Ödev

İsim-Soyisim: Abdalkarim M A Altawil

Ö.N: 1903013084

1.Soru: iyi bir Birim Testinin dört esası size göre nelerdir? Nedenlerini örnekler üzerinde de açıklayarak cevaplamanız beklenmektedir?

1- Protection against regressions (bugs)

Protection against regressions, which is basically the protection against bugs that the software or the code could have and face with time and it's one of the most important pillars of a good unit test.
We need this type of protection to sustain our projects and to avoid getting stuck in the middle of accumulating bugs in our system. The more code we write and the more functionalities we add the more exposure it has to bugs.
In order to evaluate how well a test score is, we should take in consideration the following :

- The amount of code that is executed during the test, the more code we test the higher chance it has to get more and more regressions.
- The complexity of the code.
- The code's domain significance.

Also we should avoid doing unit test on the small parts of code that doesn't need it, because its has a lower chances of having bugs. We shouldn't forget to test the Libraries and Framework that we used in our code because it influences the working of the software and our code.

2- Resistance to refactoring

It's another pillar of a good unit test, it's the degree to which a test unit can sustain refactoring and modifications of the initial code without changing it behavior, without failing.
An example of that is, saying the we have a code works without a problem anywhere, so we decide to make some modifications and refactoring to have a better looking code, but at the end it raise some errors in the unit test, even though the functionalities are working perfectly. This problem or situation is called "False positive" which indicate that the test fails, whatever the functionalities were working just fine or not, it usually take place when we do refactoring of our code, while keeping the same behavior.

Ödev

İsim-Soyisim: Abdalkarim M A Altawil

Ö.N: 1903013084

3- Fast feedback

It’s an essential pole of a good unit test. The feedback loop can be substantially shortened to the point that tests start alerting you to bugs as soon as you break the code, nearly eliminating the cost of bug fixes. An example of this situation is : imagine having a code with many lines, and it feedback was slow, it could cause a waste of time and moving in the wrong direction.

4- Maintainability

When we talk about this, we have to consider the understandability of the code, we definitely should write test code that is simple and readable, so it will be easier to make change when its needed. In addition we should take in consideration the difficulty of running a test, if the test passes with out-of-process dependencies, you will have to spend time maintaining those dependencies. For example, you may need to restart the database server or fix a network connectivity problem.

2.Soru: Martin Fowler ismi yazılım testi dünyasında öncü isimlerdendir. «Martin Fowler, Birim Testi (Unit Testing) ve Temiz Kod (Clean Code) » üçlemesini nasıl değerlendirirsiniz?

first of all I have to say that clean code and unit test are completely connected to each other and if we want to write a good unit test we should write a clean code that we can understand and being easy to read like what we are saying There is no secret to writing tests …, There are only secrets to writing testable code!
So, based on all of that I can say Testability more important as you get more focused.

Second thing if I will talk about clean code and unit test I have to talk about Test Driven-Development (TDD).

There are three laws of TDD:

1.Write a failing test: You should write a failing unit test before writing production code

Ödev

İsim-Soyisim: Abdalkarim M A Altawil

Ö.N: 1903013084

2.Make the test pass: You should not write a more failing test you may try to write a clean test

3.Refactor as necessary: To pass the recently failing test you should not write a more of production code

Tests Enable the – ilities

Unit tests are what keep our code flexible, maintainable and reusable because when you have a test for your code you will not be afraid of making any changes of your code but if you don't have any test for your code every change is a possible bug. The higher your test coverage, the less your fear.

Clean Tests

Clean test is really very important and maybe in my opinion its more important than the unit test because without a clean code its will be so hard to make a unit test and its will be so hard to be able to read the code. So, Readability is more important in unit tests than it's in our code.

Well, how can we write a readable code?
Readable code: clarity, simplicity and density of expression. we need to write our code in a few expressions as possible as we can.

I already talked about the Unit test at the beginning but if I want to make a summary about it, I can say these things:

. Test individual classes in isolation

. Can simulate all error conditions

. Developers can run these after each file modification.

. Very fast

. Green:

. High confidence class OK
. lowest app coverage, need a lot of these
. Sure class is OK, not sure class interaction OK

Ödev

İsim-Soyisim: Abdalkarim M A Altawil

Ö.N: 1903013084

. Red:

. Easy to reproduce failures.

. No need for debugger to figure out what went wrong

Unit Tests focus on application logic by testing the smallest parts, very fast, No I/O, No need for a debugger.

Now if we are going to talk about Martin fowler and what is his opinion in a clean code and unit test we will see that his opinion is looks like what I wrote about unit test and clean code to make an example :

Martin fowler said about clean code: The software needs to be built in such a way that it's able to adapt to unexpected changes in features. One of the most important ways to do this is to write clear code, making it easy to understand what the program is supposed to do. This code should be divided into modules which allow developers to understand only the parts of the system they need to make change.

And he also talked about the refactoring and how its very important in our unit test and I already talked about how the refactoring is very important in our code but now I will explain what Martin fowler is said about it.

Martin fowler said about Refactoring:
Refactoring is a disciplined technique for restructuring an existing body of code, altering its internal structure without changing its external behavior
And he said when how much refactoring it's small, it's less likely to go wrong.

And he also talked about the testing and how is the testing it's also very important in our code and how it's very important to make a test for any code we have.

Martin fowler said about the testing:
with a range of automated tests that can give us confidence that our changes haven't introduced any bugs. This leads us to integrate testing into programming, which can act to improve our architecture.

Ödev

İsim-Soyisim: Abdalkarim M A Altawil

Ö.N: 1903013084

And that's exactly like what I already said about testing when I said that when we have a test for our code that will help us to not faire about any changing we are going to do in our production code .

So, in summary:
Martin fowler talked about clean code, Unit testing and the refactoring and how is these thing very important for our code, to make it easy and readability.

3.Soru: Unit Test Frameworks ve çalıştıkları ortamları araştırdıktan bunlarla ilgili olarak size göre nasıl bir sınıflandırma yaparsınız? Açıklayarak cevaplayın.

There is a lot of frameworks for a Unit test and honestly, choosing between frameworks is not that simple. Since each of these frameworks offers a distinct set of capabilities and employs a different coding style, we cannot say that one is superior to the others. So, the requirements of your project are basically what matter.

So, what are these frameworks:
1.Jest (Category: Unit Testing)
2.NUnit (Category: Unit Testing)
3.XUnit (Category: Unit Testing)
4.MStest (Category: Unit Testing)
5.Jasmine (Category: Unit Testing, End-to-End Testing)
6.JUnit (Category: Unit Testing, Regression Testing)
7.Cypress (Category: Unit Testing, integration Testing, End-to-End Testing)
8.Mocha (Category: Unit Testing, integration Testing, End-to-End Testing)

Now I want to compare between Jest, Mocha and Jasmine because all of them a JavaScript testing framework for unit testing:

. For apps built by using React or Next.JS, and for smaller projects as it provides a single integrated its recommended to use Jest

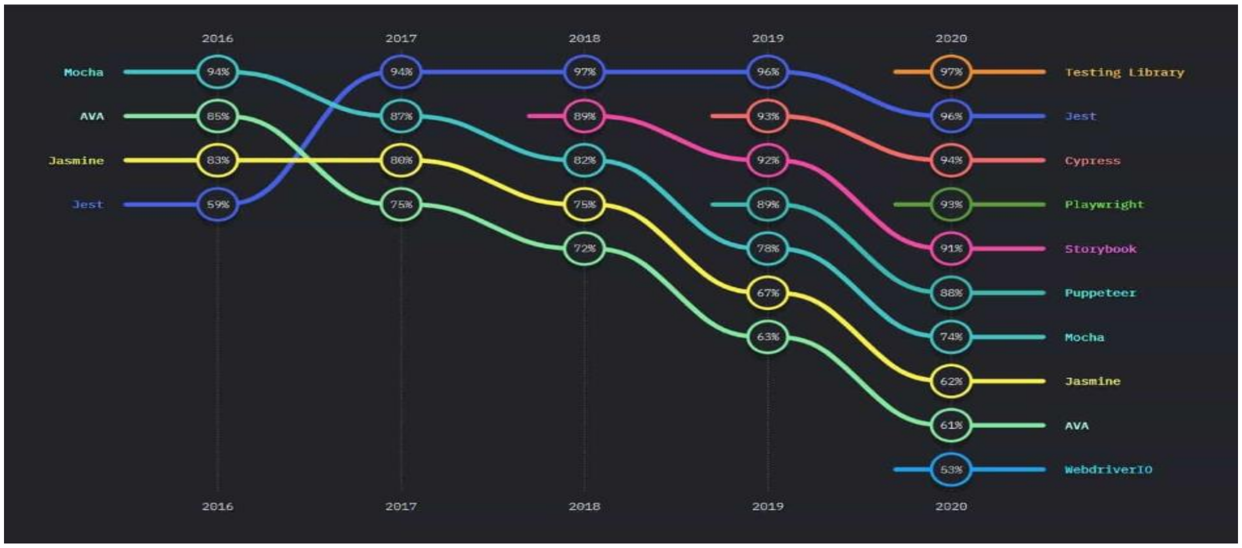
. if the project required debugging and if the project quite big and requires the integration of various external libraries the best option will be Jasmine

. Mocha might be the best option for big Node.js projects

Ödev

İsim-Soyisim: Abdalkarim M A Altawil

Ö.N: 1903013084



Now I want to compare between NUnit, XUnit and MStest framework:

1.Isolation of Tests

XUnit framework provides much better isolation of tests in comparison to NUnit and MSTest frameworks.

2. Extensibility

When we want to choose which one more extensibility than the other , This choice might depend on the needs of the project . But Because [Fact] and [Theory] attributes are used by XUnit framework so that's make it more extensible than Mstest and NUnit test.

3.Assertion mechanism

XUnit framework makes use of Assert, NUnit and MSTest using [ExpectedException], The problem of using [ExpectedException] that the errors might not be reported if they occur in the wrong part of the code

Ödev

İsim-Soyisim: Abdalkarim M A Altawil

Ö.N: 1903013084

4.Soru: Herhangi bir test framework indirerek işlevinin basit olmadığı bir kod parçasının birim testini 3A yaklaşımı ile yazın, gerekli açıklamaları yaparak ekran çıktılarını paylaşın. Ayrıca problem ifadesinin açık olarak verilmesi gerekmektedir.

First of all, I want to explain what my code is do:

What this code is doing is that allows us to create employee instances where it will set the employees firs-name and last name and pay and then we have some methods that return the employees email address which is their first name and last name at email.com, then we have a method that returns the employees full name which is just their first name and last name combined, and we also have a regular method where we can apply arrays and it will set their pay to the current pay times, the rays amount which by default us 5% up here. now the way that this code set up, if an employee's first name or last name changes then that should automatically by reflected in the email and first name.

employee.py dosyamdaki kodlar:

```python
import requests

class Employee:
    raise_amt = 1.05

    def __init__(self, first, last, pay):
        self.first=first
        self.last=last
        self.pay=pay

    @property
    def email(self): #return mail adres
        return  '{}.{}@email.com'.format(self.first, self.last)
    @property
    def fullname(self): #return full name
        return '{} {}'.format(self.first, self.last)

    def apply_raise(self): #regular method where we can apply raise
        self.pay= int(self.pay * self.raise_amt)

    def monthly_schedule(self, month):
        response = requests.get(f'http://company.com/{self.last}/{month}')
        if response.ok:
            return response.text
```

Ödev

İsim-Soyisim: Abdalkarim M A Altawil

Ö.N: 1903013084

```
        else:
            return 'Bad response!'
```

test.employee.py dosyamdaki kodlar:

```
import unittest
from unittest.mock import patch
from employee import Employee #importing our employee class from the employee
module

class TestEmployee(unittest.TestCase): #here we are creating our test case
that inherits from unit test test case

    @classmethod
    def setUpClass(cls): #this will run before any test just a once
        print('setupClass')

    @classmethod
    def tearDownClass(cls):#this will run after all of tests just a once
        print('teardownClass')

    def setUp(self): #Here by using setUp we are creating 2 employees
        print('SetUp')
        self.emp_1 = Employee('Kerim', 'Tawil', 40000) #self.emp_1:thats mean
i will set them as instance attributes
        self.emp_2 = Employee('Abdalkarim', 'Altawil', 30000)

    def tearDown(self):
        print('tearDown\n')
#here we have three different tests
    def test_email(self): #Not:our test have to start with a word
test,Otherwise compiler will run 0 test.
        print('test_email')
        self.assertEqual(self.emp_1.email, 'Kerim.Tawil@email.com') #the
second argument its what we expected
        self.assertEqual(self.emp_2.email, 'Abdalkarim.Altawil@email.com')

        self.emp_1.first='Amro' #here Im changing our employees names
        self.emp_2.first='Ali'

        self.assertEqual(self.emp_1.email, 'Amro.Tawil@email.com') #here Im
checking the emails againg because its should change
        self.assertEqual(self.emp_2.email, 'Ali.Altawil@email.com')

    def test_fullname(self):
        print('test_fullname')
        self.assertEqual(self.emp_1.fullname, 'Kerim Tawil')
        self.assertEqual(self.emp_2.fullname, 'Abdalkarim Altawil')

        self.emp_1.first = 'Amro'
```

Ödev

İsim-Soyisim: Abdalkarim M A Altawil
Ö.N: 1903013084

```
        self.emp_2.first = 'Ali'

        self.assertEqual(self.emp_1.fullname, 'Amro Tawil')
        self.assertEqual(self.emp_2.fullname, 'Ali Altawil')

    def test_apply_raise(self):
        print('test_apply_raise')
        self.emp_1.apply_raise() #here im applying raise and by default
that's 5%
        self.emp_2.apply_raise()

        self.assertEqual(self.emp_1.pay, 42000)
        self.assertEqual(self.emp_2.pay, 31500)

    def test_monthly_schedule(self):
        with patch('employee.requests.get') as mocked_get: #within patch we
pass what we want to mock
            mocked_get.return_value.ok= True
            mocked_get.return_value.text = 'Success'

            schedule = self.emp_1.monthly_schedule('May')
            mocked_get.assert_called_with('http://company.com/Tawil/May')
            self.assertEqual(schedule,'Success')

            mocked_get.return_value.ok = False

            schedule = self.emp_2.monthly_schedule('June')
            mocked_get.assert_called_with('http://company.com/Altawil/June')
            self.assertEqual(schedule, 'Bad response!')


if __name__ == '__main__': #allow us to run our test from directly within our
editor
    unittest.main()
```

2.dosyam ile ilgili genel açıklamalar:

In all my testing's I'm creating a 2 employees over and over so instead of that I just used the setUp methods which is create the employees to all my test functions.
setUp method will run its code before every single test
tearDown method will run its code after every single test
@classmethod could be good if we want to run something before any test and tearDown will be useful if we want to end that code before anything else.
Mocking is something it's really helpful in our test , for example if we have a function that goes to a website and pulls down some information, now if that website is down then our functions is going to fail which will also make our test fail and we don't want that when we are testing the code because we just want our test fail if there is a bug in our code only.

Ödev

İsim-Soyisim: Abdalkarim M A Altawil

Ö.N: 1903013084

Şimdi burada kodun çıktısını göstereceğim hem bug olduğu zamanda nasıl bir çıktı vereceğini göstereceğim hemde hatasız (bug olmadan)bir şekilde nasıl bir çıktı vereceğini göstereceğim .

İlk olarak faile verdiği zamanda :

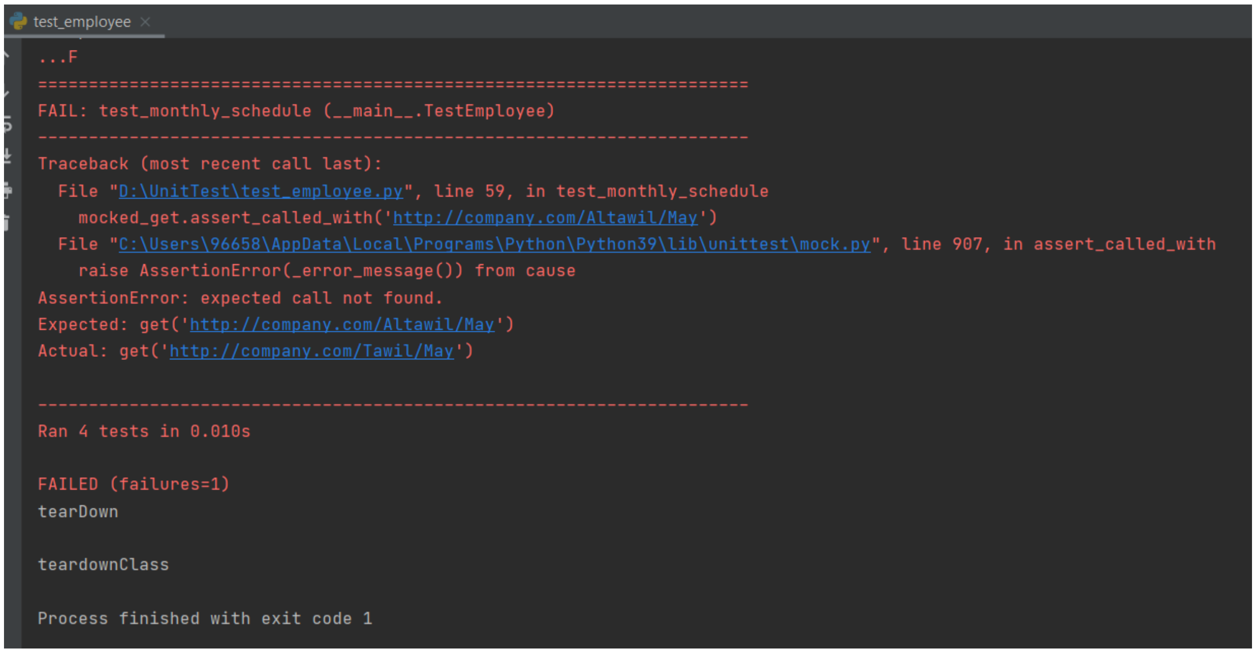
Mock kısmındaki URLdeki last name'i yanlış yazacağım bakalım nasıl bir çıktı verecek bize .

Normalde doğru kodumuz bu şekilde olmalı :

```
mocked_get.assert_called_with('http://company.com/Tawil/May')
```

ben buradaki Tawil, Altawil olarak yazacağım yani şöyle:

```
mocked_get.assert_called_with('http://company.com/Altawil/May')
```



```
...F
======================================================================
FAIL: test_monthly_schedule (__main__.TestEmployee)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "D:\UnitTest\test_employee.py", line 59, in test_monthly_schedule
    mocked_get.assert_called_with('http://company.com/Altawil/May')
  File "C:\Users\96658\AppData\Local\Programs\Python\Python39\lib\unittest\mock.py", line 907, in assert_called_with
    raise AssertionError(_error_message()) from cause
AssertionError: expected call not found.
Expected: get('http://company.com/Altawil/May')
Actual: get('http://company.com/Tawil/May')

----------------------------------------------------------------------
Ran 4 tests in 0.010s

FAILED (failures=1)
tearDown

teardownClass

Process finished with exit code 1
```

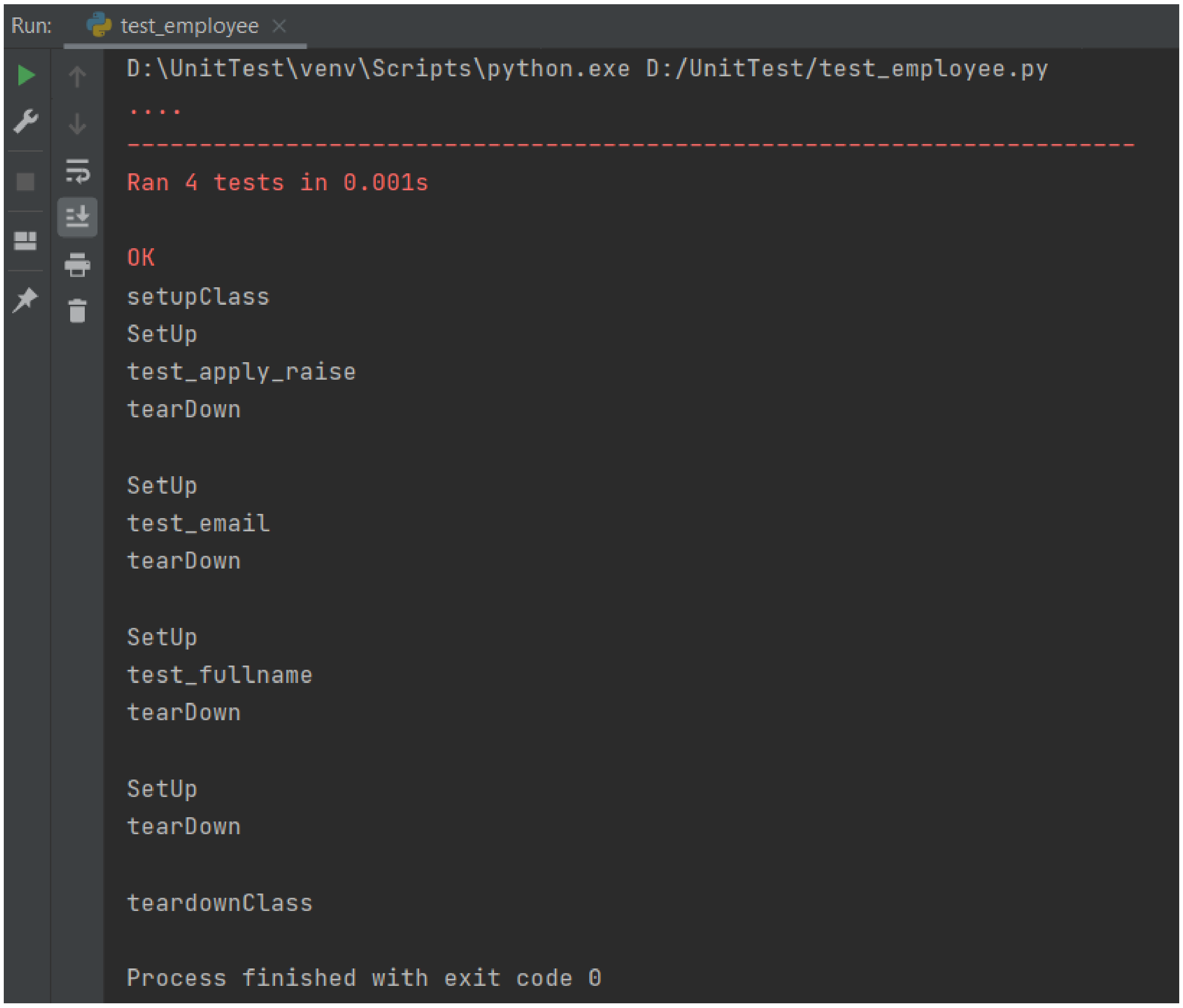
Burada gördüğümüz gibi Expected ve Actual ne olduğunu görebiliyoruz .Bu Failed'i düzelterek

Nasıl bir çıktı vereceğine bakalım.

Ödev

İsim-Soyisim: Abdalkarim M A Altawil

Ö.N: 1903013084



```
D:\UnitTest\venv\Scripts\python.exe D:/UnitTest/test_employee.py
....
----------------------------------------------------------------------
Ran 4 tests in 0.001s

OK
setupClass
SetUp
test_apply_raise
tearDown

SetUp
test_email
tearDown

SetUp
test_fullname
tearDown

SetUp
tearDown

teardownClass

Process finished with exit code 0
```

Gördüğümüz gibi testimiz başarlı bir şekilde çalıştı.

Teşekkürler Sayın **Dr. Öğr. Üyesi: Zeynep ALTAN**

Ödev

İsim-Soyisim: Abdalkarim M A Altawil
Ö.N: 1903013084

Kaynakça:

https://notesbylex.com/4-pillars-of-good-unit-tests.html

https://sd.blackball.lv/library/unit_testing_(2020).pdf

https://brightsec.com/blog/unit-testing-frameworks/

https://brightsec.com/blog/unit-testing-best-practices/#deterministic-tests

https://medium.com/ltunes/erken-uyar%C4%B1-sistemi-unit-test-9e2ab05cc760

https://www.lambdatest.com/blog/jest-vs-mocha-vs-jasmine/

https://insights.stackoverflow.com/survey/2021#overview

https://knapsackpro.com/testing_frameworks/difference_between/nunit/vs/cypress-io

https://knapsackpro.com/testing_frameworks/difference_between/nunit/vs/junit

https://www.lambdatest.com/blog/nunit-vs-xunit-vs-mstest/

https://www.lambdatest.com/blog/cypress-vs-selenium-comparison/

https://martinfowler.com/

https://martinfowler.com/aboutMe.html

https://refactoring.com/