**A TDD szépségei**

**Mi az a TDD?**

A TDD (Test-Driven development) tesztvezérelt fejlesztést jelent. Ez egy [szoftverfejlesztési folyamat](https://hu.wikipedia.org/wiki/Szoftverfejleszt%C3%A9si_folyamat), ami egy nagyon rövid fejlesztési ciklus ismételgetésén alapul, tehát a követelményeket speciális teszt esetekként fogalmazzuk meg, a kódot pedig ehhez mérten írjuk meg, így az át fog menni a teszten. Ez tökéletes ellentéte a hagyományos szoftverfejlesztésnek, ami megengedi olyan kódrészletek meglétét is, amelyek nem felelnek meg teljesen a követelményeknek.

Ha a módszert egy névhez kell kötnünk akkor Kent Beck amerikai szoftvermérnök kell, hogy eszünkbe jusson. Ő fejlesztette ki a technikát vagy úgymond „újra-felfedezte” azt. 2003-ban úgy fogalmazott, hogy a tesztvezérelt fejlesztés ösztönzi az egyszerű tervezési elemek használatát, valamint hatással van az önbizalom növekedésére.

A TDD összefüggésben áll az [extrém programozás](https://hu.wikipedia.org/wiki/Extr%C3%A9m_programoz%C3%A1s) koncepciójával, miszerint először teszteljünk és ha minél többet tesztelünk, annál több hibát tudunk kiküszöbölni a kódban.Az utóbbi 1999-ben indult útjára, a TDD-nek pedig az idő előrehaladtával kialakultak sajátos megoldásai.

Rengeteg programozó alkalmazza ezt a technikát arra is, hogy egyszerűbbé tegyék az olyan [legacy kódok](https://hu.wikipedia.org/w/index.php?title=Legacy_k%C3%B3d&action=edit&redlink=1" \o "Legacy kód (a lap nem létezik)) fejlesztését, hibakövetését, amik régebbi eszközök felhasználásával készültek.

**Tesztvezérelt fejleztési ciklus**

**1. Teszt hozzáadása**

A tesztvezérelt fejlesztés során minden új funkció teszt írásával kezdődik. Olyan tesztet kell írni, amely tömören és pontosan meghatározza az új funkciót vagy a már meglévő, javításra szánt funkciót. Teszt írásához a fejlesztőnek egyértelműen meg kell értenie a szolgáltatás specifikációját és követelményeit. A fejlesztő [felhasználási esetek](https://hu.wikipedia.org/w/index.php?title=Haszn%C3%A1l%C3%A1si_eset&action=edit&redlink=1) és a [felhasználóval történt beszélgetések](https://hu.wikipedia.org/wiki/Felhaszn%C3%A1l%C3%B3i_t%C3%B6rt%C3%A9net) alapján pontosan lefedheti a követelményeket és az elvárásokat, így képes lesz a teszteket megírni, függetlenül a tesztelési és fejlesztői környezettől. Mindez alkalmazható már meglévő tesztek módosítására is. Ez a tulajdonság különbözteti meg igazán a tesztvezérelt fejlesztést az olyan unit tesztek írásától, amelyek már a kód megírása után készültek. Apró, de fontos eltérés ez.

**2. Minden korábbi teszt futtatása, hogy kiderüljön az új teszt megbukik e**

Ez megmutatja, hogy a tesztkihasználtság jól működik, az új teszt új kód igénylése nélkül nem teljesít sikeresen, mert még nem létezik a kívánt viselkedés, és ez megfelelően kiküszöböli annak a lehetőségét, hogy az új teszt selejtes és minden esetben átmegy a teszten. Az új tesztnek kudarcot kell vallania a várt eredmények alapján. Ez a lépés növeli a fejlesztő magabiztosságát az új tesztek megírásával kapcsolatban.

**3. Kód írás**

A következő lépés egy olyan kód írása, amely alapján a teszt sikeres lesz. Az ebben a szakaszban írt új kód nem lesz tökéletes, és valószínűleg egy nem túl elegáns módon fogja teljesíteni a tesztet. Ez azért elfogadható, mert az 5. lépésben úgyis csiszolni, szépíteni fogjuk a kódot. Ezen a ponton az írott kód egyetlen célja a teszt teljesítése. A programozó nem írhat olyan kódot, amely meghaladja a teszt által ellenőrzött funkciókat.

**4. Minden teszt újbóli futtatása**

Ha az összes tesztelési eset működik, a programozó biztos lehet benne, hogy az új kód megfelel a tesztelési követelményeknek, és nem ront a meglévő szolgáltatások minőségén. Ha mégis hibát okoz, akkor az új kódot addig kell módosítani, amíg meg nem felel az elvárásoknak.

**5. Kódszépítés**

A növekvő kódot rendszeresen meg kell [tisztítani](https://hu.wikipedia.org/w/index.php?title=K%C3%B3d_refaktor%C3%A1l%C3%A1s&action=edit&redlink=1) a tesztvezérelt fejlesztés során. Az új kód áthelyezhető onnan, ahol alkalmas volt a tesztelésre, oda, ahova logikusan tartozik. A [másolatot](https://hu.wikipedia.org/w/index.php?title=M%C3%A1solatk%C3%B3d&action=edit&redlink=1) el kell távolítani. [Az objektum-](https://hu.wikipedia.org/wiki/Objektum), [osztály-](https://hu.wikipedia.org/wiki/Oszt%C3%A1ly), [modul-](https://hu.wikipedia.org/w/index.php?title=Modul%C3%A1ris_programoz%C3%A1s&action=edit&redlink=1), [változó-](https://hu.wikipedia.org/w/index.php?title=V%C3%A1ltoz%C3%B3_(informatika)&action=edit&redlink=1) és [metódusneveknek](https://hu.wikipedia.org/w/index.php?title=M%C3%B3dszer_(sz%C3%A1m%C3%ADt%C3%B3g%C3%A9pes_programoz%C3%A1s)&action=edit&redlink=1) egyértelműen tükrözniük kell jelenlegi céljaikat és felhasználás mikéntjét, hogy az extra funkciók gond nélkül hozzáadhatóak legyenek. A szolgáltatások hozzáadásával a metódusok testei hosszabbá válhatnak, és más objektumok nagyobbak lehetnek. Előnyünkre válhat az ilyen részek feldarabolása az [olvashatóság](https://hu.wikipedia.org/wiki/Sz%C3%A1m%C3%ADt%C3%B3g%C3%A9p-programoz%C3%A1s) és a [karbantarthatóság](https://hu.wikipedia.org/w/index.php?title=Szoftver_karbantart%C3%A1s&action=edit&redlink=1) javításának érdekében, ami értékes lépésnek fog bizonyulni a [szoftver későbbi életciklusában](https://hu.wikipedia.org/w/index.php?title=A_szoftver_%C3%A9letciklusa&action=edit&redlink=1) . [Az öröklési hierarchiákat](https://hu.wikipedia.org/w/index.php?title=%C3%96r%C3%B6kl%C3%A9s_(objektum-orient%C3%A1lt_programoz%C3%A1s)&action=edit&redlink=1) át lehet rendezni, hogy logikusabbak és hasznosabbak legyenek, és hogy előnyükre váljanak a különböző [tervezési mintáknak](https://hu.wikipedia.org/wiki/Programtervez%C3%A9si_minta). Ismeretesek speciális és általános iránymutatások a refaktoráláshoz és a tiszta kód létrehozásához.[[6]](https://hu.wikipedia.org/wiki/Tesztvez%C3%A9relt_fejleszt%C3%A9s#cite_note-6) [[7]](https://hu.wikipedia.org/wiki/Tesztvez%C3%A9relt_fejleszt%C3%A9s#cite_note-7) A teszt esetek folyamatos újraindításával az egyes refaktorozási szakaszok során a fejlesztő biztos lehet benne, hogy a folyamat nem változtat meg meglévő funkciókat.

A sokszorosítás megszüntetése minden szoftvertervezési minta fontos aspektusa. Ebben az esetben azonban ez vonatkozik a duplikációs kód eltávolítására is a teszt kód és a végleges kód között - mint ahogyan azt említettük is a 3. lépésben.

**Lépések ismétlése**

Új teszt írásával a ciklus újra fog indulni. A lépések méretének mindig kicsinek kell lennie, mindegyik teszt futtatása között csak körülbelül 1-10 szerkesztéssel. Ha az új kód nem elégíti ki gyorsan az új tesztet, vagy ha más tesztek váratlanul sikertelennek bizonyulnak, akkor a programozónak [vissza](https://hu.wikipedia.org/wiki/Undo) kell vonnia vagy vissza kell állítania a kódot szemben a túlzott [hibakereséssel](https://hu.wikipedia.org/w/index.php?title=Hibakeres%C3%A9s&action=edit&redlink=1). A [folyamatos integráció](https://hu.wikipedia.org/wiki/Folyamatos_integr%C3%A1ci%C3%B3) visszafordítható ellenőrzőpontok állításával segít ezen a problémán. Külső könyvtárak használatakor fontos, hogy ne dolgozzunk olyan kicsi lépésekben, amikkel effektíven teszteljük csupán csak magát a könyvtárat,[[4]](https://hu.wikipedia.org/wiki/Tesztvez%C3%A9relt_fejleszt%C3%A9s#cite_note-Newkirk-4) kivéve, ha okkal feltételezhető, hogy a könyvtár hibás vagy nem elégíti ki a szolgáltatásokat teljes körűen.