Quality Code

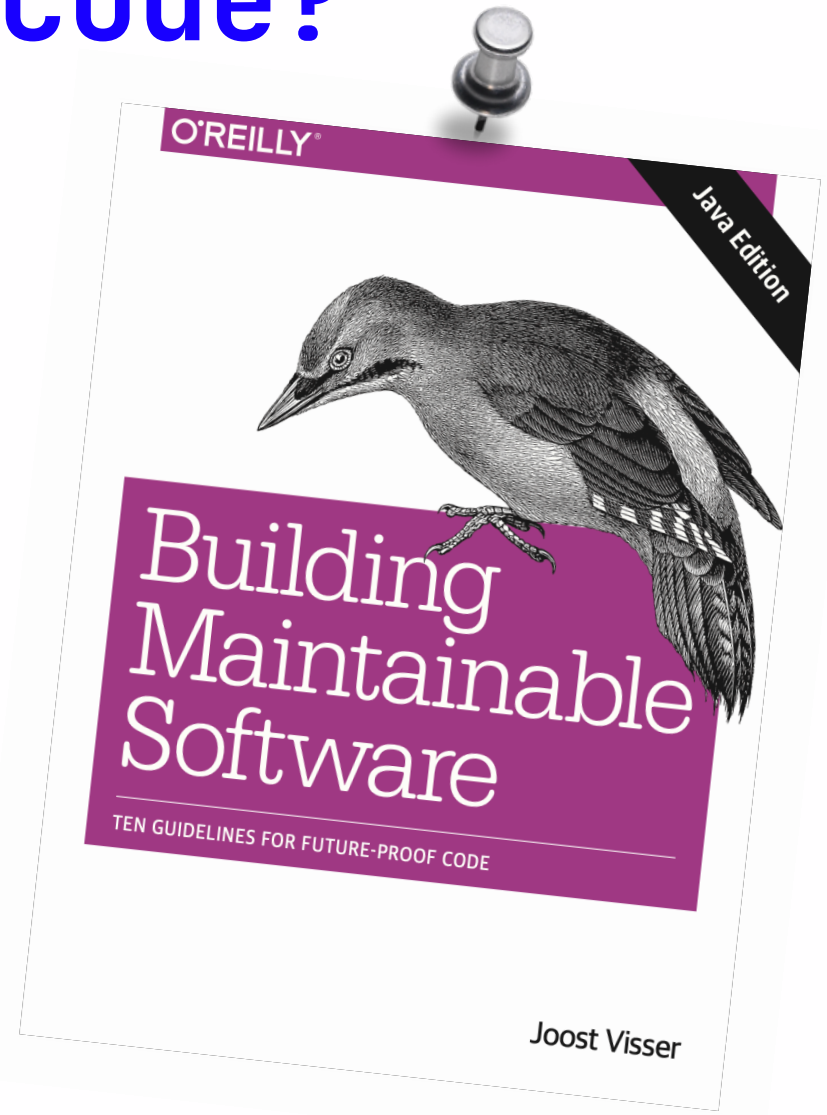
Software Systems — Design — L6T5

Dr. Vadim Zaytsev aka @grammarware, November/December 2020

UNIVERSITY OF TWENTE.
Formal Methods & Tools

How to Write Quality Code?

- Quality code is
  - clear to understand
  - pleasant to work with
  - easy to change
- These guidelines are
  - generic
  - widely applicable
  - inspired by industrial practice



https://www.softwareimprovementgroup.com/resources/ebook-building-maintainable-software/

Write Short Methods

- Shorter methods are easier to
  - analyse
  - test
  - reuse
- 10–20 lines is a good heuristic
  - in practice 95%+ have 1–15 LoC
- Use refactoring
  - e.g. Extract Method

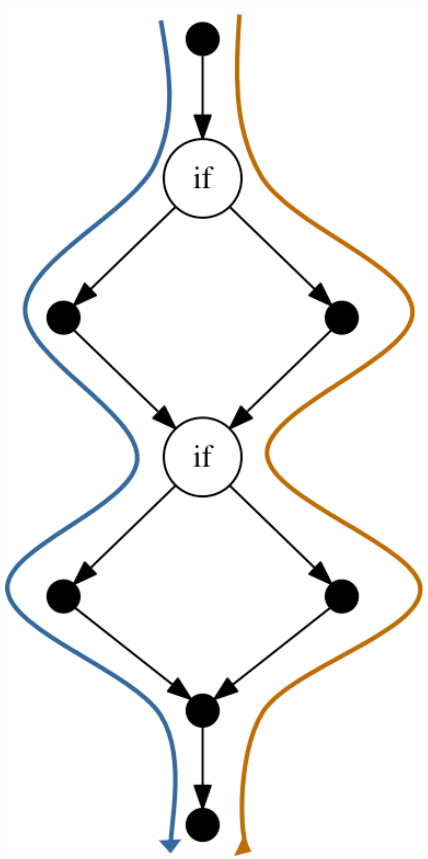
Give Meaningful Names

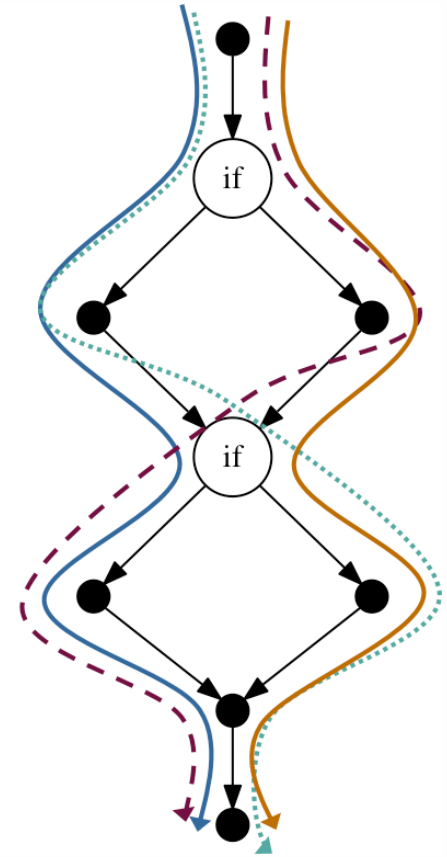
- x, y, z
  - good for maths
- i, j, k
  - good for loops (local throwaway indices)
- Good names are
  - understandable
  - revealing the intentions
  - searchable

Code Complete, Part III; *Clean Code*, Chapter 2

Write Simple Methods

- Methods with fewer branching points are easier to
  - analyse
  - test
  - debug
- 4–5 points is a good heuristic
- if/case/while/for/catch
  - ?: && ||





BMS, Chapter 3; cf L6T1

Write Code Once

- Copying is easy
  - reuse is not much harder
- When you copy code, you also copy its bugs
  - hard to analyse
  - hard to modify
- Heuristic: **do not copy code**.
- From a different place?
  - sometimes

BMS, Chapter 4

Keep Method Signatures Small

- Methods with fewer parameters are easier to
  - reuse
  - test
- Heuristic: 3–4 parameters per method
- Need more?
  - group together into objects

Separate Concerns in Classes

- Smaller classes
  - depend less on one another
  - are easier to modify
  - force to think about encapsulation
- Isolated maintenance is good

Keep Components Balanced

- Not too many
- Not too few
- No trivially small ones
- No monoliths
- No surprises
- Heuristic:
  - 5–10 components
  - of comparable size

BMS, Chapter 8

Automate

- Manual work does not scale
- You deserve feedback
- Automate the build process
- Automate testing
- Automate the deployment
- Generate code from models

Keep Your Code Clean

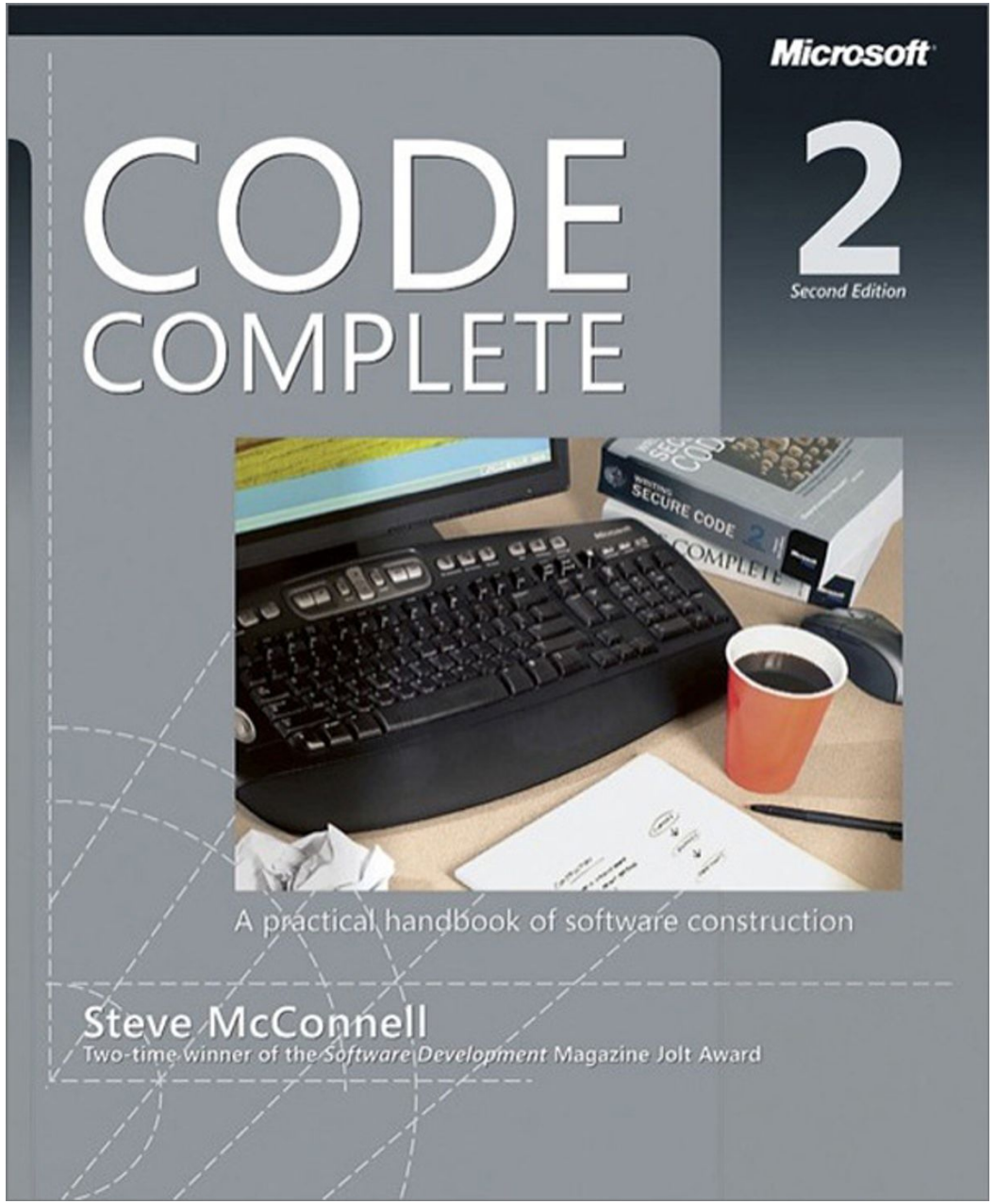
- `// TODO`
- `return true; // return Flag;`
- Comments do not compensate for bad code!
- Tests are meant to be green
  - unless in active development

Keep Improving

All successful software gets changed.
Fred Brooks

- Emergent design
  - Design is a wicked problem!
- Stick to the rules
  - keep it clean, short, maintainable
- Rewrite & refactor code
  - clones, nesting, length, abstraction
- Reserve time for this
- Large systems are hard to maintain

BMS, Chapter 9; *Clean Code*, Chapter 12 & 14



Steve McConnell, *Code Complete*, Chapter 24

CHECKLIST: Reasons to Refactor

- ❑ Code is duplicated.
- ❑ A routine is too long.
- ❑ A loop is too long or too deeply nested.
- ❑ A class has poor cohesion.
- ❑ A class interface does not provide a consistent level of abstraction.
- ❑ A parameter list has too many parameters.
- ❑ Changes within a class tend to be compartmentalized.
- ❑ Changes require parallel modifications to multiple classes.
- ❑ Inheritance hierarchies have to be modified in parallel.
- ❑ *case* statements have to be modified in parallel.
- ❑ Related data items that are used together are not organized into classes.
- ❑ A routine uses more features of another class than of its own class.
- ❑ A primitive data type is overloaded.
- ❑ A class doesn't do very much.
- ❑ A chain of routines passes tramp data.
- ❑ A middleman object isn't doing anything.
- ❑ One class is overly intimate with another.
- ❑ A routine has a poor name.
- ❑ Data members are public.
- ❑ A subclass uses only a small percentage of its parents' routines.
- ❑ Comments are used to explain difficult code.
- ❑ Global variables are used.
- ❑ A routine uses setup code before a routine call or takedown code after a routine call.
- ❑ A program contains code that seems like it might be needed someday.

Topics/Slides Disclaimer

- Good ✅
  - watch before Q&A
  - embrace reality
  - try out at labs
  - ask for feedback
  - apply to project
  - dig deeper
  - recall from slides
- Bad ❌
  - slides over videos
  - assumptions
  - blanks
  - timing