
TU Alum, has a cool job!

Didn't take this class

# Advanced Web Development

Siddharth Kaza
Computer and Information Sciences, Towson University

# Outline

- Introduction and motivation
- Syllabus and course organization
- A brief history of web development
  - HTML
  - CGI
  - Frameworks
- Ruby on Rails
  - Setup
  - Why RoR?
  - A quick video
- Administrative To Dos
  - Tell us more about yourself
  - Project groups (decide this week)
  - Presentations (decide this week)

# About me

- Associate Professor, Department of Computer and Information Sciences

- Ph.D. in Management Information Systems, University of Arizona
- M.S. in Computer Science, Central Michigan University
- B.Sc. In Mathematical Sciences, University of Delhi

Don’t let the management fool you, I can code!

# Research and teaching

- Research interests
  - Cybersecurity Education, Data/Web Mining, Learning Sciences

- Teaching
  - Web Development (COSC 617, COSC 490)
  - Data Communications and Networking (CIS 350)
  - Java programming (COSC 236, 237)
  - Mobile Application Development (COSC 490ish)

# Introduce yourselves

- What languages have you programmed in?
- Have you done any web development before?
- Have you worked before or work now in a dev position?
- What do you expect from this course?

- Prerequisites
  - Database
  - Love coding!
  - HTML, CSS basics – you can acquire these within the first week

# My responsibilities

- Prepare *useful* and *interesting* knowledge
- Offer *challenging* but *reasonable* assignments, projects and exams
- Encourage a *collaborative*, *learning* environment
- Grade *fairly* without bias
- Return graded work *promptly*
- Goals:
  - Have *interesting* lectures
  - Make the class *fun* and *applicable*
  - Use *technology* and *tools* appropriately

Adapted from: http://cs.gmu.edu/~offutt/softwaretest/bs-level-syllabus/slides/swe437-2b-responsibilities.pdf

Credit also to Dr. Josh Dehlinger

# Your responsibilities

- Come to class *on time* and *stay* if you came
- If you miss a class, *learn* the material you missed on your own
- *Listen* and *read* all instructions
- Turn in assignments and projects *on time*
- *Ask* for *help* when you are confused or don't understand
- *Read* the material

- **Work hard / learn enough to *earn* a good grade**
  - **Don't cheat!**
  - **Ask Questions!**

# Teaching Philosophy

*What I hear, I forget*
*What I see, I remember*
*What I do, I understand*

*Confucius*

# Learning philosophy

*You can get help from teachers, but you are going to have to learn a lot by yourself, sitting alone in a room*

-- Dr. Seuss

*with StackOverflow*

-- Dr. Kaza's addendum

you won't learn anything from siddarth (sic) but you will from Google, YouTube, and Stackoverflow.
-- One of my student evaluations

stackoverflow

# Why this course?

- Web development is much more than "web-pages"

- Any serious web development project at a minimum requires all of the following:
  - Handling user interaction
  - Handling errors
  - Retrieving, processing, and managing data
  - Presenting information
  - Handling scalability issues

# How is a web-project different from a non-web system?

- Not very different in some aspects..
  - Both are powerful systems that handle user interactions, errors, manage data, etc.
  - Both need to deal with networks, scalability (though more so in the web)..
  - Both need to manage legacy data
  - Be compatible with changing technology

- Different in others..
  - Assume a light-weight client (almost all processing will be server side)
  - Worry about security issues (you may be accessible to the world)
  - Worry about compatibility issues, users will use different operating systems, different browsers
  - Different devices

# Web development is Software Engineering

- Just like other software development projects, you need a combination good techniques and tools for web development.
  - It's not each webpage on its own ..
  - Web projects are complex and if not designed well coding can be repetitive.
- For a good project you need:
  - Good techniques
  - Powerful tools
- Techniques: Guide you on how things should be done
- Tools: Help you in getting things done

# Techniques and tools

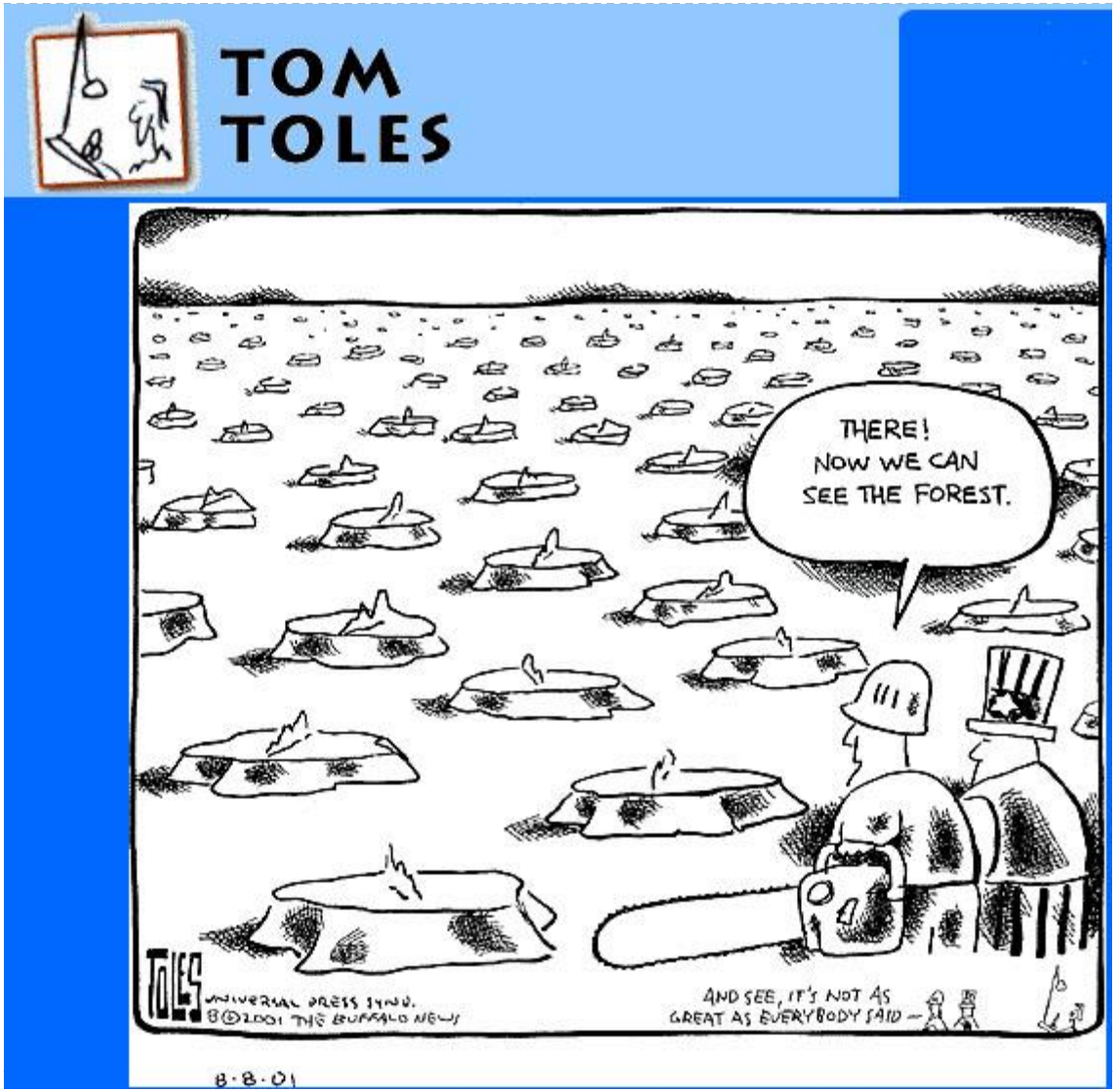
- Techniques
  - Software engineering approaches
    - Water-fall, iterative, spiral..
    - Model-view-controller (MVC)
    - Agile development
    - Integrated testing
    - and others.
  - Why study techniques
    - long shelf life
    - applicable to multiple tools

- Tools
  - Most of them boil down to a technique
    - MVC – Struts, JavaServerFaces, Rails (Ruby), ASP .NET MVC, Django (Python), Zend (PHP)

# Aim

- Not to miss the forest for the trees!
  - Forest: important general principals for a good web application
  - Trees: specific implementation details of the tools

# But, of course, you have to learn and use a tool



# Ruby on Rails (ROR)

- www.rubyonrails.org
  - Ruby: interpreted object orientated language
  - Rails: web development framework built on Ruby
- SQLite 3 – open-source RDBMS
- Alternatives:
  - Frameworks: .NET, J2EE, …
  - Databases: MySQL, Sql Server, Postgress, …

# Ruby on Rails (ROR)

- Relatively simple
  - Makes it easy to develop, deploy, and maintain
  - https://www.youtube.com/watch?v=JaL9ul17kx0
  - http://rubyonrails.org/screencasts
- Powerful
  - Works for production websites: http://rubyonrails.org/
- Open source
- Lots of "buzz" in the developer community
  - However, feel free to mention your experiences with other frameworks in class

# Outline

- Introduction and motivation
- Syllabus and course organization
- A brief history of web development
  - HTML
  - CGI
  - Frameworks
- Ruby on Rails

# Course organization

- The syllabus and schedule are on blackboard, we'll refer to them.
- Any changes will be posted in updated files there.

# Outline

- Introduction and motivation
- Syllabus and course organization
- A brief history of web development
  - HTML
  - CGI
  - Frameworks
- Ruby on Rails

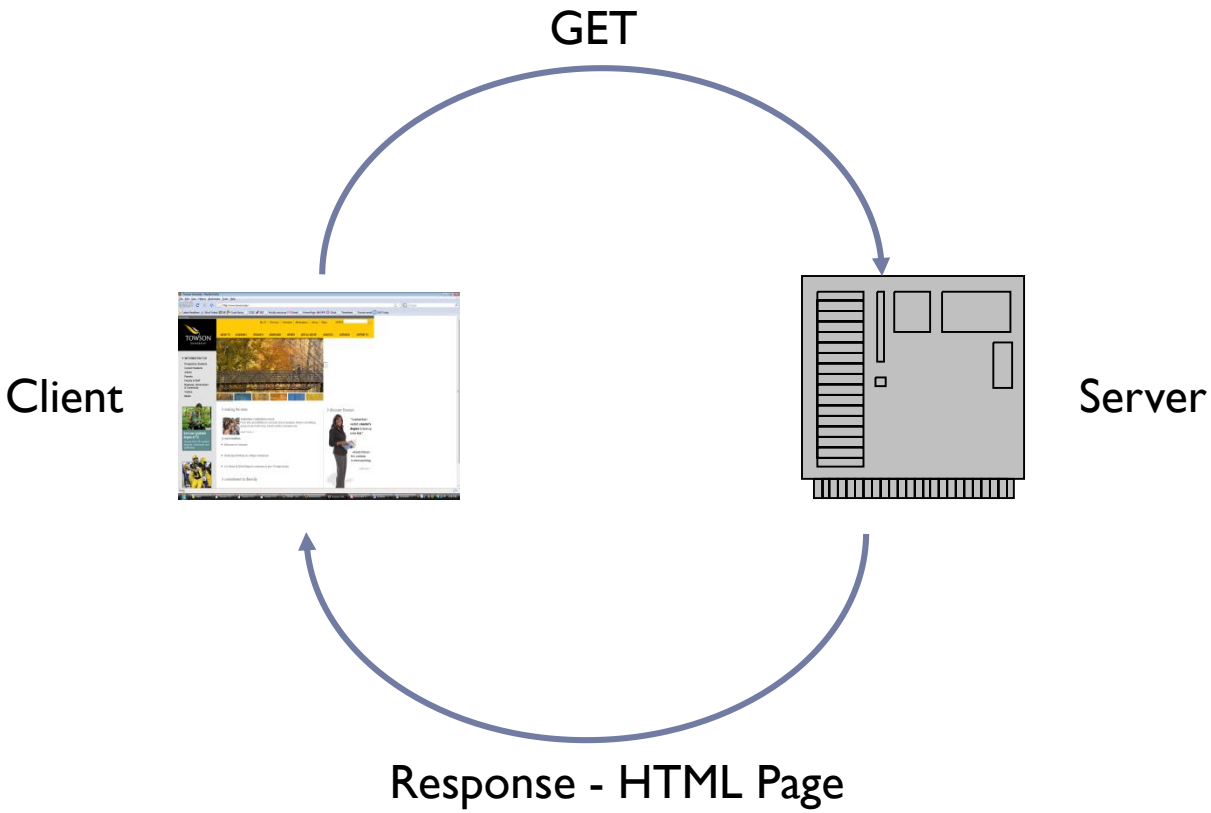
# A brief history of web development

- Hypertext Markup Language (HTML) and Hypertext Transfer Protocol (HTTP)

- HTML – markup and structure
  - What's on a page and where
  - Content and layout
  - Links to other pages
  - Static content stored in files on fileserver

- HTTP
  - Protocol for transfer of information (web pages included)
  - Various methods: HEAD, GET, POST, DELETE ..

# Original Web Application



# A quick review of HTML

- This is a good resource: http://www.w3schools.com/html/html_intro.asp

```
<HTML>
        <HEAD>
        <TITLE>This is a title</TITLE>
        </HEAD>
        <BODY>
        </BODY>
</HTML>
```

- Also XHTML – which is well-formed HTML. All tags are closed.
  - Easy to parse among other benefits

# HTML Elements

- Headings
  - <H1> … </H1>
  - H2, H3, …

- Bullet lists

```
<UL>
        <LI> Item 1</LI>
        <LI> Item 2</LI>
</UL>
```

- Numbered Lists - <OL> instead of <UL>

# More HTML

- Tables:

```
<TABLE>
        <TR> <TD>Row 1, Col1</TD> <TD>Row 1, Col2 </TD> </TR>
        <TR> <TD>Row 2, Col1</TD> <TD>Row 2, Col2 </TD> </TR>
</TABLE>
```

- <P> - paragraph

- Links:

```
<A HREF="http://www.towson.edu">Towson
University</A>
```

# HTML Grouping Elements

- Add structure without any layout
  - <DIV>..</DIV> - blocks, browsers usually put an empty line before and after a <DIV>
  - <SPAN> .. </SPAN> inline, no lines inserted

- Used with id and class attributes to specify layout using CSS

```
<div id = "foo">...</div>
<div class = "foo">..</div>
```

# HTML element identifiers: id and class

- id - unique identifier for an element. Tell one from another

```
<div id="address">...</div>
<div id="phone">..</div>
```

  - Useful for CSS
  - Scripting languages (Javascript) can identify items based on ID
  - Must be unique

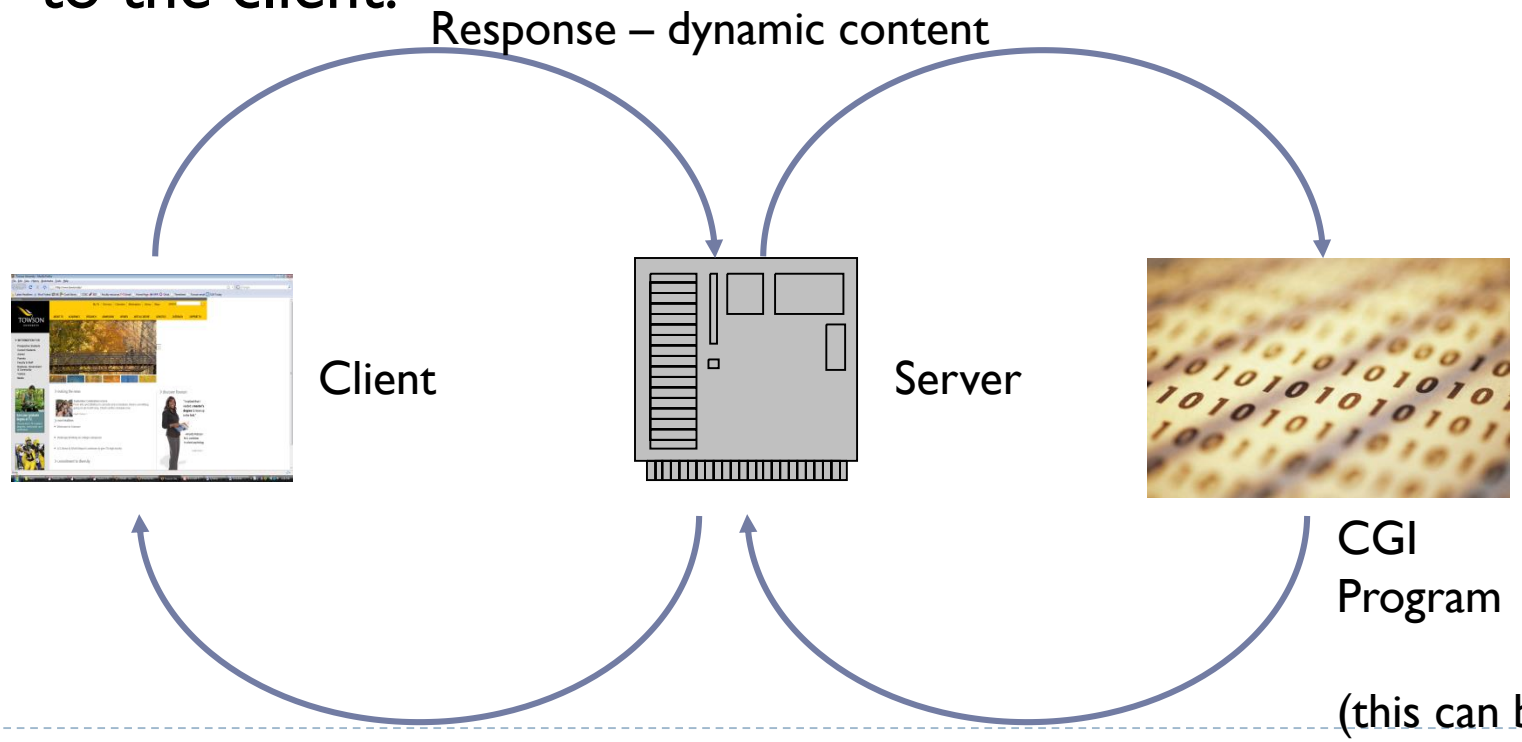
- Class – assigns class names to elements, allowing sharing of attributes and CSS layout.
  - More on this when we discuss CSS in a bit more detail

# What's wrong with this?

- Static content gets boring.
  - Need to respond to user input
  - Generate content that is timely?
- HTML is limiting
  - Both layout and content
- User interaction is not sufficiently powerful
  - Need scripting on user side to make things interesting
- Full page request and response is clunky and slow

# Beyond Static Content: Common Gateway Interface

- Instead of returning a static web page,
- Web server runs a program
- This program prints out HTML , which is then returned to the client.



# CGI Examples

- From:
  - http://inconnu.isu.edu/~ink/perl_cgi/lesson1/hello_world.html

```
#!/usr/bin/perl
print "Content-type: text/HTML\r\n\r\n";
print "<HTML>\n";
print "<HEAD><TITLE>Hello World!</TITLE></HEAD>\n";
print "<BODY>\n";
print "<H2>Hello World!</H2>\n";
print "</BODY>\n";
print "</HTML>\n";
exit (0);
```

  - What are the advantages? drawbacks?

# HTML form elements

- Forms are almost a must for dynamic content
- Good reference (and source of these examples):
  - http://www.fincher.org/tips/web/SimpleForm.shtml

```
<FORM action="http://host/resource" method="GET">
..
</FORM>
```

- Method is "get" if request has no side-effects
  - Database is not modified, etc.
  - Example: read a blog entry
- Otherwise "POST"
  - Example: post a comment in response to a blog entry

# HTML form elements (cont.)

- Short text field: text

```
NAME: <input type="text" name="name"
value="default" size="20" maxlength =
"20">
```

- Longer text block

```
Text Area:
<textarea cols="40" rows="6"
name="comments"></textarea>
```

# HTML form elements (cont.)

- Password

```
Password: <input type=”password” value=”” name =”mypassword”/>
```

- Radio Buttons (group by name)

```
<input type="radio" name="title" value="mr"/>Mr.<br />
<input type="radio" name="title" value="ms"/>Ms.<br />
<input type="radio" name="title" value="decline” checked="checked" />decline<br />
```

# HTML form elements (cont.)

- Selection list, with multiples allowed

```
<select name="computerbrands"
multiple="multiple">
     <option value="DELL"
     selected="selected">DELL</option>
     <option value="IBM">IBM</option>
     <option value="HP">HP</option>
     <option value="Compaq">Compaq</option>
     <option value="Sony">Sony</option>
</select>
```

# Resources on HTML forms

- HTML specification:
  - http://www.w3schools.com/html/html_forms.asp
- CodeAcademy
  - Exercise in the lab

# Evolution of CGI

- Add databases on the back end
  - Store persistent data
  - Shopping carts, etc.
- Faster processing
- Templating languages for embedding code in HTML
  - PHP, Javascript, Embedded Ruby, ASP.NET
- Cookies for managing user state across requests
  - HTTP is "stateless" otherwise. Each request is independent of predecessors.

# Databases

- Assume data for web systems is stored in a RDBMS
  - MySQL, Postgres, SQL Server, Oracle, etc.
  - This assumption may not be true because of cloud storage
- SQL for queries, data definition, etc.
- Frameworks like Ruby on Rails abstract details of data model
  - Bridge between underlying storage/retrieval tools and web system
  - Object-Relational Mapping
  - ActiveRecord (RoR), LinQ (.Net), Entity framework (.Net), Hibernate (Java) …
    - http://en.wikipedia.org/wiki/List_of_object-relational_mapping_software
- Other alternatives possible
  - XML, RDF
  - Storing in the cloud (Amazon Simple DB etc.)

# HTML with DB-driven Content

- PHP (http://www.php.net) - "hypertext preprocessor"
  - Database-driven web sites via templates.
  - Example from http://devzone.zend.com/node/view/id/641

- Mysql

```
Mysql –user=.. --password=...
Create database testdb;
CREATE TABLE `symbols` (
`id` int(11) NOT NULL auto_increment, `country` varchar(255) NOT NULL default '',
`animal` varchar(255) NOT NULL default '', PRIMARY KEY (`id`)
) TYPE=MyISAM;
INSERT INTO `symbols` VALUES (1, 'America', 'eagle');
INSERT INTO `symbols` VALUES (2, 'China', 'dragon');
INSERT INTO `symbols` VALUES (3, 'England', 'lion');
INSERT INTO `symbols` VALUES (4, 'India', 'tiger');
INSERT INTO `symbols` VALUES (5, 'Australia', 'kangaroo');
INSERT INTO `symbols` VALUES (6, 'Norway', 'elk');
```

# Displaying Data

```
<HTML>
<body>
<?php
// set database server access variables:
$host = "localhost"; $user = "test"; $pass = "test"; $db = "testdb";
// open connection
$connection = mysql_connect($host, $user, $pass) or die ("Unable to connect!");
// select database
mysql_select_db($db) or die ("Unable to select database!");
// create query
$query = "SELECT * FROM symbols";
// execute query
$result = mysql_query($query) or die ("Error in query: $query. ".mysql_error());
```

# Displaying Data (cont.)

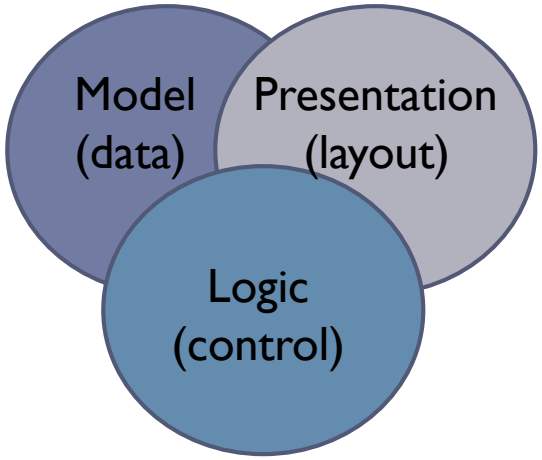
```
// see if any rows were returned
if (mysql_num_rows($result) > 0) {
// yes
// print them one after another
echo "<table cellpadding=10 border=1>";
while($row = mysql_fetch_row($result)) {
echo "<tr>";
echo "<td>".$row[0]."</td>";
echo "<td>" . $row[1]."</td>";
echo "<td>".$row[2]."</td>";
echo "</tr>";
} echo "</table>";
}
else {
// no -print status message
echo "No rows found!";
}
// free result set memory
mysql_free_result($result);
// close connection
mysql_close($connection);
?>
</body>
</HTML>
```

# What's wrong with this?

- What happens if we want to
  - Change underlying database?
  - Change data model?
  - Change appearance/layout of items?
- Imagine dozens or hundreds of pages working off of similar data.
- Tightly coupled data model, presentation, and logic
  - Maintenance, generalization, hard.
  - "Brittle" - small changes in application require lots of work to implement.
- Not sufficiently abstracted

# Evolution from CGI to Frameworks

- Move from templates with embedded processing languages and database to more powerful systems and frameworks
- Don't leave SQL in the embedded code of a web page
- Separate out
  - Data model
  - User interface
  - Application logic
- Model-View-Controller!




# A Good Framework

- Separation of concerns
  - Data, layout, logic
  - Model -view-controller
  - Don't Repeat Yourself (DRY)

- Other aspects
  - Convention over Configuration
  - Dynamic Languages?
  - Object Relational Mapping
  - Testing tools

# A Good Framework (cont.)

- Flexibility of tools
  - Easy migration between data stores?
  - Easy of use of outside resources (javascript, CSS etc.)
  - Support for legacy (pre-existing) data?

- Simplicity
  - Single language? Minimize need for multiple alternative development tools and environments
    - No separate languages for backend, frontend, DB, aspects, …
  - Abstracted handling of common elements and patterns

# A Good Framework (cont.)

- Support for re-use
  - Layout – common templates for pages across a site
  - Site-wide functionality: authentication?
  - Deployment features: URL routing, integration with production quality servers (either as part of framework or stand-alone)

- General desirables
  - Documentation
  - Support
  - Open source
  - Community "buy-in"

# A Good Framework – The Software Engineering Point of View

- Support for evolving, iterative development

- Questions for the semester:
  - How well does Ruby on Rails meet these needs?
  - How do other tools compare?
  - Are there other desirables that we should be looking for?

  - We'll answer some of these with the readings and assignments

# Ruby on Rails – in action

- Installation
  - http://www.rubyonrails.org/down
    - Ruby
    - SQLite3
    - RubyGems
    - Rails – install with "gem install rails"
- Install Radrails, Rubymine, Sublime ….
- See instructions on blackboard
- Video on RoR
  - https://www.youtube.com/watch?v=JaL9ul17kx0