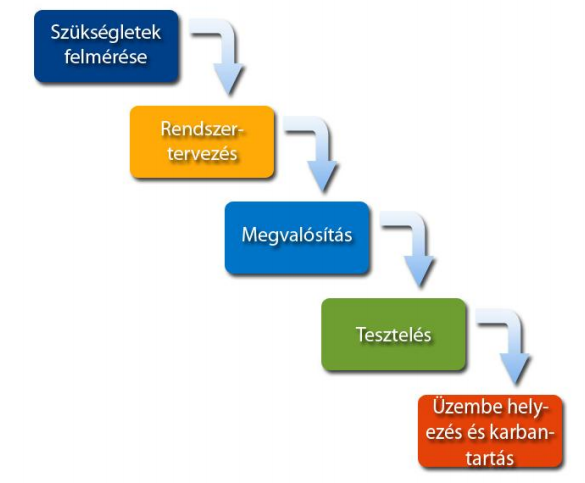
**Informatikai ismeretek**

**6. Tétel**

# Hagyományos módszertanok

## Vízesés modell

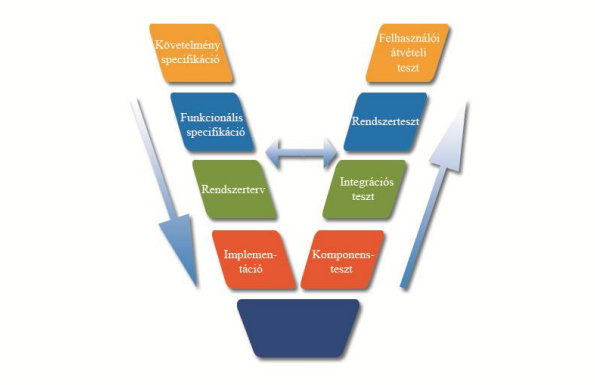


* Strukturális módszertan, vagyis az elvégzendő feladatot modulokra bontja.
* Ideális nagy megrendelők, nagy projektjeihez, melyek rugalmatlanok.
* Az életciklus lineáris, nincs lehetőség a követelmények módosítására, azok meghatározása után.
* Csak akkor lehet egy következő fázisba lépni, ha már az előzőt tökéletesen befejeztük.
* Részletes dokumentációt feltételez ez a módszertan, melyet a megrendelőnek is tüzetesen át kell olvasnia.

### Fázisok

1. szükségletek felmérése
2. rendszertervezés
3. megvalósítás
4. tesztelés
5. üzembe helyezés, karbantartás

## V-modell

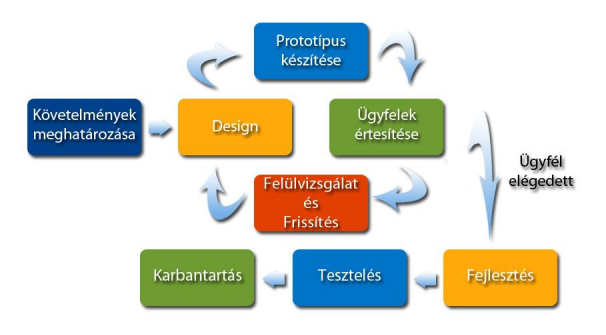


* A vízesés modell továbbfejlesztésének tekinthető.
* Az egyik szára megegyezik a vízesés modellel (fejlesztési szár), a másik pedig a létrejövő termékek tesztjeit tartalmazza (teszt szár).
* Az egy szinten lévő fejlesztési és tesztelési lépések összetartoznak. A teszt mindig az adott fejlesztési szinten lévő dokumentumot használja, vagy a terméket teszteli.
* Először a fejlesztést kell végrehajtani, és utána kell tesztelni.
* Ha a teszt hibát talál, akkor vele egy szinten lévő fejlesztési lépéshez kell visszatérni.

### Fázisok

1. **Követelmény specifikáció – Acceptance teszt:** Üzleti elemzők felmérik az igényeket és elkészítik a követelmény specifikációt. Jól meghatározott átvételi kritériumokat tartalmaz. Ez fogja az alapját képezni a felhasználói átvételi teszteknek. Fontos, hogy a követelmény specifikáció minden igényt kielégítsen, ugyanis nem lehet a fejlesztés során módosítani azt.
2. **Funkcionális specifikáció – Rendszerteszt:** Ebben határozzák meg, hogy hogyan kell működnie a programnak. Például, hogy egy gomb megnyomásakor mi történjen. Ez adja a rendszerteszt alapját.
3. **Rendszerterv –** **Integrációs teszt:** Leírja, hogy az egyes funkciókat milyen osztályokkal, metódusokkal tervezzük megvalósítani. A rendszerterv leírja azt is, hogy az egyes komponensek hogyan működnek együtt. Ez lesz az integrációs teszt alapja.
4. **Implementáció –** **Komponensteszt:** Az implementálás során minden metódushoz egy vagy több unit tesztet kell írni. A nagyobb egységeket, osztályokat pedig komponens tesztek alá kell vetni.

## Prototípus modell



* A vízesés modell sikertelenségére válaszol, mely abból eredt, hogy a felhasználók a csak a fejlesztés végén találkoztak a szoftverrel, így nem derülhetett ki időben, hogy a felek esetleg félreértették egymást.
* Cél: **A végső átadás előtt több prototípust kell leszállítani.**
* Sikere abban rejlik, hogy a fejlesztés során lehetőség van a változó követelményekhez alkalmazkodni.
* A legtöbb manapság elterjedt módszertan prototípus alapú.
* Akkor érdemes használni, ha a felhasználó és a fejlesztő között sok a kommunikáció.

### Fázisok

1. **Az alap követelmények meghatározása**: Itt bemeneti és kimeneti adatokon van a hangsúly, nem a teljesítményen vagy a biztonságon.
2. **Kezdeti prototípus elkészítése:** A felhasználói felület elkészítése, mögöttes funkciók nélkül.
3. **Bemutatás:** A felhasználó kipróbálja prototípust és javaslatokat, hogy mit kéne másképp csinálni.
4. **Követelmény pontosítása:** A felhasználói visszajelzések alapján pontosítjuk a követelmény specifikációt. Ha még nem elég pontos a specifikáció, akkor egy újabb prototípus készül, majd ugrunk a 3. lépésre. Ha már kész a követelmény specifikáció, akkor kezdődhet a fejlesztés.

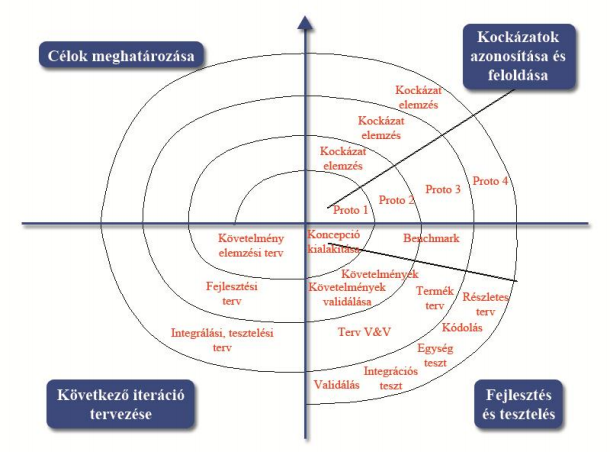
### Változatai

* **Eldobható prototípus:** A prototípus nem lesz része a végső kiadásnak, csak az elemzéshez használják. Előnye, hogy a felhasználó hamar képet kap arról, hogy mire számíthat a programtól, és gyorsan lehet változtatni követelményeken. **Fontos, hogy az első prototípus hamar elkészüljön, különben ennek a megoldásnak nincs értelme.**
  + **Felület kialakításához 2 módszert használnak:**
    - papiron lerajzolt felület
    - GUI szerkesztő programmal készített felület
* **Evolúciós prototípus:** Már a fejlesztés elején készül egy robosztus prototípus, mely az alapja lesz a végső rendszernek, későbbiekben csak finomítanak rajta. Előnye, hogy a felhasználó már az elején működő rendszert kap. A fejlesztők csak a rendszer bizonyos részeire koncentrálnak, melyek követelményei már teljesen érthetőek.

### Előnyök és hátrányok

* Előnyök:
  + **minőségnövelés**: Csökken a félreértések száma a követelmény specifikációban.
  + **költségcsökkentés:** Hamar kiderülnek a megrendelő igényei, így nem kell a fejlesztés késői szakaszában változtatásokat eszközölni.
  + **bevonja a felhasználót a fejlesztésbe:** Már a fejlesztés elején a fejlesztők kapnak visszajelzéseket a felhasználóktól, így a termék magasabb minőségi szintet érhet el.
* Hátrányok:
  + **Probléma az elemzés miatt:** Ha csak a prototípusra koncentrálnak a fejlesztők, akkor lehet, hogy nem találják meg a legjobb megoldásokat.
    - Rendszer teljesítménye gyenge lesz.
    - Nehezen lesz karbantartható.
    - Nehezen lesz skálázható.
  + **Prototípus összekeverése a végtermékkel:** A felhasználók rossz visszajelzést küldenek, mert azt hiszik, hogy a prototípusnak úgy kellene működnie, mint a végterméknek.
  + **Ragaszkodás a prototípushoz:** Előfordulhat, hogy a fejlesztők mindenképpen a prototípusból szeretnék a végső kiadást elkészíteni, még ha annak az architektúrája nem is megfelelő.

## Spirális fejlesztési modell

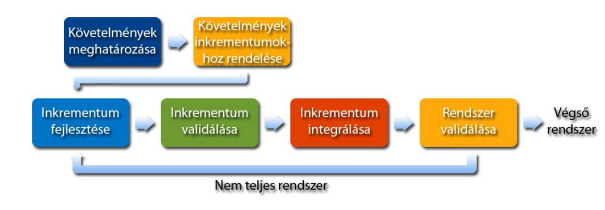


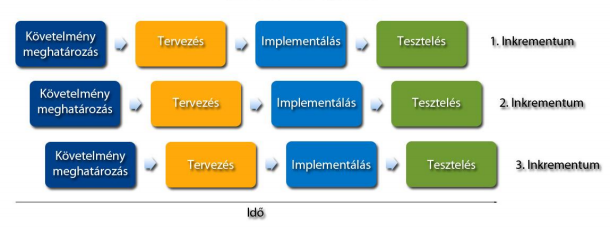
* A vízesés modell és a prototípus modell egyes részeit kombinálja.
* Első modell, melyben már van iteráció. (Prototípus esetén a ciklusok csak a követelmények felderítésére szolgáltak.)
* Minden ciklus végére egy működő prototípust kell előállítani.
* A prototípusoknak egyre közelebb kell lenniük a végtermékhez. (Evolúciós prototípus megközelítés)
* Nagyon fontos minden ciklusban a kockázat elemzés, hiszen lehet, hogy a megrendelő nem elégedett, és elölről kell kezdeni a ciklust.
* Az utolsó fázis nagyjából a vízesés modellel egyezik meg, hiszen itt már tisztázott a program elvárt kinézete és működése.
* Nagy, bonyolult rendszerek esetén ajánlott.

### Fázisok (Minden ciklusban)

1. Célok meghatározása
2. Kockázatok azonosítása és feloldása
3. Fejlesztés és tesztelés
4. Következő iteráció tervezése

## Iteratív és inkrementális módszertanok



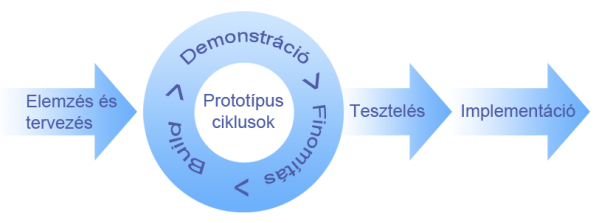


* A fejlesztést az igényfelméréstől az üzemeltetésig iterációkra bontjuk.
* Minden iterációban van tervezés és implementáció.
* Minden iteráció kiegészíti az előző iterációban elkészült prototípust. Ezt a kiegészítést nevezik **inkrementumnak.**
* Minden iteráció tartalmaz elemzést, tervezést, fejlesztést és tesztelést. Így félreértés esetén nem kell visszalépni, hanem egy következő iterációban meg lehet azt oldani.
* **Iteratív módszertan:** A hangsúlyt a folyamatra/iterációra helyezi.
* **Inkrementális módszertan:** A hangsúlyt az iterációk végtermékére helyezi. A mai módszertanok ebbe a családba tartoznak.

### Iteráció fázisai

1. **Üzleti folyamatok elemzése**: Meg kell ismerni a megrendelő üzleti folyamatait. A lemodellezett üzleti folyamattal kapcsolatban egyeztetni kell a megrendelővel, az esetleges félreértések elkerülése végett. (Általában üzleti elemzők végzik.)
2. **Követelményelemzés**: A rendszer funkcionális és nem funkcionális követelményeit kell meghatározni. Kezdetben ennek a célja a követelmények felállítása, későbbi iterációkban pedig csak a funkcionális terv finomítása.
3. **Elemzés és tervezés**: Követelményelemzés termékei alapján elemezni kell a rendszert és meg kell tervezni. Kezdetben a nemfunkcionális követelményeken van a hangsúly (architektúrális terv), későbbi iterációkban pedig a funkcionális követelményekkel kell foglalkozni, azaz az osztályok, metódusok és adattáblák megtervezésével.
4. **Implementáció**: A fejlesztők a fejlesztik a rendszert a tervek alapján, szoros kapcsolatban a tervezőkkel. Szokott lenni egy kódelemző, aki ellenőrzi, hogy a kódok megfelelnek-e a terveknek és a programozási irányelveknek. Ebben a szakaszban unit teszttel biztosítják a kód minőségét.
5. **Tesztelés**: A tesztelők komponens és integrációs teszteket hajtanak végre. Továbbá ellenőrzik, hogy a korábbi kódok nem romlottak-e el. (Regressziós teszt)
6. **Értékelés**: Itt dől el, hogy az iteráció végtermékét elfogadják-e vagy sem. Az értékelés része az átvételi tesztek végrehajtása is. Ha elfogadásra kerül, akkor lehet feltölteni a verziókövető rendszerbe a forrást.
7. **\*Támogató tevékenységek**: Az iterációktól függetlenül végrehajtott folyamatok. Pl.: a verziókövető rendszerben lévő forrás napi fordítása.

## Gyors alkalmazásfejlesztés (**R**apid **A**pplication **D**evelopment)



* Lényege, hogy a szoftvert gyorsabban és jobb minőségben készítjük el.
* Ehhez a következőek szükségesek:
  + Korai prototípuskészítés és ismétlődő felhasználói átvételi tesztek.
  + A csapat - megrendelő és a csapaton belüli kommunikációban kevésbé formális.
  + Szigorú ütemterv, így az újítások mindig csak a termék következő verziójában jelennek meg.
  + Követelmények összegyűjtése fókusz csoportok és munkaértekezletek használatával.
  + Komponensek újrahasznosítása.
* Több szoftvergyártó készített a fentiekhez segédeszközöket. Pl.: követelmény összegyűjtő eszközök, prototípus készítő eszközök stb.
* A RAD esetén az elemzés és a tesztelés között csak hat-hét hét telik el, ami csökkenti annak esélyét, hogy a követelmények megváltoznak a fejlesztés során.
* RAD esetén gyakori, hogy már jól bevált komponenseket használnak fel a fejlesztők, ami lehet saját vagy külsős fejlesztésű.
* Ebben az esetben is ciklusokban történik a fejlesztés. A fejlesztők kis csoportokban dolgoznak.
* Hátránya, hogy magasan képzett fejlesztőkre van hozzá szükség. Továbbá probléma, ha a projekt nehezen modularizálható.

### Fázisok

1. **Üzleti modellezés:** Lényege, hogy felmérjük a megrendelő üzleti folyamatait, és azokról készítsünk egy modellt.
2. **Adat modellezés:** Az üzleti modellezés során összegyűjtött adatokból objektumokat hozunk létre, azonosítjuk az attribútumokat és a kapcsolatokat.
3. **Folyamat modellezés:** Az adatmodellhez szükséges műveletek meghatározása. (Pl.: bővítés, törlés, módosítás)
4. **Alkalmazás elkészítése:** A szoftver elkészítése automatizáló eszközök bevonásával.
5. **Tesztelés:** Az új komponensek tesztelése. A régieket már nem kell tesztelni, csak integrációs tesztre és rendszertesztre van szükség.

# Agilis szoftverfejlesztési módszertanok

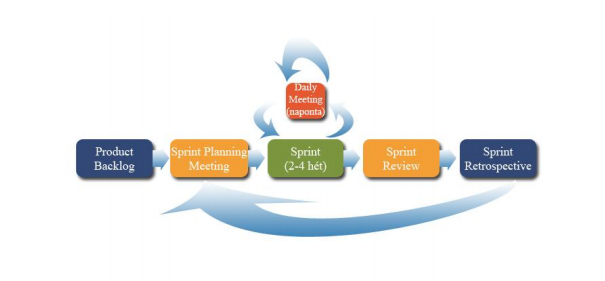
## Agilis módszertanok 12 alapelve

1. A legfontosabb a megrendelő kielégítése használható szoftver gyors és folyamatos átadásával.
2. Még a követelmények kései változtatása sem okoz problémát.
3. A működő szoftver / prototípus átadása rendszeresen, a lehető legrövidebb időn belül.
4. Napi együttműködés a megrendelő és a fejlesztők között.
5. A projektek motivált egyének köré épülnek, akik megkapják a szükséges eszközöket és támogatást a legjobb munkavégzéshez.
6. A leghatékonyabb kommunikáció a szemtől-szembeni megbeszélés.
7. Az előrehaladás alapja a működő szoftver.
8. Az agilis folyamatok általi fenntartható fejlesztés állandó ütemben.
9. Folyamatos figyelem a technikai kitűnőségnek.
10. Egyszerűség, a minél nagyobb hatékonyságért.
11. Önszervező csapatok készítik a legjobb terveket.
12. Rendszeres időközönként a csapatok reagálnak a változásokra, hogy még hatékonyabbak legyenek.

## Közös jellemzők

* **Kevesebb dokumentáció:** A projekt apró részekre van bontva, és 1-4 hétig tartó ciklusokban készül el. Ennek megfelelően nincs részletes, hosszútávú tervezés, csak minimális, ami az adott ciklushoz szükséges.
* **Növekvő rugalmasság, csökkenő kockázat:** A változásokhoz adaptálható technikákat helyezik előnybe a jól tervezhető technikákkal szemben. Minden iteráció tartalmaz tervezést, követelmények elemzését, kódolást és tesztelést. Az iterációk végén futtatható változatot kell kiadni.
* **Könnyebb kommunikáció, javuló együttműködés:** A fejlesztő csoportok önszervezőek, a tagok többféle szakterületről kerülnek ki. A csoportok mérete ideálisan 5-9 fő, lehetőleg egy irodában dolgoznak. Ennek megfelelően könnyen tudnak egymással kommunikálni.
* **A megrendelő bevonása a fejlesztésbe:** A megrendelő vagy egy kijelölt személy folyamatosan a fejlesztők rendelkezésére áll. Részt vesz a ciklus végi bemutatókon. Tovább fontos szerepe van a kifejlesztendő funkciók fontossági sorrendjének felállításában. \**Return of Invetsment, ROI: A befektetés megtérülése az üzleti érték és a fejlesztési idő hányadosa*

## Scrum



### Fejlesztési folyamat

* A Product Owner létrehoz egy Product Backlog-ot, amelyre felhasználói sztoriként felviszi a teendőket. Minden sztori rendelkezik prioritással és üzleti értékkel
* A Sprint Planning Meeting-en a csapat tagjai megbeszélik, hogy mely sztorikat vállalják. Az elvállalt sztorikat pedig kisebb részekre bontják. Továbbá meghatározzák a megvalósítás idejét (sprint hosszát), amit később már nem lehet módosítani.
* A sprint folyamán a csapat és a Scrum Master naponta megbeszéli a történteket.
  + Mit csináltál a tegnapi megbeszélés óta?
  + Mit fogsz csinálni a következő megbeszélésig?
  + Milyen akadályokba ütköztél az adott feladat megoldása során?
* A sprint végén van a Sprint Review, ahol a csapat bemutatja a sprint alatt elkészült sztorikat.
* Ezután következik a Sprint Retrospective, ahol a sprint során felmerült problémákat vitatja meg a csapat, és konkrét javaslatokat tesznek.
* Ezt követően újra a Sprint Planning Meeting következik.

### Szerepkörök

* **Scrum Mater**: Felügyeli és segíti a csapat munkáját. Garantálja, hogy a sprint időtartama nem térhet el az előre megbeszélttől, továbbá, hogy a csapat csak az elvállat feladatokkal foglalkozik. Tehát ő a projekt menedzsere.
* **Product Owner**: A megrendelő szerepét tölti be, ő a felelős azért, hogy a csapat mindig azt a részét fejlessze a terméknek, amely éppen a legfontosabb, vagyis a felhasználói sztorik fontossági sorrendbe állítása a feladata a Product Backlog-ban. A Product Owner és a Scrum Master nem lehet ugyanaz a személy.
* **Csapat:** Az aktuális sprintre bevállalt feladatokat végzik el. Ideálisan 5-9 fő. A csapatban vannak fejlesztők, tesztelők, elemzők.
* **Üzleti szereplők:** Megrendelők, forgalmazók, tulajdonosok. A Sprint Review során kapnak szerepet.
* **Menedzsment:** A megfelelő környezetet biztosítja a csapatok számára.

### Megbeszélések

* **Spring Planning Meeting:** Itt dönti el a csapat, hogy pontosan mely sztorikat vállalja el a Backlog-ból. Részletesen átbeszélik a Product Ownerrel, hogy az adott sztoriktól mit vár a megrendelő.
* **Backlog Grooming:** Itt zajlik jobb esetben a Product Backlog finomítása. Pontosítják a megrendelői igényeket, és a túl nagynak ítélt taskokat kisebb részekre bontják. Ha nincs Backlog Grooming, akkor a Planning Meeting hosszúra nyúlhat.
* **Daily Meeting:** Sprint közben, minden nap maximum 15 perc megbeszélést tartanak a csoport tagjai és a Scrum Master. Ajánlás szerint mindenkinek állnia kell a megbeszélés során. 3 kérdésre kell válaszolnia mindenkinek:
  + Mit csináltál a tegnapi megbeszélés óta?
  + Mit fogsz csinálni a következő megbeszélésig?
  + Milyen akadályokba ütköztél az adott feladat megoldása során?
* **Sprint Review Meeting:** A sprint végén az összes szereplő (a megrendelő is) összeülnek, és ellenőrzik az elkészült sztorikat. Elkészültnek minősítik a sztorit, ha minden task-ja elkészül, megfelel a követelményeknek, és a Review során elfogadták.
* **Sprint Retrospective:** A sprint során felmerült problémákat, a csapat munkáját hátráltató tényezőket vitatják meg ezen a megbeszélésen. A csapatmunka továbbfejlesztése az elsődleges, szemben az egyes személyeket érintő problémákkal, melyeket a Daily Meetingeken beszélnek meg.

### Termékek

* **Product Backlog:** Ebben a dokumentumban helyezi el a Product Owner az elvégzendő sztorikat. A Product Owner minden sztorihoz prioritást rendel, vagyis üzleti értéket. A csapat pedig minden sztorihoz az elvégzendő munka mennyiségét határozza meg. A prioritás és az üzleti érték hányadosa az ROI.
* **Sprint Backlog:** Ebben a dokumentumban az aktuális sprintre bevállalt munkák, sztorik vannak felsorolva, ezeket kell adott időn belül a csapatnak megvalósítania. A sztorik tovább vannak bontva taszkokra, és ezeket a taszkokat vállalják el a tagok a Daily Meeting során.
* **Burn down chart:** Ez egy diagram, amely segít megmutatni, hogy az ideális munkatempóhoz képest hogyan halad a csapat az aktuális sprinten belül.

### Egyéb fogalmak

* **Sprint:**  Egy előre megbeszélt hosszúságú fejlesztési időszak, általában 2-4 hétig tart, kezdődik a Sprint Planning-gel, majd a Retrospective-vel zárul. Ez egy iterációs ciklus, mely addig folytatódik, míg a Product Backlogból el nem fogytak a sztorik. Minden sprint végére el kell készülnie egy prototípusnak.
* **Akadály:** Hátráltató tényező, ami gátolja a hatékony munkavégzést. (A magánéleti problémák nem tartoznak ide.) Pl.: lejárt szoftver licence, túl lassú gép, 2 tag megsértődött egymásra. A Scrum Master feladata elhárítani ezeket az akadályokat.

## Extrém programozás (XP)

Az eddigi módszertanokból a jól bevált technikákat emeli át, és azokat „extrém” jól alkalmazza, minden mást feleslegesnek tekint.

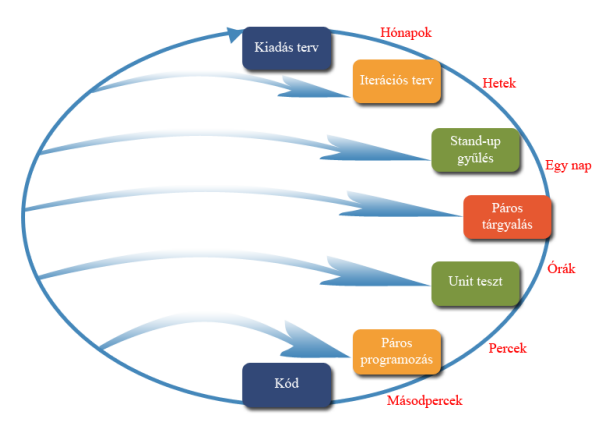
### Tevékenységek

* **Kódolás:** A projekt legfontosabb terméke a forráskód. Kódolás közben jönnek ki igazán a nehézségek, továbbá két programozó közti kommunikációra is kiváló, hiszen nem lehet többféleképpen értelmezni.
* **Tesztelés:** Extrém sok tesztelésre van szükség az összes hiba megtalálásához. A tesztek lényegében helyettesítik a dokumentációt. Metódus dokumentációjának felel meg a unit teszt, követelményspecifikációnak pedig az átvételi tesztesetek.
* **Odafigyelés:** Kiemelten fontos, hogy a fejlesztők megértsék a megrendelők igényeit. Ha egy igény kivitelezhetetlen, azt meg kell értetni a megrendelővel.
* **Tervezés:** Fontos része a fejlesztésnek a tervezés. Lehetőleg úgy kell megtervezni a programot, hogy az egyes komponensek függetlenek legyenek egymástól. (OOP alapelvek)

### Értékek

* **Kommunikáció:** Dokumentumok írása helyett az a legjobb, ha a fejlesztő csapat tagjai megbeszélik egymással a terveket, a problémákat és azok megoldását.
* **Egyszerűség:** Mindig a lehető legegyszerűbb megoldásból kell kiindulni, és ehhez kell hozzáadni az extra funkciókat. Előfordulhat, hogy újra kell írni a rendszert, de nem megy el idő olyan funkciók fejlesztésére, melyekre lehet később nem is lesz szükség.
* **Visszacsatolás:**
  + Visszacsatolás a rendszer felől: A unit, integrációs és elfogadási tesztek alapján lehet tudni, hogy a rendszer milyen állapotban van.
  + Visszacsatolás a megrendelő felől: A gyakori prototípus bemutatások során a megrendelőnek lehetősége van elmondani, hogy mit szeretne másképpen.
  + Visszacsatolás a csapat felől: A megrendelő új igényeire a csapat megbecsüli a kifejlesztés idejét.
* **Bátorság:** Bátorság kell ahhoz, hogy az egyszerűbb utat válasszuk annak fényében, hogy lehet később az egészet újra kell írni. Bátorság kell ahhoz is, hogy mások kódját kezdjük el szépíteni.
* **Tisztelet:** Tisztelni kell a csapat tagjainak munkáját. Nem szabad feltölteni hibás kódot a verziókövető rendszerbe. Saját kód esetén törekedni kell a legnagyobb minőségre és átláthatóságra.

### Technikák



* **Páros programozás:** Két programozó együtt dolgozik, az egyik írja a kódot, a másik figyeli. Ha valamit nem ért, akkor szól és átbeszélik.
* **Tesztvezérelt fejlesztés:** Metódus megírása előtt a teszteseteket írjuk meg hozzá.
* **Code review:** A vezető fejlesztő átnézi a kódot, és javaslatokat tesz, hogy hogyan lehetne, szebben/jobban megvalósítani az adott implementációt.
* **Folyamatos integráció:** A verziókövető rendszerbe bekerült kódokat integrációs teszt alá vonjuk a nap vagy a hét végén.
* **Kódszépítés:** Kódban lassú, rugalmatlan vagy csúnya részeket át lehet alakítani. Fontos, hogy csak tesztelt kódon lehet szépíteni, úgy, hogy a kód funkcionalitása nem változhat.

# Összevetés

|  | **Hagyományos** | | | | | | **Agilis** | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Vízesés modell** | **V-modell** | **Prototípus modell** | **Spirál modell** | **Iteratív, inkrementális** | **RAD** | **Scrum** | **XP** |
| **Életciklus fázisainak sorrendje** | lineáris | lineáris | általában iteratív | spirális | iteratív | iteratív | iteratív | iteratív |
| **Implementációs nyelv** | strukturált | strukturált | nem strukturált | nem strukturált | objektum orientált | objektum orientált | objektum és szerviz orientált | objektum orientált |
| **Technikai célok** | jól dokumentált | jól dokumentált | prototípus, rapid, agilis, extrém | prototípus | prototípus, rapid, agilis, extrém | rapid | prototípus, agilis, rapid | prototípus, agilis, extrém |
| **Dokumentáltság** | nehézsúlyú | nehézsúlyú | könnyűsúlyú | könnyűsúlyú | könnyűsúlyú | könnyűsúlyú | könnyűsúlyú | könnyűsúlyú |
| **Modell „középpontja”** | követelmény központú | teszt központú | követelmény központú | követelmény központú | követelmény központú, használati eset központú | követelmény központú, megrendelő és csapat központú | csapat központú | csapat központú |