Clean Code by Robert C. Martin

Alexander Olivero

April 10, 2019

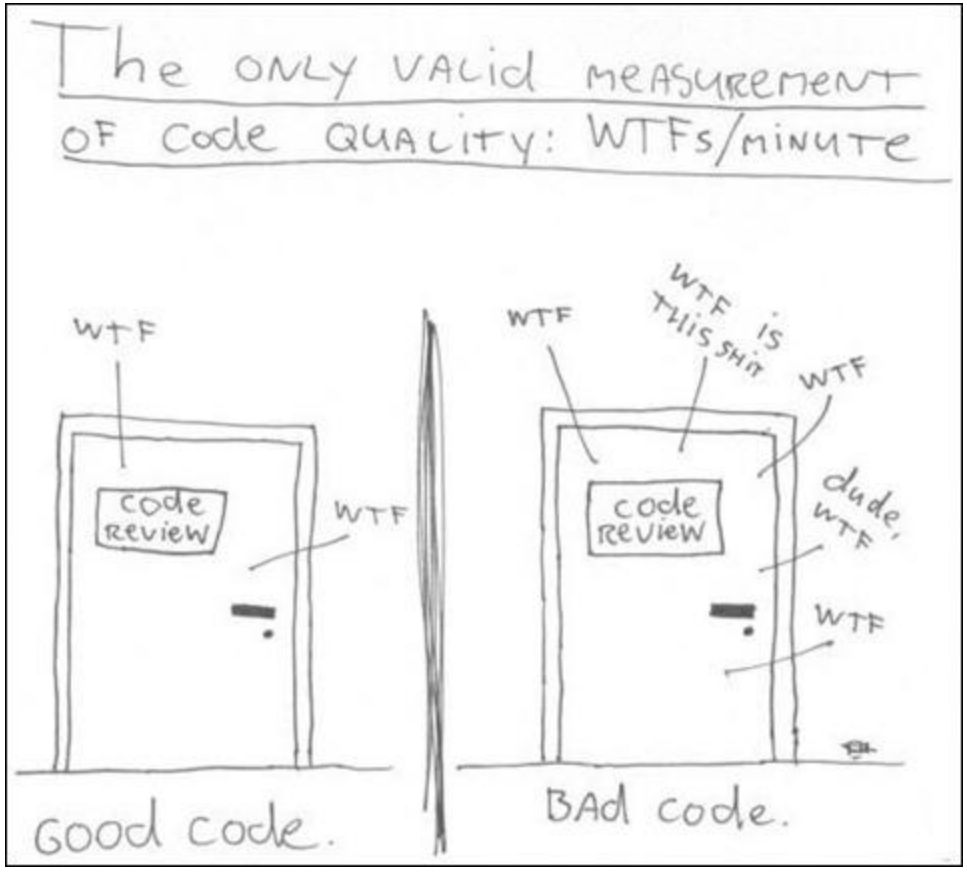
Overview

- About the author
- Code as a craft
- Summary of principals of clean code

About the author

- commonly called "Uncle Bob"
- one of the authors of the Agile Manifesto
- prolific writer on software development best practices
    - 10 published books and a fairly active blog

Code as a craft



Code as a craft

"There are two parts to learning craftsmanship: knowledge and work. You must gain the knowledge of principles, patterns, and practices, and heuristics that a craftsman knows, and you must also grind that knowledge into your fingers, eyes and guy by working hard and practicing."

What is "Clean Code"?

- Bad code can bring a company down
  - slow releases, bloated features, hard to patch, slow load times
- Most people can identify good art, but few can paint
- "Clean code never obscures the designer's intent, but rather is full of crisp abstractions and straightforward lines of control."
- Code reading to writing ratio is well over 10:1
- Boy Scouts Rule: Leave the campground cleaner than you found it

Meaningful Names

- Intention revealing
- Pronounceable
- Searchable
- Classes and objects should be nouns, functions and methods should be verbs
- Avoid disinformation
- Avoid mental mappings

Functions

- Should be small (hardly ever 20 lines)
- "Do one thing, do it well, do it only."
- Less arguments is better
- Flag arguments are ugly
- DRY Principle

Comments

"Don't comment on bad code, rewrite it."

- Comments are a necessary evil
- Proper use is to compensate for our failure to express ourself in code
- Get rid of commented out code
- Good naming can replace comments

Formatting

- Small files are usually easier to understand than large files
- Vertical openness between concepts -> each expression should be separated by a blank line
- Where things are vs. how things work
- Variables should be declared as close to their usage as possible
- Consistency is the most important aspect for a team

Objects and Data Structures

"Procedural code makes it easy to add new functions without changing the existing data structures. Object-Oriented code makes it easy to add new classes without changing existing functions."

- Law of Demeter: a module should not know about the innards of the objects it manipulates
- Train Wreck Code: linked code that should be split up

  `object.getOptions().getDir().get_Path()`

Error Handling

- If it obscures logic, it's wrong
- Use try-catch-finally paradigm to control final state
- Exceptions should provide enough context to determine source and location of error

Unit Tests

Three Laws of TDD

1. Don't write production code until you have a failing unit test
2. Don't write more of a test than is sufficient to fail
3. Don't write more production code than is necessary to pass the failing test

Having "quick and dirty" test is equivalent to, if not worse than, having no tests as all.

Tests should be as well written as production code.

Systems

- Dependency Injection
  - Inversion of Control: object should not be responsible for instantiating dependencies itself
- Myth: We can get systems right the first time
- Myth: we need to do Big Design Up Front (BDUF)
- Never forget to use the simplest thing that can possible work

Design

Kent Beck's Simple Design (ordered by importance):

1. Runs all the tests
2. Contains no duplication
3. Expresses intent of the programmer
4. Minimizes the number of classes and methods

The majority of cost of a software project is maintenance

Successive Refinement

- Best way to ruin a program is to make massive changes to structure in the name of improvement
- Satisfaction with simply "code that works" is unprofessional
- Easier to clean a mess made 5 minutes ago rather than 5 months ago