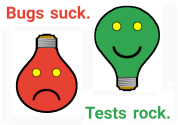
Testing on the Toilet Presents... ***Healthy Code on the Commode***



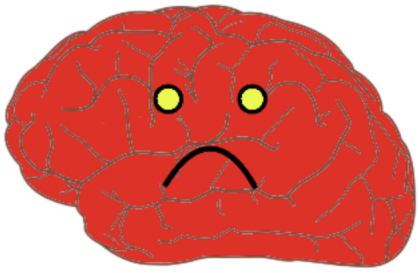
Write Clean Code to Reduce Cognitive Load

by Andrew Trenk

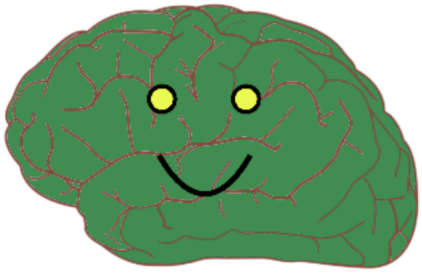
Do you ever read code and find it hard to understand? You may be experiencing cognitive load!

Cognitive load **refers to the amount of mental effort required to complete a task**. When reading code, you have to keep in mind information such as values of variables, conditional logic, loop indices, data structure state, and interface contracts. **Cognitive load increases as code becomes more complex**. People can typically hold up to 5–7 separate pieces of information in their short-term memory (source); code that involves more information than that can be difficult to understand.

Cognitive load is often higher for other people reading code you wrote than it is for yourself, since readers need to understand your intentions. Think of the times you read someone else's code and struggled to understand its behavior. One of the reasons for code reviews is to allow reviewers to check if the changes to the code cause too much cognitive load.
Be kind to your co-workers: reduce their cognitive load by writing clean code.



Complex code:
Too much cognitive load



Simple code:
Minimal cognitive load

The key to reducing cognitive load is to *make code simpler* so it can be understood more easily by readers. This is the principle behind many code health practices. Here are some examples:

- *Limit the amount of code in a function or file.* Aim to keep the code concise enough that you can keep the whole thing in your head at once. Prefer to keep functions small, and try to limit each class to a single responsibility.
- *Create abstractions to hide implementation details*. Abstractions such as functions and interfaces allow you to deal with simpler concepts and hide complex details. However, remember that over-engineering your code with too many abstractions also causes cognitive load.
- *Simplify control flow.* Functions with too many `if` statements or loops can be hard to understand since it is difficult to keep the entire control flow in your head. Hide complex logic in helper functions, and reduce nesting by using early returns to handle special cases.
- *Minimize mutable state.* Stateless code is simpler to understand. For example, avoid mutable class fields when possible, and make types immutable.
- *Include only relevant details in tests.* A test can be hard to follow if it includes boilerplate test data that is irrelevant to the test case, or relevant test data is hidden in helper functions.
- *Don't overuse mocks in tests.* Improper use of mocks can lead to tests that are cluttered with calls that expose implementation details of the system under test.

Learn more about cognitive load in the book The Programmer's Brain, by Felienne Hermans.

More information and archives: `testing.googleblog.com`