Tuesday 10/21/14

4156

now its time to start coding & testing
or testing & coding (TDD)

new assignment

but beforehand
- coding conventions (style)
- tools (may come w/ framework)

most languages have a "standard" coding convention, find out what it is & use it

coding conventions generally address:

- commenting & inline documentation

- consistent indentation & line wrapping

- avoid obvious comments
  no "fix this later" or
  "this is a hack"

- code grouping
  blank lines between tasks w/in same method
  (altho usually there should only be one task per method)

Tuesday 10/21/14 4156

- consistent naming scheme

  2 popular options
  camel Case
  under-scores

- DRY principle don't repeat yourself
  copy/paste code is BAD!
  why?? imagine bug fixes
  in one copy but not others
  (aka DIE - duplication is evil)

- avoid deep nesting for sake of readability

- limit line length (e.g., 80 characters)

- file + folder organization
  often supplied by framework or IDE

- consistent names for temporary + loop
  iteration variables
  - these can be short i, j, k, ret

- Uppercase SQL special words to
  distinguish from table + column names

-

Tuesday 10/21/14 4156

tools

- version control
  - check out
  - commit - comment
  - revert to back out of changes
  - merge conflicts
  - tag release/milestone
  - branch/trunk

- build
  - ant, maven, etc..
  - often also does regression testing
    - besides actual build
      - compilation
      - find dependencies
      - package for deployment
      - generate documentation
      - "clean"

continuous integration vs. continuous deployment

IDE - editor, checkers, etc.

~~good coding practices vs bad~~

Tuesday 10/21/14 4156

code "smells"

comments - illuminate vs. obscure
explain why & not what
for people, not machines

long method - a short method is
easier to read, understand,
& troubleshoot
refactor long methods into short

long parameter list - the more
parameters, the more complex
limit the parameters needed,
or use an object to combine

duplicated code - don't repeat yourself
also look for near-duplication

conditional complexity - watch out
for conditional logic blocks
that grow over time
consider instead decorator,
strategy, state design patterns

Tuesday 10/21/14 4156

combinatorial explosion - lots of code that does almost the same thing, but with tiny variations can you use some design pattern to factor out the variation

large classes - like long methods, difficult to read, understand, troubleshoot too many responsibilities? restructure or break into smaller classes

type embedded in method name - redundant, & forces change to method name if type changes

uncommunicative name - does method name succinctly describe what method does, can you tell the name to another developer (who hasn't seen code), & that developer can tell you what it does

inconsistent names - pick standard terminology or metaphor & stick to it

Tuesday 10/21/14 4156

dead code - delete code that isn't being used (if needed, it can be retrieved later from version control)

speculative generality - write code to solve today's problems, not all possible future problems (unless known likely) "you (probably) ain't gonna need it"

oddball solution - only one way of solving the same problem, if more than one maybe need adapter

temporary field - objects that contain optional or unnecessary fields if passing an object as parameter, make sure all public fields are used

alternative classes with different interfaces - ~~can they~~ similar on inside, different on outside, can they be modified to share common interface

primitive obsession - don't use gaggle of primitive data type variables, combine in a class

Tuesday 10/24/14 4156

data class - avoid classes that passively store data, usually should have both methods & data

data clumps - if data travels together, roll into a class

refused bequest - if inheriting from a class but never use anything inherited, should inheritance be used?

inappropriate intimacy - classes should know as little as possible about each other

indecent exposure - refactor classes to minimize public surface

feature envy - methods that make extensive use of another class should perhaps move to that class

lazy class - can a class not doing much be collapsed into another class

Tuesday 10/21/14

message chains - avoid long sequences of method calls or temporary variables extra dependencies

middleman - if a class is delegating all its work, why does it exist

divergent change - if different changes affect different parts of class, over time, maybe separate into another class

shotgun surgery - if changes to one class cascade to other classes should refactor to limit changes to a single class

parallel inheritance hierarchies - consider folding into single hierarchy if always must change both

incomplete library class - need a method that's missing so gets tacked onto another class

solution sprawl - simplify & consolidate design → leads to hard to write unmaintainable code