Art of Readable Code

PART IV

1. Testing and Readability

Main purpose for writing test case is **to check the behavior of another (“real”) piece of code.**

Points to consider:

- Make Tests Easy to Read and Maintain
- Making The Test More Readable
- Making Error Messages Readable
- Choosing Good Test Inputs
- Naming Test Functions
- Test-Friendly Development

1.1 Make Tests Easy to Read and Maintain

KEY IDEA: **Test code should be readable so that other coders are comfortable changing or adding tests.**

When test code is big and scary, here's what happens:

• **Coders are afraid to modify the real code.** Oh, we don't want to mess with that code— updating all the tests would be a nightmare!

• **Coders don't add new tests** when they add new code. Over time, less and less of your module is tested, and you are no longer confident that it all works.

1.2 Making The Test More Readable

```
void MakeScoredDoc(ScoredDocument* sd, double score, string url) {
    sd->score = score;
    sd->url = url;
}
```

Using this function, our test code becomes slightly more compact:

```
void Test1() {
    vector<ScoredDocument> docs;
    docs.resize(5);
    MakeScoredDoc(&docs[0], -5.0, "http://example.com");
    MakeScoredDoc(&docs[1], 1, "http://example.com");
    MakeScoredDoc(&docs[2], 4, "http://example.com");
    MakeScoredDoc(&docs[3], -99998.7, "http://example.com");
    ...
}
```

1.2 Making The Test More Readable

```
void AddScoredDoc(vector<ScoredDocument>& docs, double score) {
    ScoredDocument sd;
    sd.score = score;
    sd.url = "http://example.com";
    docs.push_back(sd);
}
```

ing this function, our test code is even more compact:

```
void Test1() {
    vector<ScoredDocument> docs;
    AddScoredDoc(docs, -5.0);
    AddScoredDoc(docs, 1);
    AddScoredDoc(docs, 4);
    AddScoredDoc(docs, -99998.7);
    ...
}
```

1.3 Making Error Messages Readable

➢ **Unreadable Error Messages:**

In Python, the built-in statement **assert(a==b)** produces a plain error message like:

File "file.py", line X, in <module>

    assert a == b

AssertionError

What were the values of a and b???

➢ **Readable Error Messages:**

Instead, you can use the **assertEqual(a,b)** method in the unittest module:

File "file.py", line X, in <module>

    assertEqual(a, b)

AssertionError: 1 != 2

1.4 Choosing Good Test Inputs

Key Idea: **In general, you should pick the simplest set of inputs that completely exercise the code**.

For example: here are four tests for SortAndFilterDocs():
CheckScoresBeforeAfter("2, 1, 3", "3, 2, 1"); // Basic sorting
CheckScoresBeforeAfter("0, -0.1, -10", "0"); // All values < 0 removed
CheckScoresBeforeAfter("1, -2, 1, -2", "1, 1"); // Duplicates not a problem
CheckScoresBeforeAfter("", ""); // Empty input OK

1.5 Naming Test Functions

Key Idea: Avoid Naming meaningless names like Test1(), Test2(), etc.

Instead, you should use the name to **describe details about the test with a "Test_" prefix.**

For example:

Test_<FunctionName>() format:

- void **Test_SortAndFilterDocs()** { ... }

Or Test_<FunctionName>_<Situation>() format:

- void **Test_SortAndFilterDocs_BasicSorting()** { ... }

1.6 Test-Friendly Development

Key Idea:

“ Keeping testing in mind while writing code helps make the code better. ”

Summary - Testing And Readability

Here are specific points on how to improve your tests:

- The top level of each test should be as clearly as possible; ideally, each test input/output can be described in one line of code.
- If your test fails, it should emit/release an error message that makes the bug easy to track down and fix.
- Use the simplest test inputs that completely exercise your code.
- Give your test functions a fully descriptive name so it's clear what each is testing. Instead of Test1(), use a name like Test_<FunctionName>_<Situation>