CH05: Designing the System

- To design a system is to determine a set of components and inter-component interfaces that satisfy a specified set of requirements.
- * What is Design?
- * Decomposition and Modularity
- * Architectural Styles and Strategies

TECH

Computer Science

Designing the System (continue)

- Issues in Design Creation
- Characteristics of Good Design
- Techniques for Improving Design
- Design Evaluation and Validation
- Documenting the Design

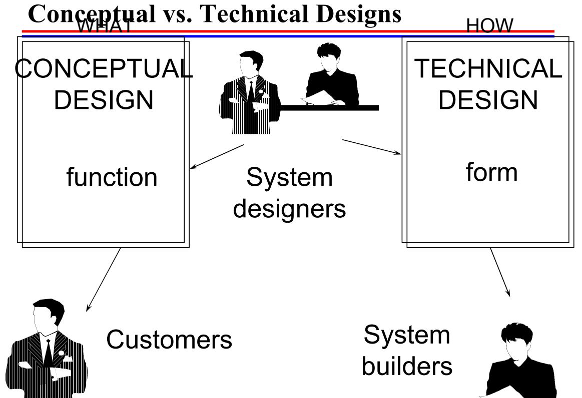
What is Design?

- Design (process) is the creative process of transforming the problem into a solution. (software engineer's definition)
- Design (product) is the description of a solution.
- What is a solution?
  - **We declare something to be a solution to a problem if it satisfies all the requirements in the specification**

Conceptual and Technical Designs

- We produce **conceptual design** that tells the **customer** exactly what the system will do. (The What of the solution)
- We produce **technical design** that allows system builders (**developers**) to understand the actual hardware and software needed to solve the customer's problem. (The How of the solution)
- Merged the two into one document.

Conceptual vs. Technical Designs



Good Conceptual Design

- written in customer's language
- explaining the observable external characteristics of the system
- contains no technical jargon (If it does, define it.)
- describes the functions of the system
- is independent of implementation
- is linked to the requirements documents

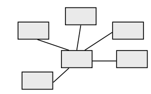
Good technical design

- describes of major hardware components and their functions
- shows hierarchy (organization) and functions of the software components
- shows data structures and data flow
- shows interfaces

Conception vs technical design document

"The user will be able to route messages to any other user on any other network computer."

CONCEPTUAL DESIGN



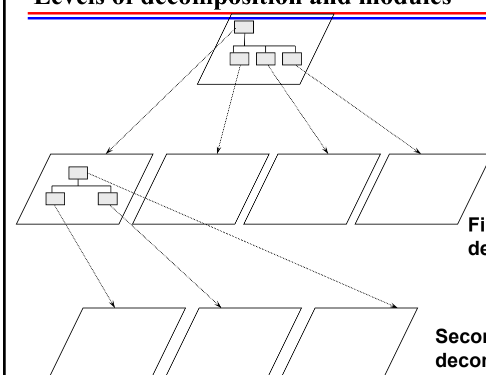
Network topology
Protocol used
Prescribed bps rat
...

TECHNICAL DESIGN

Decomposition and Modularity

- Decomposition
  - **starting with a high-level depiction of the system's key elements**
  - **creating lower-level looks at how the system's features and functions will fit together**
- Modularity
  - **the results of decomposition form composite parts called modules or components.**
  - **Modular: when each activity of the system is performed by exactly one component.**

Levels of decomposition and modules



Methods for Decomposition

- Modular decompositions (assigning functions to components)
- Data-oriented decomposition (external data structures)
- Event-oriented decomposition (events that the system must handle)
- Outside-in design (user's inputs, to processing, to output)
- Object-oriented design (classes of objects and their inter-relationships)

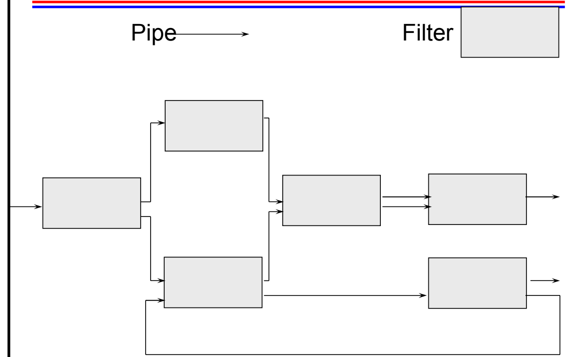
Architectural Styles and Strategies (Three design levels)

- Architecture: requirements -> system modules
- Code Design: modules -> algorithms and data structures
- Executable Design: algorithms (codes) -> memory allocation, execution time, code optimizations, ...

Architectural Styles

- Piles and Filters,
- Object-oriented Design
- Implicit Invocation,
- Layering, Repositories
- Interpreters, Process Control
- Distributed Systems, Client-Server, domain-specific architectures

Piles and Filters



Piles and Filters: properties

- easy to understand in terms of input->transformation ->output
- filters are independence
- filters can be reused easily
- easy to add or remove filters
- easy to analyze throughput
- allow concurrent execution of filters

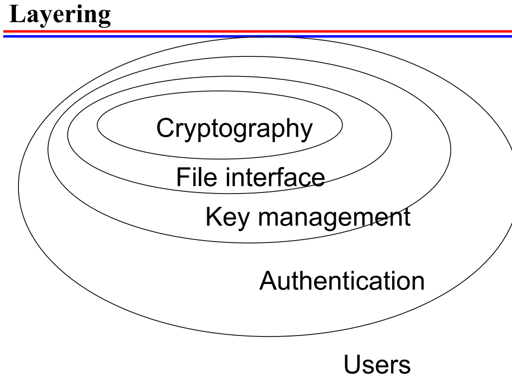
Object-oriented Design

- decompose the underlying problem into a collection of interacting agents
- for agents to interact, one agent must know the identity (interface) of the interacted agents
- interdependent on one another
- changing the identity (interface) of an agent requires all other interacted agents to change

Implicit Invocation (event-driven)

- An agent broadcasts an event
- some other agents decide to work on the event
- Problem: when the agent broadcasts an event, it does not know which other agent (if any) will work on the event
- Solution: agents can be arranged in levels, the closest levels will first pick up the event, then the next levels, and so on, to a default level pick up what is left.

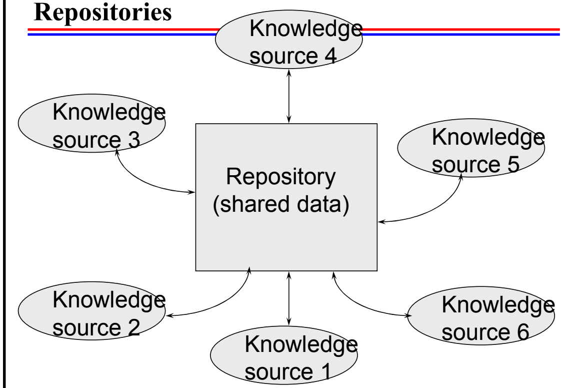
Layering



Layering Architecture

- case 1: one layer has access only to adjacent layers
- case 2: one layer has access to some or all other layers
- case 1 is used in most layering architectures
- layer represents levels of abstraction

Repositories



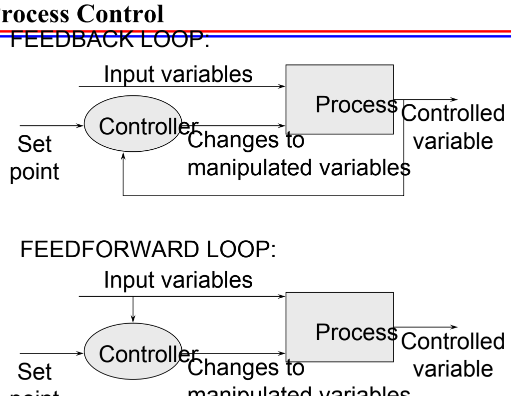
Repositories Architectures

- a central data store (blackboard)
- knowledge sources (agents)
- changing data (state) in central data store will trigger agents to react (respond)

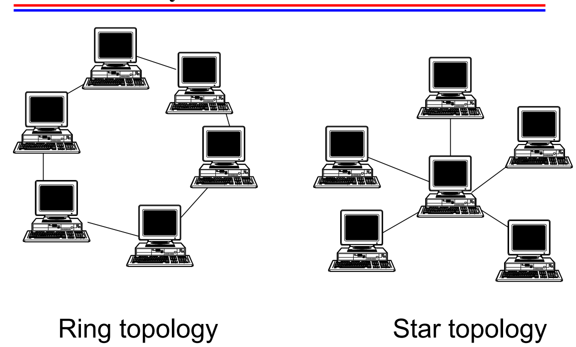
Interpreters

- An interpreter takes a string of characters, and converts it into actual code that is then executed.
- Translation or the conversion takes be place in sequence of steps

Process Control

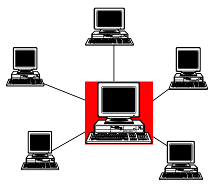


Distributed Systems



Client-Server

- A client request an service
- A server responds to the request



domain-specific architectures //

- takes advantage of particular application domain, such as avionics or automobile manufacturing.
- match software architectures with physical architectures or application domain processes.

Issues in Design Creation

- issues that must be addressed by designers, when selecting an appropriate style, or when creating the design details:
  - **Modularity and Levels of Abstraction**
  - **Collaborative Design**
  - **Designing the User Interface**
  - **Concurrency**
  - **Design Patterns and Reuse**

Modularity and Levels of Abstraction

- In a modular design, the components have clearly defined inputs and outputs, and each component has a clearly stated purpose.
- Levels of abstraction: the components at one level refine those in the level above.

Collaborative Design

- On most projects, the design is not created by one person.
- A team works collaboratively to produce a design, often by assigning different parts of the design to different people.
- Collaborative groups may locate all over the world.
- Communication problems in language as well as missing personal touch.

Designing the User Interface

- an user interface should address several key elements:
  - **metaphors: fundamental images and concepts.**
  - **a method model: the organization and representation of information**
  - **the navigation of rules for the model: how to move among, and spacial model**
  - **look: appearance conveys information to the users**
  - **feel: the interaction techniques that provide an appealing experience for the user**

Issues of designing the user interface

- Cultural Issues
- User Preferences
- Guidelines for Determining User-interface Characteristics

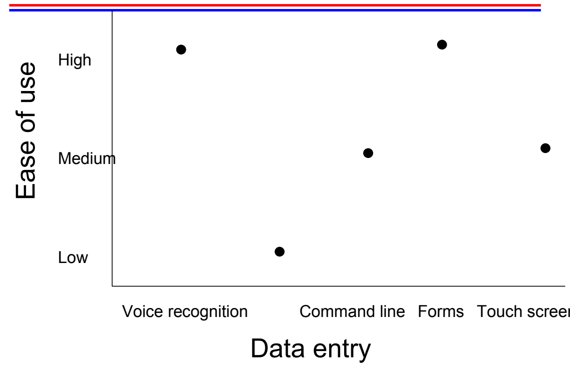
Cultural Issues

- Which color to use? Purple?
  - **In England, purple represents royalty**
  - **In Japan, purple signifies dignity and nobility**
  - **In Greece, however, purple symbolized death and evil.**
- Two steps to make our systems multi-cultural
  - **(1) eliminate specific cultural reference or biases**
  - **(2) tailors (1) for the cultures that will be using the software**

User Preferences

- She like it. He may not.
- No universal interface can be applied to anyone.
- prototyping with the particular target audience
- allowing customizing the user interface, e.g. Microsoft Words vs. WordPerfect.

Guidelines for Determining User-interface Characteristics //



Concurrency

- actions must take place concurrently
- problem: One of the biggest problems with concurrent system is the need to assure the consistency of the data shared among components that execute at the same time.
- Solution: Synchronization
  - **Mutual exclusion**
  - **Monitors**
  - **Guardians**

Mutual exclusion

- it makes sure that when one process is accessing a data element, no other process can affect that element.
- tests and locks: **if** an operation tests the value of the state of an object,
  then that object should be locked
  so that the state does not change between the time the test is done and the time an action is taken based on the value produced by the test.

Monitors

- A monitor is component that controls the mutual exclusion of a particular resource.
- Anyone want to access the resource, it has to go through the monitor
- the monitor allows one access to the resource at a time and suspend others that are also trying to access the resource.

Guardians

- A guardian is a task that is always running: its only purpose is to control access to an encapsulated resource.
- Others interact with the guardian, and does not interact directly with the encapsulated resource.

Design Patterns and Reuse

- We want to take advantage of the commonality among systems, so that we need not develop each "from scratch".
- to identify the commonalities is to look for design patterns.

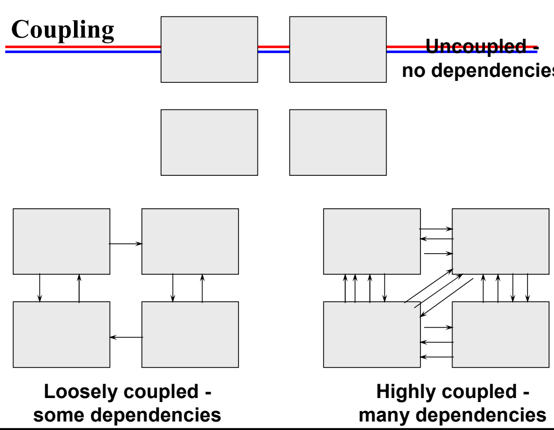
Characteristics of Good Design

- Looking in more detail at attributes that reflect design quality
  - **Component Independence**
  - **Exception Identification and Handling**
  - **Fault Prevention and Fault Tolerance**

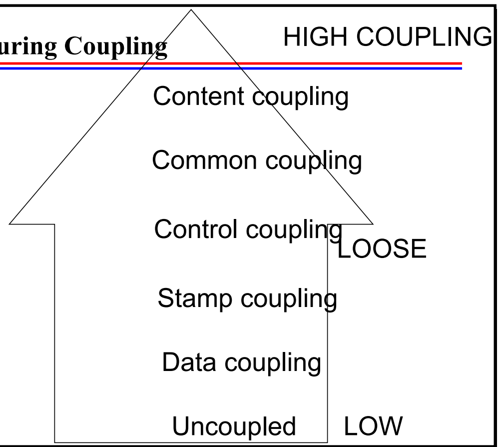
Component Independence

- We strive in most designs to make the components independent of one another.
- Coupling: between component
- Cohesion: within component

Coupling



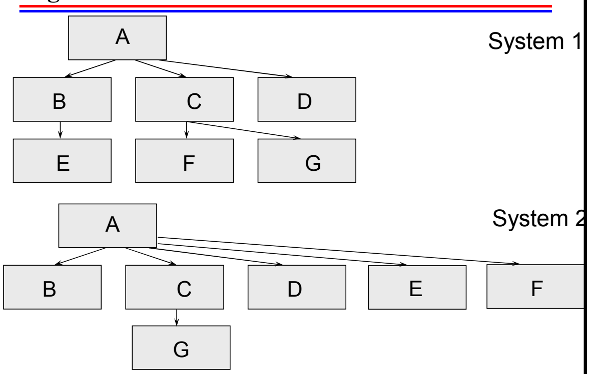
Measuring Coupling



Defining Coupling

- Content coupling: one component modifies the content of another
- Common coupling: sharing common data store
- Control coupling: one component passes parameters to control the activity of another component
- Stamp coupling: one pass data structure to another
- Data coupling: one pass data to another.

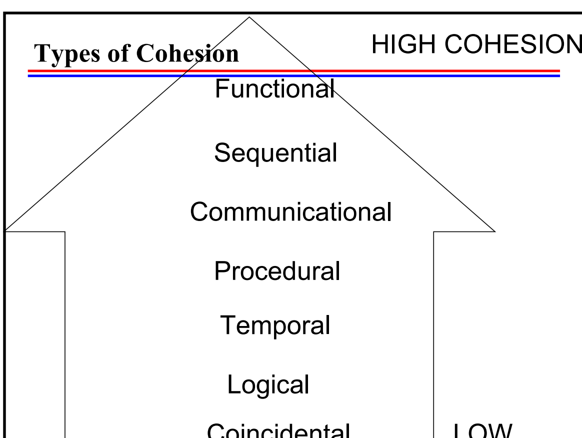
Control Issues: creating components with high fan-in and low fan-out



Cohesion

- refers to the internal "glue" with which a component is contracted.
- The more cohesive a component, the more related are the internal parts of the component to each other and to its overall purpose.

Types of Cohesion



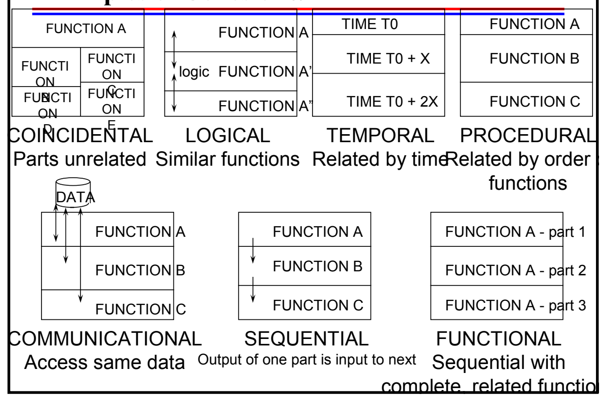
Defining Types of Cohesion

- **Coincidental**: parts are unrelated to one another
- **Logical**: several logically related functions or data elements are placed in the same component
- **temporal**: functions are related only by the timing involved

More Definitions for Types of Cohesion

- **Procedural**: grouping to ensure functions performing in certain order
- **Communicational**: grouping functions sharing same data set
- **Sequential**: output from one part is the input to the next part
- **Functional**: every part is essential to the performance of a single function

Examples of Cohesion //



Exception Identification and Handling

- Design defensively, trying to anticipate situations that might lead to system problems
- Exception Include
  - **failure to provide a service**
  - **providing the wrong service or data**
  - **corrupting data**
- Exception Handling
  - **Retrying, Correct, Report**

Fault Prevention and Fault Tolerance

- try to anticipate faults and handle them in ways that minimize disruption and maximize safety
- Fault: When a human makes a mistake, the human error results in a fault in some software product.
- Failure: is the departure of a system from its required behavior.
- Good design characteristic: to prevents or tolerates faults to become failure.

Techniques for Fault Prevention

- Active Fault Detection
- Fault Correction
- Fault Tolerance

Active Fault Detection

- periodically check symptoms of faults, or try to anticipate when failures will occur
- redundancy: the results of two or more processes are compared to determine if they are the same.
- e.g. In space shuttle, seven computers vote to determine the next operation

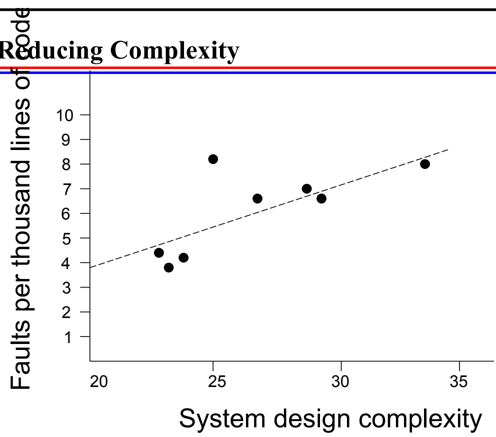
Fault Correction and Tolerance

- Fault Correction
  - fixes the damage done by the fault
  - changing the system to eliminate the fault
- Fault Tolerance
  - isolation of damage caused by a fault
  - prevent fault to become failure

Techniques for Improving Design

- Reducing Complexity
- Design by Contract
- Prototyping Design
- Fault-tree Analysis

Reducing Complexity



Design by Contract

- components whose interaction is based on a precisely defined specification of what each component is supposed to do
- their is contract between two components to perform a task
- each contract cover mutual obligations (preconditions), benefits (post-conditions), and consistency constraints (invariants).

Prototyping Design

- “Brooks (1975) recommends building a system, throwing it away, and building it again, so that second system will profit from our learning as we discover mistakes made in the process of building the first.”
- Try it out.

Fault-tree Analysis

- helping us to decompose design and look for situations that might lead to failure
- fault tress that display the logical path from effect to cause

Design Evaluation and Validation

- Validation: making sure that the design satisfies all requirements specified by the customer
- Verification: ensuring that the characteristics of a good design are incorporated

Techniques for performing validation and Verifications

- **Mathematical Validation**
- **Measuring Design Quality**
- **Comparing Designs**
  - One Specification, Many Designs
  - Comparison Tables
- **Design Reviews**
- **Critical Design Review**
- **Program Design Review**
- **Value of Design Reviews**

Mathematical Validation

- proves that the design is correct
- Demonstrating:
  - **If the set of inputs is formulated correctly, it is transformed properly into set of expected outputs.**
  - **The process terminates without failure.**

Measuring Design Quality

- measuring high-level design, including cohesion and coupling
- measuring complexity within each component and complexity of relationships among components

Comparing Designs

- One Specification, Many Designs
  - **generate several designs for the same specification based on different architectural styles**
  - **deciding which design is best suited for the system's purpose**
- Comparison Tables
  - **Easy to change algorithm**
  - **Easy to change data representation**
  - **Easy to change function**
  - **Good performance and Easy to reuse**

Program Design Review

- After program designs are completed, but before coding begins, the program designers resent their plans to a team of other designers, analysts, and programmers for comment and suggestions
- Design Reviews
  - **Moderator: leads the discussion, and making sure the review moves forward**
  - **Recorder: record the issues that arise and action items.**

Value of Design Reviews

- The sooner we find a problem, the fewer places we have to look to find its cause and fix it.
- Ask all the questions to insure the design cover everything!

Documenting the Design

- An important product of the design process is a set of document that describe the system to be build.
- Cross reference design to requirements
- the solution to the problem