

Haladó fejlesztési technikák

Tesztelés a fejlesztői munkában

NUnit

Unit teszt

**„Tesztelés =
kipróbálom kézzel”**

Tesztelés szükségessége

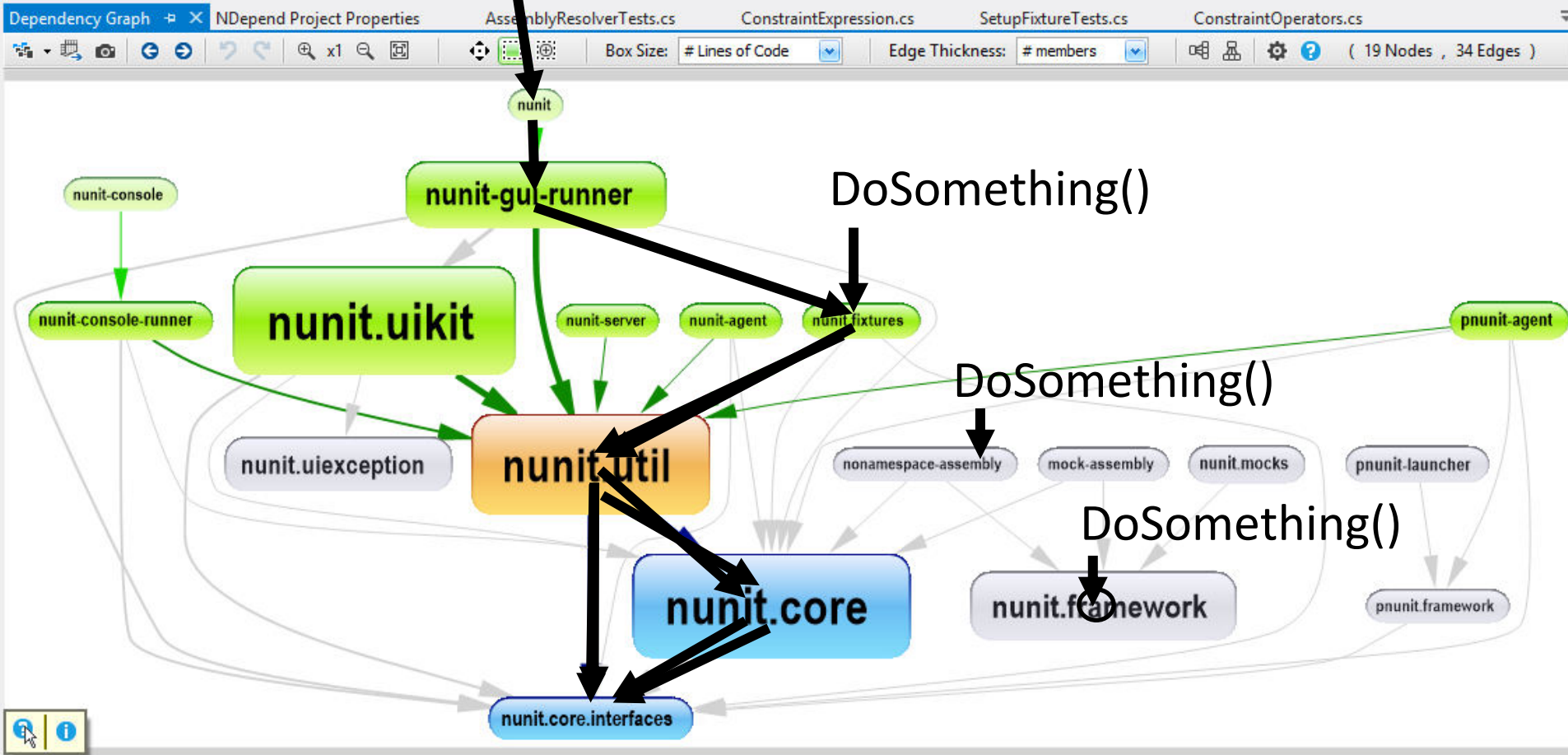
- Tesztesetek száma?

$> 2^n \dots ?$

```
if (condition1)
{
    // something happens ← Random
    if (condition2)
    {
        // something happens ← exception
        if (condition3)
        {
            // something happens ← befolyásolja condition3-at
            // something happens ← user inputot használ
        }
        // something happens ← adatbáziselérés
    }
}
```

- Mekkora egy projekt?

Tesztelés szintjei



- **UI teszt**
- **Integrációs teszt**
- **Komponensteszt**
- **Unit teszt**

(Az elnevezések fejlesztői közösségenként változhatnak)

**„Majd a tesztelő
le teszteli”**

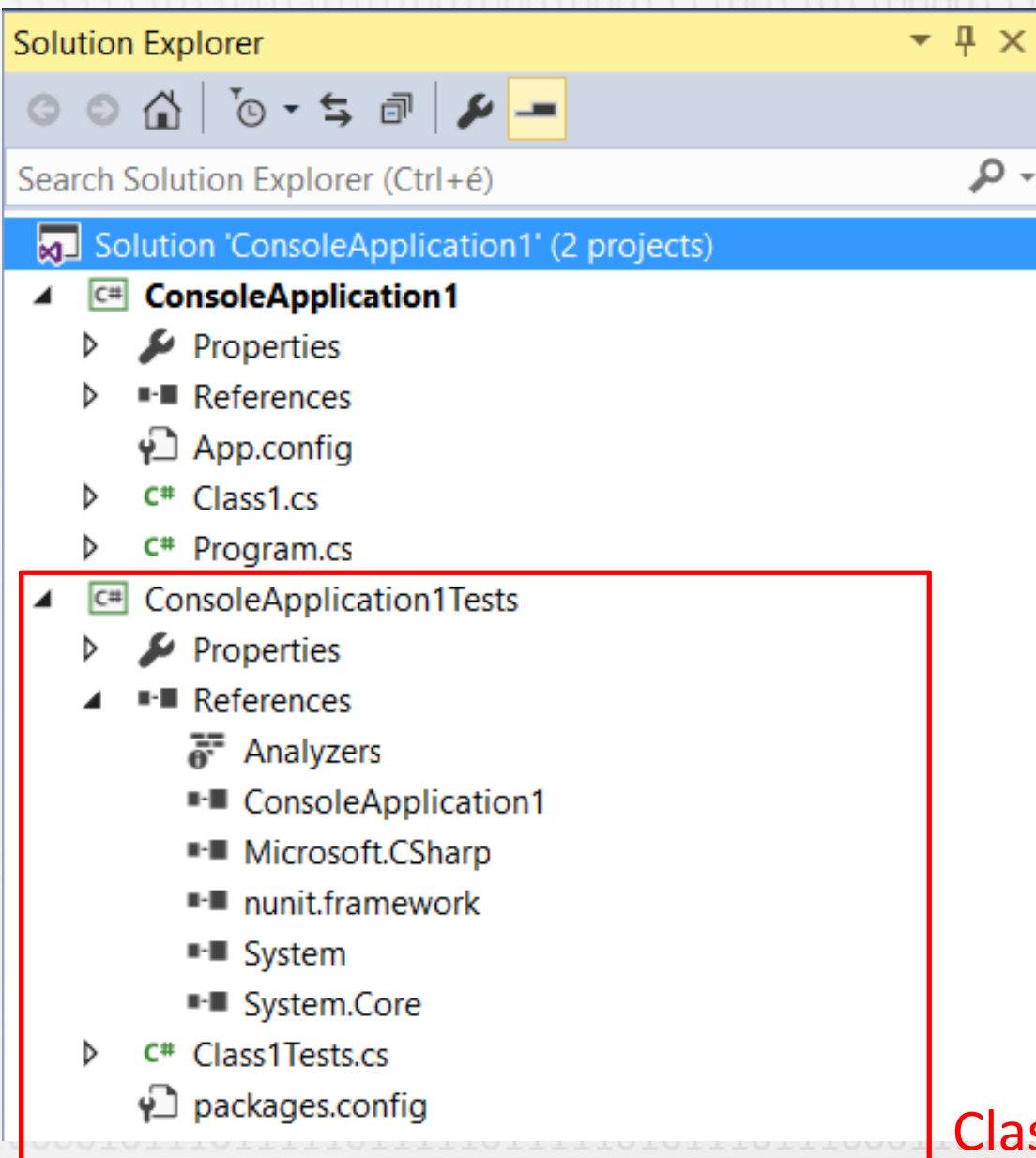
Tesztelés a fejlesztői munkában

- **A tesztelés közelebb került az implementációhoz, mert:**
 - Korai feedbacket ad
 - Védi a lefedett kódot a későbbi véletlen hibák ellen
 - Követelmények tisztázását segíti
 - Tiszta, jól strukturált kód írását kényszeríti ki
- **(Bizonyos módszertanokban a tesztelés az implementáció elé került)**
- **„Agile crossfunctional team”**
 - A termék/funkció fejlesztésének összes aspektusáért – tervezésért, implementációjáért, minőségellenőrzéséért, teszteléséért felel
 - Egyre inkább a fenntartásért / supportért is
 - A régi szerepek elmosódnak vagy szándékosan hiányoznak – „crossfunctional team”

NUnit

- **Teszt keretrendszer .NET nyelvekhez (Java: junit)**
- **Nagyobb programcsalád része**
 - Unitteszt: **NUnit** (vs. Visual Studio Unit Testing Framework)
 - Mocking: NSubstitute (vs. **Moq**, Rhino Mocks)
 - IoC container: Ninject (vs. Spring.Net, Castle Windsor, Unity, AutoFAC)
 - Tesztlefedettség: NCover (vs. dotCover)
- **Elterjedtebb (...)**
 - Jobb/olvashatóbb szintaxis
 - Unit tesztől integrációs teszthoz minden tesztszinthez jó
 - Saját GUI
 - Command line Test Runner (Dotnet Core: beta)
 - VS-ben Test Explorer (NUnit3TestAdapter kell hozzá)

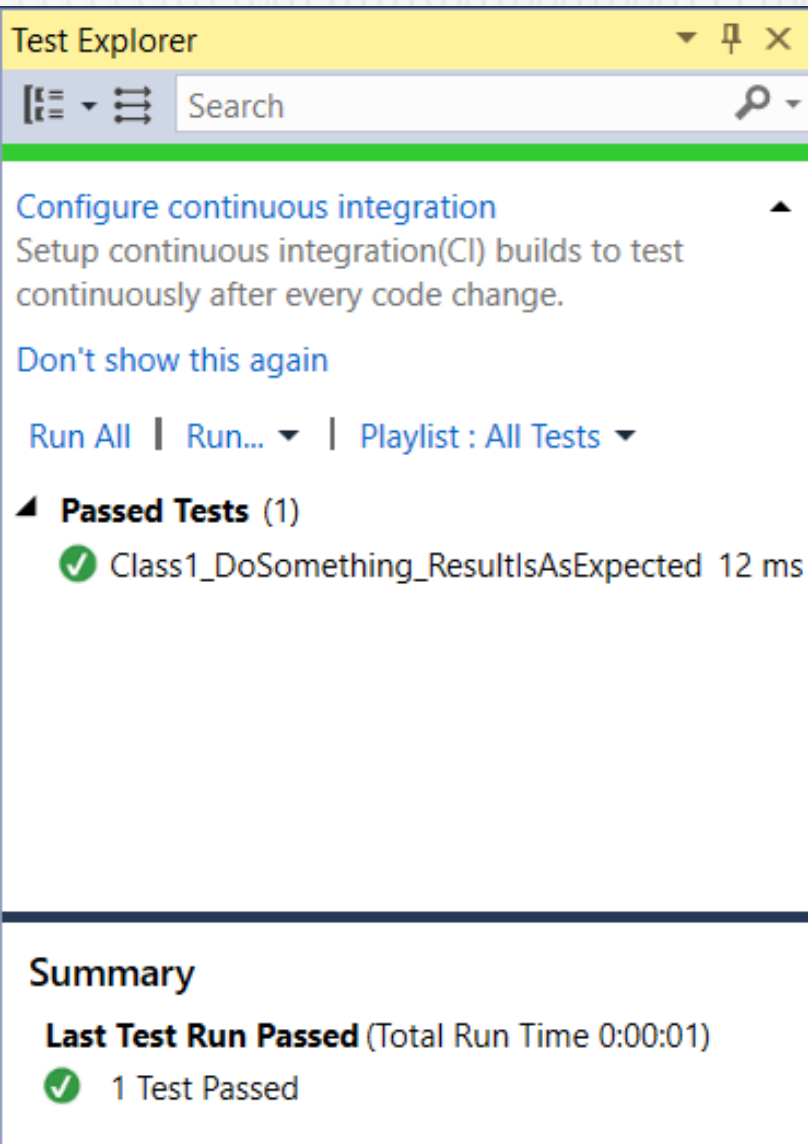
Egyszerű tesztelt projekt



- Éles projekt(ek) + tesztprojekt(ek) (Class Library)
- Tesztprojekt referenciái:
 - Tesztelt projekt
 - Tesztkeretrendszer
- A tesztprojektben a tesztelt projekt **publikus** részei tesztelhetők
 - Internal részek:
[InternalsVisibleTo(„ConsoleApplication1Tests”)]

Class Library

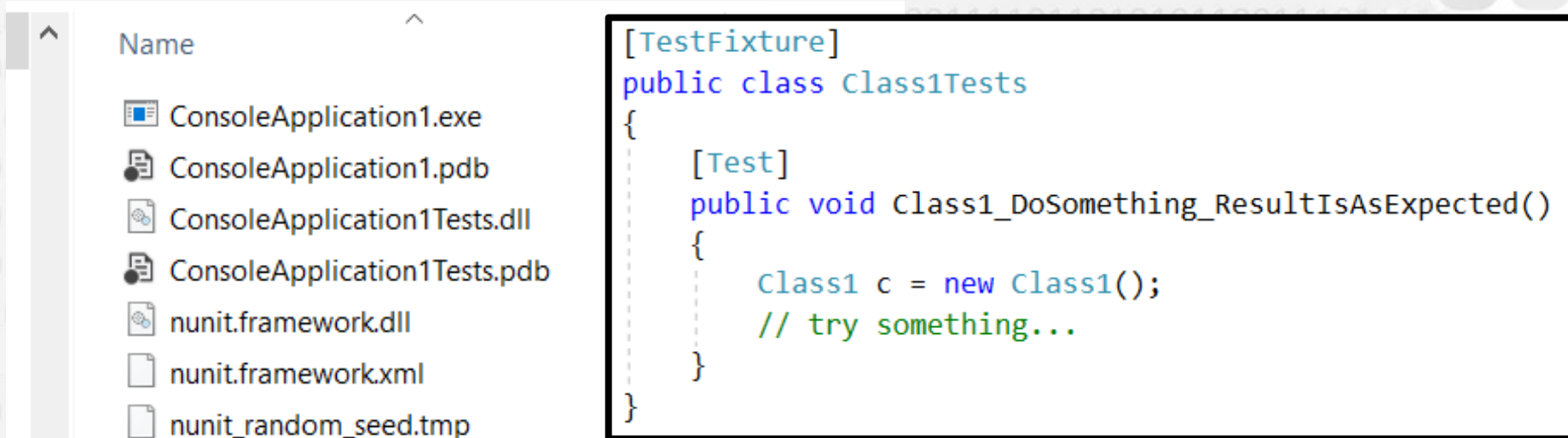
Egyszerű tesztelt projekt



- Test / Windows / Test Explorer
- NUnit3TestAdapter kell hozzá! (Nuget)
- Fejlesztői közösség szokásától és tesztektől függően
 - folyamatosan,
 - vagy lényeges kódváltoztatás után,
 - merge előtt, merge után,
 - push előtt mindenképpen futtatják.
- Piros teszttel nem pusholnak kódot

Hogyan működik?

- **Reflection**



```
[TestFixture]
public class Class1Tests
{
    [Test]
    public void Class1_DoSomething_ResultIsAsExpected()
    {
        Class1 c = new Class1();
        // try something...
    }
}
```

- **Studio Test Explorer => NUnit**

- NUnit reflexióval végigkeresi a solution(ból keletkező exék és dll-ek) összes elérhető osztályát [TestFixture] attribútum után
- [TestFixture] osztály [Test] attribútummal ellátott függvényei a tesztek, ezeket is megkeresi
- Értelmezi a tesztek működését/inputjait/stb. befolyásoló egyéb attribútumokat ([TestCaseSource], [Explicit],)
- Reflexióval futtatja a teszteket

NUnit

```
[TestFixture]
public class Class1Tests
{
    [Test]
    public void Class1_DoSomething_ResultIsAsExpected()
    {
        // ARRANGE
        Class1 tc = new Class1();

        // ACT
        tc.DoSomething();

        // ASSERT
        Assert.That(tc.Result, Is.EqualTo(0));
    }
}
```

- **Gyakori felépítés („AAA”)**

- Arrange: előkészítés, objektumok létrehozása, beállítása stb.
- Act: tesztelendő **egyetlen** lépés végrehajtása
- Assert: az eredmény az elvárásnak megfelelő-e:
eredmény / állapotváltozás /
viselkedés (kivételek, események) / körülmények (backend hívások)

- **Given...When...Then()**

- **Spec / SpecFlow (BDD)**

NUnit

```
// ASSERT
```

```
Assert.That(tc.Result, Is.EqualTo(0));
```

- **Rengeteg lehetőség**

```
// Elvárások :
```

```
Assert.That(result, Is.EqualTo("MyResult"));
```

```
Assert.That(result, Is.Null);
```

```
Assert.That(result, Is.LessThan(2));
```

```
Assert.That(result, Is.SameAs(otherReferenceToTheSameObject));
```

```
Assert.That(result, Is.Not.Null); // tagadás
```

```
// Exception-ellenőrzések:
```

```
Assert.That(() => t.MyTestedMethod(),  
            Throws.TypeOf<NullReferenceException>());
```

```
Assert.That(() => t.MyTestedMethod(), Throws.Nothing);
```

```
// régi szintaktika: (ugyanaz, mint az Is.EqualTo)
```

```
Assert.AreEqual(result, 42);
```

NUnit

```
[TestCase(1, 2)]  
[TestCase(2, 4)]  
[TestCase(5, 10)]  
[TestCase(128, 256)]  
public void Class1_DoSomethingWithInput_ResultIsAsExpected(int input, int expected)  
{  
    // ARRANGE  
    Class1 tc = new Class1();  
  
    // ACT  
    tc.DoSomething(input);  
  
    // ASSERT  
    Assert.That(tc.Result, Is.EqualTo(expected));  
}
```

- Akárhány input paraméter lehet a tesztmetódusban – ugyanennyit kell megadni a TestCase attribútumban
 - Attribútum csak konstans értékeket kaphat -> **TestCaseSource!**
- Hasonló lehetőségek:
 - [Sequential] – az input paraméterek megadott értékek közül sorra kapnak értékeket
 - [Combinatorial] – az input paraméterek számára megadott értékek összes lehetséges kombinációjával meghívódik a teszt
 - [Pairwise] – az előzőhöz képest optimalizált (kevesebb tesztesetet csinál)

NUnit

- **TestCaseSource**

- Dinamikusan készített v. nem konstans értékekkel dolgozó tesztesetekhez – akár saját osztályok átadása értékeként
- Sokféle formában használható: object[] helyett TestCaseData utód, property helyett metódus, osztály...

```
public static IEnumerable<TestCaseData> MyTestCases
{
    get
    {
        List<TestCaseData> testCases = new List<TestCaseData>();
        for (int i = 0; i < 10; i++)
        {
            testCases.Add(new TestCaseData(new object[] { i, i * 2 }));
        }

        return testCases;
    }
}

[TestCaseSource(nameof(MyTestCases))]
public void Class1_DoSomethingWithInput_ResultIsAsExpected(int input, int expected)
{
    // ... mint előbb
}
```

NUnit

- **[Setup]**

- Az így jelölt metódus minden egyes teszt vagy teszteset előtt le fog futni

- **[TearDown]**

- Az így jelölt metódus minden egyes teszt vagy teszteset futása után lefut

- **[TestFixtureSetUp] / [OneTimeSetUp]**

- Az így jelölt metódus az adott [TestFixture] tesztjeinek futtatása előtt fut, egyszer
- Pl. az összes teszt által használt objektumok hozhatók létre így – de vigyázzunk, a tesztek ideális esetben nem befolyásolhatják egymást!

- **[TestFixtureTearDown] / [OneTimeTearDown]**

- Az így jelölt metódus az adott [TestFixture] tesztjeinek futtatása után fut, egyszer

- **[SetUpFixture]**

- Osztályra rakható, namespace szintű setup/teardown

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)

Tesztesetek – egyszerű kód?

```
char[,] map;
```

```
// Generate map every time this
```

```
public char[,] Map
```

```
{
```

```
    get
```

```
{
```

```
        for (int x = 0; x < map.GetLength(0); x++)
```

```
        {
```

```
            for (int y = 0; y <
```

```
            {
```

```
                map[x, y] = '-';
```

```
            }
```

```
        }
```

✓ WhenGameIsCreatedWithNegativeWidthOrHeight_ThrowsException(-

✓ WhenGameIsCreatedWithValidWidthOrHeight_MapReturnsNxNArray(

✗ WhenGameIsNXN_MapReturnsNxNArray(-1,100)

✗ WhenGameIsNXN_MapReturnsNxNArray(100,-1)

```
[TestCase(100, -1)]
```

```
[TestCase(-1, 100)]
```

```
[TestCase(3, 3)]
```

```
[TestCase(100, 100)]
```

```
public void WhenGameIsNXN_MapReturnsNxNArray(int width
```

```
[TestCase(100, -1)]
```

```
[TestCase(-1, 100)]
```

```
public void WhenGameIsCreatedWithNegativeWidthOrHeight_ThrowsException(int wi
```

```
{
```

```
    Assert.That(() => new Game(width, height), Throws.ArgumentException);
```

```
}
```

```
public Game(int max_x, int max_y)
```

```
{    if (max_x < 0 || max_y < 0) throw new ArgumentException("...");
```

```
    map = new char[max_x, max_y];
```

```
}
```

Tesztesetek – egyszerű kód?

```
char[,] map;  
// Generate map every time  
public char[,] Map  
{  
    get  
    {  
        for (int x = 0; x < map.GetLength(0); x++)  
        {  
            for (int y = 0; y < map.GetLength(1); y++)  
            {  
                map[x, y] = 'X';  
            }  
        }  
        foreach (var akt in Enumerable.Range(0, 100).Select(i => i * 10))  
        {  
            map[akt, Position] = 'X';  
        }  
        return map;  
    }  
}
```

- ✓ WhenGameIsCreatedWithInvalidWidthOrHeight_ThrowsException()
- ✓ WhenGameIsCreatedWithValidWidthOrHeight_MapReturnsNxNArray(100,100)
- ✓ WhenGameIsCreatedWithValidWidthOrHeight_MapReturnsNxNArray(3,3)
- ✓ WhenGameIsCreatedWithValidWidthOrHeight_MapReturnsNxNArray(100,100)
- ✓ WhenGameIsCreatedWithValidWidthOrHeight_MapReturnsNxNArray(3,3)

```
[TestCase(0, 10)]  
[TestCase(10, 0)]  
[TestCase(3, 3)]  
[TestCase(0, 1)]  
[TestCase(1, 0)]  
[TestCase(100, -1)]  
[TestCase(-1, 100)]  
public void WhenGameIsCreatedWithInvalidWidthOrHeight_ThrowsException()  
{  
    Assert.That(() => new Game(width, height), Throws.ArgumentException);  
}
```

```
public Game(int max_x, int max_y)  
{  
    if ((max_x <= 0) || (max_y <= 0)) throw new ArgumentException("n(.)");  
    map = new char[max_x, max_y];  
}
```


Tesztesetek – komplex(ebb) kód

```
char[,] map;  
// Generate map every time  
public char[,] Map  
{  
    get  
    {  
        for (int x = 0; x <  
        {  
            for (int y = 0;  
            {  
                map[x, y] = '-';  
            }  
        }  
        foreach (var  
        {  
            map[akt.  
        }  
        return map;  
    }  
}
```

```
public Game(int max_  
{  
    if (max_x <= 0 || max_  
        map = new char[m
```

- ✓ WhenGameIsCreatedWithInvalidWidthOrHeight_ThrowsException()
- ✓ WhenGameIsCreatedWithValidWidthOrHeight_MapReturnsNxNArray()
- ✓ WhenGameDoesntContainItems_MapContainsDashes(100,1)
- ✓ WhenGameDoesntContainItems_MapContainsDashes(3,3)
- ✓ WhenGameContainsSingleItem_MapContainsItemChar
- ✓ WhenGameContainsMultipleItems_MapContainsItemChars
- ✓ WhenGameContainsMultipleItemsOnTheSamePlace_LastAddedItemCharIsVisible
- ✓ WhenMapsGetTwice_ReturnsTheSameArray
- ✓ WhenMapsGetTwice_AndNewItemWasAdded_MapContainsNewItemChar

[Test]

public

{

Game

game

game

cha

Ass

Ass

}

[Test]

public void WhenMapIsGetTwice_AndNewItemWasAdded_MapContains

{

Game game = new Game(3, 3);

char[,] map = game.Map;

game.AddPlayer(new FollowerEnemy());

char[,] map2 = game.Map;

Assert.That(map, Is.SameAs(map2));

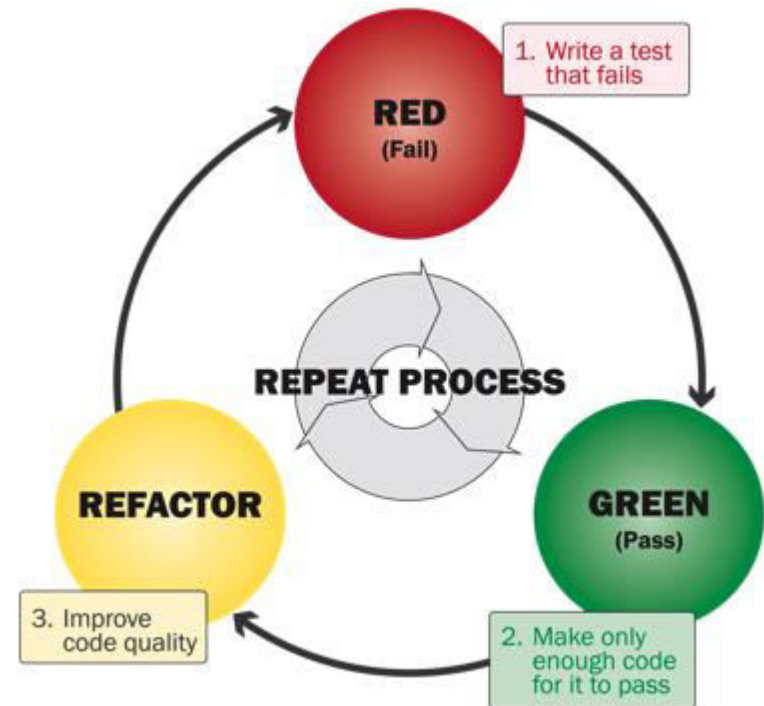
Assert.That(map2, Has.Exactly(1).EqualTo('F'));

Assert.That(map2, Has.Exactly(8).EqualTo('-'));

}

Módszertanok

- **Először kód, utána tesztek**
 - Nehéz, nem hatékony!
- **Test First: a fejlesztő a tesztek készíti el először, utána a kódot**
 - Előre leírja a kívánt követelményeket – eközben valószínűleg többet is tisztázni kell!
 - Vagy biztosítja egy létező kód jelenlegi működésének megtartását átírás előtt
- **TDD (Test-Driven Development):**
 - Sokszor pair programming-ben csinálják
 - A kód ~100%-ban tesztelhetőre „íródik” meg
 - Fejlesztési idő kb. duplázódik
 - Nem mindent érdemes így tesztelni (elsősorban „algoritmikusabb” feladatokhoz ideális)
- **BDD, ATDD...**



Lefedettség

The screenshot displays the Microsoft Visual Studio IDE. The main editor shows the `Chaser.Common.Game` class with a `Map` property and a `Game` constructor. The `Map` property is a `char[,]` array that is generated every time it is read. The `Game` constructor takes `max_x` and `max_y` as parameters and throws an `ArgumentException` if either is less than or equal to 0. The `Map` property is implemented as a `get` method that iterates over the map dimensions and sets the `ItemChar` for each item.

The Unit Test Coverage window on the right shows the coverage for the `Chaser.Common` project. The table below summarizes the data shown in the window:

Symbol	Coverage (%)	Uncovered/Total Stmt.
Total	61%	79/201
Chaser.FollowerEnemy	3%	28/29
Chaser.UserPlayer	14%	12/14
Chaser.Common	59%	37/91
Chaser.Common.Game	50%	34/68
OneTick()	0%	17/17
ToString()	0%	17/17
AddPlayer(IGameItem)	100%	0/6
Game(int,int)	100%	0/7
Map	100%	0/21
MoveDirection	80%	3/15
MyPoint	100%	0/4
ItemWithPos	100%	0/4

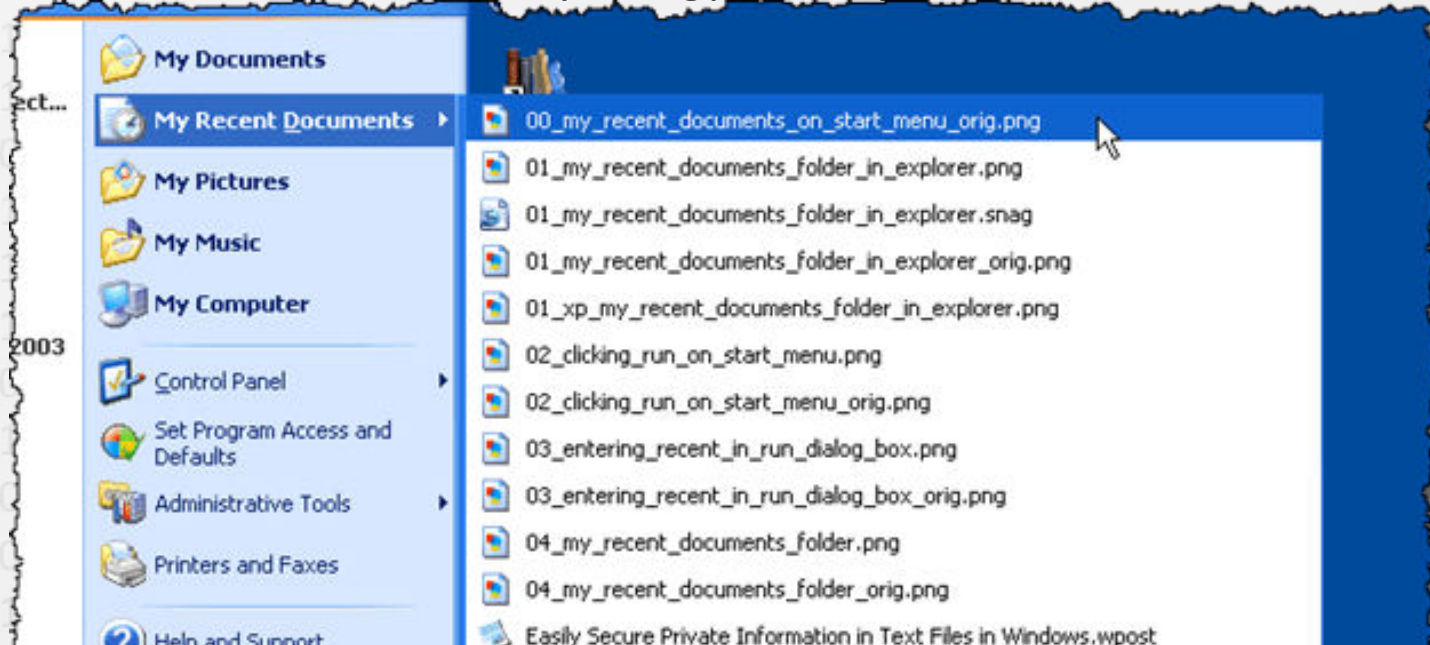
- dotCover
- NCover
- OpenCover

Lefedettség

- Gyakran van alsó határ, de nem érdemes 100%-ra hajtani
- Nagyon sokféleképp lehet mérni – vs Use Case Coverage...
 - Statement Coverage: minden utasítás lefutott (if, ciklus nincs benne)
 - Branch Coverage: minden döntési ág lefutott (if, else)
 - Condition Coverage: minden boolean kifejezés igaz és hamis értéket is kapott
 - Loop Coverage: minden ciklus futott 0x, 1x és >1x, illetve ha lehetséges, a maximum limittel és annál eggyel nagyobb.
 - Parameter Value Coverage: paraméterek összes jellegzetes értéke tesztelendő (pl. string null, üres, whitespace...)
 - Inheritance Coverage: visszaadott objektumtípus tesztelése
 - ...

Feladat

- Készítsen osztályt, ami a Last Recently Used (LRU) funkcionalitást implementálja
- Az osztályban legyen egy maximum kapacitással rendelkező lista, és egy *public void Add(object instance)* metódus
- A fejlesztés közben kövesse a TDD megközelítést
 - Először legyen egy „éppen, de működik” osztály
 - Írjunk tesztek, amíg piros tesztbe nem futunk
 - Javítsuk ki az osztályt, hogy átmenjen az a teszt, utána további tesztek ...



Extra feladat

- **Egy könyvesbolt 5 részes csomagokat szeretne eladni. Egy könyv alapára 8 EUR. Engedmények:**
 - 2 különböző könyv \rightarrow 5% engedmény mindkét könyvre
 - 3 különböző könyv \rightarrow 10% engedmény mindegyik könyvre
 - 4 különböző könyv \rightarrow 20% engedmény mindegyik könyvre
 - 5 különböző könyv \rightarrow 25% engedmény mindegyik könyvre
- **Ha pl. 4 könyvet veszek, közte 3 különböző könyvvel, akkor 10% engedményt kapok arra a háromra, ami különböző, a negyedik marad 8 EUR:**
 - 1, 2, 3, 1 \Rightarrow (1, 2, 3) 10% engedmény, (1) 0% engedmény $\Rightarrow 3 \times 7,2 + 8 = 29,6$
- **Néhány „kosár” többféleképpen csoportosítható!**
 - 1, 1, 2, 2, 3, 3, 4, 5 \Rightarrow (1, 2, 3, 4, 5) (1, 2, 3) or (1, 2, 3, 4) (1, 2, 3, 5) ?
- **Írjon kódot (és teszteket), ami egy „kosár” árát számolja ki**
 - A legegyszerűbb módon csoportosítva: mindig a legnagyobb elemszámú csoportot preferáljuk, akkor is, ha ez nem a legolcsóbb árat eredményezi

