The Linus Torvalds Taste Test

The author claims that the core of the taste requirement is "eliminating edge cases." However, this does not explain why it's natural that good-taste code might be more efficient, seen in both examples in the reading. What is another way of describing the "good taste" requirement which also explains why good code is often more efficient?

Answer

One student puts it nicely:

Code that is in good taste equates one conceptual state of the program with one representation at compile time and run time. Each of the poor-taste examples could be in multiple states when taking the same action from a conceptual standpoint. In efficiency terms, fewer representations implies fewer runtime checks, fewer branches, less control-flow in general.

Another interpretation, which I think is valid if less directly linked to the theme of this module, is that good taste is characterized by code that doesn't check what it can prove by construction. In Linus's example, the `indirect` pointer is definitionally the thing to be updated, so there's no need to check and see what it's pointing to. In Barto's code, his first attempts were checking whether his loop variable referred to a point along one of his four edges, whereas his ending state was code that constructed those points so that they were already known to be edge points.

Bugs and Battleships

ezyang talks about splitting software into different cases, each of which corresponds to one test. Give some examples of these cases. What causes a case to split into two?

Answer

All disjunctions in the preconditions/postconditions of a line of code. Explicit conditionals in the code correspond to a subset of these disjunctions.

E.g.: input is null or not; input is in bounds or not.

But there can also be cases in the specification.

E.g.: If you implement the absolute value function as `sqrt(x*x)`, there are still two cases to test because the mathematical definition has two cases.

The Most Dangerous Code in The World

What would the libcurl developers need to do to be able to simplify [the API] again?

Answer

Break all existing applications by turning it into a 0-1 binary choice or using something other than integers to make the choices.

Note that, if they wanted to use an enum to represent these values, I'd consider making the enum start at a higher value and giving a runtime error on 0/1, because C types do not distinguish enums from ints from bools.

The big point: They're screwed. There is no backwards-compatible way to do this.

Where to Draw the Boundary

How does Plato's/Yudkowsky's concept of "carving reality at its joints" manifest when designing data structures?

Answer

One student's answer:

> If we want to represent things in our data, we first have to determine what those things are. That can be difficult. Is this a File, or is it a Download? Or is a File a part of a Download. Is a Download even a thing?
>
> Often we have to choose where the joints are. For example, it would be particularly difficult to model something like mental illness where the categories are very much man made, and can shift. I've been in many meetings when designing systems where clients realize they don't have a clear idea of their business concepts, and if that lack of clarity continues (often for political reasons, and because boundaries can be naturally blurred) the business can be hard to model.
>
> I work in Property Tech at the moment. And we often have discussions about where to carve reality. Can you have a semi-detached bungalow for example. Technically yes, but that's not how they're presented to the user. Only houses can be semi-detached. But isn't a bungalow a type of house. Etc.

Jimmy's addendum: And any confusion in this modeling of the reality will yield confusion in the code (e.g.: being inconsistent about where certain functionality is handled, it being unclear when a menu option should be enabled vs. not, etc).