**1. modul**

Introduction to Basics

SDLC

Csapatmunka, fejlesztési módszertanok

# Bevezetés

Miért kell az IT-ban csapatban dolgozni, amikor egyedül is lehet? A válasz viszonylag egyszerű azért, mert akkor gyorsabban elkészülünk. Egy kellően senior fejlesztő bármilyen alkalmazást meg tud egyedül is csinálni, de egy viszonylag komplex webes alkalmazás esetén ez 5-8 évet is igénybe vehet még ha főállásként is dolgozik rajta. Viszonylag nehéz addig bevétel nélkül létezni… Tehát kulcsfontosságú megtanulni csapatban dolgozni és hogy hogyan kell jól együttműködni. Az IT nem egy irígy közeg, ne féljünk segítséget kérni és ne érezzük rosszul magunkat miatta, illetve mi is érezzük kötelességnek segíteni a másikat.

# Csapatmunka

A csapatmunka az, amikor együtt dolgozunk másokkal, azaz nem egy ember dolgozik, nem egy ember ötletét valósítjuk meg, hanem a csapat minden tagja hozzátesz valamit a munkához (érdemileg).

# Menete

1. **leírás/analízis**: a feladat megfogalmazása és megértése
2. **tervezés**: a fő feladat felosztása kisebb feladatokra, a munkamenet felvázolása és **felosztása**, vagyis a kisebb feladatok megvalósításának kiosztása
3. **megvalósítás**: a kisebb feladatok megvalósítása (egyénileg vagy közösen), kommunikáció a csapattagokkal, feladatok újratervezése (szükség szerint), a felmerülő hibák kezelése
4. **kiadás:** elérhetővé tenni a felhasználóknak
5. **utómunka**: a program minden funkciójának tesztelés, a hibák javítása, refaktorálás, szükség szerint újratervezés vagy módosítás

## Leírás/Analízis

Az első és legfontosabb feladat az, hogy magát a feladatot megértsük

* minden csapattagnak értenie kell, hogy mi a feladat, mi a cél
* a legfőbb kérdések, amelyekre válaszolni kell:
  + Mi a feladat? (Egy mondatban összefoglalva.)
  + Mit várunk? Mi lesz a végeredmény?
  + Mit kell feltétlenül tudnia a programnak?
  + Mi az, amivel esetleg fejleszthetjük még a programot? (Plusz funkciók.)
  + Milyen eszközökre van szükségünk a megvalósításhoz? (Például: valamilyen szöveges nyersanyag, IDE license, API-k stb.)

Megjegyzés: nyilván a programozási nyelv a mi esetünkben egyértelmű...

## 

## 

## Tervezés

A fő feladat megfogalmazása után fel kell osztani azt kisebb feladatokra. Erdemes olyan kis feladatokra felosztani, amelyek könnyen megfogalmazhatóak és elvégezhetőek például: a szakácskönyvekben a receptek nem annyit írnak, hogy “vedd a hozzávalókat, állítsd össze, azután főzd/süsd, tálald” - hanem lépésről lépésre leírják, mit csinálj.

A kisebb feladatokat további, még kisebb feladatokra lehet osztani. Nincs szabály arra, hogy hány (és mekkora) feladatokra kell osztani a fő feladatot - ez szokás, megállapodás, “érzés” kérdése.

A megfogalmazott kisebb feladatokat sorrendbe kell állítani. Melyik a legfontosabb? Melyik kevésbé fontos? Mi a feladatok közötti összefüggés? A kisebb feladatok sorrendbe állítása lényegében a munkamenet maga

Nem szabad elfelejteni, hogy a feladatok és a munkamenet változhat a projekt terve nincs kőbe vésve!

### Felosztás

A tervezés része a felosztás. Mivel csapatmunkáról van szó, ezért a csapattagok mind részt vesznek a munkában tehát a kisebb feladatokat ki kell osztani a csapattagok között. A kiosztás azt jelenti, hogy a csapattagok választanak maguknak feladatot (vagy feladatokat), amit (vagy amiket) elvégeznek. Nem kötelező az összes kisebb feladatot már a munka elején elvállalni(kiosztani) - a lényeg, hogy előbb-utóbb mindegyik feladatot elvégezze valaki.

Egy feladaton több csapattag is dolgozhat együtt - de akkor tényleg együtt dolgozzanak, és ne egymás mellett párhuzamosan!

A feladatok kiválasztásánál érdemes követni a prioritási listát.

Több megközelítés is lehetséges a feladatok kiosztásánál, több tényezőt figyelembe véve. A csapattagok kompetenciáját, a feladat sürgősségét vagy tudásmegosztást szem előtt tartva.

## Megvalósítás

A tervezés és feladatválasztás után jöhet a tényleges kódolás. A csapattagok önállóan vagy akár együtt is dolgozhatnak (ugye), a megvalósítás alatt folyamatos kommunikáció szükséges.

Tájékoztatni kell a csapatot arról, hogyan áll a feladat megvalósítása. Időről-időre meg kell osztani az elkészült kódot (kódrészletet) a csapattal.

A munka közben felmerülő kérdéseket, problémákat, esetleges tervezési hibákat meg kell vitatni (szükség szerint újra kell tervezni). Szabad segítséget kérni (nyilván), sőt!

Egy kisebb feladat befejezése után jöhet a következő, de csak azután ha a **tesztelés** és refaktorálás megtörtént és jó minőségű kódot adunk ki a kezünkből. A különböző funkciók mellett az egész programot (alkalmazást) is tesztelni kell.

## Kiadás

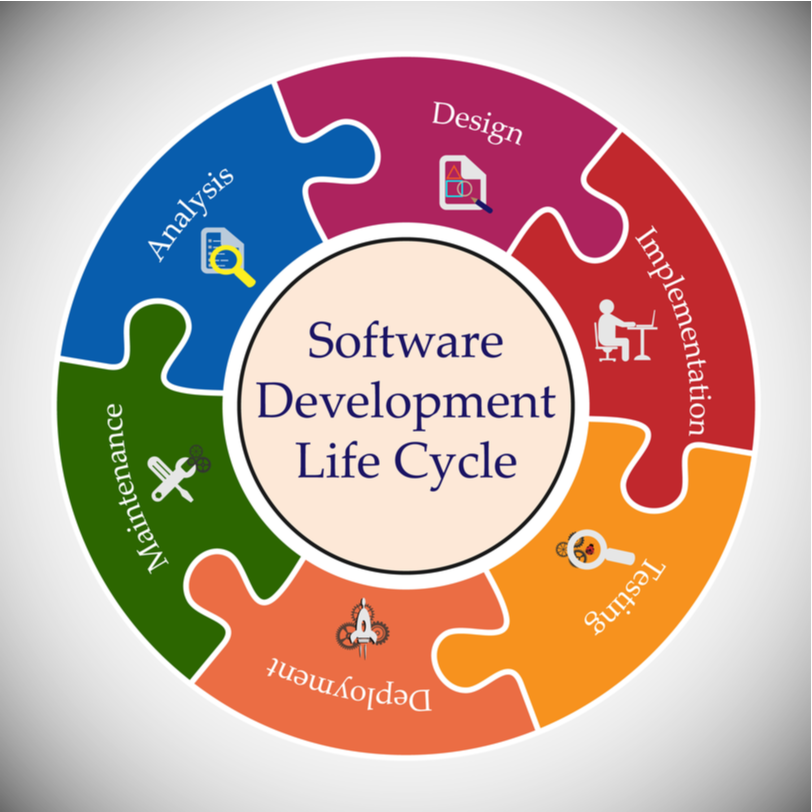
Ha a (rész) fejlesztés és tesztelés rendben volt, akkor lehet kiadni a kódot, terméket, programot és mindenki számára elérhetővé tenni. (release)

## Utómunka/Karbantartás

Már a megvalósítás alatt tesztelni kell a különböző funkciókat, valamint javítani az esetleges hibákat. Hibás, nem működő kódot soha nem adunk ki a kezünkből!Azonban bármikor előfordulhat, hogy az általunk már késznek nyilvánított feladatot, javítani vagy bővíteni kell. A fejlesztés, tesztelés és hibajavítás után át kell nézni a tervezett feladatokat.

# SDLC

A fent vázolt csapatmunka menetét, az IT-ban SDLC-nek, másnéve Software Development Life Cycle-nek nevezik.



Ahogy megfigyelhető az ábra egy kör, és ez nem véletlen. Az SDLC nem egy egyszeri folyamatot mutat be, ez egy folytonosan megismétlődő vagyis iterációkon áteső feladatok halmaza. Jó esetben…

# SDLC Modellek

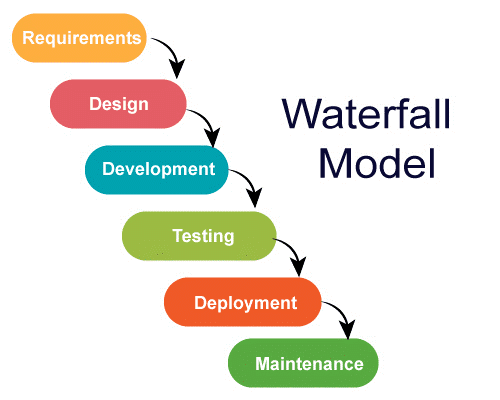
Népszerű vagy régebben népszerű modellek a következőek:

* Vízesés modell - Waterfall Model
* V-Model
* Iteratív modell - Iterative Model

## Vízesés modell

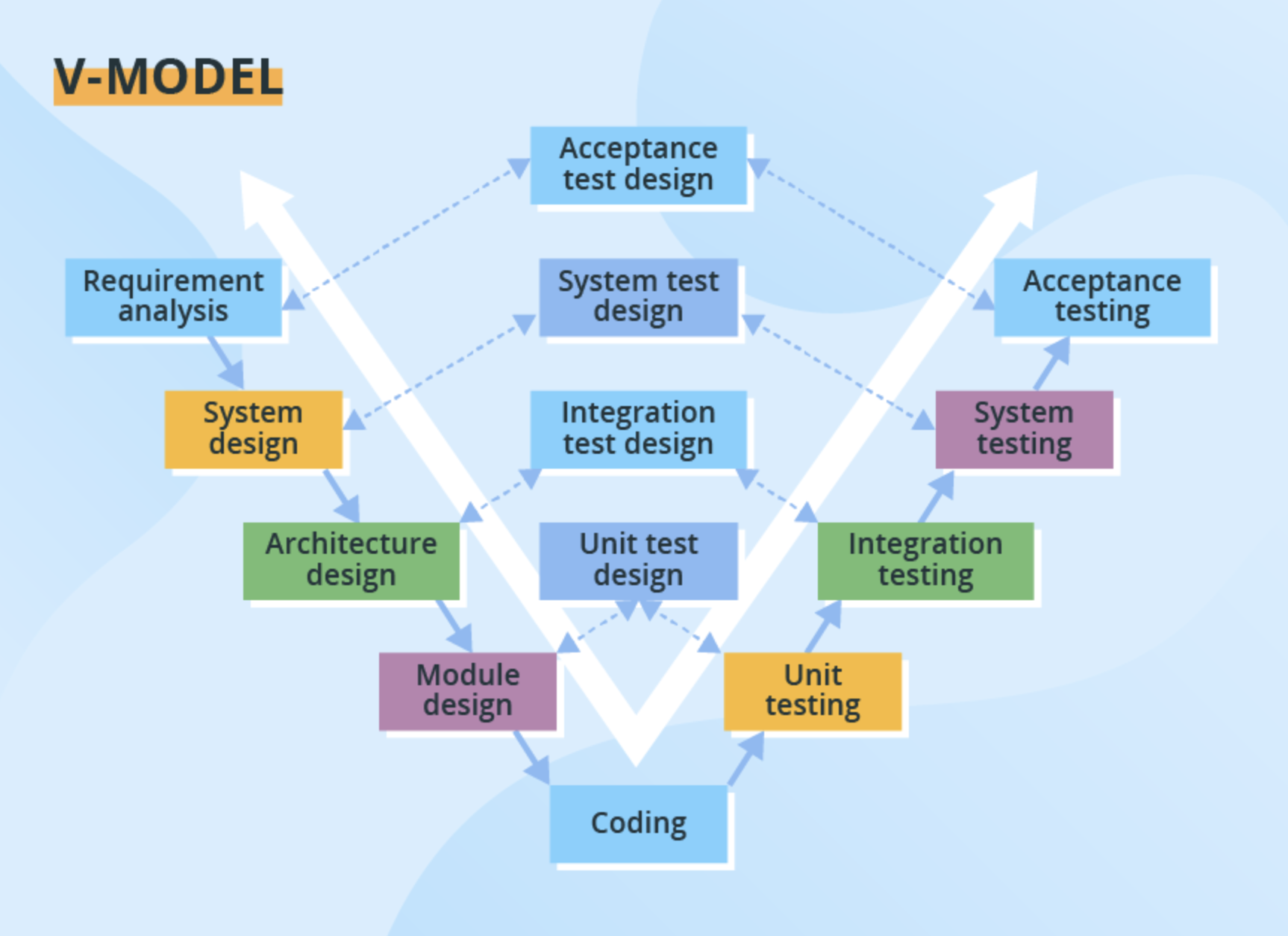
Itt a folyamatok egymás után hajtódnak végre csak és kizárólag akkor lépünk a következő lépcsőfokra, ha a fenti teljes egészében befejeződött. A tesztelés itt túl későn kezdődött meg, így nagyon költséges volt visszalépni az előző fázisok egyikére. Nagyszámú dokumentum mennyiség, fajsúlyos tervezés biztosítja a program megfelelő elkészültét. Ám a termék kis-mértékű utólagos módosítása is horribilis költségekkel járhat.

Nagy hátránya lehet még, hogy lehet akkor derül ki a felhasználó számára, hogy “de nem is ezt kértem” vagy “legyen inkább máshogy”, amikor már átadásra került a teljes termék. A megrendelés és átvétel között pedig eltelt mondjuk pár év….



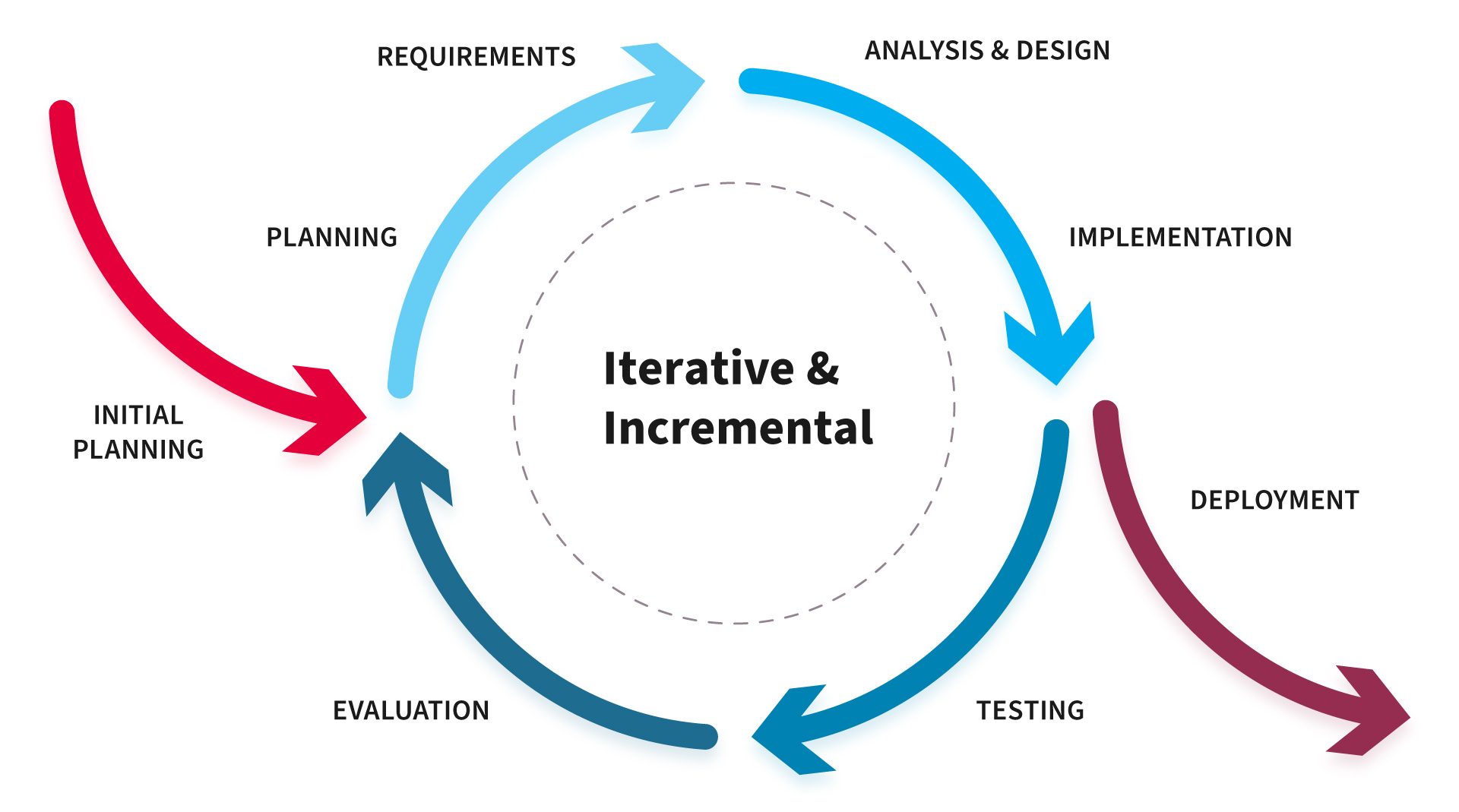
## V-modell

Itt vannak bizonyos folyamatok, amik jól végezhetőek egyidőben, ezzel gyorsítva a fejlesztés egészét. Minden lépcsőfoknak megvan a maga technikája, annak a helyességének a tesztelésére, tehát ez gyakorlatilag a vízesés modell egy kiegészítése. Gyakorlatilag, mire a fejlesztő elkezdi írni a kódot, már a helyességét ellenőrizni hivatott tesztek mind készen vannak.



## Iteratív modell

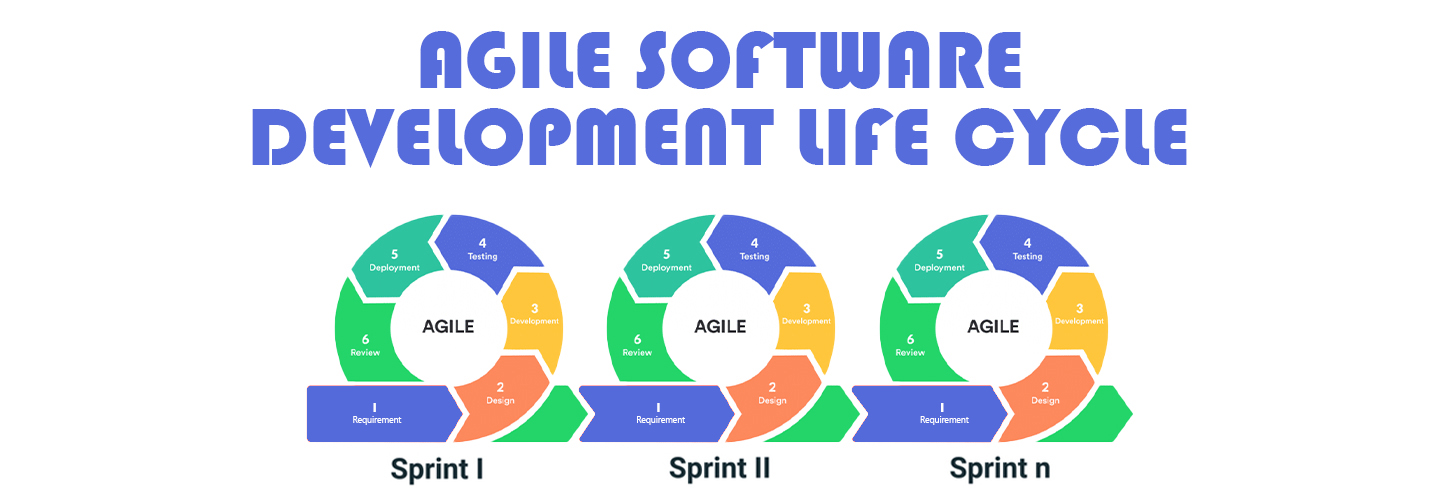
Itt a kulcsszó az iteráció, ami ebben az esetben azt jelenti, hogy mihelyst van egy müködő kis önálló egység, azt ki is adom és elérhetővé teszem a felhasználónak, hogy minél hamarabb visszajelzést kapjunk.



## Agile model

El is érkeztünk manapság a legdivatosabb SDLC-hez, az agilehoz. Ez egyébként az iteratív modell továbbgondolt, továbbfejlesztett változata. Itt a kulcsszó az a mihamarabbi visszajelzés, ami ebben az esetben azt jelenti, hogy mihelyst van egy müködő kis önálló egység, azt ki is adom és elérhetővé teszem a felhasználónak, megrendelőknek, hogy minél hamarabb elmondják a véleményüket. Tehát folyamatos az együttműködés, ahogy folyamatos az alkalmazkodás a változásokhoz is.

A fejlesztési ciklusokat legtöbbször sprint-eknek nevezzük. Minden sprint végén a cél, egy **működő** termék vagy termékfrissítés kiadása. Egy sprint tipikusan 2-3 hét hosszú és a megrendelő vagy egy megbízottja folyamatosan a fejlesztőcsapat rendelkezésére áll a kérdések megválaszolására. Ideális csapatméret az 5-9 fős létszám. Mottó: csapat van nem egyén!



Hátránya lehet, hogy a bővíthetőség, karbantarthatóság nehézségekbe ütközhet. Illetve kell legyen valaki, akitől kapja a csapat a visszacsatolást folyamatosan. Viszont előnye, hogy flexibilis, kevesebb tervezést igényel és nem akkora baj ha menet közben változik a követelmény.

### SCRUM

Az egyik megvalósítása az Agile SDLC-nek a SCRUM.

Fontos tudni, hogy a következő “kötelező” megbeszélések vannak SCRUM esetén:

* grooming
* sprint planning
* daily standup
* sprint review/demo
* retrospective

A spinten belüli feladatokat a követelmények alapján ki kell dolgozni, pontosítani kell. Meg kell tervezni, hogy hogyan kell megvalósítani. Valamint mit miután, milyen sorrendben. Ezeket általában a csapat egy **grooming** megbeszélésen bonyolítja le. Jellemzően 2-4 óra ha jól felkészült mindenki A fejlesztőcsapatból legalább egy-egy fejlesztő, tesztelő, csapatvezető és a scrum master, project manager vagy product owner.

Az, hogy az adott sprint-ben mi fog elkészülni (nagyjából) a sprint tervezésen (**sprint planning**) dől el, ami egy olyan megbeszélés amin az egész fejlesztőcsapat részt vesz (sokszor a megrendelő, képviselője is valamint csapatvezető, architech…). Jellemzően 1-2 óra és mindenki részt vesz.

Mivel a kommunikáció,visszacsatolás nagyon fontos, így a sprint minden napján van egy napi státusz, **daily standup** amelyen mindenki egyesével elmondja, hogy hogy haladt előző nap a feladatával, mi a terve mára és van-e valami amiben segítséget szeretne. Jellemzően 10-15 perc és a fejlesztő csapat vesz részt rajta.

Az pedig, hogy a csapat jól dolgozott-e, gyakorlatilag a **sprint review**, vagy más néven sprint demo-n dől el. Itt van egy kellően kompetens személy, aki meg tudja ítélni az átadott funkció(k) helyességét. Jellemzően a csapatból egy vagy több személy élőben bemutatja annak működését és megbeszélik azt.

Ha már visszacsatolás, akkor bevett gyakorlat, hogy a sprint végeztével a csapat megbeszéli, kiértékeli az adott sprint teljesítményét egy megbeszélésen. Ez a **retrospective**. Mindenki elmondhatja mi az amit úgy látott, hogy jól működött, mi az ami tetszett vagy hálás, de ugyanúgy mindenki (illedelmesen!!) elmondhatja mi az ami mehetett volna jobban. Megbeszélik van-e valami olyan tennivaló amit végre lehet hajtani hogy a “fejlesztendő” dolgok jobbak legyenek. Jellemzően 1-2 óra hosszú és a fejlesztési csapat vesz részt rajta.

# Linkek

<https://www.tutorialspoint.com/sdlc/sdlc_overview.htm>

<https://www.javatpoint.com/software-engineering-software-development-life-cycle>

# Feladatok

1. Miért jobb csapatban dolgozni?
2. Mik az SDLC fázisai?
3. Milyen modellekről tanultunk?
4. Mi az előnye az Agile-nak a Vízesés-sel szemben?
5. Milyen SCRUM meeting-eket ismertek? Melyiket mire használják?
6. Rajzolós feladat - Waterfall
7. Rajzolós feladat - Agile