implementation due Thursday
~~are~~ demos scheduled

go over midterm on Thursday

~~continuous integration~~
testing
black box
grey box
white box

black box focuses on input & output
functionality
does it do all the user stories/use cases
set of test cases for each - why useful?
end user point of view
user input validation
reject bad input in a way that end-user understands
non-good
define good input all else assume bad
outputs - are they correct?
how do you know?
test oracle
(we'll learn about what to do if don't know later on)

from external viewpoint
state transitions
map states of full system
not individual objects as in statecharts

boundary cases & off by one errors
users typically make small mistakes & so do programmers

grey box testing
peek beneath covers
independent testers point of view
- verify auditing & logging
- data sent to other systems
check output format + content
- system added information
checksums, hashes, timestamps
- scraps
security risk + resource leak
make sure data really deallocated & deleted
make sure uninstalls cleanly
files, registry entries

example in book
send a picture to other users
considers standard simple case
invalid image & exception handling
lose connectivity while transferring

→ exercise for class - do as a separate tester, not developer
get in groups of ~4
consider how you would test the client side software for phone text messages
you do NOT control server

White box testing
try to break ~~your own code~~

done by developers themselves — (not necessarily same people tho)
- test all branches
  what inputs are needed to force each path?
- proper error handling
  invalid data, cleaning up
- check for thread-safe, null arguments, security roles
- resource constraints
  memory, disk, network connections
  what happens if cannot get?
  need to fail gracefully

code-on-code not exercised by human
use a testing framework like JUnit (nbook)
↗ runs the tests
but does not write them for you

run ALL the tests after every change

" continuous integration

wraps version control, build & testing into single repeatable process

CruiseControl in book (opensource)

code coverage - make sure you tested *all* your code

including exceptional conditions
including GUI

test coverage tools will tell you percent
of functions ← not very useful
of statement (or lines)
of branches
hard to hit 100%, maybe ~~90%~~ 85%

checklist
test success cases - "happy paths"
test failure cases
test backend systems like databases
later → code review ←
match requirements to test cases
test external failure conditions
* network outage disk full
browser shutdown high load
test security problems like
SQL injection, CSS
multiple platforms - OS, browser, hardware

(steps to testing covered later)