CLEAN CODE

By: Jan de Vries

E-mail: jan@jan-v.nl

General

Title:	Clean Code A Handbook of Agile Software Craftmanship
Author:	Robert C. Martin
Pages:	349
Chapters:	17
Target Audience:	Developers
Slides:	29

Goal

- Awareness
- Tips on how to produce better code
- Readability
- The ‘boy scout rule’
- Starting out with tips and examples
- Later on real world examples
- It’s not the bible!

Smart/Professional

- Smart developer
  - Difficult code
  - Has great developing skills
  - `r` = lowercase url

- Professional developer
  - Readable code
  - Maintanable code
  - 'Clarity is king'
  - `lowercaseUrlOfCurrentPage` = lowercase url

5S philosophy

- Seiri – Organize/Sort
- Seiton – Systemize/Tidiness
- Seiso – Cleaning
- Seiketsu – Standardization
- Shutsuke - Discipline

Bad code

- Go fast?
- Angry boss?
- Tired of project?
- Get working now, clean up later?
- Everybody does it!

What happens

- Redesign of system
- Everybody wants in
- Everything the old system does + better
- Takes a long time
- Team members leave

Why

- Requirements change and not meet design
- Schedules too tight
- Stupid managers
- Intolerant customers
- Useless sales

Why (2)

It’s us, the developers!

Why (3)

- They ask **us** for information
- If they don't, make **yourself** noticed
- Users ask **us** if requirements fit the system
- Project managers ask **us** to help with schedule

Developer part

Meaningfull names

- Take your time
- Not `theList`
- Searchable names
  - Constants
- Hungarian Notation
  - IDE
- Member prefixes
- Interfaces

Meaningfull names (2)

- Mental mapping
  - i, j, k, l
- Pronouncable
- 1 word per concept
  - Get, retrieve, fetch
  - Controller, manager, driver
- Use domain names
  - Read by programmers

Functions

- Small
  - < 150 characters per line
  - < 20 lines
- Blocks, indenting
- 1 thing!
- Arguments
- Side effects
- Prefer exceptions

Functions (2)

- Extract try-catch
- Error handling is 1 thing
- Don't repeat
- Not all at once

Comments

- **Don't**
- Self-explaining
- If you can't do any better
- Clarification
- Warning
- TODO

Comments (2)

- Redundant
- Misleading
- Noise
- Copy-pasting
- Use a function or variable
- Closing brace comments
- Commented out code
- Version control!

Formatting

- Variable declarations
  - Where they are used
- Instance variables
  - Top of class
- Dependancy
  - Top-down
- Not 1-line functions
- Team rules
  - 10 minutes
  - IDE formatter

Objects & data structures

- Law of Demeter
  - Method `f` of a class `C` should only call the methods of these:
    - `C`
    - An object created by `f`
    - An object passed as an argument to `f`
    - An object held in an instance variable of `C`
- Objects
- Data structures

Error handling

- Narrow down exceptions
- Provide context
- Don’t return null; Don’t pass null
  - Prevents null-checking

Boundaries

- Exploring & learing
  - Experiment & learn
  - Reading manual
- Using non-existing code
  - Interface
  - Write your own

Unit tests

- Three laws of TDD
  - You may not write production code until you have written a failing unit test
  - You may not write more of a unit test than is sufficient to fail, and not compiling is failing
  - You may not write more production code than is sufficient to pass the currently failing test

Unit tests (2)

- Keeping tests clean
  - Maybe more important as actual code
- No fear of refactoring
- One assert per test
- Single concept per test

Unit tests (3)

- F.I.R.S.T.
  - Fast
  - Independent
  - Repeatable
  - Self-validating
  - Timely

Classes

- Should be small
- Single Responsibility Principle
  - SQL → Select, Update
- Easy to change code

Systems

- Start small
- Add features later

Concurrency

- Keep synchronized blocks small
- Don’t share objects
- Suspicious failures
  - Are bugs, not cosmic glitches

Summary

- Start with bad code and clean it
- Keeping things small
- Keeping code readably
- Self-explaining
- Standards
- Unit testing
- Interfaces, abstract classes, OO
- **You** are responsable!

Questions