CMSC 128

Introduction to Software Engineering
Second Semester AY 2007-2008

jachermocilla@uplb.edu.ph

Design Concepts and Principles

- Design is the first step in the development stage
  - "process of applying various techniques and principles for the purpose of defining a device, a process or a system in sufficient detail to permit physical realization"
- Software design is not that mature compared to design in other engineering fields

Software Design and SE

- Sits at the technical kernel of the software engineering process and is applied regardless of the software process model
- Makes use of the analysis models developed during the software requirements analysis and specification phase to develop the *design model*
- Design model: data design, architectural design, interface design, procedural design

Software Design and SE

- Data design (data dictionary+ERD)
  - Transforms information domain model into data structures
- Architectural Design (DFD)
  - Defines the relationship among major structural elements of the program
  - Modular framework of program derived from the interaction of subsystems from analysis model

Software Design and SE

- Interface Design (DFD)
  - Describes how the software communicates within itself, to systems that interoperate with it, and with humans who use it
  - Implies a flow of information
- Procedural Design (CSPEC+PSPEC+STD)
  - Transforms structural elements of the program architecture into a procedural description of software components

Software Design and SE

- Importance of Design
  - **QUALITY**
  - The way to accurately translate the customer requirements into a finished software product or system

Design Process

- An iterative process where requirements are translated into a blueprint for constructing the software
- Expressed at a high level of abstraction
  - Traceable to specific data, functional, and behavioral requirements
- Subsequent refinements are made at each iteration but still traceable to the requirements

Design and Quality

- Formal Technical Reviews/Walkthroughs
- Characteristics of a good design (Goals)
  - Must implement explicit requirement in the analysis model including implicit requirements from customer
  - Must be a readable, understandable guide for programmers, testers, and maintainers
  - Must provide a complete picture of the software: data, functional, and behavioral domains

Design and Quality

- Technical criteria of a good design
  1. Exhibits hierarchical organization that makes intelligent use of control among elements
  2. Modular, logically partitioned into elements that perform specific functions
  3. Contains contain both data and procedural abstractions
  4. Should lead to modules that exhibit independent functional characteristics

Design and Quality

- Technical criteria of a good design

5. Should lead to interfaces that reduce the complexity of connections between modules and with external environment
6. Should be derived using repeatable method that is driven by information obtained during software requirements analysis

Design Principles

- Design process should not suffer from "tunnel vision"
  - Consider alternative approaches
- Design should be traceable to analysis model
  - Must be able to track what requirements have been satisfied by a design model
- Design should not reinvent the wheel
  - Use of "design patterns"

Design Principles

- Design should “minimize intellectual distance” between the software and the problem as it exists in the real world
  - Whenever possible, structure of design should mimic structure of problem domain
- Design should exhibit uniformity and integration
  - It appears that one person developed the entire thing

Design Principles

- Design should be structured to accommodate change
- Design should be structured to degrade gently
  - Should accommodate unusual circumstances
- Design is not coding, coding is not design
  - Level of abstraction, even at procedural design, should be higher than source code

Design Principles

- Design should be assessed for quality as it is being created, not after the fact
- Design should be reviewed to minimize conceptual (semantic) errors
  - Don't miss forest for the trees

Design Concepts

- Answers the following questions
  - What criteria can be used to partition software into individual components?
  - How is function or data structure detail separated from a conceptual representation of the software?
  - Are there uniform criteria that define the technical quality of a software design?

Abstraction

- Permits one to concentrate on a problem at some level of generalization without regard to irrelevant low level details
- Each step in the SE process is a refinement in the level of abstraction of the software solution
- Procedural Abstraction
  - A named sequence of instructions that has a specific and limited function
  - ex. 'open' the door

Abstraction

- Data Abstraction
  - A named collection of data that describes a data object
  - ex. open the “door”
- Control Abstraction
  - Program control mechanism
  - ex. semaphores, for synchronization

Refinement

- Stepwise Refinement
  - Top down strategy
  - Developed by Niklaus Wirth (inventor of the Pascal programming language)
- Architecture of the program is developed by successively refining levels of procedural detail
- Problem decomposed into procedure/functions, then into program statements

Modularity

- Software is divided into separately named and addressable components, called modules, that are integrated to satisfy problem requirements
- Easier to solve a complex problem when you break it into manageable pieces
- Up to when should we “break” a complex problem?
  - Consider the cost of integration

Modularity

- Criteria for evaluating a design method with respect to ability to define a modular system
  - Modular Decomposability
  - Modular Composability
  - Modular Understandability
  - Modular Continuity
  - Modular Protection
- Possible for design to be modular but implementation is monolithic

Software Architecture

- The overall structure of the software and the ways in which that structure provides conceptual integrity for a system
- Hierarchical structure of program components (modules), manner in which these components interact, and the structure of the data that are used by the components
- Major system elements and their interactions

Software Architecture

- Properties that must be specified as part of architectural design
  - Structural Properties
    - Defines components of system (modules, objects, filters) and the manner they interact
  - Extra-functional Properties
    - Address how architecture achieves requirements for performance, capacity, reliability, security, adaptability, and other system characteristics
  - Families of Related Systems
    - Reuse of architectural building blocks

Software Architecture

- Representing Architectural Design
  - Structural Model
  - Framework Models
  - Dynamic Models
  - Process Models
  - Functional Models

Control Hierarchy

- aka Program Structure
  - Organization of program components and implies a hierarchy of controls
- Represented as a tree-like diagram
- Terminologies
  - *Depth* – number of levels of control
  - *Width* – overall span of control
  - *Fan-out* – number of modules directly controlled by a module
  - *Fan-in* – indicates how many modules control a modules

Control Hierarchy

- Terminologies
  - *Superordinate* – a modules that controls another module
  - *Subordinate* – a module that is controlled by another module
  - *Visibility* – set of program components that may be invoked or used as data by a given component
  - *Connectivity* – set of components that are directly invoked or used as data by a component

Structural Partitioning

- Horizontal and vertical partitioning should be done
- Horizontal
  - Defines separate branches for each function
  - Control modules coordinate communication between functions
  - Benefits
    - Easy to extend, test, and maintain
    - Fewer propagation of side-effects
  - Disadvantage: more data needs to be passed

Structural Partitioning

- Vertical
  - Also called factoring, top-down control
  - Top level modules are *control modules*
  - Low level modules perform the processing (*worker modules*)
  - Generally, change in software revolves around input, processing, and output
    - Fewer side effect propagation because changes occur at the worker modules
    - Justifies vertical partitioning

Data Structure

- Representation of the logical relationship among individual elements of data
- As important as program structure, needed at procedural design
- Dictates the organization, methods of access, degrees of associativity, and processing alternatives for information
- Think CMSC 123!

Data Structure

- Scalar
  - Single element of information addressed by an identifier
- Sequential Vector
  - Scalar items organized as a list or contiguous group
- N-dimensional space, Arrays
  - Extended Sequential Vectors

Data Structure

- Abstract Data Types
  - List
  - Stack
  - Queue
  - Tree
  - Graph
  - Set

Software Procedure

- Focus on the processing details of each module individually
- Precise specification of processing
  - Sequence of events
  - Exact decision points
  - Repetitive operations
  - Data organization/structure
- Zoom-in on modules

Information Hiding

- Modules should be “characterized by design decision that (each) hides from all others”
- Modules should be designed so that information contain within a module is inaccessible to other modules that have no need for such information
- Enforces access constraints to both procedural detail within a module and any local data structure

Information Hiding

- Beneficial when modifications are required during testing and maintenance
- Prevents propagation of errors – side effects

Effective Modular Design

- Benefits
  - Reduces complexity
  - Facilitates change
  - Parallel implementation

Functional Independence

- Each module addresses a specific subfunction of requirements and has a simple interface
- Benefits
  - Easier to maintain and test
  - Reduced error propagation
  - Reusability
- Qualitative Criteria: cohesion and coupling

Cohesion

- Each module should do only one thing
- Strive for high cohesion
- Types
  - Coincidentally cohesive
    - Tasks relate to each other loosely
  - Logically cohesive
    - Tasks are related logically, produce output regardless of type
  - Temporal cohesion
    - Tasks must be executed at the same span of time

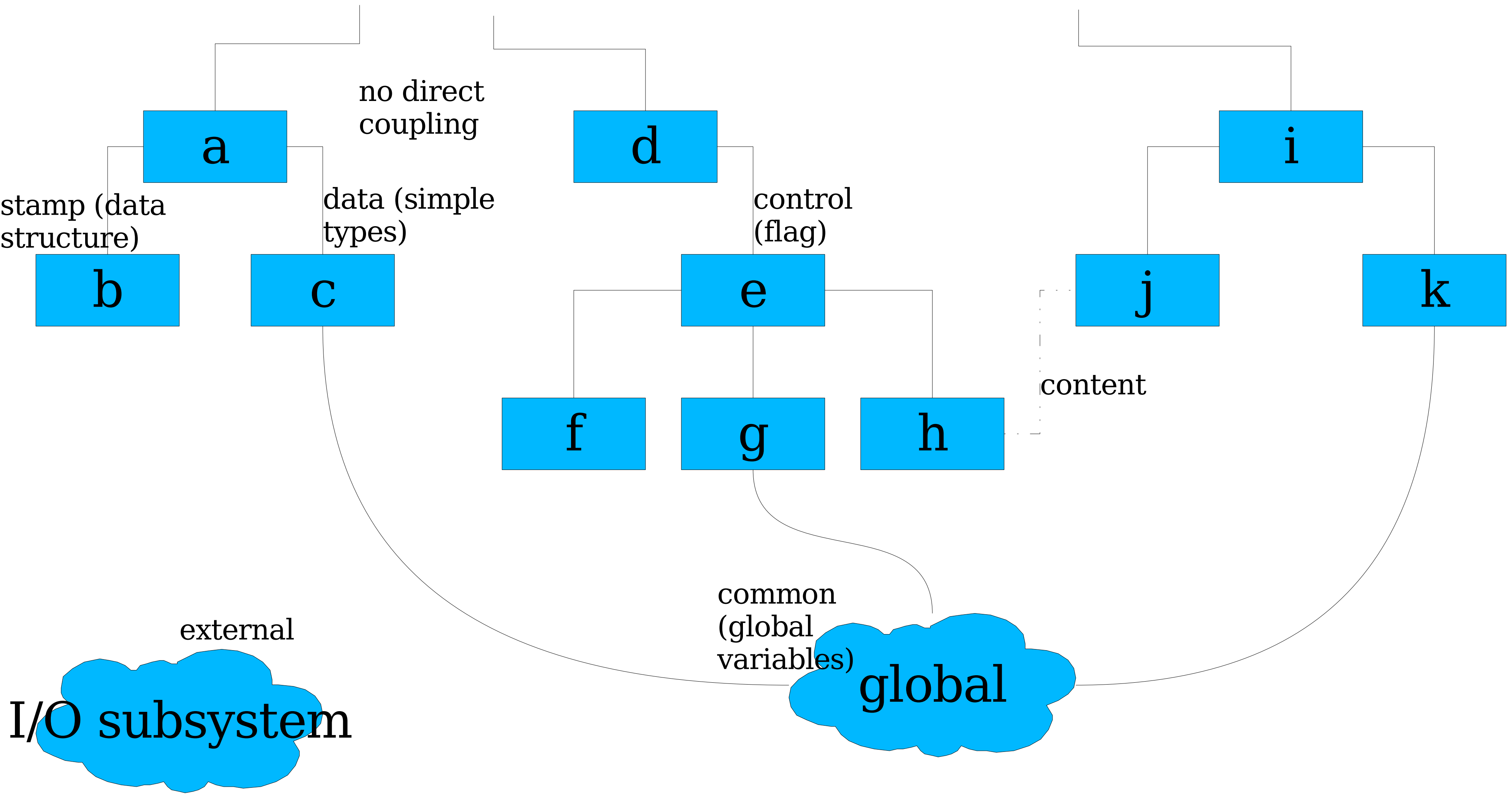
Cohesion

- Types
  - Procedural cohesion
    - Processing elements are related and must be executed in a specific order
  - Communicational cohesion
    - Processing elements concentrate on one area of a data structure

Coupling

- Measure of interconnection among modules
- Depends on interface complexity, point at which entry or reference is made to a module, what data pass across the interface
- In software design, strive for lowest possible coupling
- Coupling occurs because of design decisions

Coupling


no direct coupling
a
d
i
stamp (data structure)
data (simple types)
control (flag)
b
c
e
j
k
f
g
h
content
common (global variables)
external
global
I/O subsystem

Heuristics for Modularity

1. Evaluate the “first iteration” of the program structure to reduce coupling and improve cohesion
   - implode or explode modules
2. Minimize structures with high fan-out; strive for fan-in as depth increases
3. Keep *scope of effect* of module within the *scope of control* of that module
   - scope of effect – modules affected by decision
   - scope of control – subordinate modules

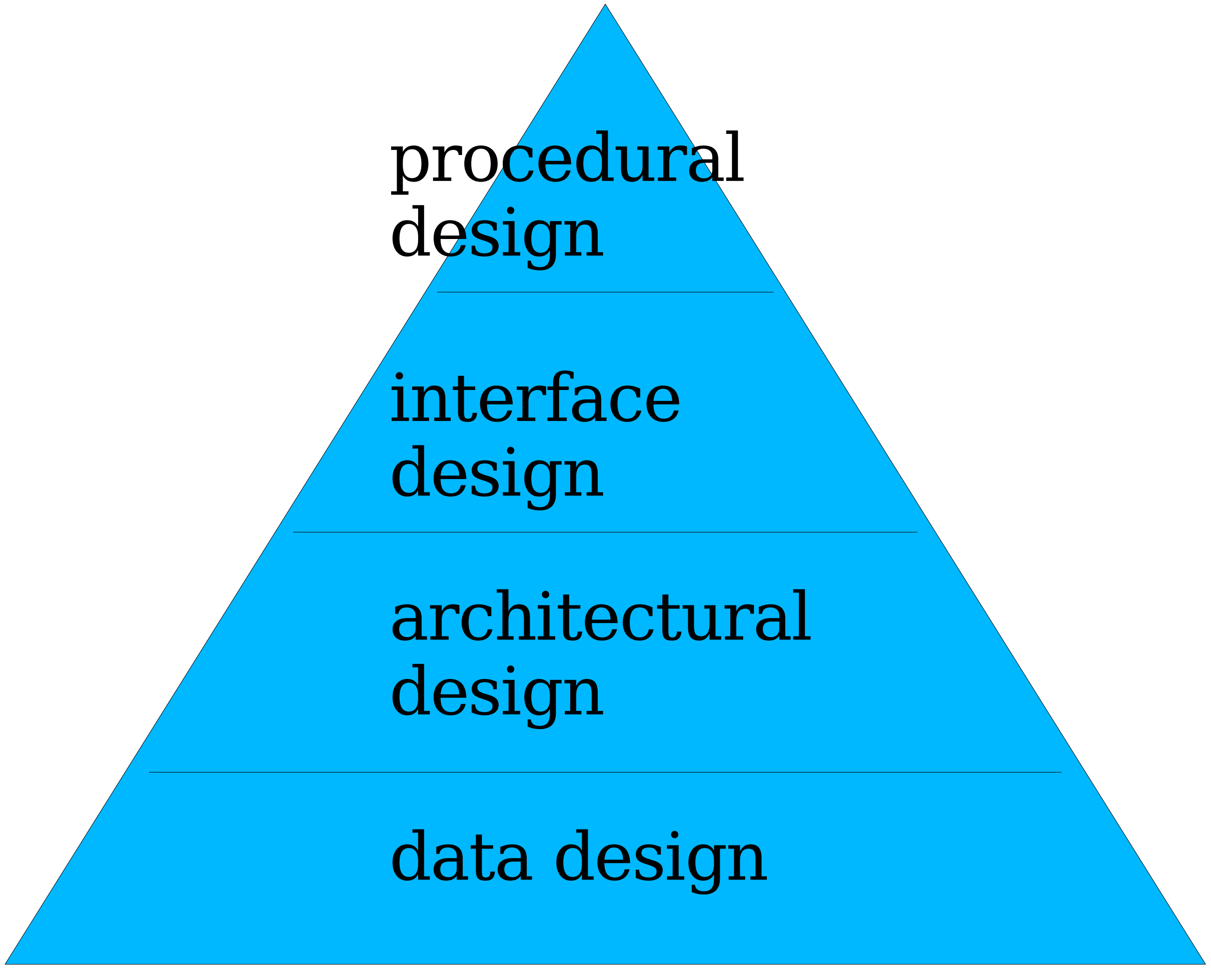
Guidelines for Modularity

4. Evaluate module interfaces to reduce complexity and redundancy and improve consistency
5. Define modules whose function is predictable, but avoid modules that are overly restrictive
6. Strive for “controlled entry” modules, avoiding “pathological connections”
    - avoid branches to other modules in the middle of the module

Guidelines for Modularity

7. Package software based on design constraints and portability requirements

Design Model



Design Documentation

- Software Design Specification
  - I. Scope
  - II. Data Design
  - III. Architectural Design
  - IV. Interface Design
  - V. Procedural Design
  - VI. Requirements Cross-Reference
  - VII. Test Provisions
  - VIII. Special Notes
  - IX. Appendices

Summary

- Design is the technical kernel of software engineering
- Progressive refinements of data structure, program architecture, interfaces, and procedural detail are developed
- Design is used to assess quality
- Modularity (data and program) is important in design
- Design should not be rushed

Reference

- Roger S. Pressman.Software Engineering: A Practitioner's Approach, 4th Ed.McGraw-Hill,1997. Chapter 13