Haladó fejlesztési technikák

Dependency Injection

Tesztduplikátumok, Mock-ok, Fake-ek

„Tesztelés = kipróbálom kézzel”

„Majd a tesztelő leteszteli”

Jól tesztelni nehéz

- **Legyenek a tesztek gyorsak**
  - Lassú tesztek folyamatos futtatása nem lehetséges => hibák késői kiderülését okozhatja
- **Legyenek lehetőleg egymástól függetlenek**
  - Sorrend, időzítés stb. nem hathat ki az eredményre
- **Ránézésre olvasható nevekkel**
  - A jól elkészített tesztlista dokumentálja a kód képességeit (requirement-listának fogható fel)
- **Nem kell és nem is szabad minden inputlehetőséget lefedni**
  - Sokszor példákkal tesztelnek
  - Corner case-ek megtalálása fontosabb
- **Egyszerre egyetlen művelet, és egy osztály tesztelése**
  - Mindig függetlenül az éles adattól, ami változhat – tehát nem olvasunk éles adatbázist, éles settingsfájlt, stb.
  - Osztályok függőségeit is helyettesítjük: Dependency Injection + tesztduplikátumok (test doubles)

Warning – nem unittest!

```
public class Class1
{
    Dependency2 dep2;

    public Class1(Dependency2 dep2)
    {
        this.dep2 = dep2;
    }

    public void DoSomething(Dependency3 dep3)
    {
        Dependency1 dep1 = new Dependency1();
        dep1.DoSomething();
        dep3.DoSomething();
    }
}
```

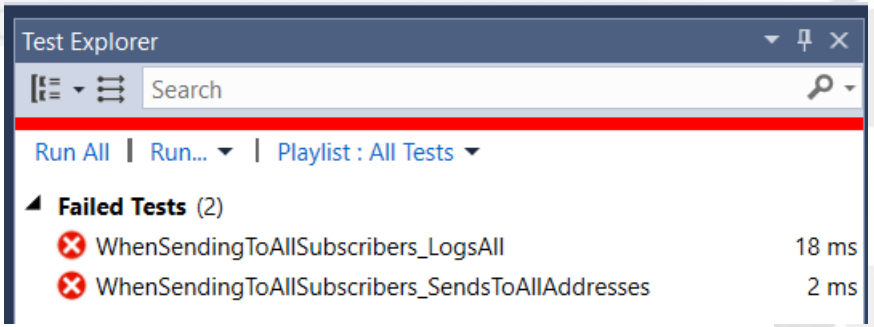
```
[TestFixture]
public class Class1Tests
{
    [Test]
    public void Class1_DoSomething_ResultIsAsExpected()
    {
        Class1 class1 = new Class1(new Dependency2());
        class1.DoSomething(new Dependency3());
        // Assert something...
    }
}
```

Példa

```
public class EmailSender
{
    public void SendEmail(string from, string to, string subject, string body)
    {
        // ...
    }
}

public class Logger
{
    public void Log(int logLevel, string logMessage)
    {
        // ...
    }
}
```

Cél: Refaktorálás




```
class NewsletterService
{
    EmailSender emailSender;

    public NewsletterService()
    {
        this.emailSender = new EmailSender();
    }

    public void SendToAllSubscribers(string subject, string body)
    {
        NewsletterServiceEntities entities = new NewsletterServiceEntities();
        EmailLogger logger = new EmailLogger("logsender", "sends.logs@to.this"); // ???

        logger.Log(1, "Sending newsletter: ");
        foreach (var subscriber in entities.Subscribers)
        {
            emailSender.SendEmail("NewsletterService", subscriber.Email, subject, body);
            logger.Log(1, subscriber.Email);
        }
    }
}
```

Tight coupling

- **Refaktorálás = a kód struktúrájának módosítása a viselkedés módosítása nélkül**
- **Refactoring for testability**

1. Paraméterek használata

```
class NewsletterService
{
    EmailSender emailSender;

    public NewsletterService(EmailSender emailSender)
    {
        this.emailSender = emailSender;
    }

    public void SendToAllSubscribers(EmailLogger logger, string subject, string body)
    {
        NewsletterServiceEntities entities = new NewsletterServiceEntities();

        logger.Log(1, "Sending newsletter: ");
        foreach (var subscriber in entities.Subscribers)
        {
            emailSender.SendEmail("NewsletterService", subscriber.Email, subject, body);
            logger.Log(1, subscriber.Email);
        }
    }
}
```

2. Interfész típusú paraméterek

```
class NewsletterService
{
    IEmailSender emailSender;

    public NewsletterService(IEmailSender emailSender)
    {
        this.emailSender = emailSender;
    }

    public void SendToAllSubscribers(ILogger logger, string subject, string body)
    {
        NewsletterServiceEntities entities = new NewsletterServiceEntities();

        logger.Log(1, "Sending newsletter: ");
        foreach (var subscriber in entities.Subscribers)
        {
            emailSender.S
            logger.Log(1,
        }
    }
}
```

```
public interface ILogger
{
    void Log(int logLevel, string logMessage);
}

public class EmailLogger : ILogger
{
    string from;
    string to;

    public EmailLogger(string from, string to)
    {
        this.from = from;
```

3. Rejtett függőségek kerülése

```
class NewsletterService
{
    IEmailSender emailSender;
    ISubscriberRepository subscriberRepository;

    public NewsletterService(IEmailSender emailSender, ISubscriberRepository subscriberRepository)
    {
        this.emailSender = emailSender;
        this.subscriberRepository = subscriberRepository;
    }

    public void SendToAllSubscribers(ILogger logger, string subject, string body)
    {
        logger.Log(1, "Sending newsletter: ");
        foreach (var subscriber in subscriberRepository.Subscribers)
        {
            emailSender.SendEmail("NewsletterService", subscriber.Email, subject, body);
            logger.Log(1, subscriber.Email);
        }
    }
}
```

Loose coupling

!!! Dependency Injection !!!

```
interface ISubscriberRepository
{
    IEnumerable<Subscriber> Subscribers { get; }
}
```

Constructor Injection

```
interface IMyDependency
{
    string DoSomething();
}

class MyClass
{
    private IMyDependency dependency;

    public MyClass(IMyDependency dependency)
    {
        this.dependency = dependency;
    }

    // metódusaiban: dependency.DoSomething()
}

// használata
IMyDependency myDependency; // értékadás, példányosítás ...
MyClass myClass = new MyClass(myDependency);
```

Method Injection, Setter Injection

```
class MyClass
{
    public void DoSomethingUsingDependency(IMyDependency dependency)
    {
        // ... dependency.DoSomething() ...
    }
}
```

```
class MyClass
{
    public IMyDependency Dependency { get; set; }

    // metódusaiban: dependency.DoSomething()
}
```

- **Constructor Injection – gyakori, ha sok helyen kell a függőség**
- **Method Injection – gyakori, ha egy helyen kell a függőség**
- **Setter Injection – ritka**
  - Mindenhol ellenőrizni kell, hogy be lett-e már állítva
  - Többszálúság még több gondot okoz (nem lett-e újra null?)
- **Most már használhatunk test double-öket!**

Dependency Injection

„

I give you dependency injection for five-year-olds.

When you go and get things out of the refrigerator for yourself, you can cause problems. You might leave the door open, you might get something Mommy or Daddy doesn't want you to have. You might even be looking for something we don't even have or which has expired.

What you should be doing is stating a need, "I need something to drink with lunch," and then we will make sure you have something when you sit down to eat.

„

https://stackoverflow.com/questions/1638919/how-to-explain-dependency-injection-to-a-5-year-old/1638961#1638961

(Injection vs Inversion, ebben a félévben csak az Injection kell)

Dependency Injection

- **Az osztályok függőségeit kívülről biztosítjuk – interfészek formájában**
  - A DI lehetővé teszi a fejlesztés előrehaladtával a függőség lecserélését – az interfész megtartása mellett
  - Lehetővé teszi a „valódi” unit- („egység-”) tesztelést is
- **Tesztelésnél szándékosan cseréljük a függőséget egy ismert, egyszerű, e célra létrehozott objektumra**
  - Osztály függetlenítése a függőségeitől („Mindig függetlenül az éles adattól” ... „Egyszerre egyetlen művelet, és egy osztály tesztelése”)
  - Gyorsaság
    - Komplex osztály helyett nagyon egyszerű csereobjektum, csak azokkal az aspektusokkal, amelyekről a teszt szól
  - Függetlenség
    - 1 teszt => 1 csereobjektum = Test Double
    - Ha a tesztelt osztály esetleg írni akar az objektumba, az se gond
  - Technikái: Dummy, Stub, Spy, Fake, Mock: Azonos interface, más filozófia

Test doubles

Pattern	Purpose	Has Behavior	Injects **indirect inputs** into SUT	Handles **indirect outputs** of SUT	Values provided by test(er)	Examples
Test Double	Generic name for family					
Dummy Object *(page X)*	Attribute or Method Parameter	no	no, never called	no, never called	no	Null, "Ignored String", new Object()
Test Stub *(page X)*	Verify indirect inputs of SUT	yes	yes	ignores them	inputs	
Test Spy *(page X)*	Verify indirect outputs of SUT	yes	optional	captures them for later verification	inputs (optional)	
Mock Object *(page X)*	Verify indirect outputs of SUT	yes	optional	verifies correctness against expectations	outputs & inputs (optional)	
Fake Object *(page X)*	Run (unrunnable) tests (faster)	yes	no	uses them	none	In-memory database emulator
Temporary Test Stub *(see Test Stub)*	Stand in for procedural code not yet written	yes	no	uses them	none	In-memory database emulator

Forrás: http://xunitpatterns.com/Mocks,%20Fakes,%20Stubs%20and%20Dummies.html

Test doubles

	Sources and Names Used in them								
Pattern	Astels	Beck	Feathers	Fowler	jMock	UTWJ	OMG	Pragmatic	Recipes
Test Double								Double or stand-in	
Dummy Object	Stub				Dummy				Stub
Test Stub	Fake		Fake	Stub	Stub	Dummy		Mock	Fake
Test Spy						Dummy			Spy
Mock Object	Mock		Mock	Mock	Mock	Mock		Mock	Mock
Fake Object						Dummy			
Temporary Test Stub						Stub			
OMG's CORBA Stub							Stub		

- **Astels = David Astels: Test-Driven Development - A practical guide**
- **Beck = Kent Beck: Test-Driven Development By Example**
- **Feathers = Michael Feathers: Working Effectively with Legacy Code**
- **Fowler = Martin Fowler: Mocks are Stubs**
- **jMock = Steve Freeman, Tim Mackinnon, Nat Pryce, Joe Walnes: Mock Roles, Not Objects**
- **UTWJ = Morgan Kaufmann: Unit Testing With Java**
- **OMG = Object Management Group's CORBA specs**
- **Pragmatic = Andy Hunt, Dave Thomas: Pragmatic Unit Testing with Nunit**
- **Recipes = J.B.Rainsberger: JUnit Recipes**

Fake-ek

- **Azonos interfészt implementál a helyettesítendő objektummal, de szándékosan egyszerűsítő, tesztcélú logikát tartalmaz, más „célponttal”**

```
public class FakeEmailSender : IEmailSender
{
    public List<string> SentTo { get; set; }

    public FakeEmailSender()
    {
        SentTo = new List<string>();
    }

    public void SendEmail(string from, string to, string subject, string body)
    {
        SentTo.Add(to);
    }
}

public class FakeLogger : ILogger
{
    public void Log(int logLevel, string logMessage)
    {
    }
}
```

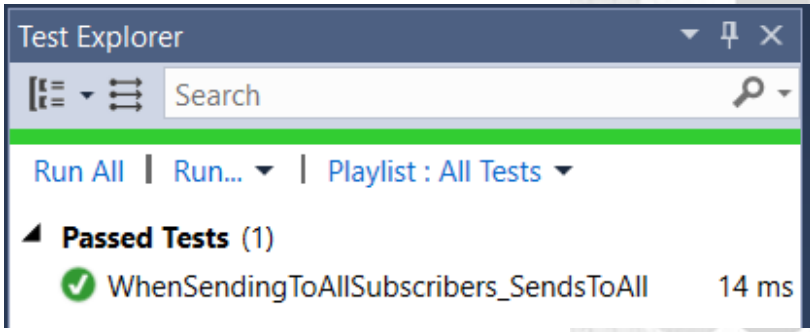
Fake-ek

```
public class FakeSubscriberRepository : ISubscriberRepository
{
    public IEnumerable<ISubscriber> Subscribers
    {
        get
        {
            yield return new Subscriber() { Email = "a@a.hu" };
            yield return new Subscriber() { Email = "b@b.hu" };
        }
    }
}

[TestFixture]
public class NewsletterServiceTests
{
    [Test]
    public void WhenSendingToAllSubscribers_SendsToAll()
    {
        FakeEmailSender sender = new FakeEmailSender();
        FakeSubscriberRepository repository = new FakeSubscriberRepository();

        NewsletterService newsletterService = new NewsletterService(sender, repository);
        newsletterService.SendToAllSubscribers(new FakeLogger(), "subject", "body");

        Assert.That(sender.SentTo, Is.EquivalentTo(new[] { "a@a.hu", "b@b.hu" }));
    }
```



- **Fake Repository ➔ túl sok kód, nehéz univerzális fake-et készíteni**

Mock-ok

- **„Részben” implementál interfészt a helyettesítendő objektummal – nincs logika, fix adat**
- **Mocking framework segítségével szokás készíteni**

```
Mock<IEmailSender> mockMailSender = new Mock<IEmailSender>();

// mock elvárt viselkedésének beállítása...

NewsletterService service = new NewsletterService(mockMailSender.Object);
```

  - Mocking framework képességeitől függően a mock, fake, stub stb. különbség elmosódhat
  - Nagyon gyakori, hogy egyszerűen fake-nek használják a mockokat (és nincs expectation-ellenőrzés)
- **Az egyik legelterjedtebb .NET mocking framework: Moq**
  - CSAK Interfészt mockolunk (bár újabb Moq esetén lehetne mást is)
- **Általában lehetőséget ad a végrehajtódó logika megfigyelésére is**
  - Expectation-ök: „a DoSomething() függvény pontosan egyszer hívódott”

Mock elvárt viselkedésének beállítása

- **Interfész implementáció készíthető tényleges osztály nélkül**
  - Bármely metódusa meghívható, nem dob exceptiont, default értéket ad vissza (MockBehavior.Loose vs Strict)
  - Property nem jegyzi meg az értékét, default értéket ad vissza
  - .Setup/.SetupAllProperties által módosítható a működése
- **Különböző mock készíthető különböző tesztesetekhez**
  - Csak a szükséges műveleteket kell setup-olni a teszthez megfelelő adatokkal

```
Mock<ISubscriberRepository> repositoryMock = new Mock<ISubscriberRepository>();
Mock<IEmailSender> senderMock = new Mock<IEmailSender>();

// különféle elvárt viselkedések beállítása, paraméterektől függően
repositoryMock.Setup(m => m.Subscribers).Returns(expectedSubscribers);
repositoryMock.Setup(m => m.Subscribers).Returns(Enumerable.Empty<Subscriber>());
repositoryMock.Setup(m => m.Subscribers).Throws<NullReferenceException>();

senderMock.Setup(
     m => m.SendEmail("a@a.hu", "to", "subject", "body")).Returns(true);
senderMock.Setup(
     m => m.SendEmail(null, It.IsAny<string>(), It.IsAny<string>(), It.IsAny<string>()))
          .Throws<NullReferenceException>();
```

Logika ellenőrzése mockkal (csak Setup után)

```
// ARRANGE
Mock<IEmployeeBusinessLogic> mbl = new Mock<IEmployeeBusinessLogic>();
// ... Setup a később ellenőrzendő függvényekre, propertykre...

// ACT ...

// ASSERT
// 1x hívódott a@a.hu-ra
senderMock.Verify(
    m => m.SendEmail("a@a.hu", It.IsAny<string>(), It.IsAny<string>(), It.IsAny<string>
    Times.Once);

// sosem hívódott nullal
senderMock.Verify(
    m => m.SendEmail(null, It.IsAny<string>(), It.IsAny<string>(), It.IsAny<string>()),
    Times.Never);

// 0x volt 1-nél nagyobb log
loggerMock.Verify(
    m => m.Log(It.Is<int>(i => i > 1), It.IsAny<string>()), Times.Never);
```

Példa

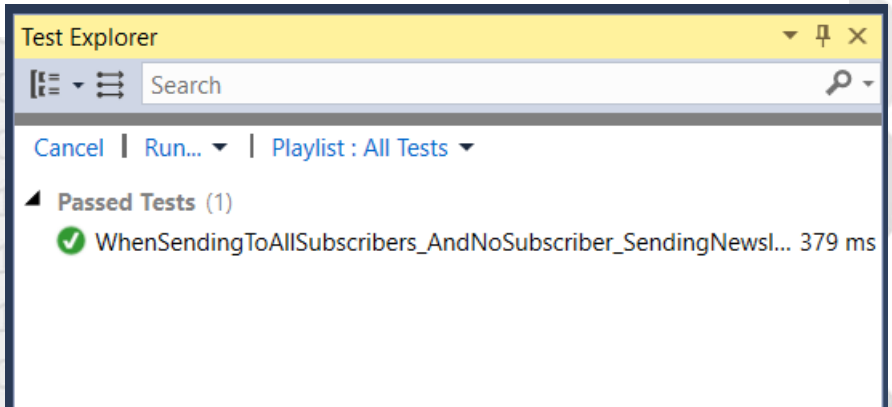
```
[Test]
public void WhenSendingToAllSubscribers_AndNoSubscriber_SendingNewsletterLogMessageIsAdded()
{
    // ARRANGE
    Mock<IEmailSender> senderMock = new Mock<IEmailSender>();
    Mock <ISubscriberRepository> repositoryMock = new Mock<ISubscriberRepository>();
    repositoryMock.Setup(m => m.Subscribers).Returns(Enumerable.Empty<Subscriber>());

    NewsletterService newsletterService =
        new NewsletterService(senderMock.Object, repositoryMock.Object);

    Mock<ILogger> loggerMock = new Mock<ILogger>();
    loggerMock.Setup(m => m.Log(It.IsAny<int>(), It.IsAny<string>()));

    // ACT
    newsletterService.SendToAllSubscribers(loggerMock.Object, "subject", "body");

    // ASSERT
    loggerMock.Verify(m => m.Log(It.IsAny<int>(), "Sending newsletter: "), Times.Once);
}
```



Testability Code Smells

- **Referenciák osztály és nem interfész típusú változókban**
  - Refaktorálás interfész típusú változókká
- **new()**
  - Refaktorálás DI irányba (gyakran Factory pattern)
- **DI: túl sok paraméter**
  - Túl sok dologért felelős az osztály, refaktorálás / bontás
  - Esetleg összefoglalás közös objektumba
- **Statikus osztályok és változók használata (pl. Singleton pattern)**
  - Refaktorálás példányok használatára
- **Sok/hosszú statikus helper metódus**
  - Strukturált programozás OO helyett, kiszervezés nemstatikus osztályba
- **Felhasználói interakciótól függő részek, ablakok**
  - Kiszervezés külön osztályba (Humble Object pattern)
- **Adatbázis, fájlírás, egyéb külső erőforrás**
  - Kiszervezés külön osztályba (Repository, Humble Object pattern)
- **Túl sokat kell mockolni**
  - Túl sok dologért felelős az osztály, refaktorálás / bontás
- **A mock.Object közvetlen elérése → TILOS**

Feladat

- **Hozzon létre üzleti logikát az autókhoz és a márkákhoz**
  - Adathozzáférés Repository és interfészek segítségével
  - A logic legyen képes márkára szűrni, illetve a márkánkénti átlagokat többféleképp kiszámolni

- **A logika teszteléséhez NE a tényleges adatbázist használja!**
  - Moq segítségével az Arrange fázisban hozzon létre mockolt repository-t
  - Az adatlekérő metódusok az aktuális tesztesethez illő adatokat adja vissza
  - A teszt során ellenőrizze, hogy a logika jó eredménnyel tér vissza, illetve a repository adatkezelő metódusait megfelelően használja