Tuesday 9/16/14

remove JW ✓ 4156 as from coursework

intro to TDD

basic idea is write tests before code

typically presented in terms of adding a new feature to an existing system - unit level

and leaves to reader's imagination how to start new system from scratch ("greenfield") - acceptance level

forces developer to 1st think about how to use a system or component, & only then how to implement

So as much a design technique as a testing technique
- results in code that is easy to test & that has tests
- also easy to enhance & adapt

particularly at system/acceptance level, since at component/unit level may have existing design to fit into (or refactor into)

Tuesday 9/16/14 4156

TDD procedure typically presented as

red - write test that fails
(may not even compile)

green - make test pass quickly
(committing w/e sins necessary)

refactor - eliminate redundancy,
improve design,
clean up code

Beck's book presents a miniscule red/green/refactor steps & commits countless sins!

e.g. for a test that passes inputs & checks the result, code that "passes" test ignores input & returns expected result as constant

this is a BAD idea

real projects would not proceed in such tiny steps or with so many sins

Tuesday 9/16/14 4156

TDD relies on tool support

automate testing - JUnit

IDE - may auto-generate missing code skeletons (to make tests compile) - Eclipse

build environment - execute tests during build, check coverage - Maven + plugins

Beck's book describes a full program, but a very small one
- does not address acceptance testing
- tests only at unit level

for a real system, not practical to first write code to pass tests & only afterwards design the system

need to start by defining architecture & designing interfaces between major subsystems - then can write tests to interfaces

Tuesday 9/16/14 4156

by interfaces here, I mean code interfaces — this course does not address UI, assume simple command line or HTML web page for TDD purposes

GUI testing is a big problem but outside our scope

next week we'll study design for now, lets talk about TDD abstractly, at the full system acceptance testing level

we will return to testing later at the code level, including code-specific concepts like coverage

so where do we get these abstract acceptance tests from?
- user stories &, more so, use cases

basic idea is set of (prose) test cases for happy path, then another set of test cases for each alternate flow or scenario

Tuesday 9/16/14 4156

why a set rather than one test?

might be just one, but chances are the alternate flows do not consider every possibility of user inputs, environment state, things that can go wrong

consider laundry use case as example

if time permits, divide into groups & consider pre-assignment from last year