CSCI 5828: Foundations of Software Engineering

Lecture 20, 21, and : Software Design

Slides created by Pfleeger and Atlee for the SE textbook

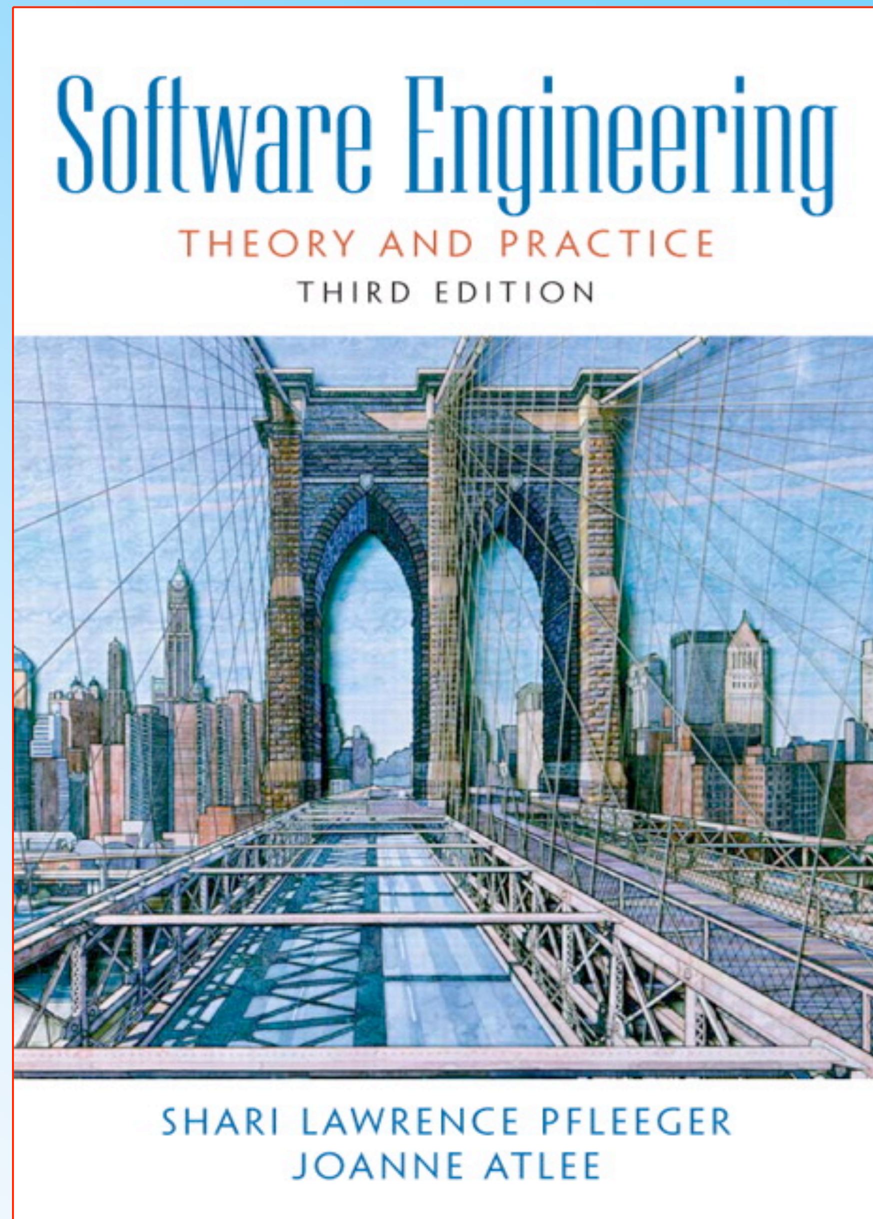
Some modifications to the original slides have been made by Ken Anderson for clarity of presentation

03/20/2008 — 04/01/2008 — 04/08/2008

Chapter 5

Designing the System

ISBN 0-13-146913-4
Prentice-Hall, 2006



Copyright 2006 Pearson/Prentice Hall. All rights reserved.

Contents

5.1 What Is Design?
5.2 Decomposition and Modularity
5.3 Architectural Styles and Strategies
5.4 Issues in Design Creation
5.5 Characteristic of Good Design
5.6 Techniques for Improving Design
5.7 Design Evaluation and Validation
5.8 Documenting the Design
5.9 Information System Example
5.10 Real Time Example
5.11 What this Chapter Means for you

© 2006 Pearson/Prentice Hall

Chapter 5 Objectives

- Conceptual design and technical design
- Design styles, techniques, and tools
- Characteristic of good design
- Validating designs
- Documenting the design

© 2006 Pearson/Prentice Hall

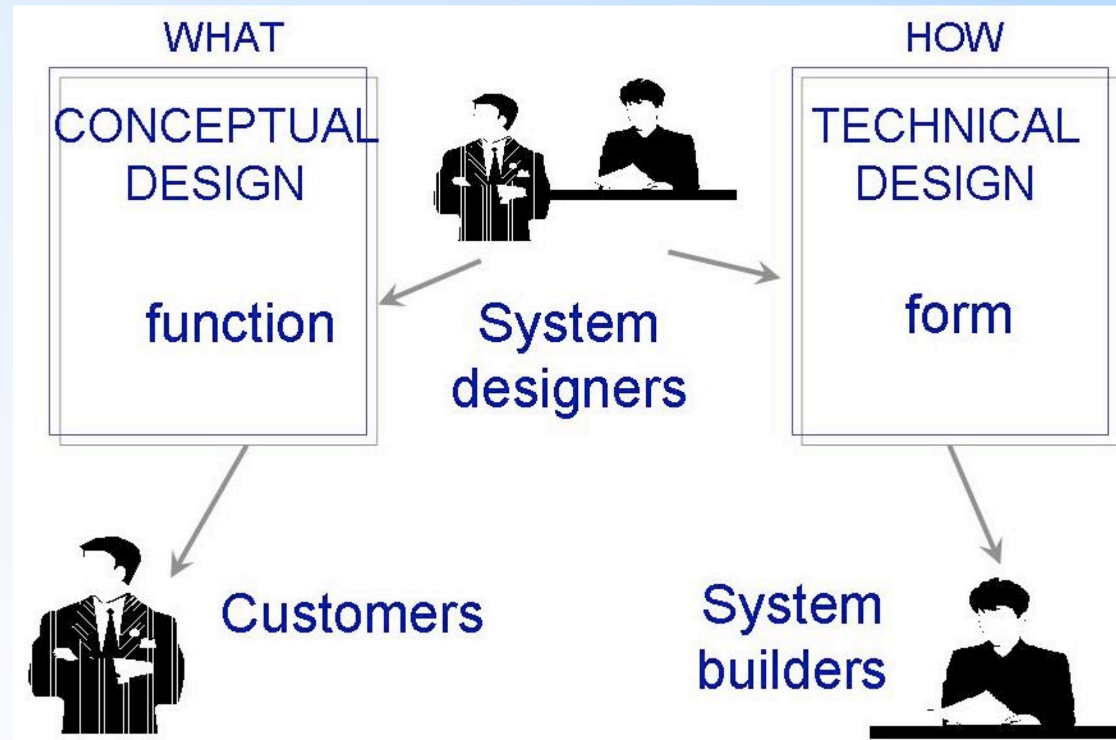
5.1 What Is Design?

- **Design** is the creative process of transforming a problem into a solution
- The description of a solution is also known as “the design”
  - The requirements specification defines a problem
  - The design document specifies a particular solution to that problem

© 2006 Pearson/Prentice Hall

5.1 What Is Design?

- Design is a two-part interactive process
  - Conceptual design (system design)
  - Technical design



© 2006 Pearson/Prentice Hall

5.1 What Is Design?
Conceptual Design

- Tells the customer what the system will do
  - Where will the data come from?
  - What will happen to the data in the system?
  - What will the system look like to users?
  - What choices will be offered to users?
  - What is the timing of events?
  - What will the reports and screens look like?

© 2006 Pearson/Prentice Hall

5.1 What Is Design?
Conceptual Design (continued)

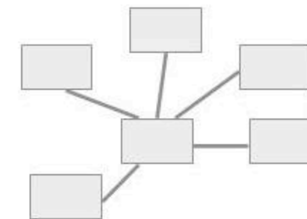
- Characteristics of good conceptual design
  - in customer’s language
  - no technical jargon
  - describes system functions
  - independent of implementation
  - linked to requirements

© 2006 Pearson/Prentice Hall

5.1 What Is Design?
Conceptual Design (continued)

- Graphical representation of the differences in design documentation

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



Network topology
Protocol used
Prescribed bps rate
. . .

CONCEPTUAL DESIGN

TECHNICAL DESIGN

© 2006 Pearson/Prentice Hall

5.1 What Is Design?
Technical Design

- Tells the programmers what the system will do
  - major hardware components and their function
  - hierarchy and functions of software components
  - data structures
  - data flow

© 2006 Pearson/Prentice Hall

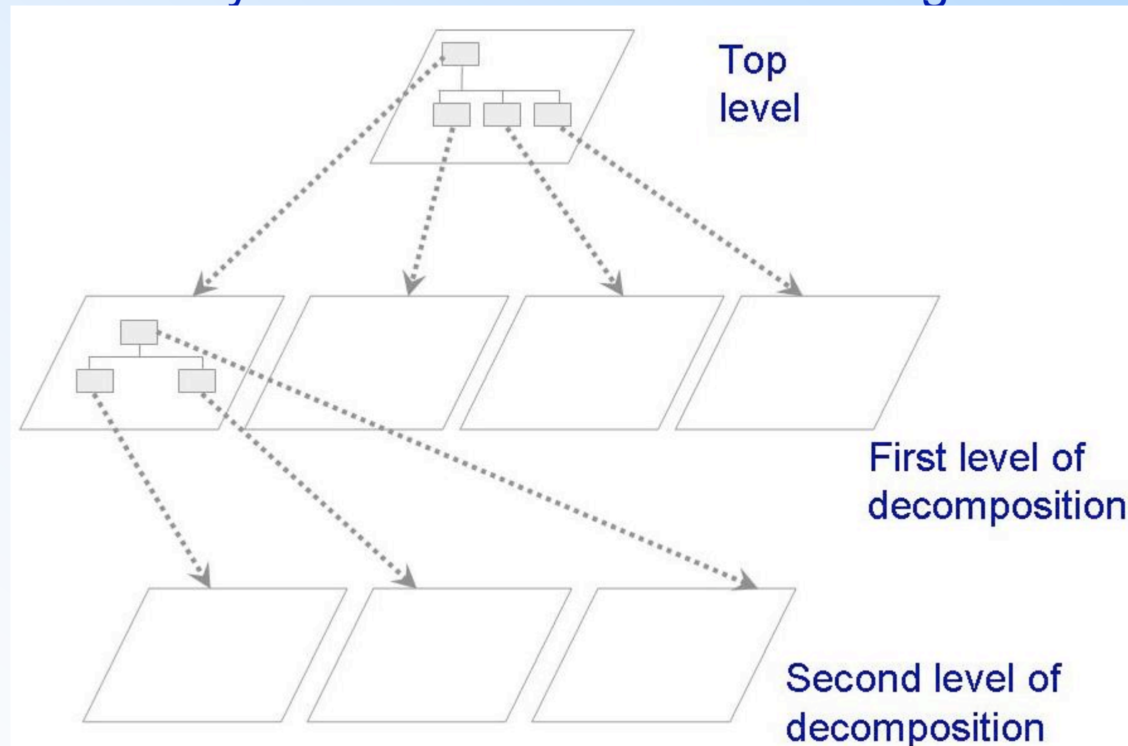
5.2 Decomposition and Modularity
Five Ways to Create Designs

- Modular decomposition
- Data-oriented decomposition
- Event-oriented decomposition
- Outside-in design
- Object-oriented design

© 2006 Pearson/Prentice Hall

5.2 Decomposition and Modularity
Levels of Decomposition

- System data description
- High level functional descriptions
- Creating a hierarchy of information with increasing details



© 2006 Pearson/Prentice Hall

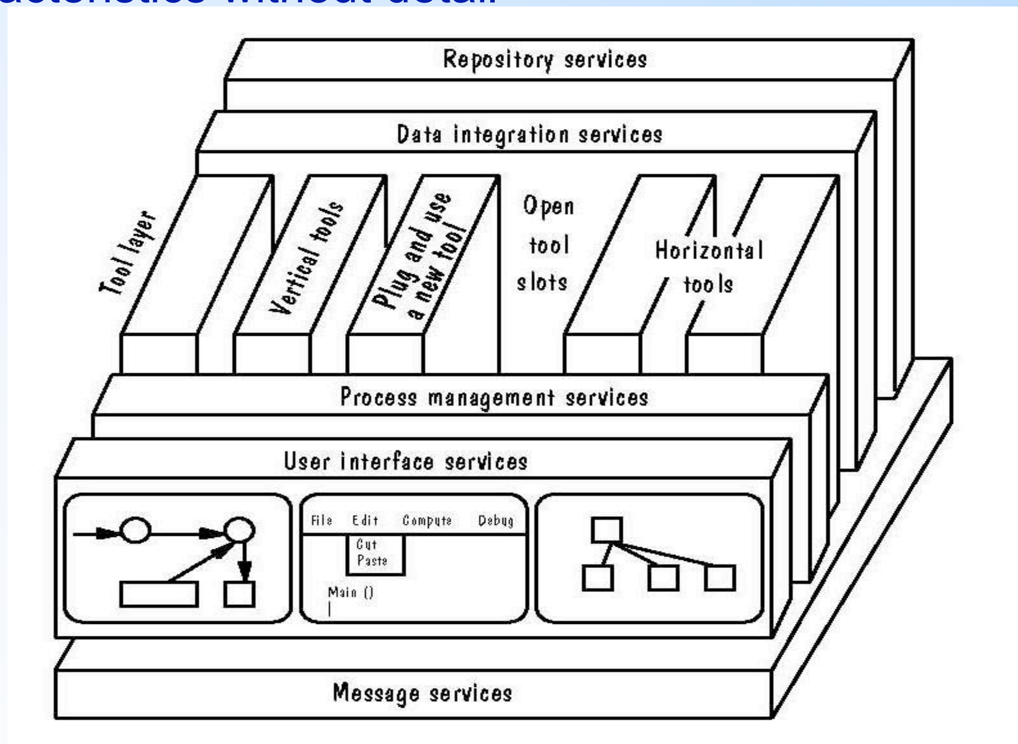
5.2 Decomposition and Modularity
Modularity

- **Modules** or **components**: composite parts of design
- A system is **modular** when
  - each activity of the system is performed by exactly one component
  - inputs and outputs of each component are well-defined
    - all inputs to it are essential to its function
    - all outputs are produced by one of its actions

© 2006 Pearson/Prentice Hall

5.2 Decomposition and Modularity

- Graphical representation of the NIST/ECMA model for environment integration
  - a software architect uses a high level design to explain general characteristics without detail



© 2006 Pearson/Prentice Hall

5.3 Architectural Styles and Strategies
Three Design Levels

- **Architecture**: associates system components with capabilities
- **Code design**: specifies algorithms and data structures for each component
- **Executable design**: lowest level of design, including memory allocation, data formats, bit patterns

© 2006 Pearson/Prentice Hall

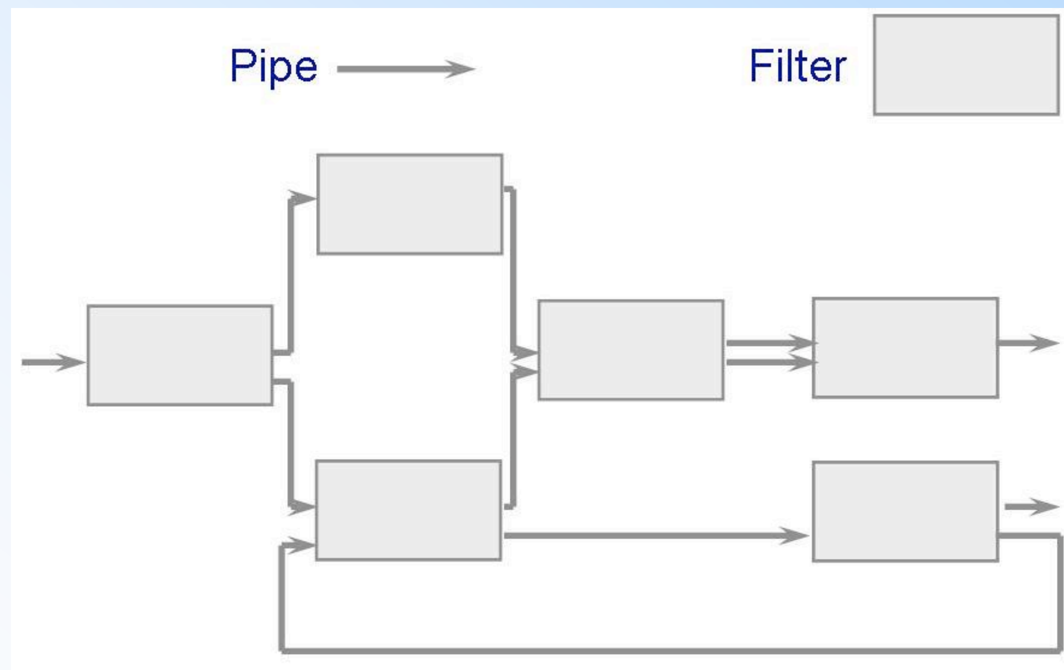
5.3 Architectural Styles and Strategies
Design Styles

- Pipes and filters
- Object-oriented design
- Implicit invocation
- Layering
- Repositories
- Interpreters
- Process control
- Client-server

© 2006 Pearson/Prentice Hall

5.3 Architectural Styles and Strategies
Pipes and Filters

- The system has
  - Streams of data (pipe) for input and output
  - Transformation of the data (filter)



© 2006 Pearson/Prentice Hall

5.3 Architectural Styles and Strategies
Pipes and Filters (continued)

- Several important properties
  - The designer can understand the entire system's effect on input and output as the composition of the filters
  - The filters can be reused easily on other systems
  - System evolution is simple
  - Allow concurrent execution of filters
- Drawbacks
  - Encourages batch processing
  - Not good for handling interactive application
  - Duplication in filters' functions

© 2006 Pearson/Prentice Hall

5.3 Architectural Styles and Strategies
Object-Oriented Design

- Must have two characteristics
  - the object must preserve the integrity of data representation
  - the data representaion must be hidden from other objects
    - easy to change the implementation without perturbing the rest of the system
- One object must know the identity of other objects in order to interact

© 2006 Pearson/Prentice Hall

5.3 Architectural Styles and Strategies
Implicit Invocation

- Event-driven, based on notation of broadcasting
- Data exchange is through shared data in a repository
- Applications
  - packet-switch networks
  - databases to ensure consistency
  - user interfaces
- Useful for reusing design components from other system
- Disadvantage: lack of assurance that a component will respond to an event

© 2006 Pearson/Prentice Hall

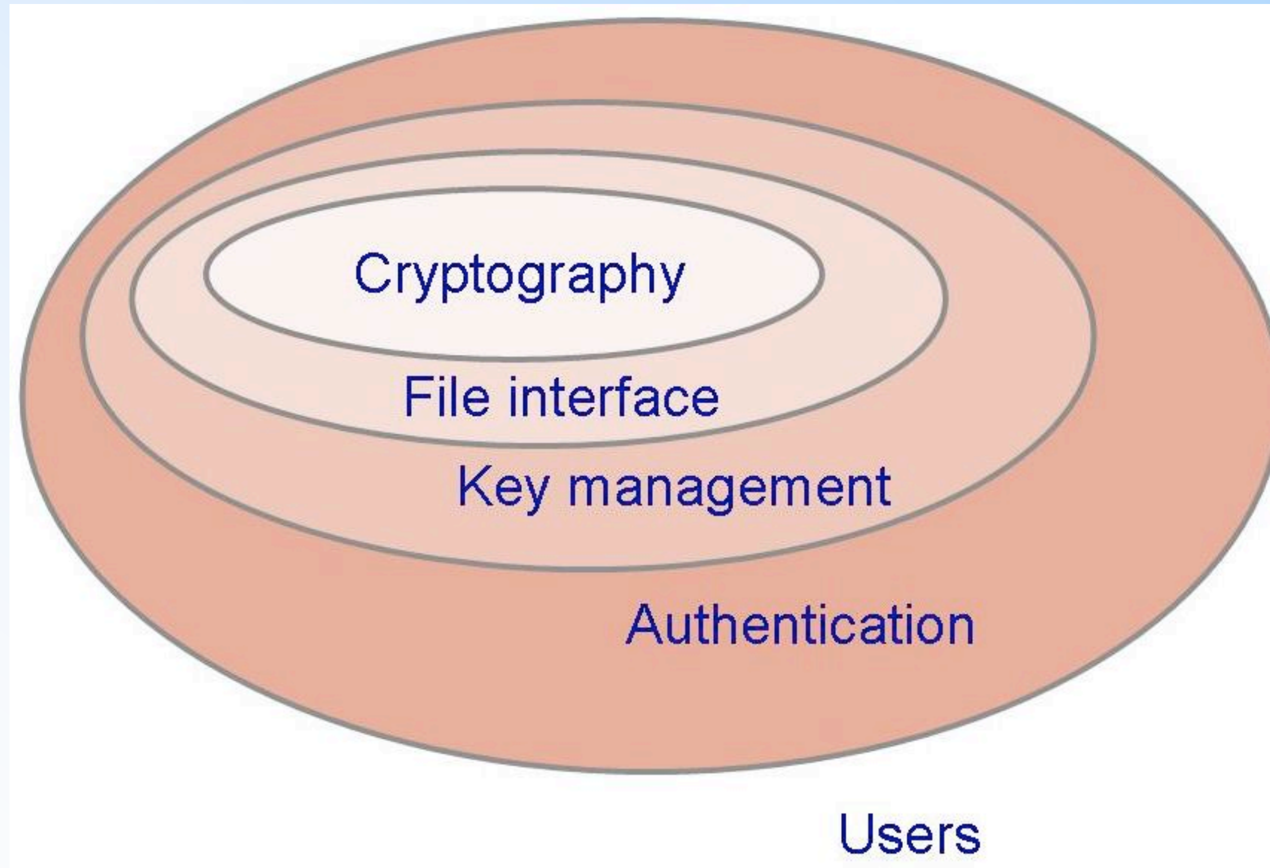
5.3 Architectural Styles and Strategies
Layering

- Layers are hierarchical
  - Each layer provides service to the one outside it and acts as a client to the layer inside it
- The design includes protocols
  - Explain how each pair of layers will interact
- Advantages
  - High levels of abstraction
  - Relatively easy to add and modify a layer
- Disadvantages
  - Not always easy to structure system layers
  - System performance may suffer from the extra coordination among layers

© 2006 Pearson/Prentice Hall

5.3 Architectural Styles and Strategies
Example of Layering System

- A system to provide file security



© 2006 Pearson/Prentice Hall

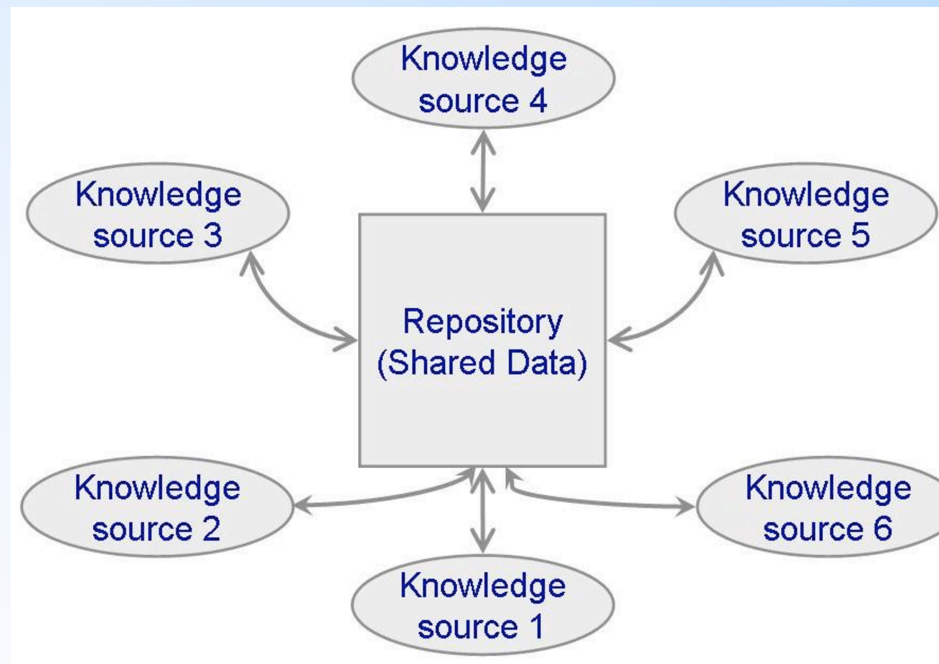
5.3 Architectural Styles and Strategies
Repositories

- Two components
  - A central data store
  - A collection of components that operate on it to store, retrieve, and update information
- The challenge is deciding how the components will interact
  - A traditional database: transactions trigger process execution
  - A blackboard: the central store controls the triggering process

© 2006 Pearson/Prentice Hall

5.3 Architectural Styles and Strategies
Repositories (continued)

- Major advantage: openness
  - Data representation is made available to various programmers (vendors) so they can build tools to access the repository
  - But also a disadvantage: the data format must be acceptable to all components



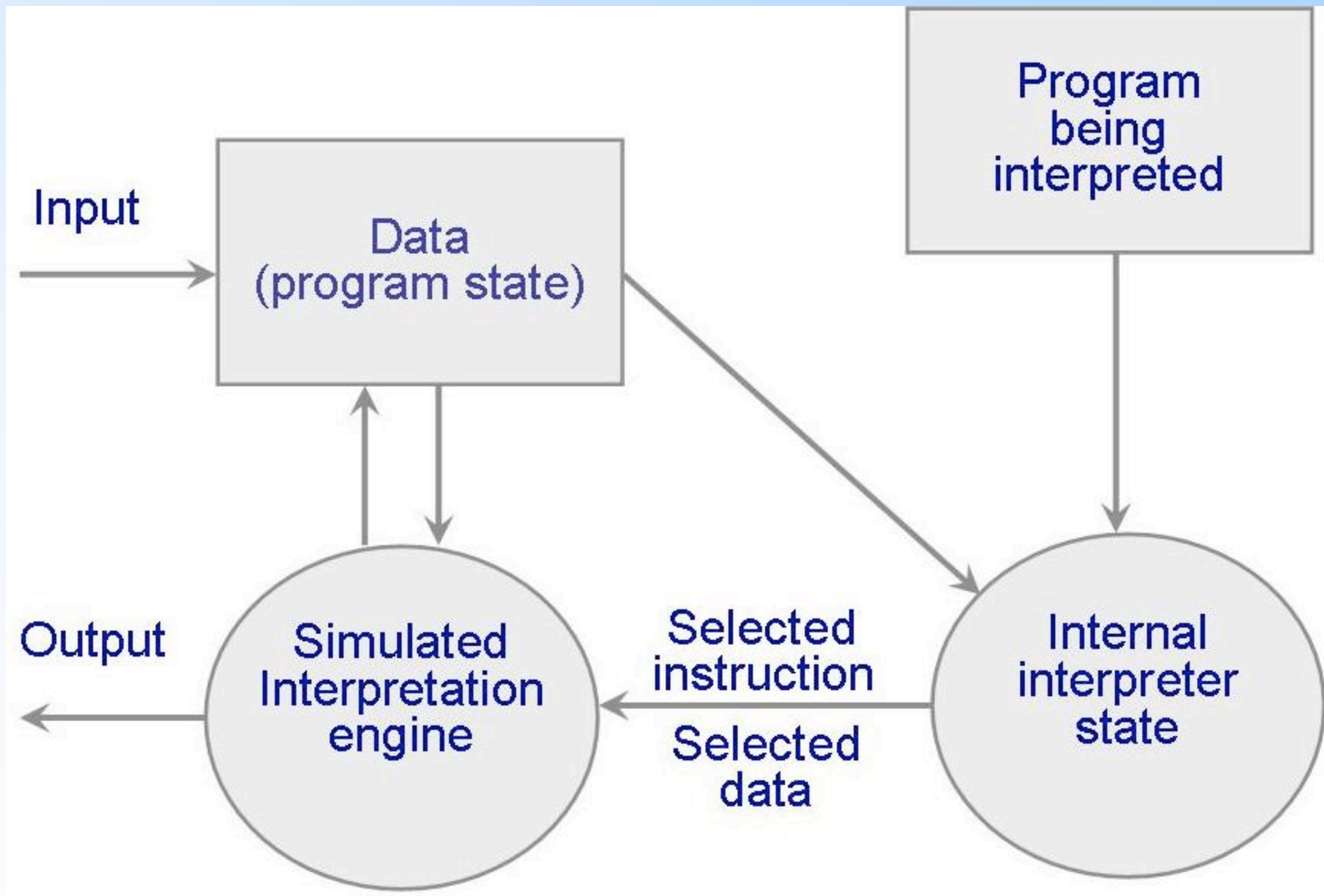
© 2006 Pearson/Prentice Hall

5.3 Architectural Styles and Strategies
Interpreters

- A virtual machine that “interprets” pseudocode in a way that makes it executable
  - Used not only to convert programming language, but also to convert any kind of encoding to a more explicit form
- Composed of four components
  - A memory to contain pseudocode to be interpreted
  - An interpretation engine
  - The current state of the interpretation engine
  - The current state of the program being simulated

© 2006 Pearson/Prentice Hall

5.3 Architectural Styles and Strategies
Example of an Interpreter



© 2006 Pearson/Prentice Hall

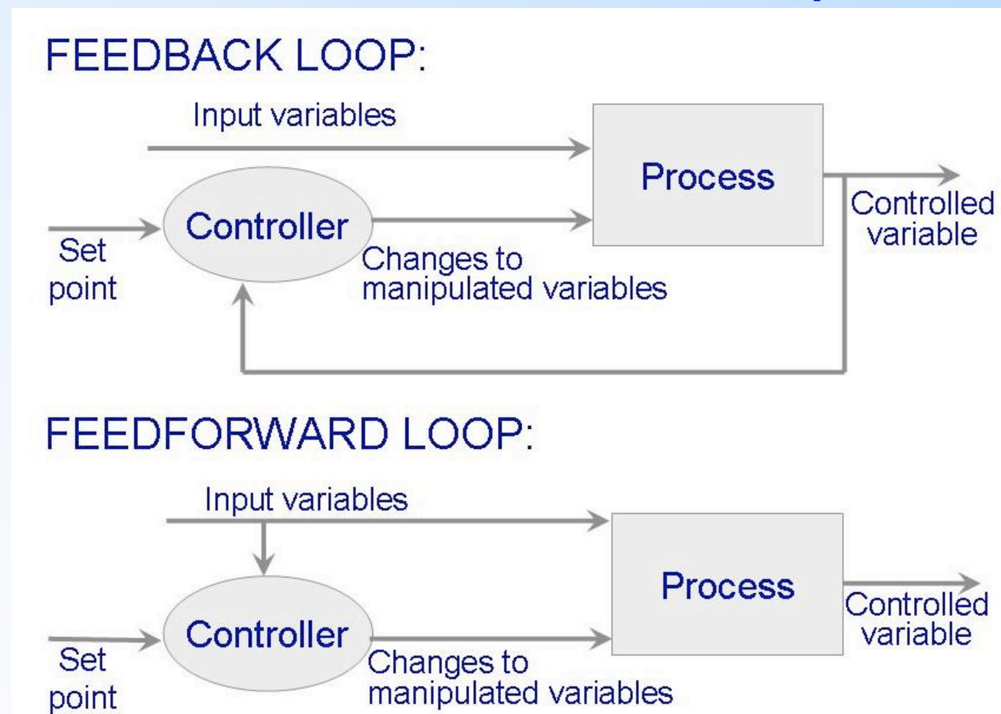
5.3 Architectural Styles and Strategies
Process Control

- Characterized by
  - the type of component
  - the relationships that hold among components
- Purpose: maintain specified properties of process outputs at or near specified reference values called *set points*
- Issues in designing a process control system
  - What variables to monitor
  - What sensor to use
  - How to calibrate them
  - How to deal with the timing of sensing and control

© 2006 Pearson/Prentice Hall

5.3 Architectural Styles and Strategies
Process Control (continued)

- Software-based control system involves a closed loop in one of two forms, *feedback* and *feedforward* as illustrated in the picture



© 2006 Pearson/Prentice Hall

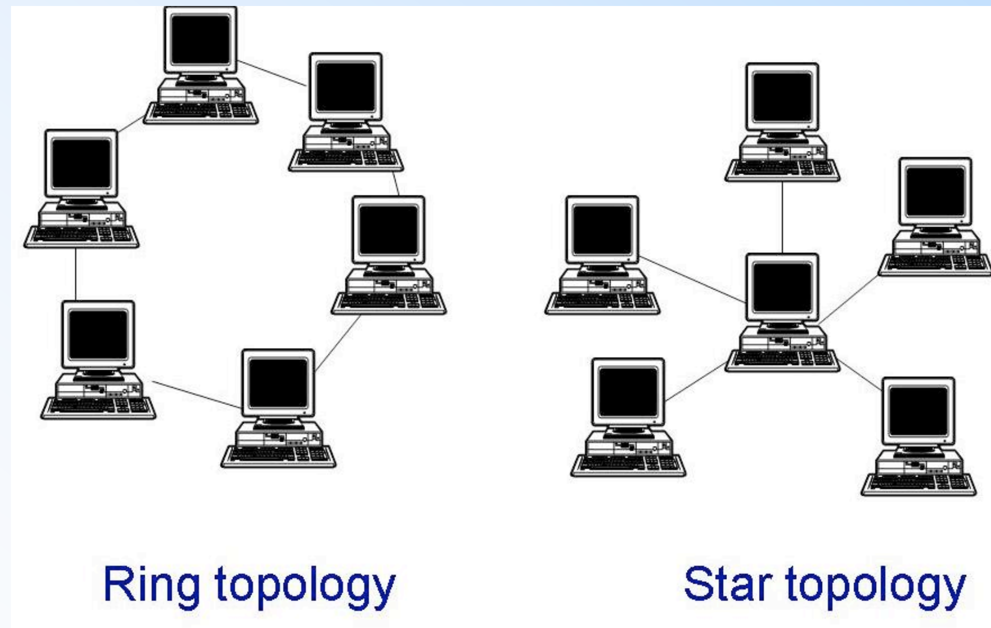
5.3 Architectural Styles and Strategies
Other Styles

- Distributed system architecture: client-server
  - Advantage
    - Users get information they need only when they need it
  - Disadvantage
    - Need more sophisticated security, system management, and application development
- Domain-specific architecture
  - Take advantage of the commonalities afforded by the application domain (e.g., avionics)
- Heterogeneous architectures

© 2006 Pearson/Prentice Hall

5.3 Architectural Styles and Strategies
Client-Server

- Distributed systems usually described in terms of the topology of their configuration.
- They can be organized as a ring or as a star as shown in the picture



© 2006 Pearson/Prentice Hall

5.3 Architectural Styles and Strategies
Sidebar 5.1 The World Cup Client-Server System

- Required both central control and distributed functions
- The system built included a central database for ticket management, security, news service, and Internet link
- The server also calculated games statistics, provided historical information, security photographs, and clips of video action
- The clients ran on 1600 Sun workstations

© 2006 Pearson/Prentice Hall

5.4 Issues in Design Creation

- Modularity and levels of abstraction
- Collaborative design
- Designing the user interface
- Concurrency
- Design patterns and reuse

© 2006 Pearson/Prentice Hall

5.4 Issues in Design Creation
Modularity and Levels of Abstraction

- **Levels of abstraction:** the component at one level refines those in the level above, as we move to lower levels, we find more detail about each component
- **Information hiding**: hide design decisions from others
- Modularity provides the flexibility
  - to understand the system
  - to trace the flow of data and function
  - to target the pockets of complexity

© 2006 Pearson/Prentice Hall

5.4 Issues in Design Creation
Sidebar 5.2 Using Abstraction

Rearrange L in non-decreasing order

```
DO WHILE I is between 1 and (length of L)-1:
  Set LOW to index of smallest value in L(I), ..., L(length of L)
 Interchange L(I) and L(LOW)
END DO
```

```
DO WHILE I  is between 1 and (length of L)              - 1
  Set LOW to current value of I
    DO WHILE J is between I+1 and (length of L)              - 1:
      IF L(LOW) is greater than L(J)
        THEN set LOW to current value of J
      ENDIF
    ENDDO
  Set TEMP to L(LOW)
  S  et L(LOW) to L(I)
  Set L(I) to TEMP
ENDDO
```


© 2006 Pearson/Prentice Hall

5.4 Issues in Design Creation
Collaborative Design

- Most projects are collaborative work
- Issues in collaborative design
  - Who is the best suited to design each aspect of the system
  - How to document the design
  - How to coordinate the design components
- Problems in performing collaborative design
  - Differences in personal experience, understanding, and preference
  - People sometimes behave differently in groups from the way they would behave individually

© 2006 Pearson/Prentice Hall

5.4 Issues in Design Creation
Collaborative Design: Multi Sites Development

- Four stages
  - A project is performed at a single site with on-site developers from foreign country
  - On-site analysts determine the system's requirements. Then the requirements are provided to off-site's designers and developers groups
  - Off-site developers build generic products and components that are used worlwide
  - Off-site developers build products that take advantage of their individual expertise
- Issues
  - Languages
  - Communication paths

© 2006 Pearson/Prentice Hall

5.4 Issues in Design Creation
Sidebar 5.3 The Causes of Design Breakdown

- Lack of specialized design schemas
- Lack of a meta-schema about the design process leading to poor allocation of resources to the various design activities
- Poor prioritization of issues leading to poor selection of alternative solutions
- Difficulty in considering all the stated or inferred constraints in defining a solution
- Difficulty in performing mental simulation with steps or test cases
- Difficulty in keeping track and returning to subproblems whose solution has been postponed
- Difficulty in expanding or merging solutions from individual subproblems to form a complete solution

© 2006 Pearson/Prentice Hall

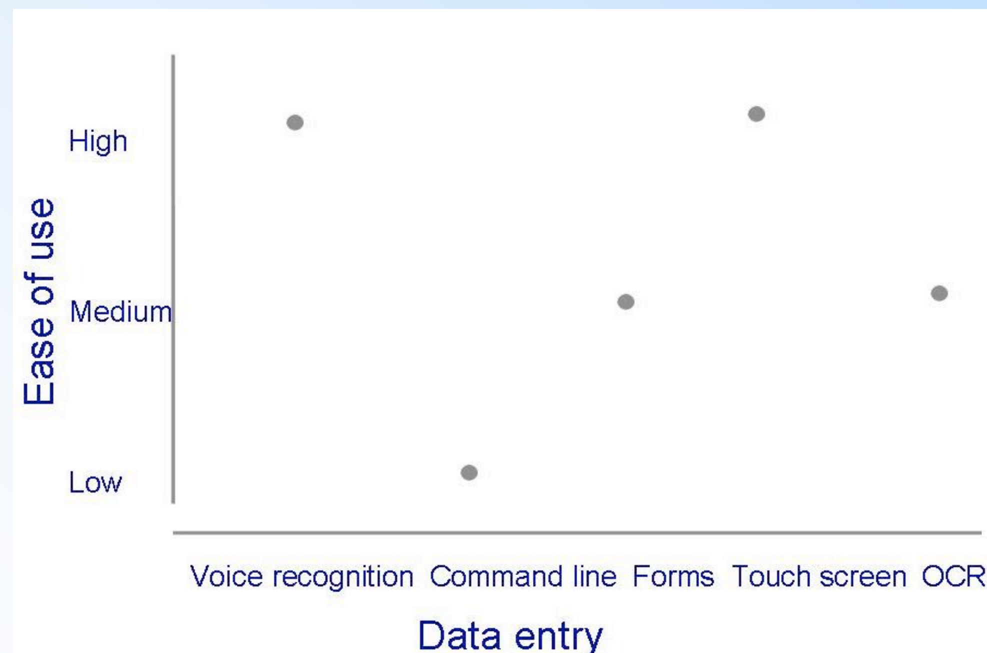
5.4 Issues in Design Creation
Designing the User Interface

- Key elements to be addressed
  - Metaphors
  - A mental model
  - The navigation rules for the model
  - Look: characteristics of the system that convey information to the user
  - Feel: interaction techniques
- Key issues to be considered
  - Cultural issues
  - User preferences

© 2006 Pearson/Prentice Hall

5.4 Issues in Design Creation
Guidelines for Determining User-Interface Characteristics

- Consider design choices in terms of a design space
- Each trade-off reflects at least two dimensions of the choice
- We can view the choices as



© 2006 Pearson/Prentice Hall

5.4 Issues in Design Creation
Issues to Consider in Trade-off Analysis

Functional dimensions	*Structural dimensions*
External event handling:	Application interface abstraction level
• No external events	• Monolithic program
• Process events while waiting for input	• Abstract device
• External events preempt user commands	• Toolkit
	• Interaction manager with fixed data types
	• Interaction manager with extensible data types
	• Extensible interaction manager
User customizability	Abstract device variability
• High	• Ideal device
• Medium	• Parameterized device
• Low	• Device with variable operations
	• Ad hoc device
User interface adaptability across devices	Notation for user interface definition
• None	• Implicit in shared user interface code
• Local behavior changes	• Implicit in application code
• Global behavior change	• External declarative notation
• Application semantics change	• External procedural notation
	• Internal declarative notation
	• Internal procedural notation
Computer system organization	Basis of communication
• Uniprocessing	• Events
• Multiprocessing	• Pure state
• Distributed processing	• State with hints
	• State plus events
Basic interface class	Control thread mechanisms
• Menu selection	• None
• Form filling	• Standard processes
• Command language	• Lightweight processes
• Natural language	• Non-preemptive processes
• Direct manipulation	• Event handlers
	• Interrupt service routines
Application portability across user interface styles	
• High	
• Medium	
• Low	

© 2006 Pearson/Prentice Hall

5.4 Issues in Design Creation
Concurrency

- Problems
  - Consistency of data shared among components that execute at the same time
  - Ensuring that one action does not interfere with another
- Solutions
  - *Synchronization*: method for allowing two activities to take place concurrently without interfering with one another
  - *Mutual exclusion*: one process accessing a data element, no other process can affect the element
  - *Monitor*: an abstract object that controls the mutual exlusion of a particular process

© 2006 Pearson/Prentice Hall

5.4 Issues in Design Creation
Design Patterns and Reuse

- A design pattern names, abstracts, and identifies the key aspects of a common design structure that make it useful for creating reusable design
- Key aspects
  - participating classes and instances
  - roles and collaborations
  - the distribution of responsibilities

© 2006 Pearson/Prentice Hall

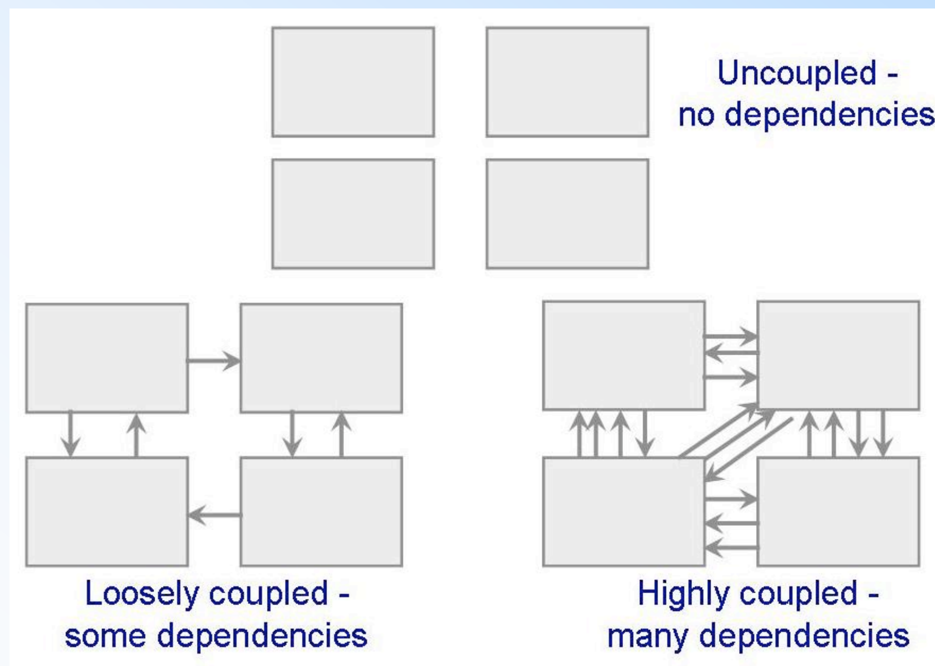
5.5 Characteristics of Good Design

- Component independence
  - coupling
  - cohesion
- Exception identification and handling
- Fault prevention and tolerance
  - active
  - passive

© 2006 Pearson/Prentice Hall

5.5 Characteristics of Good Design
Coupling

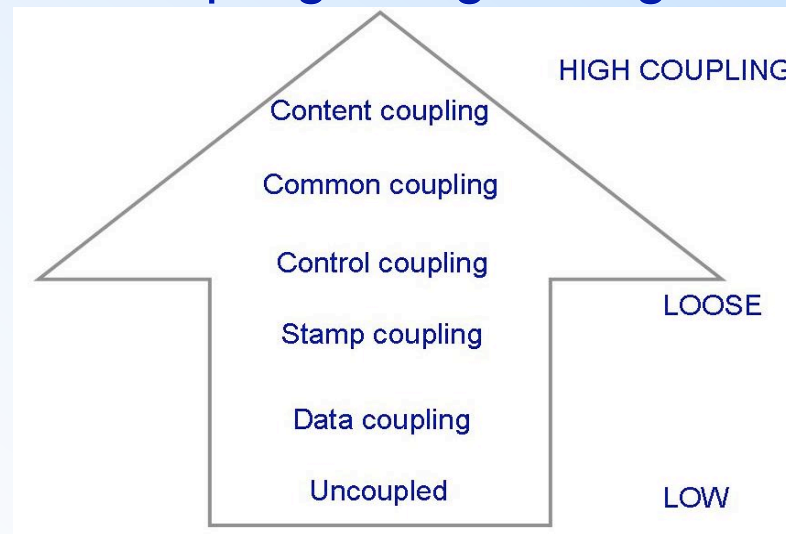
- Highly coupled when there is a great deal of dependencies
- Loosely coupled components have some dependency, but the interconnections among components are weak
- Uncoupled components have no interconnections at all



© 2006 Pearson/Prentice Hall

5.5 Characteristics of Good Design
Coupling (continued)

- Coupling among components depends on
  - the references made
  - the amount of data passed
  - the amount of control
  - the degree of complexity in the interface
- We can measure coupling along a range of dependence



© 2006 Pearson/Prentice Hall

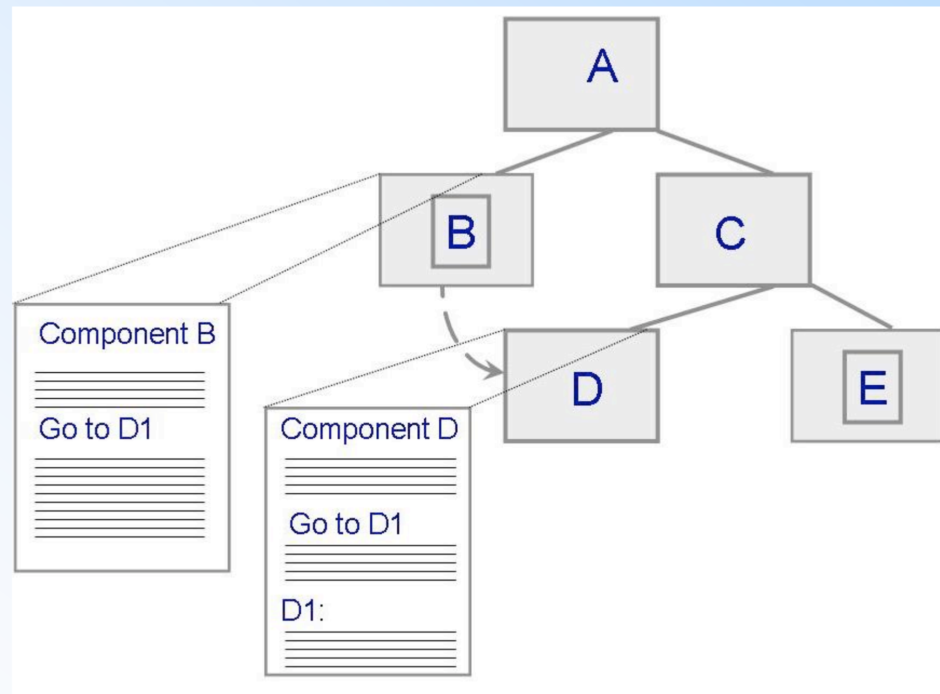
5.5 Characteristics of Good Design
Coupling: Types of Coupling

- Content coupling
- Common coupling
- Control coupling
- Stamp coupling
- Data coupling

© 2006 Pearson/Prentice Hall

5.5 Characteristics of Good Design
Content Coupling

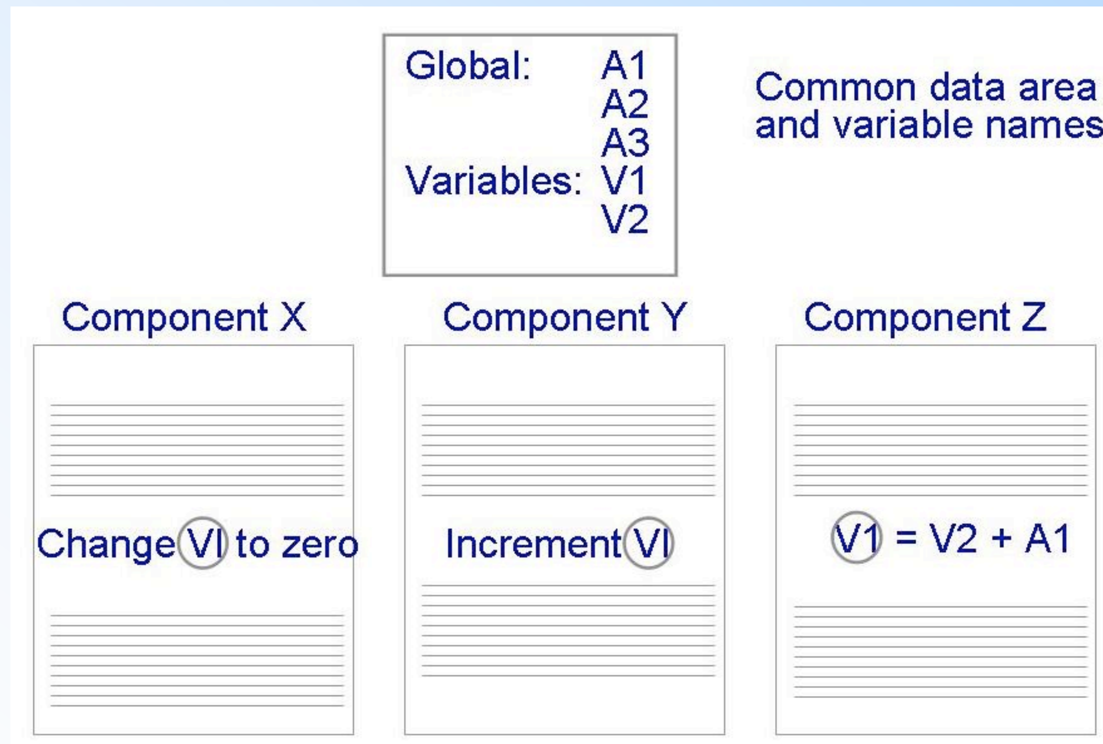
- Occurs when one component modifies an internal data item in another component, or when one component branches into the middle of another component



© 2006 Pearson/Prentice Hall

5.5 Characteristics of Good Design
Common Coupling

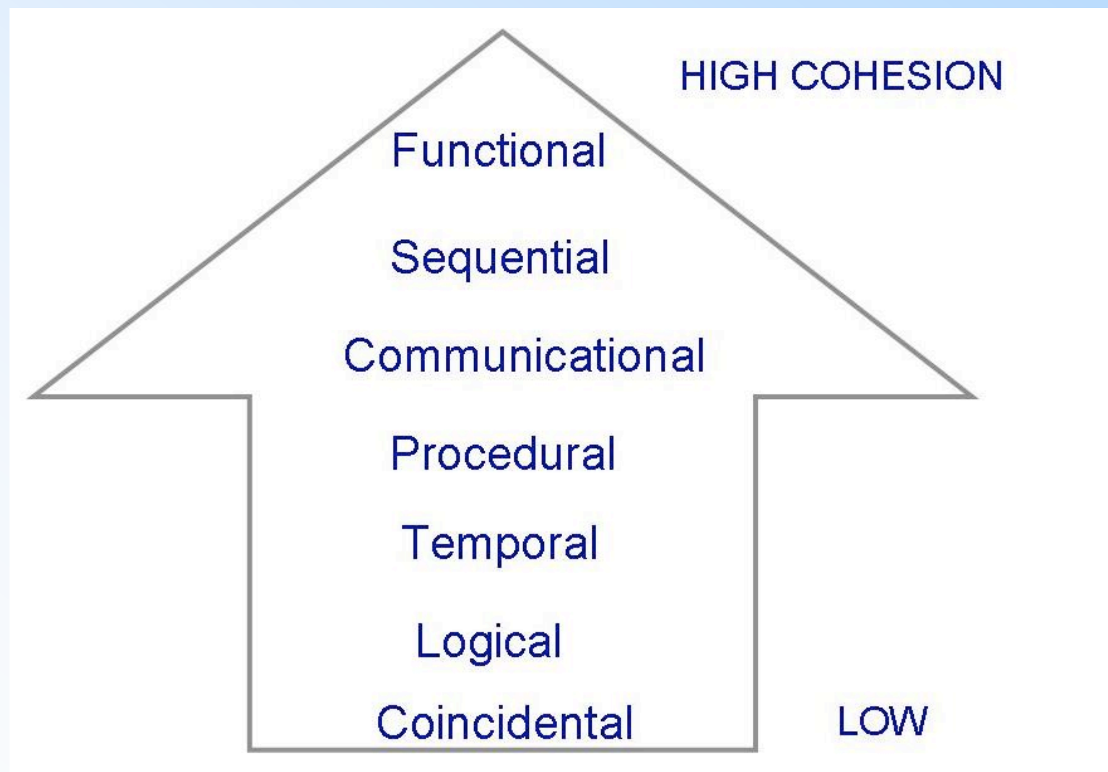
- Making a change to the common data means tracing back to all components that access those data to evaluate the effect of the change



© 2006 Pearson/Prentice Hall

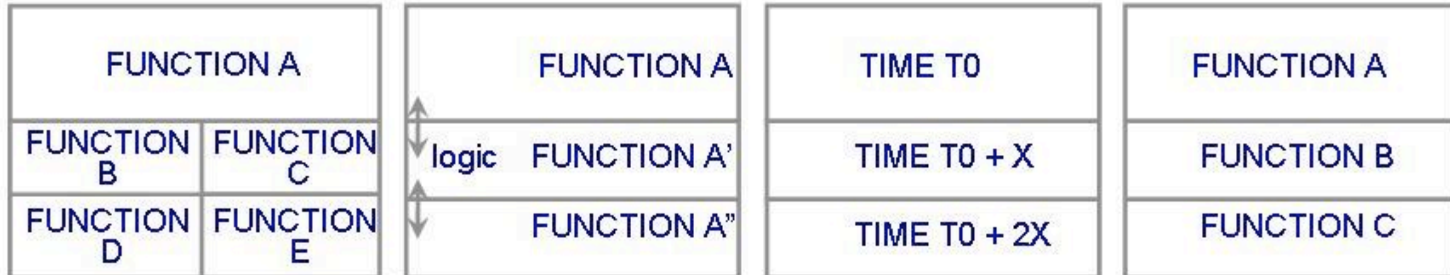
5.5 Characteristics of Good Design
Cohesion

- A component is cohesive if all elements of the component are directed toward and essential for performing the same task
- Several forms of cohesion



© 2006 Pearson/Prentice Hall

5.5 Characteristics of Good Design
Example of Cohesion

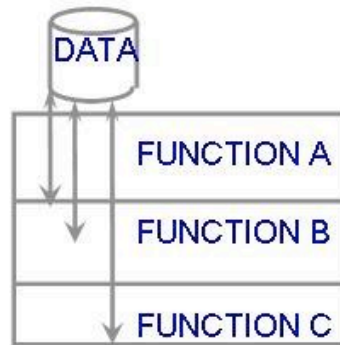


Coincidental
Parts unrelated

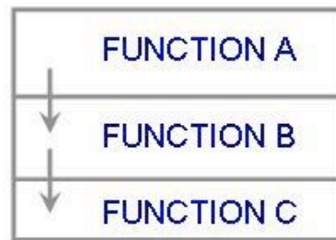
Logical
Similar functions

Temporal
Related by time

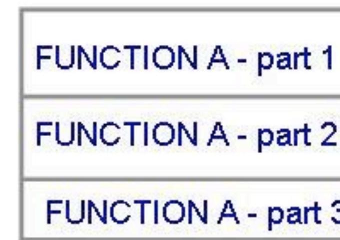
Procedural
Related by order of functions



Functional
Access same data



Sequential
Output of one part is input to next



Functional
Sequential with complete, related functions

© 2006 Pearson/Prentice Hall

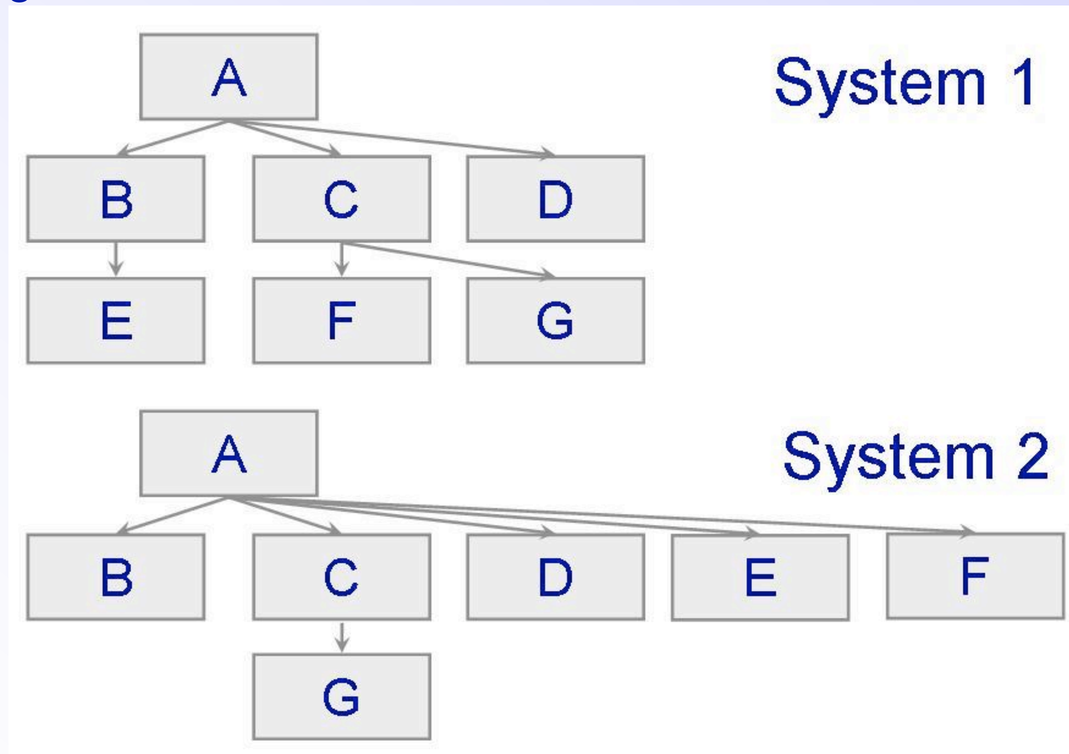
5.5 Characteristics of Good Design
Exception Indentification and Handling

- Exceptions: situations that we know are counter to what we really want the system to do
  - failure to provide a service
  - providing the wrong service or data
  - corrupting data
- Exceptions can be handled in one of three ways
  - retry
  - correct
  - report

© 2006 Pearson/Prentice Hall

5.5 Characteristics of Good Design
Sidebar 5.4 Control Issues

- System 1 and 2 are two possible designs for the same system
  - *Fan-in* is the number of components controlling particular design
  - *fan-out* is number or components controlled by a component
- Better design when it has low fan out



© 2006 Pearson/Prentice Hall

5.5 Characteristics of Good Design
Fault Prevention and Tolerance

- **Active fault detection**: periodically check for symptoms of faults, or try to anticipate when failure will occur
- **Passive fault detection**: wait until a failure occurs during execution
- **Fault correction**: the system's compensation for a fault's presence
- **Fault tolerance**: the isolation of damage caused by a fault

© 2006 Pearson/Prentice Hall

5.5 Characteristics of Good Design
Sidebar 5.5 The Need for Safe Design

- From 1986 to 1997 there were over 450 reports filed with U.S Food and Drug Administration, detailing software defects in medical devices, 24 of which led to death or injury
- Leveson and Turner describe in great detail the user-interface design probem that led to at least three deaths and several injuries from a malfunctioning radiation therapy machine
- June 1997, new federal regulations authorized the FDA to examine the software design of medical devices
- Software designers must see directly how their products will be used, rather than rely on salespeople and marketers

© 2006 Pearson/Prentice Hall

5.6 Techniques for Improving Design

- Reducing complexity
- Design by contract
- Prototyping design
- Fault-tree analysis

© 2006 Pearson/Prentice Hall

5.6 Techniques for Improving Design
Reducing Complexity

- Look for ways to reduce the complexity of diagrams
  - e.g. reduce “crossovers”
  - even better: simplify the diagram by finding structure that are not “pulling their own weight”
    - reassign their responsibilities and eliminate
- “It seems that perfection is reached not when there is nothing left to add, but when there is nothing left to take away.” — Antoine de Saint-Exupéry, Terre des hommes, 1939

© 2006 Pearson/Prentice Hall

5.6 Techniques for Improving Design
Design by Contract

- Suggested by Meyer to ensure that a design meets its specifications (*contracts)*
- Meyer applies the notion of contract to software
  - A *client*: a software component
  - *Supplier*: perform subtask requested by a client
  - *Precondition*: mutual obligation
  - *Postcondition*: benefits
  - *Invariant:* consistency constraint
  - *Assertions:* contract properties

© 2006 Pearson/Prentice Hall

5.6 Techniques for Improving Design
Example of Design by Contract

- Suppose the client component has a table where each element is identified by a character string used as a key
- Supplier's component's task is to insert an element from the table to the dictionary.
- The formalized contract in the object oriented language

```
put (x: ELEMENT; key: STRING) is
--  insert x so that it will be retrievable through key.
require
count <= capacity;
not key.empty
do
... Some insertion algorithm...
ensure
has (x);
item (key) = (x);
count = old count + 1
end
```


© 2006 Pearson/Prentice Hall

5.6 Techniques for Improving Design
Example of Design by Contract

- Meyer's implementation of Design by Contract is "heavyweight"
  - preconditions, postconditions, and invariants in Eiffel are part of the syntax
    - required for each method you create!
- A lightweight approach is to use your programming languages assertion mechanism
  - Typically as simple as
    - assert (boolean condition)
    - If the condition is false, an exception is thrown
  - An assertion at the start of method is a pre-condition, and assertion at the end is a post-condition
    - Class invariants are harder to achieve with this method

© 2006 Pearson/Prentice Hall

5.6 Techniques for Improving Design
Prototyping Design

- Same advantages provided during design stage
- A feasibility prototype can explore whether the proposed solution will actually solve the problem
  - Such prototypes are often "throwaways"
- But not always, sometime parts of a prototype can be saved to be used in the actual system
  - In this situation, since you have a design in hand, the prototype can be built to match the design and then "evolved" into the production system

© 2006 Pearson/Prentice Hall

5.6 Techniques for Improving Design
Fault-tree Analysis: Steps

- Identifying possible failures
- Building a graph
  - Nodes are failures, either of single components, system functions, or the entire system
  - Edges indicate the relationships among nodes
- Searching for several types of design weakness
  - single point of failure
  - uncertainty
  - ambiguity
  - missing components

© 2006 Pearson/Prentice Hall

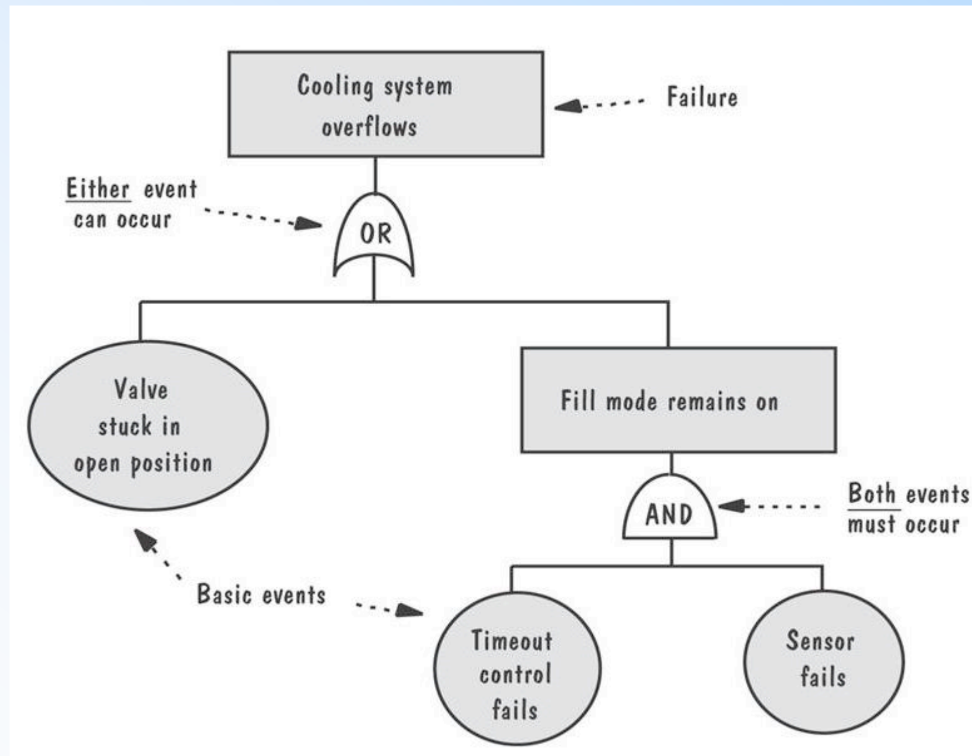
5.6 Techniques for Improving Design
Guidewords for Identifying Possible Failures

Guideword	Interpretation
no	No data or control signal was sent or received
more	The volume of data is too much or too fast
less	The volume of data is too low or too slow
part of	The data or control signal is incomplete
other than	The data or control signal has another component
early	The signal arrives too early for the clock
late	The signal arrives too late for the clock
before	The signal arrives too early in the expected sequence
after	The signal arrives too late in the expected sequence

© 2006 Pearson/Prentice Hall

5.6 Techniques for Improving Design
Fault-tree Analysis: An Example

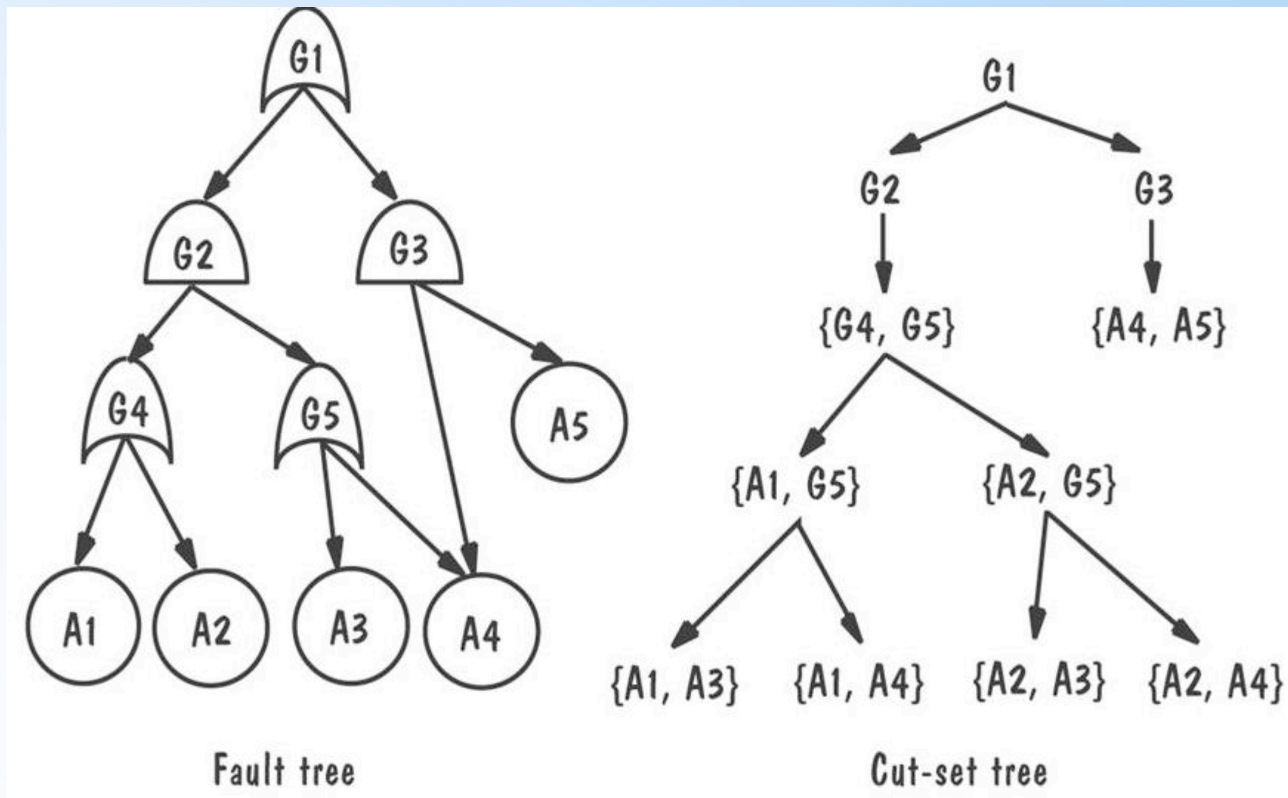
- Portion of power plant control system
- From this fault tree we can construct another tree, known as a cut-set tree



© 2006 Pearson/Prentice Hall

5.6 Techniques for Improving Design
Fault-tree Analysis: Example (continued)

- Cut-set tree generated from the fault tree of a portion of the power plant control system



© 2006 Pearson/Prentice Hall

5.6 Techniques for Improving Design
Fault-tree Analysis: Example (continued)

- The leaf-nodes in the cut-set identify events that can lead to the failure of the system
  - We then examine the design, assume a failure has occurred and see if we can find a set of events that will produce it;
  - Note: this is different from the original fault tree, which is constructed by asking how a failure can occur, not whether the current design will cause that failure
  - If we conclude that the failure can occur with the present design, then we have to work to remove and/or mitigate the identified fault

© 2006 Pearson/Prentice Hall

5.7 Design Evaluation and Validation

- Mathematical validation
- Measuring design quality
- Comparing designs
  - one specification, many designs
  - comparison table
- Design reviews

© 2006 Pearson/Prentice Hall

5.7 Design Evaluation and Validation
Mathematical Validation

- Break the system into a set of processes
  - A set of inputs
  - A set of expected outputs
  - A set of assertions about the process
- For each process, we demonstrate
  - If the set of inputs is formulated correctly, it is transformed properly into the set of expected output
  - The process terminates without failure
- This procedure “proves” that the design is correct

© 2006 Pearson/Prentice Hall

5.7 Design Evaluation and Validation
Measuring Design Quality

- Proposed measurements to assess certain key aspects of design quality
  - Measures of cohesion for OO design Measures high-level design, including cohesion and coupling
- Complexity involves two aspects
  - Complexity within each component
  - The complexity of the relationships among the components

© 2006 Pearson/Prentice Hall

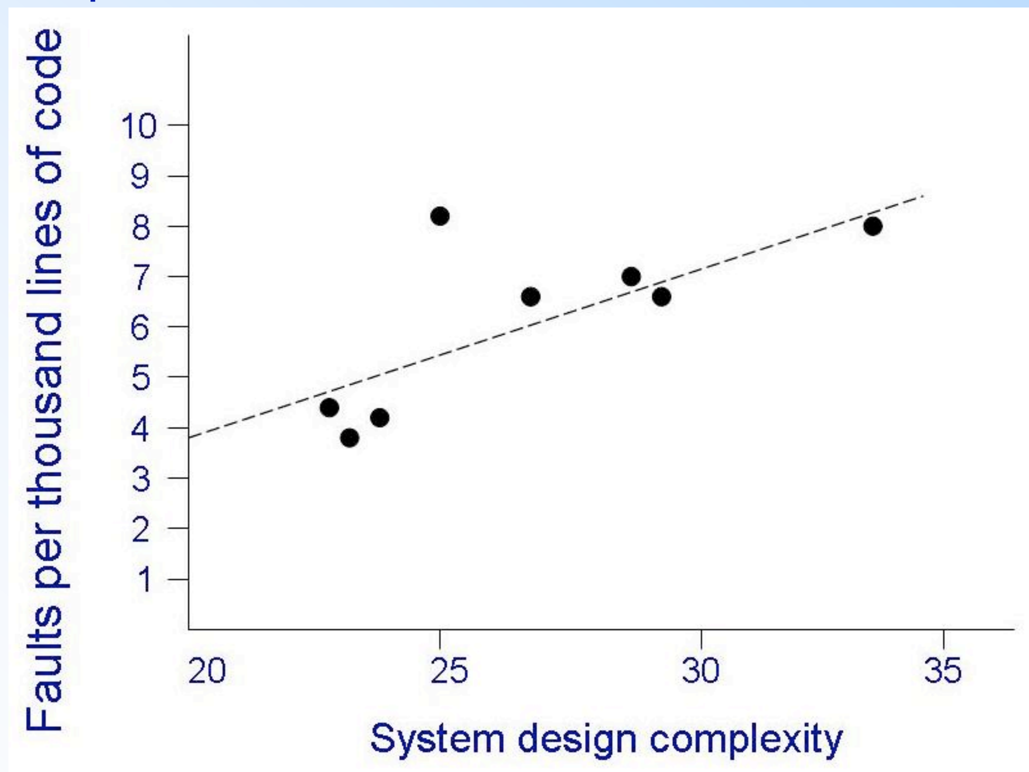
5.7 Design Evaluation and Validation
Card and Glass's Measure of Complexity

- $C = S + D$
- where
  - $S = (1/n) \Sigma f^2(i)$
  - $D = V(i)/[f(i) + 1]$
    - S = the structural complexity (between comps)
    - D = the data complexity (within components)
    - $f(i)$ = the fan-out of component i
    - $V(i)$ = the number of input and output variables in component i
    - n = the number of components

© 2006 Pearson/Prentice Hall

5.7 Design Evaluation and Validation
System's Complexity vs. Number of Faults

- Fault rate graphed against system design complexity
  - Each increase of one unit of complexity increased the fault rate by 0.4 faults per thousand lines of code



© 2006 Pearson/Prentice Hall

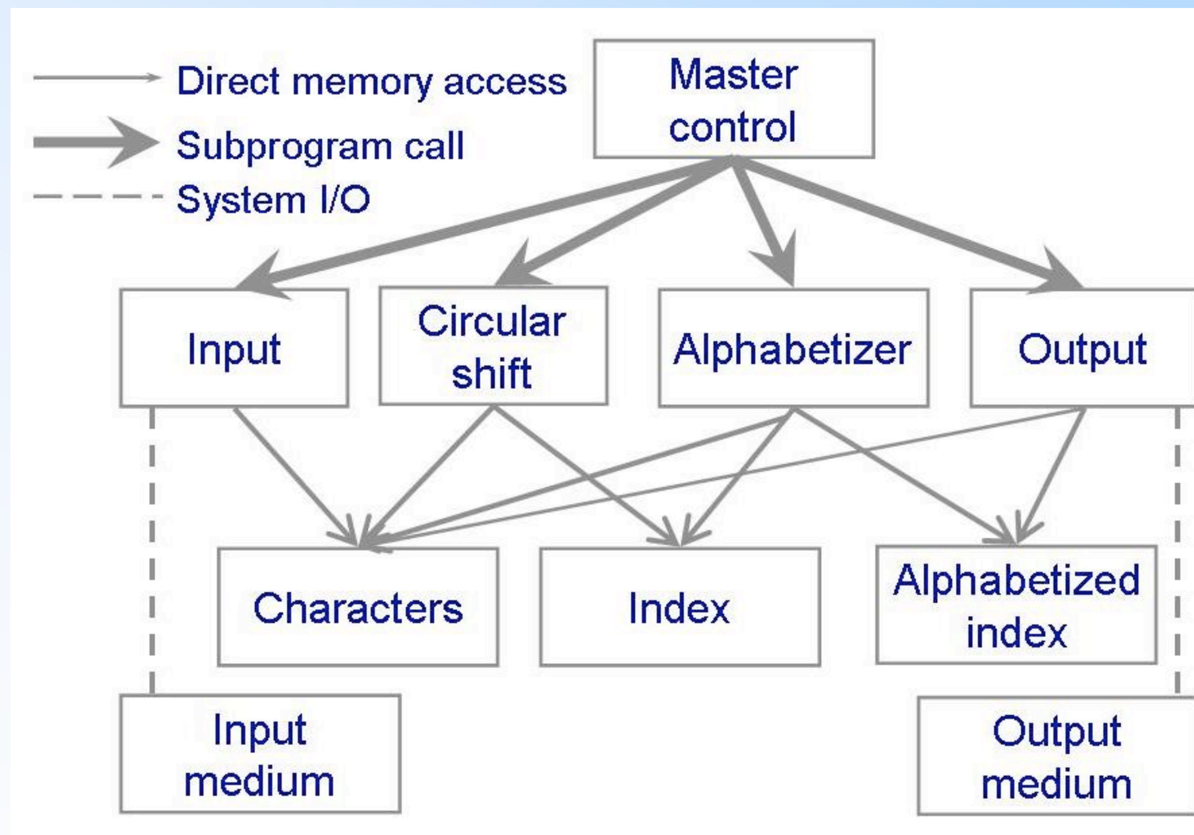
5.7 Design Evaluation and Validation
Comparing Designs

- **One specification, many designs**: to see how different designs can be used to solve the same problem
- Example
  - Shaw and Garland present four different architectural designs to implement KWIC (key word in context)
    - shared data
    - abstract data type
    - implicit invocation
    - pipe and filter

© 2006 Pearson/Prentice Hall

5.7 Design Evaluation and Validation
Shared Data Solution for KWIC

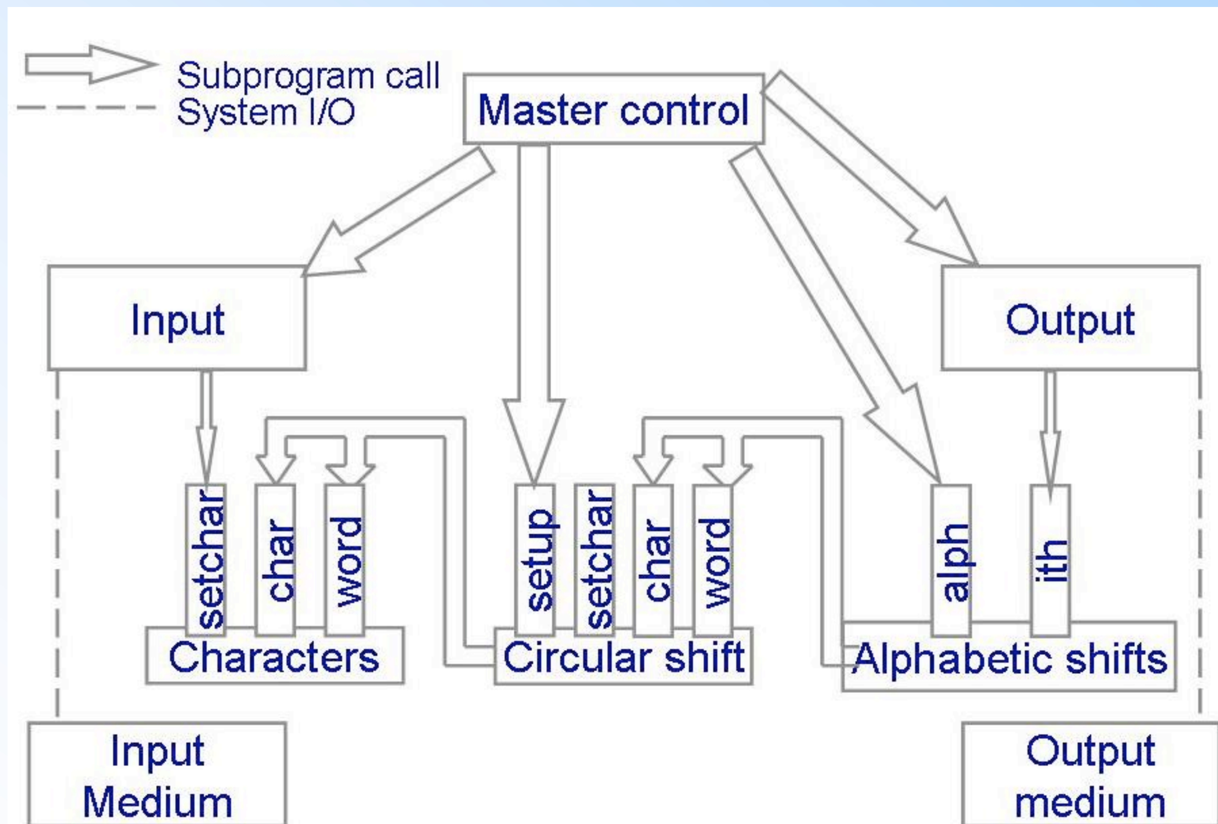
- The problem is broken into its four functional parts: input, circular shift, alphabetize, and output



© 2006 Pearson/Prentice Hall

5.7 Design Evaluation and Validation
Abstract Data Type Solution for KWIC

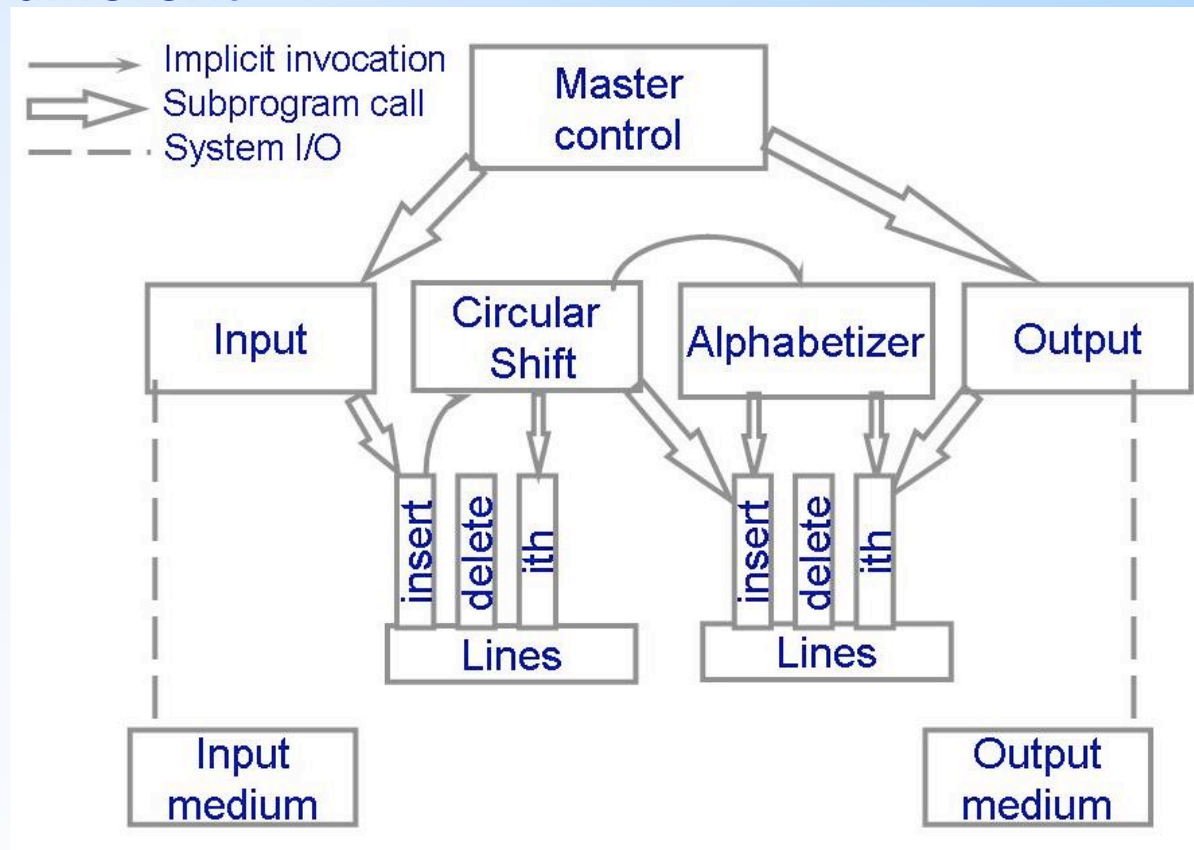
- Data is no longer centrally stored and shared, but the decomposition process is the same



© 2006 Pearson/Prentice Hall

5.7 Design Evaluation and Validation
Implicit Invocation Solution for KWIC

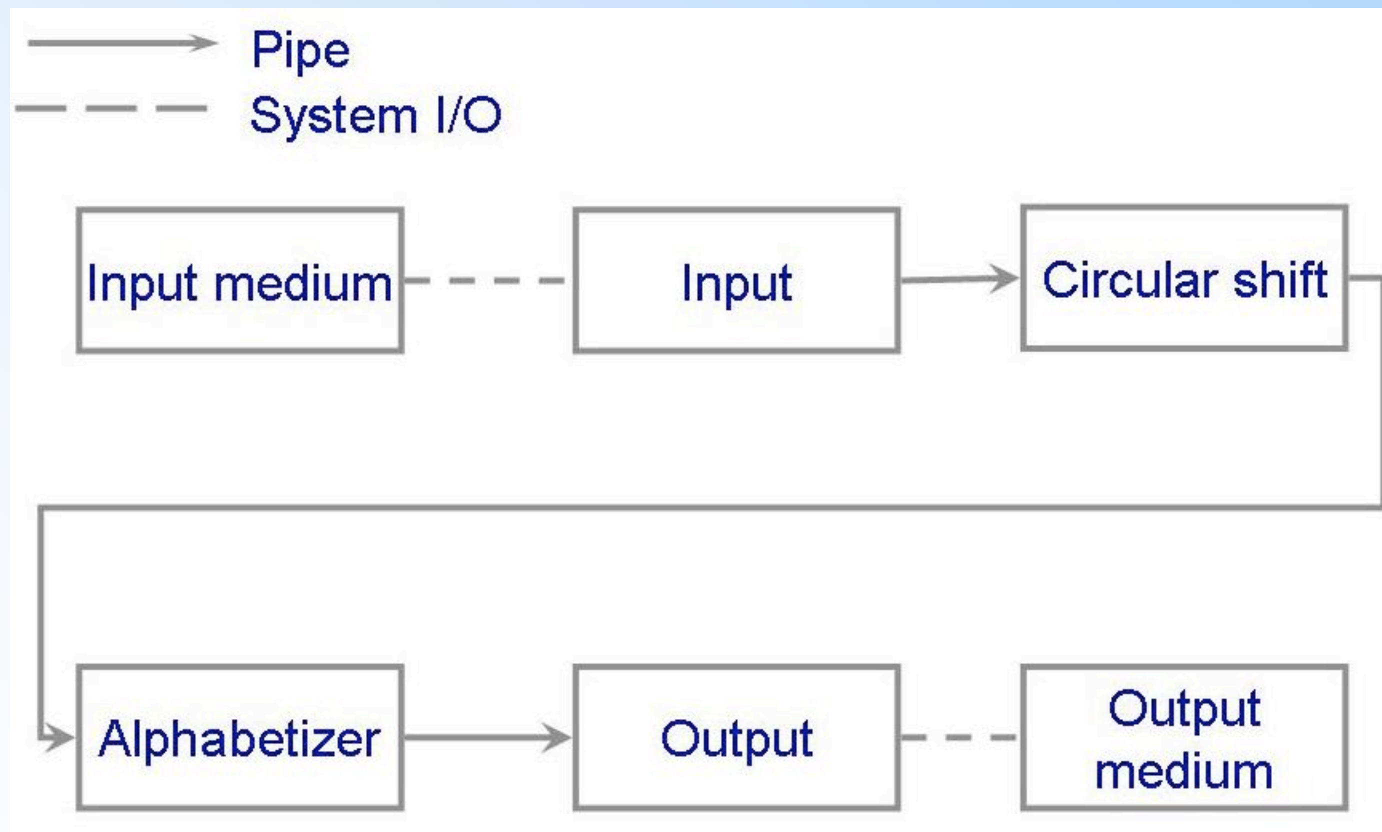
- Another shared data solution, but the interface to the data is very different



© 2006 Pearson/Prentice Hall

5.7 Design Evaluation and Validation
Pipe-and-Filter Solution for KWIC

- The sequence of processing is controlled by the sequence of filters



© 2006 Pearson/Prentice Hall

5.7 Design Evaluation and Validation
Shaw and Garland's Comparison

Attribute	*Shared data*	*Abstract data type*	*Implicit invocation*	*Pipe and filter*
Easy to change algorithm	-	-	+	+
Easy to change data representation	-	+	-	-
Easy to change function	+	-	+	+
Good performance	+	+	-	-
Easy to reuse	-	+	-	+

© 2006 Pearson/Prentice Hall

5.7 Design Evaluation and Validation
Comparison Tables (continued)

- Weighted comparison of Shaw and Garland design

Attribute	***Priority***	*Shared data*	*Abstract data type*	*Implicit invocation*	*Pipe and filter*
Easy to change algorithm	1	1	2	4	5
Easy to change data representation	4	1	5	2	1
Easy to change function	3	4	1	4	5
Good performance	3	5	4	2	2
Easy to reuse	5	1	4	2	5

© 2006 Pearson/Prentice Hall

5.7 Design Evaluation and Validation
Comparison Tables (continued)

- With the previous table, we can then assign a weighted score to each design
  - Sum(priority(i) x design(i)) where i represents ith att
  - Shared Data design is
  - 1x1 + 4x1 + 3x4 + 3x5 + 5x1 = 37
- You compute a value for each design and choose the design with the highest value
  - This is a subjective technique, since attributes, priorities, and design values are all assigned by a project's managers/designers

© 2006 Pearson/Prentice Hall

5.7 Design Evaluation and Validation
Design Reviews

- Preliminary design review
  - examines conceptual design with customer and users
- Critical design review
  - presents technical design to developers
- Program design review
  - programmers get feedback on their designs before implementation

© 2006 Pearson/Prentice Hall

5.7 Design Evaluation and Validation
Questions for any Design Review

- Is it a solution to the problem?
- Is it modular, well-structured, and easy to understand?
- Can we improve the structure and understandability?
- Is it portable to other platforms?
- Is it reusable?
- Is it easy to modify or expand?
- Does it support ease of testing?
- Does it maximize performance, where appropriate?
- Does it reuse components from other projects, where appropriate?
- Are the algorithms appropriate, or can they be improved?
- If this system is to have a phased development, are the phases interfaced sufficiently so that there is an easy transition from one phase to the next?
- Is it well-documented, including design choices and rationale?
- Does it cross-reference the components and data with the requirements?
- Does it use appropriate techniques for handling faults and preventing failures?

© 2006 Pearson/Prentice Hall

5.8 Documenting the Design
Document Contains

- Design rationale
  - Outlining the critical issues and trade-offs
    - especially with respect to non-functional issues
- Descriptions of the system's component
- A section that addresses how the user interacts with the system
- A set of diagrams or formal notations that describes the overall organization and structure of the system
- If our system is distributed, then we also include a topology of the system's network

© 2006 Pearson/Prentice Hall

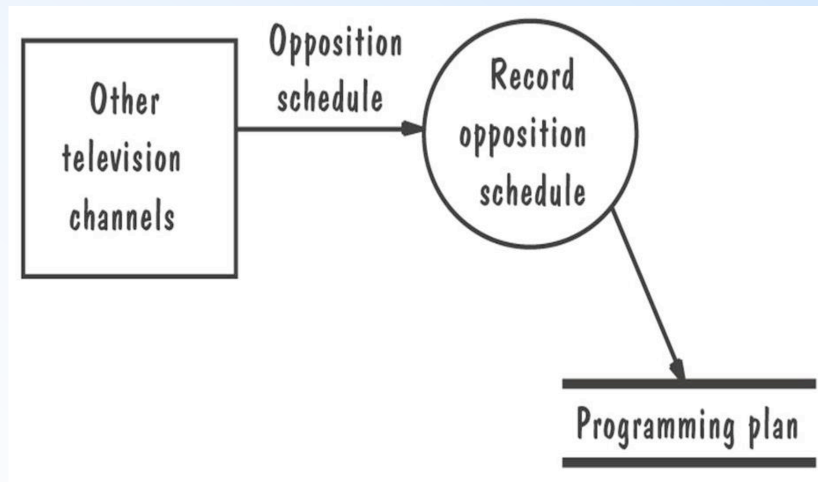
5.8 Documenting the Design
Section for How Users Interact with the System

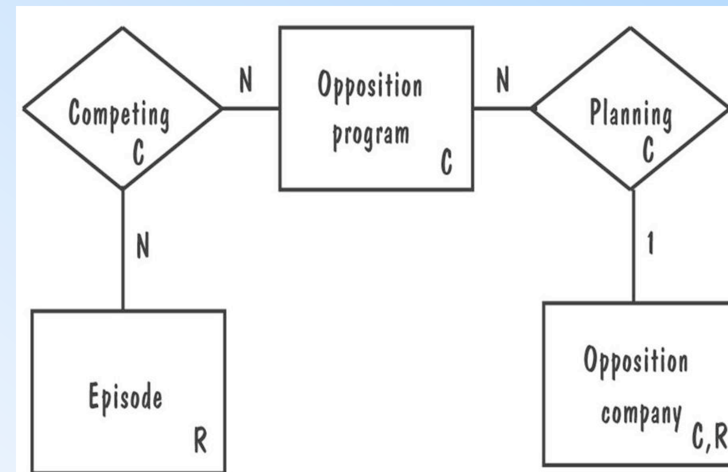
- menus and other display-screen formats
- human interfaces: function keys, touch screen descriptions, keyboard layouts, use of a mouse or joystick
- report formats
- input: where data come from, how they are formatted, on what media they are stored
- output: where data are sent, how they are formatted, on what media they are stored
- general functional characteristics
- performance constraints
- archival procedures
- fault-handling approach

© 2006 Pearson/Prentice Hall

5.9 Information System Example
Picadilly System

- Using a combination of techniques for documenting the design
- A system for tracking opposition schedule: data flow and the data model





© 2006 Pearson/Prentice Hall

5.9 Information System Example
Picadilly System Data Dictionary

Opposition schedule = * Data flow *
Television company name
+ {Opposition transmission date
+ Opposition transmission time + Opposition program name
+ (Opposition predicted rating)}

Input: *Opposition schedule*
For each *Television company name*, create *Opposition company*.
For each *Opposition schedule*,
Locate the *Episode* where *Episode schedule date* = *Opposition transmission date* AND *Episode start time* = *Opposition transmission time*
Create instance of *Opposition* program
Create the relationships *Planning* and *Competing*
Output: List of *Opposition programs*

© 2006 Pearson/Prentice Hall

Ariane 5 Disaster

- On June 4, 1996, after 7 billion dollars of development, an unmanned Ariane 5 rocket exploded just forty seconds after lift-off
  - The rocket and its cargo were valued at $500 million for a total cost of 7.5 billion dollars!
- The error was traced to a software component in the Inertial Reference System that had been reused from the Ariane 4 flight software
  - The reused component was more than 10 years old and had flown successfully on numerous Ariane 4 flights
  - The problem => certain assumptions changed between the Ariane 4 and the Ariane 5 and the software was not updated in response

© 2008, University of Colorado

Ariane 5, background info

- The flight software was written in Ada which has a first class exception construct
  - (it predates C++ and Java in this regard)
- If an exception is thrown but not caught, the error will “percolate” up through the call stack and will eventually terminate the entire system

© 2008, University of Colorado

Ariane 5, the details

- The failure of the Ariane 5 can be traced to the conversion of a 64-bit integer to a 16-bit signed integer
  - The 64-bit value was greater than 2^{15} which caused an exception to be generated
  - This exception was not caught and it caused the termination of the flight control software 37 seconds into the launch
  - The rocket shortly thereafter (3 seconds) lost control and was destroyed

© 2008, University of Colorado

More information

- Jean-Marc Jézéquel and Bertrand Meyer wrote a paper that traces the problem to an inappropriate reuse of a 10-year old software component
  - They reveal that one “vexing” aspect of this disaster is that the error occurred in a software system that was not needed during launch!
    - The calculation was supposed to be stopped 9 seconds before launch, but the inertial reference system had been reset during a hold in the countdown and its initialization sequence proceeded during launch.
      - This is what caused the rocket to veer off course… the initialization sequence was sending random sequences of 1s and 0s to the flight control software, which was interpreting them as commands to fire various sets of booster jets in completely random patterns!

© 2008, University of Colorado

Details, continued

- Their paper reveals that sufficient software dev. processes were in place and the system that caused the error had even been reviewed extensively before launch
  - exception handlers had been placed around 4 of 7 variables; unfortunately, the data conversion error occurred in one of the 3 unprotected variables
  - why leave 3 variables unprotected? Performance! If you add exception handling code, you slow the performance of the system
  - plus, the developers had an analysis that showed that overflow could not occur with the 3 unprotected variables
  - so they had good reason to leave them unprotected

© 2008, University of Colorado

Details, continued

- The problem?
  - The overflow analysis was conducted for the Ariane 4, not the Ariane 5
    - Its prediction that overflow could not occur for the three unprotected variables was no longer valid!
- So, it was a reuse error!

© 2008, University of Colorado

Ariane 5, summary

- The authors conclude that “Design by Contract” was needed in this situation
  - In particular, the component needed to specify a “contract” with its users; one aspect of this contract is specifying the legal input values
  - If the component had done something similar to an assert construct like this
    - proc foo(actual_value: int)
      - assert(actual_value <= maximum_value)
  - The authors argue that the error may well have been detected during system test; they further argue that such “contracts” should be a first-class, required programming language construct; not an optional construct that few use

© 2008, University of Colorado

5.10 Real System example
Ariane-5 Failure

- Jesequel and Meyer suggest that design by contract might have caught the Ariane-5
  - There was no precise specification for the component reuse from Ariane-4
- The code did not check the condition to check the variable representing horizontal bias that fit in 16 bits
- Had this condition been made explicit, it might have looked like

```
convert (horizontal_bias : DOUBLE): INTEGER is
    require
           horizontal_bias <= Maximum_bias
        do
         ......
    ensure
        ......
end
```


© 2006 Pearson/Prentice Hall

5.11 What This Chapter Means for you

- Looked at what it means to design a system
- Design begins at a high level, with important decisions about system architecture based on
  - system requirements
  - desirable attributes
  - the long-term intended use of the system
- Need to keep in mind the several characteristics as we build a design
  - Modularity and level of abstraction
  - Coupling and cohesion
  - Fault tolerance, prototyping and user interface

© 2006 Pearson/Prentice Hall