Clean Code (why not do it)

Diego Pacheco

About me...

- ❑ Cat's Father
- ❑ Principal Software Architect
- ❑ Agile Coach
- ❑ SOA/Microservices Expert
- ❑ DevOps Practitioner
- ❑ Speaker
- ❑ Author

diegopacheco

@diego_pacheco

http://diego-pacheco.blogspot.com.br/

https://diegopacheco.github.io/

The boy scout rule

"Always leave the code you're editing a little better than you found it"

- Robert C. Martin (Uncle Bob)

Clean Code: Comments

- comments
- + Good Comments
- + Informative
- + Explain
- + Bad Comments
- + Redundant Comments
- + Journal Comments (// 1 ok 2 not ok 3 john did 4 luke did it...)
- + Noise Comments (/* var x */
- + Don't use a comment when it could be a variable or a function
- + Poison Markers (//////////// action)
- + Commented Out Code (DEAD)
- - Too Much Information

Clean Code: Emergent Design, Concurrency, Refactoring

- Emergence design ++ (existed prior clean code)
- Concurrency (existed prior clean code)
  - + Thread-Safe Collections
  - + Beware of dependencies being sync
  - + Keep Synchronization small
  - + Testing Thread Code
- refactoring / Smells (existed prior clean code)
  - + Inappropriate information
  - + Obsolete Comment
  - + Redundant Comment
  - + Poorly Written Comment
  - + Commented Out Code
  - + Duplication
  - + Dead Code

Clean Code: Naming

- ❑ +- intention revealing
- ❑ + avoid disinformation
- ❑ + avoid member_prefix
- ❑ - classes (i.g Processor)
- ❑ - avoid mental mapping (k,j)
- ❑ - searchable name (everything is searchable)

Clean Code: Functions

- ❑ +- small
- ❑ +- do one thing
- ❑ +- Prefer Exception
- ❑ + DRY (excited before clean code)
- ❑ + Have no Side Effects (FP not created by clean code)

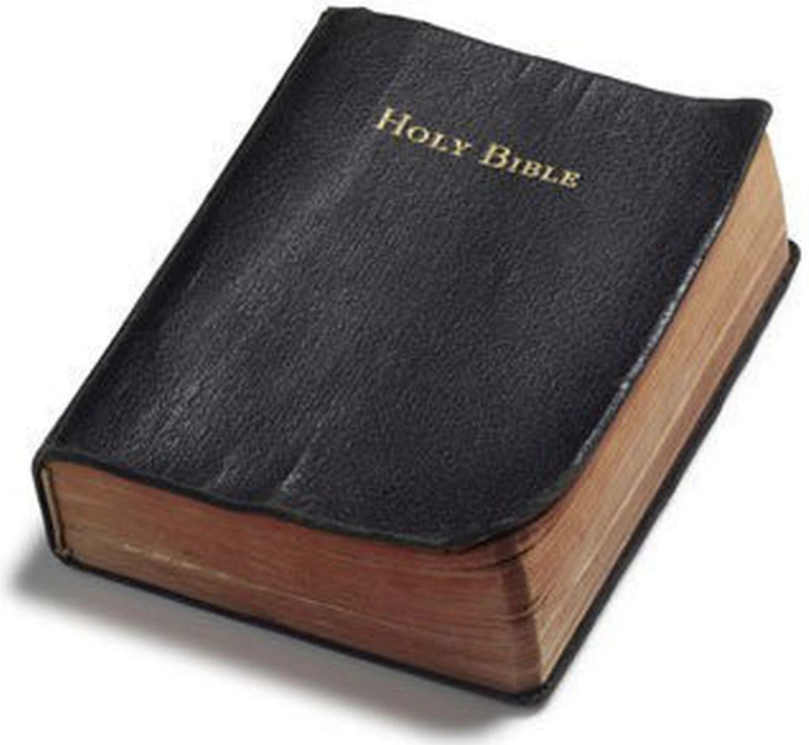
Clean Code: Formating, objects and data structures, errors

- Formatting
  - - Pure Style (Go Default lang per land and you should be fine) / Diff CR/PR Matters.
- Objects and data structures
  - + Law of Demeter (Module should not know about innards of objects it manipulates)
  - + Hiding Structure
  - +- DTO
- Error Handler
- +- Exception rather return codes
- +- Use Unchecked Exceptions
- +- Define Exceptions in terms of called needs
- + Don't return null (FP / Defensive Programing ... pior clean code)
- + Don't pass null (FP / Defensive Programing ... pior clean code)

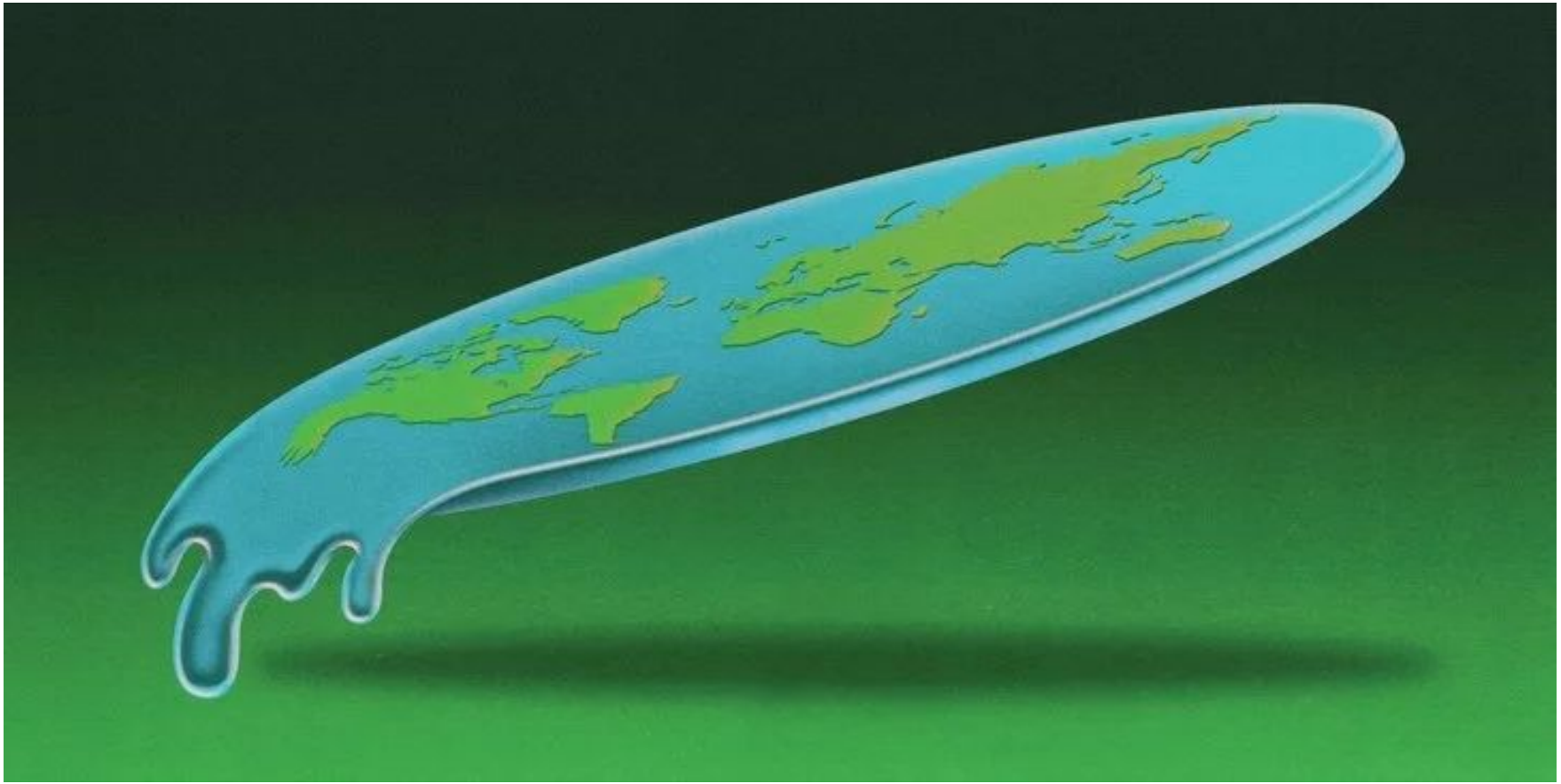
Clean Code: Boundaries, Unit Testing, Classes, Systems

- Boundaries
  - -+ Learning boundaries are hard, Tests to understand (Prefer POC)
- Unit Testing
  - - TDD (Does not improve design, JR devs)
  - - One Assert Per Test
- Classes
  - -+ Classes should be small
  - - Cohesion (existed prior clean code)
- Systems
  - + DI (really good for testing)
  - - EJB
  - + Java Proxy / AOP (Spring uses a lot for TXs mgmt) -ORM Sucks (uses proxy CGLIB)

Clean code / TDD as a Cult == No Thinking(sometimes not even read the book...)

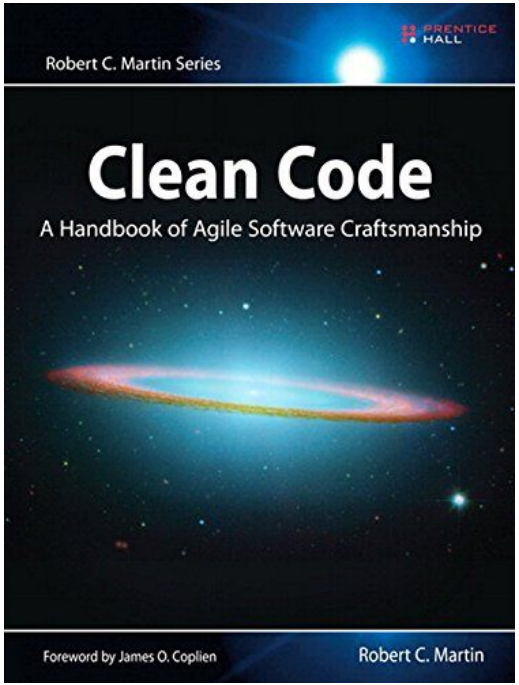


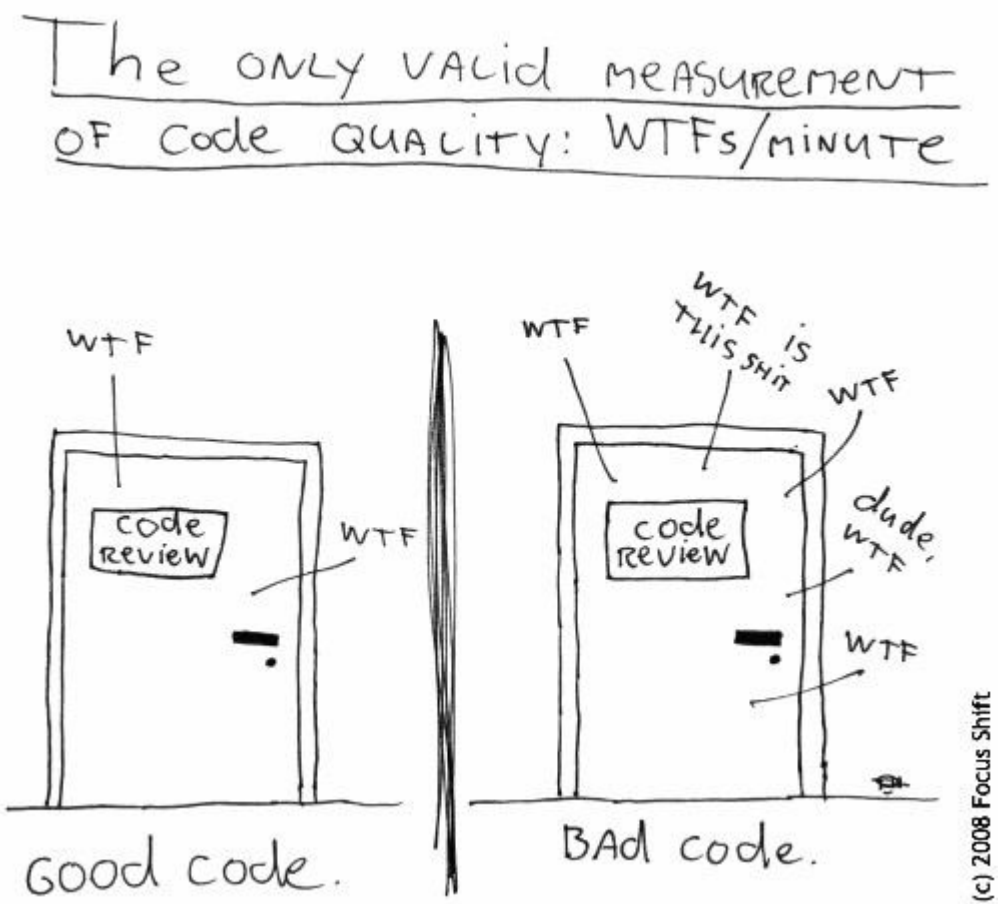
Clean code / TDD is the "Flat Earth" of Engineering



Clean Code

Start as good intention...





Clean Code

- Clean Code misses the point
- Agile misses architecture badly
- Architecture / Design are far more important than clean code
  - SOA > Clean Code
  - Microservices > Clean Code
  - Design > Clean Code
  - Interfaces > Clean Code
- Why ?
  - Does not matter for the caller
  - It's abstracted anyway
  - Can be refactored
  - What "clean" really mean?

Clean Code == Style

What's really matters?

Flexibility

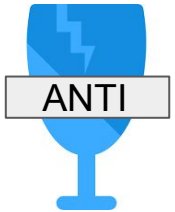
Architecture

Design

Reliability

Performance

Scalability



Anti-Fragility

Testability

Exercises

Constraints

1. You can use Java, Scala or Clojure

1. Revier all your previous code(previous exercises) make sure they follow the #8 principles listed (What's really matters).
2. Fix at least 3 codes (if applies)

Clean Code (why not do it)

Diego Pacheco