Unit Test Workshops

Krystian Kolad

2018

Agenda

1. Teoria
2. Praktyka
3. Zakończenie

Czym są Unit Testy?

- Fragment kodu testujący daną jednostkę (najczęściej pojedynczą metodę)
- Banalne założenie - dla zadanych danych wejściowych weryfikuje, czy rezultat wykonania metody jest poprawny
- Nie wymaga ingerencji programisty w trakcie trwania testu

Plusy pisania Unit Testów?

- Weryfikacja, czy stworzona przez nas logika działa prawidłowo
- Poprawa jakości kodu - zły kod jest trudniej testowalny
- Brak konieczności uruchomienia aplikacji aby przetestować naszą logikę
- Regresja - dobrze napisane testy wykryją błędy powstałe na skutek zmiany kodu metody testowanej
- Dokumentacja - dobrze napisane testy powiedzą osobie, która widzi pierwszy raz nasz kod, co powinien robić

Czym jest Mock,Stub?

- Dostarczenie własnej implementacji metod lub klas, z której korzysta testowana metoda
- Rozwiązanie problemu zależności pomiędzy klasami
- Pozwala nam uniezależnić testy jednej metody od działania metod klasy mockowanej
- Stub - potrafi zwracać wartości, nie wyrzuca błędów przy niezdefiniowanych stanach
- Mock - potrafi to, co Stub, dodatkowo potrafi weryfikować zachowanie obiektów

Mały przegląd frameworków

Frameworki testów w środowisku .Net:

- NUnit
- xUnit
- MSTest

Frameworki mockowania w środowisku .Net:

- Moq
- FakeItEasy
- Rhino Mocks

Piszemy pierwszy test

- Instalacja potrzebnych paczek: NUnit oraz NUnit3TestAdapter
- Podział testu na 3 części: arrange, act, assert.
- TestFixture, Test, TestCase, SetUp, TearDown

```
[TestFixture]
0 references
public class TestClass
{

}
```

```
[SetUp]
0 references
public void SetUp()
{

}
```

```
[Test]
0 references | run test | debug test
public void Test_Method()
{

}
```

```
[TearDown]
0 references
public void TearDown()
{

}
```

DEMO

Czas na demo

Mockujemy zależności w testach

- Instalacja paczki Moq
- Mockowanie tylko interfejsów
- Metody Setup, Returns, Throws, Verify

```
Mock<IInterface> _interfaceMock;

_interfaceMock = new Mock<IInterface>();

_interfaceMock.Setup(x=>x.GetName()).Returns("name");

_interfaceMock.Setup(x=>x.SetName()).Throws(new ArgumentNullException());

_interfaceMock.Verify(x=>x.SetName());
```

DEMO

Czas na demo

Dobre praktyki testowania

- Nazwa testu składająca się z trzech rzeczy - nazwy testowanej metody, parametrów, jakie jej przekazujemy oraz oczekiwanego wyniku
- Nazwa testu zapisana w Snake Case(notacja wężowa?) np. Add_With_Two_And_Five_Returns_Eight
- Oddzielenie testów jednostkowych od testów integracyjnych
- Unikanie logiki w testach
- Każda zmiana w kodzie(poza nazwami zmiennych) powinna być wykryta przez testy

O TDD słów kilka

- Metodyka zwinna
- Tworzenie testu do metody przed jej konkretną implementacją
- Zapewnia szybie wychwytywanie błędów podczas implementacji
- Wymaga więcej czasu niż standardowe podejście
- Wymagające, nieprawidłowe testy prowadzą do większej ilości błędów
- Pozwala pisać lepszy kod

Przydatne linki

- https://github.com/KrystianKolad/UnitTestTutorial
- https://devstyle.pl/ - blog pełen materiałów o Unit Testach

To już jest koniec

Dziękuję za uwagę

Krystian Kolad, Wrocław, 2018 r.