Tesztelés

Jánosi-Rancz Katalin Tünde

Sapientia EMTE
tsuto@ms.sapientia.ro

Minőségbiztosítás

- **verifikáció** (verification) ellenőrzi, hogy a szoftvert a követelményeknek megfelelően valósították meg
  - formális vagy szintaktikus módszerekkel
- **validáció** (validation) ellenőrzi, hogy a szoftver megfelel-e a felhasználók elvárásainak, azaz jól specifikáltuk-e eredetileg a követelményeket
  - teszteléssel

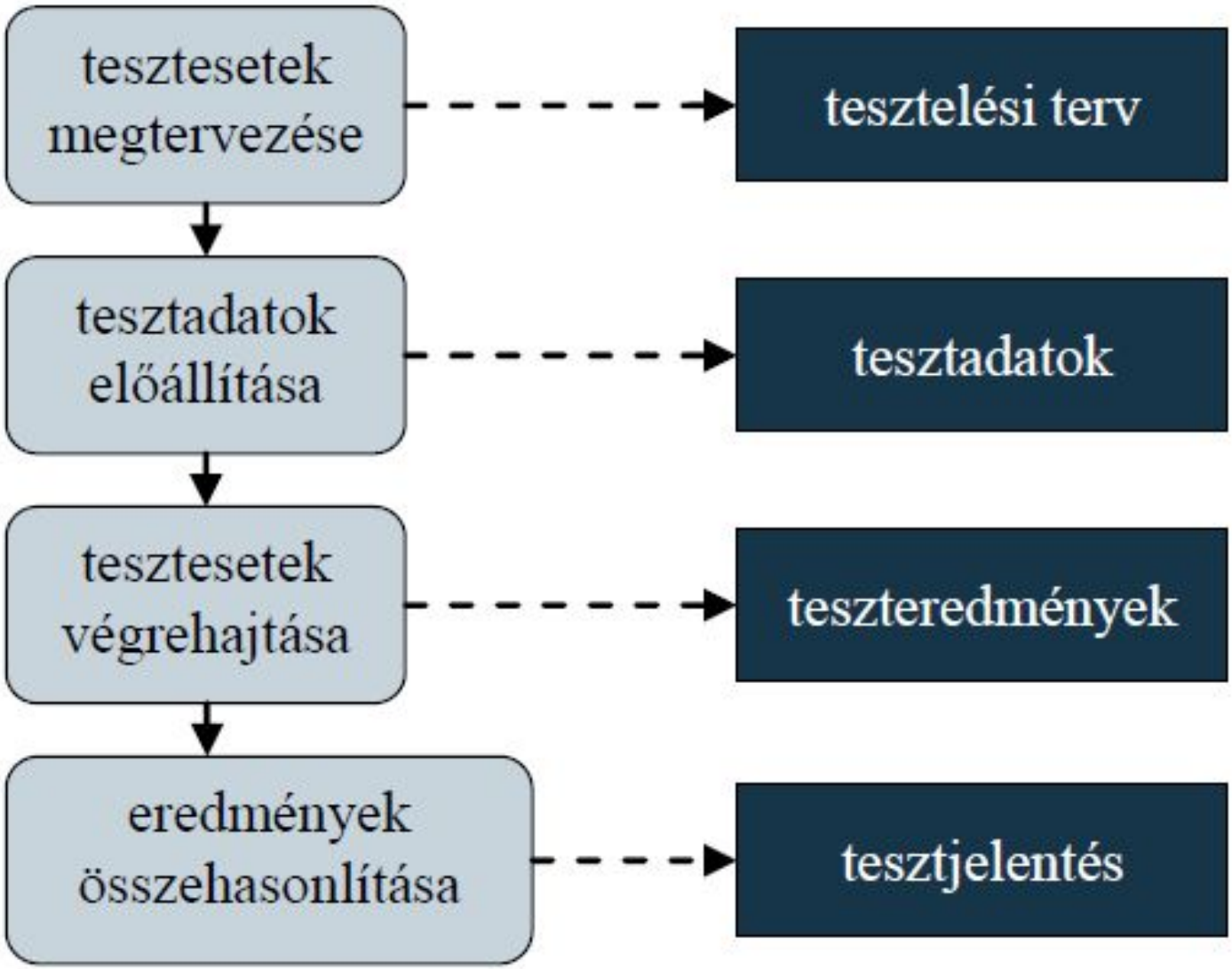
Tesztelés

- feladata: a szoftverben fellépő hibákat csökkentse, az ügyfél elégedettségét növelje és a meghatározott követelményeknek való megfelelőségét biztosítsa
- teszteléskor az adott programot hibakeresési szándékkal futtatjuk
- kapcsolatban áll a szoftver tervezésével, elkészítésével és kiértékelésével
- minden utólagos kódmódosítás csökkenti a kód koherenciáját, és gyengíti az architektúrát
- egy hiba kijavítása annál drágább, minél **később** derül rá fény

A tesztelés költsége

A hibajavítás költsége a hiba felfedezésének helye (ideje) függvényében		Hiba felfedezésének helye				
		Követelmények	*Tervezés*	*Implementáció*	*Tesztelés*	*Üzemeltetés*
Hiba helye	*Követelmények*	1x	3x	5-10x	10x	10-100x
	Tervezés		1x	10x	15x	25-100x
	Implementáció			1x	10x	10-25x

Tesztelési folyamat



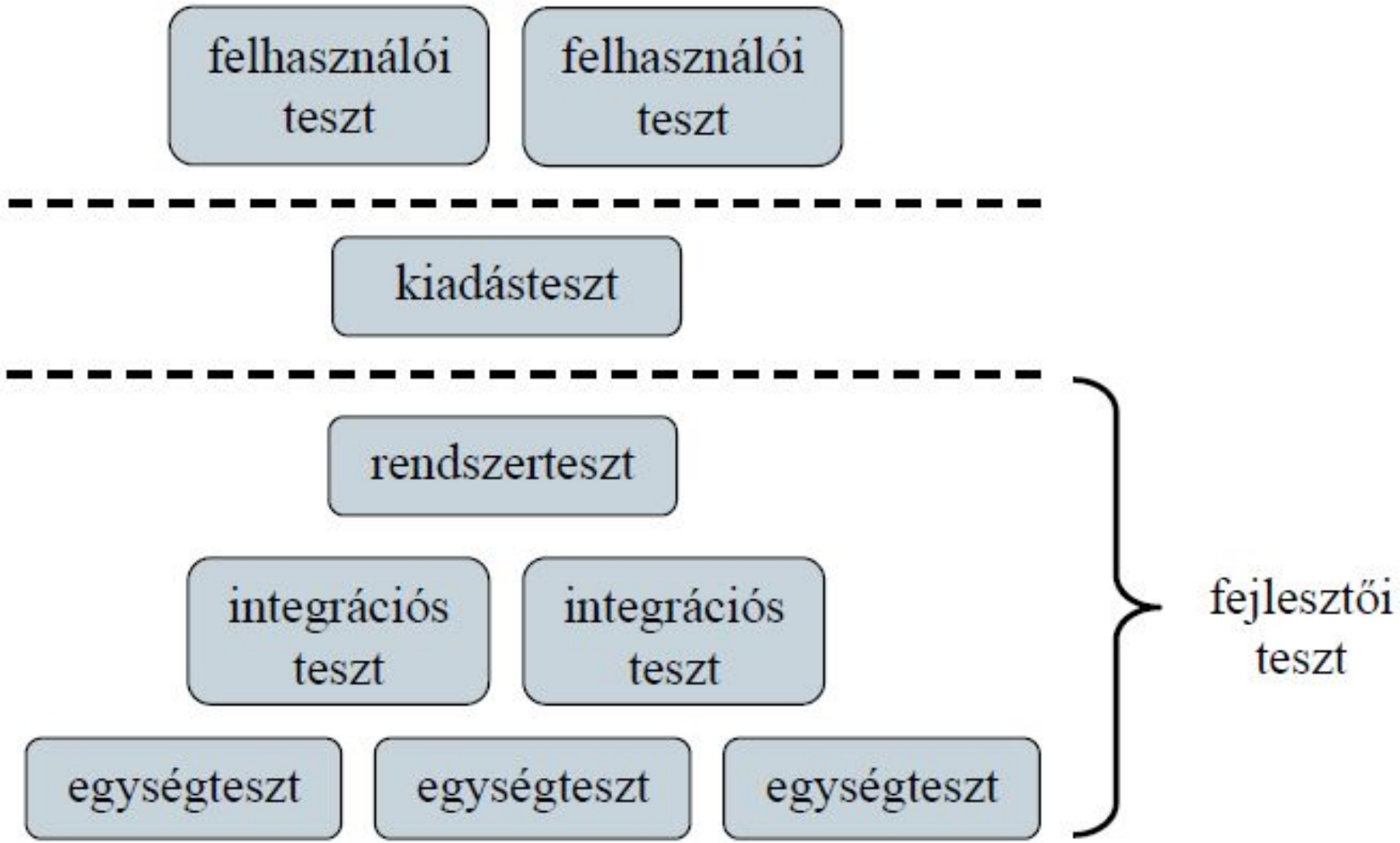
A tesztelés lépései

- nem a teljes program elkészülte után történik, hanem 3 szakaszból áll
  - **fejlesztői teszt** (development testing): a szoftver fejlesztői ellenőrzik a program működését
    - jellemzően White Box tesztek, a fejlesztő ismeri, követi a kódot
  - **kiadás teszt** (release testing) - egy külön tesztcsapat ellenőrzi
  - **felhasználói teszt** (acceptance testing) - a felhasználók tesztelik a programot a felhasználás környezetében
    - jellemzően Black Box tesztek

Fejlesztési teszt további szakaszai

- egység vagy komponensteszt
  - a rendszer legkisebb önállóan működő egységeit egymástól függetlenül teszteli (osztályok és azok metódusai)
- integrációs teszt
  - programegységeket, modulokat, azok egymással és a környezettel történő együttműködését teszteli (osztályok, esetleges adatelérés is)
- rendszerteszt
  - a megvalósított szoftver megfelel-e a kritériumoknak (a tesztautomatizálás a legnagyobb kihívás)
- átvételi teszt
  - az ügyfélnek bizonyít, hogy a szoftver megfelel az általa megfogalmazott elvárásoknak
  - a teszt magába foglalja a kihelyezést (telepítést)

A tesztelés lépései



Stubbing és Mocking

- Stubbing technika
  - a függőségek viselkedésének kontrollálhatóságát teszi lehetővé
  - működése: az adott függőség interfészét megvalósító saját típust készítünk, és az eredeti megvalósítást lecseréljük az általunk készített stub példányra
- Mocking keretrendszerek
  - a stubbing technikával újra és újra megvalósított tesztelési részfeladatokat fogja össze és ad egy egységes keretet a tesztesetek megvalósításához
  - elvárásokat rögzít adott metódushívásokra, és definiálja, hogy bizonyos paraméterek esetén milyen eredményt adjuk vissza az adott metódus
  - viselkedéssel kapcsolatos elvárásokat is rögzíthetünk (meghívódott-e a metódus bizonyos paraméterekkel, hányszor hívódott meg)

Kód lefedettség

A tesztgyűjtemények által letesztelt programkód mértéke

- a 70 százaléknál > jó
- számos szempont szerint mérhető
  - alprogram (function) - mely alprogramok lettek végrehajtva
  - utasítás (statement)- mely utasítások lettek végrehajtva
  - elágazás (branch) - az elágazások mely ágai futottak le
  - feltételek (condition) - a logikai kifejezések mely részei lettek kiértékelve

Teszttípusok

- Alfa-teszt, Béta-teszt
- teljesítmény teszt (válaszidő, sebesség, viselkedés terhelés mellett)
  - terheléses teszt (konkurens felhasználószám, tranzakciószám, ki/be jelentkezés)
  - stresszteszt (addig növeljük a terhelést, míg a rendszer összeomlik)
  - kitartási teszt (egy jól meghatározott terhelés alatt tartsa a rendszert hosszú időn keresztül)
  - csúcsteszt (normál terhelés után, hirtelen nagy terheléssel árasztjuk el)
  - mennyiségi teszt (nagy adatbázison, nagy rekordszám mellett vizsgáljuk)
- használhatósági teszt (mennyire egyszerű, kényelmes)
- biztonsági teszt (titoktartás, sebezhetőségi vizsgálat, behatolási teszt)
- skálázhatósági teszt (mennyire képes a növekvő igényeknek megfelelni)
- visszaállíthatósági teszt (képes-e helyreállni hibából)
- hordozhatósági teszt (op. rendszer, böngésző...
- linearitás vizsgálat (ha a paramétereik száma/értéke lineárisan növekszik akkor a futás idő is lineárisan változzon)

Tesztelési technikák

- Statikus technikák
  - nem igényelnek futtatást, alkalmazhatók dokumentációkon is
  - a forráskód ”helyesírása” elemezhető statikusan:
    - szintaktikai hibák
    - típushelyesség ellenőrzése
    - deklarálatlan/inicializálatlan változók
    - nem használt változók
    - elérhetetlen kód
- Dinamikus technikák
  - futtatást igényelnek
  - fekete doboz és fehér doboz technikák
  - feladata:
    - meghatározni az előfeltételeket
    - teszteseteket definiálni
    - meghatározni a tesztesetek végrehajtásának a módját

Black box és white box technikák

- Black box
  - a tesztelt rendszert egy átlátszatlan doboznak tekintjük, aminek csak a bemenetét és a kimenetét tudjuk kezelni, olvasni
- White box
  - magának a rendszernek és a komponenseknek a működésére koncentrál
  - **Unit test** – egységteszt

Unit test - Egységteszt

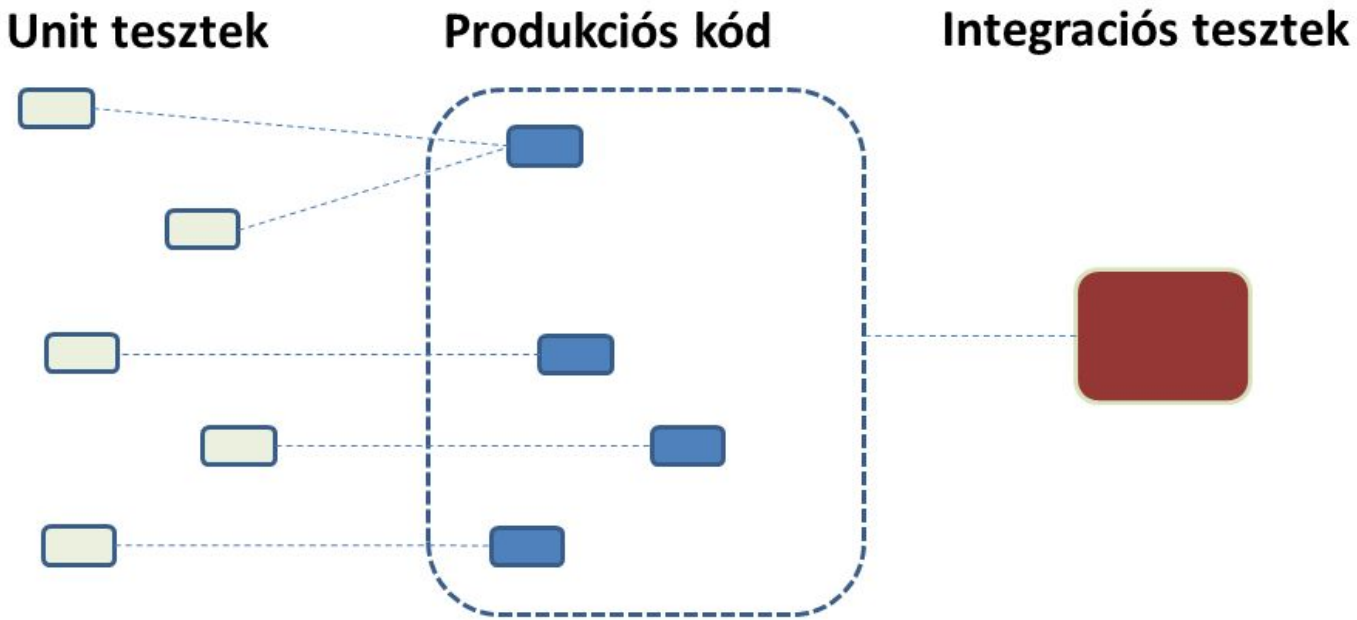
- moduláris kód (unit) (függvények, osztályok) egységenkénti tesztelése
- korai tesztelhetőség
- automatizált
- hatékony hibalokalizálás
- könnyű létrehozás és karbantartás
- egyetlen unithoz több teszt is tartozhat, de 1 teszt csak 1 unitot tesztel

Unit test - megszorítások

- nem minden egyes metódushoz kell készíteni Unit tesztet, csak a komponensek fontosabb műveleteit kell tesztelni
- sosem kapcsolódunk AB-okhoz/fileokhoz tesztelés esetén (Mockupokat használjunk)
- sosem kommunikálunk hálózaton keresztül más gépekkel
- kevesebb mint 10 sec, ”Egységteszt az a teszt, ami gyors. Ha egy teszt lassú, akkor az nem egységteszt!” - Michael Feathers (agilis fejlesztés)
- NE az implementációt teszteljük hanem a viselkedést
  - NE azt: A osztály meghívja-e B osztály adott metódusát, és használja-e a C osztályt
  - azt: metódus helyesen van-e leimplementálva

Egységtesztek

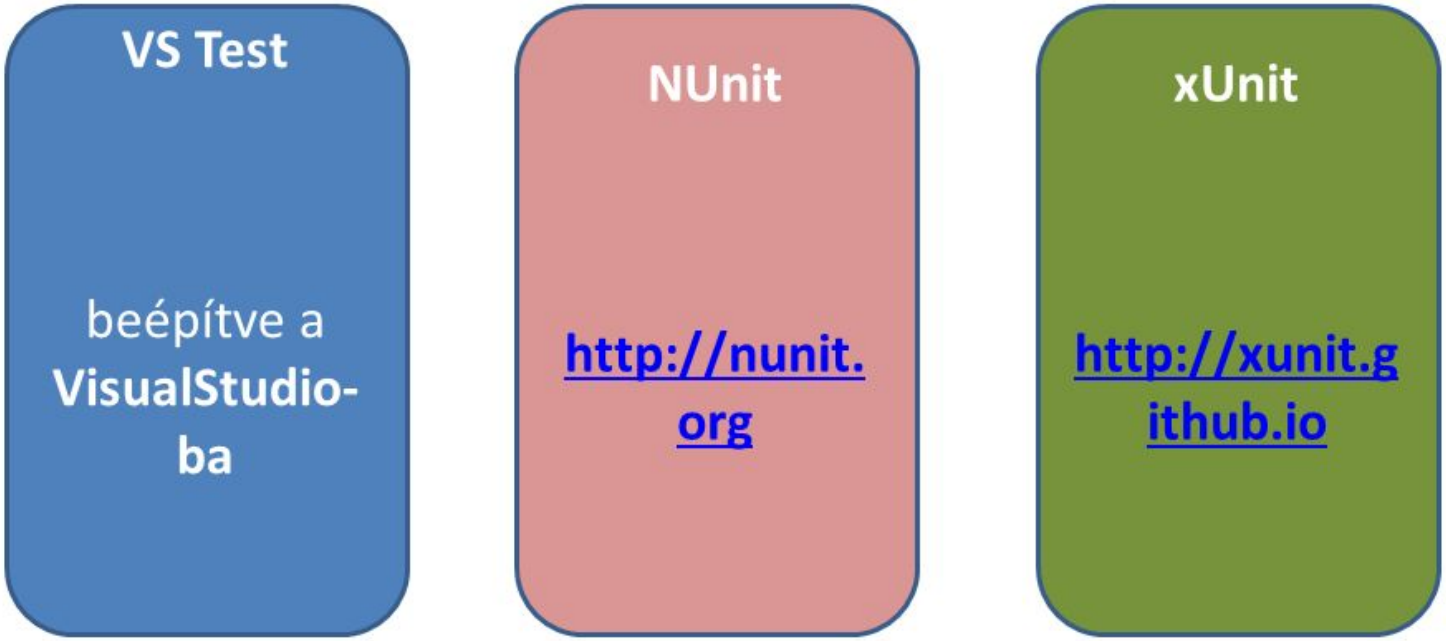
- előnye: nagy módosításokat írhatunk a kódba anélkül, hogy a funkcionalitás elromolna, figyelmeztet ha nem lett jó
- **F** – Fast
- **I** – Independent (of each other)
- **R** – Repeatable
- **S** – Self-validating
- **T** - Timely



Unit Test - Példa

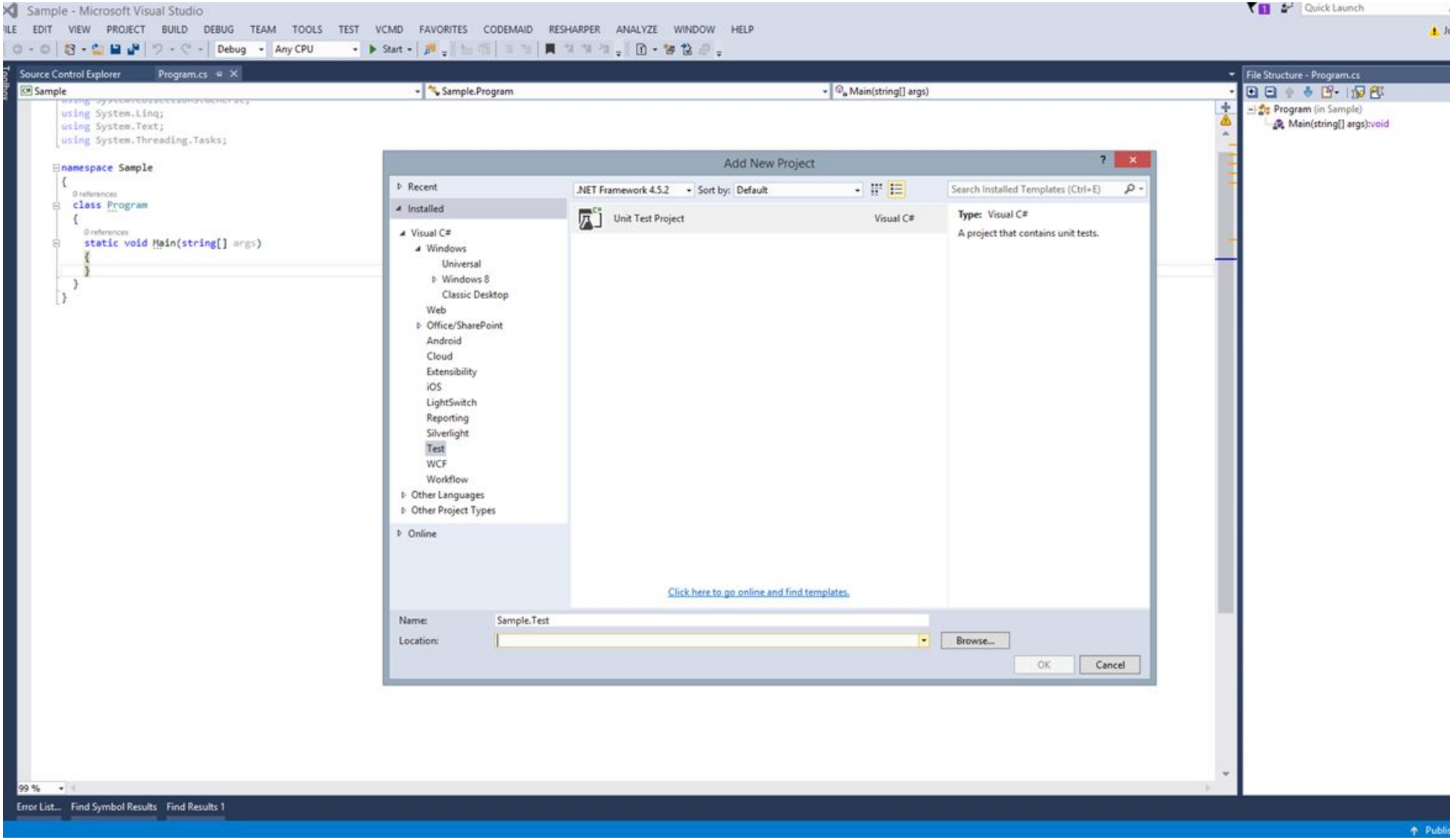
```
[Fact] vagy [TestMethod()]
public void AddShouldReturnSumofTwoInt()
{
//Arrange
var calc = new Calculator();

//Act
int result = calc.Add(2, 3);

//Assert
Assert.AreEqual(5, result);
}
```

Tesztelési .Net keretrendszerek

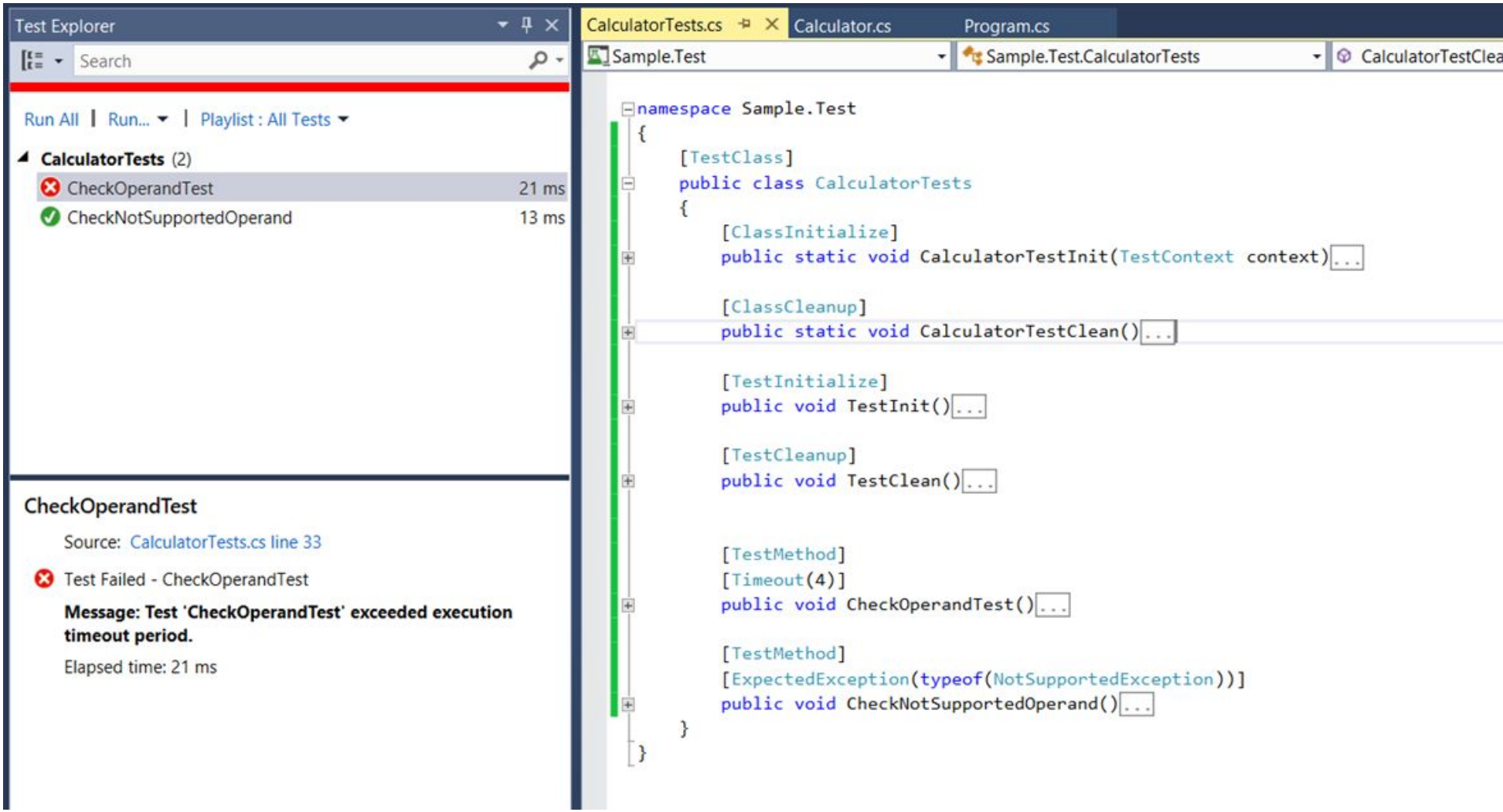


Más keretrendszerek: JUnit, JWalk, CppTest, QTestLib

Tesztelési .Net keretrendszerek



Tesztelési .Net keretrendszerek



Test Driven Development – Tesztvezérelt fejlesztés

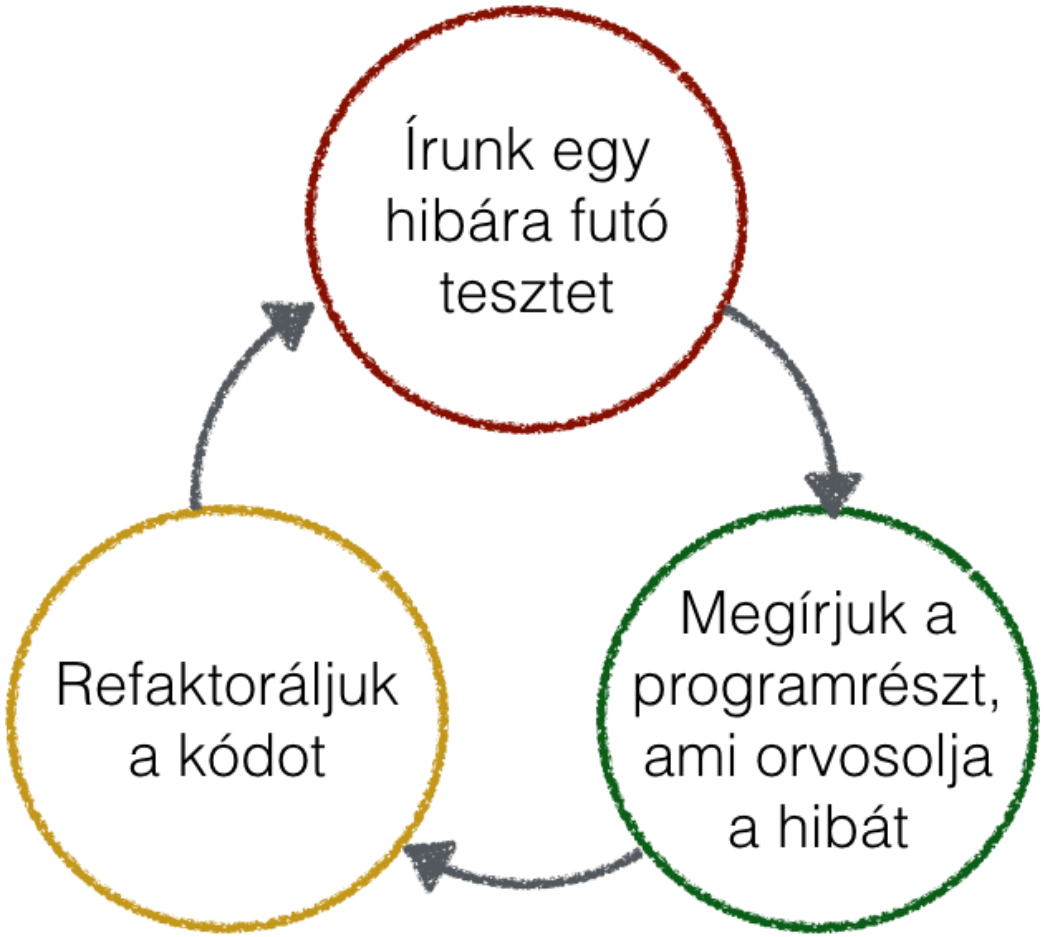
TDD - Tesztvezérelt fejlesztés

- egy olyan eszköz, ami folyamatosan jelzi a fejlesztői hibákat, azonnal ellenőrzi a program helyes működését, amint a programozó leír egy kódrészletet
- a fejlesztőnek kell megírni saját programjának tesztjeit, először a teszteket, majd utána a programkódot úgy, hogy az a teszteket kielégítse
- a jól megszokott ”utólag majd tesztelünk valamit” szemlélet többet már nem működik
- akár dokumentációként is szolgálhat, a teszt neve sugalja a funkcionalitást **AddShouldReturnSumofTwoInt**

A TDD három szabálya

- csak akkor írhatsz éles kódot, ha van egy teszt, amit ki szeretnél javítani
- csak annyi tesztkódot írhatsz, hogy pontosan egy hibát idézz elő
- legfeljebb annyi éles kódot írhatsz, hogy az éppen hibára futó teszt megjavuljon

A TDD fázisai



A TDD fázisai -2

- piros (sikertelen) fázis
  - írunk egy rövid tesztet (max 4-5 sor), ami egy alap funkcionalitást tesztel
  - hibát eredményez, mert nincs kód
  - futtassuk le az összes tesztet, bizonyosodjunk meg, hogy egyedül az utoljára megírt bukik el és az az elvárt módon lesz sikertelen
- zöld fázis (sikeres)
  - **minimális** implementáció írása, ami ahhoz szükséges, hogy a teszt átmenjen
  - ha ez pusztán annyi, hogy visszatérünk egy fix értékkel, akkor csak és kizárólag ennyi kódot írjunk
  - csak a teszt kizöldítésével foglalkozzunk
  - futtassuk le az összes tesztet, legyen minden rendben
- refaktorálás
  - a kód minőségének javítása
  - a kódot minél elegánsabbá és olvashatóbbá tegyük
  - futtassuk le a teljes tesztkészletet
- ismétlés
  - egy újabb elbukó tesztet írunk, addig folytatjuk, amíg az elvárt viselkedést meg nem kapjuk

A TDD előnyei és következményei

- előnyei:
  - előre megvannak a tesztjeink és biztosak lehetünk, hogy mindegyik tesztnek megfelel a programunk
  - sebesség (némi gyakorlás után gyorsabb)
  - kevesebb megszakítás tesztelőtől
  - stabilitás (egyszer gondolkozunk utána kódolunk)
  - tesztek dokumentálják a kódot
- hátrányai:
  - hosszú teszt futás
  - alacsony lefedettség, hibázó tesztek
  - sokan azt hiszik ettől hibamentes lesz a program működése
  - több időt vesz igénybe, hogy elkészüljön az **első** működő verzió

Példák

- lásd Code/

(:

- A tesztelés legyen veletek!