Code, without tests, is not clean. No matter how elegant it is, no matter how readable and accessible, if it hath not tests, it be unclean.

In priority order, simple code:

• Runs all the tests;

• Contains no duplication;

• Expresses all the design ideas that are in the system;

• Minimizes the number of entities such as classes, methods, functions, and the like.

**Meaningful Names**

### **USE INTENTION-REVEALING NAMES**

Choosing names that reveal intent can make it much easier to understand and change code.

### **AVOID DISINFORMATION**

We should avoid words whose entrenched meanings vary from our intended meaning. Spelling similar concepts similarly is information. Using inconsistent spellings is disinformation. A truly awful example of disinformative names would be the use of lower-case L or uppercase O as variable names, especially in combination.

### **MAKE MEANINGFUL DISTINCTIONS**

or example, because you can’t use the same name to refer to two different things in the same scope, you might be tempted to change one name in an arbitrary way. Sometimes this is done by misspelling one, leading to the surprising situation where correcting spelling errors leads to an inability to compile.

It is not sufficient to add number series or noise words, even though the compiler is satisfied. If names must be different, then they should also mean something different.

In the absence of specific conventions, the variable moneyAmount is indistinguishable from money, customerInfo is indistinguishable from customer, accountData is indistinguishable from account, and theMessage is indistinguishable from message. Distinguish names in such a way that the reader knows what the differences offer.

### **USE PRONOUNCEABLE NAMES**

If you can’t pronounce it, you can’t discuss it without sounding like an idiot.

### **USE SEARCHABLE NAMES**

Single-letter names and numeric constants have a particular problem in that they are not easy to locate across a body of text.

### **AVOID ENCODINGS**