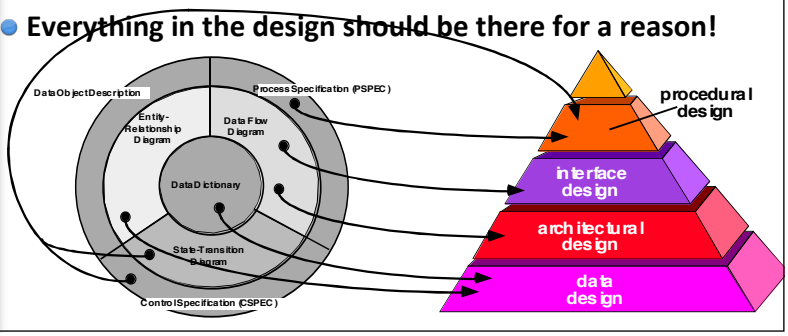
Software Design and Software Engineering

- Design follows analysis and conforms to requirements
- Design Model
  - Data/Class Design
  - Architectural Design
  - Interface Design
  - Component Level Design
- Everything in the design should be there for a reason!



Traceability

- The data/class design is derived from the data dictionary and the data/class model
- The architectural design is derived from the functional model
- The interface design comes from the data flow and control flow
- The component level design is derived from the PSPEC, CSPEC, and the behavioral model
- All design work products must be traceable to requirements
- All design work products must be reviewed for quality.

Design Process

- Software design is an iterative process traceable to requirements analysis process
- Many software projects iterate through the analysis and design phases several times
- Pure separation of analysis and design may not always be possible or desirable

> "There are two ways of constructing a software design. One way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are obviously no deficiencies. The first method is far more difficult."
> - C.A.R Hoare

Generic Design Guidelines

- Design should exhibit an architectural structure that:
  - Has been created using recognizable patterns
  - Has components with good design characteristics
  - Can be implemented in evolutionary fashion
- Design should be modular
  - Logically partitioned into elements performing specific functions
- Design should contain distinct representations of each model
- Design should lead to data structures that are appropriate
- Design components should exhibit independent functional characteristics
- Design should lead to interfaces that reduce complexity
- Design should be derived from requirements systematically

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" 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.

Design Principles (cont.)

- 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.

Design Concepts

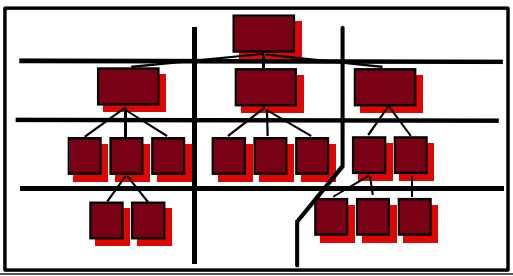
- Abstraction
- Software architecture
- Patterns
- Separation of concerns
- Modularity
- Information Hiding
- Functional Independence
- Refinement
- Aspects
- Refactoring

Design concepts

- Abstraction
  - Concentrate at some level w/o considering irrelevant details
    - e.g., procedural abstraction is a sequence of instructions with specific and limited function, but details are suppressed
    - e.g., data abstraction is a single name for a collection of data
  - Work with concepts/terms familiar in the problem environment
- Modularity
  - Division into separately named and addressable components
  - Allows complexity to be managed ("divide and conquer")
- Software architecture
  - Structural properties (modules, objects, etc.)
  - Extra-functional properties (performance, capacity, etc.)
  - Families of related systems (design patterns)
- Patterns
  - "A named nugget of insight which conveys the essence of a proven solution to a recurring problem within a certain context amidst competing concerns"

Functional Independence

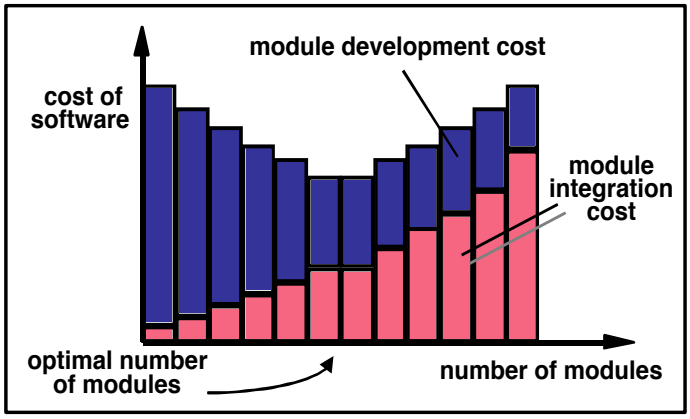
- Involves separation of concerns
  - Develop modules with "single-minded" function and an "aversion" to excessive interaction with other modules
- Often involves structural partitioning
  - Horizontal partitioning (division of functionality): separate branches of control hierarchy
  - Vertical partitioning (factoring): distributing work to "worker" modules



Effective Modular Design

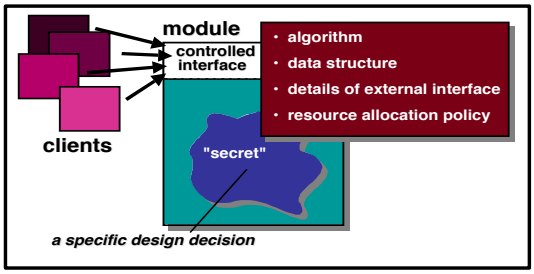
- Sizing modules – what's inside it? How big is it?
- Coupling – degree to which a module is "connected" to other modules in a system
- Cohesion – degree to which a module performs one and only one function
- Seek high cohesion and low coupling
  - functional independence is the goal for each module
  - More likely when modules have single purposes (high cohesion) and rely on their own resources for data and control information (low coupling).

Sizing of Modules



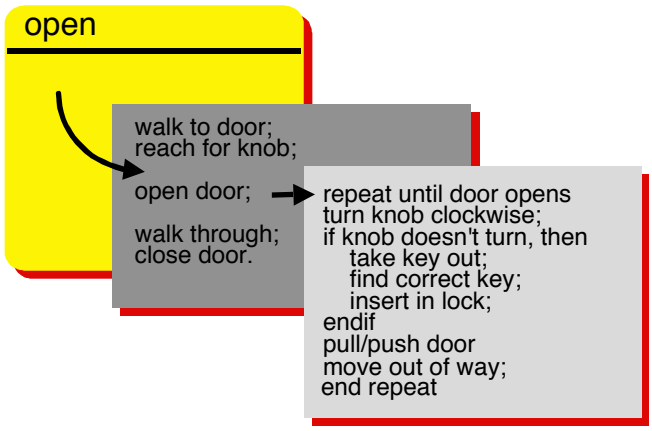
Information Hiding

- Reduces the likelihood of "side effects"
- Limits global impact of local design decisions
- Emphasizes communication between modules through controlled interfaces
- Discourages the use of global data
- Leads to encapsulation — an attribute of high quality design



Refinement

- Successively decomposing or refining specifications in stepwise fashion



Data Design

- Create a model of information represented at a high level of abstraction (the customer's view)
  - 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
- Data structure design is at least as important as algorithm design

Data Design – Component Level

- Apply systematic design principles to data
- Identify all data structures and the operations to be performed on each one
- Establish a data dictionary
  - Use it to define both data and program design.
- Defer low level data design decisions until later.
- The representation of data structure should be known (visible) only to modules that must make direct use of the data contained within the structure.
- Develop a library of useful data structures and the operations that may be applied to them
- A software design and programming language should support specification/realization of abstract data types.

Software Architecture

- Framework of:
  - System structures comprising the software components,
  - Their properties
  - Relationships among these components
- Architecture is a model enabling a software engineer to:
  - Analyze effectiveness of the design in meeting its requirements,
  - Consider architectural alternatives when it's easy to make design changes
  - Reduce construction risks
- Use control and data issues as the basis for assessing quality of the initial architectural framework before detailed analyses of the proposed software architecture

Architectural Design

- Goal: allow software engineer to view/evaluate the system as a whole before detailed design
- Architectural design encompasses both the data architecture and the program structure layers of the design model
- Thoughts on architecture:
  - "Provide a designer with the concentrated wisdom of many preceding designers faced with similar problems." -- Klein and Kazman 1999
  - "You can use an eraser on the drafting table or a sledge hammer on the construction site" -- Frank Lloyd Wright

Architectural Style Elements

- A set of components (e.g., a database, computational modules) that perform a function required by a system
- A set of connectors that enable
  - communication
  - coordination
  - cooperation

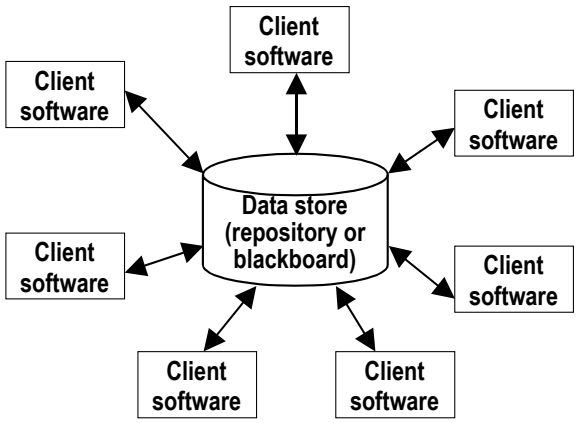
  among components
- Constraints that define how components can be integrated to form the system
- Semantic models to understand overall properties of a system by analyzing known properties of its constituent parts.

Different Architectural Styles

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

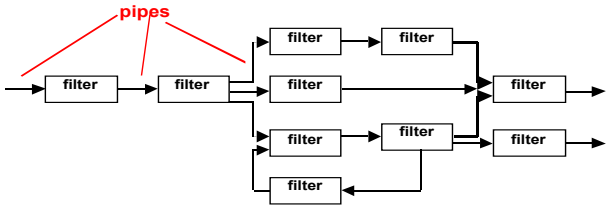
Data-centered architecture

- Common data repository
- Relatively independent clients access the repository

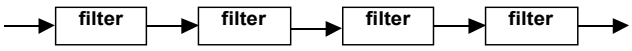


Data Flow Architectures

- Relatively simple input-output behavior
- Relatively complicated or distributed processing behavior
- Pipe-and-filter data flow architecture

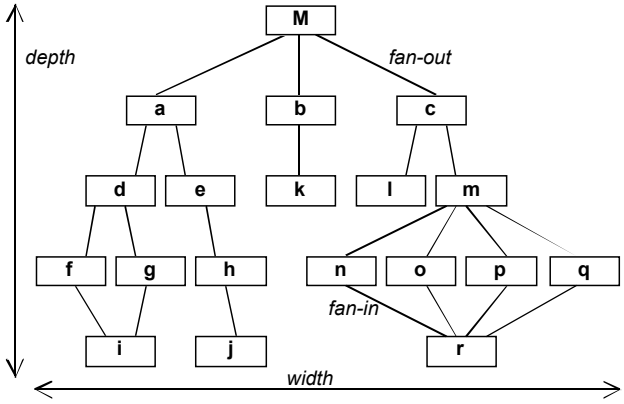


- Sequential batch data flow architecture



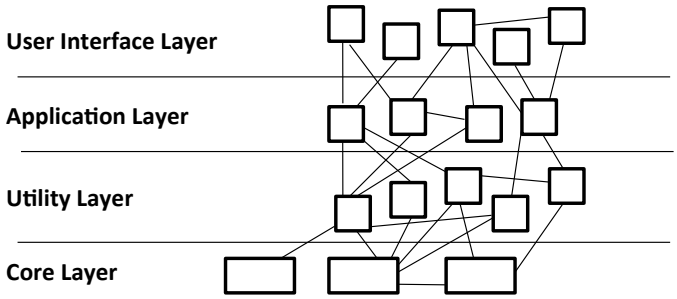
Call-and-return architecture

- Input-process-output model
- Procedural partitioning is central focus



Layered architecture

- Modifiability and adaptability are central focus
- Each layer is a "client" of the layer beneath it
- No process "skips" through a layer to the next
- Example of a layered architecture:



Analyzing Architectural Styles

- Describe the candidate architectural styles/patterns chosen to address scenarios and requirements:
  - Module view
  - Process view
  - Data flow view
- Perform trade-off analyses of alternative architectural designs for the same software system
- Evaluate quality attributes by considering each attribute in isolation.
- Identify sensitivity of quality attributes to various architectural attributes for each architectural style.
- Critique candidate architectures using the sensitivity analysis

Questions to consider

1. **Are all design concepts (e.g., abstraction, modularity) equally important? How would you rank them? Why?**
2. **Why are low coupling and high cohesion in tension with one another?**
3. **Why might it be easier to design 2*N* modules each of size *S* rather than *N* modules each of size 2*S*?**
4. **What are the main components of an architectural style? Pick a system and identify each of those main components.**
5. **Choose a system you know about, identify the best architectural style and explain why it's appropriate.**
6. **Are the architectural styles shown here completely different from each other? How do they compare?**