Lecture-19,20

Software Design

Dronacharya College of Engineering

Topics

- The Design Process
- Design Principles
- Design Concepts-Abstraction & Refinement
- Software Architecture
- Program Partitioning
- Coupling and Cohesion

Design Concepts

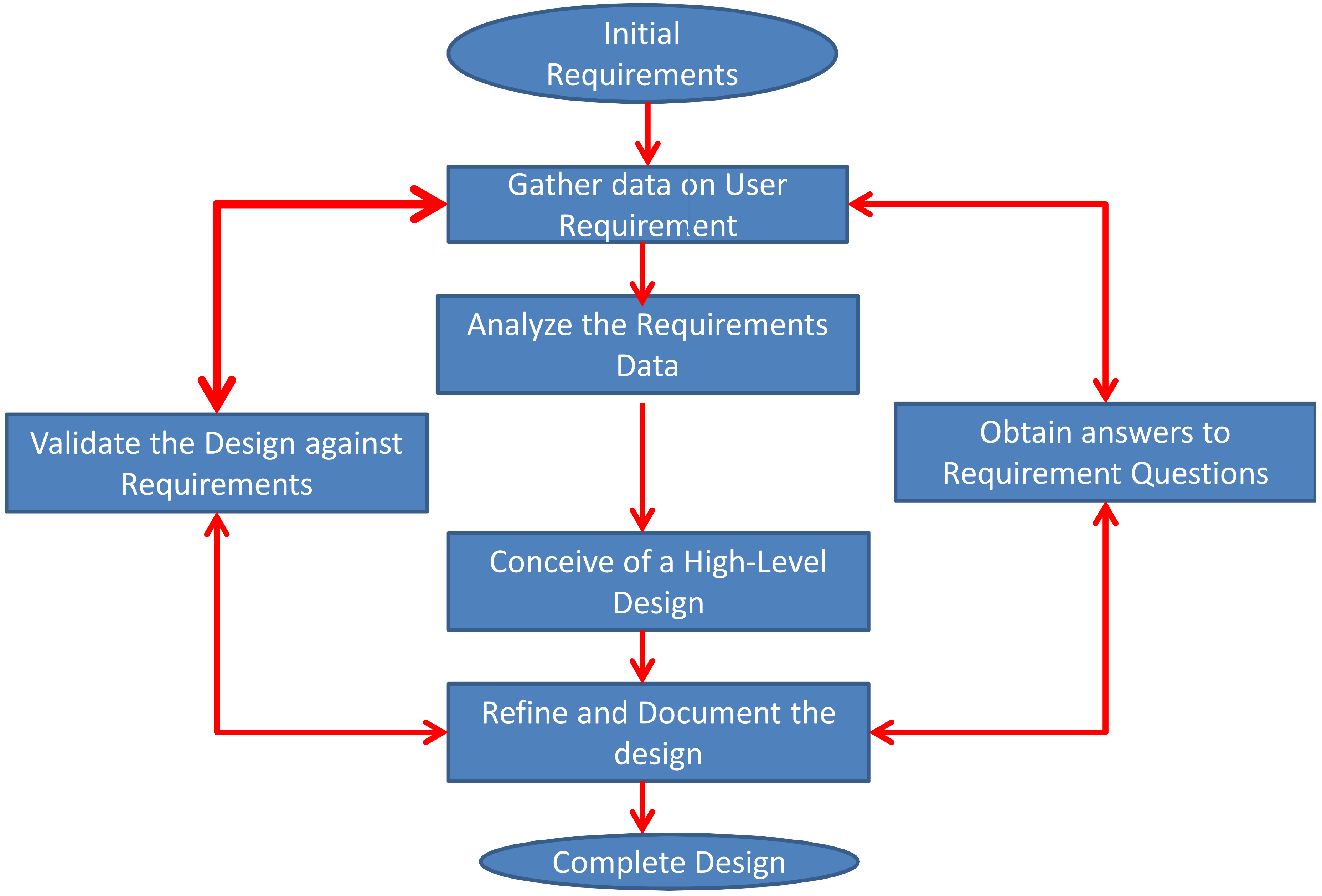
The design process for software systems often has two levels.

1. At the first level the focus is on deciding which modules are needed for the system this is called the "System design" or "Top-level design".
2. In the second level, the internal design of the modules or how the specifications of the modules can be satisfied is decided. This design level is often called "detailed design" or "logical design".

Software Design

Software Design -- An iterative process transforming requirements into a "blueprint" for constructing the software.

Software Design Framework


Initial Requirements
Gather data on User Requirement
Analyze the Requirements Data
Validate the Design against Requirements
Obtain answers to Requirement Questions
Conceive of a High-Level Design
Refine and Document the design
Complete Design

Relation of Analysis to Design

After analyzing and specifying all the requirements, the process of software design begins.

As you all now ,the elements of analysis model are

- ➢ Data Dictionary
- ➢ Entity Relationship Diagram
- ➢ Data Flow Diagram
- ➢ State Transition Diagram
- ➢ Control Specification
- ➢ Process Specification

Now the elements of Design Model are

- ✓ Data Design
- ✓ Architectural Design
- ✓ Interface Design
- ✓ Component-level Design

Relation of Analysis to Design

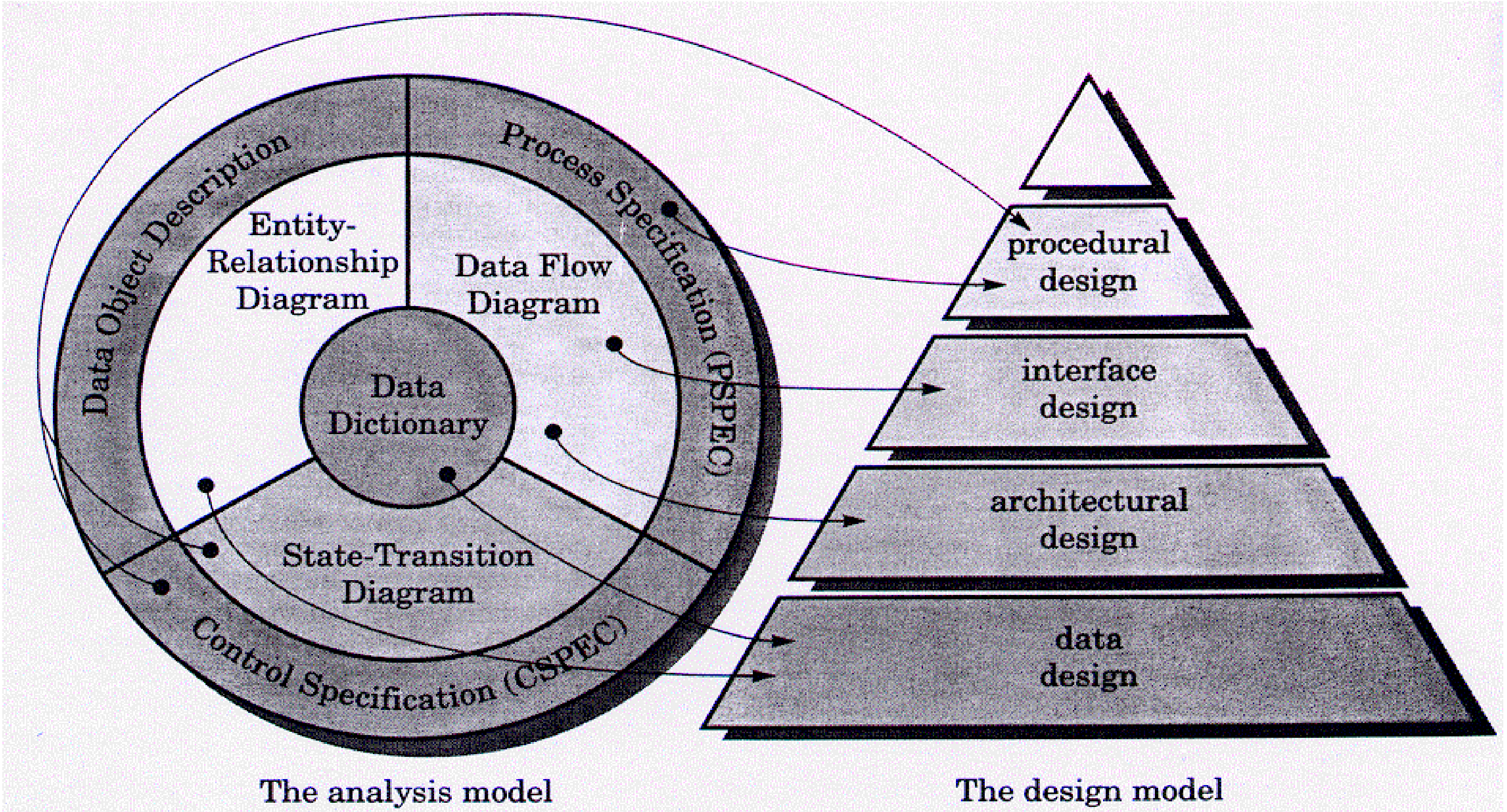
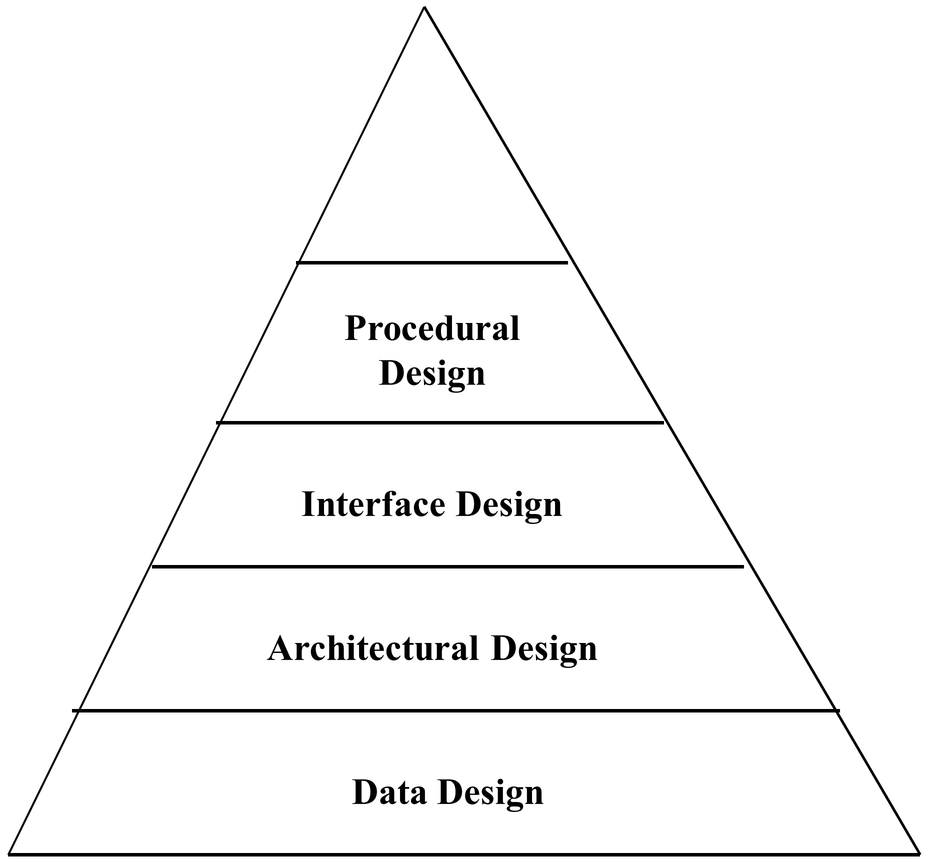


FIGURE 13.1. Translating the analysis model into a software design

The Design Model

- Data Design
  - Transforms information domain model into data structures required to implement software
- Architectural Design
  - Defines relationship among the major structural elements of a program

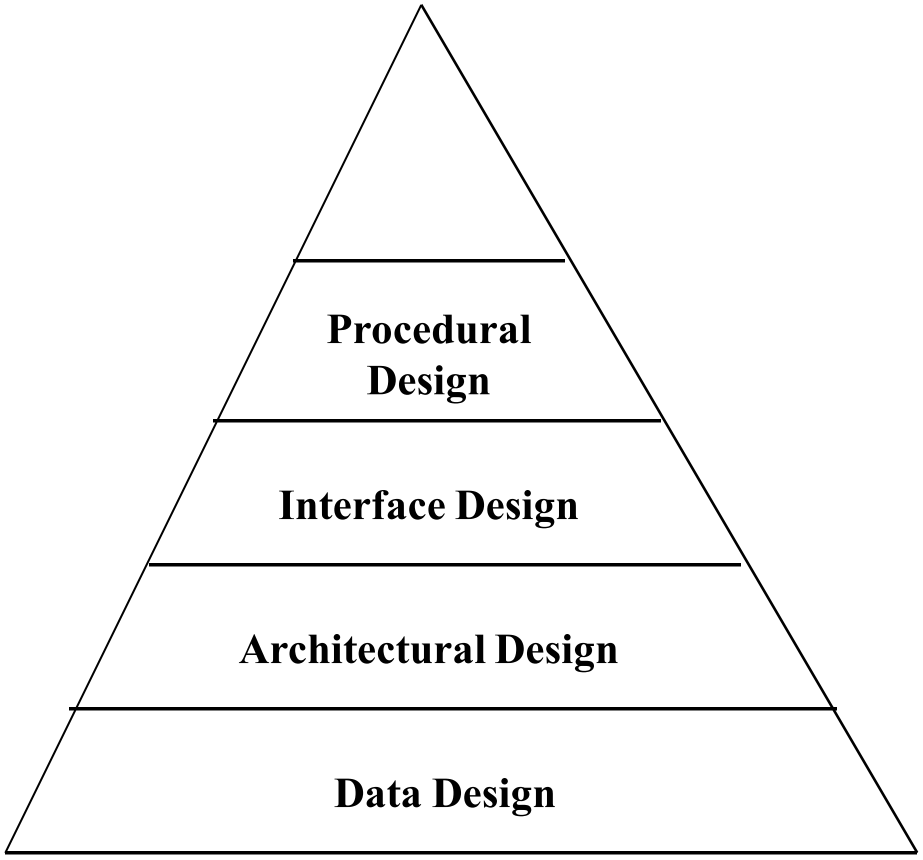


The Design Model

Which is mapped from the Analysis model

The Design Model

- Interface Design
  - Describes how the software communicates with itself, to systems that interact with it and with humans.
- Procedural Design
  - Transforms structural elements of the architecture into a procedural description of software construction



The Design Model

Which is mapped from the Analysis model

The Design Process

- Mc Glaughlin’s suggestions for good design:
  - Design must enable all requirements of the analysis model and implicit needs of the customer to be met
  - Design must be readable and an understandable guide for coders, testers and maintainers
  - The design should address the data, functional and behavioral domains of implementation

Design Guidelines

- A design should exhibit a hierarchical organization
- A design should be modular
- A design should contain both data and procedural abstractions
- Modules should exhibit independent functional characteristics
- Interfaces should reduce complexity

Design Principles

- **Design** principles:
  - The design process should consider various approaches based on requirements
  - The design should be traceable to the requirements analysis model
  - Design should be uniform and exhibit integrity
  - Design should accommodate change
  - Design should minimize coupling between modules
  - Design and coding are not interchangeable

Design Principles

- Design should be structured to degrade gently
  - It should terminate gracefully and not bomb suddenly

- Design should have quality assessment during creation, not afterwards
  - This is to reduce development time

- Design should be reviewed to minimize on conceptual errors -- Formal design reviews!

- There is a tendency to focus on the wrong things
  - All conceptual elements have to be addressed

What is not Design

- Design is not programming.
- Design is not modeling. Modeling is part of the architectural design.
- Design is not part of requirements.
- Where requirements finishes and where design starts ?.
- Requirements = What the system is supposed to do.
- Design = How the system is built.

Design Concepts

– The three basic design concepts are:

Problem Partitioning

Abstraction

Top-down and Bottom up design

Design Concepts-Abstraction

- Wasserman: “Abstraction permits one to concentrate on a problem at some level of abstraction without regard to low level details”
- Data Abstraction
  - This is a named collection of data that describes a data object
- Procedural Abstraction
  - Instructions are given in a named sequence
  - Each instruction has a limited function
- Control Abstraction
  - A program control mechanism without specifying internal details, e.g., semaphore, rendezvous

Refinement

- Refinement is a process where one or several instructions of the program are decomposed into more detailed instructions.
- Stepwise refinement is a top down strategy
  - Basic architecture is developed iteratively
  - Step wise hierarchy is developed
- Forces a designer to develop low level details as the design progresses
  - Design decisions at each stage

Modularity

- In this concept, software is divided into separately named and addressable components called modules
- Follows "divide and conquer" concept, a complex problem is broken down into several manageable pieces
- Let p_1 and p_2 be two program parts, and E the effort to solve the problem. Then,

  $E(p_1+p_2) > E(p_1)+E(p_2)$, often >>
- A need to divide software into optimal sized modules

Modularity

Objectives of modularity in a design method

- Modular Decomposability
  - Provide a systematic mechanism to decompose a problem into sub problems
- Modular Composability
  - Enable reuse of existing components
- Modular Understandability
  - Can the module be understood as a stand alone unit? Then it is easier to understand and change.

Modularity

- Modular Continuity
  - If small changes to the system requirements result in changes to individual modules, rather than system-wide changes, the impact of the side effects is reduced (note implications in previous example)

- Modular Protection
  - If there is an error in the module, then those errors are localized and not spread to other modules

Software Architecture

Desired properties of an architectural design

- Structural Properties
  - This defines the components of a system and the manner in which these interact with one another.
- Extra Functional Properties
  - This addresses how the design architecture achieves requirements for performance, reliability and security
- Families of Related Systems
  - The ability to reuse architectural building blocks

Structural Diagrams

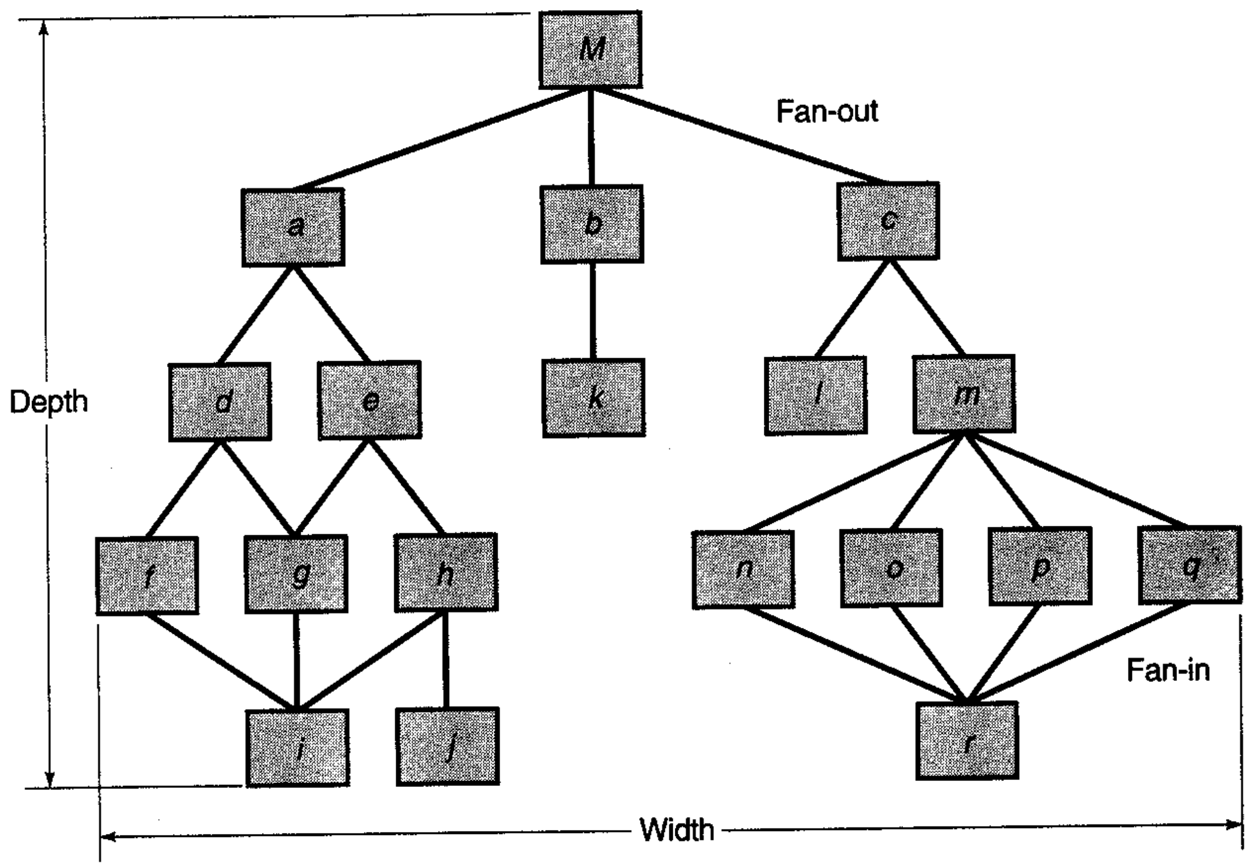


FIGURE 13.3.
Structure terminology

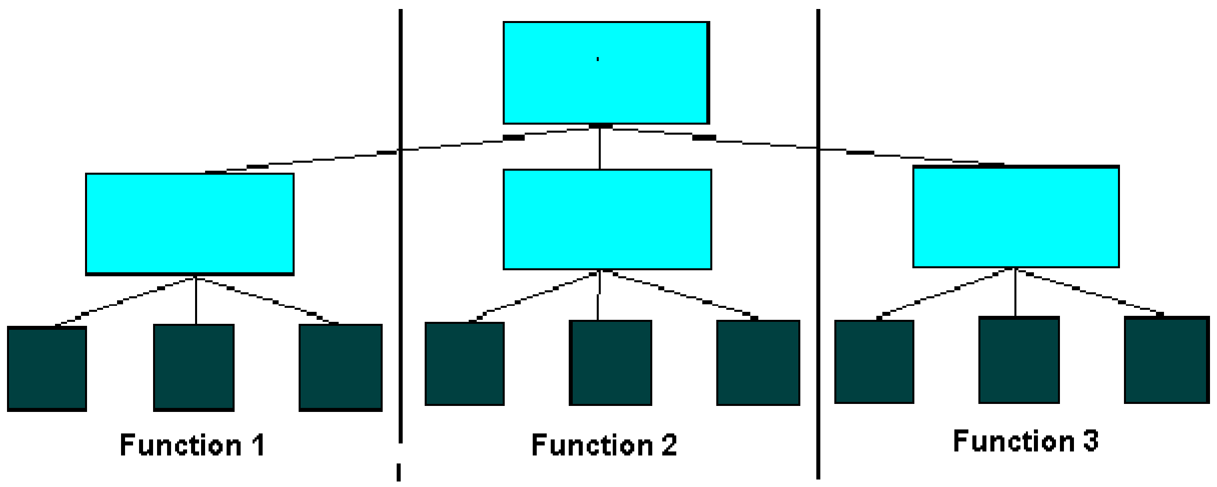
Kinds of Models

- Terminology
  - Structural models
    - Organized collection of components
  - Framework models
    - Abstract to repeatable architectural patterns
  - Dynamic models
    - Behavioral (dynamic) aspects of structure
  - Process models
    - Business or technical process to be built
  - Functional models
    - Functional hierarchy of the system

Program Structure Partitioning

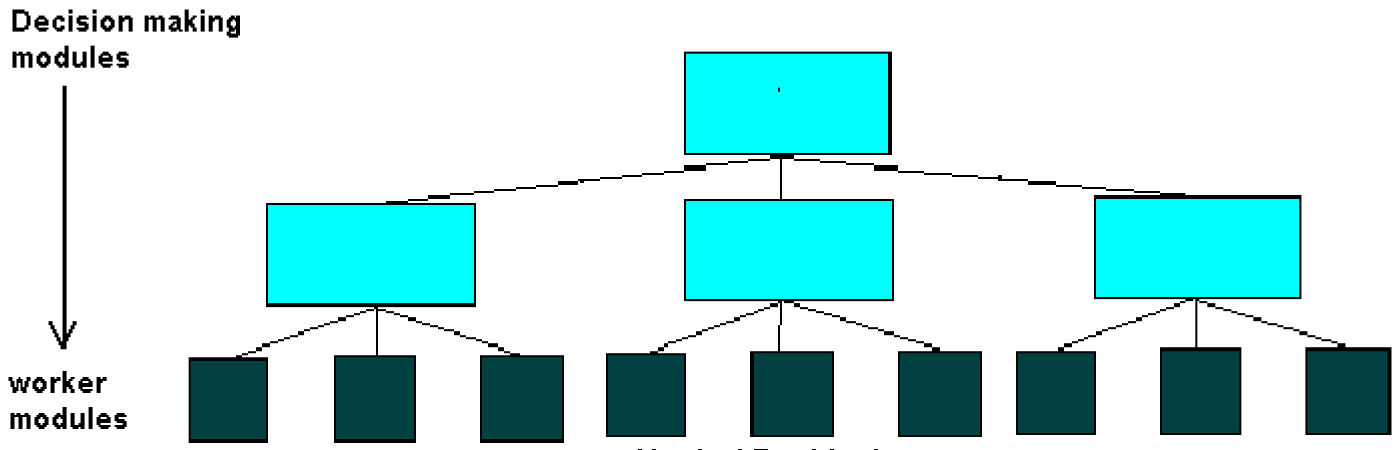
- Horizontal Partitioning
  - Easier to test
  - Easier to maintain (questionable)
  - Propagation of fewer side effects (questionable)
  - Easier to add new features

F1 (Ex: Input) F2 (Process) F3(Output)



Program Structure Partitioning

- Vertical Partitioning
  - Control and work modules are distributed top down
  - Top level modules perform control functions
  - Lower modules perform computations
    - Less susceptible to side effects
    - Also very maintainable



Vertical Partitioning

Information Hiding

- Modules are characterized by design decisions that are hidden from others
- Modules communicate ***only*** through well defined interfaces
- Enforce access constraints to local entities and those visible through interfaces
- Very important for accommodating change and reducing coupling

Module B body

```
s: A.shuttle;
x_coord: float;
…
s := A.get;
A.display(s);
…
x_coord := A.get_x(s);
...
```

Module A specification

```
type shuttle is private;
function get return shuttle;
function get_lat(s) return float;
function get_x(s) return float;
function get_long(s) return float;
…
procedure display(s:shuttle);
…
private
type shuttle is record
   x,y,z: float;
   roll, pitch,yaw: float;
end record;
```

Functional Independence

- Critical in dividing system into independently implementable parts
- Measured by two qualitative criteria
  - Cohesion
    - Relative functional strength of a module
  - Coupling
    - Relative interdependence among modules

Modular Design -- Cohesion

- A cohesive module performs a single task
- Different levels of cohesion
  - Coincidental, logical, temporal, procedural, communications, sequential, functional

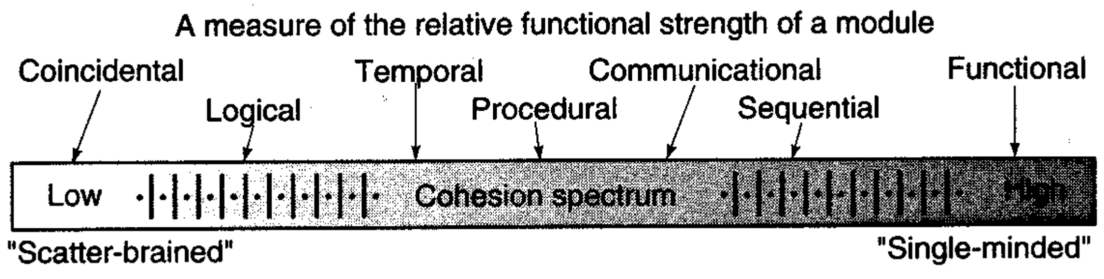


FIGURE 13.6.
Cohesion

Modular Design -- Cohesion

- Coincidental Cohesion
  - Occurs when modules are grouped together for no reason at all
- Logical Cohesion
  - Modules have a logical cohesion, but no actual connection in data and control
- Temporal Cohesion
  - Modules are bound together because they must be used at approximately the same time

Modular Design -- Cohesion

- Communication Cohesion
  - Modules grouped together because they access the same Input/Output devices
- Sequential Cohesion
  - Elements in a module are linked together by the necessity to be activated in a particular order
- Functional Cohesion
  - All elements of a module relate to the performance of a single function

Modular Design -- Coupling

- Coupling describes the interconnection among modules
- Data coupling
  - Occurs when one module passes local data values to another as parameters
- Stamp coupling
  - Occurs when part of a data structure is passed to another module as a parameter

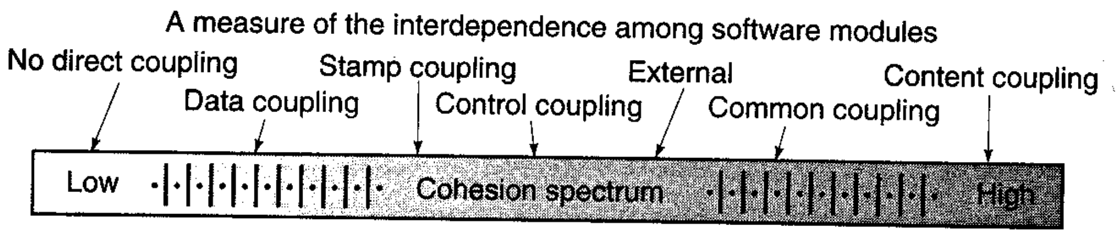


FIGURE 13.7. Coupling

Modular Design -- Coupling

- Control Coupling
  - Occurs when control parameters are passed between modules
- Common Coupling
  - Occurs when multiple modules access common data areas such as Fortran Common or C extern
- Content Coupling
  - Occurs when a module data in another module
- Subclass Coupling
  - The coupling that a class has with its parent class

Examples of Coupling

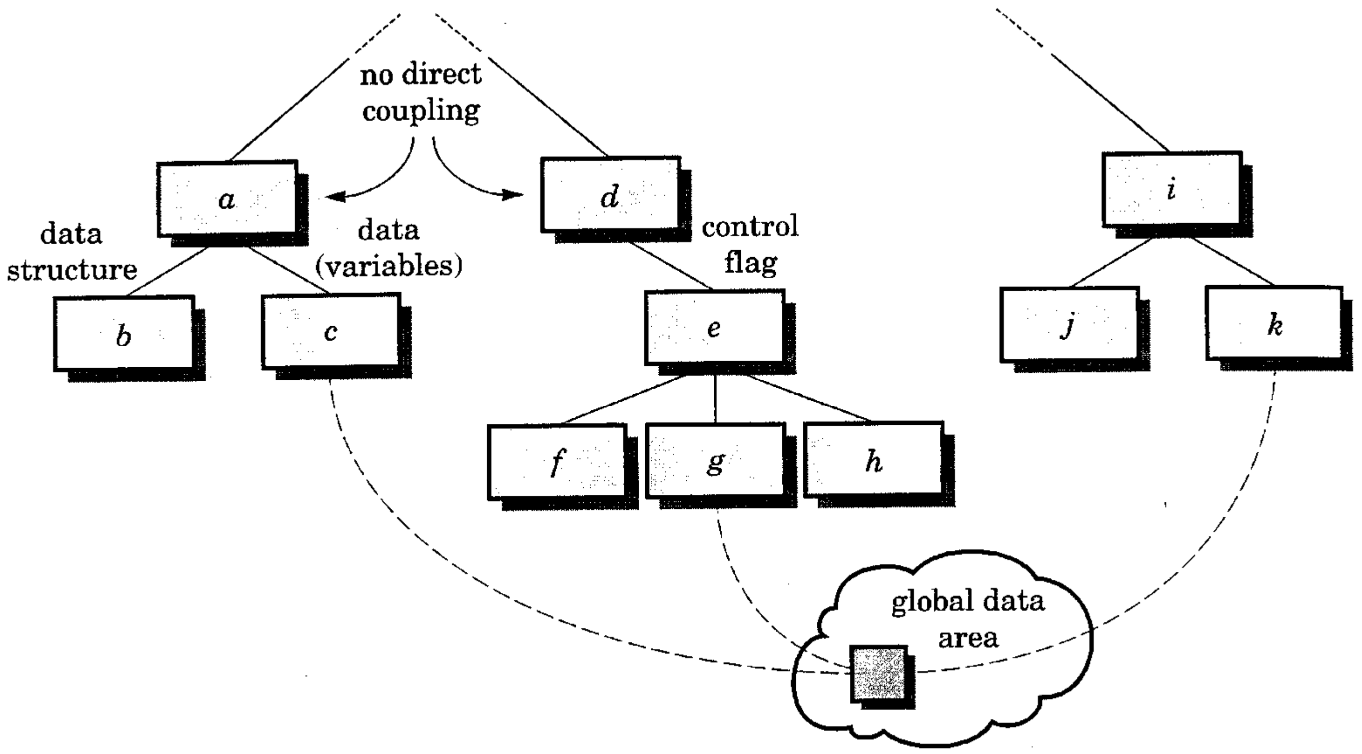


FIGURE 13.8. Types of coupling

Design Heuristics

- Evaluate 1st iteration to reduce coupling & improve cohesion
- Minimize structures with high fan-out; strive for depth
- Keep scope of effect of a module within scope of control of that module
- Evaluate interfaces to reduce complexity and improve consistency

Design Heuristics

- Define modules with predictable function & avoid being overly restrictive
  - Avoid static memory between calls where possible
- Strive for controlled entry -- no jumps into the middle of things
- Package software based on design constraints and portability requirements

Program Structure

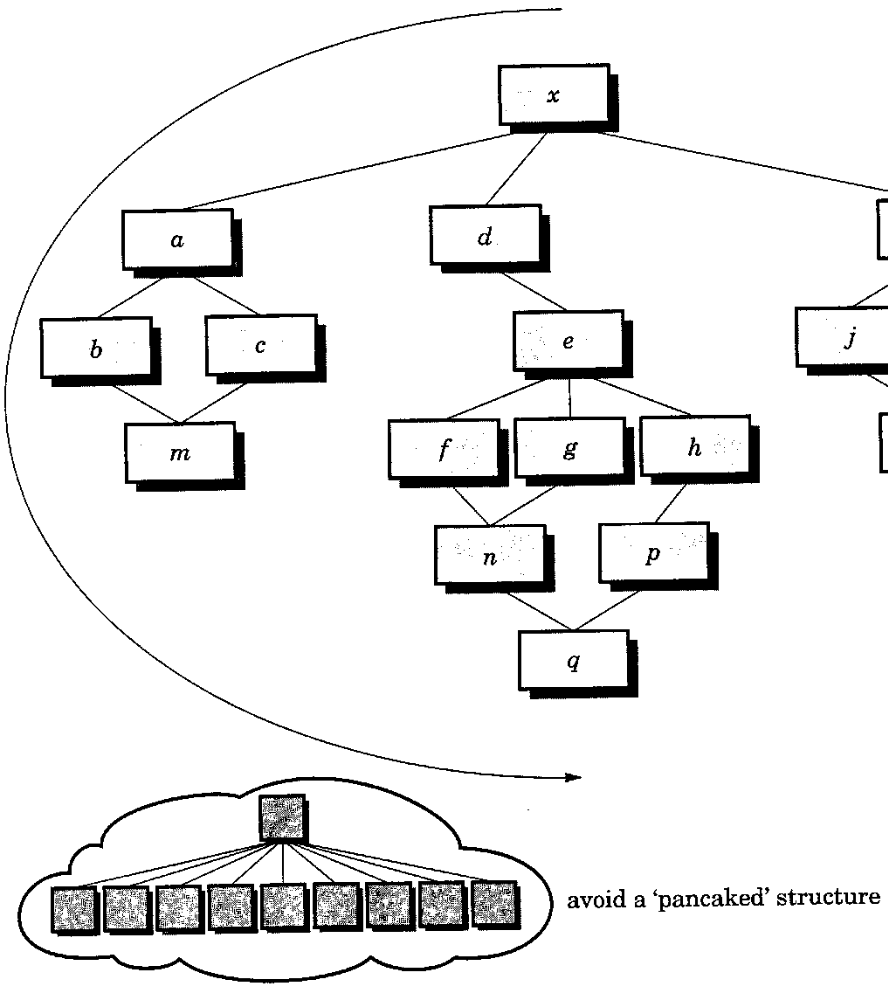
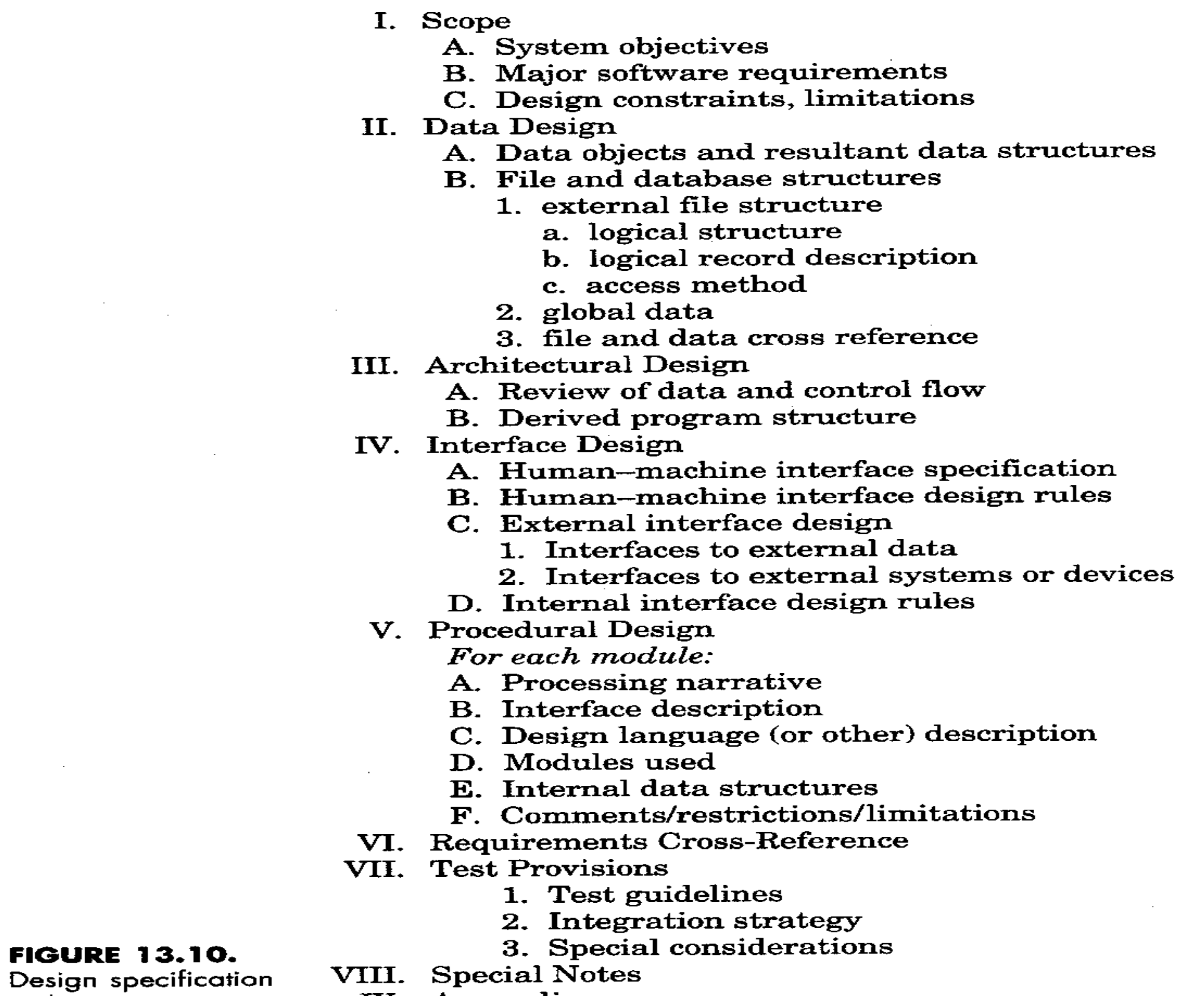


FIGURE 13.9. Program structures

Documentation



I. Scope
  A. System objectives
  B. Major software requirements
  C. Design constraints, limitations
II. Data Design
  A. Data objects and resultant data structures
  B. File and database structures
    1. external file structure
      a. logical structure
      b. logical record description
      c. access method
    2. global data
    3. file and data cross reference
III. Architectural Design
  A. Review of data and control flow
  B. Derived program structure
IV. Interface Design
  A. Human–machine interface specification
  B. Human–machine interface design rules
  C. External interface design
    1. Interfaces to external data
    2. Interfaces to external systems or devices
  D. Internal interface design rules
V. Procedural Design
  For each module:
  A. Processing narrative
  B. Interface description
  C. Design language (or other) description
  D. Modules used
  E. Internal data structures
  F. Comments/restrictions/limitations
VI. Requirements Cross-Reference
VII. Test Provisions
    1. Test guidelines
    2. Integration strategy
    3. Special considerations
VIII. Special Notes

FIGURE 13.10.
Design specification

Summary

- Design is the core of software engineering
- Design concepts provide the basic criteria for design quality
- Modularity, abstraction and refinement enable design simplification
- A design document is an essential part of the process