International Journal of Core Engineering & Management
Volume-7, Issue-05, 2023 ISSN No: 2348-9510

PARAMETERIZED UNIT TESTING IN JAVA AND PYTHON

Nilesh Jagnik
Los Angeles, USA
nileshjagnik@gmail.com

Abstract

Unit tests can get repetitive and bloated over time. This can make unit tests complex and hard to maintain which in turn may reduce the quality and coverage of such tests. A common reason for this is that unit tests repeat a lot of setup and assertion logic. Parameterized testing aims to solve this problem by reusing test logic over multiple test inputs. Parameterized unit testing is supported by multiple unit testing frameworks in multiple programming languages. This paper covers parameterized unit testing with JUnit5 in Java and Abseil in Python.

Keywords: unit testing practices, code reuse, unit testing frameworks

I. INTRODUCTION

Unit test is the most basic level of testing in software development. These tests are used for testing the functionality of the smallest units of code. Unit testing is generally easy to do because the test setup requires very few dependencies [1]. Due to this, it is generally considered good practice to write exhaustive unit tests covering all possible edge case behaviour for the code under test.

The practice of doing thorough unit tests results in improved code health. But sometimes writing a lot of unit tests can lead to duplication of code within the unit tests too. As a result, over time unit test code becomes harder to read, maintain and extend.

In this paper we discuss a strategy that can drastically alleviate this issue by making unit tests easier to read, maintain and extend. The strategy is to parameterize unit tests so that duplication of code within unit tests can be reduced. Parameterized testing as a concept can be used in any programming language. In this paper, we discuss ways to parameterize unit tests in Java and Python.

II. PARAMERIZED UNIT TESTING

Parameterized testing is a testing paradigm that involves running the same test with difference data inputs. In this setup, tests are written in a generic manner. These generic tests should be able to test all or some functionality of the system in various test scenarios. The test scenarios are then passed as input parameters to the test.

The scenario usually consists of input data and expected results from executing the system under test. Each test scenario creates a new invocation of the unit test. Writing tests in this way reduces the number of tests that need to be written.

International Journal of Core Engineering & Management
Volume-7, Issue-05, 2023 ISSN No: 2348-9510

III. WHY PARAMETERIZE UNIT TESTS

Parameterized testing allows running the same test multiple times with different inputs and expected outputs [2]. Naturally this has its uses.

1. Code Reuse

As the complexity of a code module increases, its unit tests need to be also be updated to test all corner and normal cases. In traditional unit testing paradigm, new test cases are created. These cases may have a lot of common functionality with previously existing tests, like setting up the test code, invoking the system under test and verifying that the correct output is produced. This means a significant amount code is duplicated every time a new unit test is added. Creating parameterized tests involves creating a single test, and providing a set of input and output pairs to the test framework. The framework then creates multiple separate instances of the test, one corresponding to each input and output pair. This reuse of code leads to fewer lines of test code which also makes it easy to maintain.

2. Test Coverage

Parameterized tests are easy to extend. Since the core test logic is already written, adding a new test case is simply a matter of appending an input and output pair to the parameter list. This reduces friction of adding new tests and helps improve test coverage.

3. Reliability

A parameterized test is, in practice, a combination of what would otherwise be multiple tests. This means its test conditions are typically a superset of all of them. In practice, this ensures every test case is thoroughly tested using an established test of test conditions. These tests end up catching bugs in an easier manner as compared to non-parameterized tests.

IV. STRUCTURE OF A UNIT TEST

The ideal coding pattern for unit tests organizes code into three blocks. Each block contains code focused on one thing. This enables code to be better organized and easier to read and maintain [3].

1. Arrange

Code in this block is responsible for setting up the test environment. The code here should simulate a real-world environment as closely as possible. This includes creating all dependencies that the test object needs. And creating the test object itself. All mocks, fakes and spy objects should be created in this block. Parameter inputs to the test object and its member functions should also be created in this block.

2. Act

The test object/method is executed in this block using the dependencies and inputs created in the Arrange block. This block is usually short.

3. Assert

This part of the unit test is responsible for performing checks on the output or side effects produced by the Act block. The Assert block is responsible for determining whether a test was successful or not.

International Journal of Core Engineering & Management
Volume-7, Issue-05, 2023 ISSN No: 2348-9510

V. PARAMETERIZED TESTING IN JAVA

1. JUnit Test Framework

JUnit is an open-source framework for Java that provides easy and automated ways for developers to write unit tests to ensure that their code works as expected and detect presence of errors in code [4]. JUnit is one of the most popular frameworks for testing Java code. JUnit5 is the most recent generation of JUnit and focusses on Java 8 and above. In addition to unit tests, JUnit can also be used for functional and integration tests.

2. Writing Unit Tests

Writing a unit test in JUnit is quite straight forward [5]. The normal practice is to create one test class corresponding to every class that should be tested. Method of the test class are annotated with the @Test annotation which tells the framework to execute these methods for unit testing. The framework runs all the @Test annotated methods and reports success or failure based on their results.

```
import org.junit.jupiter.api.Test;

class MyClassTest {
  @Test
  void simpleTest() {
    assertEquals(100, 10 * 10);
  }
}
```

Fig. 1. A simple unit test using JUnit5

3. Writing Parameterized Unit Tests

Writing parameterized tests is not much more complicated either [6]. Similar to basic unit tests, a test method should be annotated. The annotation changes to @ParameterizedTest instead of @Test to let the framework know that this test expect parameters. Parameters are then passed to this test using a source annotation. The source annotation specifies the source from where the parameters should be read. There are several ways to provide the source of arguments. These can be values specified directly, csv files or a method that generates parameters. Fig. 2 showcases the use of the method source using the @MethodSource annotation.

Note that a parameterized test can be annotated with multiple source annotations allowing it to read arguments from multiple sources

```
class MyClassTest {

  @ParameterizedTest
  @MethodSource("inputAndOutput")
  void testSquaresCorrectly(int input, int output) {
    assertEquals(output, squareFunc(input));
  }

  static Stream<Arguments> inputAndOutput() {
    return Stream.of(
      arguments(10, 100),
      arguments(3, 9)
    );
  }
}
```

Fig. 2. A parameterized unit test using JUnit5

International Journal of Core Engineering & Management
Volume-7, Issue-05, 2023 ISSN No: 2348-9510

VI. PARAMETERIZED TESTING IN PYTHON

1. Abseil Testing Modules

Parameterized testing is supported by the Abseil Python Common Libraries' testing modules [7]. The support for parameterized testing is very similar in essence to the JUnit in Java. The library allows easy unit test development.

2. Writing Parameterized Unit Tests

Unit tests in Abseil Test require that the test class be derived from the parameterized. Testcase class [8]. We can then use the @parameterized. parameters decorator which allows specifying a method as a parameterized unit test along with providing the parameters expected by the test. Similar to JUnit5, the test method should accept parameters as input and the decorator specifies where the source of parameters.

```
from absl.testing import parameterized


class MyClassTest(parameterized.TestCase):

  @parameterized.parameters((10, 100), (3 9))
  def test_squares(self, inp, output):
    self.assertEqual(output, square(inp))
```

Fig. 3. A parameterized unit test using Abseil Testing Modules

3. Parameterizing a Test Class

In addition to parameterizing a test method, we can parameterize the entire test class using Abseil as well. This allows defining some common test cases which should be used as an input to all unit test methods defined inside the test class.

```
from absl.testing import parameterized


@parameterized.parameters((10, 100), (3 9))
class MyClassTest(parameterized.TestCase):


  def test_squares(self, inp, output):
    self.assertEqual(output, square(inp))
```

Fig. 4. Parameterizing a test class using Abseil Testing Modules

4. Specifying Parameters

There are several ways to specify parameters inputs in Abseil tests. Parameters are normally specified as a tuple, but they can also be created dynamically from a single non tuple iterable. Developers can also use a list/tuple as a single argument and then parse it inside the unit test method.

The Abseil Testing framework also allows running the test method over a cartesian product of parameters. This allows running the test with an auto-generated combination of inputs.

VII. PARAMETERIZED TESTS LIMITATIONS

1. Test Becomes Too Complex

Parameterized testing is best suited to cases where the unit test logic can be kept simple. Parameterizing test cases should not come at the cost of violating other best practices. In particular, control structures and conditional statements should be avoided in unit tests [9]. Parameterizing tests should be limited to cases where such anti-patterns are not introduced. Having an excessive number of parameters in a test is sign that a test might be too complicated [10].

2. Naming

Naming test cases appropriately is important for readability of test code. Overuse of parameterized testing can lead to unclear test naming. Test names being too generic is also a way to detect overuse of parameterized testing.

VIII. CONCLUSION

Parameterized unit testing can have benefits when used correctly.

1. It can improve test coverage while reducing the complexity of unit tests at the same time.
2. Many test frameworks have support for parameterized testing making it an accessible tool to improve quality of tests.

REFERENCES

1. Abhinav, “Intro to Unit Tests (Jan 2022),” https://medium.com/interleap/intro-to-unit-tests-f2b7750c2d3c
2. Carlos Schults, “A Practical Guide to JUnit Parameterized Tests (Jan 2023),” https://www.waldo.com/blog/parameterized-test-junit
3. Andrew Knight, “Arrange-Act-Assert: A Pattern for Writing Good Tests (Jul 2020),” https://automationpanda.com/2020/07/07/arrange-act-assert-a-pattern-for-writing-good-tests
4. Stefan Bechtold, Sam Brannen, Johannes Link, Matthias Merdes, Marc Philipp, Juliette de Rancourt, Christian Stein, “JUnit 5 User Guide (Sep 2022),” https://junit.org/junit5/docs/current/user-guide
5. Shinji Kanai, “JUnit: A Complete Guide (May 2022),” https://www.headspin.io/blog/junit-a-complete-guide
6. Carlos Schults, “Writing A Parameterized Test In JUnit With Examples (Mar 2023),”

https://coderpad.io/blog/development/writing-a-parameterized-test-in-junit-with-examples
7. "Abseil Python Devguide: Testing (May 2022)", https://abseil.io/docs/python/guides/testing
8. "absl.testing.parameterized module (May 2022)," https://byoshimi2-abseil-py.readthedocs.io/en/latest/absl.testing.parameterized.html
9. Anthony Sciamanna, "Unit Test Refactoring and Avoiding Complexity (Mar 2016)," https://anthonysciamanna.com/2016/03/22/unit-test-refactoring-avoiding-complexity.html
10. Sergio Sastre, "Better unit tests with Parameterized testing (June 2021)," https://medium.com/geekculture/multiplying-the-quality-of-your-unit-tests-part-1-parameterized-tests-539428367222