Chapter 8:
System Design

Key Ideas

- The purpose of the analysis phase is to figure out what the business needs. The purpose of the design phase is to figure out *how to provide it*

Key Ideas

- The steps in both analysis and design phases are highly *interrelated* and may require much "going back and forth"

Verifying and Validating (V&V) the Analysis Models

Walkthroughs

- Peer reviews of models and diagrams created during analysis
  - Conducted by teams of analysts, designers, and clients
  - Main purposes:
    - Test the fidelity of the models
    - Uncover errors or faults
- Potential danger is that analysts be punished for errors uncovered

Functional Model V&V

1. Events in Use Case descriptions should map to activities in the Activity Diagram
2. Object node in an activity diagram must be mentioned in Use Case descriptions
3. Sequential ordering within the Use Cases should match ordering in Activity Diagram
4. There must be a one-to-one correspondence of Use Cases in the Use Case Diagram and Use Case descriptions.

Functional Model V&V (cont'd)

5. All actors listed in a use case description must be portrayed on the use-case diagram
6. Include stakeholders listed in the use case description as actors in the use-case diagram
7. All relationships listed in a use-case description must be portrayed on a use-case diagram

Structural Model V&V

1. Every CRC card should be associated with a class on the class diagram
2. Responsibilities listed on the CRC card must be operations in a class on a class diagram
3. Collaborators on the CRC card imply some type of association on the class diagram
4. Attributes listed on CRC cards must be attributes in a class on a class diagram

Structural Model V&V (cont'd)

5. Class attributes with a type that is another class imply a relationship between classes
6. Relationships on the CRC cards must show up on the class diagram
7. Use association classes only if the association has unique attributes not on either class

Behavioral Model V&V

1. Actors & objects on sequence diagrams must be included on communication diagrams
2. Messages on sequence diagrams require associations on communications diagrams
3. Every message on a sequence diagram must appear as a message on an association in the corresponding communication diagram
4. Guard conditions messages in sequence diagrams require equivalent guard conditions on the corresponding communication diagrams

Behavioral Model V&V (cont'd)

5. The sequence number on message labels in communications diagrams must correspond to the top-down ordering of the messages being sent on the sequence diagram
6. State machine transitions must be associated with a messages on sequence & communication diagrams
7. All entries in a CRUD matrix imply a message being sent between an actor or object and another

Evolving the Analysis Models into Design Models

Factoring

- Creating modules that account for similarities and differences between units of interest

Factoring

- New classes
  - Generalization
  - Aggregation
- Abstracting
- Refinement

Layers

- Consider system environment information to help evolve the analysis model
- Model-view-controller (MVC) architecture
- Separating application logic from user interface logic

5 Layers

- Foundation
- Problem Domain
- Data Management
- Human-Computer Interaction
- Physical Architecture

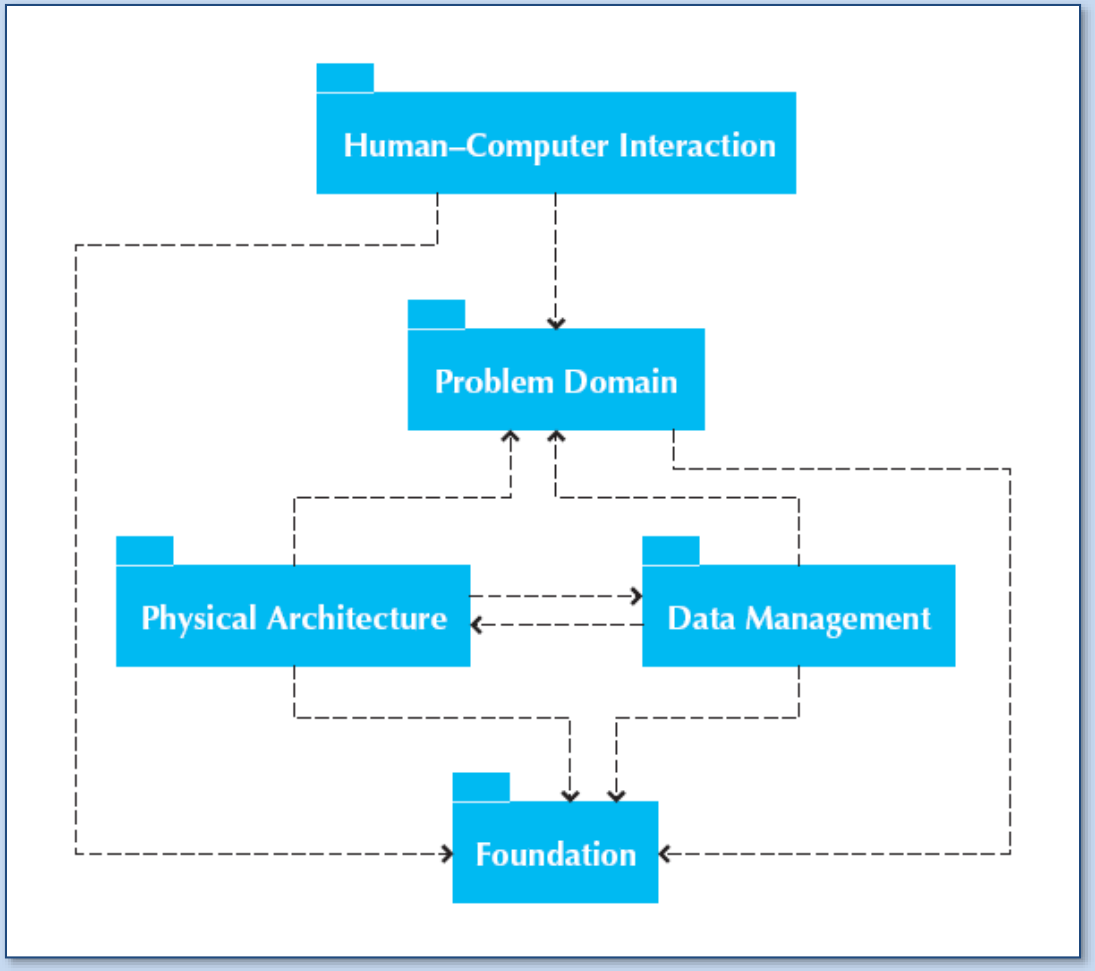
Packages and Package Diagrams

Package

- A general construct that groups units together
- Used to reduce complexity of models
- A package diagram shows packages only

A package: ■ Is a logical grouping of UML elements ■ Is used to simplify UML diagrams by grouping related elements into a single higher-level element	
A dependency relationship: ■ Represents a dependency between packages, i.e., if a package is changed, the dependent package also could have to be modified. ■ Has an arrow drawn from the dependent package toward the package on which it is dependent.	

Package Diagram for 5 Layers



Design Strategies

Custom Development

- Allows for meeting highly specialized requirements
- Allows flexibility and creativity in solving problems

Custom Development

- Easier to change components
- Builds personnel skills
- May tax firm's resources
- May add significant risk

Packaged Software

- Software already written
- May be more efficient
- May be more thoroughly tested and proven
- May range from components to tools to whole enterprise systems

Packaged Software

- Must accept functionality provided
- May require change in how the firm does business
- May require significant “customization” or “workarounds”

System Integration

- The process of combining packages, legacy systems, and new software
- Key challenge is integrating data
- Write data in the same format
- Revise existing data formats
- Develop “object wrappers”

Outsourcing

- Hire external firm to create system
- May have more skills
- May extend existing resources

Outsourcing

- Never outsource what you don't understand
- Carefully choose vendor
- Prepare contract and payment style carefully

Selecting a Design Strategy

- Business need
- In-house experience
- Project skills
- Project management
- Time frame

Selecting a Design Strategy

	Use Custom Development when…	Use a Packaged System when…	Use Outsourcing when…
Business Need	The business need is unique	The business need is common	The business need is not core to the business
In-house Experience	In-house functional and technical experience exists	In-house functional experience exists	In-house functional or technical experience does not exist
Project Skills	There is a desire to build in-house skills	The skills are not strategic	The decision to outsource is a strategic decision
Project Management	The project has a highly skilled project manager and a proven methodology	The project has a project manager who can coordinate vendor's efforts	The project has a highly skilled project manager at the level of the organization that matches the scope of the outsourcing deal
Time frame	The time frame is flexible	The time frame is short	The time frame is short or flexible

Developing the Actual Design

The Alternative Matrix

- Combines several feasibility analyses into one grid
- Revisits technical, economic, and organizational feasibility

Request for Proposals

- Description of the system you propose to be built
- Vendors, developers, service providers respond with proposals including how they will address needs as well as stating cost and time requirements.

Summary

- Verifying and Validating the Analysis Models
- Evolving the Analysis Models into Design Models
- Packages and Package Diagrams
- Design Strategies
- Developing the Actual Design

Chapter 9:
Class and Method Design

Design Criteria

Coupling

- Interdependency among modules
- Interaction coupling through message passing

Law of Demeter

Messages should be sent only by an object:
to itself
to objects contained in attributes of itself or a superclass
to an object that is passed as a parameter to the method
to an object that is created by the method
to an object that is stored in a global variable

Types of Interactive Coupling

Level	Type	Description
Good	No Direct Coupling	The methods do not relate to one another; that is, they do not call one another.
	Data	The calling method passes a variable to the called method. If the variable is composite (i.e., an object), the entire object is used by the called method to perform its function.
	Stamp	The calling method passes a composite variable (i.e., an object) to the called method, but the called method only uses a portion of the object to perform its function.
	Control	The calling method passes a control variable whose value will control the execution of the called method.
	Common or Global	The methods refer to a "global data area" that is outside the individual objects.
Bad	Content or Pathological	A method of one object refers to the inside (hidden parts) of another object. This violates the principles of encapsulation and information hiding. However, C++ allows this to take place through the use of "friends."

Source: These types were adapted from Meilir Page-Jones, *The Practical Guide to Structured Systems Design*, 2nd ed. (Englewood Cliffs, NJ: Yardon Press, 1988); and Glenford Myers, *Composite/Structured Design* (New York: Van Nostrand Reinhold, 1978).

Cohesion

- "Single-mindedness of a module"
- Method cohesion
- Class cohesion

Types of Method Cohesion

Level	Type	Description
Good	Functional	A method performs a single problem-related task (e.g., calculate current GPA).
	Sequential	The method combines two functions in which the output from the first one is used as the input to the second one (e.g., format and validate current GPA).
	Communicational	The method combines two functions that use the same attributes to execute (e.g., calculate current and cumulative GPA).
	Procedural	The method supports multiple weakly related functions. For example, the method could calculate student GPA, print student record, calculate cumulative GPA, and print cumulative GPA.
	Temporal or Classical	The method supports multiple related functions in time (e.g., initialize all attributes).
	Logical	The method supports multiple related functions, but the choice of the specific function is chosen based on a control variable that is passed into the method. For example, the called method could open a checking account, open a savings account, or calculate a loan, depending on the message that is send by its calling method.
Bad	Coincidental	The purpose of the method cannot be defined or it performs multiple functions that are unrelated to one another. For example, the method could update customer records, calculate loan payments, print exception reports, and analyze competitor pricing structure.

Source: These types were adapted from Page-Jones, *The Practical Guide to Structured Systems*, and Myers, *Composite/Structured Design*.

Types of Class Cohesion

Level	Type	Description
Good	Ideal	The class has none of the mixed cohesions.
↓	Mixed-Role	The class has one or more attributes that relate objects of the class to other objects on the same layer (e.g., the problem domain layer), but the attribute(s) have nothing to do with the underlying semantics of the class.
↓	Mixed-Domain	The class has one or more attributes that relate objects of the class to other objects on a different layer. As such, they have nothing to do with the underlying semantics of the thing that the class represents. In these cases, the offending attribute(s) belongs in another class located on one of the other layers. For example, a port attribute located in a problem domain class should be in a system architecture class that is related to the problem domain class.
Worse	Mixed-Instance	The class represents two different types of objects. The class should be decomposed into two separate classes. Typically, different instances only use a portion of the full definition of the class.

Source: Page-Jones, *Fundamentals of Object-Oriented Design in UML*.

Connascence

- Creating the need to change another module as a result of changing one
  - Minimize overall connascence
  - Minimize across encapsulation boundaries
  - Maximize within encapsulation boundary

Types of Connascence

Type	Description
Name	If a method refers to an attribute, it is tied to the name of the attribute. If the attribute's name changes, the content of the method will have to change.
Type or Class	If a class has an attribute of type A, it is tied to the type of the attribute. If the type of the attribute changes, the attribute declaration will have to change.
Convention	A class has an attribute in which a range of values has a semantic meaning (e.g., account numbers whose values range from 1000 to 1999 are assets). If the range would change, then every method that used the attribute would have to be modified.
Algorithm	Two different methods of a class are dependent on the same algorithm to execute correctly (e.g., insert an element into an array and find an element in the same array). If the underlying algorithm would change, then the insert and find methods would also have to change.
Position	The order of the code in a method or the order of the arguments to a method is critical for the method to execute correctly. If either is wrong, then the method will, at least, not function correctly.

Source: Meilir Page-Jones, "Comparing Techniques by Means of Encapsulation and Connascence" and Meilir Page-Jones, *Fundamentals of Object-Oriented Design in UML.*

Object Design Activities

Additional Specification

- First review the current set of models
  - Sufficient but only necessary classes to solve problem
  - No missing attributes or methods
  - No extraneous attributes or methods
- Examine visibility

Signatures for each method

- Name of the method
- Parameters or arguments to pass
- Type of value the method will return to the calling method

Define constraints

- Pre-conditions
- Post conditions
- Invariants
- How to handle violations (exceptions in C++ and Java)?

Identify Opportunities for Reuse

- Patterns
- Framework
- Class libraries
- Components

Restructure the Design

- Factoring
- Map design to current language
- Normalization
- Assure all inheritance relationships support only generalization/specialization semantics

Optimizing the Design

- Review access paths
- Review attributes of each class
- Review direct and indirect fan-out

Optimizing the Design

- Consider execution order of statements in often-used methods
- Avoid re-computation by creating derived attributes and triggers

Chapter 10:
Database Design

The Data Management Layer

- Includes both
  - data access and manipulation logic, and
  - the actual design of the storage
- Four-step design approach
  1. Selecting the format of the storage
  2. Mapping problem-domain objects to object-persistence format
  3. optimizing the object-persistence format
  4. designing the data access & manipulation classes

Object Persistence Formats

- Files (Sequential and Random)
- Relational databases
- Object-relational databases
- Object-oriented databases

Chapter 11:
User Interface Design

Overview

- Interface Design formalizes the interaction of the system with external entities
  - *System Interfaces* are machine-machine and are dealt with as part of systems integration
  - *User Interfaces* are human-computer and are the focus of this chapter

Principles of User Interface Design

- Layout
- Content Awareness
- Aesthetics
- User Experience
- Consistency
- Minimal User Effort

General Layout

Navigation Area

Reports & Forms Area

Status Area

Layout

- Each area may be further subdivided
- Each area is self-contained
- Areas should have a natural intuitive flow
  - Users from western nations tend to read from left to right and top to bottom
  - Users from other regions may have different flows

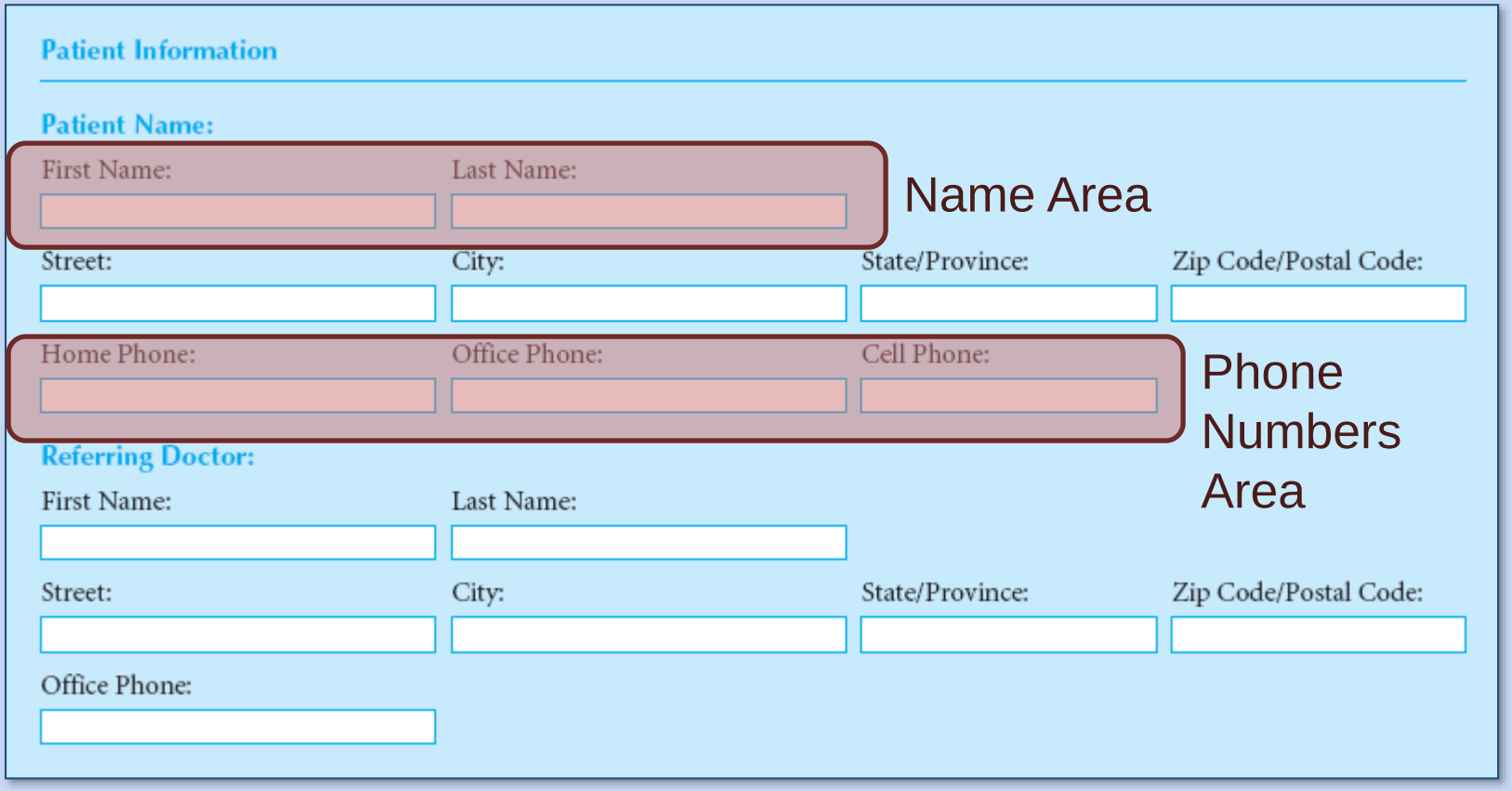
Content Awareness

- Intuitively answers the users' questions:
  - Where am I?
  - What am I supposed to be doing here?

Content Awareness

- Content awareness applies to sub-areas within a form or window
  - Related form fields (e.g. address information) are grouped together
  - Related report information (e.g. records) are grouped together

Form Content Awareness



Report Content Awareness

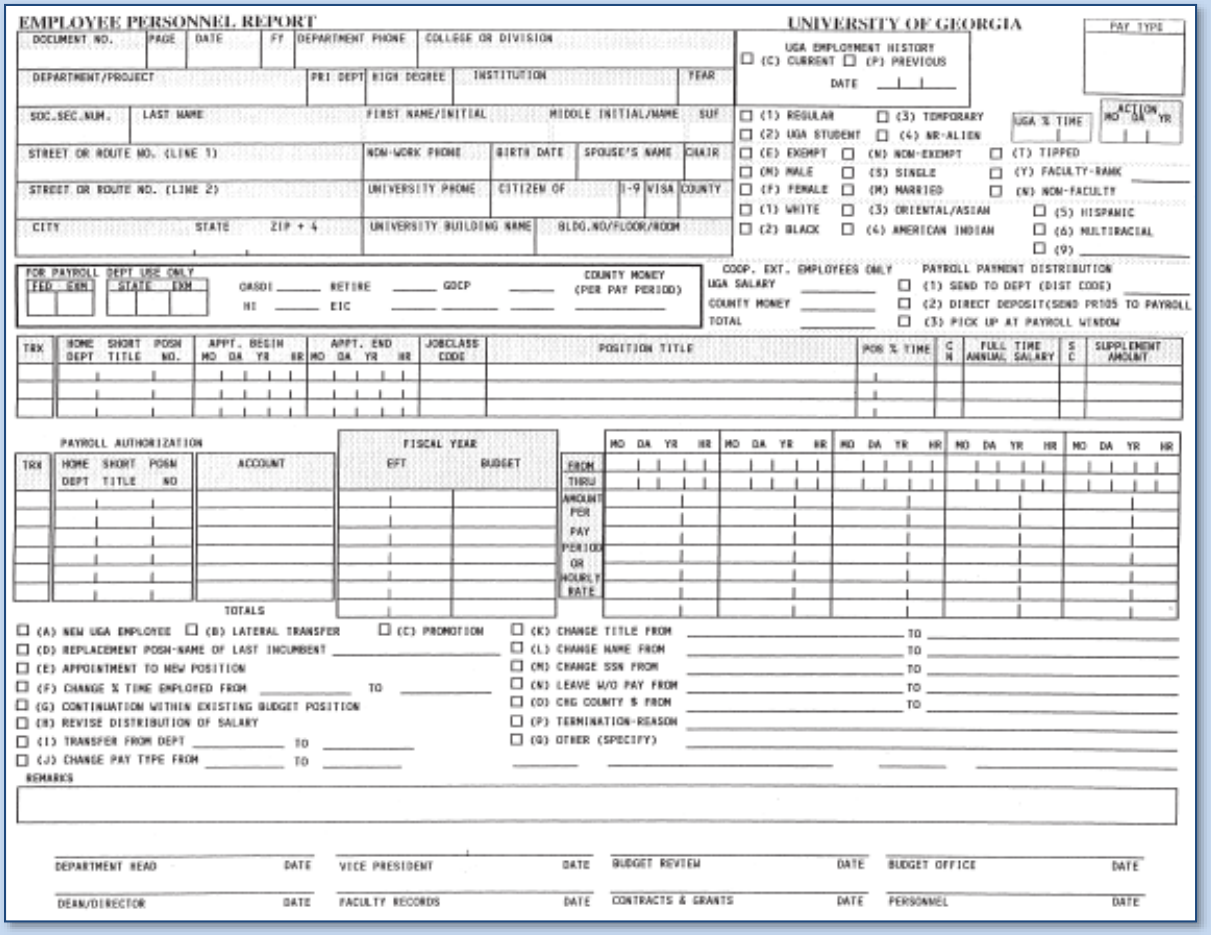
Aesthetics

- Interfaces should be functional, inviting to use, and pleasing to the eye
- In most cases, less is more (minimalist design)

Aesthetics

- White space is important
- Acceptable information density is proportional to the user's expertise
  - Novice users prefer less than 50% density
  - Expert users prefer more than 50% density

Bad Aesthetics



EMPLOYEE PERSONNEL REPORT

UNIVERSITY OF GEORGIA

DOCUMENT NO. | PAGE | DATE | FY | DEPARTMENT PHONE | COLLEGE OR DIVISION

DEPARTMENT/PROJECT | PRI DEPT | HIGH DEGREE | INSTITUTION | YEAR

UGA EMPLOYMENT HISTORY
☐ (C) CURRENT ☐ (P) PREVIOUS
DATE

PAY TYPE

SOC.SEC.NUM. | LAST NAME | FIRST NAME/INITIAL | MIDDLE INITIAL/NAME | SUF

STREET OR ROUTE NO. (LINE 1) | NON-WORK PHONE | BIRTH DATE | SPOUSE'S NAME | CHAIR

STREET OR ROUTE NO. (LINE 2) | UNIVERSITY PHONE | CITIZEN OF | I-9 | VISA | COUNTY

CITY | STATE | ZIP + 4 | UNIVERSITY BUILDING NAME | BLDG.NO/FLOOR/ROOM

UGA % TIME

ACTION MO DA YR

☐ (1) REGULAR ☐ (3) TEMPORARY
☐ (2) UGA STUDENT ☐ (4) NR-ALIEN
☐ (E) EXEMPT ☐ (N) NON-EXEMPT ☐ (T) TIPPED
☐ (M) MALE ☐ (S) SINGLE ☐ (Y) FACULTY-RANK
☐ (F) FEMALE ☐ (M) MARRIED ☐ (N) NON-FACULTY
☐ (1) WHITE ☐ (3) ORIENTAL/ASIAN ☐ (5) HISPANIC
☐ (2) BLACK ☐ (4) AMERICAN INDIAN ☐ (6) MULTIRACIAL
☐ (9)

FOR PAYROLL DEPT USE ONLY
TED | ERN | STATE | EXM
OASDI | RETIRE | GDCP | COUNTY MONEY (PER PAY PERIOD)
HI | EIC

COOP. EXT. EMPLOYEES ONLY
UGA SALARY
COUNTY MONEY
TOTAL

PAYROLL PAYMENT DISTRIBUTION
☐ (1) SEND TO DEPT (DIST CODE)
☐ (2) DIRECT DEPOSIT(SEND PR105 TO PAYROLL
☐ (3) PICK UP AT PAYROLL WINDOW

TRX	HOME DEPT	SHORT TITLE	POSN NO.	APPT. BEGIN MO DA YR HR	APPT. END MO DA YR HR	JOBCLASS CODE	POSITION TITLE	POS % TIME	C H	FULL TIME ANNUAL SALARY	S C	SUPPLEMENT AMOUNT

PAYROLL AUTHORIZATION

TRX	HOME DEPT	SHORT TITLE	POSN NO	ACCOUNT	FISCAL YEAR EFT	FISCAL YEAR BUDGET
				TOTALS		

	MO DA YR HR	MO DA YR HR	MO DA YR HR	MO DA YR HR	MO DA YR HR
FROM					
THRU					
AMOUNT PER PAY PERIOD OR HOURLY RATE					

☐ (A) NEW UGA EMPLOYEE ☐ (B) LATERAL TRANSFER ☐ (C) PROMOTION
☐ (D) REPLACEMENT POSN-NAME OF LAST INCUMBENT
☐ (E) APPOINTMENT TO NEW POSITION
☐ (F) CHANGE % TIME EMPLOYED FROM TO
☐ (G) CONTINUATION WITHIN EXISTING BUDGET POSITION
☐ (H) REVISE DISTRIBUTION OF SALARY
☐ (I) TRANSFER FROM DEPT TO
☐ (J) CHANGE PAY TYPE FROM TO

☐ (K) CHANGE TITLE FROM TO
☐ (L) CHANGE NAME FROM TO
☐ (M) CHANGE SSN FROM TO
☐ (N) LEAVE W/O PAY FROM TO
☐ (O) CNG COUNTY $ FROM TO
☐ (P) TERMINATION-REASON
☐ (Q) OTHER (SPECIFY)

REMARKS

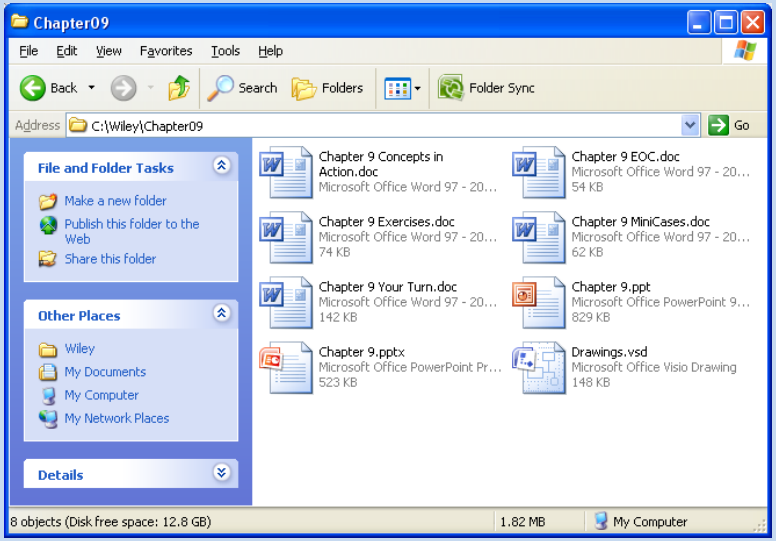
DEPARTMENT HEAD DATE | VICE PRESIDENT DATE | BUDGET REVIEW DATE | BUDGET OFFICE DATE

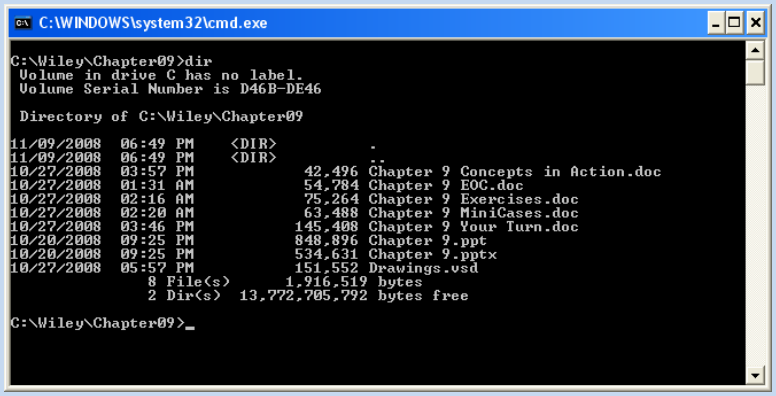
DEAN/DIRECTOR DATE | FACULTY RECORDS DATE | CONTRACTS & GRANTS DATE | PERSONNEL DATE

User Experience

- Ease of learning
  - Significant issue for inexperienced users
  - Relevant to systems with a large user population
- Ease of use
  - Significant issue for expert users
  - Most important in specialized systems
- Sometimes ease of learning and use of use go hand in hand

Multiple Interfaces





- Microsoft Windows has multiple interfaces for the same functionality
- Most users prefer to use Windows Explorer for handling files
- Expert users sometimes prefer the command line interface

Consistency

- All parts of the system work in the same way
- Key areas of consistency are
  - Navigation controls
  - Terminology

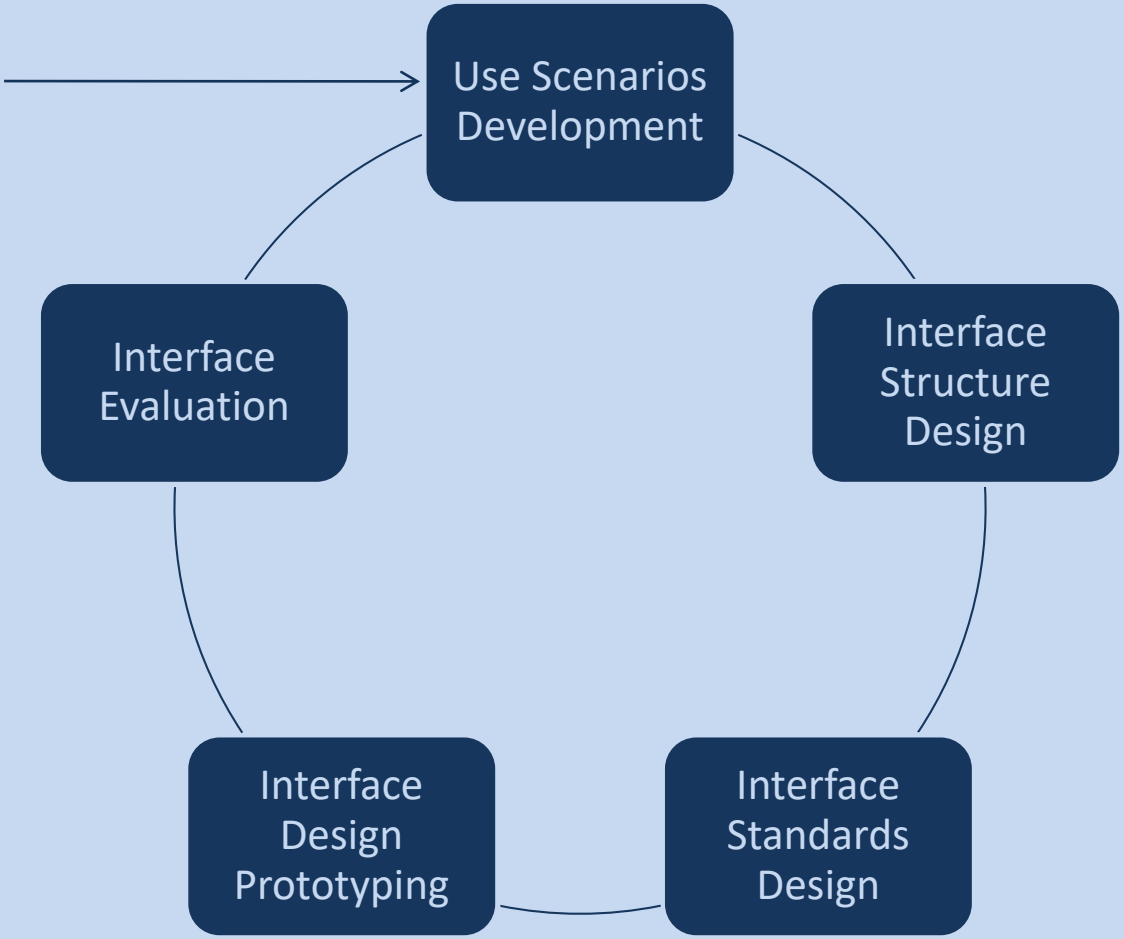
Consistency

- Probably most important concept in making the system simple because it allows the users to *predict what is going to happen*

Minimal User Effort

- Interfaces should be designed to minimize the effort needed to accomplish tasks
- A common rule is the tree-clicks rule
  - Users should be able to go from main menu of a system to the information they want in no more than three mouse clicks

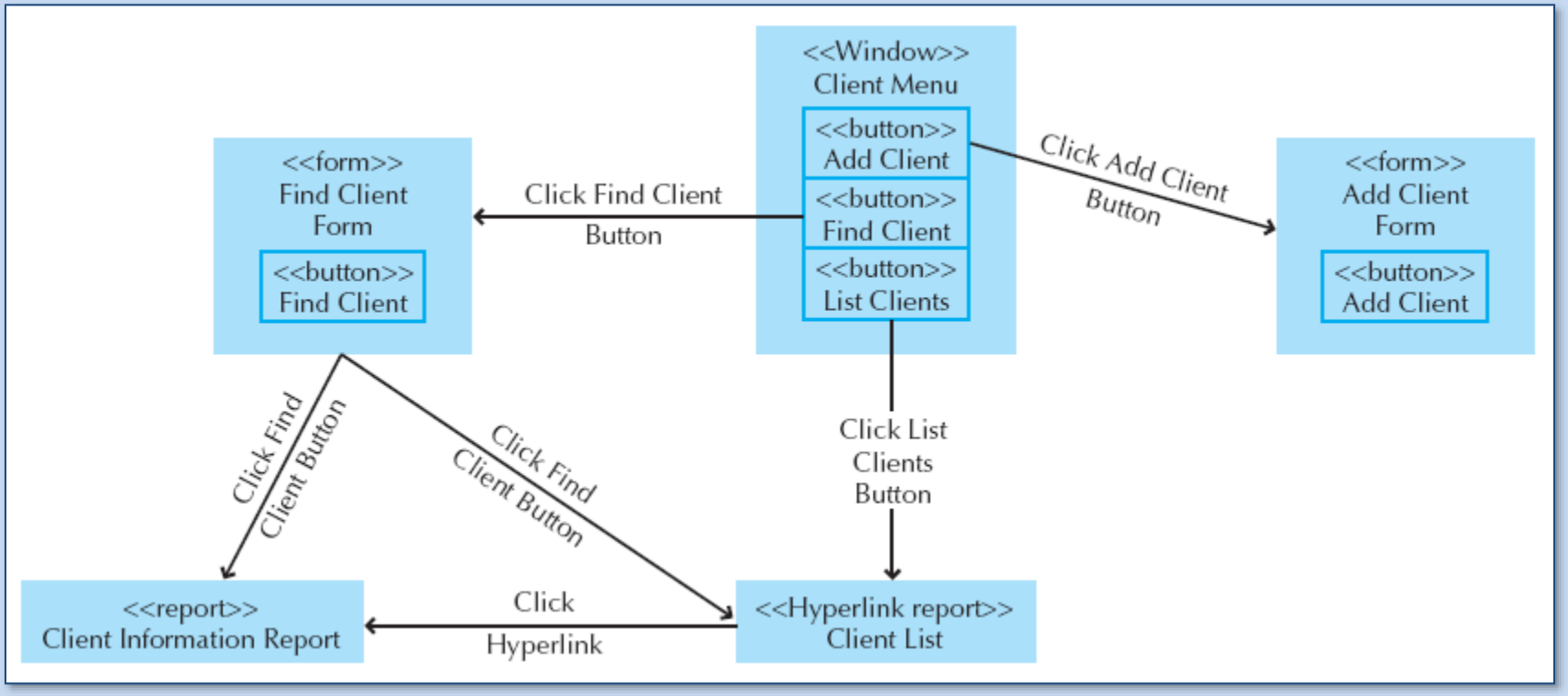
5-Step UI Design Process



Windows Navigation Diagrams

- Like a state diagram for the user interface
  - Boxes represent components
    - Window
    - Form
    - Report
    - Button
  - Arrows represent transitions
    - Single arrow indicates no return to the calling state
    - Double arrow represents a required return
  - Stereotypes show interface type

Sample WND



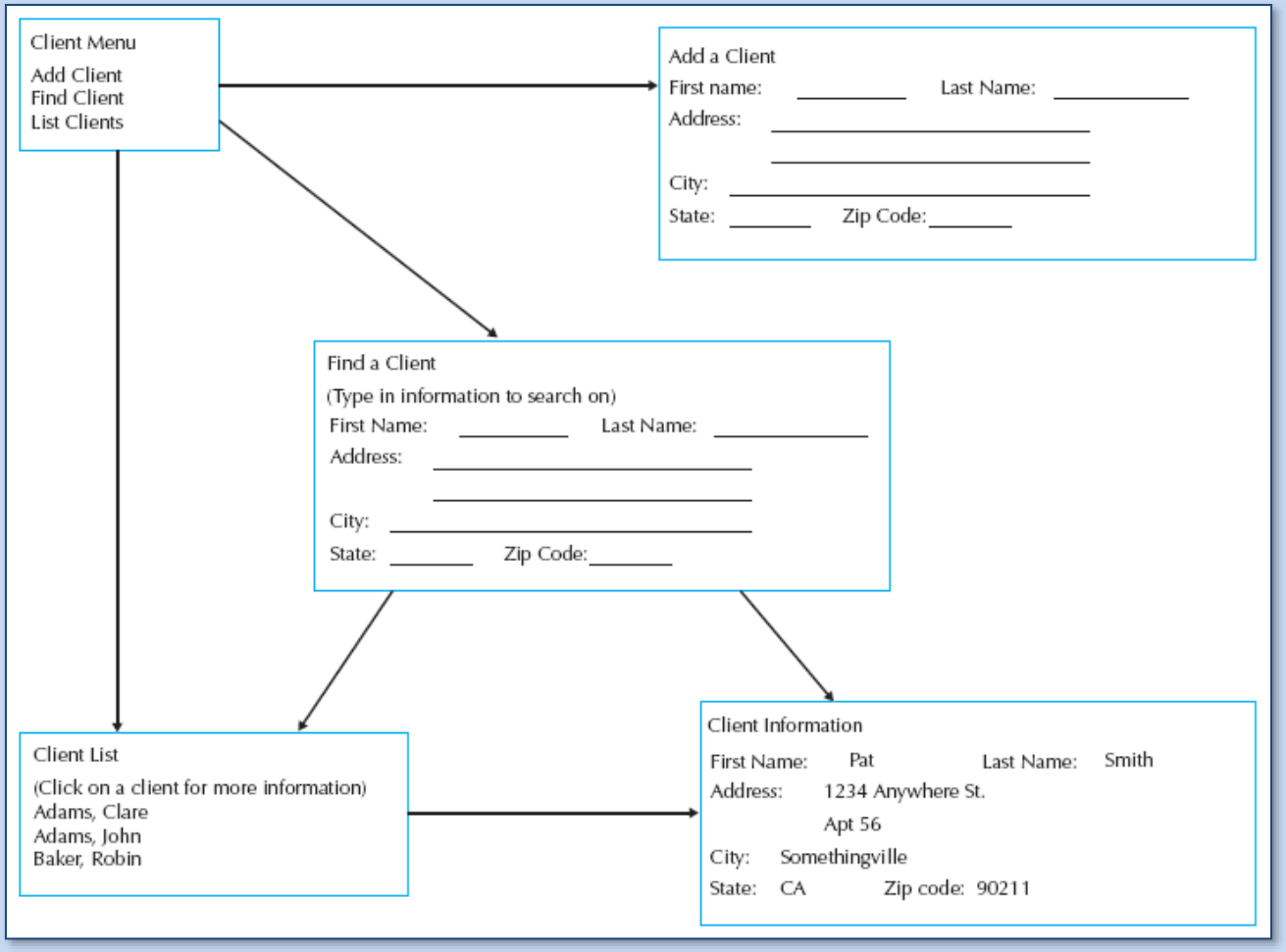
Interface Standards Design

- Interface standards are basic design elements found across the system user interface
- Standards are needed for:
  - Interface metaphor
  - Interface objects
  - Interface actions
  - Interface icons
  - Interface templates

Interface Design Prototyping

- Mock-ups or simulations of computer screens, forms, and reports
- Four common approaches
  - Storyboard
  - Windows layout diagram
  - HTML prototype
  - Language prototype

Sample Storyboard



Interface Evaluation

- Goal is to understand how to improve the interface design before the system is complete
- Have as many people as possible evaluate the interface
- Ideally, interface evaluation is done while the system is being *designed—before it is built*

Chapter 12: Architecture

Introduction

- Most modern systems span two or more networked computers
- The physical architecture layer design specifies
  - How the system will be distributed across the computers
  - What hardware and software will be used
- Most systems' design is constrained by existing systems and networks

Elements of the Physical Architecture Layer

Architectural Components

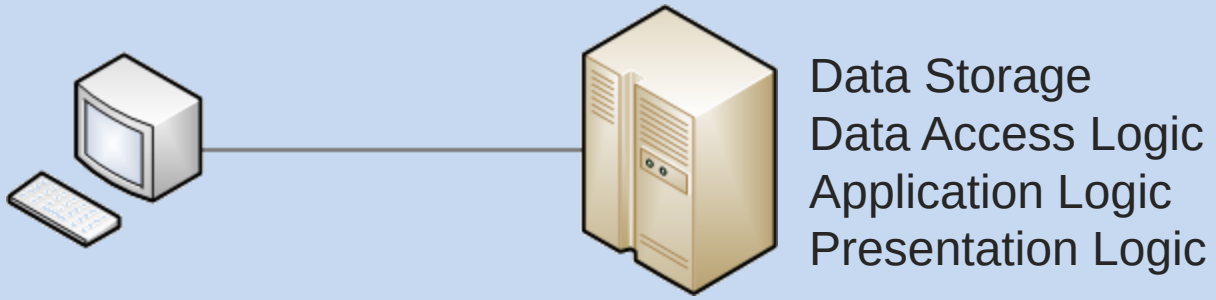
- Software components
  - Data storage
  - Data access logic
  - Application logic
  - Presentation logic

Architectural Components

- Hardware components
  - Computers (clients, servers)
  - Networks

Server-Based Architectures

- The server performs all four application functions
- The client only needed a monitor, a keyboard, and a communications device (e.g. modem)



Client-Based Architectures

- All logic resides on the client computer
- A separate computer may hold the data
- Simple to develop, but difficult to maintain

Client-Server Architectures

- Balance processing between client and server
- Predominant architecture in modern systems
- Amount of client processing varies
  - Thin clients do only presentation logic
  - Thick clients do presentation and application

Client-Server Tiers

- Client server architectures can have two or more tiers depending on application logic partitioning
  - 2-tier: all application and data logic on one server
  - 3-tier: application logic on one server, data logic on another
  - 4-tier: application logic split among two servers, data logic on another
- Sometimes called an *n*-tier architecture

Distributed Objects Computing

- Next generation of client-server computing
- The client need not know which server to call
- Middleware intercepts the client request and sends it to the appropriate server

Distributed Objects Computing

- Three competing approaches
  - CORBA
  - Enterprise Java Beans
  - .NET

Selecting a Physical Architecture

- Cost of infrastructure
- Cost of development
- Ease of development
- Interface capabilities
- Control and security
- Scalability

Architecture Characteristics

	Server-Based	Client-Based	Client-Server
Cost of infrastructure	Very high	Medium	Low
Cost of development	Medium	Low	High
Ease of development	Low	High	Low-Medium
Interface capabilities	Low	High	High
Control and Security	High	Low	Medium
Scalability	Low	Medium	High

Infrastructure Design

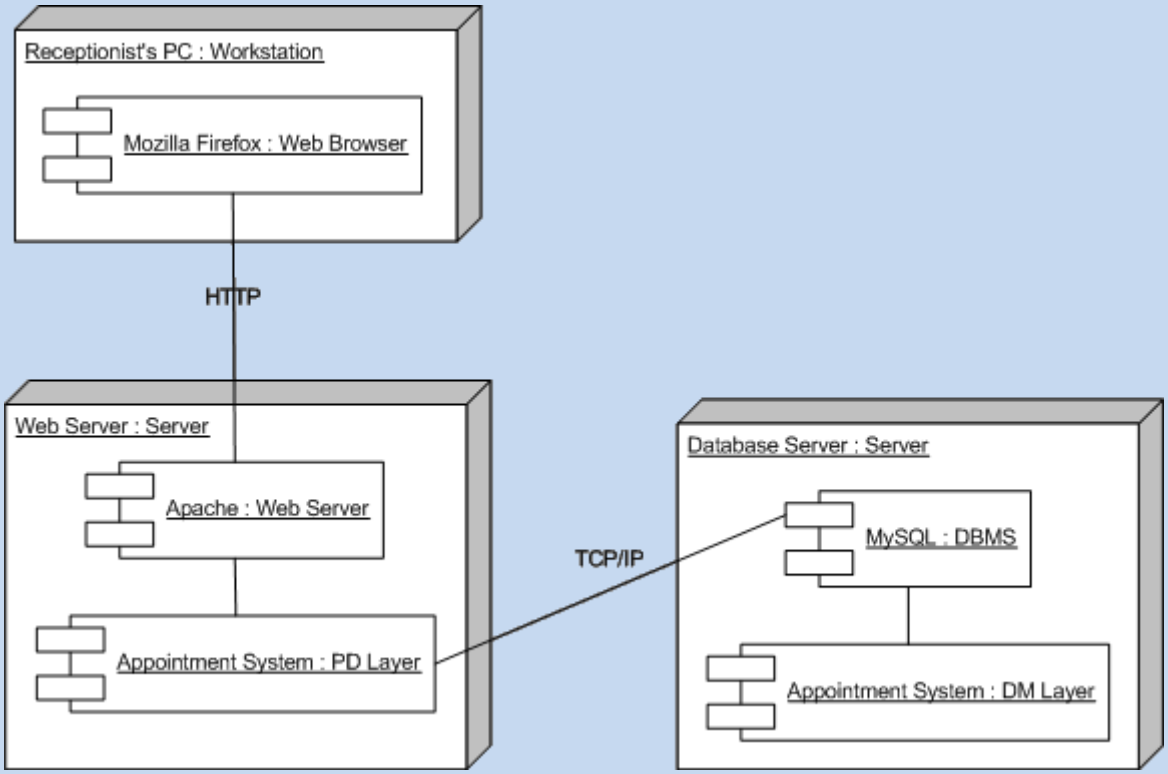
Deployment Diagram

- Represent relationships between hardware components of an information system

Deployment Diagram

- Elements of a deployment diagram
  - Nodes: a computational resource
  - Artifacts: a piece of the information system which will be installed on a node
  - Communication paths: a communication link such as a network connection or a USB cable

Sample Deployment Diagram



Network Model

- Shows the major components of the information system and their geographic locations throughout the organization

Network Model

- Purposes of the network model:
  - To convey the complexity of the system
  - To show how the system's software components will fit together

Sample Network Model