**Chapter 14 Testing and Readability**

* Testing means different things to different people.
* A piece of code whose purpose is to test another code.

**Make Tests Easy to Read and Maintain**

* if tests are readable, users will understand clearer how the code behaves.
* test codes should be readable coders are comfortable changing and adding test.
* When test codes are not readable:
  + coders are afraid to modify the real code.
  + coders don’t add new tests.

**Making Test More Readable**

* Always hide less important details from the user, so more important details are most prominent.

**Creating Minimal Test Statement**

* the technique that was introduced in the previous chapter turning thoughts into code can be applied also in making minimal test statement.



**Implementing Custom Minilanguages**

* creating a helper class that will do simple things.

**Making Error Message Readable**

* getting detailed error message will help the tester identify the problem.
* indicating the value of the variables will be more helpful in identifying the error.
* there are builtin libraries that can help you in printing the error like BOOST\_REQUIRE\_EQUAL() from C++.

**Hand-Crafted Error Messages**

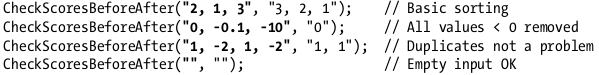
* writing your own error message will be more readable by you and the tester.

**Choosing Good Test Inputs**

* good inputs should thoroughly test the code but should also be simple so that they’re easy to read.
* pick the simplest set of inputs that completely exercise the code.
* test input should be able to trigger all possible outcomes of the code.
* prefer clean and simple test values that still get the job done.
* large inputs, on the other hand helps in exposing a lot of bugs like buffer overruns, like stress testing.
* hard coding large test inputs is not effective in stress-testing. Constructing large inputs programmatically will be more effective.

**Multiple Test of Functionality**

* more effective way in creating test code and more readable is to write multiple smaller tests.



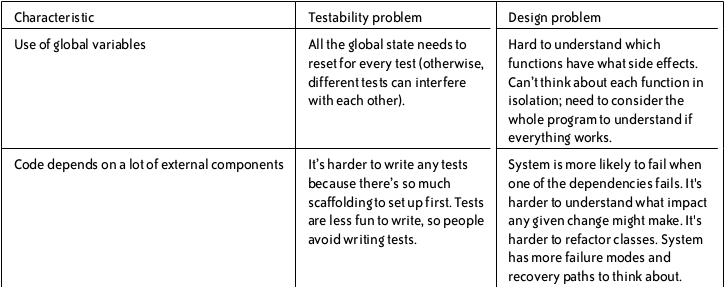
**Naming Test Functions**

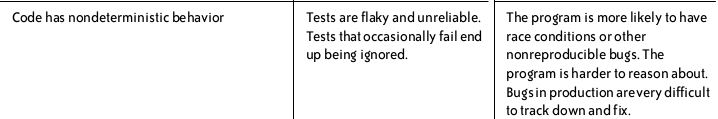
* use the name to describe details about the test.
* its better if the tester can figure out:
  + the class being tested.
  + function being used.
  + situation or bug being tested.
* good way in naming function is concatenate that information together with “Test\_” as prefix.
* test function can also be separated in different function depending on the situation being tested.
* in naming **helper functions** it’s useful if the function does any assertions itself or just an ordinary “test-unaware” helper.

**Test-Friendly Development**

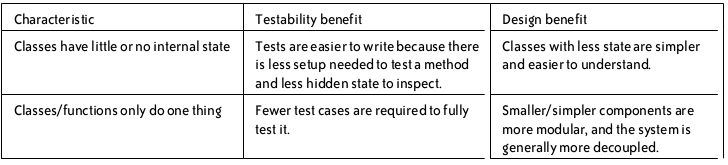
* in writing a code, there must be a design in which creating a test for the written code will be easy too.
* writing in this way also leads in having an organized code.

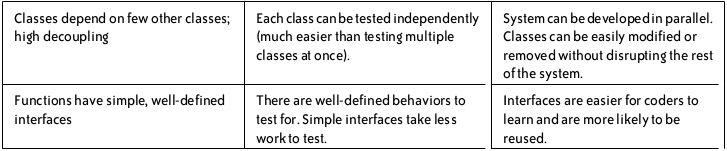
Characteristic of less testable code:





Characteristic of a testable code:





* Realizations and Facts:
  + Designing the real code to be testable should be a win-win situation. Codes must be simple and decoupled and at the same time are easy to write.
  + Testing the first 90% will be simpler than testing the last 10%. There are many factors that could contribute the bugs like missed feature or the spec might be change. So you really cannot test your code 100%.