Politechnika Świętokrzyska w Kielcach Wydział Elektrotechniki, Automatyki i Informatyki	
Technologie obiektowe - projekt	
Temat: Porównanie rozwiązań związanych z testowaniem	Przemysław Pyk 1ID21B Krzysztof Siczek 1ID21B
Numer projektu: 41	

1. Wstęp

Celem projektu było stworzenie porównania rozwiązań związanych z testowaniem. Projekt daje mozliwość testowania wzorca projektowego Buildera w języku Java oraz C#. Tematem Buildera są raporty w firmie zajmującej się sprzedażą aut. Składa się on z 13 klas takich jak na przykład Salesman, Vehicle, Cars, Employee, Report, ReportPattern. Do testów wykorzystano AssertJ EasyMock, AssertJ Mockito w Javie. W C# NUnit JustMock, NUnit Moq oraz Xunit JustMock, Xunit Moq. Wstępnie testy były pisanie w JustMocku oraz Typemocku, lecz zrezygnowaliśmy z Typemocka, ponieważ wymagał on płatności, a darmowa wersja nie działała w odpowiedni sposób.

2. Użyte technologie

NUnit vs xUnit.net

Nunit został wprowadzony w roku 2002 i nadal jest szeroko stosowany, bardzo dobrze udokumentowany i posiada duże community użytkowników, gdzie xUnit.net jest bardziej nowoczesny, bardziej rozszczerzalny i zyskuje na popularności w .NET Core.

Główna różnica polega na sposobie w jaki xUnit.net testuje metody.

W NUnit mamy klasę testową i zbiór metod testowych w niej. NUnit tworzy nową instancję klasy testowej, a następnie uruchamia wszystkie metody testowe w tym samym momencie.

Gdzie xUnit.net tworzy nową instancję klasy testowej dla każdej z metod testowych. W związku z tym nie można używać pól ani właściwości do udostępniania danych między metodami testowymi, co jest złą praktyką, ponieważ metody testowe byłyby od siebie zależne, co jest niedopuszczalne w TDD (Test Driven Development). Dlatego używając Xunit.net trzeba być pewnym, że wszystkie metody są kompletnie odizolowane. Jednakże, jezeli chcemy udostępnić jakieś dane miedzy naszymi metodami testowymi to xUnit nam na to zezwoli.

JUnit – W JUnit, test jednostkowy, który pozwala sprawdzic czy dana funkcjonalność działa wzamierzony sposób. Używanie JUnitów znacznie przyśpiesza pracę nad projektem. JUnity powinno się dzielić na wybrane testy metody lub funkcjonalności, idąc od najprostszych do bardziej zaawansowanych. JUnit jest przykładem architektury xUnity dla frameworków testów jednostkowych.

Junit vs xUnit.net

Porównanie:

	Junit	xUnit.net
Język programowania	Java	.NET
Strona klienta	JUnit umożliwia testowanie front-endowych komponentów takich jak indywidualne klasy oraz funkcje, które wchodzą w skład front-endu.	Możliwość niezależnego testowania równych komponetnów frontendowych, ponieważ jest to framework do testów jednostowych.
Strona serwera	Możliwość testów klas oraz funkcji, które składają się na back-end takie jak np. połączenia bazy danych.	Możliwość niezależnych testowów różnych komponentów back-endu, ponieważ jest to framework do testów jednostkowych
Stany danych (Fixtures)	JUnit zawiera metodę setUp(), która jest uruchamiana przed każdym wywołaniem testu, oraz metodę tearDown(), która jest uruchamiana po każdej metodzie testowej.	Posiada klasy, które są konfirurowane po przetestowaniu klasy.
Grupowanie stanów danych (Group fixtures)	Możliwość używania wbudowanych funkcji setUp() i tearDown() jako grupowania określonych stanów danych.	xUnit.net zawiera narzędzia do zbierania danych, które pozwalają na współdzielenie kontekstu między wieloma testami.

AssertJ powstał jako opensource'owa biblioteka, dzięki której możemy pisać w uproszczony sposób. AssertJ znaczenie upraszcza weryfikację wyjątków nawet w porównaniu z JUnit 5. AssertJ zdecydowanie wygrywa czytelniejszym API, które ułatwia zapoznawanie się z napisanymi testami. Obie te biblioteki czyli AssertJ i JUnit 5 możemy łaczyć, aby użyskać ciekawy rezultat. Metoda statyczna assertAll pochodzi z JUnit, natomiast dobrze znany assertThat z AssertJ. Jak widać można wyciągnąć z każdej biblioteki co najlepsze i połączyć to w czytelne testy jednostkowe.

Java

Mockito to biblioteka udostępniająca API do tworzenia mokowalnych obiektów w Javie. Obiekt mokowany to atrapa implementacji obiektu.
Po utworzeniu takiego obiektu można zdefiniować atrybuty obiektu bez powoływania jego instancji wraz z uzupełnianiem wszystkich właściwości.

EasyMock to framework podobny do Mockito. Pozwala na utworzenia własnych mockowych obiektów, w których można zasymulować dany obiekt,
ustalić sposób zachowania oraz sprawdzić czy funkcjonalności działają jak przewidujemy.

Porównanie

Mockito	EasyMock
działa na licencji Mockito	działa na licencji Apache
pozwala na tworzenie mocków oraz szpiegów(szpiedzy działają na realnych obiektach, jeśli nie zrobi się mockowej wersji metody wywoływana jest prawdziwa)	pozwala na stworzenie mocków
wywoływanie mocków Mockito.when(mock.method(args)).thenReturn(value)	wywoływanie mocków EasyMock.expect(mock.method(args)).andReturn(Value)
weryfikowanie mocków Mockito.verify(mock).method(args)	weryfikowanie mocków EasyMock.verify(mock)
przechwytywanie wyjątków .thenThrow(ExceptionClass.class)	przechwytywanie wyjątków .andThrow(new ExceptionClass())
możliwość konfiguracji za pomocą adnotacji	możliwość konfiguracji za pomocją adnotacji
brak potrzeby wywoływania powtórki tworzenia mocku	konieczność wywoływania @replay do ponownego wykorzystania mocku
możliwość weryfikacji nieoczekiwanych inwokacji, niepotrzebnych inwokacji oraz weryfikacji kolejności	możliwość weryfikacji nieoczekiwanych inwokacji, niepotrzebnych inwokacji oraz weryfikacji kolejności
lepsza czytelność kodu verify(), when()	gorsza czytelność kodu expect(mock.foo()), mock.foo()
weryfikacja jest wyraźna, wskazanie miejsca błędu	weryfikacja nie jest wyraźna, brak miejsca błędu
weryfikacja w kolejności jest elastyczna Nie wymaga weryfikacji każdej pojedynczej interakcji	weryfikacja w kolejności nie jest elastyczna

JustMock jest open source'owym rozwiązaniem pozwalającym na łatwe wykonywanie testów jednostkowych, najlepiej dla projektów w metodologii SOLID.
Pozwala na stworzenie obiektów mokowanych, w których uzupłniamy podstawowe istotne wartości do celów testowych.

TypeMock to płatne rozwiązanie, pozwalające na tworzenie obiektów mokowanych, działające wspólnie z frameworkiem .NET.
Pozwala na podłączenie udających zależności w jednym podejścią oraz ostrzega o wewnętrznych zależnościach w testach.

Porównanie

JustMock	TypeMock
działa z frameworkiem .NET	działa z frameworkiem .NET
nie pozwala na mockowanie obiektów na produkcji	pozwala na mockowanie obiektów, do których nie można dotrzeć z poziomu testu np. zainicjowane na produkcji

3. Przykładowe testy

Fragment klasy VehicleList:

```
13      public void initList(){
14          Car car = new Car();
15          Car car1 = new Car();
16          Car car2 = new Car();
17          Car car3 = new Car();
18          Car car4 = new Car();
19          Car car5 = new Car();
20          Car car6 = new Car();
21          Car car7 = new Car();
22          Car car8 = new Car();
23          Car car9 = new Car();
24
25
26          car.setCar("YV1MK76F2A2182012", "WR065CJ", "BMW", "sedan", "ADAYUB1997");
27          car.setVehicleOverview(true);
28          vehicleList.add(car);
29
30          car.setCar("W0L0AHL4878062512", "TI069AJ", "OPEL", "kombi", "AGASAB5423");
31          car.setVehicleOverview(true);
32          vehicleList.add(car1);
33
34          car.setCar("WAUZZZ4L47D017597", "WRAWNSRA", "AUDI", "sedan", "TGGSTF3123");
35          car.setVehicleOverview(false);
36          vehicleList.add(car2);
```

```
67
68      public List<Vehicle> getVehicleList() { return vehicleList; }
71
72      public Vehicle getCar(String identificationNumber){
73          for (int i = 0; i < vehicleList.size(); i++){
74              if(vehicleList.get(i).getIdentificationNumber().equals(identificationNumber)){
75                  return vehicleList.get(i);
76              }
77          }
78          return null;
79      }
80
81      public List<Vehicle> getListPositiveOverview(){
82          List<Vehicle> overViewPositive = new ArrayList<>();
83          for (int i = 0; i < vehicleList.size(); i++){
84              if(vehicleList.get(i).getVehicleOverView()==true){
85                  overViewPositive.add(vehicleList.get(i));
86              }
87          }
88          return overViewPositive;
89      }
```

Zapełnienie list, aby wydłużyć czasy testów, a w szczególności zlikwidować czasy poniżej 1ms.

```
for(int i = 0; i < 1000000; i++){
    Salesman newSalesman = new Salesman();
    newSalesman.setEmployee( name: "Jan", surname: "Jarzabek", pesel: "9703234212", phone_numer: "523123123", address: "Swietokrzyska 5", position: "salesman");
    newSalesman.setCarDealer(carDealerList.getCarDealerList().get(2));
    addSalesmanToList(newSalesman);
}
```

Przykładowe testy VehicleList w Javie:

AssertJ EasyMock:

```java
@SpringBootTest
public class VehicleListAssertJEasyMockTest {

    @Test
    public void getVehicleList(){

        //given
        VehicleList vehicleList = createNiceMock(VehicleList.class);

        //when
        expect(vehicleList.getVehicleList()).andStubReturn(setVehicle());
        replay(vehicleList);

        //then
        assertThat(vehicleList.getVehicleList().size()).isEqualTo(2000010);

    }

    private List<Vehicle> setVehicle(){

        VehicleList vehicleList = new VehicleList();
        vehicleList.initList();

        return vehicleList.getVehicleList();

    }

    @Test
    public void quantityOfPositiveOverviews(){

        //given
        VehicleList vehicleList = createNiceMock(VehicleList.class);

        //when
        expect(vehicleList.getListPositiveOverview()).andStubReturn(setPositiveOverview());
        replay(vehicleList);

        //then
        assertThat(vehicleList.getListPositiveOverview().size()).isEqualTo(1000001);
    }
```

AssertJ Mockito:

```java
@SpringBootTest
public class VehicleListAssertJMockitoTest {

    @Test
    public void getVehicleList(){

        //given
        VehicleList vehicleList = mock(VehicleList.class);

        //when
        when(vehicleList.getVehicleList()).thenReturn(setVehicle());

        //then
        assertThat(vehicleList.getVehicleList().size()).isEqualTo(2000010);

    }

    private List<Vehicle> setVehicle(){

        VehicleList vehicleList = new VehicleList();
        vehicleList.initList();

        return vehicleList.getVehicleList();

    }

    @Test
    public void quantityOfPositiveOverviews(){

        //given
        VehicleList vehicleList = mock(VehicleList.class);

        //when
        when(vehicleList.getListPositiveOverview()).thenReturn(setPositiveOverview());

        //then
        assertThat(vehicleList.getListPositiveOverview().size()).isEqualTo(1000001);
    }

    private List<Vehicle> setPositiveOverview(){
```

EasyMock:

```java
@SpringBootTest
public class VehicleListEasyMockTest {

    @Test
    public void getVehicleList(){

        //given
        VehicleList vehicleList = createNiceMock(VehicleList.class);

        //when
        expect(vehicleList.getVehicleList()).andStubReturn(setVehicle());
        replay(vehicleList);

        //then
        Assertions.assertEquals(vehicleList.getVehicleList().size(), 2000010);

    }

    private List<Vehicle> setVehicle(){

        VehicleList vehicleList = new VehicleList();
        vehicleList.initList();

        return vehicleList.getVehicleList();

    }

    @Test
    public void quantityOfPositiveOverviews(){

        //given
        VehicleList vehicleList = createNiceMock(VehicleList.class);

        //when
        expect(vehicleList.getListPositiveOverview()).andStubReturn(setPositiveOverview());
        replay(vehicleList);

        //then
        Assertions.assertEquals(vehicleList.getListPositiveOverview().size(), 1000001);
    }

```

EasyMock Mockito:

```
12  @SpringBootTest
13  public class VehicleListTest {
14
15      @Test
16      public void getVehicleList(){
17
18          //given
19          VehicleList vehicleList = mock(VehicleList.class);
20
21          //when
22          when(vehicleList.getVehicleList()).thenReturn(setVehicle());
23
24          //then
25          Assertions.assertEquals(vehicleList.getVehicleList().size(), 2000010);
26
27      }
28
29      private List<Vehicle> setVehicle(){
30
31          VehicleList vehicleList = new VehicleList();
32          vehicleList.initList();
33
34          return vehicleList.getVehicleList();
35
36      }
37
38      @Test
39      public void quantityOfPositiveOverviews(){
40
41          //given
42          VehicleList vehicleList = mock(VehicleList.class);
43
44          //when
45          when(vehicleList.getListPositiveOverview()).thenReturn(setPositiveOverview());
46
47          //then
48          Assertions.assertEquals(vehicleList.getListPositiveOverview().size(), 1000001);
49      }
50
51      private List<Vehicle> setPositiveOverview(){
```

4. Wydajność testów

a. Testy w c#

Lp.	Test name	Test 1	Test 2	Test 3	Test 4	Test 5	Min	Max	**Avg**
1	Mock_compare	66000	59864	59434	59952	60293	59434	66000	61108.6
2	mock_compare.Tests.NUnit.Builder	2070	2045	2044	2050	2069	2044	2070	2055.6
3	CarDealerListNUnitJustMockTest	247	235	236	230	240	230	247	237.6
4	CarDealerListNUnitMoqTest	180	164	173	175	185	164	185	175.4
5	CarListNUnitJustMockTest	61	57	58	57	62	57	62	59
6	CarListNUnitMoqTest	299	285	286	285	289	285	299	288.8
7	ReportListNUnitJustMockTest	57	55	52	53	57	52	57	54.8
8	ReportListNUnitTypeMockTest	110	112	105	102	111	102	112	108
9	SalesmanListNUnitJustMockTest	262	246	230	243	260	230	262	248.2
10	SalesmanListNUnitTypeMockTest	263	275	254	273	260	254	275	265
11	VehicleListNUnitJustMockTest	8900	8800	8800	8800	8900	8800	8900	8840
12	VehicleListNUnitMoqTest	1030	999	1030	1020	1020	999	1030	1019.8
13	CsvImportNUnitTest	4	4	4	5	4	4	5	4.2
14	mock_compare.Tests.XUnit.Builder	4250	4220	4210	4210	4250	4210	4250	4228
15	CarDealerListXUnitJustMockTest	52	50	51	50	53	50	53	51.2
16	CarDealerListXUnitMoqTest	874	873	863	859	873	859	874	868.4
17	CarListXUnitJustMockTest	63	58	61	59	64	58	64	61
18	CarListXUnitMoqTest	731	728	720	723	730	720	731	726.4
19	ReportListXUnitJustMockTest	64	59	61	62	65	59	65	62.2
20	ReportListXUnitMoqTest	898	893	890	891	895	890	898	893.4
21	SalesmanListXUnitJustMockTest	12000	1200	1200	1200	1200	1200	12000	3360
22	SalesmanListXUnitMoqTest	19000	1800	1800	1900	1900	1800	19000	5280

23	VehicleListXUnitJustMock	18500	18600	18200	18500	18600	18200	18600	18480
24	VehicleListXUnitMoqTest	18200	18100	18100	18200	18200	18100	18200	18160
25	mock_compare.Tests.XUnit.Files	6	6	6	5	6	5	6	5.8

Czas jest podawany w ms.

NUnitJustMock – 3 vs NUnitMoq – 2

Justmock w Nunit okazał się szybszy w trzech na pięć przypadków.

XUnitJustMock – 4 vs XUnitMoq – 1

JustMock w Xunit był szybszy w czterechy przypadkach na pięć.

b. Testy w Java

Lp.	Test name	Test 1	Test 2	Test 3	Test 4	Test 5	Min	Max	**Avg**
1	builder	6166	5325	5123	5342	5643	5123	6166	5519.8
2	CarDelaerListAssertJEasyMockTest	300	253	245	252	277	245	300	265.4
3	CarDelaerListAssertJMockitoTest	622	432	435	333	589	333	622	482.2
4	CarDelaerListEasyMockTest	36	25	30	25	34	25	36	30
5	CarDealerListTest	403	364	401	366	391	364	403	385
6	CarListAssertJEasyMockTest	127	94	91	85	123	85	127	104
7	CarListAssertJMockitoTest	176	123	119	127	164	119	176	141.8
8	CarListListEasyMockTest	52	40	42	40	43	40	52	43.4
9	CarListTest	70	43	39	42	40	39	70	46.8
10	ReportListAssertJEasyMockTest	47	34	31	36	32	31	47	36
11	ReportListAssertJMockitoTest	124	85	81	87	89	81	124	93.2
12	ReportListEasyMockTest	96	74	72	75	79	72	96	79.2
13	ReportListTest	29	21	20	20	25	20	29	23
14	SalesmanListAssertJEasyMockTest	198	164	156	166	184	156	198	173.6
15	SalesmanListAssertJMockitoTest	235	189	182	180	194	180	235	196
16	SalesmanListEasyMockTest	138	123	125	124	131	123	138	128.2
17	SalesmanListTest	307	254	245	256	253	245	307	263
18	VehicleListAssertJEasyMockTest	931	812	798	813	820	798	931	834.8
19	VehicleListAssertJMockitoTest	538	491	485	489	490	485	538	498.6
20	VehicleListEasyMockTest	868	743	745	750	744	743	868	770
21	VehicleListTest	869	753	698	754	756	698	869	766

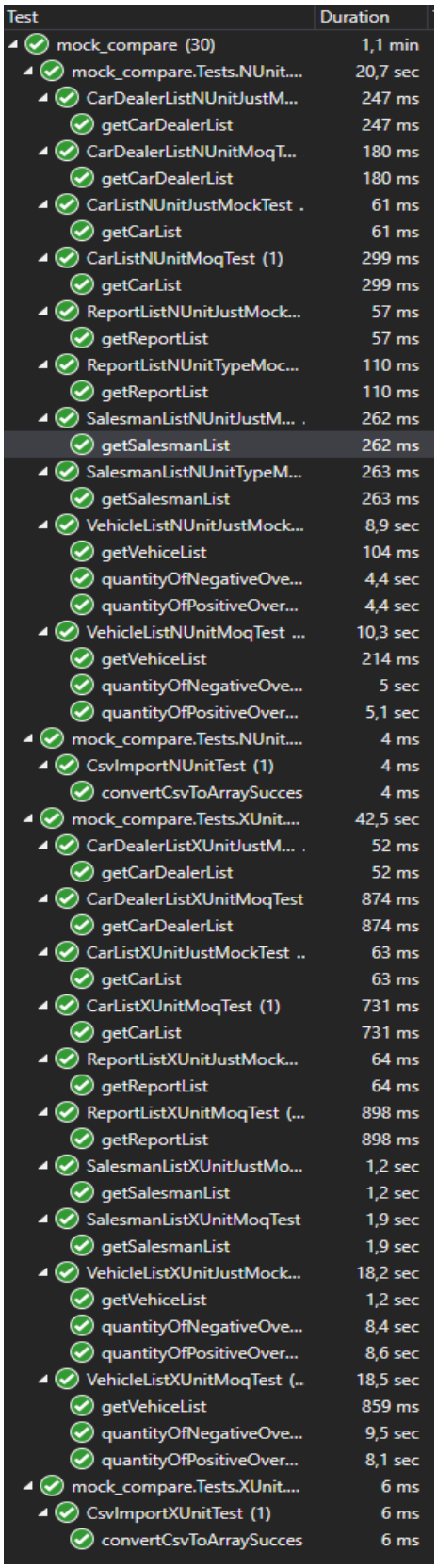
AssertJEasyMock – 0 vs AssertJMockito – 1

JUnitEasyMock – 2 vs JUnitMockito – 2

Według tabeli testów najszybszymi sposobami na testy są EasyMock oraz Mockito.
Używanie AssertJ znacznie przedłuża wykonywanie testów, lecz AssertJMickito raz wykonał się wcześniej.

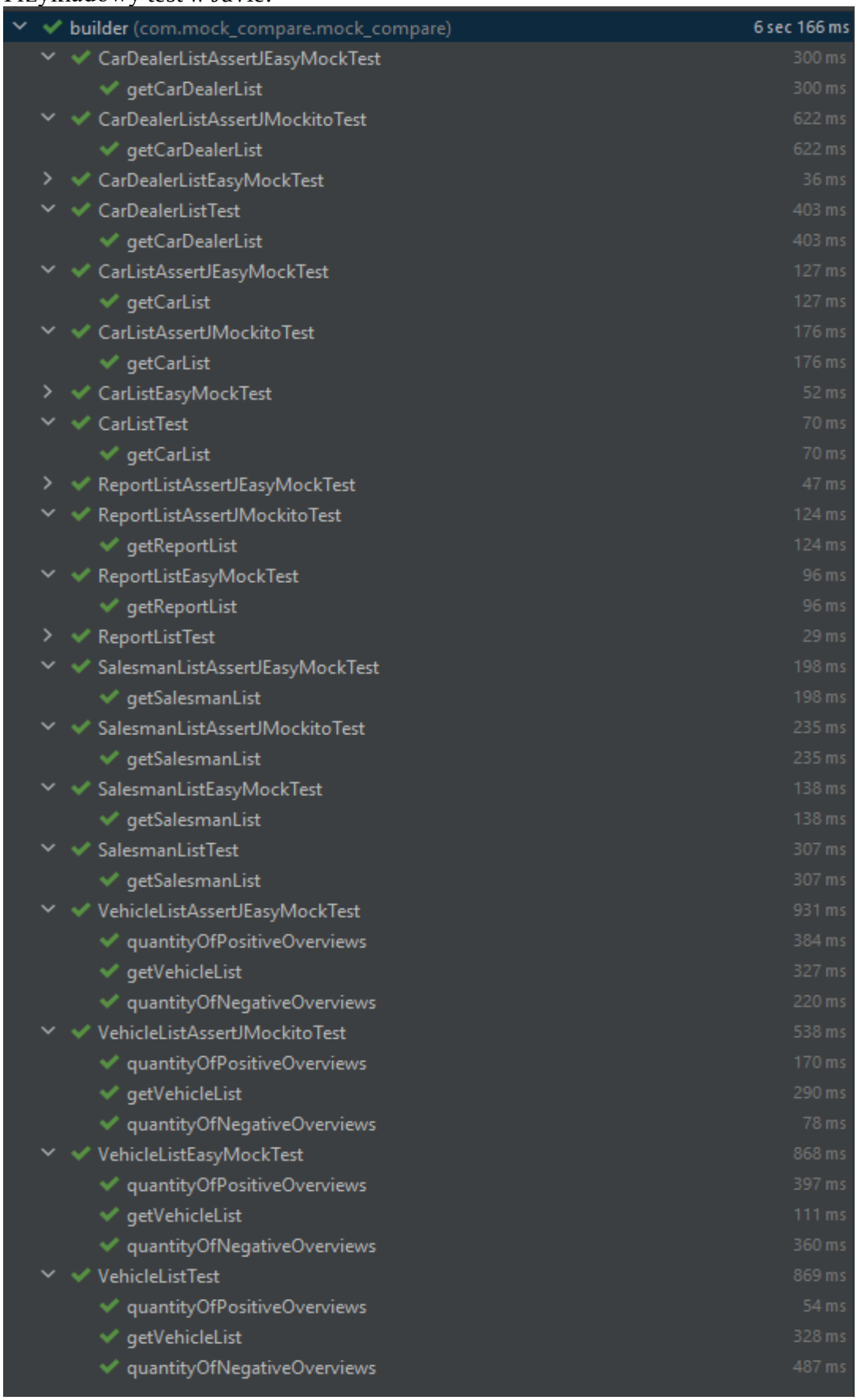
5. Screeny testów

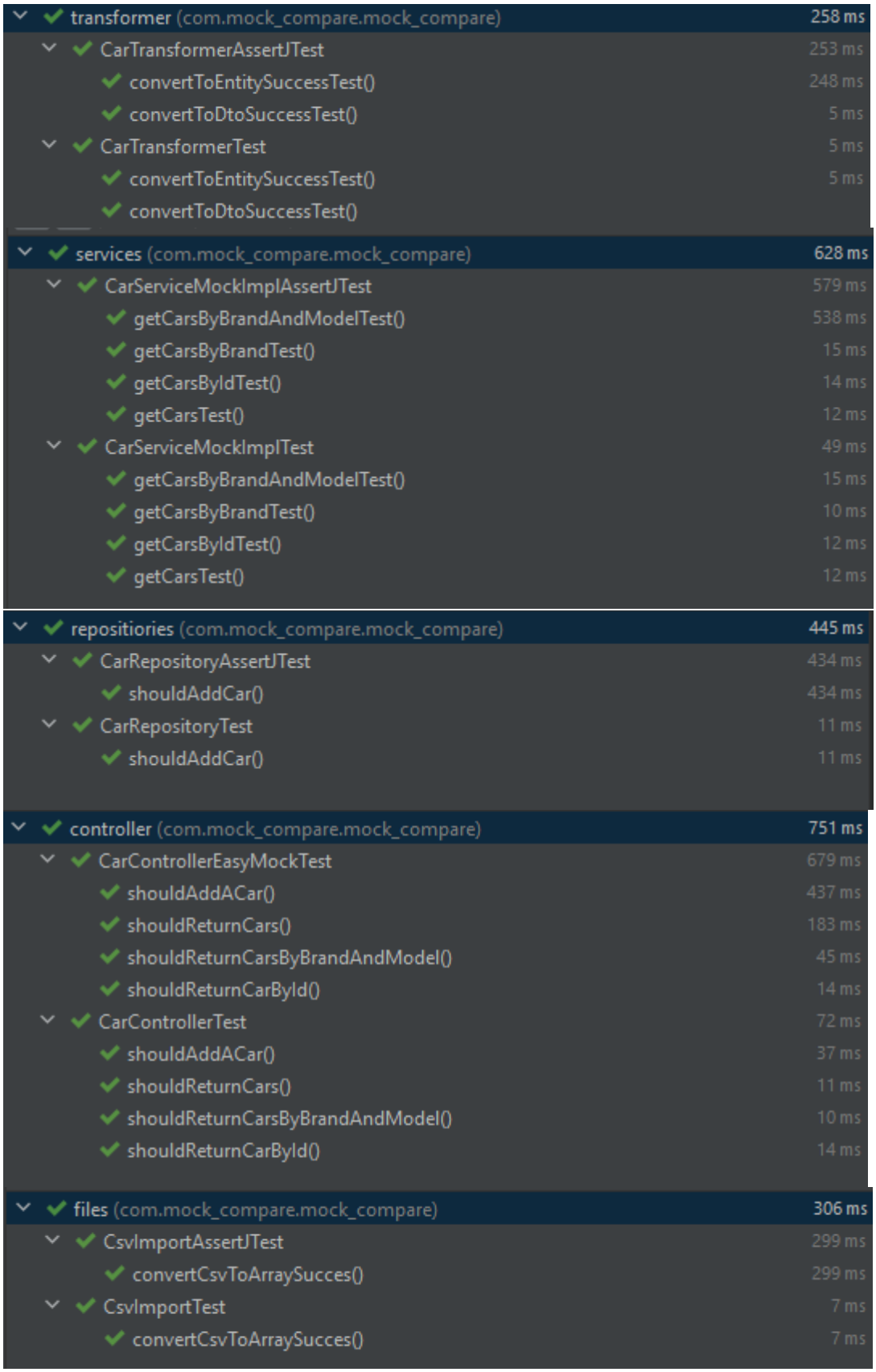
C#
Przykładowy test w C#:



Java

Przykładowy test w Javie:

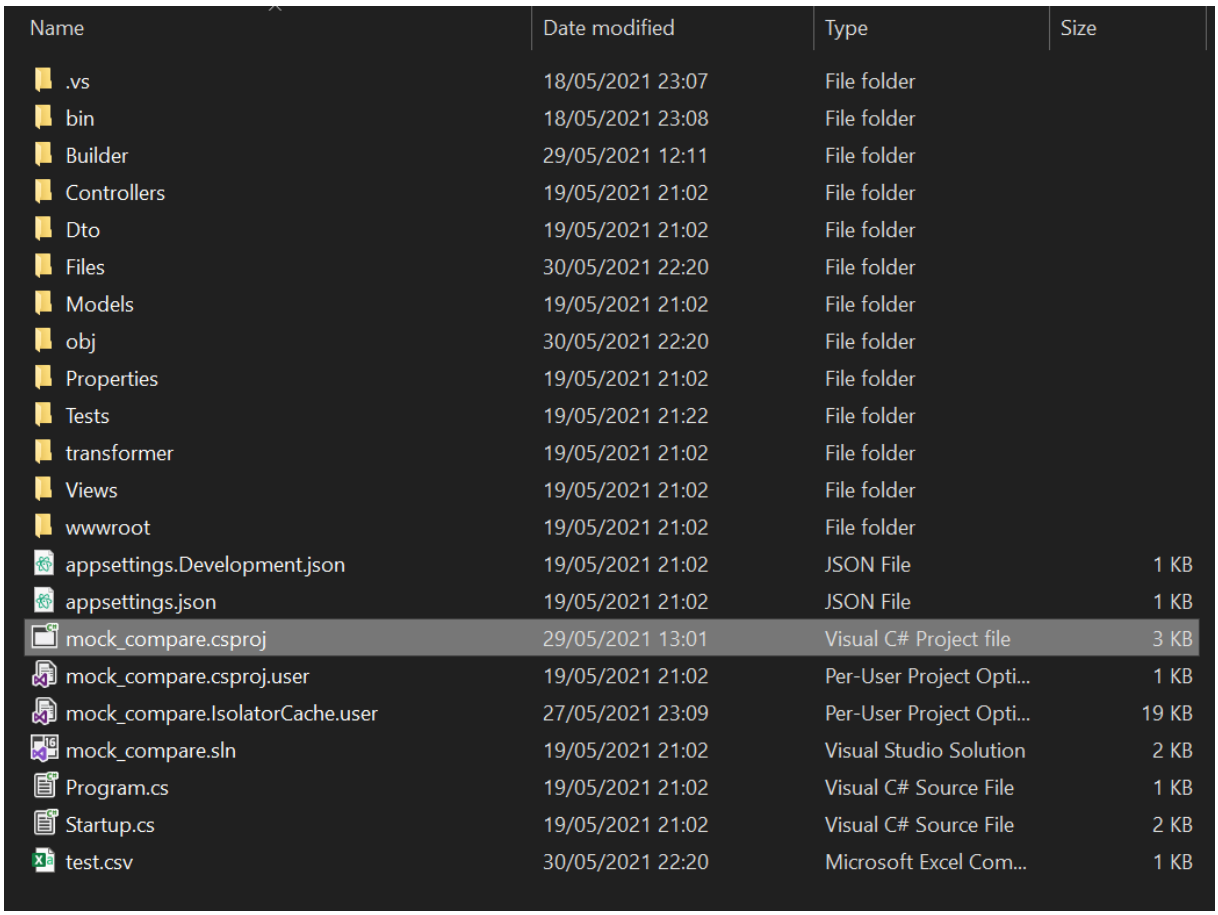



transformer (com.mock_compare.mock_compare) 258 ms
CarTransformerAssertJTest 253 ms
convertToEntitySuccessTest() 248 ms
convertToDtoSuccessTest() 5 ms
CarTransformerTest 5 ms
convertToEntitySuccessTest() 5 ms
convertToDtoSuccessTest()
services (com.mock_compare.mock_compare) 628 ms
CarServiceMockImplAssertJTest 579 ms
getCarsByBrandAndModelTest() 538 ms
getCarsByBrandTest() 15 ms
getCarsByIdTest() 14 ms
getCarsTest() 12 ms
CarServiceMockImplTest 49 ms
getCarsByBrandAndModelTest() 15 ms
getCarsByBrandTest() 10 ms
getCarsByIdTest() 12 ms
getCarsTest() 12 ms
repositiories (com.mock_compare.mock_compare) 445 ms
CarRepositoryAssertJTest 434 ms
shouldAddCar() 434 ms
CarRepositoryTest 11 ms
shouldAddCar() 11 ms
controller (com.mock_compare.mock_compare) 751 ms
CarControllerEasyMockTest 679 ms
shouldAddACar() 437 ms
shouldReturnCars() 183 ms
shouldReturnCarsByBrandAndModel() 45 ms
shouldReturnCarById() 14 ms
CarControllerTest 72 ms
shouldAddACar() 37 ms
shouldReturnCars() 11 ms
shouldReturnCarsByBrandAndModel() 10 ms
shouldReturnCarById() 14 ms
files (com.mock_compare.mock_compare) 306 ms
CsvImportAssertJTest 299 ms
convertCsvToArraySucces() 299 ms
CsvImportTest 7 ms
convertCsvToArraySucces() 7 ms

7. Problemy z wywołaniem testu w C#

Na początku próby testowania, testy nie startowały. W celu pozbycia się tego błędu należy dodać odpowiednią linijkę kodu do pliku o rozszerzeniu .csproj.

```
<PropertyGroup>
  <TargetFramework>netcoreapp3.1</TargetFramework>
  <IsPackable>false</IsPackable>
  <GenerateProgramFile>false</GenerateProgramFile>
</PropertyGroup>
```



Name	Date modified	Type	Size
.vs	18/05/2021 23:07	File folder	
bin	18/05/2021 23:08	File folder	
Builder	29/05/2021 12:11	File folder	
Controllers	19/05/2021 21:02	File folder	
Dto	19/05/2021 21:02	File folder	
Files	30/05/2021 22:20	File folder	
Models	19/05/2021 21:02	File folder	
obj	30/05/2021 22:20	File folder	
Properties	19/05/2021 21:02	File folder	
Tests	19/05/2021 21:22	File folder	
transformer	19/05/2021 21:02	File folder	
Views	19/05/2021 21:02	File folder	
wwwroot	19/05/2021 21:02	File folder	
appsettings.Development.json	19/05/2021 21:02	JSON File	1 KB
appsettings.json	19/05/2021 21:02	JSON File	1 KB
mock_compare.csproj	29/05/2021 13:01	Visual C# Project file	3 KB
mock_compare.csproj.user	19/05/2021 21:02	Per-User Project Opti...	1 KB
mock_compare.IsolatorCache.user	27/05/2021 23:09	Per-User Project Opti...	19 KB
mock_compare.sln	19/05/2021 21:02	Visual Studio Solution	2 KB
Program.cs	19/05/2021 21:02	Visual C# Source File	1 KB
Startup.cs	19/05/2021 21:02	Visual C# Source File	2 KB
test.csv	30/05/2021 22:20	Microsoft Excel Com...	1 KB

mock_compare.csproj - Notepad

File Edit Format View Help

```
<Project Sdk="Microsoft.NET.Sdk.Web">

  <PropertyGroup>
    <TargetFramework>netcoreapp3.1</TargetFramework>
    <IsPackable>false</IsPackable>
    <GenerateProgramFile>false</GenerateProgramFile>
  </PropertyGroup>

  <ItemGroup>
    <Compile Remove="Tests\NUnit\Models\**" />
    <Compile Remove="Tests\NUnit\Repository\**" />
    <Compile Remove="Tests\XUnit\Dto\**" />
    <Compile Remove="Tests\XUnit\Models\**" />
    <Content Remove="Tests\NUnit\Models\**" />
    <Content Remove="Tests\NUnit\Repository\**" />
    <Content Remove="Tests\XUnit\Dto\**" />
    <Content Remove="Tests\XUnit\Models\**" />
    <EmbeddedResource Remove="Tests\NUnit\Models\**" />
    <EmbeddedResource Remove="Tests\NUnit\Repository\**" />
    <EmbeddedResource Remove="Tests\XUnit\Dto\**" />
    <EmbeddedResource Remove="Tests\XUnit\Models\**" />
    <None Remove="Tests\NUnit\Models\**" />
    <None Remove="Tests\NUnit\Repository\**" />
    <None Remove="Tests\XUnit\Dto\**" />
    <None Remove="Tests\XUnit\Models\**" />
  </ItemGroup>
```

8. Podsumowanie

C#:

Wyniki testów przy większej ilości obiektów kompletnie się zmieniły. Przy małej ilości obietków wygrywał Moq. Był on szybszy w czterech z pięciu porównań. Po dodaniu większej ilości obietków zdecydowanie wygrywa JustMock, zarówno w NUnit jak i XUnit.

NUnitJustMock – 3 vs NUnitMoq – 2

Justmock w Nunit okazał się szybszy w trzech na pięć przypadków.

XUnitJustMock – 4 vs XUnitMoq – 1

JustMock w Xunit był szybszy w czterechy przypadkach na pięć.

Java:

Według tabeli testów najszybszymi sposobami na testy są EasyMock oraz Mockito.
Używanie AssertJ znacznie przedłuża wykonywanie testów, lecz AssertJMickito raz wykonał się wcześniej.

AssertJEasyMock – 0 vs **AssertJMockito** – 1

JUnitEasyMock – 2 vs **JUnitMockito** - 2

Java vs c#:

Porównanie testów C#, Java w builderze

JUnitEasyMock	**4**
JUnitMockito	**2**
AssertJEasyMock	**0**
AssertJMockito	**1**
NUnitJustMock	**1**
NUnitMoq	**0**
XUnitJustMock	**1**
XUnitMoq	**0**

Według wyliczonej średniej czasu z testów określonych we wcześniejszych tabelach Java była dziesięciokrotnie szybsza od C#. JunitEasymock był najszybszy z porównywanych technologii.

Średnia C#: 5065.66 ms

Średnia Javy: 505.7 ms

Z przeprowadzonych testów wynika, że Java była dziesięć razy szybsza od C#. Co daje kompletnie odmienny wynik od poprzedniej prezentacji sprawozdania. Po dodaniu większej ilości obiektów do tabeli, wynik zmienił się z trzykrotnie szybszego C#, na dziesięciokrotnie szybszą Javę. Trzeba też wziąć pod uwagę czas jaki jest wymagany, aby dany test napisać i tutaj zdecydowanym zwycięzcą jest Java. Przysporzyła ona mniej problemów podczas pisania, a praca testera była wydajniejsza i w tym samym przedziale czasu mógł on napisać więcej testów.