Mockování v .NET (Moq, NSubstitute)

Mockování je technika v jednotkovém testování, která umožňuje nahradit reálné, složité objekty (závislosti) jejich zjednodušenými simulacemi. To je nezbytné pro izolaci testovaného kódu od externích vlivů, jako jsou databáze, API nebo souborový systém.

1. Proč mockovat?

Bez mockování by unit testy nebyly izolované. Pokud testovaná metoda volá databázi, test by mohl selhat kvůli výpadku sítě, nikoliv kvůli chybě v logice.

- **Rychlost:** Mocky existují pouze v paměti, nevykonávají žádné náročné operace.
- **Předvídatelnost:** Mocku můžete přikázat, co přesně má vrátit (např. simulovat chybu nebo prázdný seznam).
- **Zaměření:** Testujete pouze logiku dané třídy, nikoliv integritu celého systému.

—

2. Moq: Standard v .NET světě

Moq je nejrozšířenější knihovna pro mockování v .NET. Využívá silné typování a Lambda výrazy.

Příklad použití Moq

Předpokládejme, že testujeme službu, která potřebuje data z repozitáře.

```
// 1. Vytvoření mocku pro rozhraní
var mockRepo = new Mock<IUserRepository>();

// 2. Nastavení (Setup) - co má mock udělat
mockRepo.Setup(repo => repo.GetById(1)).Returns(new User { Name = "Jan" });

// 3. Použití mocku v testované službě
var service = new UserService(mockRepo.Object);
var result = service.GetUserName(1);

// 4. Ověření (Verify) - byla metoda skutečně zavolána?
mockRepo.Verify(repo => repo.GetById(1), Times.Once());
```

—

3. NSubstitute: Elegantní alternativa

NSubstitute se zaměřuje na co nejjednodušší a nejčitelnější syntaxi. Často je preferován pro svou stručnost.

Příklad použití NSubstitute

Stejný scénář jako výše, ale s NSubstitute:

```
// 1. Vytvoření náhrady
var repo = Substitute.For<IUserRepository>();

// 2. Nastavení (Returns)
repo.GetById(1).Returns(new User { Name = "Jan" });

// 3. Akce
var service = new UserService(repo);
var result = service.GetUserName(1);

// 4. Ověření (Received)
repo.Received().GetById(1);
```

—

4. Srovnání Moq a NSubstitute

Vlastnost	Moq	NSubstitute
Syntaxe	Explicitní (.Setup, .Object)	Implicitní a velmi stručná
Typování	Silně typované (Lambda)	Silně typované (Extension metody)
Čitelnost	Trochu upovídanější	Velmi blízká přirozenému jazyku
Učení	Vyžaduje pochopení Mock<T>	Intuitivní od prvního použití

—

5. Best Practices pro mockování

- **Mockujte pouze rozhraní (Interfaces):** Rozhraní jsou pro mockování ideální. Mockování konkrétních tříd vyžaduje virtuální metody a je složitější.
- **Nepřemockujte (Over-mocking):** Pokud mockujete úplně všechno, testujete pouze to, že jste správně nastavili mocky, nikoliv reálnou logiku.
- **Dodržujte DIP:** Mockování je možné pouze tehdy, pokud vaše třídy přijímají závislosti skrze konstruktor (Dependency Injection).

Související články:

- Unit Testing a kvalita kódu
- xUnit.net Framework
- Principy SOLID (Dependency Inversion)

Tagy: programming dot-net testing mock moq nsubstitute csharp