4156

add lnk to Martin Fowler catalog

12/1/15

video at end

"left overs"

book mentions Refactoring many times, but only presents a one-page appendix (read it)

you should know the basic gist of what refactoring is, but do not need to know specific "refactoring patterns" (there are many!)

many ~~most~~ IDEs include a refactoring tool or refactoring plugins are available
e.g. see IntelliJ link

basic idea is to clean up the code to make it easier to maintain (change)

the original idea was to make semantics-preserving transformations, then re-run the regression test suite to make sure nothing broken

in practice, "refactoring" often does change semantics, e.g., by introducing a design pattern or other approach that ~~make~~ reorganizes code so some tests no longer apply

4156 12/1/15

bugs - there will always be more bugs

how to detect them? ideally before end-user does

how to localize them - requires reproduction

how to fix them & check that fix indeed removes bug & does not introduce any new bugs)

how to understand someone else's code

detection - many open problems
e.g., test oracle doesn't exist or is too expensive

so how can you tell if the result computed by a test case is the right answer??

comes up in machine learning, data mining, scientific computing, optimization, simulation, search

One approach: metamorphic testing

4156

12/1/15

reproduction / localization -
also many open problems

you know the behavior is buggy,
but <u>where</u> is the bug in the code?

probably not too hard when
detected by unit testing -
must be that unit

much harder if detected by
full system testing or
by end-user after deployment

may need to execute over & over,
but execution with same input
may not always show bug
non-determinism
thread scheduling

some
approaches: "live" debugging
record / replay
<u>mutable</u> debug / replay
- also helps
check fix works

4156

12/1/15

program understanding -
huge open problem

need to understand code before
can fix bug or otherwise
change, e.g., to add feature

but developers have limited
time & want to spend minimal
time understanding, just
want fix that works

one approach: "code clones"
"similar" code that looks
the same or does the same
possibly, correctly, w/o bug
possibly, more easily, understandable

(also useful to find refactorings)

my lab is seeking project students
for spring 2016 & beyond

can also work on these topics in 6156