Class announcements
- Database Project due Today
- Reading for Tuesday (Nov 29th):
  - American Airlines Case
  - Student Presentations Tuesday (Nov 29th)
    - Jennifer DeBoer: American Airlines case
  - Business Paper Presentations
    - Natalie Barroales
    - Kirk Ewell
    - Francesco Gallerani
    - Aileen Lu
    - Amanda Ho
    - Daniel Gonzalez
    - Blanca Rosales Alcaraz

We will enforce a strict 5 minute time limit for all talks!

Student Presentations
- Romiro Gomez
- Jonathan Bermudas
- Danny Macias
- Alfred Lee
- Hoang Dinh: Business Paper Presentation

Domain Names
IP addresses are inconvenient for people
- 32 bits hard to remember
- 128 bits very hard to remember

Domain names
- e.g. argus.eecs.berkeley.edu

- Easier to remember than IP addresses
- However, we need some way of mapping domain names to IP addresses.

Domain Name System (DNS)

Hierarchy in Addresses vs. Names
Addresses hierarchical in topology
- Maximize "wild cards" and distribute address administration

Names hierarchical in administration
- Single administered organizations often distributed topologically (e.g. ibm.com)
Chapter 9

Build vs. Buy?

Purchase off the Shelf
- less time and cost
- benefits of using a "standard" solution
- support available
- must mold org to app
- no potential for competitive advantage

Outsource
- developers not as familiar with org as you
- more opportunity for customizing than off the shelf
- contractor may share knowledge with competitors
- contractor may have too much bargaining power

Make
- most customizable of 3
- easier iteration between conceptualization and development needed
- most risky
- org may lack competency to do it

Application Lifecycle

It is important to think beyond acquiring an application
- How do we come with the idea?
- How do we architect it?
- How do we implement?
- How do we extend and maintain it?

For this reason, the software engineering community came up with:
- Application Lifecycle Model

1) Conceptualization

What is the vision?
- What are the objectives?
- What is the business case?

- EXAMPLE: HHC to inform flight attendants which passengers are low and high value.
  - Present diagram to FA's
  - HHC customer info updated wirelessly at gate
  - Also has reporting function for misbehaving passengers.

- Business Case:
  - Increase repeat business from high value customers.

Application Lifecycle

Stages:
1. Conceptualization
2. Analysis
3. Architecture Design
4. Development Evolution
5. Testing and Evaluation
6. Deployment
7. Operations, Maintenance, and Upgrade

1) Conceptualization -- Example:
2) Analysis

- Describe what the application will do.
- Enough info to allow "stakeholders" to review idea
- Don't make highly detailed specifications
- Describe scenarios in which it is used
  (Use Cases)

2) Analysis -- Example

- Example
  NORMAL FUNCTION
  - When at gate, WiFi AP sends pass. data of next flight to HHC
  - HHC displays info on color coded seat map
  - If FA clicks on seat she gets more info about passenger
  REPORTING FUNCTION
  - FA wants to report that passenger in 13F is bad.
  - FA clicks "report pass." button followed by 13f
  - HHC finds from its data that Joe Schmoe is in 13f
  - When HHC is in radio range of WiFi AP, HHC tells server that Joe Schmoe is bad.

3) Architecture Design

- Decompose the application into subsystems
  - Hardware, software
  - Try use commercial off the shelf subsystems
  - Try to use standard infrastructure layers
    - Operating system, network, middleware, etc.

Architecture

Design hierarchical architecture.

HHC Architecture

- HHC Application
  - Palm OS
  - Networking Infrastructure
- Coordination (With HHC Server)
- User Interface
  - Data Management

HHC Server

- HHC Server Application
  - Windows OS
  - Networking Infrastructure
- Computation of key statistics
- Communication with HHC
  - Communication with airline database
  - Standard Database "queries" (SQL) relayed to DBMS via OS and infrastructure

Again, we see layering and hierarchy. Between each module we specify an interface.
3) Architecture Continued

- Define the functionality, interaction and interfaces of subsystems
- Consider
  - Scalability
    - How easily can we increase the number of users and maintain performance?
  - Extensibility
    - How easily can we add new features in the future?
  - Administration
    - How much work will it take by humans to keep this running properly?
    - (Remember Sun thin vs fat client discussion)

4) Development Evolution

- Develop the details
  - Develop/program custom subsystems
  - Have contractor build outsourced pieces
  - Put together with off-the-shelf components
- Incremental
  - Start with simplest implementation and get it working
  - Later add more features.

5) Testing

- A must!
- If architected well, we can test subsystems independently.
- Alpha test - offline test of prototype
- Beta test - test in intended environment with cooperative users
  - Example - give hhc to initial group of FA's

6) Deployment

- Convert from previous processes if necessary
  - Example: CISCO ERP (all at once)
  - Or, you could do incrementally
- Train users
  - Example: Frito-Lay HHC
- Data importation
  - (if necessary)

7) Operations, Maintenance, Upgrade

- Maintain Security
- Repair Problems
- Correct performance short commings (Cisco ERP)
- Add features
Application Lifecycle Model
concluding remarks

- ALM rarely followed precisely
- Many times projects loop between stages
- ALM followed more closely in larger companies
- Alternative:
  - Rapid Iterative Prototyping
    - (Cisco did some of this in the ERP case.)

Chapter 10 - Application Architecture

Application Architecture

- Decomposition - Divide the architecture into interacting modules.
- Assembly - Find subsystems available for purchase

Most architecture design is a mixture of decomposition and assembly.

Decomposition Example

- Example: manage bank accounts
- Decompose into software modules for
  - transaction processing,
  - statement generation
- Further decompose transaction processing module into deposit and withdraw modules...

Assembly Example

- example - ecommerce platform
  - Acquire
    - Linux pc (application server)
    - IBM Mainframe (data server)
    - Oracle DBMS
    - Apache Web Server Software
    - Websphere application middleware
  - Assemble all pieces together.
  - Mix with custom developed application logic module.

Object-Oriented Architectures

- Object-Oriented Programming (OOP)
  - Languages
    - C++
    - Java
    - Smalltalk
  - The basic unit of modularity in OOP is an object.
Objects

- Example: Bank account
  - has a balance of $5000
  - belongs to Joe Schmoe
  - is a checking account
- can have money deposited to it
- can have money withdrawn from it

- Attribute
- Behavior

An attribute is a numerical value or data that is externally visible, and may be changeable.
- Ex: The bank account's balance is $5000

A method is an action available at the object interface.
- Other objects invoke method, pass parameters and get returned data or other objects.
- We can invoke the "check_balance" method and get returned the number $5000

Object Classes and Instances

- Some objects share types of attributes and methods.
- They have the same class
- Example
  - Class: Bank_Account
  - Instances:
    - Schmoe_Account
    - Smith_Account
- Each instance is a separate object with its own data

Method Invocation

- Objects communicate with each other by invoking each other's methods
  - (method invocation)

Software Objects

- In OOP an object can
  - Represent a real world entity
  - Bank account
  - Be a proxy of a real world entity
  - Proxy of a customer
  - Other software talks to proxy using method invocations
  - Model a real-world entity
    - For purposes of simulation
    - Motion of a train

ORMDBMS

- Earlier in the class we talked about relational DBMS
- The most common database management system that organizes data into tables.

- ORDBMS (Object Relational DBMS)
  - Retrieve and store object instance data in a relational database
Remote Method Invocation

- Sometimes we want to allow an object to invoke methods on an object on another machine.
- This is called Remote Method Invocation (RMI)
- Doing this requires middleware called Distributed Object Management (DOM)

Software Reuse

- Size and complexity of applications growing dramatically
- In order to contain costs, we need to be able to reuse pieces of software
- Reuse is difficult. Why?
- OOP was developed in part to promote re-use, but has had limited success in that regard.

Software components

- Software components are reusable modules that can be bought from outside vendors.
- How is a component different from an object?
  - More importance on
    - Encapsulation
    - Well defined and documented interfaces

Component Assembly Tools

- Visual or integrated development environment (IDE)
  - MS Visual Studio
  - IBM Visual Age
  - Symantec Visual Café
- Scripting Assembly - Text based
  - TCL
  - Perl
  - JavaScript

Software Frameworks

- A preexisting architecture and library of components from a common vendor to help developers
- Enables reuse, and ensures component interoperability.
- Examples:
  - Sun J2EE/Java Beans
  - Microsoft .Net