Naming Things Book Review

Diego Pacheco

Book Scope / Impressions

- Name identifiers - classes, functions, and variables
- “A bit of clean code feelings” but a bit better.
- Rules
- Samples

Understand code

- Maintenance takes 60-80% of the Code
- 40-60% of the time is spent trying to understand the Code
- 25-50% of the time is expended trying to make sense of software
- Identifiers (variables, classes, function names) are 70% of the code base.
- Identifiers influence:
  - Comprehension: Engineer's ability to understand the Code's logic, structure, and purpose.
  - Recall: Engineer's ability to remember the Code.
  - Defer Resolution time: Engineers' time to fix a bug.
- Some research says fixing a bug with single-word identifiers vs multi-word identifiers: Multi-word people were able to fix the bugs 14% faster.
- 19% found more defects faster when words were used as identifiers.

Understand code

Can we quantify the importance of a codebase's understandability? Studies show that maintenance consumes 60%-80% of software development life cycle costs. [8] Approximately 40%-60% of this effort is spent understanding the software that is being modified. [9] Thus, 25%-50% of life cycle time (the product of the previous numbers) is spent understanding the software.

Why naming is hard?

- Newly introduced concepts are often poorly described.
- Definitions often evolve over.
- Naming is bound to a domain that evolves over time
- Different engineers with different domain backgrounds will need to understand the naming
- Naming is a problem beyond software engineering (Brand Naming, Terminology Planning). Even among the general public, there is little agreement on names.
- Studies say that the chance of two different people naming the same object the same is between 7-18%.

Naming Principles

- List of principles
  - Understandability: A name should describe the concept that it represents.
  - Conciseness: Only use the words necessary to express the concept.
  - Consistency: Names should be used and formatted consistently.
  - Distinguishability: The name should be visually and phonetically distinguishable from another.

Rules :: Classes

Identifier type	Parts of speech	Examples
Classes	Nouns or noun phrases	`User`, `PaymentMethod`
Variables	Nouns, noun phrases, or linking verb and subject complement	`name`, `birth_date`, `is_valid`
Methods	Verbs, verb phrases, or linking verb and subject	`validate`, `delete_all`, `is_valid`

Rules

- ❏ Include units in the measurement
- ❏ Avoid un-conventional single-letter names
- ❏ Avoid Abbreviations
- ❏ Avoid non-standard symbolic names “<<_>>”
- ❏ Avoid Cleverness (No Humor)
- ❏ The code should be boring
- ❏ Avoid the usage of temporary or irrelevant concepts (kill_em_all(processes))
- ❏ Consider if the audience is familiar with the term

Rules :: Conciseness

Conciseness

- Long identifiers take more time to read and process in your mind
- Rules
  - Use the appropriate level of abstraction
  - Details of implementation should not be present on the method name (because code changes)
  - Use words with rich meaning
  - Omit metadata (name_string)
  - Omit implementation details
  - Omit Unnecessary words

Rules :: Consistency

Consistency

- Rules
  - Obey popular conventions - Language conventions.
  - Avoid synonyms
  - Avoid abbreviations
  - Use consistent antonyms (add/remove) - (install/uninstall)

Rules :: Distinguishability

Distinguishability

- Rules:
  - Avoid homographs and near-homographs (multiple meanings), which are the same spelling but have different meanings.
    - invoice.check() # Bad
    - invoice.validate() ; invoice.written_check() # Good
  - Avoid Polysemy
  - Avoid names with distinct technical and non-technical meanings (other examples: window, event, transaction)
    - class Class # BAD
    - class Course #Good

Naming Things Book Review

Diego Pacheco