Tesztelés tananyag jegyzete  
Forrás: <https://learn.microsoft.com/hu-hu/training/paths/visual-studio/>

Bevezető

# Tesztelés lehetséges okai

A tesztelés alapvető célja annak ellenőrzése, hogy a kódmódosítások nem szegik meg a meglévő funkciókat, és biztosítják, hogy az újonnan hozzáadott kód a jövőben is a várt módon működjön.  
A tesztelés segít szüneteltetni és mérlegelni, hogy ugyanazt a logikát többször ismételi-e meg, és használhat helyette egy metódust. Lehetővé teszi a fejlesztők számára a kód újrastrukturálását a lehető legjobb újrafelhasználhatóság és stabilitás érdekében  
Ajánlott teszteket futtatni a fejlesztési ciklus főbb lépései előtt  
A fejlesztési ciklus korai szakaszában elkapott hibák időt és pénzt takaríthatnak meg.

* **A kódminőség javítása**
* **Társbeszélgetés**
* **Változó követelmények:** annak ellenőrzésére, hogy a növekvő követelményeknek továbbra is megfelel-e a szoftver.

# A tesztelési folyamat

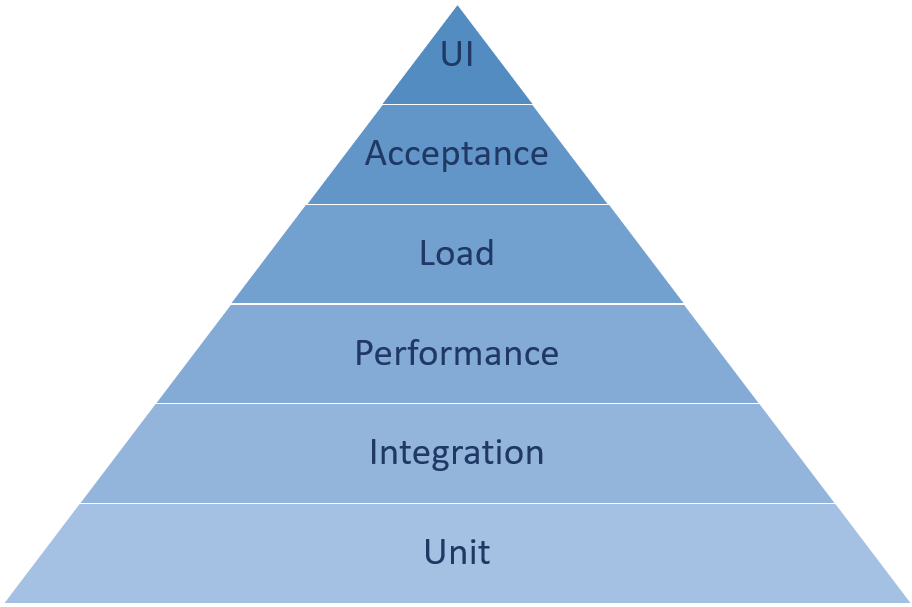
A tesztelés tesztmetődusokból áll, amin vagy átmegy az adott rész vagy begbukik. Ez egy példa egy tesztmetódusra, amely a calculator összeadás funkcionalitását teszteli egy másik (nem teszt) programból.  
  
[TestMethod]  
public void AddTest()  
{ var calculator = new Calculator();  
 var actual = calculator.Add(1, 1);  
 Assert.AreEqual(2, actual); }  
  
Mivel a tesztkód nem fut „magától”, a kód futtatásánál külön kell elindítanunk, például a *Visual Studio Test Explorer* használatával.

# Kódlefedettség

A kódlefedettség egy olyan metrika, amely azt jelzi, hogy az alkalmazás éles kódjának mekkora részét fedik le a tesztek. *A Visual Studio* olyan eszközökkel is rendelkezik, amelyekkel kiemelheti[, hogy mely vonalakat fedik le a tesztek](https://learn.microsoft.com/hu-hu/visualstudio/test/using-code-coverage-to-determine-how-much-code-is-being-tested),

# A tesztelési piramis

Ahogy felmászik a piramisra, a tesztek általában drágábbak lesznek az íráshoz, a karbantartáshoz és a futtatáshoz. A felhasználók például általában több egységteszttel rendelkeznek, mint integrációs tesztekkel, és több integrációs teszttel, mint a felhasználói felületi tesztekkel.



# A piramis szintjei – a tesztek típusai

## Egységtesztek (Unit tesztek)

Ezeknek egy metódusra vagy a legkisebb tesztelhető kódegységre kell vonatkozniuk. Ezek a tesztek a legegyszerűbben írhatók.

## Integrációs tesztek (Integration)

Ezek már több metódust vagy szélesebb körű viselkedést fednek le. Az egységteszt és az integrációs teszt együtt összefoglalja, hogy az alkalmazás részei a várt módon működnek-e együtt.

## Teljesítménytesztek (Performance)

A teljesítménytesztek általában rögzítik azt az időt, amely a program bizonyos részeinek futtatásához szükséges.

## Terheléses tesztek (Load)

A terheléses tesztek segítenek megállapítani, hogy az alkalmazás lelassul-e, vagy kivételeket okoz-e nagy léptékű használat esetén. (Hogyan működik az alkalmazás, ha 1000 felhasználó használja?)

## „Acceptance” teszt

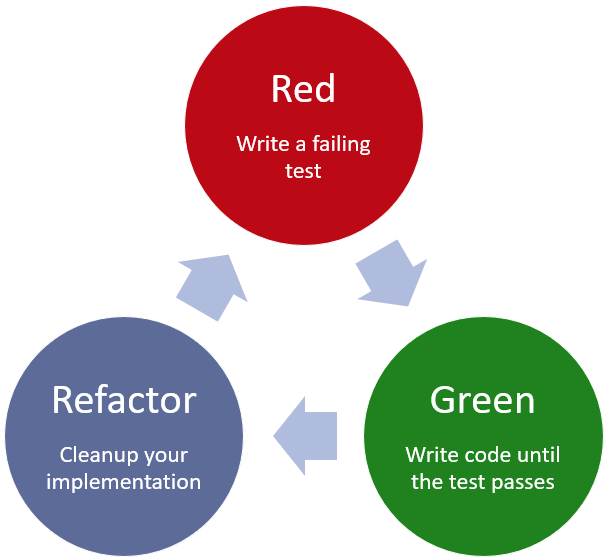
Az egy másik módja annak, hogy a specifikációt integrálják a kóddal.

## Felhasználói felületi tesztek (UI)

A felhasználói felületi tesztek közvetlenül tesztelhetik az alkalmazások felhasználói felületének interakcióit. A Playwright[például](https://playwright.dev/dotnet/)automatizálhatja a böngészőt a felhasználó utánzásához a webhely használatával. A felhasználói felület tesztelésének másik típusa a manuális tesztelés, amikor egy tényleges személy használja az alkalmazást

# Feljelsztési módszerek

## Tesztalapú fejlesztés – TDD (test driven development)

A tesztalapú fejlesztés( TDD) egy olyan módszer, amellyel közvetlenül összekapcsolhatja a tesztelést a napi fejlesztés minden részével. A TDD-t használó fejlesztők általában először megírnak egy sikertelen tesztet, majd megírják a tesztet végző kódot. Ez a megközelítés azt jelenti, hogy a termékkód nagyobb valószínűséggel kap magas tesztelési lefedettséget, és a tesztek általában az elsők.  
Létezik egy *Red/Green/Refactor* nevű TDD-típus.

## Viselkedésalapú fejlesztés - BDD (behaviour driven development)

Viselkedésalapú fejlesztés (vagy BDD) hasonló a TDD-hez, de nagyobb hangsúlyt fektet az acceptance tesztek használatára.

## Ne ismételd önmagad – DRY (Don’t repeat yourself)

Ne ismételd meg magad, vagyis DRY egy másik gyakorlat a tesztelés területén. A DRY azt állítja, hogy kerülni kell az információ és logika ismétlődését mindenhol, ahol csak lehetséges.