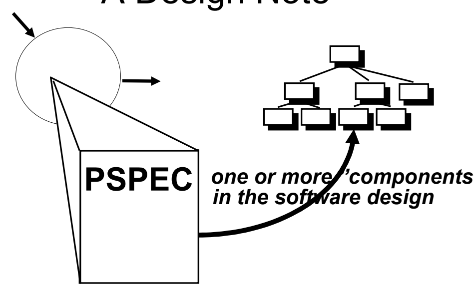
IF2261 Software Engineering

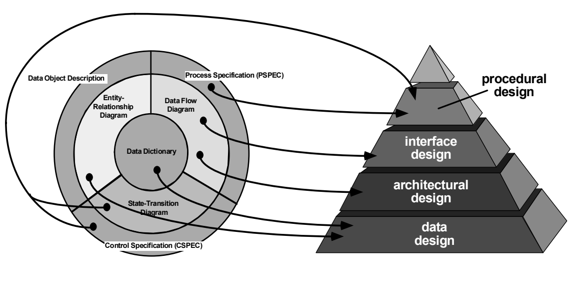
Design Concept and Principles

Program Studi Teknik Informatika
STEI ITB

A Design Note



Analysis to Design



Design Process

- An iterative process through which requirements are translated into a “blueprint” for constructing the S/W
- Throughout the design process, the quality of the evolving design is assessed with a series of formal technical reviews or design walkthroughs
- Guide for evaluation of a good design:
  - The design must implement all of the explicit and implicit requirements
  - The design must be readable
  - The design should provide a complete picture of the software

Evolution of S/W Design

- Development of modular program
- Structural programming
  - Procedural aspect of design definition
- Translation of data flow or data structure into a design definition
- OO design

Design Principles

- The design process should not suffer from “tunnel vision” → should consider alternative approachs
- The design should be traceable to the analysis model
- The design should not reinvent the wheel → use design patterns
- The design should “minimize the intellectual distance” between the S/W and the problem as it exist in the real world
- The design should exhibit uniformity and integration

Design Principles (cont.)

- 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) error

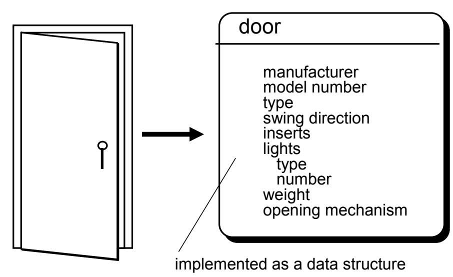
Fundamental Concepts

- **Abstraction** - allows designers to focus on solving a problem without being concerned about irrelevant lower level details (procedural abstraction - named sequence of events, data abstraction - named collection of data objects)
- **Refinement** - process of elaboration where the designer provides successively more detail for each design component
- **Modularity** - the degree to which software can be understood by examining its components independently of one another
- **Software architecture** - overall structure of the software components and the ways in which that structure provides conceptual integrity for a system

Fundamental Concepts (2)

- **Control hierarchy or program structure** - represents the module organization and implies a control hierarchy, but does not represent the procedural aspects of the software (e.g. event sequences)
- **Structural partitioning** - horizontal partitioning defines three partitions (input, data transformations, and output); vertical partitioning (factoring) distributes control in a top-down manner (control decisions in top level modules and processing work in the lower level modules)
- **Data structure** - representation of the logical relationship among individual data elements (requires at least as much attention as algorithm design)
- **Software procedure** - precise specification of processing (event sequences, decision points, repetitive operations, data organization/structure)
- **Information hiding** - information (data and procedure) contained within a module is inaccessible to modules that have no need for such information

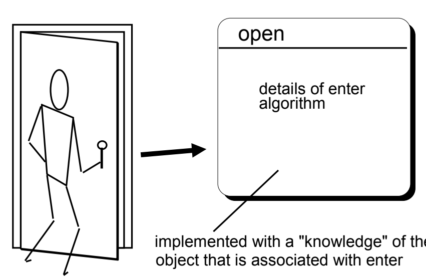
Data Abstraction



Data Design

- refine data objects and develop a set of data abstractions
- implement data object attributes as one or more data structures
- review data structures to ensure that appropriate relationships have been established
- simplify data structures as required

Procedural Abstraction



Stepwise Refinement

open

walk to door;
reach for knob;

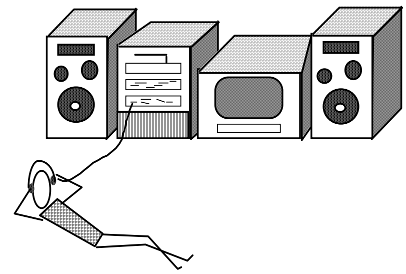
open door;

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

Modular Design

easier to build, easier to change, easier to fix .



Functional Independence

COHESION - the degree to which a module performs one and only one function.

COUPLING - the degree to which a module is "connected" to other modules in the system.

Cohesion and Coupling Spectrum

low

- Coincidental
- Utility
- Temporal
- Procedural
- Sequential
- Communicational
- Layer
- Functional

high

low

- No direct coupling
- External coupling
- Inclusion or import coupling
- Type use coupling
- Routine call coupling
- Data coupling
- Stamp coupling
- Control coupling
- Common coupling
- Content coupling

high

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

Why Architecture?

The architecture is not the operational software. Rather, it is a representation that enables a software engineer to:

(1) analyze the effectiveness of the design in meeting its stated requirements,

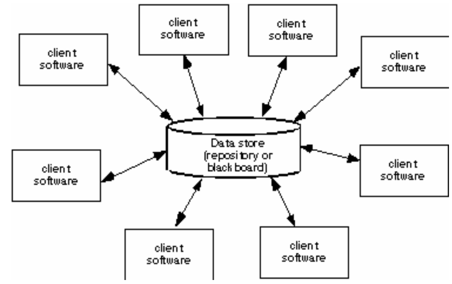
(2) consider architectural alternatives at a stage when making design changes is still relatively easy, and

(3) reduce the risks associated with the construction of the software.

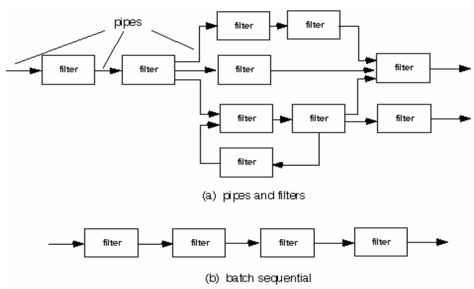
Architectural Styles

- Data-centered architectures
- Data flow architectures
- Call and return architectures
- Object-oriented architectures
- Layered architectures

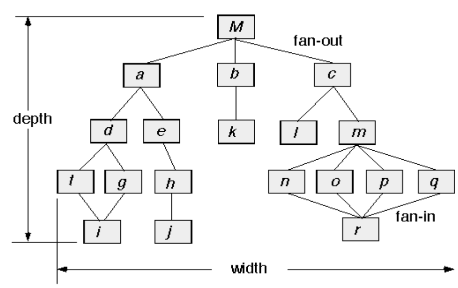
Data-Centered Architecture



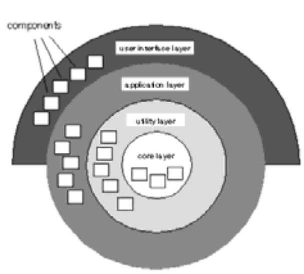
Data Flow Architecture



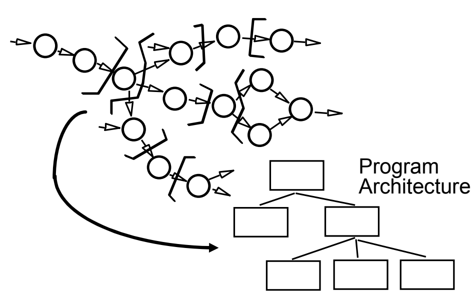
Call and Return Architecture



Layered Architecture

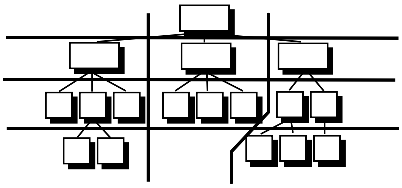


Deriving Program Architecture



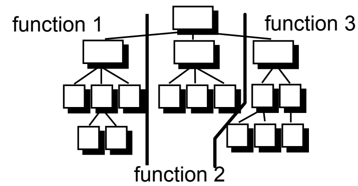
Partitioning the Architecture

- "horizontal" and "vertical" partitioning are required



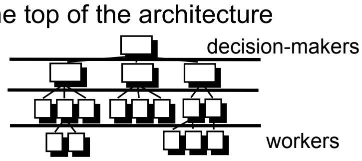
Horizontal Partitioning

- define separate branches of the module hierarchy for each major function
- use control modules to coordinate communication between functions



Vertical Partitioning: Factoring

- design so that decision making and work are stratified
- decision making modules should reside at the top of the architecture



Why Partitioned Architecture?

- results in software that is easier to test
- leads to software that is easier to maintain
- results in propagation of fewer side effects
- results in software that is easier to extend