Lecture 2:

Project Information

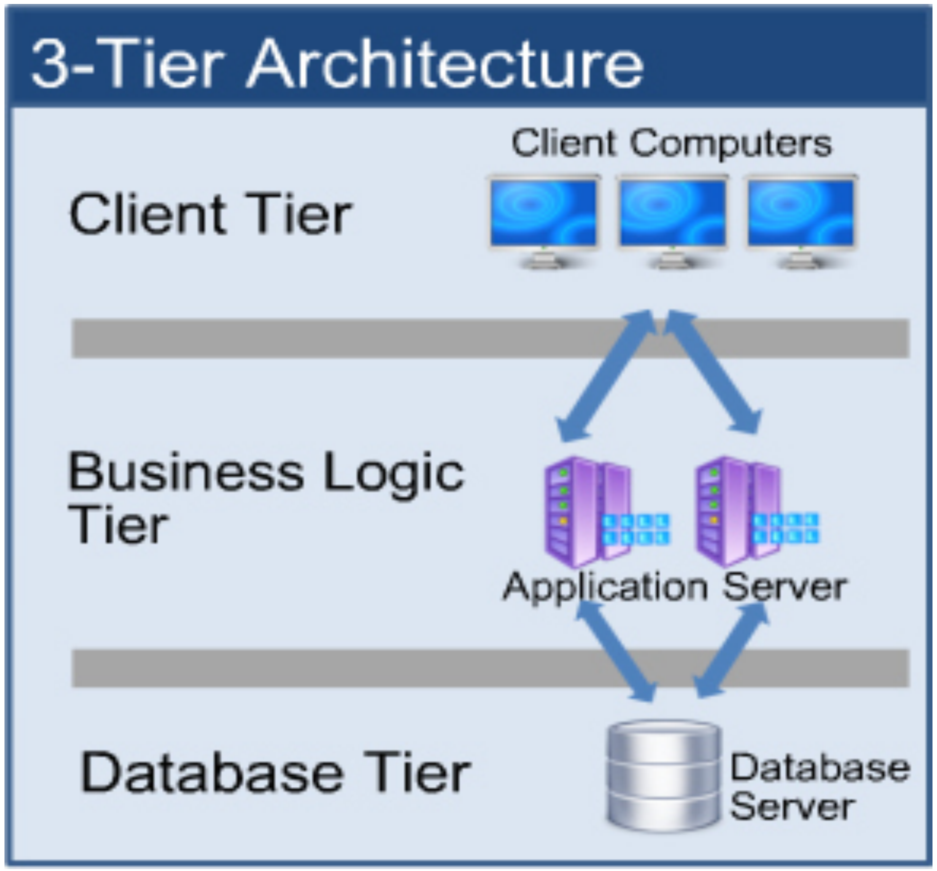
and a start on scripting languages

Project

“a simulation of reality”

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

groups of 3-5 people



Projects from across campus

- Tiger Energy (Caroline Savage, Office of Sustainability)
- Anti-gerrymandering (Sam Wang, Neuroscience)
- Dynamic Frist displays (Abby Klionsky '14, EVP)
- Co-curricular portfolios (Claire Pinciaro '13, ODUS)
- Geospatial maps and data (Wangyal Shawa, Library)
- Prison Teaching Initiative (Jill Stockwell, McGraw Center)
- ...

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

- Now: think about potential projects; form a team

  talk to us; look at previous projects; use Piazza

  look around you; check out the external project ideas page
- by Fri Mar 1: each group meets with bwk (<u>earlier is better</u>)
  - to be sure your project idea is generally ok
  - you must have one pretty firm idea (maybe 2), not several vague ones
- by Sun Mar 10: design document
  - ~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
- design must be based on significant thought and discussion
  - we will organize small-group design reviews
- don't throw it together at the last minute
  - all components of the project are graded
- by Fri Mar 15: first meeting with your TA advisor/mentor

Process: organizing what to do

- you must use an orderly process or it won't work

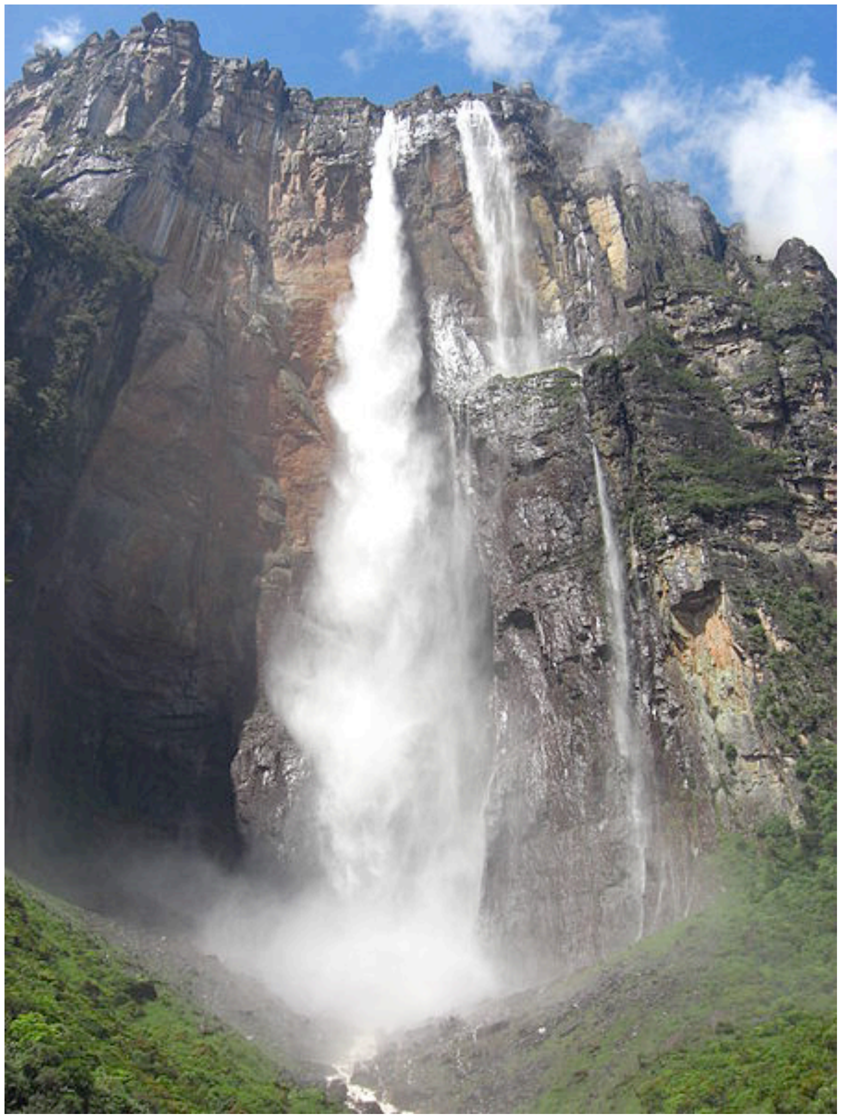
- this is NOT a successful process:

```
repeat {
    talk about the system at a party
    hack some code together
    test it a bit
    do some debugging
    fix the obvious bugs
} until the semester ends
```

Process: organizing what to do

classic "waterfall" model: a very formal process

- specification
  - requirements
    - architectural design
      - detailed design
        - coding
          - integration
            - testing
              - delivery

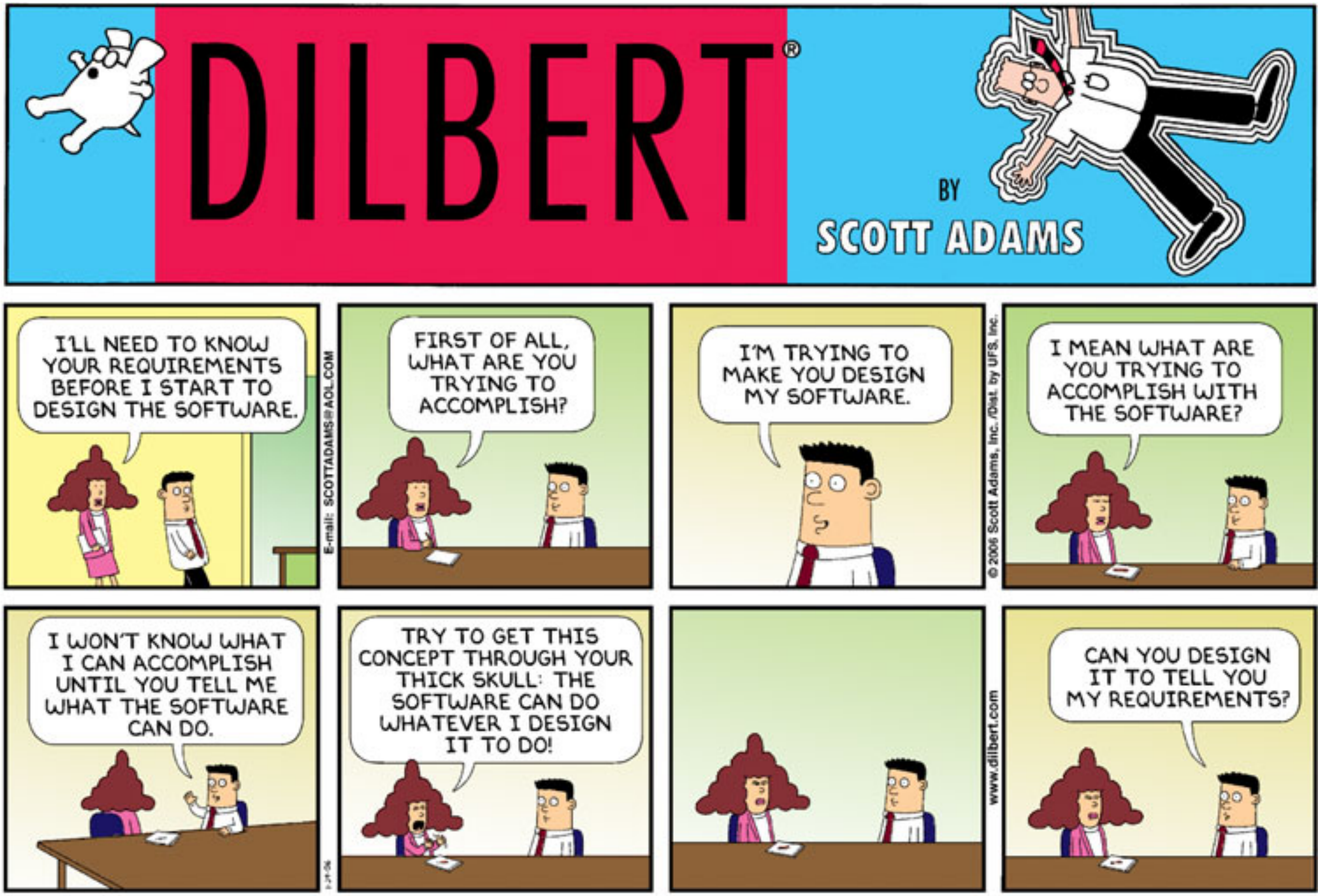


this is overkill for 333,

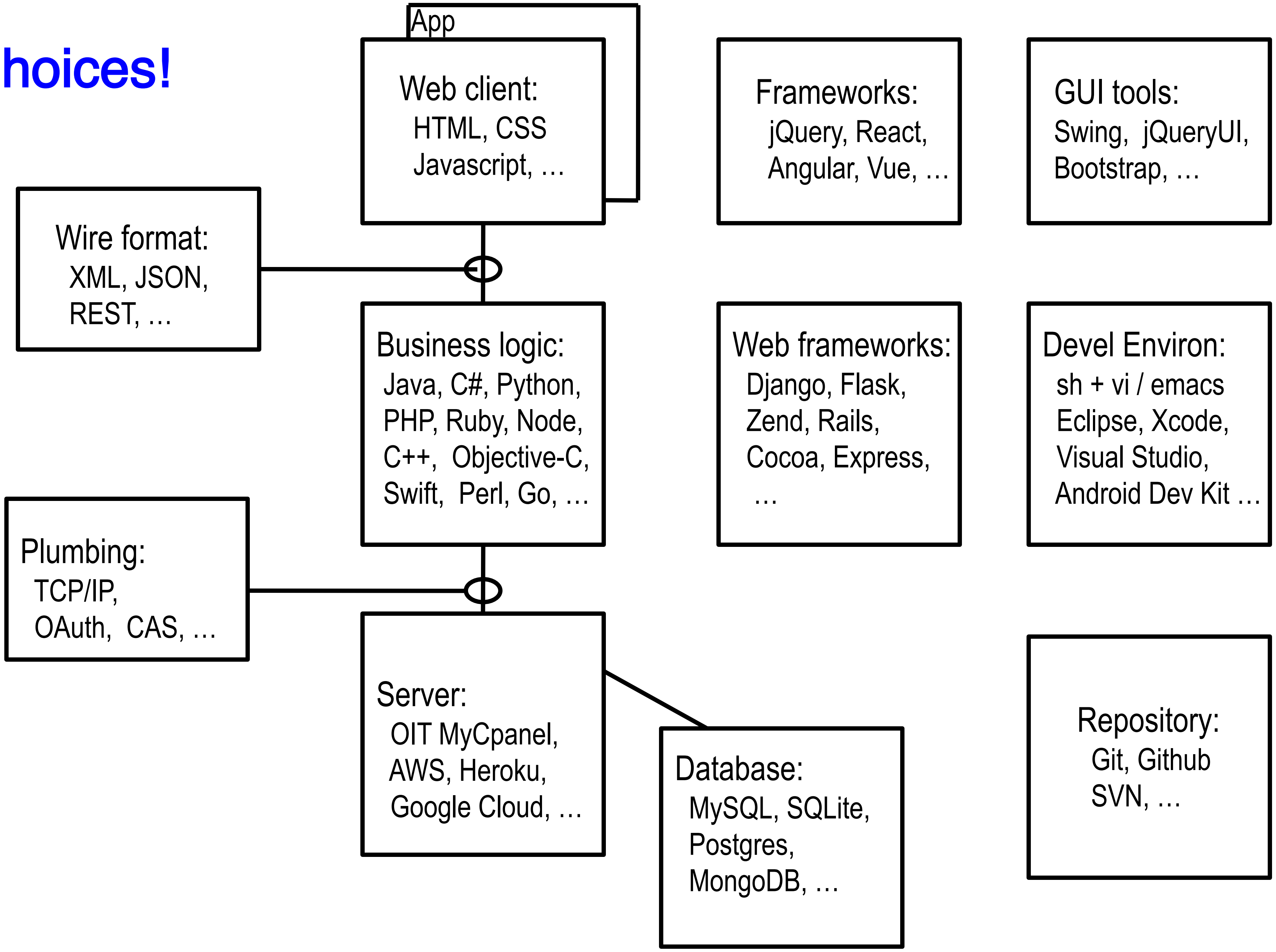
but some process is essential

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 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


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?
E-mail: SCOTTADAMS@AOL.COM
© 2006 Scott Adams, Inc. /Dist. by UFS, Inc.
www.dilbert.com

Choices!


App
Web client: HTML, CSS Javascript, …
Frameworks: jQuery, React, Angular, Vue, …
GUI tools: Swing, jQueryUI, Bootstrap, …
Wire format: XML, JSON, REST, …
Business logic: Java, C#, Python, PHP, Ruby, Node, C++, Objective-C, Swift, Perl, Go, …
Web frameworks: Django, Flask, Zend, Rails, Cocoa, Express, …
Devel Environ: sh + vi / emacs Eclipse, Xcode, Visual Studio, Android Dev Kit …
Plumbing: TCP/IP, OAuth, CAS, …
Server: OIT MyCpanel, AWS, Heroku, Google Cloud, …
Database: MySQL, SQLite, Postgres, MongoDB, …
Repository: Git, Github SVN, …

“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

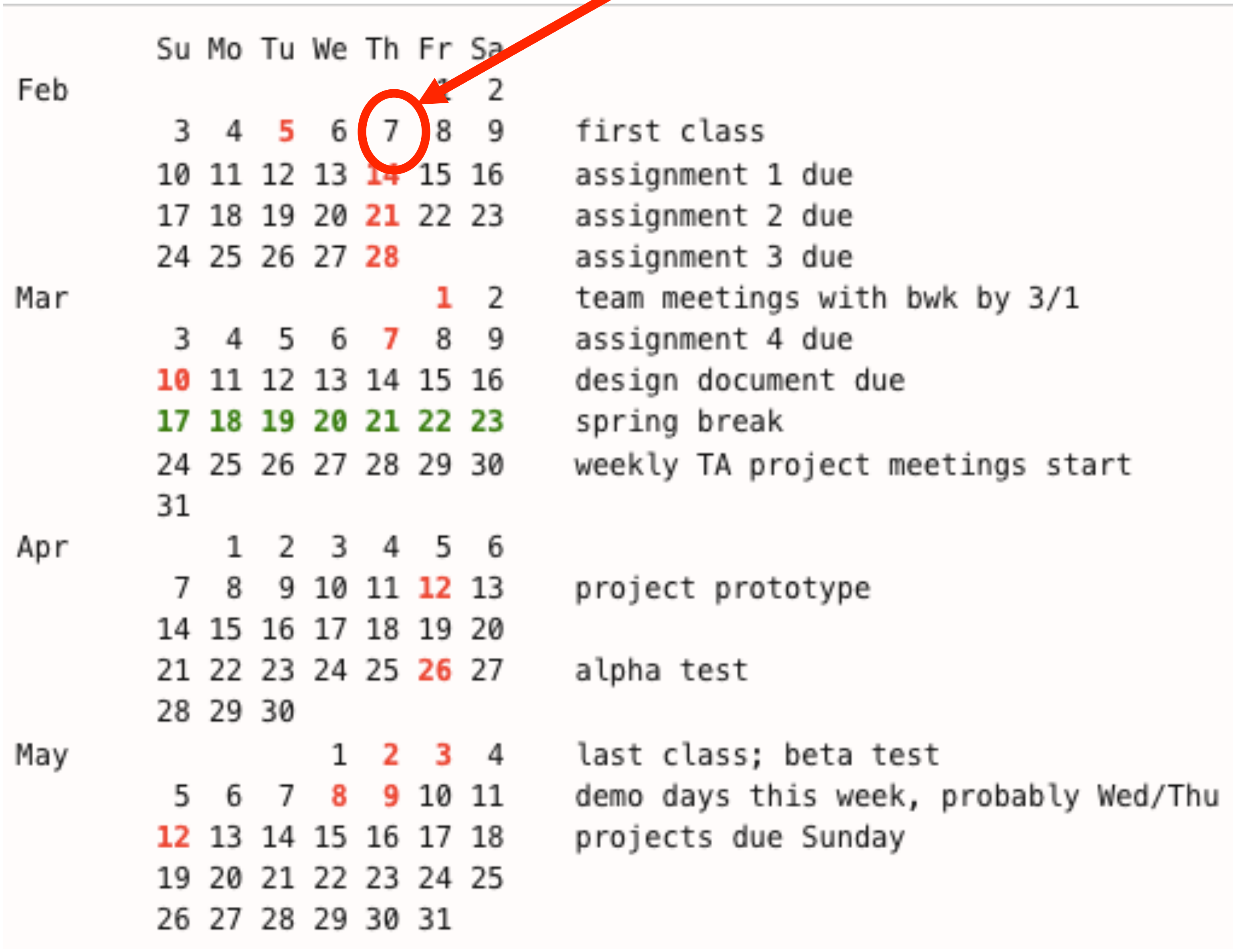
- it's fine to build on what others have done
  - identify what you have used, where it came from
- it’s fine to cooperate with other project groups
  - help each other with insight, knowledge, …

Things to do from the beginning

- think about schedule
  - keep a log of what you did and what you will do next (keep it 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 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!

2019 Project Schedule

You are here



```
     Su Mo Tu We Th Fr Sa
Feb                     1  2
      3  4  5  6  7  8  9     first class
     10 11 12 13 14 15 16     assignment 1 due
     17 18 19 20 21 22 23     assignment 2 due
     24 25 26 27 28           assignment 3 due
Mar                     1  2  team meetings with bwk by 3/1
      3  4  5  6  7  8  9     assignment 4 due
     10 11 12 13 14 15 16     design document due
     17 18 19 20 21 22 23     spring break
     24 25 26 27 28 29 30     weekly TA project meetings start
     31
Apr      1  2  3  4  5  6
      7  8  9 10 11 12 13     project prototype
     14 15 16 17 18 19 20
     21 22 23 24 25 26 27     alpha test
     28 29 30
May            1  2  3  4     last class; beta test
      5  6  7  8  9 10 11     demo days this week, probably Wed/Thu
     12 13 14 15 16 17 18     projects due Sunday
     19 20 21 22 23 24 25
     26 27 28 29 30 31
```

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
  - start now – 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 (maybe before too)
  - 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
  - we're trying to ensure that you don't leave it all to the end

Assignment 4 – a small example

- everyone should experience the full stack
- even if you only work on one part of it for your project

- assignment 4: create a simple interface to Course Offerings
  - read current data from OIT feed in JSON
  - write a server that handles REST queries for courses
  - make a bare-bones web interface
  - use a CSS/Javascript library to make it nicer and responsive
    - w3.css or Bootstrap
  - accumulate information in a database like SQLite
  - [optionally use a server framework like Flask, Bottle]
  - host it on Heroku [or AWS or GCP or ...]
  - [optionally make it run native on phones]

- tentative spec will be posted "real soon now"
- due Thursday March 4 (regular schedule)

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

  ```
  rm '*'
  ```
  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

Quis quotodiet ipsos quotodes?

with apologies to Juvenal and any Latinists in the class

The shell parses these differently:

```
"..."
'...'
```

\ and $ and ` are interpreted within `"..."`

they are NOT interpreted within `'...'`

The International Obfuscated C Code Contest

Obfuscate: tr.v. -cated, -cating, -cates. 1. a. To render obscure. b. To darken. 2. To confuse: His emotions obfuscated his judgement. [LLat. obfuscare, to darken : ob(intensive) + Lat. fuscare, to darken < fuscus, dark.] -obfuscation n. obfuscatory adj.

How it was started:

The original inspiration of the International Obfuscated C Code Contest came from the Bourne Shell source and the finger command as distributed in 4.2BSD. If this is what could result from what some people claim is reasonable programming practice, then to what depths might quality sink if people really tried to write poor code?

I put that question to the USENET news groups net.lang.c and net.unix-wizards in the form of a contest. I selected a form similar to the contest (Bulwer-Lytton) that asks people to create the worst opening line to a novel. (That contest in turn was inspired by disgust over a novel that opened with the line "It was a dark and stormy night.") The rules were simple: write, in 512 bytes or less, the worst complete C program.

Thru the contest I have tried to instill two things in people. First is a disgust for poor coding style. Second was the notion of just how much utility is lost when a program is written in an unstructured fashion...

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