Lecture 2:

COS 333 Project Information

and a start on scripting
Project

“a simulation of reality”

a “three-tier” system for any interesting/useful application

groups of 3-5 people
Where do project ideas come from?

The way to get startup ideas is not to try to think of startup ideas. It's to look for problems, preferably problems you have yourself.

The very best startup ideas tend to have three things in common: they're something the founders themselves want, that they themselves can build, and that few others realize are worth doing.

Paul Graham, co-founder of Y Combinator

www.paulgraham.com
Getting started

• **think about potential projects; form a group**
  
  talk to us; look at previous projects;
  look around you; check out the external project ideas page

• **by Wed Mar 14: each group meets with bwk** *(earlier is better)*
  
  – to be sure your project idea is generally ok
  – you should have one pretty firm idea, not several vague ones

• **Sun Mar 18: design document** *(before break)*
  
  – ~3-5 pages of text, pictures, etc.
  – overview of major pieces, how they fit together
  – initial web page, elevator speech
  – milestones: clearly defined pieces either done or not
  – risks

• **must be based on significant thought and discussion**

• **don't throw it together at the last minute**
  
  – all components of the project are graded
Process: organizing what to do

classic "waterfall" model: a very formal process
  specification
  requirements
    architectural design
    detailed design
  coding
    integration
    testing
    delivery

this is overkill for 333,
but some process is essential
Informal process

• what's the big picture?
  – in a sentence or two, what is it, what does it do, for who?
• more detail: what will users see?
  – scenarios / use cases, sketches, screenshots
• what are the components / pieces / subsystems?
  – structure and appearance with diagrams, prototypes
  – specify interactions and interfaces between components
• how will you build it?
  – languages, tools, environment, database, ...
  – make versus buy -- what will you use from elsewhere?
  – resolve issues of connectivity, access to data, software, etc.
  – make prototypes; establish end to end connectivity
  – get real users as early as possible
  – deliver in stages, so that each does something and still works
  – test as you go: if your system is easy to break, it gets a lower grade
Interfaces

- the boundary between two parts of a program
- a contract between the two parts
- what are the inputs and outputs?
- what is the transformation?
- who manages resources, especially memory and shared state?

- hide design & implementation decisions behind interfaces, so they can be changed later without affecting the rest of the program
  - database system, data representations and file formats
  - specific algorithms
  - visual appearance

- "I wish we had done interfaces better" is one of the most common comments
  - less often: "We thought hard about the interfaces so it was easy to make changes without breaking anything."
DILBERT
by SCOTT ADAMS

I'll need to know your requirements before I start to design the software.

First of all, what are you trying to accomplish?

I'm trying to make you design my software.

I mean what are you trying to accomplish with the software?

I won't know what I can accomplish until you tell me what the software can do.

Try to get this concept through your thick skull: the software can do whatever I design it to do!

Can you design it to tell you my requirements?

© Scott Adams, Inc./Dist. by UFS, Inc.
The academic software life cycle
Choices!

Business logic:
Java, C#, Python, PHP, Ruby, Node, C++, Objective-C, Swift, Perl, Go, ...

Frameworks:
jQuery, React, Angular, Vue, ...

Web frameworks:
Django, Flask, Zend, Rails, Cocoa, Express, ...

GUI tools:
Swing, jQueryUI, Bootstrap, ...

Devel Environ:
sh + vi / emacs, Eclipse, Xcode, Visual Studio, Android Dev Kit ...

Server:
OIT MyCpanel, AWS, Heroku, Google Cloud, ...

Wire format:
XML, JSON, REST, ...

Web client:
HTML, CSS Javascript, ...

Plumbing:
TCP/IP, OAuth, CAS, ...

Database:
MySQL, SQLite, Postgres, MongoDB, ...

Repository:
Git, Github SVN, ...

App

Wire format: XML, JSON, REST, ...

Business logic: Java, C#, Python, PHP, Ruby, Node, C++, Objective-C, Swift, Perl, Go, ...

Frameworks: jQuery, React, Angular, Vue, ...

Web frameworks: Django, Flask, Zend, Rails, Cocoa, Express, ...

GUI tools: Swing, jQueryUI, Bootstrap, ...

Devel Environ: sh + vi / emacs, Eclipse, Xcode, Visual Studio, Android Dev Kit ...

Server: OIT MyCpanel, AWS, Heroku, Google Cloud, ...

Wire format: XML, JSON, REST, ...

Web client: HTML, CSS Javascript, ...

Plumbing: TCP/IP, OAuth, CAS, ...

Database: MySQL, SQLite, Postgres, MongoDB, ...

Repository: Git, Github SVN, ...

App
“Make versus buy”

• you can use components and code from elsewhere
  – copy or adapt open source

• overall project design has to be your own
• so does the selection and assembly of components
• so does the bulk of the work

• it's fine to build on what others have done
  – identify what you have used, where it came from

• it’s fine to cooperate with other project groups
  – help each other with insight, knowledge, …
Things to do from the beginning

• think about schedule
  – keep a log of what you did and what you will do next (always current)
• plan for a sequence of stages
  – do not build something that requires a "big bang" where nothing works until everything works
  – always be able to declare success and walk away
• simplify
  – don't take on too big a job
  – don't try to do it all at the beginning, but don't try to do it all at the end
• use source code control for everything
  – Git or equivalent is mandatory
• leave lots of room for "overhead" activities
  – testing: build quality in from the beginning
  – documentation: you have to provide written material
  – deliverables: you have to package your system for delivery
  – changing your mind: decisions will be reversed and work will be redone
  – disaster: lost files, broken hardware, overloaded systems, ...
  – sickness: you will lose time for unavoidable reasons
  – health: there is more to life than this project!
## 2018 Project Schedule

<table>
<thead>
<tr>
<th>Sun</th>
<th>Mon</th>
<th>Tue</th>
<th>Wed</th>
<th>Thu</th>
<th>Fri</th>
<th>Sat</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Feb</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
</tr>
<tr>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
</tr>
<tr>
<td>29</td>
<td>30</td>
<td>31</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Mar</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
</tr>
<tr>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
</tr>
<tr>
<td>29</td>
<td>30</td>
<td>31</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Apr</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
</tr>
<tr>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
</tr>
<tr>
<td>29</td>
<td>30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>May</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
</tr>
<tr>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
</tr>
<tr>
<td>29</td>
<td>30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Feb 8**: First class
- **Feb 16**: Assignment 1 due
- **Feb 17**: Assignment 2 due
- **Mar 18**: Assignment 3 due
- **Mar 20**: Assignment 4 due; team meetings with bwk
- **Mar 22**: Design document due by midnight Saturday
- **Mar 23**: Spring break
- **Apr 18**: Assignment 5 due; weekly TA meetings start
- **Apr 27**: Project prototype
- **May 13**: Alpha test
- **May 19**: Last class; beta test
- **May 17**: Demo days
- **May 19**: Projects due by midnight Sunday
Some mechanics

• **groups of 3 to 5**
  - *find your own partners*
    - use Piazza for match-making
    - meet potential partners before or after class
    - we will try to help, but it is your responsibility
  - **don't leave this to the end !!**

• **TA's will be your first-level "managers"**
  - more mentoring and monitoring than managing
    - it's your project, not the TA's

• **meet with your TA every week after spring break**
  - everyone in the group must attend all of these meetings

• **be prepared**
  - what you accomplished
  - what you didn't get done
  - what you do plan to do next

• **these meetings are a graded component**
  - this is our attempt to make sure that you don't leave it all to the end
Scripting languages

- originally tools for quick hacks, rapid prototyping, gluing together other programs, ...
- evolved into mainstream programming tools
- characteristics
  - text strings as basic (or only) data type
  - regular expressions (maybe built in)
  - associative arrays as a basic aggregate type
  - minimal use of types, declarations, etc.
  - usually interpreted instead of compiled

- examples
  - shell
  - Awk
  - Perl, PHP, Python, Ruby, Tcl, Lua, ...
  - Javascript
  - Visual Basic, (VB|W|C)Script, PowerShell
  - ...
Shells and shell programming

- shell: a program that helps run other programs
- an ordinary program, not part of the system
- popular Unix shells
  - `sh` Bourne shell (Steve Bourne, Bell Labs)
  - `csh` C shell (Bill Joy, Berkeley)
  - `ksh` Korn shell (Dave Korn, Bell Labs)
  - `bash` GNU shell; mostly ksh + much of csh
  - `tcsh`
  - `zsh` (written in 1990 by Paul Falstad ‘92)
Features common to Unix shells

- **command execution**
  + built-in commands, e.g., cd
- **filename expansion**
  + * ? [...]
- **quoting**
  + Careful !!!
  - `echo "It's now `date`"
- **variables, environment**
  + `PATH=/bin:/usr/bin` in ksh & bash
  + `setenv PATH /bin:/usr/bin` in (t)csh
- **input/output redirection, pipes**
  + `prog < in > out, prog >> out`
  + `who | wc`
  + `slow.1 | slow.2 &` asynchronous operation
- **executing commands from a file**
  + arguments can be passed to a shell file ($0, $1, etc.)
  + if made executable, indistinguishable from compiled programs
  + provided by the shell, not each program
Shell programming

- shell programs are good for personal tools
  - tailoring environment
  - abbreviating common operations
    (aliases do the same)
- gluing together existing programs into new ones
- prototyping
- sometimes for production use
  - e.g., configuration scripts

- But:
  - shell is poor at arithmetic, editing
  - macro processing is a mess
  - quoting is a mess
  - sometimes too slow
  - can't get at some things that are really necessary

- this leads to scripting languages