Nicole P. Satiembre March 27, 2024

Cebu Technological University-Main Campus

Clean Code

Chapter 9-Unit Tests

The evolution of software engineering has come a long way, before, no one heard of Test Driven Development. Modern developers now prioritize comprehensive test coverage, leveraging techniques such as isolation, mocking, and continuous integration to ensure the reliability and efficiency of their code. While we have come a long way, we recognize that further refinement is needed to fully harness the potential of testing practices, with some developers still needing to grasp the subtler nuances of writing effective tests amidst the rush to adopt these new methodologies.

The Three Laws of TDD.

First Law. You may not write production code until you have written a failing unit test.

Second Law. You may not write more of a unit test than is sufficient to fail, and not compiling is failing.

Third Law. You may not write more production code than is sufficient to pass the currently failing test.

Keeping Tests Clean. Having dirty tests is equivalent to, if not worse, having no tests. The problem arises as the tests evolve together with the production code. The dirtier the tests, the harder they are to change. As you modify the production code, old tests start to fail, and the mess in the test code makes it hard to get those tests to pass again. So the tests become viewed as an ever-increasing liability. Test code is just as important as production code. It requires thought, design, and care. It must be kept as clean as the production code.

Tests Enable the -ilities. If you don't keep your tests clean, you will lose them. Unit tests keep our code flexible, maintainable, and reusable. It helps us feel safe when making changes to our code. So having an automated suite of unit tests that cover the production code is the key to keeping your design and architecture as clean as possible. Tests enable all the -ilities, because tests enable change.

Clean Tests. Readability is what makes a clean test, and is perhaps even more important in unit tests than it is in production code. To make our tests readable, we must prioritize clarity, simplicity, and density of expression. Anyone who reads these tests should be able to work out what they do very quickly, without being misled or overwhelmed by details.

Domain-Specific Testing Language. Rather than using the APIs that programmers use to manipulate the system, we build up a set of functions and utilities that make use of those APIs and that make the tests more convenient to write and easier to read. These functions and utilities become a specialized API used by the tests. They are a testing language that programmers use to help themselves write their tests and to help those who must read those tests later on.

Dual Standard. The text highlights the dual standard of prioritizing cleanliness and readability in testing code while being more lenient on issues of memory or CPU efficiency that would be crucial in production environments. While testing code should still be simple, succinct, and expressive, it doesn't need to prioritize efficiency to the same extent as production code since it runs in a test environment rather than a live one. By abstracting away unnecessary details and focusing on the overall behavior being tested, developers can make test code easier to read and understand.

One Asset Per Test. Every test function in a JUnit test should have one and only one assert statement. The number of asserts in a test ought to be minimized.

Single Concept per Test. A good rule to follow is to test a single concept in each test function and minimize the number of asserts per concept.

F.I.R.S.T.

Fast. Tests should be fast. They should run quickly.
Independent. Tests should not depend on each other.
Repeatable. Tests should be repeatable in any environment.
Self-Validating. The tests should have a boolean output. Either they pass or fail.
Timely. The tests need to be written in a timely fashion. Unit tests should be written just before the production code that makes them pass.

Conclusion. Think of tests like the guardians of a project's well-being—they're just as vital as the actual code. In fact, some argue they're even more crucial because they maintain and improve the code's flexibility, ease of maintenance, and ability to be reused. So, it's important to always keep your tests neat and tidy. Make sure they're easy to understand and to the point. And don't hesitate to create special tools (like testing APIs) that make writing tests a breeze.