

# Szoftverfejlesztési módszertanok

---

## 1. Életciklus

---

Minden szoftver rendelkezik életciklussal, amely meghatározza létét a feladat kitűzésétől a program használatának befejeztéig

Az életciklus általában négy fő fázisra bontható:

1. specifikáció: a szoftver funkcionalitásának és megszorításainak megadása
2. tervezés és implementáció: a specifikációnak megfelelő szoftver előállítása
3. verifikáció és validáció: a szoftver ellenőrzése a specifikációnak történő megfelelésre
4. evolúció: a szoftver továbbfejlesztése a változó elvárásoknak megfelelően

## 2. Módszertanok

---

Amellett, hogy a szoftverfejlesztés betartja az életciklus fázisait, a folyamat lefolyása különféle módokon történhet, amiket szoftverfejlesztési módszereknek nevezünk

- klasszikus módszerek: vízésés, prototipizálás, inkrementális, iteratív, spirális, V-Model
- agilis módszerek: Scrum, Lean, Kanban, XP (extreme programming), RAD (rapid application development)
- speciális célú módszerek: BDD (behavior-driven development), TDD (test-driven development), FDD (feature-driven development)
- formális módszerek: B-módszer

Csoportok:

- terv-vezérelt (plan-driven): célja a rend fenntartása, a szoftver fejlesztése előzetes specifikáció és tervezés alapján történik, igyekszik garantálni a minőséget
- agilis: célja a változáshoz történő alkalmazkodás, az egyszerűség, így kevésbé garantálja a minőséget
- formális: garantálja a minőséget, az implementáció bizonyíthatóan helyes megoldását adja a specifikációnak

A szoftverfejlesztési modell határozza meg az életciklus egyes fázisai közötti kapcsolatot, időbeliséget

A gyakorlatban a fejlesztőcsapat és a feladat befolyásolja leginkább a választott módszert

- sokszor a különböző módszerek vegyítve jelennek meg

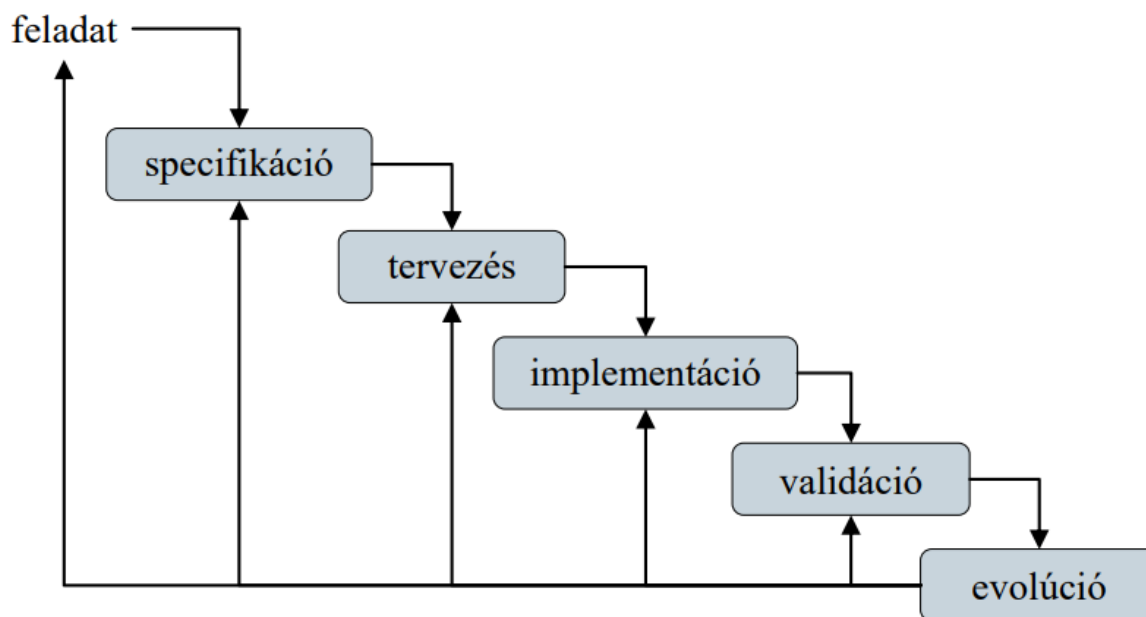
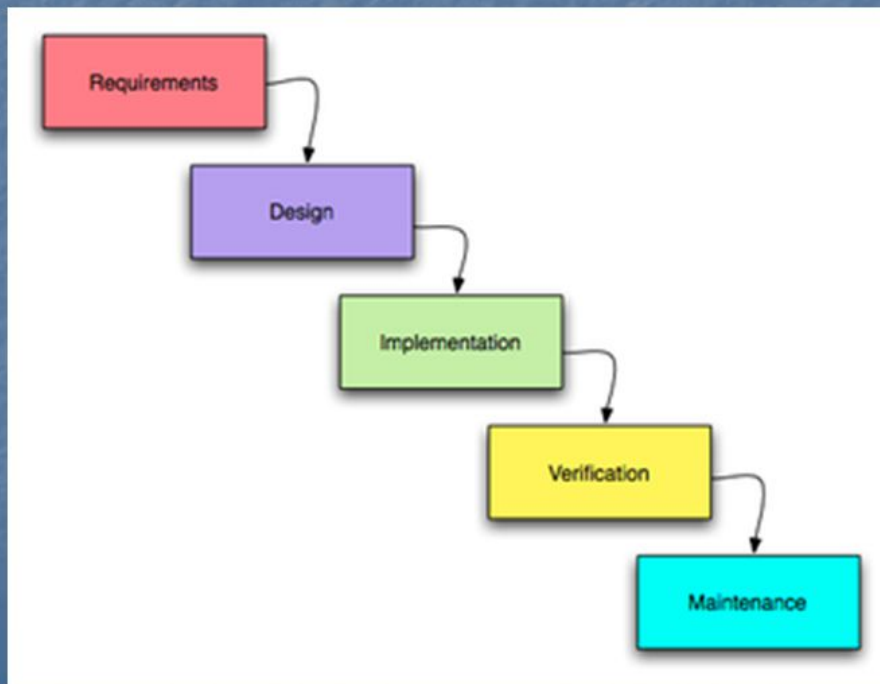
### 2.1. Vízésés modell

A legegyszerűbb fejlesztési modell a vízésés (waterfall) modell, amelyben az egyes fázisok lineárisan követik egymást

- előre megtervezi a projekt időtartamát, ráfordításait
- elvárja minden fázis megfelelő dokumentálását, amely tartalmazza annak eredményeit
- előnyei: jól strukturált, dokumentált folyamatot biztosít
- hátrányai: nem teszi lehetővé a követelmények megváltoztatását, nem készül fel az esetleges nehézségekre (nincs kockázatkezelés)

# A vízésés modell

A modell eredeti fázisai (Royce, 1970)



## 2.2. Prototípusok

A szoftverfejlesztés során felmerülő nehézségek könnyebben előreláthatóak, ha a szoftvernek elkészítjük a prototípusait (prototyping), amely lehet:

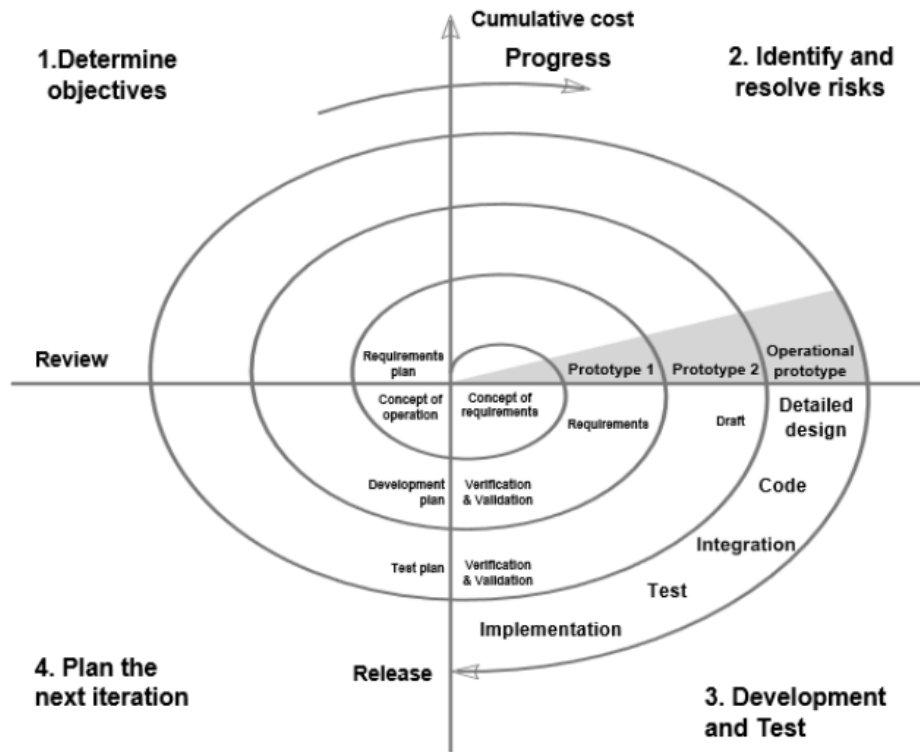
- horizontális prototípus: interakciós szempontból mutatja be szoftvert (pl. felhasználói felület)
- vertikális prototípus: egy adott funkció(csoport) részletes megvalósítása (pl. adatkezelés)

A folyamat során megvalósított prototípusok a szoftver részévé válhatnak (evolutionary prototyping), vagy szolgálhatják csak a bemutatást/ellenőrzést, és ténylegesen nem kerülnek felhasználásra (throwaway prototyping)

## 2.3. Spirális modell

A (Boehm-féle) spirális (spiral) modell egy kockázatvezérelt fejlesztési modell, amelyben a folyamat során elsőként prototípusok kerülnek megvalósításra, amelyek kiértékelése után kerül megvalósításra a tényleges szoftver

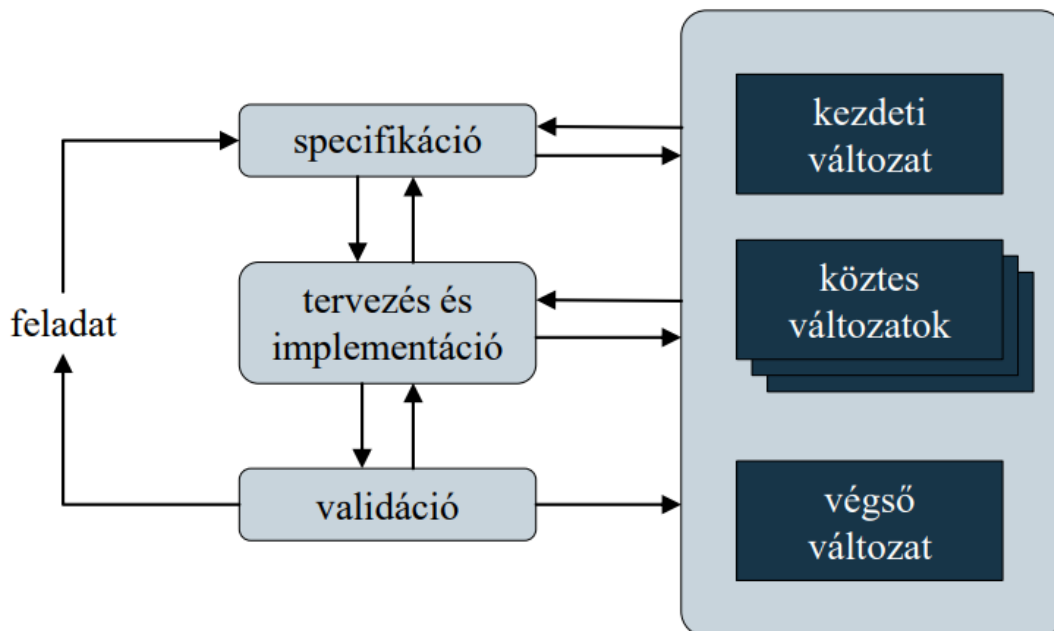
- a fejlesztés ciklusokban történik, amelyben az elkészített prototípusok, valamint a továbbfejlesztésével kapcsolatos kockázatok kiértékelésre kerülnek
- előnyei: jobban alkalmazkodik a változó követelményekhez, a prototípusok lehetővé teszik a nehézségek előrelátását
- hátrányai: költségesebb a prototípus elkészítése és a kockázatküértékelés végett, továbbá a prototípusok megzavarhatják a felhasználót



## 2.4. Inkrementális modell

Az inkrementális (incremental) modell több lépésből építi fel a folyamatot, és több változatban állítja elő a szoftvert

- minden változat egy újabb funkcionalitással bővíti a szoftvert, a fázisok rövidek, gyors visszajelzésekkel (a felhasználói oldalról)
- az egyes fázisok átfedésben vannak, és kihatnak egymásra
- előnyei: gyorsan alkalmazkodik a változó követelményekhez, a felhasználó jobban követheti a fejlesztési folyamatot
- hátrányai: kevésbé menedzselhető, tervezhető, áttekinthető, nehezebben validálható



## 2.5. Agilis szoftverfejlesztés

Az agilis szoftverfejlesztés (agile software development) célja a gyors alkalmazásfejlesztés megvalósítása, inkrementális alapon

- a szoftver folyamatos fejlesztés és kiadás alatt áll (continuous delivery), a sebesség állandó, a változtatások minden lépésben beépíthetőek (welcome changes)
- a működő szoftver az előrehaladás mérőeszköze, előtérben az egyszerűség, ugyanakkor folyamatos odafigyelés a megfelelő tervezésre, optimalizációra
- a fejlesztést általában önszervező, kis csapatok végzik, megosztott felelősséggel, folytonos interakcióval, gyors visszajelzésekkel

Az [Agilis Kiáltvány](#) a következőket mondja ki:

Azzal leplezzük le a szoftverfejlesztés jobb módjait, hogy csináljuk és segítünk másoknak is csinálni. Ezen a munkán keresztül következő értékekhez jutottunk el:

- Egyének és kölcsönhatások előnyben részesítése a folyamatok- és eszközökkel szemben
- Működő szoftver előnyben részesítése az átfogó dokumentációval szemben
- Ügyféllel való együttműködés előnyben részesítése a szerződéses megállapodással szemben
- Változásokra adandó válasz előnyben részesítése egy terv követésével szemben Habár a jobb oldali elemekben is van érték, mi sokkal értékesebbnek tartjuk a baloldali elemeket. (© 2001, Beck, K., et. al.)

Az agilis fejlesztés

- előnyei: jól alkalmazkodik a változtatásokhoz, hatékonyabbá teszi a fejlesztési folyamatot
- hátrányai: egyes tényezők nehezen megvalósíthatóak, különösen nagyobb szoftverek esetén a megvalósításhoz képzett fejlesztők kellenek, a dokumentáció hiánya megnehezíti a későbbi evolúciót Az agilis fejlesztés elveit több speciálisabb módszertan ültette át a gyakorlatba
- pl. Extreme programming, Scrum, Kanban, Lean
- a módszertanok nem szeparáltak, a gyakorlatban sokszor vegyítve alkalmazzák őket

### 2.5.1. Extreme programming

Az Extreme programming (XP) a gyors fejlesztési ciklusokra támaszkodik

- elvárja a követelmények viselkedés alapú felbontásával (BDD), a tesztek előre történő megadását (TDD), a folyamatos integrációt és refaktorálást
- támogatja a párban történő programozást

### 2.5.2. Lean

A Lean software development (LSD) az egyszerűséget, a felesleges tényezők kiküszöbölését hangsúlyozza

- kiküszöbölendő tényezők (TIMWOOD): munka és munkaerő átvitel, eszköztár túlhalmozás, várakoztatás, túlteljesítés, hibák
- támogatja a döntések elhalasztását, a folyamatos, globális áttekintést

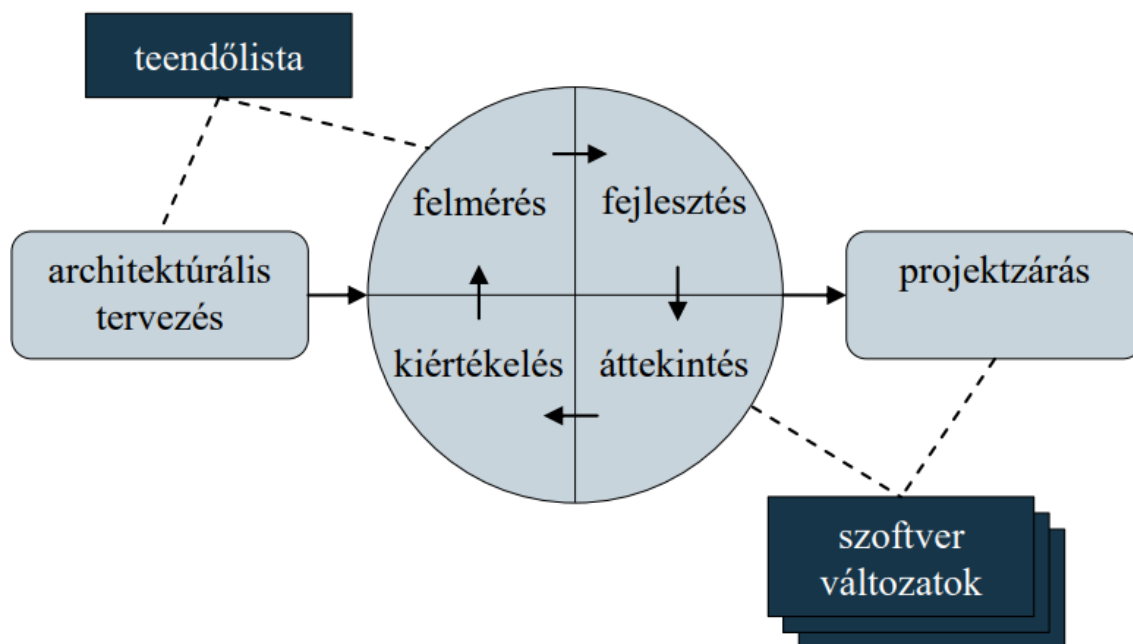
### 2.5.3. Scrum

Az agilis fejlesztés menedzselését az egyes változatok előállítására szempontjából közelítik meg, amelyhez a Scrum módszer ad egy általános modellt, amelynek fő lépései:

1. architektúrális tervezés, amely megadja a szoftver magas szintű vázát
2. futamok (sprint), amelyek az egyes változatokat állítják elő, és rögzített hosszúságúak (2-4 hét)
3. projektzárás, a szükséges dokumentáció előállítása Nincs projektmenedzser, de minden futamnak van felelőse (scrum master), akinek a személye futamonként változik

Minden futam egy összetett folyamat megtervezett lépésekkel

- feladatok felmérése (select), lefejlesztése (develop), áttekintése (review), kiértékelése (assess)
- a megvalósítandó funkciók a termékgazdával egyetértésben kerülnek kiválasztásra a teendők listájából (product backlog)
- naponta rövidebb megbeszélések (stand-up meeting) a teljes csapat számára
- ciklus elején/végén hosszabb megbeszélések (sprint planning, sprint review), valamint visszatekintés (retrospective) a termékgazdával



### 2.5.4. Kanban

A Kanban módszer célja a döntéshozatal elősegítése, és a fejlesztési folyamat gyors áttekintése

- eszköze a Kanban tábla (Kanban board), amely vizualizálja a fejlesztési feladatok állását
- elvei:
  - kiinduló állapot áttekintése, folyamatos, kísérleti továbbfejlesztés
  - fókuszban a felhasználói elvárások, folyamatos visszajelzés
  - egyéni döntéshozatal, önszervezés támogatása, tanulás és fejlődés
  - munkamennyiség és kiegészítő tevékenységek korlátozása

