Chapter 3 Functions:

Small!

• Keep them concise.

• Avoid exceeding 100 lines in length.

Blocks and Indenting

• Keep blocks within statements succinct, ideally one line.

• Avoid nesting deeply.

• Limit indent levels to one or two.

Do One Thing

• Ensure functions have a single responsibility, executed well.

• Avoid multifunctional behaviors; split into distinct functions if needed.

Sections within Functions

• Functions adhering to the single responsibility principle cannot be logically subdivided.

One Level of Abstraction per Function

• Maintain consistency in abstraction levels within functions to enhance clarity.

Reading Code from Top to Bottom: The Stepdown Rule

• Understand code by descending through levels of abstraction.

• Mastering this technique is vital for maintaining concise, focused functions.

Use Descriptive Names

• Clean code yields routines that match expectations.

• Concise functions enable clear, descriptive naming.

• Employ consistent and informative naming conventions.

Function Arguments

• Aim for zero or few arguments.

• Justify any arguments exceeding three, considering testing complexity.

• Output arguments introduce complexity and should be minimized.

Common Monadic Forms

• Functions should either pose a question or perform an operation.

• Clearly denote event-calling functions.

• Avoid using output arguments for transformations.

Flag Arguments

• Avoid passing Boolean flags, as they complicate function calls.

Dyadic Functions

• Functions with two arguments are more intricate to grasp than monadic ones.

Triads

• Functions with three arguments are notably more complex than dyadic ones.

Argument Objects

• Consider encapsulating arguments into dedicated classes.

• Provide meaningful names for argument objects.

Verbs and Keywords

• Choose descriptive names to clarify function purpose.

• Form verb/noun pairs for functions and arguments.

Have No Side Effects

• Side effects introduce ambiguity and coupling, undermining code reliability.

Output Arguments

• Prefer interpreting arguments as inputs to functions.

• Minimize reliance on checking function signatures for clarity.

Command Query Separation

• Segregate functions into either performing actions or providing information.

Prefer Exceptions to Returning Error Codes

• Reserve error handling for dedicated functions to maintain clarity.

Error Handling is One thing

• Limit functions to handling errors exclusively.

Don’t Repeat Yourself

• Reduce code duplication to prevent bloating and potential errors.

Structured Programming

• Ensure each function has a single entry and exit point.

System Development Perspective

• Recognize systems as domain-specific languages.

• View functions as verbs and classes as nouns.

• Approach system development akin to storytelling.

Chapter 4 Comments

 Nothing can be quite helpful than a well-placed comment

 Comments are necessary evil

 Compensate for our failure to express ourself in code

 Inaccurate comments are far worse than no comments at all

Comments Do Not Make Up for Bad Code

• Comments should not be used as a band-aid for poorly written code.

• Prioritize code cleanliness over excessive commenting.

Explain Yourself in Code

Legal Comments

• Adhere to corporate coding standards for legal requirements like copyright and authorship statements.

Informative Comments

• Provide basic informational context through comments when necessary.

Explanation of Intent

• Comments should elucidate the reasoning behind code decisions, not just provide factual data.

Clarification

• Translate complex or ambiguous elements of code into understandable language.

Warning of Consequences

• Use comments to alert other programmers about potential pitfalls or side effects.

TODO Comments

• Utilize comments to mark unfinished tasks or future actions.

Bad Comments

• Avoid using comments as an excuse for subpar code quality.

Amplification

• Use comments to emphasize the importance or significance of specific code sections.

Misleading Comments

• Ensure comments accurately reflect the code; imprecise comments are worse than none at all.

Nonlocal Information

• Place comments where they are relevant to the nearby code.

Inobvious Connection

• Comments should seamlessly integrate with the code and not require additional explanation.