Unit Testy

Jakub Fojtl
Software Architect @ Y Soft

CodeRetreat

- http://coderetreat.cz/

- http://coderetreat.org/

Co jsou to Unit Testy

- Unit Testy jsou malé funkce, které mají za úkol otestovat jednu část funkcionality (jednu “unitu”/jednotku)

- Pokud mi selže test, čím méně věcí ho mohlo rozbít, tím více jde “unit” test

- Unit Test má všechny svoje závislosti načtené v paměti, nesmí si rezervovat žádné prostředky (soubory, databázi) - na rozdíl od integračních

- díky tomu nezáleží, v jakém pořadí test spustím a mohu dokonce všechny testy spustit paralelně

Proč psát Unit Testy

- UT **neslouží** primárně k testování kódu, co jsem právě napsal

- Nicméně mohou pomoct pro **rychlý feedback**, jestli můj předpoklad o tom, jak kód funguje, opravdu platí

- Bez Unit Testů musím **celou aplikaci** spustit a dostat ji do **stavu**, kde se nachází můj výkonný kód, který chci testovat

Proč psát Unit Testy

- Hlavní přínos Unit Testů je pro **budoucí změny**
- když provedu nějakou změnu v kódu, chci **rychle vědět**, jestli jsem něco **nerozbil**
- **živá dokumentace** - komentáře mohou zastarat, testy nikdy
- k tomu, aby nezastarávaly, je potřeba je psát tak, aby přidávaly pouze **minimální práci** na **údržbu**

.NET UT frameworky

- **MSTest** - součástí Visual Studia, ale **nikdo** je **nepoužívá**, dokonce ani Microsoft :)

- **xUnit** - http://xunit.github.io - Brad Wilson, zaměstnanec Microsoftu, používán třeba **ASP.NET**

- **NUnit** - http://nunit.org/ - de facto **standard**, asi nejpoužívanější

- do VS přidáte pomocí "**Tools/Extensions and Updates…**"

Použití v .NET projektu

- StringCalculator - projekt s produkčním kódem, Console Application

- StringCalculator.**Tests** - Class Library, projekt s testy
  (pokud nechcete používat MSTest, nevytvářejte “Unit Test Project”)

- přidejte UT Framework přes **NuGet** (NUnit)

- ```
  [TestFixture]
  class StringCalculatorTests {
      [Test]
      public void Example() { … }
  }
  ```

- **NUnit Test Adapter**, podpora NUnit pro VS Test Runner
  ```

Best Practices

- **Jmenná** konvence
- **Rozvržení** jednotlivých testů
- **Rychlost** - unit testy musí běžet v rámci sekund, spouští se po **každém** ~~sestavení~~ **buildu**
- izolované, nezáleží na pořadí spuštění, testují jednu věc (jeden **Assert** na test)

Jmenná konvence

- **Given** - stav 1, tedy stav systému **před testem** - pouze **relevantní** stav pro test

- **When** - akce, co chceme skutečně testovat, typicky **volání** produkční metody

- **Then** - co se má stát - zkontroluji, že produkční metoda skutečně **udělala to, co má**

Jmenná konvence

- GivenUserIsLoggedIn_WhenSheTriesToLogOut_Then_SheIsLoggedOut

- GivenUserListsHisExams_WhenHisMarkIsLowerThanF_ThenTheRowIsGreen

Rozvržení Testu

- **AAA** struktura, vizuálně oddělené celky v jednom testu
- tahle dohodnutá konvence mi dokáže říct, co je jen boilerplate kód a co je skutečně testovací kód
- **Arrange** - dostat se do stavu 1, typicky inicializuji produkční kód
- **Act** - volání produkční metody
- **Assert** - kontrola, zda produkční metoda udělala, co měla

Situace, do kterých se s testem můžete dostat

- Dle typu, jak se chování produkční metoda
- Referenčně transparentní metoda
- Metoda, která volá jiný modul/třídu
- Metoda, která mění vnitřní stav objektu

Referenčně transparentní metoda

- Produkční metoda je **referenčně transparentní,** tedy její výstup závisí pouze na jejích parametrech

- public int sum(int a, int b) { return a + b; }

- [Test] public void WhenSum1and1_ThenReturn2() {
    var actual = sum(1, 1);

    Assert.That(2, Is.EqualTo(expected));
  }

Metoda, která volá jiný modul/třídu

- Produkční metoda zavolá jiný modul

- public UserService {
    readonly IUserRepository _repository;

    public UserService(IUserRepository repository) {
        _repository = repository;
    }

    public void Save(User user) {
        if(user.IsCreated)
            _repository.Update(user);
        else
            _repository.Create(user);
    }
}

Metoda, která volá jiný modul/třídu

- class FakeUserRepository : IUserRepository {
    public bool CreateCalled { get; set; }
    public bool UpdateCalled { get; set; }

    public FakeUserRepository() {
        CreateCalled = false; UpdateCalled = false;
    }
    public void Create(User user) {
        CreateCalled = true;
    }
    public void Update(User user) {
        UpdateCalled = true;
    }
}

Metoda, která volá jiný modul/třídu

- [Test] public void GivenUserIsNotCreated_WhenISaveUser_ThenHeIsCreated() {
    // Arrange
    var repository = new FakeRepository();
    var service = new UserService(repository);

    // Act
    service.Save(new User());

    // Assert
    Assert.Equals(repository.CreateCalled, true);
}

Metoda, která mění vnitřní stav objektu

- nemůžete dělat nic, pouze se spolehnout na side efekt - event, volání následující metody, kterou byste nemohli ve špatném stavu volat a podobně

- ```
  [Test] public void OpenFile() {
      var file = File.Open("a.txt");

      // Should not throw an exception
      file.ReadAllLines();
  }
  ```
  ```

Odkazy

- **The Art of Unit Testing** http://www.manning.com/osherove2/

- **Test Driven Development by example** - http://www.amazon.com/Test-Driven-Development-By-Example/dp/0321146530 - **bible** :)

- **CastleWindsor** https://github.com/castleproject/Windsor - **reálný projekt** řízen testy