Testing in Software Development & Data Science

Fenyx IT Academy

Agenda

- ➔ What is testing?
- ➔ Why we need testing?
- ➔ Importance of testing
- ➔ Writing testable code
- ➔ Static Analysis
- ➔ Software Testing Types
- ➔ Data Science Testing Types

What is testing?

Software testing is a method to check whether the actual software product matches expected requirements and to ensure that software product is bug-free.

As your codebase expands, small errors and edge cases you don't expect can cascade into larger failures. Bugs lead to bad user experience and ultimately, business losses. One way to prevent fragile programming is to test your code before releasing it into the wild.

Why we need testing?

We're humans, and humans make mistakes. Testing is important because it helps you uncover these mistakes and verifies that your code is working. Perhaps even more importantly, testing ensures that your code continues to work in the future as you add new features, refactor the existing ones, or upgrade major dependencies of your project.

Why testing is important?

Software Testing is important because if there are any bugs or errors in the software, it can be identified early and can be solved before delivery of the software product. Properly tested software product ensures reliability, security and high performance which further results in time saving, cost effectiveness and customer satisfaction.

In May 2015, Airbus issued an alert for urgently checking its A400M aircraft when a report detected a software bug that had caused a fatal crash earlier in Spain. Prior to this alert, a test flight in Seville has caused the death of four air force crew members and two were left injured.

Source: [Theguardian.com]

For over 2 years Nissan recalled over a million cars, thanks to a software glitch in the airbag sensory detectors. Practically, the affected cars were unable to assess whether an adult was seated in the car's passenger seat and consequently would not inflate the airbags in case of a crisis.

Source: [Computerworlduk.com]

Software update for the Nest 'smart' thermostat (owned by Google) went wrong and literally left users in the cold. When the software update went wrong, it forced the device's batteries to drain out, which led to drop in the temperature. Consequently, the customers were unable to heat their homes or use any amenities.

Source: [The New York Times]

Static Analysis

The first step to improve your code quality is to start using static analysis tools. Static analysis checks your code for errors as you write it, but without running any of that code.

Linters analyze code to catch common errors such as unused code and to help avoid pitfalls, to flag style guide no-nos like using tabs instead of spaces (or vice versa, depending on your configuration).

Type checking ensures that the construct you're passing to a function matches what the function was designed to accept, preventing passing a string to a counting function that expects a number, for instance.

Writing Testable Code

To start with tests, you first need to write code that is testable. Consider an aircraft manufacturing process - before any model first takes off to show that all of its complex systems work well together, individual parts are tested to guarantee they are safe and function correctly. For example, wings are tested by bending them under extreme load; engine parts are tested for their durability; the windshield is tested against simulated bird impact.

Writing Testable Code

Software is similar. Instead of writing your entire program in one huge file with many lines of code, you write your code in multiple small modules that you can test more thoroughly than if you tested the assembled whole. In this way, writing testable code is intertwined with writing clean, modular code.

Software Testing Types

Unit Tests

Unit tests cover the smallest parts of code, like individual functions or classes.

When the object being tested has any dependencies, you'll often need to mock them out.

The great thing about unit tests is that they are quick to write and run. Therefore, as you work, you get fast feedback about whether your tests are passing.

Mocking

Sometimes, when your tested objects have external dependencies, you'll want to "mock them out."

Mocking is when you replace some dependency of your code with your own implementation.

Mocking

Imagine you're writing an app that shows the current weather in your city and you're using some external service or other dependency that provides you with the weather information. If the service tells you that it's raining, you want to show an image with a rainy cloud. You don't want to call that service in your tests, because:

- It could make the tests slow and unstable (because of the network requests involved).
- The service may return different data every time you run the test.
- Third party services can go offline when you really need to run tests.

Test Coverage

Test coverage is defined as a metric in software testing that measures the amount of testing performed by a set of test.

In simple terms, it is a technique to ensure that your tests are testing your code or how much of your code you covered by running the tests.

Integration Tests

When writing larger software systems, individual pieces of it need to interact with each other. In unit testing, if your unit depends on another one, you'll sometimes end up mocking the dependency, replacing it with a fake one.

In integration testing, real individual units are combined (same as in your app) and tested together to ensure that their cooperation works as expected. This is not to say that mocking does not happen here: you'll still need mocks (for example, to mock communication with a weather service), but you'll need them much less than in unit testing.

Regression Testing

Regression Testing is defined as a type of software testing to confirm that a recent program or code change has not adversely affected existing features. Regression Testing is nothing but a full or partial selection of already executed test cases which are re-executed to ensure existing functionalities work fine.

This testing is done to make sure that new code changes should not have side effects on the existing functionalities. It ensures that the old code still works once the latest code changes are done.

End-to-end Testing

End-to-end testing involves testing of a complete application environment in a situation that mimics real-world use, such as interacting with a database, using network communications, or interacting with other hardware, applications, or systems if appropriate. It may be performed by QA (Quality Assurance) teams.

Other types of testing

- Smoke Testing
- User Acceptance Testing
- Endurance Testing
- Stress Testing
- White Box / Black Box Testing
- Penetration Testing
- ...

Data Science Test Types

Types of Data Quality Tests

There are two broad levels of tests we can define for data quality: At the example level (e.g. “id field should not be NULL”), and at the dataset level (e.g. “mean value of feature X should be between [5.0, 8.0]”). The best configuration of tests for data quality depends on the domain and the dataset you’re working with, but data quality tests are largely analogous to unit tests in software development.

Basic quality checks

The fundamental data quality checks that can protect against data pipeline failures. Eg: Missing data checks, duplicated data checks.

Examples:

- ➔ Feature value “age” is not NULL
- ➔ “Customer ID” column value is unique

Continuous variable checks

These are checks you expect the continuous variables in your data to pass.

Examples:

- ➔ Feature value “weight” is between [50, 1000]
- ➔ Feature value “height” is always of type ‘float’
- ➔ Mean (“weight”) in the inference data is in range [lower, upper]

Categorical variable checks

These are checks you expect the categorical variables in your data to obey to ensure that all values in a column only contain specific values, which may be distributed in a certain way.

Example:

- ➔ Feature value “gender” is in set [“male”, “female”, “other”, “unknown”]
- ➔ Feature value = ‘unknown’ for “gender” in <= 5% of the inference data

What else?

- ➔ Custom data checks
- ➔ Embedding variable checks
- ➔ Distributional checks
- ➔ Quantifying label noise
- ➔ Behavioral Test for ML (Pre-train & Post-train tests)

Questions?

Thanks!

Codebase:

- ➔ https://github.com/fenyx-it-academy/TechTalks-Testing

References:

- ➔ Test Coverage in Software Testing
- ➔ What is Software Testing? Definition
- ➔ Testing · React Native
- ➔ Checking data quality
- ➔ Getting Started With Testing in Python
- ➔ Coverage.py
- ➔ Pytest
- ➔ Getting started with Great Expectations
- ➔ Explore Expectations
- ➔ How to quickly explore Expectations in a notebook