Bible for Software Engineers: Clean Code by Robert C. Martin

Chapter 2: Meaningful Names

- **Core Idea**: Good naming is essential. Code should tell a story without needing extra commentary.
- **Key Practices**:
  - Names should be *descriptive*, *specific*, and *pronounceable*.
  - Avoid abbreviations or misleading terms.
  - Choose names that reveal intent — e.g., use `getActiveAccount` rather than `getData`.
- **Impact**: Enhances readability and maintainability; reduces reliance on comments.

Chapter 3: Functions

- **Core Idea**: Functions should be **small** and do **one thing only**.
- **Best Practices**:
  - Use meaningful names and limit arguments (prefer none or one).
  - Avoid side effects and deep nesting.
  - Structure functions for clarity, not cleverness.
- **Impact**: Leads to modular, testable, and scalable code that's easier to debug and refactor.

Chapter 4: Comments

- **Core Idea**: Comments are a last resort — code should ideally be self-expressive.
- **Proper Usage**:
  - Use comments to explain *why* something is done, not *what* the code does.
  - Avoid redundant or misleading comments.
  - Prefer expressive naming and clean structure to reduce comment necessity.
- **Impact**: Promotes transparency while discouraging clutter and outdated annotations.

Chapter 6: Objects and Data Structures

- **Core Idea**: There's a key trade-off between **objects (behavior-focused)** and **data structures (data-focused)**.
- **Concepts**:
  - Objects encapsulate behavior and hide data; structures expose data and hide behavior.
  - Understand when to use getters/setters vs. procedural access.
  - Favor principles like encapsulation and abstraction for complex systems.
- **Impact**: Equips students to design architectures that balance clarity, flexibility, and control.