Principles in Refactoring
(chapter 2)

Refactoring is

changing

the internal structure of software

keeping

the same observable behavior

Goal

- The goal is to make the software easier to understand (clean up code, reduce complexity) without noticeable behavior changes.

- Performance optimization, instead, is not a goal!

Writing Tests

- Do not add any new tests, only restructure the code
- Only change existing tests to reflect occurred interface changes

The Two Hats

- Add new functions first
- Then refactor them considering the previous existing code

Regular refactoring helps

- Improving the design of software
- Tiding up the code, removing duplications
- Finding bugs
- Writing robust code
- Programming faster

When to refactor

- After writing duplicated code
- After adding new functions
- On fixing bugs
- On code reviewing

Management

- Non-technical managers are not concerned by refactoring, thus hard to accept and schedule them.
- In any case during the development process, refactoring has to be performed

Indirection

- Breaking big objects and big methods in smaller parts, reducing the complexity:
  - Sharing logic (i.e. helper method, superclass)
  - Explain intention (i.e. method name)
  - Isolate change
  - Encode conditional logic
- But it introduces the need of manage more objects and functions
  - Ensure then to avoid parasitic indirection

Problems with Refactoring

- Databases coupling
- Changing (published) interface
- Introducing exception throws
- Code is too messy and buggy (candidate to rewrite)
- Unfinished refactoring at the deadline

Design

- Do not try to have the perfect design on the first run
- Re-design over refactoring, better understand the problem to solve
- Flexibility never needed leads to overheads
- Simple solution first, then refactor for needed flexibility

Bad Smells in Code
(chapter 3)

- Duplicated code
  - Dry! Reuse helper method
- Long method
  - Decompose in smaller methods
- Large class
  - Decompose in subclasses
- Long parameter list
  - Only pass what always required, call methods to get additional needed data
  - Pass map, keeping unchanged the interface
- Divergent change
  - Separate classes to reduce dependencies on change
- Shotgun surgery
  - Collect methods depending on common change
- Feature envy
  - Implement methods within the responsible classes