Boswell's The Art of Readable Code

- **Publisher**: O'Reilly
- **Author**: Dustin Boswell
- **Presenter**: Wen-Bin Luo
- **Link**: https://www.amazon.com/Art-Readable-Code-Practical-Techniques/dp/0596802293

Contents

Surface-Level Improvements

Packing Information into Names

- Pack information into your names.
- The name retval doesn't pack much information. Instead, use a name that describes the variable's value.
- The name tmp should be used only in cases when being short-lived and temporary is the most important fact about that variable.
- If you're going to use a generic name like tmp, it, or retval, have a good reason for doing so.
- Having different formats for different entities is like a form of syntax highlighting.

Names That Can't Be Misconstrued

- The best names are ones that can't be misconstrued.
- The clearest way to name a limit is to put max_ or min_ in front of the thing being limited.
- When naming a boolean, use words like is/has/can/use to make it clear that it's a boolean.

Aesthetics

- If multiple blocks of code are doing similar things, try to give them the same silhouette.
- Aligning parts of the code into “columns” can make code easy to skim through.
- Pick a meaningful order and stick with it.
- Use empty lines to break apart large blocks into logical “paragraphs.”
- Consistent style is more important than the “right” style.

Knowing What to Comment

- The purpose of commenting is to help the reader know as much as the writer did.
- Don’t comment on facts that can be derived quickly from the code itself.
- Rule: good code > bad code + good comments.

Making Comments Precise and Compact

- Avoid pronouns like “it” and “this” when they can refer to multiple things.
- Describe a function’s behavior with as much precision as is practical.
- Illustrate your comments with carefully chosen input/output examples.
- State the high-level intent of your code, rather than the obvious details.
- Use inline comments to explain mysterious function arguments.
- Keep your comments brief by using words that pack a lot of meaning.
- Comments should have a high information-to-space ratio.

Simplifying Loops and Logic

Making Control Flow Easy to Read

- Make conditionals, loops, and other changes to control flow as “natural” as possible.
- A guideline that can be useful:
  - Left-hand side: The expression “being interrogated,” whose value is more in flux.
  - Right-hand side: The expression being compared against, whose value is more constant.
- Instead of minimizing the number of lines, a better metric is to minimize the time needed for someone to understand it.
- By default, use an if/else. The ternary ?: should be used only for the simplest cases.
- Look at your code from a fresh perspective when you’re making changes. Step back and look at it as a whole.

Breaking Down Giant Expressions

- Break down your giant expressions into more digestible pieces.
- The purpose of a *summary variable* is simply to replace a larger chunk of code with a smaller name that can be managed and thought about more easily.
- Beware of “clever” nuggets of code, which are often confusing when others read the code later.

Variables and Readability

- The more variables there are, the harder it is to keep track of them all.
- The bigger a variable’s scope, the longer you have to keep track of it.
- Make your variable visible by as few lines of code as possible.
- To restrict access to class members, make as many methods static as possible.
- **Block scope**: variables defined inside a block are confined to the nested scope of that block:
- The more places a variable is manipulated, the harder it is to reason about its current value.

Reorganizing Your Code

Extracting Unrelated Subproblems

- Aggressively identify and extract unrelated subproblems.
- Separate the generic code from the project-specific code.
- General-purpose code is great because it’s completely decoupled from the rest of your project.
- You should never have to settle for an interface that’s less than ideal.

One Task at a Time

- Code should be organized so that it’s doing only one task at a time.

Turning Thoughts into Code

- Describe what code needs to do, in plain English, as you would to a colleague.
- Pay attention to the key words and phrases used in this description.
- Write your code to match this description.

Writing Less Code

- The most readable code is no code at all.
- Keep your codebase as small and lightweight as possible.
- Create as much generic “utility” code as possible to remove duplicated code.
- Remove unused code or useless features.
- Keep your project compartmentalized into disconnected subprojects.

Selected Topics

- Testing and Readability
- Designing and Implementing a “Minute/Hour Counter”

Testing and Readability

- Test code can be seen as unofficial documentation of how the real code works and should be used.
- Test code should be readable so that other coders are comfortable changing or adding tests.
- In general, you should pick the simplest set of inputs that completely exercise the code.
- Prefer clean and simple test values that still get the job done.
- **Test-driven development (TDD)**: write the tests before the real code.