Nguyen Quoc Vuong

[Email address]

Unit Testing Overview

[Document subtitle]

Contents

[Diving into the four pillars of a good unit test 2](#_Toc190013977)

[The first pillar: Protection against regression 2](#_Toc190013978)

[The second pillar: Resistance to refactoring 2](#_Toc190013979)

[What causes false positives? 2](#_Toc190013980)

[Aim at the end result instead of implementation details 2](#_Toc190013981)

[The intrinsic connection between the first two attributes 2](#_Toc190013982)

[Maximizing test accuracy 2](#_Toc190013983)

[The importance of false positives and false negatives: The dynamics 2](#_Toc190013984)

[The third and fourth pillars: Fast feedback and maintainability 2](#_Toc190013985)

[In search of an ideal test 2](#_Toc190013986)

[Is it possible to create an ideal test? 2](#_Toc190013987)

[Extreme case #1: End-to-end tests 2](#_Toc190013988)

[Extreme case #2: Trivial tests 2](#_Toc190013989)

[Extreme case #3: Brittle tests 2](#_Toc190013990)

[In search of an ideal test: The results 2](#_Toc190013991)

[Exploring well-known test automation concepts 2](#_Toc190013992)

[Breaking down the Test Pyramid 2](#_Toc190013993)

[Choosing between black-box and white-box testing 2](#_Toc190013994)

[Summary 2](#_Toc190013995)

## Diving into the four pillars of a good unit test

A good unit test has the following four attributes:

* Protection against regressions
* Resistance to refactoring
* Fast feedback
* Maintainability

### The first pillar: Protection against regression

A *regression* is a software bug. It’s when a feature stops working as intended after some code modification, usually after you roll out new functionality.

To evaluate how well test scores are on the metric of protecting against regressions, you need to consider the following:

* The amount of code that is executed during the test
* The complexity of that code
* The code’s domain significance
* The second pillar: Resistance to refactoring

### The second pillar: Resistance to refactoring

The second attribute of a good unit test is resistance to refactoring—the degree to which a test can sustain a refactoring of the underlying application code without turning red (failing).

To evaluate how well a test scores on the metric of resisting to refactoring, you need to look at how many false positives the test generates. The fewer, the better.

The goal of unit testing is to enable sustainable project growth. The mechanism by which the tests enable sustainable growth is that they allow you to add new features and conduct regular refactoring without introducing regressions.

There are two specific benefits here:

* Tests provide an early warning when you break existing functionality.
* You become confident that your code changes won’t lead to regression. Without such confidence, you will be much more hesitant to refactor and much more likely to leave the code base to deteriorate.

False positives interfere with both benefits:

* If tests fail with no good reason, they dilute your ability and willingness to react to problems in code.
* On the other hand, when false positives are frequent, you slowly lose trust in the test suite. This lack of trust leads to fewer refactoring, because you try to reduce code changes to a minimum to avoid regressions.

### What causes false positives?

The more the test is coupled to the implementation details of the system under test (SUT), the more false alarms it generates. The only way to reduce the chance of getting a false positive is to decouple the test from those implementation details.

### Aim at the end result instead of implementation details

## The intrinsic connection between the first two attributes

### Maximizing test accuracy

### The importance of false positives and false negatives: The dynamics

## The third and fourth pillars: Fast feedback and maintainability

## In search of an ideal test

### Is it possible to create an ideal test?

### Extreme case #1: End-to-end tests

### Extreme case #2: Trivial tests

### Extreme case #3: Brittle tests

### In search of an ideal test: The results

## Exploring well-known test automation concepts

### Breaking down the Test Pyramid

### Choosing between black-box and white-box testing

## Summary