Pre-Spec Work
Part 2 - Know Thy Medium

CSE 112
Winter 2016
Prof. Powell
Tools and Tech First?

- Sometimes we do start with tools or tech first
- Corporate Incumbency/Techincal Legacy
  - “We’re a Java shop”
- Hopefully tool/tech is driven by appropriate need but if we are honest they aren’t far too often
  - “Everybody uses jQuery”, “Rails is hot!”, “We must go to the cloud”, “Let’s do an App”, “Angular sucks, react rocks”, etc.
Loving Constraints

• Constraints are good
  • They make our job easier because they are fixed points
  • Open ended pursuit of perfect tech/tool answers can lead to analysis paralysis

• Reviewing our constraints
  • Web based, JavaScript, MEAN stack, try to be modern SE using proper tools
Web Based

- Browser based as opposed to app based
- Open distribution but discovery challenges
- Avoid install issues, but less persistence
- Speed/Quality concerns
  - Discuss
- Q: Is this choice appropriate if we consider our project type?
My Environmental Assumptions

“To succeed in the Web environment you must assume the Internet is a hostile, unpredictable network, populated by users with widely varying skill levels and intentions.”
There are no guarantees

- You are not guaranteed:
  - High Bandwidth
  - Low latency
  - Predictable error free delivery
  - Safe data (data may be not just surprising but malicious)
  - A client-side with appropriate version/tech
  - A client-side configured or running properly
  - Friendly end-users (do bad things not just by mistake)
  - Technically aware end-users
It’s not just me

Douglas Crockford on browsers:

“The most hostile software development environment imaginable.”

Network computing is hard

Fallacies of networked computing

- The network is reliable.
- Latency is zero.
- Bandwidth is infinite.
Environmental Counterpoint

Am I wrong and just being too cynical? Observation says probably not:

- “Layer 8” Error Correction
- Unstable Browser as platform
- Unstable tech stack and standards
- Actual network monitoring esp. RUM JS based
- Observed hacking
- Our video of users
Development Medium

Client Side
- HTML Markup
- Images
- Java Applets, ActiveX Controls or Plug-Ins
- Browser
- Multimedia
- Scripting (e.g. JavaScript)
- Helper Applications (e.g. Excel)

Server Side
- Web Server
- Server API Programs (e.g. ISAPI, NSAPI)
- Server side scripting pages (e.g. Active Server Pages)
- CGI Program
- Backend Database

Network
Web App in Context of Medium

Web Client

- HTTP request (cleartext or SSL)
- HTTP reply (HTML, Javascript, JSON, Flash, etc)

Web Server

- Apache
- IIS
- Ngnix etc...
- J2EE
- .NET
- JS
- PHP etc.

SQL Database

- Database connection: ADO, ODBC, etc.

Firewall

DB

DB
Client-Side/Server-Side Trade-offs

• Client-side is fast, but it is unpredictable, insecure and unsafe
• Server-side is predictable but it is slow because of network round-trip
• Clear idea is to balance both sides rather than suggest a sole side solution
Thin and Fat Clients

• Thin-client: very safe, but not-rich and slow

• Fat-client: very rich and fast, but likely fragile and insecure

• If we could we would engineer a fat client solution to error gracefully or fall back to thin client version in less than ideal situations (tech, network, hack)
# Web Toolbox

<table>
<thead>
<tr>
<th>Client Side</th>
<th>Server Side</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Helper Applications</strong></td>
<td><strong>CGI scripts and programs</strong></td>
</tr>
<tr>
<td><strong>Browser API Programs</strong></td>
<td><strong>Server API Programs</strong></td>
</tr>
<tr>
<td>* Netscape Plugins</td>
<td>* ISAPI</td>
</tr>
<tr>
<td>* ActiveX Controls</td>
<td>* NSAPI</td>
</tr>
<tr>
<td>* Google’s Native Client</td>
<td>* Apache Modules</td>
</tr>
<tr>
<td><strong>Java Applets</strong></td>
<td><strong>Java Servlets</strong></td>
</tr>
<tr>
<td><strong>Scripting Languages</strong></td>
<td><strong>Scripting / Programming Frameworks</strong></td>
</tr>
<tr>
<td>* JavaScript</td>
<td>* JSP</td>
</tr>
<tr>
<td>* VBScript</td>
<td>* ASP.NET</td>
</tr>
<tr>
<td></td>
<td>* Classic Active Server Pages (ASP)</td>
</tr>
<tr>
<td></td>
<td>* ColdFusion</td>
</tr>
<tr>
<td></td>
<td>* PHP</td>
</tr>
<tr>
<td></td>
<td>* Ruby (Rails)</td>
</tr>
<tr>
<td></td>
<td>* JavaScript</td>
</tr>
<tr>
<td></td>
<td>Note: Nodejs somewhat out of box really</td>
</tr>
</tbody>
</table>
General Tool Trade-off

- Complexity and Speed
- Abstraction and Ease
- Higher/Lower Up ~ Speed
- Difficulty of coding ~ Expense
- Higher/Lower Up ~ Control
Tool Sanity

• Rough functional & trade-off equivalences within toolbox bins, but executional, ecosystem, and other differences

• General performance, feature or quality of two items in a toolbox bin is executional rather than architectural

• Scale is generally not language issue, that is app architecture

• One can be Web scale (or Web stupid) in anything
HTTP is the Key

- Sure markup (HTML), look (CSS), coding (JS), browsers, etc. are unique but the part that really makes this vastly different from programming (as is in desktop) is the network

- Theory: Knowing HTTP is key to an informed understanding of robust Web programming

- HTTP - simple stateless application layer protocol
Basic HTTP Request/Response Cycle

Asks for resource by its URL:
http://www.foo.com/page.html

HTTP Request

HTTP Server

Resource /page.html

HTTP Response

HTTP Client

Browser decodes MIME type and determines action

www.foo.com

maps file extension .html to appropriate MIME type: text/html
That was too simple

Reality actually is more like...
HTTP Up Close

07/01/04 09:07:02 Browsing http://www.ucsd.edu
Fetching http://www.ucsd.edu/

```
GET / HTTP/1.1
Host: www.ucsd.edu
Connection: close
User-Agent: Sam Spade 1.14

HTTP/1.1 200 OK
Date: Thu, 01 Jul 2004 16:07:00 GMT
Server: Apache/1.3.27 (Unix)
Last-Modified: Thu, 01 Jul 2004 16:01:00 GMT
ETag: "c992b-77df-40e4353c"
Accept-Ranges: bytes
Content-Length: 30687
Connection: close
Content-Type: text/html

<html lang="en">
<head>
<title>University of California, San Diego</title>
<meta name="generator" content=""/>
<meta name="author" content="UCSD Libraries, Information Technology Depart"/>
Fetching http://www.pint.com/badurl ...
GET /badurl HTTP/1.1
Host: www.pint.com
Connection: close
User-Agent: Sam Spade 1.14

HTTP/1.1 404 Not Found
Content-Length: 16592
Content-Type: text/html
Server: Microsoft-IIS/6.0
Date: Thu, 01 Jul 2004 16:57:38 GMT
Connection: close

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
Law of 3

Focus on three pieces in, three pieces out

Request Line

Request Headers

Request Body (null here)

Response Line

Response Headers

Response Body

GET / HTTP/1.1
Host: www.ucsd.edu
User-Agent: Thomas Super Browser
X-Demo: Yep

HTTP/1.1 200 OK
Date: Thu, 10 Oct 2013 22:09:10 GMT
Server: Apache/2.0.63 (Unix) DAV/2 mod_perl/2.0.4 Perl/v5.8.4
Last-Modified: Thu, 10 Oct 2013 21:35:31 GMT
ETag: "6ec2-c7f86ec0"
Accept-Ranges: bytes
Content-Length: 28354
Content-Type: text/html; charset=UTF-8

<!DOCTYPE html>
<html lang="en">
<head>

</head>
HTTP Challenges

• Performance
  • Issue: Lots of requests
  • Solution: Request reduction via bundling and caching

Performance Golden Rule: Less data, less often and close by
HTTP Challenges

• Security
  • Or lack there of
    • HTTPS only addresses about 1/3 of the problem

Security Golden Rule: Trust no user nor data
HTTP Challenges

- State Management (since stateless)
- Possible Solutions
  1. Dirty URLs
  2. Hidden Form Fields
  3. Cookies
  4. Local Storage (please no!)
  5. Move to stateful (ex. WebSocket)
- Privacy concerns undermine state issues
- Framework abstraction can really hurt us here if we don’t understand mechanisms
HTTP Challenges

• Protocol Misunderstanding is Widespread
  • Cave Man HTTP: Ugh, Bugha Bugha
  • GET = Ugh, POST = Bugha Bugha interchange at will

• We aim to build modern service oriented architecture
  we need a deeper understanding of HTTP than that!

• Careful though are you sure you have HTTP verb
  support end to end? I think that is assumed wrongly
  more than we might want to admit. Good engineers
  would verify a system before using it (in code not just
  in theory!)
Browsers!
Browser Families

1. Mozilla/Firefox (Gecko) - Netscape descendent
2. Google Chrome (initially WebKit now Blink)
3. Safari (Webkit)
4. Internet Explorer (Trident and new IE)
5. Opera (Presto, Blink)


• If browser is the OS should we know how browsers work?

Not Even Close

• Goal: Trying to cram 20 weeks of material on HTTP, server, HTML, JS, CSS, etc. into a few hours

• Key points still to come in parts 3 and 4 which get more and more practical

• Sad Truth: Projects are undone or overbudget because of huge misconceptions about tech, server, etc. It is always best to know what we speak of!
Our Project Medium

- Web App
  - Underline word Web
  - What does that mean?
    - HTTP, HTML, Browsers, Servers, etc.
HTML Points

• Traditional SGML Based
  • HTML 2, 3, 3.2, 4 (strict and transitional)

• XML Based
  • XHTML 1, 1.1

• Back to the future
  • HTML5 (no SGML DTD)
  • Odd Duck: XHTML5
Doctypes

- Traditionally (X)HTML doctypes indicate dialect and DTD of the markup language in use

```xml
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
 "http://www.w3.org/TR/html4/loose.dtd">

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```
What is a DTD?

- The actual reference (implicit) or linked DTD contains the rules of valid markup

- From [http://www.w3.org/TR/REC-html40/sgml/dtd.html](http://www.w3.org/TR/REC-html40/sgml/dtd.html)

- `<img>`
Uses of Doctypes

• Modern browsers are aware of the <!DOCTYPE> and will examine it to determine what rendering mode to enter (standards vs. quirk).

• This process is often dubbed the “doctype switch”

• Using the <!DOCTYPE> declaration allows validation software to identify the DTD being followed in a document, and verify that the document is syntactically correct—in other words, that all tags used are part of a particular specification and are being used correctly.
Doctype Switch in Action
Checking the conformance of a document to the stated (or inferred DTD)...do the tags meet the rules in the spec? Is it valid?
• well-formedness - following basic syntax rules for quotes, closing tags, case, etc.

• validity - adhering to the allowed vocabulary of tags and how they may be used

• “Purple is an eager flying cactus who loves a dog sadly perhaps.” is well formed English sentence but it isn’t valid*
Why Bother? Part 1

Welcome to the World of HTML

HTML really isn’t so hard!
Soon you will ♥ using HTML.
You can put lots of text here if you want. We could go on and on with fake text for you to read, but let’s get back to the book.

Wellformed Markup

Malformed Markup
Why Bother Part 2

Markup Validation Service
Check the markup (HTML, XHTML, ...) of Web documents

Jump To:  Notes and Potential Issues  Validation Output

Errors found while checking this document as HTML5!

Result: 40 Errors, 2 warning(s)
Address: http://www.google.com/
Encoding: iso-8859-1  (detect automatically)
Doctype: HTML5  (detect automatically)
Root Element: html

The W3C validators rely on community support for hosting and development.
Tag Soup

Bad Example: http://htmlref.com/ch1/malformedhelloworld.html

<TITLE>Hello HTML World</title>
<!-- Simple hello malformed world -- example -->
</head>
<body>
<h1>Welcome to the World of HTML</h1>
<hr />
<p>HTML <em>really</em> isn't so hard!</p>
<p>Soon you will &hearts; using HTML.</p>
<p>You can put lots of text here if you want. We could go on and on with fake text for you to read, <foo>but</foo> let's get back to the book.</p>
</html>
Welcome to the World of HTML

HTML really isn't so hard!
Soon you will ❤ using HTML.
You can put lots of text here if you want. We could go on and on with fake text for you to read, but let's get back to the book.
Guessing the parse

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Malformed HTML</title>
</head>
<body>
<p>Making malformed HTML <em><strong>really</strong></em><em><strong> isn't so hard!</strong></em></p>
</body>
</html>
Yeah everybody does it

<table>
<thead>
<tr>
<th>Study</th>
<th>Date</th>
<th>Total Validated</th>
<th>Passed Validation</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parnas</td>
<td>Dec 2001</td>
<td>2,034,788</td>
<td>14,563</td>
<td>0.71%</td>
</tr>
<tr>
<td>Saarsoo</td>
<td>Jun 2006</td>
<td>1,002,350</td>
<td>25,890</td>
<td>2.58%</td>
</tr>
<tr>
<td>MAMA</td>
<td>Jan 2008</td>
<td>3,509,180</td>
<td>145,009</td>
<td>4.13%</td>
</tr>
</tbody>
</table>

Markup validation studies
Enter XML / XHTML


<p>XHTML <em>really</em> isn't so hard either!</p>

Errors Caught

This page contains the following errors:

error on line 10 at column 146: Opening and ending tag mismatch; on line 5 and 6.

Below is a rendering of the page up to the first error.

Welcome to the World of XHTML

XHTML ready

ERROR!

XML parsing failed: syntax error (line 12, Character 16)
Reparse document on XML
File: mismatched end tag
Specification: http://www.w3.org/1998/xhtml

9: </div>
10: <div>Welcome to the World of XHTML<br/>
11: </div>
12: &quot;XHTML &quot;really&quot; isn't so hard either!&quot;
13: </div>
14: &quot;XHTML &quot;really&quot; isn't so hard either!&quot;
15: XML parsing failed: syntax error (line 12, Character 16)
Unfortunately... even if you wanted to

Correct Render

Parse Tree

Good news everyone! IE9+ fixed this.
Back to the Future

• But we aren’t going there

• Too hard to get people to do it right, easier to get the browser vendors to have a malformedness handler agreement and return to the looseness

• HTML5 from a markup point of view is back to HTML4 with more new stuff...but worse
HTML5 Viz Overview

by Sergey Mavrody  ©  BY-SA
Spec Kitchen Sink
Pfft..HTML 5? HTML 5.2!

The combined timelines for HTML 5.0, HTML 5.1 and HTML 5.2:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>HTML 5.0</td>
<td>Candidate Rec</td>
<td>Call for Review</td>
<td>Recommendation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HTML 5.1</td>
<td>1st Working Draft</td>
<td>Last Call</td>
<td>Candidate Rec</td>
<td>Recommendation</td>
<td></td>
</tr>
<tr>
<td>HTML 5.2(^{[23]})</td>
<td></td>
<td></td>
<td></td>
<td>1st Working Draft</td>
<td></td>
</tr>
</tbody>
</table>
Caniuse.com

CSS
- @font-face Web fonts
- calc() as CSS unit value
- 2.1 selectors
- Counters
- Feature Queries
- Filter Effects
- Generated content
- Gradients
- Grid Layout
- Hyphenation
- inline-block
- Masks
- min/max-width/height
- outline
- position:fixed
- Regions

HTML5
- Audio element
- Canvas (basic support)
- Color input type
- contenteditable attribute (basic support)
- Datalist element
- dataset & data-* attributes
- Date/time input types
- Details & Summary elements
- Download attribute
- Drag and Drop
- Form validation
- HTML5 form features
- input placeholder attribute
- New semantic elements
- Number input type
- Offline web applications
Proceed with caution?

### Canvas (basic support) - Candidate Recommendation

Method of generating fast, dynamic graphics using JavaScript

<table>
<thead>
<tr>
<th>Show all versions</th>
<th>IE</th>
<th>Firefox</th>
<th>Chrome</th>
<th>Safari</th>
<th>Opera</th>
<th>IE: Safari</th>
<th>Global</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current</td>
<td>10.0</td>
<td>20.0</td>
<td>20.0</td>
<td>6.0</td>
<td>12.1</td>
<td>6.0</td>
<td>5.0-7.0</td>
</tr>
<tr>
<td>Near future</td>
<td>21.0</td>
<td>27.0</td>
<td>22.0</td>
<td>26.0</td>
<td>27.0</td>
<td>21.0</td>
<td>27.0</td>
</tr>
<tr>
<td>Farther future</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Sub-features:**
- Text API for Canvas
- WebGL - 3D Canvas graphics

### Date/time input types - Working Draft

Forms field widget to easily allow users to enter dates and/or times, generally by using a calendar widget.

<table>
<thead>
<tr>
<th>Show all versions</th>
<th>IE</th>
<th>Firefox</th>
<th>Chrome</th>
<th>Safari</th>
<th>Opera</th>
<th>IE: Safari</th>
<th>Global</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current</td>
<td>10.0</td>
<td>20.0</td>
<td>20.0</td>
<td>6.0</td>
<td>12.1</td>
<td>6.0</td>
<td>5.0-7.0</td>
</tr>
<tr>
<td>Near future</td>
<td>21.0</td>
<td>27.0</td>
<td>22.0</td>
<td>26.0</td>
<td>27.0</td>
<td>21.0</td>
<td>27.0</td>
</tr>
<tr>
<td>Farther future</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Parent feature:**
- HTML5 form features

**Support:** 8.37%

**Participation:** 2.13%

**Total:** 85.87%
Suggestions

• For public facing applications use HTML5 with an emphasis on realistic support

• Validate your markup, but understand when it may be appropriate to ignore validation

• Automate continuous validation on build and/or deploy

• Use XML based markup in a contained environments where failures are not tolerable
  • Aim beyond well-formedness to validity
Suggestions

• Adopt an HTML style guide
  • Casing, quotes “vs ‘, indent pattern, comments, attribute order, id and class name patterns, etc.
  • Employ a pretty printer for consistency
  • Use templates and scaffolds
    • Skeleton repo
Suggestions - Boilerplates

Small Concerns

• Make sure you know what it is doing

• Are you including random stuff in a boilerplate bulking things up needlessly?

http://html5boilerplate.com
Mistakes on purpose ... theoretical adherence vs reality
Suggestion - Semantics First

• Don’t catch `<div>-itis`
  • `<div id=‘container’><div class=‘inner’>Blah...</div></div>`
• Instead more meaning based tags
  • `<header>, <footer>, <section>`
• Avoid presentational markup (and thinking)
  • `<font>, <b>, bgcolor, etc.`
  • Assuming `<h1>` makes it big!
• Loosely couple style
  • `<time style=‘background: orange’>Oct 31</time> <!-- so so -->
  `<time class=“halloween”>Oct 31</time> <!-- better -->`
MIME

• Multipurpose Internet Mail Extensions
• Used by Web browsers via the HTTP Content-Type header
  • Upload - Post encoding commonly application/x-www-form-urlencoded
  • Download - text/html, image/jpg, etc.
  • Determine what to do in browser*

* most of the time
Get the MIME Right

Welcome to the World of XHTML5

XHTML5 really isn’t so hard either!

HTML5 likes XML syntax too.

Make sure to serve it with the correct MIME type!
It’s not a detail

• Sadly people don’t know MIME and trouble can happen when we “fix” things for them - Thomas’ Law of Unintended Consequences
CSS

• Versions
  • CSS₁, CSS-P, CSS₂, CSS₂.₁, CSS₃*
  • CSS₃ is crazy it is a hodge podge of a million different things of varying states of likelihood of being real or not
Proper CSS Usage

• Separation of concerns
  • HTML for structure, CSS for style
  • External CSS files (watch downloads)
  • Organization style - alpha, general->specific, etc.
  • Watch out for classitis and repeated IDs

• The dream and reality of the Zen Garden
  • http://www.csszengarden.com/
Vendor Prefix Fun

- Prefix new features to keep implementations separate until standard
  - -moz, -webkit, -ms, -o
  - Ex: -moz-column-count, -webkit-column-count, column-count
  - Huge redundancy and headache
important

• Too many rules, bad tree, inheritance complexity -> things just don’t look right

• !important - Useful but ugly hack that should be done inline or at least last rule for best result
  
  • Ex: <p style="!important ruleiwant:valiwant"/>

  • Except when it is a !important war
Suggestions

• CSS Validation - http://jigsaw.w3.org/css-validator/
• CSS minification
• CSS bundling
  • Inlining of CSS? Sometimes makes sense
Suggestion
Use a Framework

http://getbootstrap.com
http://foundation.zurb.com

- Concerns
  “Validity”, Cruft
Suggestion - CSS Preprocessors

- Concerns
  - Solving problems that may go away?
  - Solving problems that suggest a deeper problem you have?
- [http://lesscss.org](http://lesscss.org)

---

Write some LESS:
```
@base: #f938ab;

.box-shadow(e:style, @e) when (iscolor(@e)) {
  box-shadow: @e;
  -webkit-box-shadow: @e;
  -moz-box-shadow: @e;
}

.box-shadow(e:style, @alpha: 50%) when (isnumber(@alpha)) {
  box-shadow(e:style, rgba(0, 0, 0, @alpha));
}

.box {
  color: saturate(@base, 5%);
  border-color: lighten(@base, 30%);
  div { .box-shadow(0 0 5px, 30% )
```

Compile to CSS:
```
rpm install -g less
lessc styles.less styles.css
```
Core JavaScript Discussion
Snipped

Presented in a previous lecture
Suggestion: JS Combining

• Since browser based JS shares same name space combining files doesn’t change anything code wise but improves network delivery because of request reduction

• Production JS files should be combined before deployment

• Combine before minifying in my opinion so you don’t get duplicate names

• Grunt task
Suggestion: JS Minification

• Reduce source code size for delivery
  • Minor amount of obfuscation

• Common Techniques
  • Remove white space, comments, dead code
  • Rename long names
  • Inline single usage
  • Byte shave (ex. x=x+1 to x++)
Suggestion: Minification Tools

- Google Closure Compiler
  https://developers.google.com/closure/compiler/
- UglifyJS http://lisperator.net/uglifyjs/
- Make it part of deployment process
  - GruntJS automation https://github.com/gruntjs/grunt-contrib-uglify
Minification Considerations

• Naming
  • file.js  file.min.js

• Sometimes minifiers break stuff
  • Their bugs, your bad code
  • Sourcemap to allow for in production debug
    • http://www.html5rocks.com/en/tutorials/developertools/sourcemaps/
Suggestion: Build File Naming

- If you are making somewhat frequent updates we need to be quite careful about browser & proxy caches

```html
<script src="myapp.min.js"></script>
<!-- is this dangerous? -->

<script src="myapp.1382403226836.min.js"></script>
<!-- use timestamp but build id is that better? -->
```
Suggestion: Package Management

• If you depend on other people’s code you have to keep up with it

• Rule of thumb: The more you mix the more volatile things are going to get

• Reactions - include nothing and write everything, upon error exclaim it’s not my fault, fork, contribute back
PASTING CODE FROM THE INTERNET INTO PRODUCTION CODE

IS LIKE CHEWING GUM FOUND IN THE STREET
Suggestion: Bower

- http://bower.io/
- bower install <x>
- bower install consults manifest in bowser.json and get it all
- Avoid: bower search - do some eval first!
Suggestion: Scaffolders

- [http://yeoman.io/](http://yeoman.io/)
  - Automates lots of routine tasks
  - Can help speed development - live reloads, staging server, test running, etc.
  - Lots of configuration to learn