4156 10/13/15

design review

read Ron Patton ch. 4
- book is entirely about testing, from perspective of professional tester who is not also a developer

- ch. 4 addresses review of the "specification" i.e., requirements, but some ideas also apply to design (except tester is a developer)

"static testing" - read, don't run
aka review or inspection
(we will discuss code inspection later on)

why? find problems earlier

try to take perspective of a developer who is handed requirements or design to now implement but was not involved in requirements or design (this is hard - which is why real-world reviews often involve developers from other teams or a separate quality assurance group)

4156 10/13/15

does the design adhere to relevant standards & guidelines?

corporate terminology & conventions
industry requirements
government standards
GUI
hardware & networking

what <u>are</u> the standards & guidelines for your product?

e.g. web application

- will your design work for <u>all</u> current generation desktop browsers?
- what about "old" versions of browsers?
- what about "future" versions?
- will your design work for mobile browsers with arbitrary screen sizes?
- what if user has "accessibility" options turn on?
- or is using a screen reader?

4156 10/13/15

review + test "similar" software
is it possible to obtain their
design documentation?

consider
- scale
  memory; disk, network footprint
  # of users

- complexity

- testability
  hooks to aid in testing

- quality / reliability

will your product be better, faster, cheaper?
or not...

I will not expect you to do this
kind of high-level review for your
own projects, BUT... I will ask
you to do the following low-level checks

4156 10/13/15

low-level attributes checklist

complete
- anything missing?

accurate
- will the SW developed from this design fulfill your requirements?

precise, unambiguous, clear
- exact & not vague
- easy to read & understandable
- multiple possible interpretations IS BAD

consistent
- no conflicts?

~~relevant~~

relevant
- is everything necessary & traceable to a requirement?

feasible
- will you be able to do it with available personnel & time?

4156

10/13/15

code free
design shald not include code (maybe algorithms)

testable
think in terms of future unit testings
will you be able to test each operation in isolation?
will you need other operations from same or another class for setup before testing?
- same ok
- another not ok

terminology checklist - more relevant to requirements than design

- always, every, all, none, never
  is it indeed certain/absolute?
- certainly, therefore, clearly, obviously, evidently
  ~~don't~~ don't accept as given
- some, sometimes, often, usually, ordinarily, customarily, most, mostly
  when exactly? too vague
- etc., + so forth, + so on, such as
  lists must be absolute or clear how to generate rest of list

4156 10/13/15

- good, fast, cheap, ethereal, small, stable
  unquantifiable
- handled, processed, rejected, skipped, eliminated
  what does that mean???
- if... then (but missing else)
  what about when if part doesn't happen

others
"friendly"

essential elements of a successful review

identify problems in document, not author
follow rules - distinguish moderator vs. recorder, set time limit
prepare in advance
write report afterwards - summarize problems found, where, what kind
separate from bug reports or change requests (issues)

4156

10/13/15

"code smells"

many evident in design before start coding

long parameter list (to method)

duplication or near-duplication

large class - too many responsibilities

type embedded in method name - forced to change if type changes

uncommunicative name - rename classes, attributes, operations

inconsistent names - standard terminology

speculative generality - YAGNI

temporary field, ~~scratchpad~~ - scratchpad, etc.

alternative classes w/ different interfaces

{ primitive obsession - set of primitives may
{ data clumps - belong in an object

4156 10/13/15

data class - only data, few or no operations

refused bequest - is all inherited functionality used?

inappropriate intimacy - classes know too much about each other

indecent exposure - unnecessarily expose internals (public not private visibility)

lazy class - doesn't do much

middle man - unnecessary wrappers

shotgun surgery - for each class, consider what other classes would need to change

parallel inheritance hierarchies

incomplete library class - ~~class~~ method really should be provided by library but cannot change library

4156 10/13/15

Solution Sprawl - too many classes needed to implement single use case or user story

we will consider these & other "smells" again later when we discuss code review