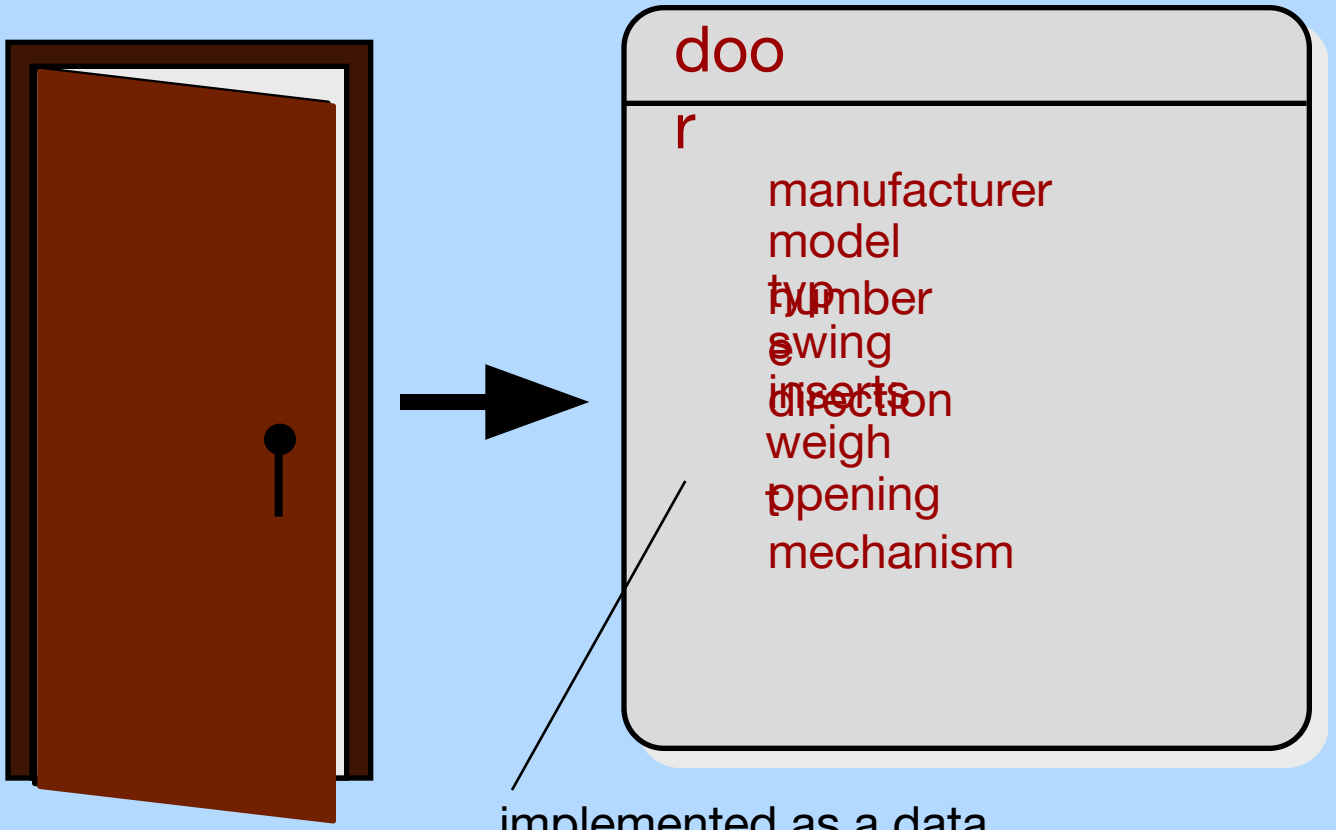
These slides are designed to accompany *Software Engineering: A Practitioner's Approach, 7/e* (McGraw-Hill, 2009) Slides copyright 2009 by Roger Pressman.

Abstraction

- Abstraction
  - In modular approaches we have multiple level of abstractions
  - The Higher the level, more the abstraction is
  - Lower level shows the implementation details
  - Data abstraction -> hiding attributes of an entity
  - Procedural abstraction -> hide process details

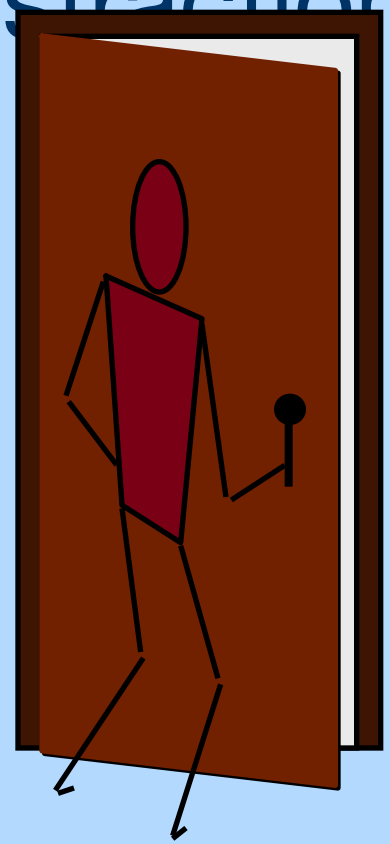
These slides are designed to accompany *Software Engineering: A Practitioner's Approach, 7/e* (McGraw-Hill, 2009) Slides copyright 2009 by Roger Pressman.

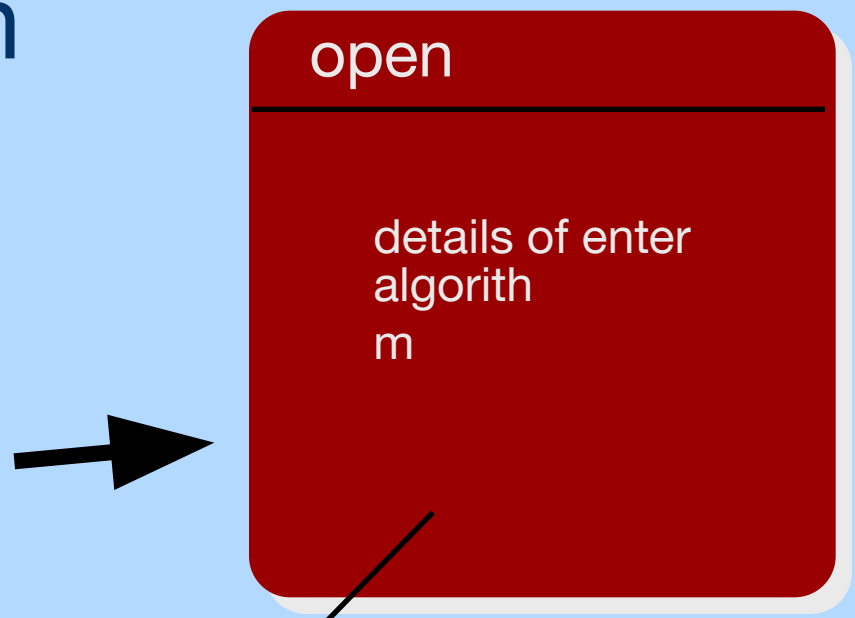
Data Abstraction



These slides are designed to accompany *Software Engineering: A Practitioner's Approach, 7/e* (McGraw-Hill, 2009) Slides copyright 2009 by Roger Pressman.

Procedural Abstraction





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

These slides are designed to accompany *Software Engineering: A Practitioner's Approach, 7/e* (McGraw-Hill, 2009) Slides copyright 2009 by Roger Pressman.