Chapter 2 – Meaningful Names – Summary

Must-Have or Recommended Practices:

- Use intention-revealing names.
- Choose names that answer the "why," "what it does," and "how it's used" questions.
- Prioritize clarity and readability over excessive brevity.
- Use meaningful names related to the problem or solution domain to avoid confusion.
- Use clear and descriptive class and method names.
- Maintain consistency in naming and avoid puns (using the same term for different ideas).
- Add meaningful context to names through well-named classes, functions, or namespaces.

Naming Clarity:

- Avoid ambiguous or overly short names.
- Avoid names that require comments to explain their meaning.
- Avoid using lowercase "l" or uppercase "O" as variable names since they closely resemble the constants one and zero.
- Choose names that can be easily pronounced.
- Ensure that names are distinguishable to prevent confusion.
- Use technical and problem domain names appropriately, depending on the audience's familiarity with the concepts.

Naming to Avoid:

- Avoid names with established meanings that differ from the intended meaning, as it can be misleading.
- Be cautious with abbreviations like "hp" that might be commonly associated with other concepts.
- Ensure that terms like "list" accurately represent their programming context.
- Avoid minor spelling variations in similar names to prevent confusion.
- Avoid using single-letter names and numeric constants, as they are hard to locate in a body of text.
- Avoid adding type or scope information to names as encodings.
- Avoid names that are difficult to pronounce and may make you sound unclear or unprofessional.

Naming to Ensure:

- Different names should have different meanings, not just different spellings.
- Noise words (e.g., Info, Data) that don't add meaning should be avoided in names.
- Noise words are redundant; avoid using words like "variable" or "table" in variable or table names.
- Prefix names only when necessary; avoid gratuitous context that makes code harder to work with.
- Keep names clear and concise, prioritizing clarity over brevity.

Chapter 3 – Functions – Summary

Best Practices:

- Keep functions concise.
- Aim for shorter functions.
- Maintain one level of abstraction.
- Focus on single responsibilities.
- Use clear, descriptive names.
- Minimize function arguments.
- Prefer exceptions over error codes.
- Separate commands and queries.
- Use try/catch blocks judiciously.
- Emphasize clear and precise code.

Things to Avoid:

- Large switch statements.
- Functions with multiple responsibilities.
- Violating the Single Responsibility or Open-Closed Principles.
- Output arguments.
- Complex error codes.
- Hidden side effects.
- Excessive function arguments.
- Lengthy, unclear function names.
- Goto statements.
- Excessive code duplication.

Writing Functions:

- Start with working code.
- Refine and reduce complexity.
- Use unit tests for correctness.
- Tell a clear coding story.

Conclusion:

- Code like storytelling.
- Construct a rich language.
- Emphasize well-organized functions.