Szoftvertesztelés ASP.NET MVC platformon

Szoftver Verifikáció és Validáció Referátum

Imre Gábor

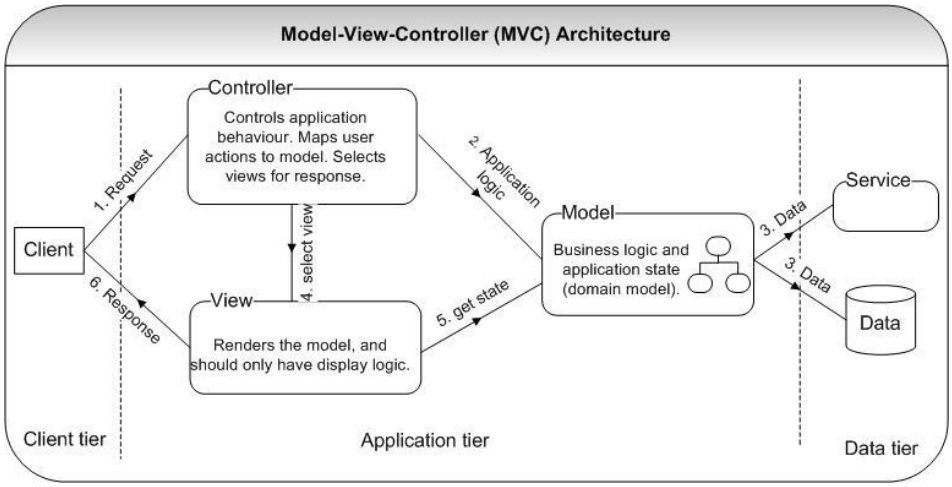
Áttekintés

Az ASP.NET MVC korunk egyik vezető webes szoftverfejlesztési környezete. A technológiára jellemző a rendkívül gyors fejlődés: gyakorlatilag évenként jön ki az újabb verzió, mindegyik jelentős továbbfejlesztéssel. A fejlesztést kézben tartó Microsoft meglepően együttműködő a fejlesztői közösséggel, számos kvázi sztenderddé vált közösségi komponenst emeltek be a hivatalos verziókba.[1] Az ASP.NET MVC keretrendszer fejlesztésében tehát központi szerepe van a fejlesztői közösségnek, ami biztosítja a használhatóság és hatékonyság irányába történő fejlődést.

Ebben a dolgozatban azt vizsgálom, hogy az ASP.NET MVC keretrendszer milyen szoftvertesztelési lehetőségeket támogat, miket tesz lehetővé, illetve melyek a fejlesztői közösségben elterjedt megoldások. Úgy vélem, ez jó képet ad a nem biztonságkritikus webes szoftverfejlesztés során is hatékonyan alkalmazható verifikációs megoldásokról.

Az ASP.NET MVC tesztelése

Az MVC tervezési minta a következő módon néz ki.[2]



A tesztelés az MVC keretrendszerhez illeszkedően az alábbiakra vonatkozhat.

- **Model.** A modell objektumok tárolják a felhasználónak megjelenítendő adatokat, valamint ebben a rétegben kerül implementálásra az üzleti logika (BL) és az adatbázis-elérés
  Verifikációs és validációs feladat: Az üzleti logika egyes elemeinek a funkcionalitása (Unit teszt), valamint együttműködésük (bottom-up Integration teszt)

[1] Ilyen a hivatalos (Microsoft fejlesztésű) JSON sorosítót felváltó Newtonsoft.Json csomag, valamint a lassan ipari sztenderddé váló Bootstrap, ami egy rendkívül népszerű CSS és JS alapot adó keretrendszer. Mindkettő a hivatalos kiadások részévé vált.

[2] A kép forrása: http://java-success.blogspot.hu/2011_10_01_archive.html, a tervezési mintáról bővebben: http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

- **Controller.** A felhasználói felület kéréseit kiszolgáló réteg
  Verifikációs és validációs feladat: A kontroller valóban továbbítja-e a kéréseket a BL felé (Unit teszt), illetve a teljes működés is validálható a kontroller vizsgálatával (Integration teszt)
- **View.** A felhasználó számára megjelenített eredmény, jelen esetben weblap.
  Verifikációs és validációs feladat: Bejárhatóság, linkek működőképessége (Tipikusan dinamikus bejárás)

Tervezési minták

Maga az MVC architektúra kiválóan alkalmas a fent vázolt tesztelésekre, hiszen a programkomponensek között eleve egy jól húzódó határ van. Emellett a következő tervezési minták segítik a tesztelést.

Dependency Injection

A Dependency Injection (DI) a szoftverkomponensek laza csatolását valósítja meg.[3]

A DI lényege a témában legnépszerűbb StackOverflow válasz alapján: *Dependency injection is basically providing the objects that an object needs (its dependencies) instead of having it construct them itself. It's a very useful technique for testing, since it allows dependencies to be mocked or stubbed out.*[4]

```
public class Samurai {
    public IWeapon Weapon { get; private set; }
    public Samurai(IWeapon weapon)
    {
        this.Weapon = weapon;
    }
}
public class WarriorModule : NinjectModule
{
    public override void Load()
    {
        this.Bind<IWeapon>().To<Sword>();
    }
}
```

Egy DI keretrendszer arra szolgál, hogy a megnevezett függőségeket automatikusan behelyettesítse. Ehhez természetesen szükséges alapfeltétel az interfészek megfelelő használata, de ezt programozási alapismeretnek tekintem.

Az egyik leggyakrabban használt DI keretrendszer a **Ninject**, a mellékelt példa az ő honlapjukról származik.[5] A példában szereplő Sword típus a tesztelés során lecserélhető egy WoodenStick-re, amennyiben az is az IWeapon leszármazottja. Így éles és teszt esetben ugyanúgy a keretrendszertől kérünk egy aktuális IWeapon objektumot, és azt adjuk át a Samurai konstruktorának.

A DI keretrendszerek rendszerint Reflectiont használnak (Javában is). A Reflection nem minden platformon elérhető, valamint teljesítménykritikus esetekben ellenjavallott a használata.

3A: Arrange, Act, Assert

A modultesztelés során használt egyik tipikus struktúra (tervezési minta) a 3A.[6] A legtöbb teszt felírható az alábbi módon, illetve annak minimális módosításával:

- **Arrange**: a teszt felállítása, vizsgált objektum létrehozása, paraméterek beállítása.
- **Act**: Vizsgálandó viselkedés végrehajtása, általában egy függvényhívás.

---

[3] A DI-ról részletesen: http://en.wikipedia.org/wiki/Dependency_injection
[4] http://stackoverflow.com/questions/130794/what-is-dependency-injection
[5] http://www.ninject.org, ez a csomag is a legelterjedtebb 3rd party csomagok egyike, a legnépszerűbb .NET-es DI könyvtár.
[6] 3A részletesen: http://xp123.com/articles/3a-arrange-act-assert/

- **Assert**: Konkrét vizsgálat: megtörténtek-e a megfelelő változások, jó-e a visszatérési érték (vagy megfelelő hibát kapunk-e), stb.

A 3A tervezési minta nem abszolút, de sokat segíthet annak az érdekében, hogy az amúgy is bőbeszédű és terjengős tesztek többé-kevésbé áttekinthetőek maradjanak.

GWT: Given, When, Then

A GWT tervezési minta lényegében megegyezik a 3A megközelítéssel, a különbség első sorban szemantikai. A TDD-ből ágazó Behaviour-Driven Development (BDD) módszer kifejlesztésekor került előtérbe.[7]

A GWT mintát több tesztelési keretrendszer használja, pl. a Cucumber.[8]

TDD és beépített lehetőségek

A TDD fejlesztésről, Unit tesztelésről és mockingról az órai tananyagban és egyéb referátumokban részletesen volt szó.[9] Részletes bemutatására nem térek ki, de van egy nagyon fontos MVC-specifikus lehetőség, amelyeket kiemelnék: Egy MVC projekt létrehozásakor a Visual Studio automatikusan felkínálja a tesztprojekt létrehozását, mint választható lehetőséget.

A teszt projektben tesztelő osztályokat és tesztfüggvényeket lehet létrehozni a megfelelő attribútumok használatával. Egy TestClass attribútummal ellátott osztály összes TestMethod attribútumú publikus függvényét lefuttatja a keretrendszer, és ahol nem talált hibát, ott elfogadja a teszt futtatását. Az eredményeket az Assert osztály statikus függvényeivel ellenőrizhetjük.

A tesztek futtatásához szükség lehet egy keretrendszer felállítására, vagy éppen a tesztadatbázis kipucolására. Ezt a TestInitialize, TestCleanup, stb. függvényekkel szabályozhatjuk. A részletekkel kapcsolatban számos tutorial, útmutató és részletes kidolgozás található a neten.[10]

Tipikusan sok fejlesztőt foglalkoztató cégeknél használnak Continuous Integration szervereket,[11] ahol a kód minden egyes verzióján felposztolás előtt automatikusan lefutnak a tesztek. Sokszor egy dedikált tesztszerver végzi ezt a feladatot. Az Agile Programming szemlélet szerint ez napi több posztot jelenthet fejlesztőnkként, aminek a tesztelése nagy szoftver esetén jelentős időt vehet igénybe.

A beépített tesztelési keretrendszer mellett rendelkezésre áll az NUnit is.[12] Az NUnit a JUnit mintájára készített Unit teszt keretrendszer, a fejlesztői közösség mind az NUnit-ot, mind a beépített megoldást széles körben használja.

Unit Tesztek, Integration Tesztek

A leggyakrabban használt tesztelési módszer MVC keretrendszer alatt a Unit tesztek írása. Egy szoftver átfogó Unit teszteléséhez a következőkre van szükség:

- DI tervezési minta

---

[7] Rövid összefoglaló: http://martinfowler.com/bliki/GivenWhenThen.html
[8] Cucumber: http://cukes.info/
[9] Pl. Kövesdán Gábor dolgozata: https://inf.mit.bme.hu/sites/default/files/edu/doktori/szvv/referatum2013/A_Test-Driven_Development_a_unit_teszteles_es_a_mock_technika_%28Kovesdan_Gabor%29.pdf
[10] Egy remek Unit teszt kidolgozást az alábbi címen lehet átnézni: http://geekswithblogs.net/Frez/archive/2013/04/26/unit-testing-an-asp.net-mvc-4-controller-using-ms-test.aspx
[11] Continuous Integration: http://en.wikipedia.org/wiki/Continuous_integration
[12] NUnit: http://www.nunit.org/

- Opcionálisan DI keretrendszer (pl. Ninject)
- Javallott: Mocking keretrendszer választása (pl. Moq)
- Tesztelési keretrendszer választása (beépített, vagy NUnit)
- Tesztesetek részletes implementálása (3A használata)

A tesztesetek TDD módszertan esetén már a kódolás előtt elkészülnek. A tesztelés sokszor nem terjed ki minden részletre, csak a fontosabb kérdésekre.

Saját tapasztalatom az, hogy egy bizonyos komplexitás felett a Unit tesztek szorosan vett implementálása rendkívül időigényes és technikailag sem egyszerű. A mockolás kapcsán sokszor kellemetlen tapasztalataim voltak: az adatelérési réteg komplexebb lehetőségeinek kihasználása után a mockolás már nem triviális feladat. Hasonlóan a Controller rétegben komplex azonosítás esetén nem egyszerű a vizsgált osztály függőségektől való leválasztása.

Ezért azt találtam a legkényelmesebb megoldásnak, ha egy teszt adatbázison futtatom a teljes keretrendszert (*fake* adatok), és az üzleti logika moduljait kvázi Unit tesztelem, azaz megnézem, hogy az egyes részek a saját feladatukat (az alattuk lévő réteggel együtt – *bottom up*) helyesen végzik-e el. Ez garantálni fogja, hogy a program a motorháztető alatt helyesen működik, viszont a webes felületre (Controller és View rétegek) nem ad garanciát.

A hagyományos Unit tesztelés ugyanakkor működhet a Controller rétegben is.

Webes tesztelés

Létezik néhány keretrendszer a webes felületek vizsgálatára is, akkor arra is rendelkezésre állnak keretrendszerek, mint pl. a Selenium vagy Lightweight Test Automation Framework.[13]

Ezek a megoldások nem tűnnek igazán népszerűnek a fejlesztők között.

IntelliTrace

Az IntelliTrace[14] a Visual Studio komplexebb (fizetős) változatainak a beépített monitorozó eszköze. Inkább debug, mint tesztelési eszköz, mégis indokoltnak tartom itt is megemlíteni, mert számos lehetősége túlnyúlik a debug szférán, például használható a karbantartási fázis alatt is a szoftver monitorozására, pontos hibajelentések kinyerésére is.

Az IntelliTrace automatikusan figyeli a felhasznált keretrendszerek főbb eseményeit, pl. adatbázis-elérés (lazy loading is), felhasználói interakciók (gombnyomás), webes kérések küldése és fogadása, fájl I/O műveletek, stb. A Visual Studio eleve erős a Debug lehetőségek vonatkozásban (pl. automatikus változófigyelés hívási szintenként, több szál kezelése), de az IntelliTrace még sokat könnyít a hibajavításban.

Személy szerint abban látom a legnagyobb lehetőséget, hogy az IntelliTrace azokba a rétegekbe is belátást enged, amit a magas absztrakciós szoftvertervezés amúgy elrejt. Erre a legszebb példa az,

[13] Selenium: http://docs.seleniumhq.org/, LTAF: http://ltaf.codeplex.com/
[14] Hivatalos bemutatás: http://msdn.microsoft.com/en-us/magazine/ee336126.aspx

hogy LINQ használatával a fejlesztő mentesül az SQL kód megírása alól – viszont az adatbázis továbbra is SQL-en keresztül elérhető. Az IntelliTrace segítségével nagyon jól lehet követni, hogy mikor milyen SQL utasításokat küld a program az adatbázisnak.[15]

Loggolás

A monitorozó keretrendszerek közül az NLog és a Log4net keretrendszerek a leginkább elterjedtebbek, képességeik tekintetében nagyjából ugyanolyan szinten lehetnek. Egy találó összefoglalás az előbbi kettő és az Enterprise Logging kapcsán:[16]

> *My recommendation for our project is this:*
>
> - *Use a logging facade (e.g. Common.Logging, SimpleLoggingFacade) to avoid direct dependencies.*
> - *If we end up using Enterprise Library for other facilities, then use it for Logging, too.*
> - *If we end up using something with a dependency on Log4Net, use Log4Net.*
> - *If none of the above, use NLog. Which I'd prefer.*

Összefoglalás

Dolgozatomban törekedtem arra, hogy a legfontosabb és az iparban széles körben alkalmazott tesztelési technológiákat és eszközöket összefoglaljam. A bemutatott módszertanok következetes használatával nagyban lehet csökkenteni a hibák előfordulását, és nagyban hozzájárulhatnak a biztonságosabb és jobb minőségű szoftverek fejlesztéséhez.

Láthatjuk azt is, hogy a széles körben használt eszközök a verifikációs és validációs technológiák szűk körét fedik csak le. Ez érthető és elfogadható abban a környezetben, ahol a hibáknál sokkal fontosabb a határidő és a produktivitás. Viszont ebben a közegben is rengeteget jelent, ha a fejlesztők ismerik a rendelkezésre álló technológiákat, és amennyire rajtuk áll, törekednek a minőségi szoftverfejlesztésre.

A technológia fejlődése pedig rendkívül érdekes kutatási terület: a verifikációs és validációs eszközök terén, valamint a modellalapú szoftverfejlesztésben is jelentős fejlődés zajlott a közelmúltban és fog zajlani a jövőben is. Ez pedig számos új lehetőséget és kérdést fog előtérbe hozni.

---

[15] Ez amúgy az Y tervezési modell egyik hátulütője: nem mindig minden tökéletes az Y függőleges ágában, azaz az automatikus leképzésekben.

[16] http://stackoverflow.com/questions/710863/log4net-vs-nlog/2121341#2121341, a téma kapcsán a legnépszerűbb kérdésre a legnépszerűbb válasz.