Fiche de lecture : "Clean Code a Handbook of Agile Software Craftmanship" Auteur : Robert C. Martin

1 Introduction
- Safari Enable Book Online
- There is great power in details
- Take the time to go fast
- Read 10 / write 1 : you write a story
- Function do one thing and do it wel
- Same abstract level
- Boy scout rule : Leave the campground cleaner than you found it

2 Meaningful Names
- Searchable name
- Not encoding Interface (ShapeFactory ShapeFactoryImp)
- Class : nouns , not be a verb
- Method : to verb...
- Rename and rename
- Negative are harder to understand than positives

3 Functions
- Very Small
- Do one thing, do it well, do it only
  - Same abstract level
  - DoSomething or answerSomething not bot
- Reading top to bottom
- Switch, only in factory, never repeated,
- SRP : Single Responsability principe (more than one reason to change it
- OCP : Open close principe (Must change whenever new type added)
- Arguments
  - No argument best, one, two, three, four too much
  - Don't transform argument, transformation should appear as return value
  - Flag argument ugly
  - Wrapped argument in class
  - Prefer report.AppendFooter() than public void appendFooter(String report)
- Prefer exception to returning error Codes (Extract body of try in method)

4 Comments
- Missleading, nonLocal and dishonest
- Prefer refactoging than comment

5 Formatting
- Single formating style for a team
- Concept close, vertically close
  - Variable
  - Dependent Function

6 Object and Data Structures
- Law of Demeter (Hide data, expose operations)
- Train Wrecks (Chains of call)
- Object make it easy to add new kinds of object, hard to add new behaviors to existing object
- Data structure make easy to add new behavior but hard to add new data structures to existing functions

7 Error Handling
- Exception rather than return codes
- Begin by Try-Catch-Finally and extract method
- Checked exception is an OCP violation, use unchecked exceptions
- Provide context with exception
- Define Exception in term of caller's needs
- Don't return null (return Collections.emptyList()
- Don't pass null (Not ideal solution)

8 Boundaries
- Learning test for third-party code
- Encapsulate to meet the need
- Using code that does not yet exist with interface

9 Unit Tests
- Keep test clean and readable because test keep code flexible
- One Assert per test, single concept per test
- F.I.R.S.T (Fast, Independent, Repeatable, Self-Validating, Timely)

10 Classes
- Small and SRP (Single Responsability Principle(one responsibility one reason to change)
- Short description in about 25 words
- organizing for change
  - Support OCP and SRP make class open for extension
  - Minimizing coupling with interface make test easy, code flexible, promote reuse and adhere to DIP (Dependency Inversion Principle)
- Make function protected for test it
- Cohesion (each method manipulate one or more instance variable)
  When class lose cohesion, split.

11 Systems
- Separate startup (construct, wire objects) from runtime logic
  - Factories
  - Dependency Injection
    - IOC
    - JNDI lookup
  - EJB2 did not separate concerns adequately
  - Proxies (CGLIB, ASM, Javassist)
  - AOP
- Use simplest thing that can possibly work

12 Emergence
- According to Kent -> design simple if (Runs all the test, Contains no duplication, Expressive, Minimizes number of classes and methods

13 Concurrency
- Keep your concurency related code separate
- SRP
- Encapsulate, limit shared data, attempt to partition data into subset than can be operate on by independant threads.
- Keep synchronized section small
- Know your library (Thread safe, executor...)
- Do not ignore system failuress as one-offs
- Think about shutdown early
- Run with more thread than processors on different platforms

14 Sucessive refinement
- It s not enought for code to work
- Bad schedule can be redone, bad requirement can be redifined, bad team dynamics can be repaired, bad code is very expensive to clean. (breaking dependencies is hard) Keeping code is relatively easy.
- Bad code dominate your destiny

15 JUnit Internals (Refactoring instance)

16 Refactoring SerialDate (Refactoring instance)

17 Smell and Heuristics