1. Szekvenciális és iteratív modellek

Szoftverek mérete

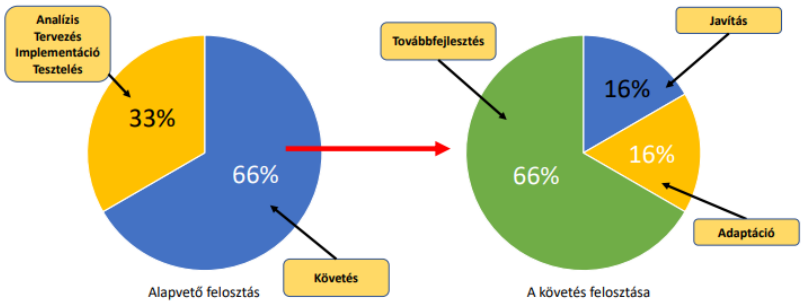
Komplexitás

- A szoftver méretével minimum négyzetesen növekszik annak komplexitása.
- **Tegyük fel, hogy:**
  - x fejlesztés, ahol **négyzetre emelünk** x^2.
  - Az alkalmazást y nap alatt le lehet fejleszteni.
- Méretben egy kis szoftver implementálása $x * y\ \text{nap} = z\ \text{munkahét}$

Egyes fázisok időigénye

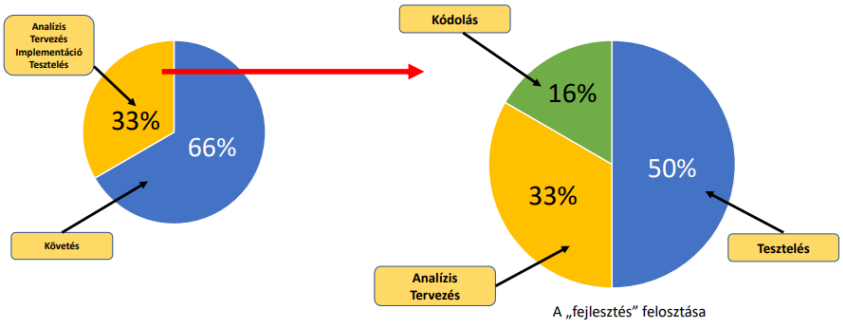
- **Tapasztalati adatok alapján**

Költségarányok



- Vehetjük időnek is, hogy melyik szakasz meddig tart.
- **Követés jelentése**
  - Továbbfejlesztési igényeknek a kielégítése

Fejlesztési költségarányok



Jelentések

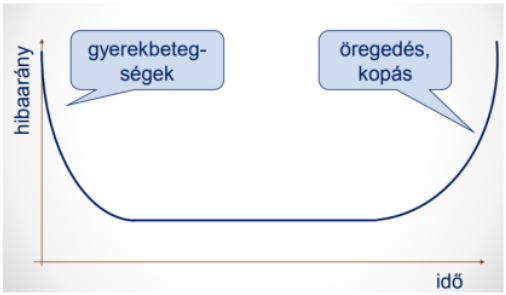
- **Analízis:** A probléma megértése a megrendelő elmondása alapján, tehát specifikáció.
- **Tervezés:** Specifikációból konkrét tervek, például adatbázis terv, rendszerterv, OOP osztályok tervezése, rétegek kialakítása.
- **Implementáció:** Tervezés alapján a szoftver lekódolása.
- **Tesztelés:** Elkészült szoftver leteszttelése, Test-Driven-Development (TDD) segítségével.
- **Javítás:** Fejlesztői részről minden működik, de még mindig vannak hibák, ez általában a rossz specifikáció miatt lehet. (Valós felhasználás alapján jön elé a probléma.)
- **Adaptáció:** Megrendelő adottságaihoz alakítás (például van már szervere, adatbázisa).

Hagyományos termékek összehasonlítása szoftver termékekkel

Szoftverkrízis

- Gyenge hardver
- Fejlesztői eszközök nem voltak
- Monolitikus programozás (spagetti kód)
- Team munkához semmilyen eszköz nincsen (manual merge)
- Konfiguráció változást nem támogatja a szoftver
- **Megoldás**: Új paradigmák (pl.: OOP), modularizálhatóság megjelenése

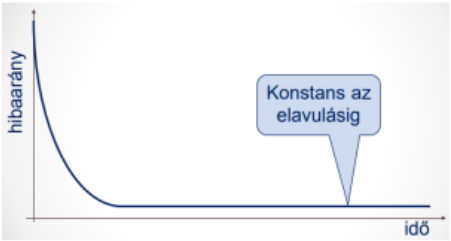
Hagyományos termékek előállítása



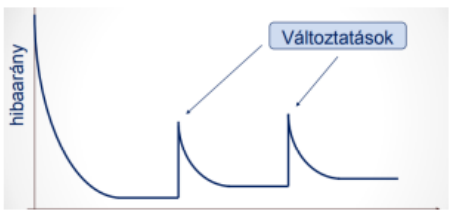
- **Lépések**
  1. **Analízis:** Követelmények meghatározása
  2. **Vázlatos tervezés:** Látványtervek
  3. **Részletes tervezés:** Műszaki tervek
  4. **Fizikai megvalósítás:** Maga az implementáció, végrehajtás
- **„Gyerekbetegségek”**
  - Tervezési, gyártási hibák
  - Hibás alapanyagok
- **Hibák okai**
  - Öregedés, szerkezeti elváltozások

Szoftver termékek

- Nincs öregedés, kopás



- Helyette állandó módosítási igény, újabb funkciók, amik újabb hibákat eredményezhetnek és ezek a hibák összeadódnak.



Megrendelő és fejlesztő

- Kommunikációs problémák, mert a megrendelő x funkciót képzelt el, de mégis y funkció lett a vége.
- **Megoldására az életciklus modelleket használhatjuk.**

Életciklus modellek

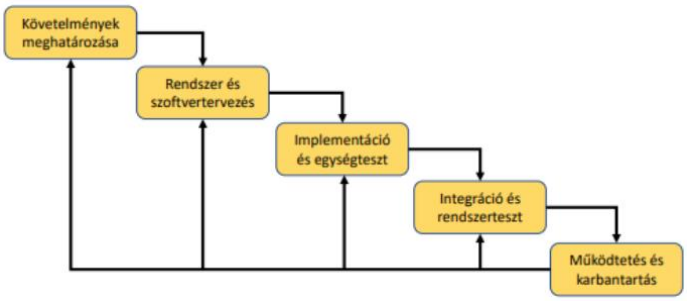
- **Céljuk a szoftverfejlesztés lépéseinek meghatározása.**

Szekvenciális modellek

- Lépésről-lépésre működnek, emiatt nincs visszalépési lehetőség.

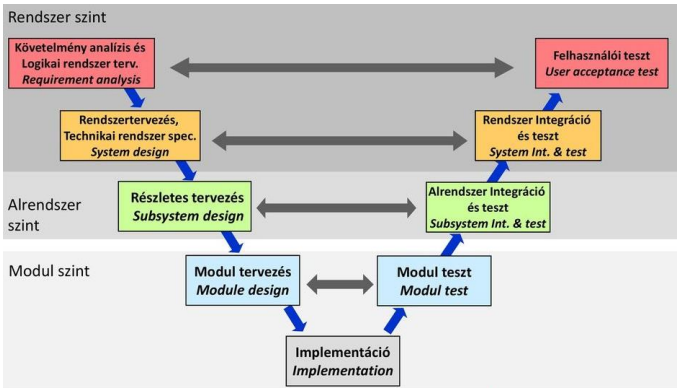
Vízesés modell

- Akkor hasznos, ha a követelmények jól ismertek és csak nagyon kis változások lehetségesek a fejlesztéskor.
    - Kevés üzleti rendszernek vannak stabil követelményei.
    - Főleg nagy rendszerek fejlesztésekor használják, ahol a fejlesztés több helyszínen történik.
- **Problémái**
    - Minden a specifikáció minőségétől függ.
    - Későn lát a megrendelő működő programot.
    - Kezdeti bizonytalanságot nehezen kezeli.
    - Tesztelés szerepe nem eléggé hangsúlyos.
- **Fázisai**



V-modell

- Azért nevezik V-modellnek, mert két szára van: **Fejlesztési** és **tesztelési** szár.
- Vízesés modell kiegészítése teszteléssel.
    - Először végre kell hajtani a fejlesztés lépéseit, ezután jönnek a tesztelés lépései.
    - Ha valamelyik teszt hibát talál, akkor vissza kell menni a megfelelő fejlesztési lépésre.
- Szigorú dokumentálást követe és nem küszöböli ki a vízesés modell problémáit.

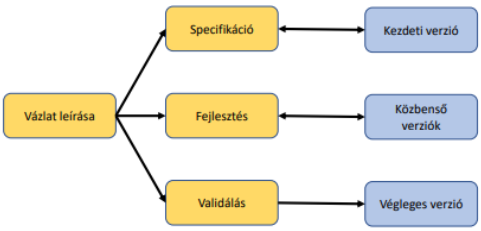


Iteratív modellek

- Ciklikusan működnek, fázisokon keresztül.

Evolúciós modell

- Kifejlesztünk egy kezdeti implementációt.
- Véleményeztetjük a felhasználókkal.
- Addig finomítjuk, míg el nem érjük a kívánt rendszert.
- **Problémák**
  - Nehezen menedzselhető a folyamat
  - Minőségbiztosítási problémák
  - Speciális eszközök és technikák igénye

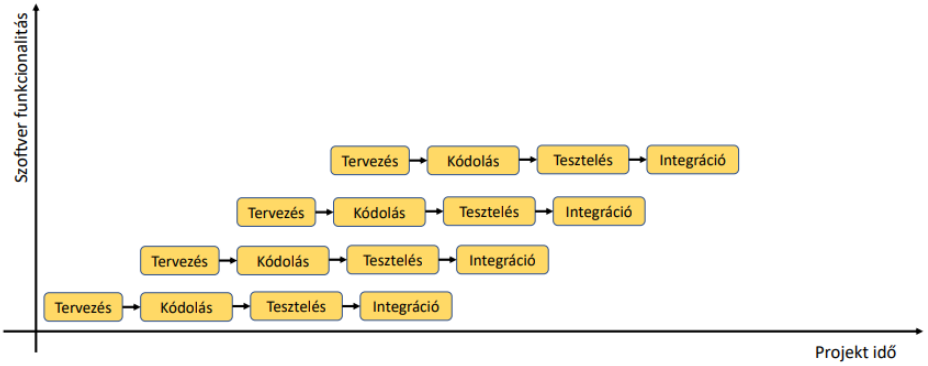


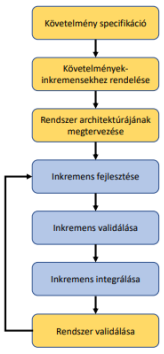
Újrafelhasználás-orientált fejlesztés (nem életciklus modell)

- Fejlesztési idő nagy mértékben lerövidíthető
- Olcsóbb a fejlesztés
- Rendszer minősége és megbízhatósága jobb, mert ellenőrzött komponenseket építünk be.
- Nagy library gyűjteménynél nehéz a pontos komponens kiválasztása.

Inkrementális fejlesztés

- Rendszert kisebb egységekre bontjuk.
- Minden egység önálló fejlesztés validálás, integrálás folyamatot kap.
- Minden inkremens külön egyeztetett, elkészülte után a megrendelő használatba veheti.
- Cél a komplexitás csökkentése.
- Alrendszerek tipikusan inkremensek.
- Lehetővé válik a pipeline-alapú fejlesztés.





- **Előnyei**
  - Szoftver már menetközben használhatóvá válik a megrendelő számára.
  - Kritikus követelmények teljesülnek először.
  - Kisebb a kockázata a projekt kudarcának, biztonságos fejlesztési koncepció.
  - A legkritikusabb funkciókat teszteljük legtöbbet.

RAD modell

- Rapid Application Development
- Vízesés modell high-speed adaptációja
- **Építőelemei**
    - Újrafelhasználás-orientált fejlesztés
    - Komponensekre bontás
    - Kódgenerálás
- Sokoldalú, párhuzamosan dolgozó teamek, akár a megrendelő szakemberei is részei lehetnek.
- **Hátrányok**: Nagy projektnél hatalmas humán erőforrás igény és ha nehezen modularizálható a rendszer, akkor nem működik.