MILTON FRIEDMAN EGYETEM

# PROGRAMOZÁSI TECHNOLÓGIÁK

## 3. óra - Módszertanok

2024.03.24

Szalai Patrik

szalai.patrik@uni-milton.hu

## TÉMAKÖRÖK:

### 2, Szoftverkrízis

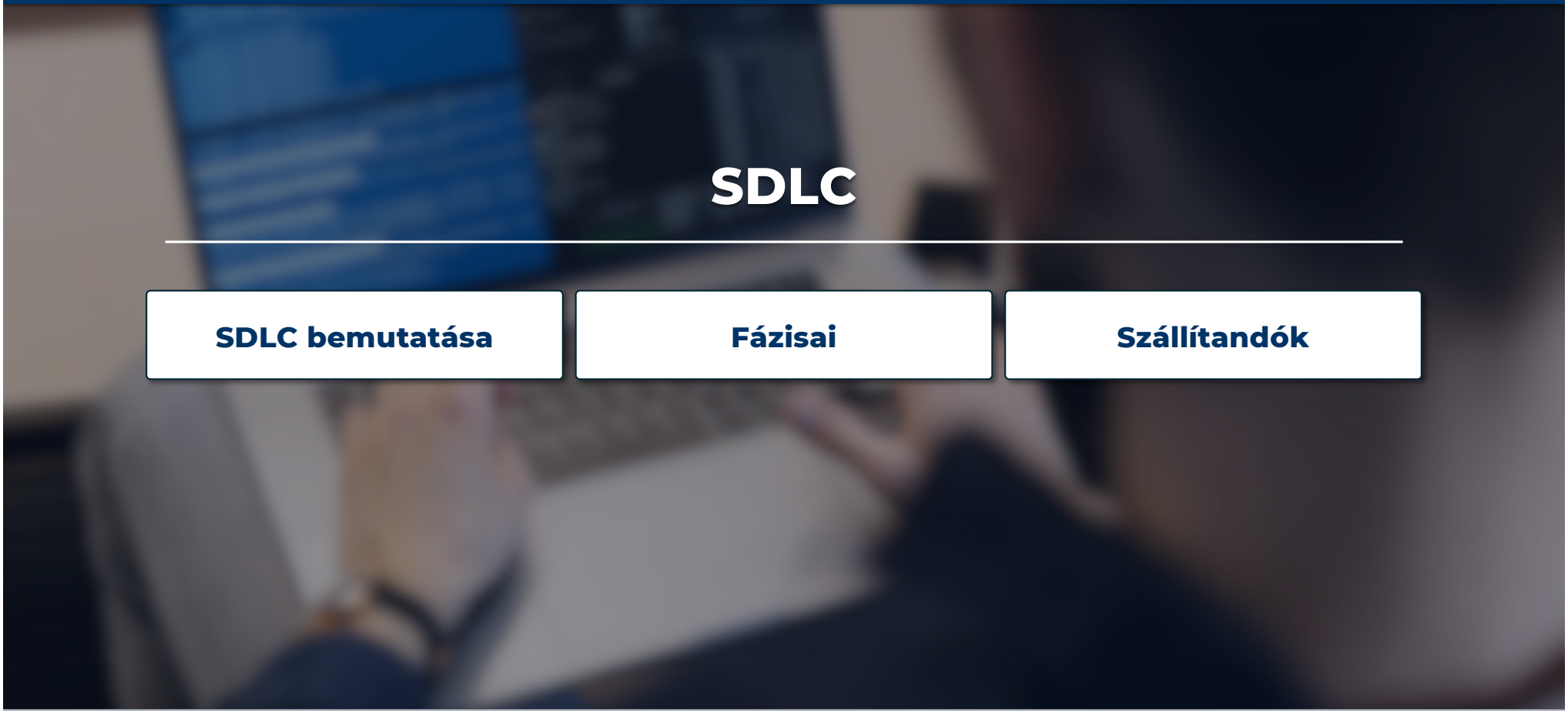
- SDLC

### 3, Módszertanok

- Strukturális modellek
  - Vízesés modell
  - SSADM
  - V-modell
- Prototípusmodell
- Spirálmodell
- Iteratív és Inkrementális
  - RUP
  - RAD
  - Agile
  - Extrém programozás
  - Scrum

### 4, Kockázatkezelés

- Bevezetés



# SDLC

SDLC bemutatása | Fázisai | Szállítandók

## SOFTWARE DEVELOPMENT LIFE CYCLE

- **A fogalom**
  - A szoftverrel egyidős fogalomról beszélünk
  - Életciklus
  - Igénytől az átadásig
  - Szakirodalom 7-9 ”Fázisra” vagy Lépésre bontja
  - Fázisait módszertanok határozzák meg
  - *„A program kódja folyamatosan változik”*


2. ábra: A szoftverfejlesztés életciklusa

**Kérdés:** Egy szoftver életciklusa meddig tart?

# SOFTWARE DEVELOPMENT LIFE CYCLE "CHEAT SHEET"

| A fázisok | A szállítandók/eredménytermékek |
|---|---|
| A felhasználókban új igény merül fel [user would like to have a new feature] | -- |
| Az igények, követelmények elemzése, meghatározása [Requirement Analysis] | **Követelményspecifikáció** |
| Rendszerjavaslat kidolgozása [Functional Analysis] | **Ütemterv, szerződéskötés** |
| Rendszerspecifikáció [Analysis & Design] | **Megvalósíthatósági tanulmány, HLD** |
| Logikai és fizikai tervezés [Analysis & Design] | **Logikai- és fizikai rendszerterv** |
| Implementáció [Implementation] | **Maga a szoftver** |
| Tesztelés [Testing & Validation] | **Teszterv, tesztesetek, tesztnapló, validált szoftver** |
| Rendszerátadás és bevezetés [Delivery & Deployment] | **Felhasználói dokumentáció** |
| Üzemeltetés és karbantartás [Operating & Maintenance] | **Rendszeres mentések, karbantartási folyamatok, felügyelet** |
| **És kezdődik elölről...** | |

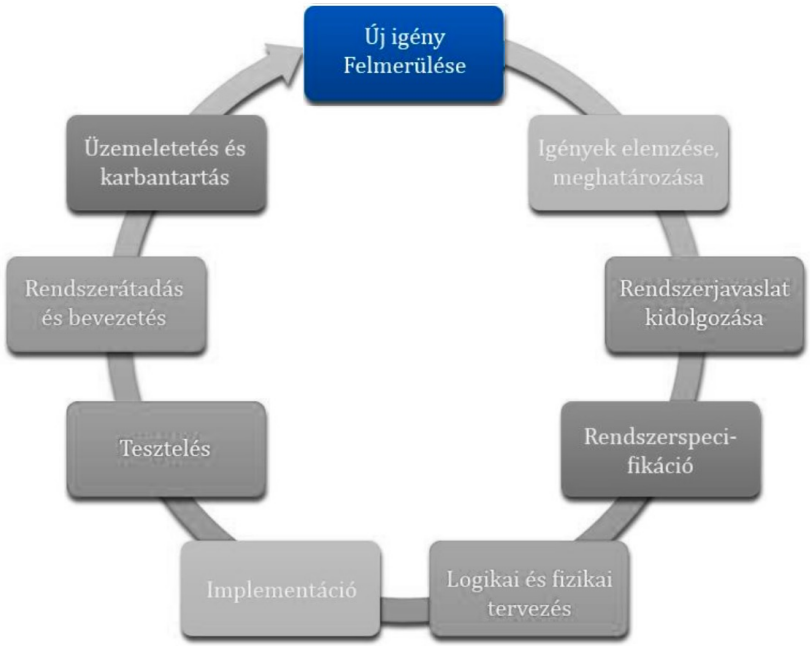
# 1 – AZ ÚJ IGÉNY

- **Milyen szotfverről van szó?**
  - Egyedi szoftver
  - Dobozos szoftver

  **Fontos még:**
  - Egyedi- és Dobozos közötti átmenetek
  - Keretrendszerek

- **Mi az igény eredete?**
  - Incident és Service Request közötti különbségek
  - Új megrendelés
  - „Belső” üzletfejlesztés

- **Milyen igényről van szó?**
  - Funkcionális – Konkrét folyamatot kell fejleszteni
  - Nemfunkcionális – „Legyen gyorsabb”




2. ábra: A szoftverfejlesztés életciklusa

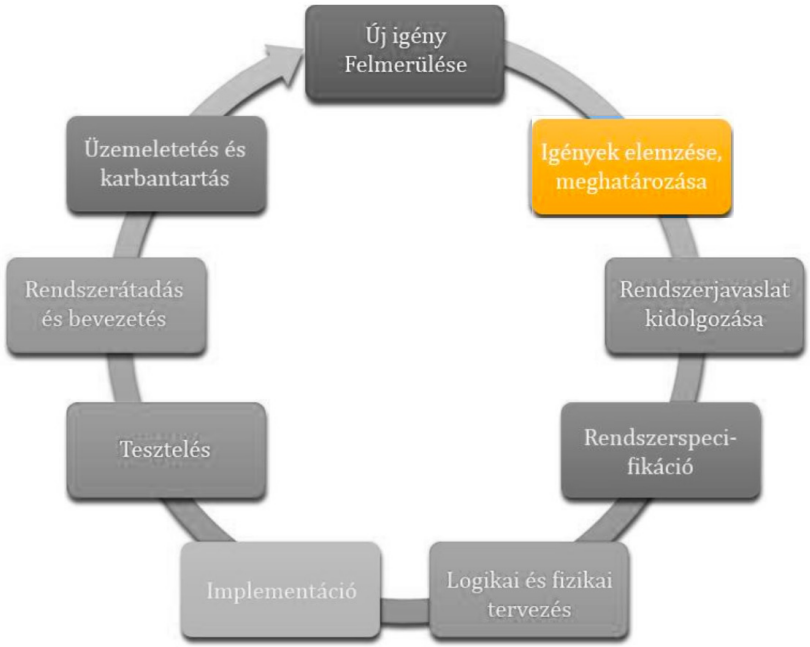
## 2 – AZ IGÉNYEK MEGHATÁROZÁSA

- **Készítsünk... Riportot!**
  - Szabad riport
  - Irányított riport

  **Fontos még:**
  - Bejelentés
  - Piackutatás

- **A riport céljai:**
  - A megrendelő üzleti folyamatának megismerése
  - A megrendelő igényeinek megismerése

- **Mi a célja az új rendszerrel?**
  - Kiváltás
  - Módosítás
  - Adatforrás
  - Adatgyűjtő




2. ábra: A szoftverfejlesztés életciklusa

## SZÁLLÍTANDÓ: KÖVETELMÉNYSPECIFIKÁCIÓ

- **Hogyan írjunk ( jó ) követelményspecifikációt?**
  - **MUST HAVE:** Követelménylista

**Kérdés:** Miket érdemes még leírni?

# SZÁLLÍTANDÓ: KÖVETELMÉNYSPECIFIKÁCIÓ

- **Hogyan írjunk ( jó ) követelményspecifikációt?**
  - **MUST HAVE:** Követelménylista


**Kérdés:** Miket érdemes még leírni?

- A **jelenlegi helyzet** leírása és a **vágyálomrendszer** leírása
  - Jelenlegi üzleti folyamatok **modellje**
  - Igényelt üzleti folyamatok **modellje**
- **Jogi háttér** – Pályázat, Törvények, Rendeletek, Szabványok, Ajánlás
  - Pl.: Egy MNB ajánlás „kötelező szorgalmi feladat”
- **Riportok** szövege „jegyzőkönyve”
- **Fogalom**szótár
  - **Példa:** Mit értünk az alatt, hogy MFA?
    - 2 Faktor vagy több?
    - Van-e benne SSO követelmény?
    - OTP vagy PUSH esetleg egyéb?
    - MFA szolgáltató?
  - *Az ügyfél a **bejelentkezési folyamat** során **Microsoft MFA-t** szeretne alkalmazni, mely a **Microsoft Authenticatorban PUSH** notification alapon validálja a bejelentkezni kívánó személyt, majd ezt követően **további bejelentkezés nélkül tovább engedi a Citrix és az SAP rendszerbe is.***

# SZÁLLÍTANDÓ: KÖVETELMÉNYSPECIFIKÁCIÓ

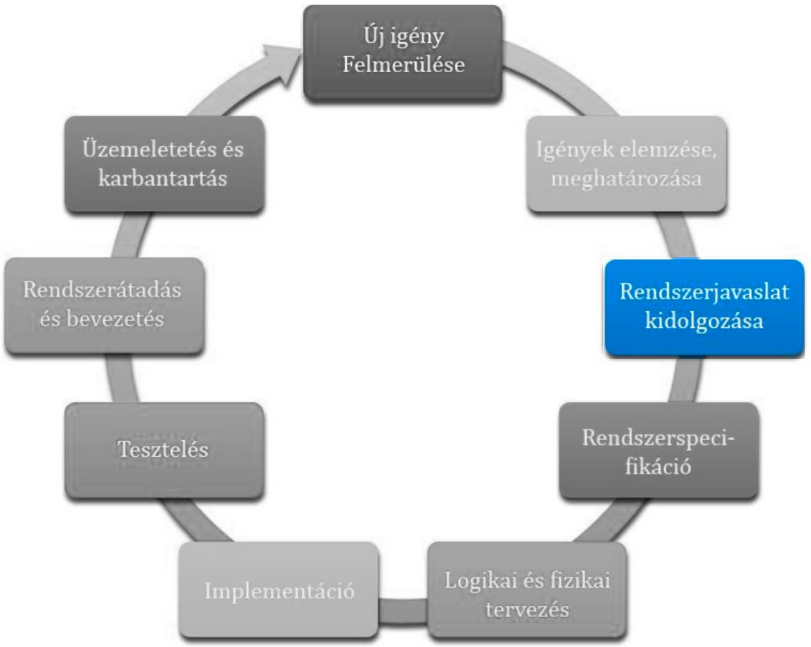
- **A követelménylista elemei**
  - Funkcionális és Nemfunkcionális követelmények
  - Példa specifikáció megtalálható a könyvben!

- **Nemfunkcionális követelmények:**
  - **Minőségi szempontokból:**
    - Helyesség
    - Használhatóság
    - Megbízhatóság
    - Hatékonyság/Teljesítmény
  - **Környezetbe illeszthetőség szempontjából:**
    - Form-factor – Méret, hordozhatóság, On-Prem
    - Robosztusság – Hibatűrés, HA, BCP
    - Kompatibilitás, Integrálhatóság, Bővíthetőség, Egyedi Konfigurálhatóság
  - **Üzemeltetés szempontjából:**
    - Könnyű deploy – Vásárlás, Letöltés, Telepítés
    - Karbantarthatóság, Könnyű betanulás
    - Megfelelő licenszelés – pl.: CAPEX / OPEX és ha már ezekről beszélünk... ROI

## 3 – A RENDSZERJAVASLAT

- **Ebben a fázisban:**
  - Funkcionális specifikáció kidolgozása
  - Szerződés (Nem minden metodológiában!)

**Kérdés:** Mi van ha mégsem sikerül megállapodni? Jól van ez így?



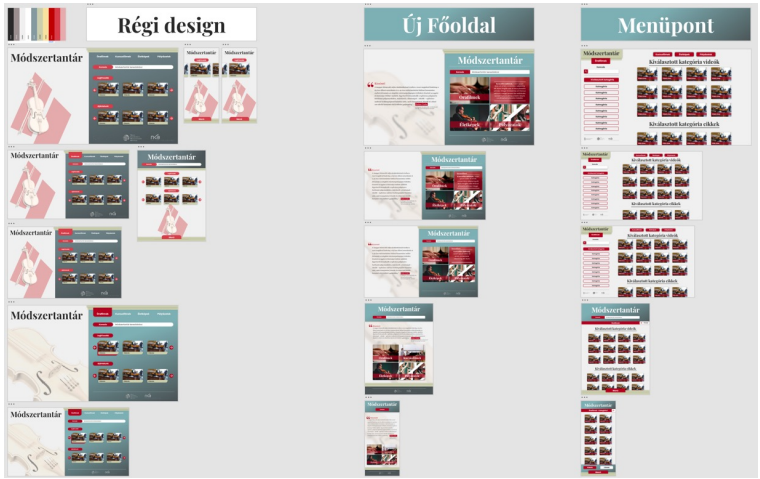

2. ábra: A szoftverfejlesztés életciklusa

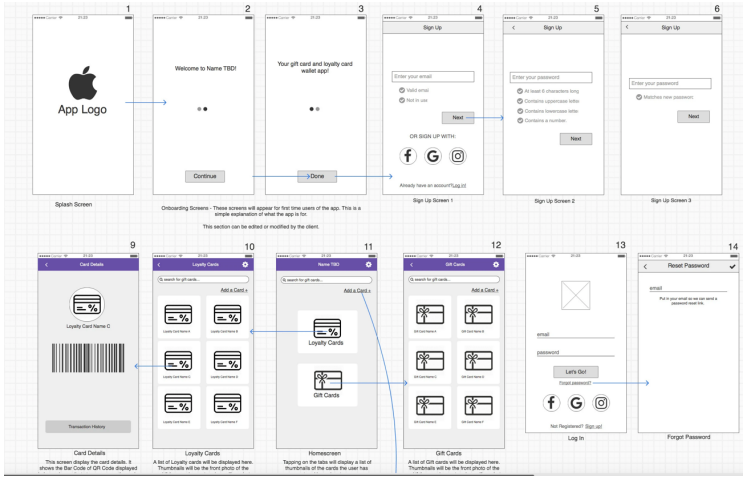
# SZÁLLÍTANDÓ: FUNKCIONÁLIS SPECIFIKÁCIÓ

- **Ami pluszba bekerül:**
  - Használati esetek (Angol szóval?)
  - Képernyőtervek – UI
  - Forgatókönyvek – UX
  - Funkciók – Követelmény **megfeleltetés – Traceability**

**Fontos:** Az ügyfél (felhasználó) szemszögéből kell írni!

- **Követelménylistánkban már megvan:**
  - A jelenlegi helyzet leírása és a vágyálomrendszer leírása
    - Jelenlegi üzleti folyamatok modellje
    - Igényelt üzleti folyamatok modellje
  - Jogi háttér – Pályázat, Törvények, Rendeletek, Szabványok, Ajánlások
  - Riportok szövege „jegyzőkönyve”
  - Fogalomszótár



## SZÁLLÍTANDÓ: ÜTEMTERV ÉS ÁRAJÁNLAT

- **Ütemterv fő pontjai:**
  - Mely funkciók, milyen ütemezéssel kerülnek be a rendszerbe
  - Verziózás
  - Példa ütemterv a tenkönyvben!

- **Árajánlat fő pontjai:**
  - Hány „manday” a fejlesztés vagy munka ideje?
  - Mennyi a napidíj?

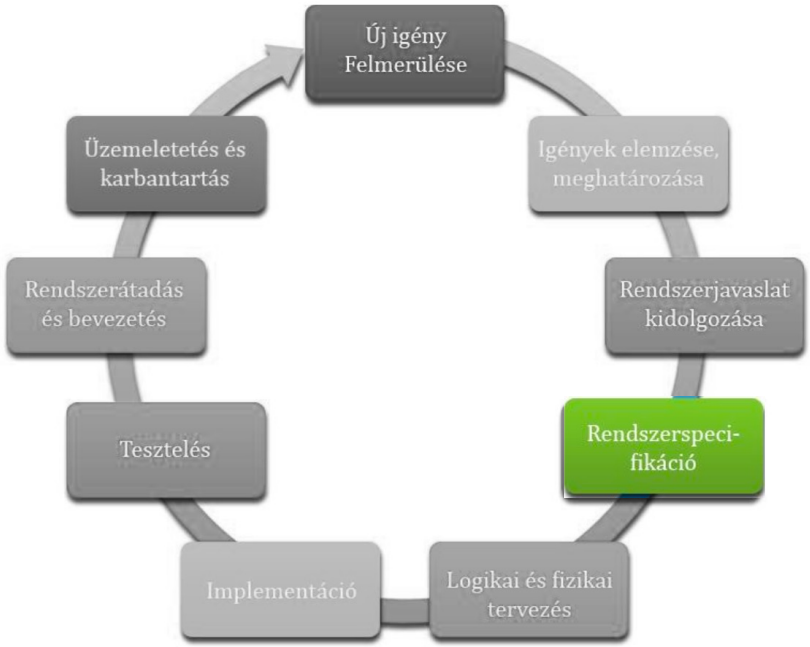
**Kérdés:** Mennyi legyen a napidíj?

# 4 – RENDSZERSPECIFIKÁCIÓ

- **Ebben a fázisban:**
  - Megvalósíthatósági tanulmány – Feasibility study
  - Magasszintű rendszerterv – HLD

- **Feasibility study**
  - Terjedelme 10 - 50 oldal
  - Döntéshozók számára készül

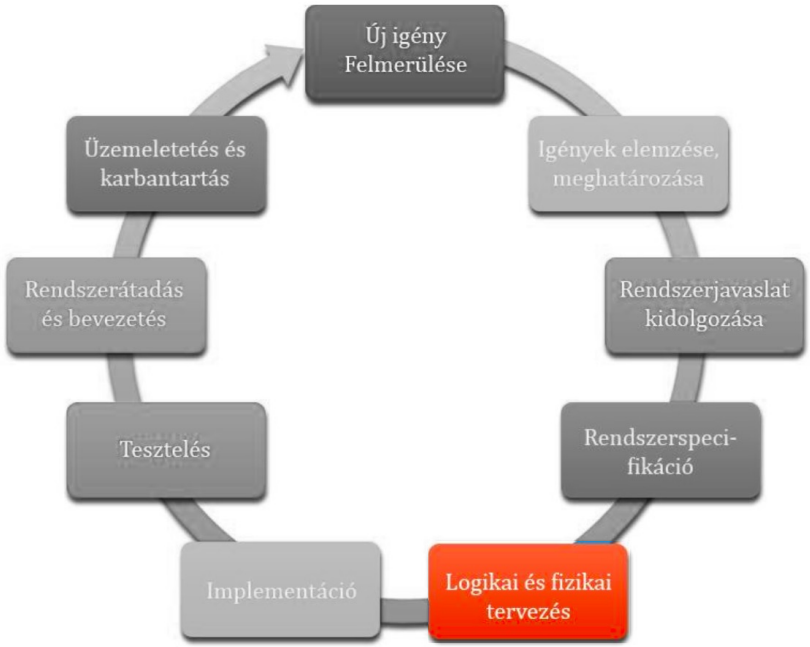
- **HLD**
  - Mit (A rendszert)
  - Miért (azzal a céllal, hogy...)
  - Hogyan (Terv alapján meghatározott módon...)
  - Mikor (Ütemterv alapján...)
  - Miből (...Erőforrásokkal)
  - ... akarunk létrehozni. Fontos, hogy reális, megvalósítandó lépéseket írjunk csak bele!
- Konceptuális, High-Level és Részletes HLD




2. ábra: A szoftverfejlesztés életciklusa

## 5 – LOGIKAI ÉS FIZIKAI TERVEZÉS

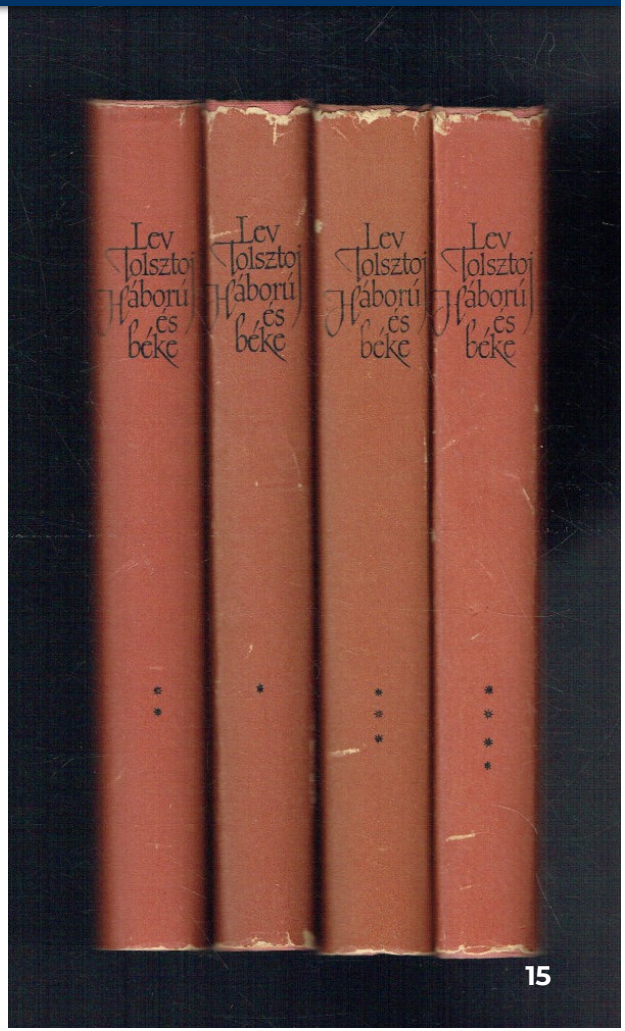
- **Logikai rendszerterv**
  - Az ideális rendszerről szól
  - Nem veszi figyelembe a fizikai limitációkat
  - Csak a rendszerből eredő limitációkkal számol
- **Fizikai rendszerterv**
  - Logikai rendszerterv finomítása
  - Figyelembe veszi a fizikai limitációkat
- **Közös jellemzők**
  - Szöveges leírások
  - UML ábrák
  - Az új, Lightweight módszertanok már elhagyják őket
  - Alapja a funkcionális specifikáció
    - De itt a programozók, üzemeltetők szemszögéből




2. ábra: A szoftverfejlesztés életciklusa

## 4 – RENDSZERTERV FEJEZETEI

- Az új módszertanok már elengedték a szükségességét
- Nagyon sok idő egy rendszertervet megírni és nincs rá de-facto keret, vállalatonként változhat!
- **Az anyag tanulásakor olvassuk el és értelmezzük minden elem lényegét, szerepét és fontosságát!**

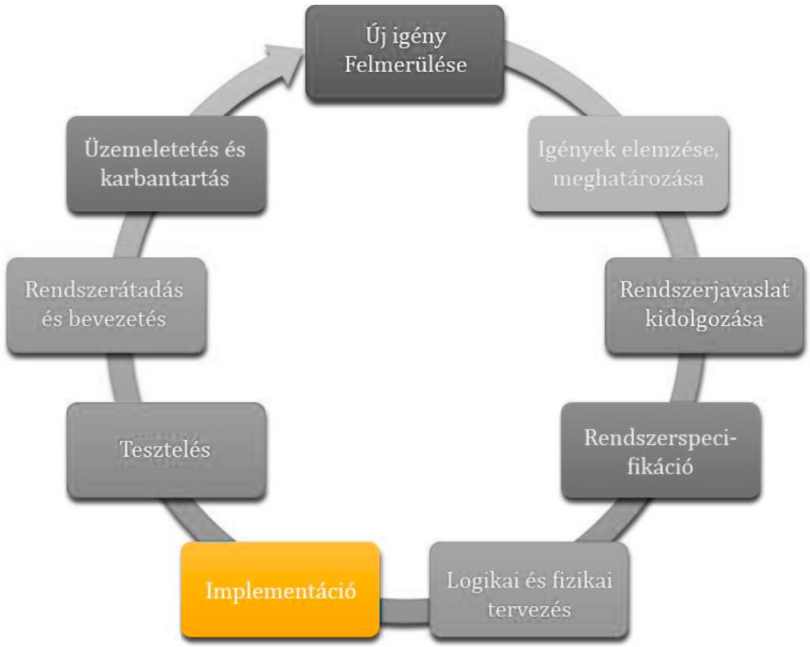


# 6 – IMPLEMENTÁCIÓ

- **Logikai rendszerterv**
  - Az ideális rendszerről szól
- **Szállítandók**
  - Forráskód
    - Régi metodológiák – PROD kód
    - Új metodológiák – Prototípus kód
  - Programozói dokumentáció

**Kérdés:** Mit tehetünk, ha Implementáció során valami mégis megvalósíthatatlannak nyilvánul?

**Kérdés:** Hogyan érdemes belefogni?

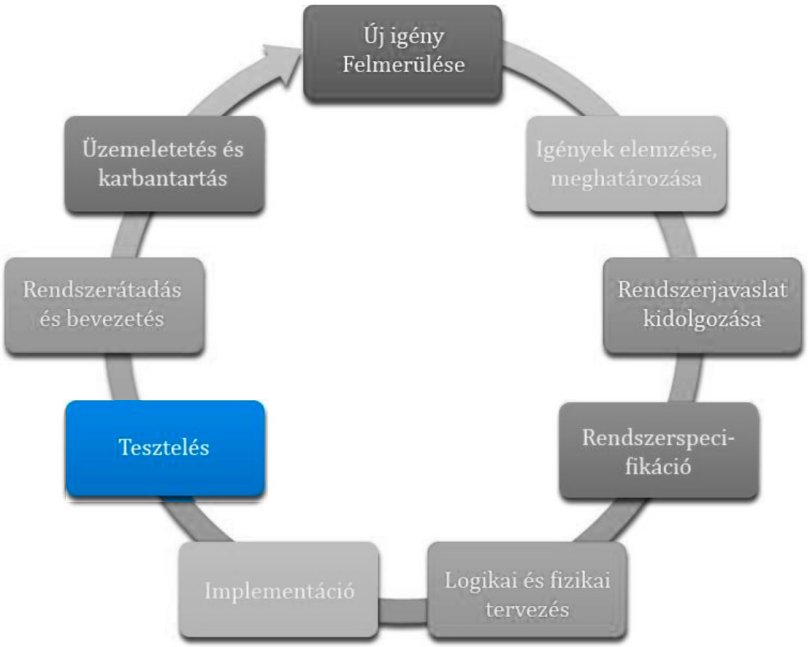



2. ábra: A szoftverfejlesztés életciklusa

# 7 – TESZTELÉS

## Miért kell átadás előtt tesztelnünk?

- A meglévő hibákat még az üzembehelyezés előtt megtaláljuk és felmérhessük az okozott kockázatot
- Emberek írják a kódot, az emberek történetesen hibáznak
- Csak az alaposan letesztelt elemekről tudjuk őszintén elmondani, hogy nincs bennük működést befolyásoló hiba




2. ábra: A szoftverfejlesztés életciklusa

# A TESZTELÉS ALAPELVEI

1. **A tesztelés a hibák jelenlétét jelzi**
   - Képes felfedni azokat
   - A szoftver minőségét és megbízhatóságát növeli
2. **Nem lehet „mindent is” tesztelni**
   - Általában csak a magas kockázatú és prioritású részeket teszteljük
3. **Korai tesztek**
   - Minél korábban jön ki a hiba, annál könnyebb és olcsóbb azt javítani
   - Már az elkészült dokumentációkat is lehet (és érdemes) tesztelni!
4. **Hibák csoportosulása**
   - Nincs végtelen időnk tesztelésre, emiatt a leginkább kitett és érzékeny rendszerekre kell koncentrálnunk
   - A bemenetek esetén használjunk szélső értékeket!
5. **A „féregírtó” paradoxon**
   - Ha ugyanazokat a teszteket futtatjuk, azok egyre kevesebb hibát fognak találni, ezért bővíteni kell őket!
   - Antibiotikumos kezelés esete – A fennmaradó 1% rezisztens, ők a legveszélyesebbek
6. **A tesztelés függ a körülményektől**
   - Másképp tesztelünk a környezet és idő függvényében
7. **A hibátlan rendszer téveszméje**
   - A hibák kijavítása nem egyenlő az igények teljesítésével!

## TESZTELÉSI TECHNIKÁK

- **Black-box**
  - Csak a specifikációk ismertek, a forráskód nem
  - Specifikáció alapúnak is nevezzük
  - Szükség van lefordított szoftverre.
  - Tudjuk például, hogy egy adott bemenetre milyen kimenet az elvárt.
- **White-box**
  - A Forráskód ismeretében végezzük a teszteket
  - Strukturális tesztelésnek is nevezzük
  - A lefedettség – A meglévő teszt esetek a struktúrának mekkora részét fedik le
    - kódsorok
    - elágazások
    - metódusok
    - osztályok
    - funkciók
    - modulok
- **Grey-box**
  - A forráskódnak csak egy része ismert

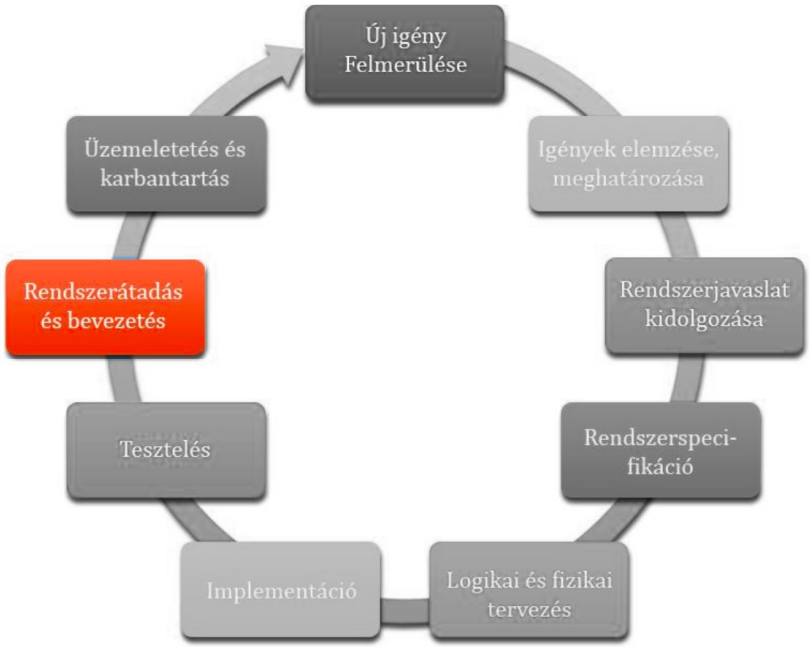
# A TESZTELÉS SZINTJEI

- **Komponensteszt**
  - A rendszernek csak egy adott komponensére tér ki
  - Egységteszt – Funkcionális teszt
  - Modulteszt – Általában nemfunkcionális teszt
- **Integrációs teszt**
  - Komponensek és rendszerek közötti interfészeket teszteljük
  - Komponens integrációs teszt – Komponensek közötti hatások vizsgálata
  - Rendszer integrációs teszt – Rendszerek közötti hatások tesztje
- **Rendszerteszt**
  - A szoftverterméket teszteli
    - Követelményspecifikáció alapján
    - Funkcionális specifikáció alapján
    - Rendszerterv alapján
- **Átvételi teszt**
  - Alpha, Beta, Felhasználói átvételi teszt (UAT) és Üzemeltetői átvételi teszt
- **Regressziós teszt**
  - A módosítás nem okozott-e hibát
  - Komponensteszttől a Rendszertesztig bezárólag találkozhatunk vele

# 8 – BEVEZETÉS ÉS RENDSZERÁTADÁS

- **Mire kell figyelni?**
  - A bevezetéshez szükséges környezet előállt?
  - Oktatások kellenek? Vagy Support szerződés lesz?
  - Bevezetésre szánt idő és átadás „deadline”

  - Dokumentumok:
    - Felhasználói dokumentáció
    - Üzembehelyezési kézikönyv
    - Átadás-átvételi jegyzőkönyv



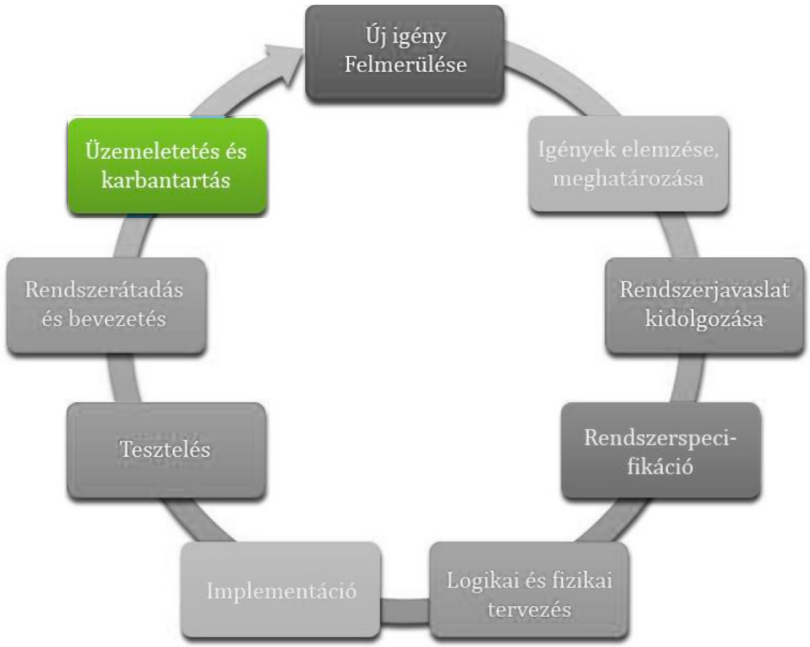

2. ábra: A szoftverfejlesztés életciklusa

# 9 – ÜZEMELTETÉS ÉS KARBANTARTÁS

- **Legfontosabb kérdés: Ki fogja üzemeltetni a rendszert?**
  - SaaS / Managed Service manapság nagy divat
  - Szükséges erőforrások és kommunikációs folyamatok
  - BCP kidolgozása
  - Támogatási szerződés – Nem mindegy, hogy Open Source, vagy Enterprise, nem mindegy, hogy 4 9's vagy 5 9's

- **Dokumentumok:**
  - Felhasználói dokumentáció
  - Üzembehelyezési kézikönyv
  - Átadás-átvételi jegyzőkönyv

- **Rendelkezésre állás – Availability**
  - Megbízhatóság – Reliability
  - Karbantarthatóság – Manintainability
  - Szolgáltatási képesség – Serviceability
  - Biztonság – Security




2. ábra: A szoftverfejlesztés életciklusa

# MÓDSZERTANOK

- Osztályozások
- Alkalmazásuk
- Fő jellemzőik

# MÓDSZERTANOK OSZTÁLYOZÁSA

- **Mi a különbség az SSADM és a SCRUM között?**
  - **Az életciklus fázisainak sorrendje**
    - Lineáris
    - Spirális
    - Iteratív vagy Inkrementális
  - **Előnyben részesített implementációs nyelv**
    - Folyamatorientált
    - Adatközpontú
    - Strukturált (Top-down és Bottom-up)
    - Objektumorientált
    - Szolgáltatásorientált

(Ezeket javaslom a slide-okról megtanulni.)

- **Milyen megközelítést alkalmaz a módszertan**
  - Jól dokumentált
  - Prototípus alapú
  - Rapid
  - Agilis
  - Extrém
  - Fentiek keveréke
- **Mennyire szigorúan veszik a dokumentációt**
  - Lightweight
  - Heavyweight
- **Mi a modell középpontja**
  - Adat
  - Folyamat
  - Követelmény
  - Use-case
  - Teszt
  - Felhasználó
  - Ember
  - Csapat
  - Fentiek keveréke

## MÓDSZERTANOK OSZTÁLYOZÁSA

- **Strukturált módszertanok**
  - Az első módszertanok
  - Feladatot és Implementációt is modulokra bontják
  - Merev sorrend az SDLC-ben
  - Heavyweight
  - Általában Adat- vagy Folyamatközpontú technikák

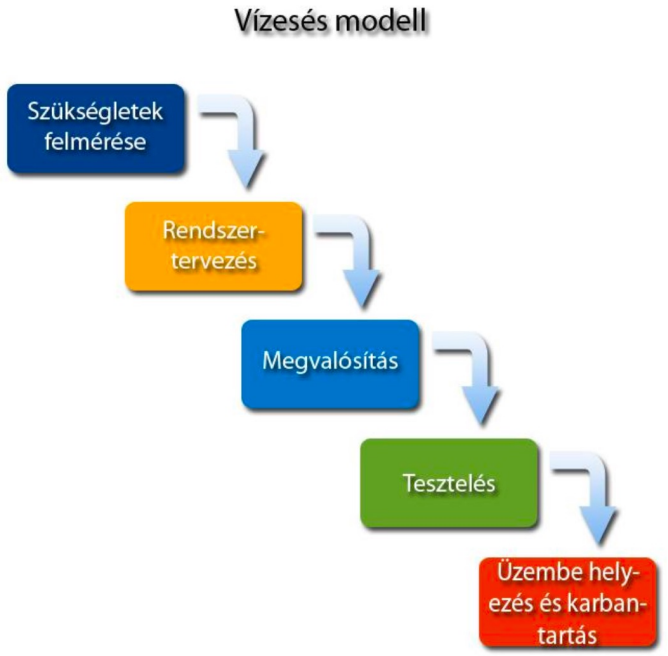
- **Ilyen módszertanok:**
  - Vízesés modell
  - V-modell
  - SSADM

## MÓDSZERTANOK OSZTÁLYOZÁSA

- **A vízesés modell**
  - Lineáris
  - Strukturált
  - Jól dokumentált
  - Heavyweight
  - Nem Agilis

- **Pro:**
  - Erős kontroll a folyamatok felett.
  - Könnyű ütemezés

- **Contra:**
  - Nincs újragondolásra lehetőség
  - Szigorúan lineáris, nincs iteráció



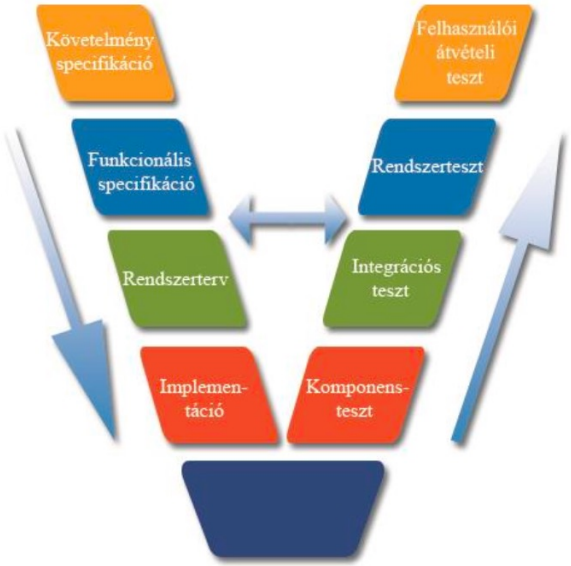

3. ábra: A vízesés modell

## MÓDSZERTANOK OSZTÁLYOZÁSA

- **A V-modell**
  - Lineáris
  - Strukturált
  - Jól dokumentált
  - Heavyweight
  - Tesztközpontú

- **Pro:**
  - Rugalmasabb, mint a vízesés modell
  - Korábban tesztelünk

- **Contra:**
  - Továbbra is nagyon merev
  - A követelmények változását nehezen kezeli



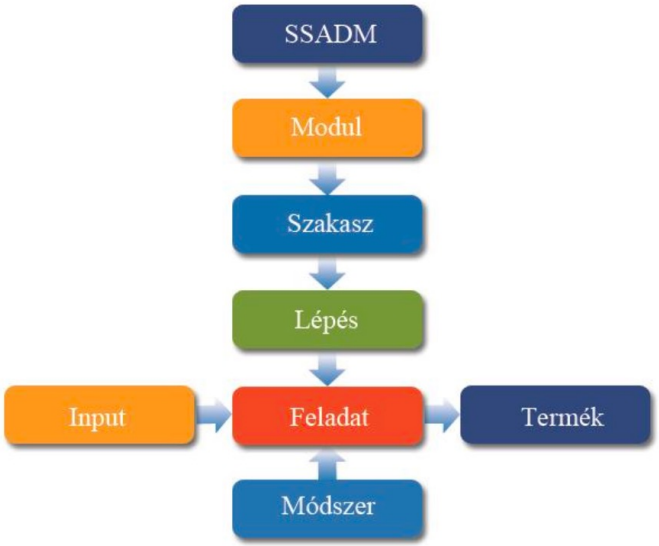

5. ábra: A V-modell

## MÓDSZERTANOK OSZTÁLYOZÁSA

- **SSADM**
  - Lineáris
  - Strukturált
  - Jól dokumentált
  - Heavyweight
  - Adat-, Folyamat- és Termékközpontú
  - Top-down
  - Nem agilis

- **Pro:**
  - A vízesés modell jól kidolgozott változata
  - Államigazgatási standard
  - Nagy projektekhez erős kontrollt ad

- **Contra:**
  - Minden, ami a Lineáris, Strukturált és Heavyweight-ből ered



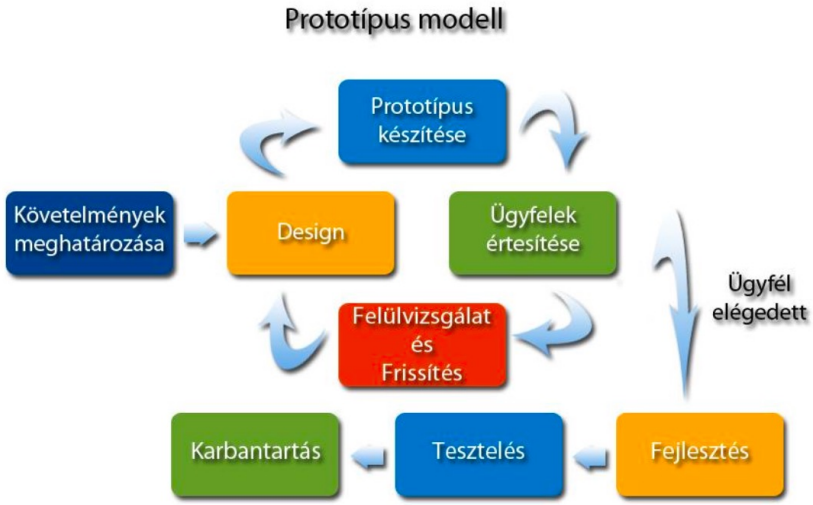

4. ábra: Az SSADM felépítése

# MÓDSZERTANOK OSZTÁLYOZÁSA

- **Prototípusmodell**
  - **Eldobható vagy Evolúciós prototípus**
  - Általában iteratív
  - Általában nem strukturált
  - Gyakran rapid, agilis, esetleg extrém
  - Általában lightweight
  - Általában követelményközpontú

- **Pro:**
  - Félreértések (és abból eredő pluszköltségek) minimalizálása
  - Különösen jó UI-UX kialakításnál

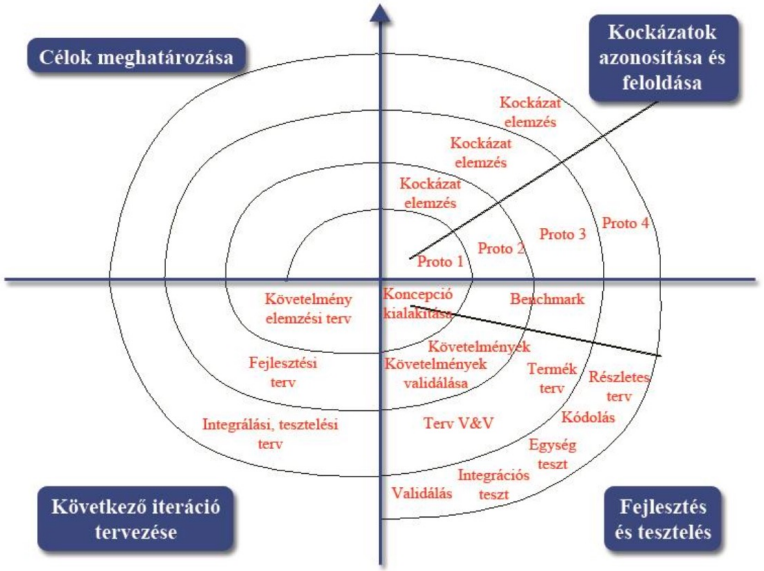
- **Contra:**
  - Confirmation-bias (NEM prototípus-bias)
  - Prototípusból eredő félreértések
  - Prototípus megtartása – Bad practice




6. ábra: A prototípusmodell

## MÓDSZERTANOK OSZTÁLYOZÁSA

- **Spirálmodell**
  - Spirális
  - Használ iterációkat
- **Pro:**
  - Nagy, bonyolult és drága rendszerekhez ajánlott
  - Prototípus- és Vízesés modell ötvözete
  - Lassú, de robosztus modell
- **Contra:**
  - Csak nagy, bonyolult és drága rendszerekhez hasznos
  - Kevésbé rugalmas, főleg a ciklusokban
  - Nehezen levezethető ügyfélnek, fel kell bontani



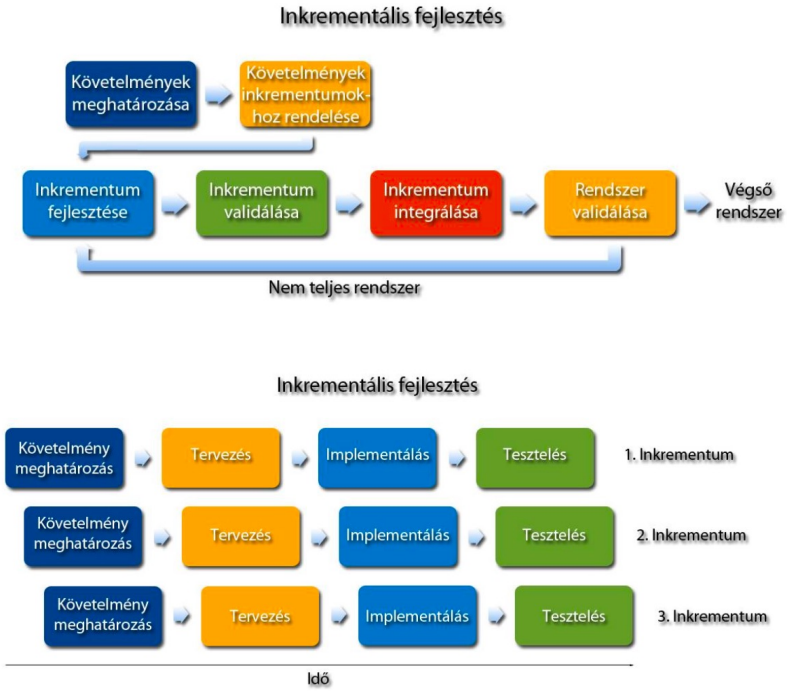

7. ábra: A spirálmodell

# MÓDSZERTANOK OSZTÁLYOZÁSA

- **Iteratív és Inkrementális módszertanok**
  - Iteratívak vagy Inkrementálisak
  - Általában objektumorientáltak
  - Általában prototípus alapúak, de lehet rapid, agilis vagy extrém is
  - Általában lightweight
  - Követelmény- vagy use-case központúak

- **Pro:**
  - Ciklus lépések fedik egymást
  - Gyorsan feloldható félreértések, gyors reagálás új igényekre

- **Contra:**
  - Adott módszertanok sajátos hátrányai
  - Nem megfelelő alkalmazásból eredő káosz
  - Nem megfelelő tervezésből eredő Backlog elúszás



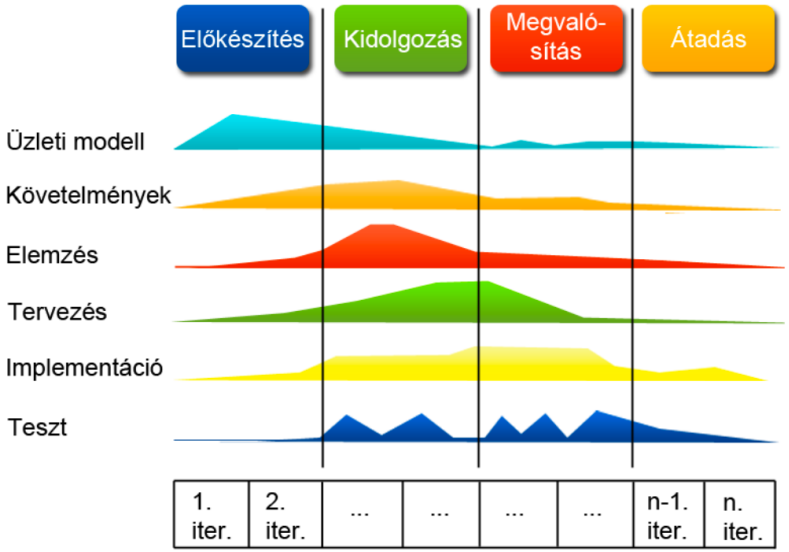

9. ábra: Az inkrementumok

# MÓDSZERTANOK OSZTÁLYOZÁSA

- **RUP – Rational Unified Process**
  - Iteratív
  - Strukturált
  - Use-case alapú
  - Heavyweight

- **Pro:**
  - Jól strukturált megközelítés
  - Iteratív fejlesztés
  - Átláthatóság

- **Contra:**
  - Kis csapatok számára túl komplexnek bizonyulhat
  - Erőforrás igényei sem optimálisak kis csapatok számára
  - Komplexitásából eredően viszonylag magas tudást igényel a megfelelő alkalmazása



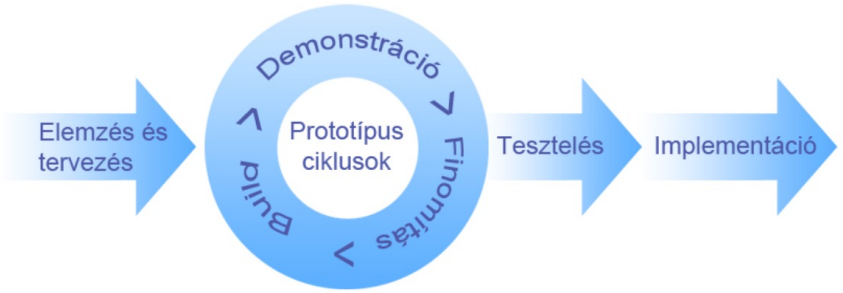

10. ábra: A RUP módszertan fázisai

# MÓDSZERTANOK OSZTÁLYOZÁSA

- **RAD – Rapid Application Development**
  - Általában iteratív
  - Objektum orientált
  - Rapid
  - Lightweight
  - Követelmény központú

- **Pro:**
  - Sok kommunikáció a félreértések minimalizálására
  - Kis rendszerek gyors szállítása
  - Újra felhasználható komponensek támogatása

- **Contra:**
  - Magasan képzett fejlesztőkkel működik jól
  - Végfelhasználó közreműködésén nagyban múlik
  - Nagyobb rendszerek fejlesztése kockázatos vele
  - Nehezen modularizálható rendszerek esetén nem jó választás




11. ábra: A RAD módszertan

# MÓDSZERTANOK OSZTÁLYOZÁSA

- **Agile**
  - Iteratív módszerek egy csoportja
  - Alkalmazkodnak a résztvevők a projekthez és a gyakori változásokhoz
  - Gyakran objektumorientált,
  - Prototípus alapú,
  - Rapid, esetleg extrém,
  - Lightweight
  - Követelmény- és csapatközpontú.

- **Értékrend**
  - Az egyének és interaktivitás fontosabb a folyamatoknál és eszközöknél
  - A működő szoftver fontosabb a terjedelmes dokumentációnál
  - Az együttműködés a megrendelővel fontosabb a szerződéses tárgyalásoknál
  - A változásokhoz való alkalmazkodás fontosabb a tervek követésénél

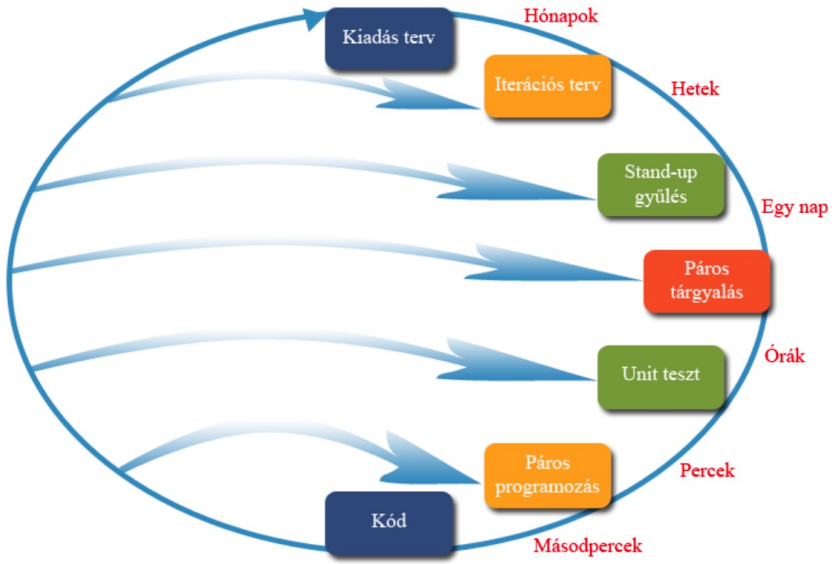
- **Általunk vizsgált Agilis módszertanok:**
  - **Scrum**
  - **XP**

## MÓDSZERTANOK OSZTÁLYOZÁSA

- **XP – Extreme Programming**

- **Tevékenységei:**
  - Programozás
  - Tesztelés
  - Odafigyelés
  - Tervezés

- **Értékrendje:**
  - Kommunikáció
  - Egyszerűség
  - Visszacsatolás
  - Bátorság
  - Tisztelet




12. ábra: Az XP jól bevált módszerei

## MÓDSZERTANOK OSZTÁLYOZÁSA

- **Scrum**
  - Agilis
  - Iteratív (sprint az iteráció)
  - Objektum- vagy Service orientált
  - Prototípus alapú és Rapid
  - Csapatközpontú
  - Ligtweight

- **Pro:**
  - Rendszeres kommunikáció
  - Gyorsan elsajátítható
  - Jól meghatározott szerepkörök
  - Flexibilisen illeszthető fejlesztési folyamatokhoz
- **Contra:**
  - Nem minden Agile igazán Agile
  - Elharapódzó Backlog
  - SOK meeting


13. ábra: A SCRUM módszertan

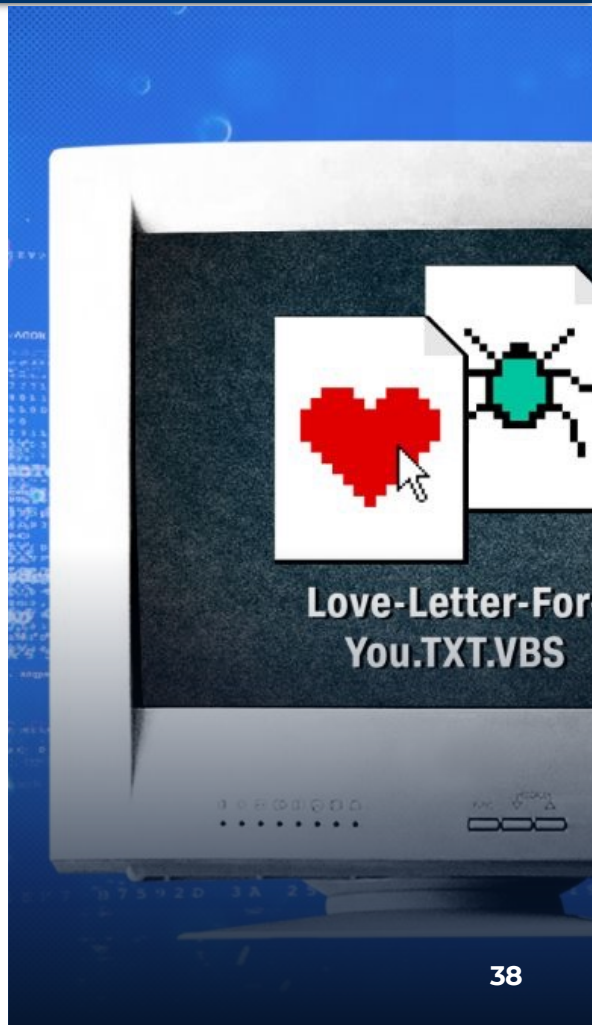
# KOCKÁZATKEZELÉS

## Bevezetés

## KOCKÁZATKEZELÉS

- **Az informatika szerepe napjainkban**
  - A kezdetekben csak apró változások
  - Majd a mindennapok része
  - Mára már civilizációnk alappillére
- **A felhasználással a rendszerek is nőnek**
  - Nő a komplexitás
  - Nő a függőség
  - Nő a kitettség
  - **Nő a kockázat**
- **Az okozott károk**

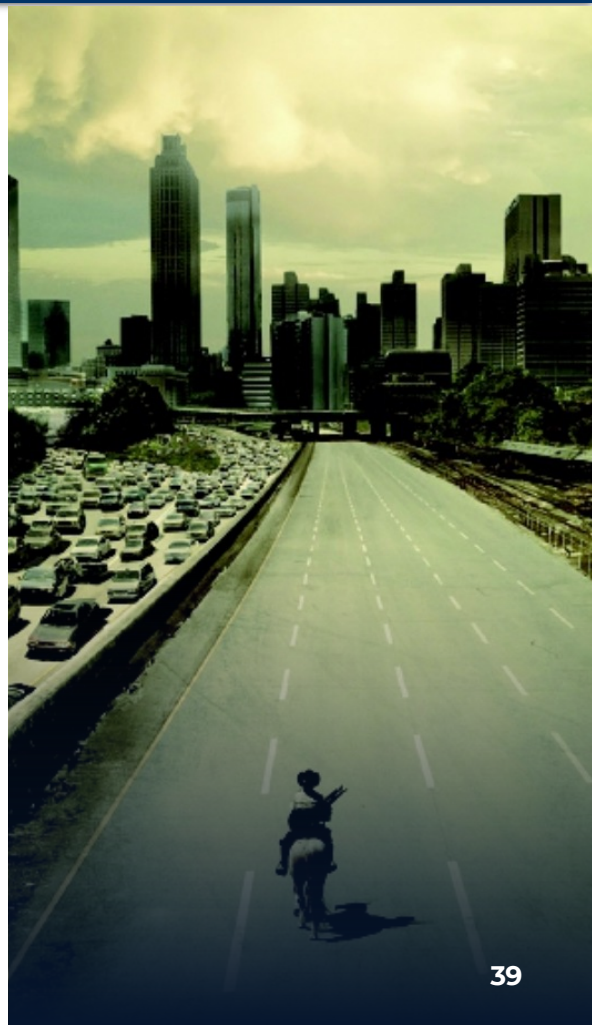
| | |
|---|---|
| „Melissa” (1999) | $ 80 000 000 |
| „ILOVEYOU” (2000) | $ 10 000 000 000 |
| „Mydoom” (2004) | $ 38 000 000 000+ |



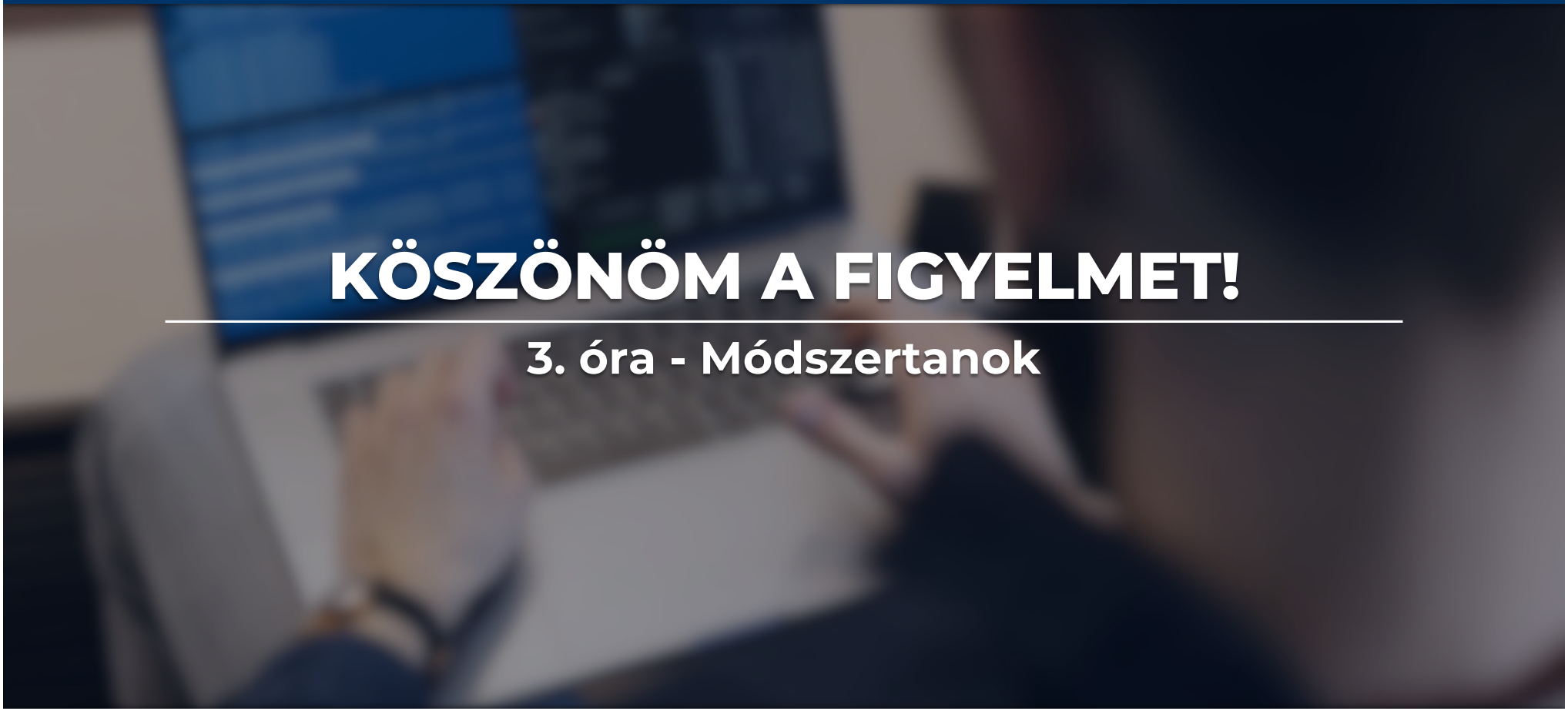
## KOCKÁZATKEZELÉS

- **A kockázat management 4 fő lépése:**
  - A kockázatok azonosítása
  - Értékelése
  - Csökkentése
  - Kommunikációja

- **A kockázat mérőszámai**
  - **A:** Valószínűség
  - **B:** Okozott kár mértéke
  - Kockázat súlyossága **= A x B**



MILTON FRIEDMAN EGYETEM



# KÖSZÖNÖM A FIGYELMET!

## 3. óra - Módszertanok

2024.03.25

Szalai Patrik

szalai.patrik@uni-milton.hu