Laborator 8: Mocking

In momentul in care se scrie unitate de testare, este foarte posibil ca obiectul care este testat sa aiba nevoie de serviciile unor alte obiecte, sau a unei intregi infrastructuri [2]. Putem avea de-a face cu:

- Resurse externe aplicatiei implementate: sisteme de fisiere, baze de date, retea, servicii web etc.
- Costisitor de setat sau inca nedisponibile (hardware sau software care se dezvolta, servicii care trebuie platite pentru acces)
- Se incetineste semnificativ executia unitatii de testare
- Resurse cu comportament nedeterminist
- Resurse ce folosesc (sau sunt) interfete utilizator

Scrierea claselor care sa permita testarea poate fi o operatie costisitoare; ca atare, se poate folosi mocking pentru a simula un comportament.

Exemplu: sa presupunem ca trebuie testata o clasa ce returneaza numele unei imagini; imaginea urmeaza sa fie incarcata intr-un GUI. Imaginea – specificata prin numele ei - este in functie de ora curenta: daca e intre 7 si 20 atunci e o imagine cu soare, altfel e imagine cu luna. Sa presupunem ca aceasta clasa este ImageManagement si are implementata metoda GetImageForTimeOfDay():

```
public string GetImageForTimeOfDay()
{
   int currentHour = DateTime.Now.Hour;
   if (currentHour >=7 && currentHour <= 20)
      return "sun.jpg";
   else
      return "moon.jpg";
}
```

Testarea trebuie sa functioneze pentru cele doua situatii: de zi si de noapte. Este dificil sa testeze direct, deoarece trebuie asteptat efectiv ca sa fie zis sau noapte – proprietatea DateTime.Now returneaza ceasul sistemului - si sa se ruleze doar testul de zi sau de noapte. Alternativ, se poate schimba ceasul sistemului, dar asta ar exclude automatizarea (si nu e intotdeauna fezabil, uneori e nevoie de drept de administrator pentru a face asta). In mod normal, se pot scrie testele (presupunem framework-ul de testare din Visual Studio; pentru NUnit se scrie similar):

```
[TestMethod]
public void TestDay()
{
    int currentHour = DateTime.Now.Hour;
    if (currentHour >= 7 && currentHour <= 20)
    {
          string expectedImageFilename = "sun.jpg";
          ImageManagement imageManagement = new ImageManagement();
          string resultedFilename = imageManagement.GetImageForTimeOfDay();
          Assert.AreEqual(expectedImageFilename, resultedFilename);
    }
    else
    {
       Assert.Inconclusive("Can only be tested during the day");
    }
}

[TestMethod]
public void TestNight()
{
```

```
        int currentHour = DateTime.Now.Hour;
        if (currentHour >= 7 && currentHour <= 20)
        {
            Assert.Inconclusive("Can only be tested during the night");
        }
        else
        {
            string expectedImageFilename = "moon.jpg";
            ImageManagement imageManagement = new ImageManagement();
            string resultedFilename = imageManagement.GetImageForTimeOfDay();
            Assert.AreEqual(expectedImageFilename, resultedFilename);
        }
    }
```

Problema este ca in orice moment al zilei, rularea celor doua teste va produce un rezultat inconcluziv. Pentru a facilitate testarea, se poate renunta la valoarea furnizata de DateTime.Now.Hour - aceasta va fi data de catre o implementare anume a unei interfete scrise de noi, IDateTime; in plus, se va folosi o simulare de implementare pentru aceasta interfata careia i se va spune ce valoare sa returneze; testarea se va face fata de aceasta simulare (mock).

Ca atare, vom declara interfata:

```
public interface IDateTime
{
    int GetHour();
}
```

cu implementarea pentru ora curenta:

```
class CurrentDateTime : IDateTime
{
    #region IDateTime Members

    public int GetHour()
    {
        return DateTime.Now.Hour;
    }

    #endregion
}
```

Metoda `GetImageForTimeOfDay` se rescrie ca:

```
public class ImageManagement
{
    public string GetImageForTimeOfDay(IDateTime time)
    {
        int currentHour = time.GetHour();

        if (currentHour >= 7 && currentHour <= 20)
        {
            return "sun.jpg";
        }
        else
        {
            return "moon.jpg";
        }
    }
}
```

In mod normal, pentru testare ar trebui sa mai scriem doua implementari pentru interfata `IDateTime:` una care sa returneze o ora de zi si una cu retur de ora de noapte. Putem face asta manual (dar numarul de clase creste si acestea nu sunt necesare decat pentru testare), sau putem folosi un framework de mocking.

Vom obtine 2 mock-uri (simulari de implementari) prin framework-ul Rhino Mocks. Biblioteca Rhino Mocks se poate descarca de la adresa http://hibernatingrhinos.com/oss/rhino-mocks sau se poate adauga via Nuget (se specifica la cautare rhinomocks). In ambele situatii biblioteca se adauga ca referinta la proiectul de testare.

Pasii sunt:
1. Crearea depozitului de mocking; acesta va furniza mock-urile

```
MockRepository mocks = new MockRepository();
```

2. Se obtine mock-ul:

```
IDateTime timeController = mocks.StrictMock<IDateTime>();
```

Se va crea "din mers" o clasa mock care implementeaza interfata `IDateTime` si care este instantiata ca variabila timeController.
3. Se explica mock-ului ce anume trebuie sa returneze la apelul metodei GetHour():

```
using (mocks.Record())
{
    Expect.Call(timeController.GetHour()).Return(9);//ora de zi
}
```

Acesta este punctul in care nu se mai astepta ca sa fie ora de zi sau de noapte, ci se simuleaza conditia respectiva.
4. In mod playback, se foloseste ceea ce s-a explicat la punctual anterior mock-ului si se face testarea efectiva:

```
using (mocks.Playback())
{
   string expectedImagePath = "sun.jpg";
   ImageManagement image = new ImageManagement();
   string path = image.GetImageForTimeOfDay(timeController);
   Assert.AreEqual(expectedImagePath, path);
}
```

Pentru alte variante de apel a mecanismului de moking a se vedea tutorialele de la http://hibernatingrhinos.com/Oss/rhino-mocks/learn/Usage/assert-that-a-method-is-called-with-a-value-in-expected-state

Cele doua teste – de zi si de noapte – pot fi scrise deci prin mocking astfel:

```
[TestMethod]
public void TestMethodDay()
{
    MockRepository mocks = new MockRepository();
    IDateTime timeController = mocks.StrictMock<IDateTime>();

    using (mocks.Record())
    {
        Expect.Call(timeController.GetHour()).Return(9);
    }

    using (mocks.Playback())
    {
        string expectedImagePath = "sun.jpg";
        ImageManagement image = new ImageManagement();
        string path = image.GetImageForTimeOfDay(timeController);
        Assert.AreEqual(expectedImagePath, path);
    }
}

[TestMethod]
public void TestMethodNight()
{
    MockRepository mocks = new MockRepository();
    IDateTime timeController = mocks.StrictMock<IDateTime>();

    using (mocks.Record())
```

```
            {
                Expect.Call(timeController.GetHour()).Return(22);
            }

            using (mocks.Playback())
            {
                string expectedImagePath = "moon.jpg";
                ImageManagement image = new ImageManagement();
                string path = image.GetImageForTimeOfDay(timeController);
                Assert.AreEqual(expectedImagePath, path);
            }
        }
```

Teme:

1. Documentati-va despre stub-uri [4] si diferenta dintre stub-uri si mock-uri [5].

Bibliografie

[1] http://aspalliance.com/1400_Beginning_to_Mock_with_Rhino_Mocks_and_MbUnit__Part_1.all
[2] http://stackoverflow.com/questions/1002257/the-purpose-of-mocking
[3] http://martinfowler.com/articles/mocksArentStubs.html#TheDifferenceBetweenMocksAndStubs
[4] pragmatic-unit-testing-in-c-with-nunit-2nd-edition
[5] http://martinfowler.com/articles/mocksArentStubs.html#TheDifferenceBetweenMocksAndStubs
[6] http://www.mockobjects.com/files/usingmocksandtests.pdf
[7] http://www.mockobjects.com/
[8] http://www.jmock.org/oopsla2004.pdf
[9] http://ayende.com/Blog/archive/2007/03/28/Hibernating-Rhinos--Episode-1-Rhino-Mocks-101.aspx
[10] http://www.codeproject.com/KB/dotnet/Rhino_Mocks.aspx
[11] http://serviciipeweb.ro/iafblog/2009/09/21/RhinoMock.aspx
[12] (excelent) http://www.techscreencast.com/language/dotnet/introduction-to-mocking-with-rhino-mocks/1627
[13] http://stackoverflow.com/questions/1595166/why-is-it-so-bad-to-mock-classes