

# CSCI0320

# Introduction to Software Engineering

## Lecture 12

## Web Application Design

# Modern Applications

- **What applications in the past 5 years do you see as the most promising, exciting, interesting, forward looking, ...?**
  - **IM**
  - **Google Maps**
  - **Google Streets**
  - **Amazon shopping**
  - **Ebay**
  - **Napster and its successors**
  - **iTunes**
  - **Skype**
  - **Wikipedia**

# Future Applications

- **What will future applications be like?**
  - **Ubiquitous computing**
    - Work on you cell phone, in your car, on your pda, as well as at home or at work
  - **Globally shared data**
    - Map data, technical papers, music, opinions, wikipedia
  - **Instant multimedia communication**
    - Telecommuting, distance learning
  - **Other trends???**

# Themes

- **What is common among these apps?**
  - **Built to work over the Internet**
    - Servers and clients
    - Use central data stores, distributed computing
  - **Built to work with web browsers**
    - Web-based front ends
    - Accommodate different web browsers
  - **Built to work at Internet scales**
    - Large numbers of concurrent users

# Normal Applications

- **User interface integrated with application**
  - High quality, tightly bound
- **Application reads/writes its own data**
  - Directly to files
  - Using a database system
- **Single user**
  - Or small number of communicating users

# Client-Server Applications

- **Splits this into components**
  - Front end handles the user interface
  - Back end handles the computation and data
  - Both are applications themselves
- **Communication using sockets**
  - Using application protocols (xml)
  - Using standard protocols (soap, rpc, rmi)
- **Multiple user**
  - 10-100 users typical

# Web Applications

- Like client-server applications
  - Separate front and back ends
- Differences
  - Restrictions imposed by the web
  - User interface is the browser
    - Any browser
    - Restrictions imposed by the browsers
  - Handle 1000's of users

# Browser Restrictions

- **Who can the browser talk to**
  - **In general, any program on the machine**
    - But this requires applets (or plugins)
    - And has severe security & privacy concerns
    - And may not work with today's firewalls, etc.
  - **In practice, with a web server**
    - On the machine at the user's specified URL
    - But the web server is not the application
      - Its only an intermediary
    - Only using **limited set of protocols** (http,...)

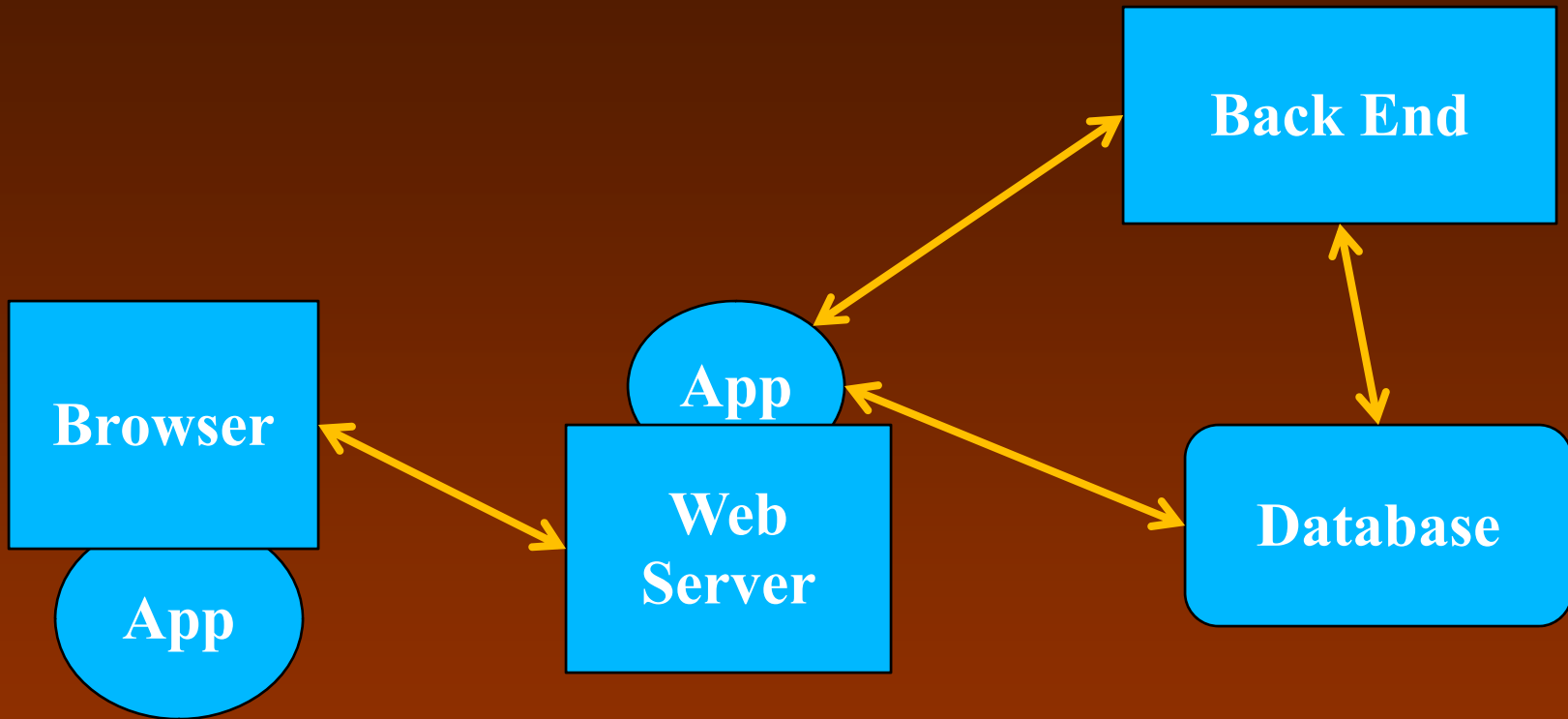
# Web App Restrictions

- **What code can the browser run**
  - Limited amount of user code
    - JavaScript
    - Restricted domain (no file access)
  - Must be portable
  - Must be safe
- **What code can the server run**
  - Depends on the server and the URL

# Web Protocol

- **Browser requests a page (or a frame)**
  - Using Http POST or GET
    - With parameters either in the URL (?x=a&y=b)
    - Or via post data
- **Server sends down a new page**
  - Replacing the current page (or frame)
  - Formatted using HTML (XHTML)
- **How does this compare to other models**
  - What does the front end look like?
  - Who does what?
  - How to talk to the actual back end?

# Web Architecture

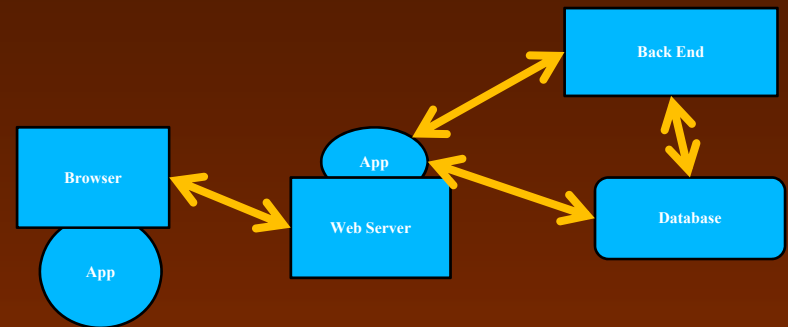


# Web App Issues

- **How to get code into the web browser**
- **How to get code into the web server**
- **How to get the web server to talk to the application**
- **How to get the browser to talk to the application**
- **How to get the application to talk HTML to the user**

# Client-Side Code

- **Different options**
  - JavaScript
  - Flex
  - Flash
  - Vbscript
  - Silverlight
  - Applets
  - Java web start
  - GWT



# Applets

- **Arbitrary java programs**
  - Run in and by the browser
  - **Sandbox** environment
    - Limited file access
    - Limited socket access
    - Limited access to properties, etc.
- **Pros**
  - Can use swing, etc. for fancy front ends
  - Can be highly interactive
- **Cons**
  - Might not run on all (most) browsers
  - Users might disallow
  - Long download times

# JavaScript

- **Scripting** programming language
  - Java-like syntax, but **NOT Java**
  - GWT lets you program this in Java
  - No declarations
  - Dynamic types (strings, numbers, objects)
  - Tacked on object model
- **Ability to hook to browser events**
  - Callbacks on input, mouse over, select, ...
- **Functions talk to the browser**
  - Request pages, set parameters
  - Change properties of current page

# Client-Side Processing

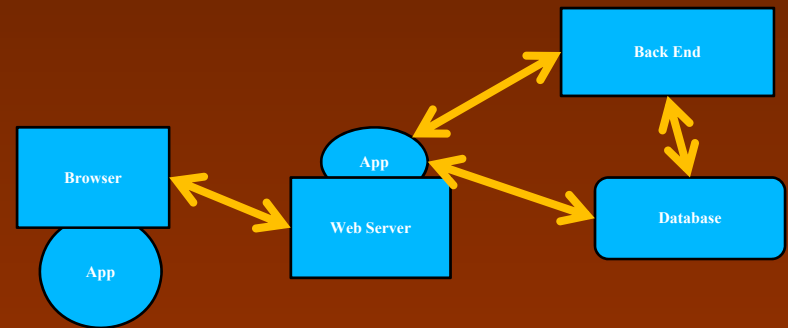
- **Browsers display HTML**
  - To do this, they build a **tree**
    - Corresponding to the HTML structure
    - This is represented in memory as the **DOM**
- **Client code (javascript) can modify this**
  - Effectively modifying the page dynamically
  - Closer to what a standard UI does
- **JavaScript can send requests to the server**
  - Without updating the page (hidden frames)
  - Can interpret the result as a message
    - XML data for example

# AJAX

- **Asynchronous JavaScript And XML**
  - Web page with JavaScript
  - JavaScript sends requests to web server
  - Response is an XML message
  - JavaScript reads the XML
  - JavaScript modifies the DOM to update
- **Pros and Cons**
  - Highly Interactive interfaces
  - Dependent on web browser
  - Code is public, security a concern

# Server-Side Code

- Different Options
  - CGI programs
  - Servlets
  - Java (Active) server pages
  - Java beans
  - Scripting languages
    - Embedded JavaScript
    - Embedded PHP
    - Ruby, python, ...



# CGI Programs

- **URL: http://host/cgi-bin/cmd?args**
  - Cgi-bin is a directory of the web server
  - Cmd is a normal executable file
    - Runnable on the server
    - Shell script, perl, php, python, Java, C, C++, ...
  - Args are named arguments passed to cmd
    - Spaces converted to +
    - Escapes for other character
    - Passed in the environment
- **Cmd is run on the web server**
  - Is this secure?
- **Typical use: format and pass on request**
  - Is this efficient?

# Servlets

- URL: <http://host/servlet/name?args>
- **Pieces of Java code that run in the server**
  - Just as applets are run in the browser
  - Dynamically loaded code
    - From a standard directory
    - Provided with access to the arguments
- **Run in a security sandbox**
  - Why would you be concerned here?
  - Limited access to files, ports, ...
- **Requires special server (tomcat)**
- **GWT generates these as well**

# Java Server Pages

- **Normal HTML page with .jsp extension**
  - Active server pages: .asp extension
- **HTML with embedded Java code**
  - **Preprocessor** reads the page
    - HTML passed along directly
    - Java code in the page is executed
    - The code is replaced by its output
  - Can include and access arbitrary Java packages, etc.
    - **Again with security-based limitations**

# Java Server Pages

- **Cleaner than using servlets**
  - Same power
  - Easier to generate HTML pages
- **Java isn't always the ideal language here**
  - Not the best string handling capabilities
  - Everything coming in is a string
    - Continually need to convert to internal types
    - Need to generate complex output strings
      - With large sections of fixed data
    - Not well aligned with Java
- **Concept of embedding is useful**
  - Perl, javascript, ruby, python, php

# PHP Basics

- **Embedded in a web page (.php)**
  - Handled by a preprocessor
  - `<?php ... ?>`
  - `<? ... ?>`
  - Php section replaced by its output
- **Simple interpreted language**
  - Untyped code
  - Basic data types: string, int (long), float (double), associative arrays
  - Classes
    - Different syntax and sense than Java
    - Vary based on which version of PHP
- **Lots and lots of built in functions**
- **Similar to JavaScript (but different)**

# PHP Features

- **All variables start with \$**
- **Strings**
  - Both ‘...’ and “...” strings
  - With “...” you can embed expressions
    - \$var {\$var expr} \${var expr}
  - Can also put in arbitrary html <<xxx ... xxx
- **Embedded: intermix HTML and PHP**
- **Include and require statements**
- **Arrays are associative and indexed**
  - A[‘value’], A[3]
  - foreach (\$x as \$y => \$z) { ... }

# PHP Libraries

- **Full regular expressions (all types)**
- **Database access (postgresql, mysql, ...)**
- **Socket access**
- **String manipulation**
- **Graphics (creating images)**
- **Sessions (web page management)**
- **All available C libraries**

# PHP Example

- <http://dweb-devel/courses/cs032/sample/primes.html>
- `primes.html`
- `result.php`
- `primetest.php`

# Rabin-Miller Test

- **Given an odd integer  $n$** 
  - Let  $n = 2^r s + 1$  with  $s$  odd
- **Choose random  $a$ ,  $1 \leq a \leq n-1$**
- **If  $a^s \equiv (1 \pmod n)$  or  $a^{2^j s} \equiv (-1 \pmod n)$** 
  - For some  $0 \leq j \leq r-1$
  - Then  $n$  passes
- **A prime always passes**
  - A composite has a **50%** chance of passing

# Designing a Web Application

- **What goes where**
  - **Division of responsibility**
    - What is done in the front end (javascript)
    - What is done in the middleware (php)
    - What is done in the back end (server)
      - Is a back end needed or is middleware enough?
  - **What are the tasks**
    - Providing a web interface
    - Checking input for correctness, validity
    - Putting together a query/request from the input
    - Getting information from backend using this
    - Formatting that information for display
  - **What are the alternatives?**

# Flow of Control

- **The UI for a web app is complex**
  - Can be 10s or 100s of template pages
  - Interactions can be complex
    - Error handling
    - Different buttons
    - Transitions
- **Design the flow logic separately**
  - Good idea to have it explicit as part of design
  - It will probably be implicit in the code
- **Difficulties with using browser**
  - Multiple pages can be up simultaneously
  - Back and forward buttons

# Development Strategies

- **Determine what each component does**
  - Most flexible if done from scratch
  - Back end often exists in advance
- **If using a database, determine schema**
- **Develop prototype web pages**
  - User interface first
  - Static html with sample inputs
  - Use CSS to allow easy modification
  - Get the look and feel right
  - Get the interactions right

# Development Strategies

- **Decide on technology to use**
  - AJAX or simple JavaScript for the front end
  - PHP or other for the back end
- **For PHP**
  - Implement simple PHP interaction pages
    - Based on static pages
  - With dummy calls to get back end data
  - Get the look and feel right
  - Get the formatting right
- **For AJAX**
  - Implement the appropriate JavaScript
    - With dummy back end data pages
  - Or use GWT

# Development Strategies

- **Then implement the back end**
  - And produce separate PHP pages to communicate with the back end
  - Isolate common routines
    - E.g. database access
- **Think about security throughout**
  - Session management (cookies, etc.)
  - SQL injection attacks
  - Passwords and keeping user information

# Web Application Guidelines

- **Where do you want the complexity**
  - Where it is most manageable
  - Where it is easiest to compartmentalize
- **What is going to change most frequently**
  - Look and feel of the pages
  - Output formatting
  - These should be isolated as well
- **Compartmentalize as much as possible**
  - Don't spread the database schema
  - Provide abstract interface to back end / data

# Next Time

- **Basic C programming**
  - **Thinking procedurally**
  - **Using C**
  - **Coding in C**