# Szoftvertechnológia és -technikák

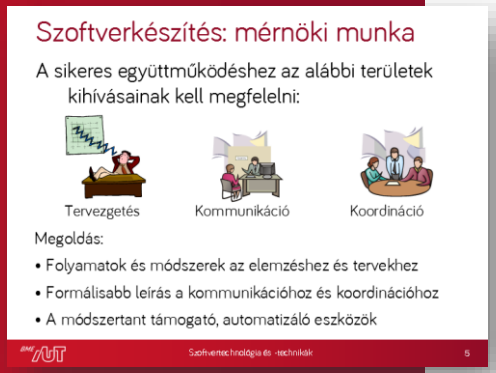
## 11. Előadás - A vízesés és a RUP módszertan

BME AUT
Automatizálási és
Alkalmazott
Informatikai Tanszék

“You’ve got to be very careful if you don’t know where you’re going, because you might not get there.”

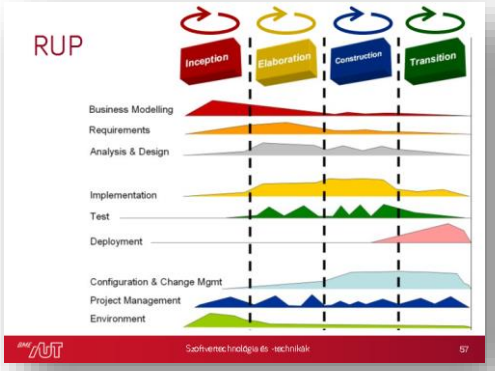
// Yogi Berra

# Tartalom


Szoftverkészítés: mérnöki munka
A sikeres együttműködéshez az alábbi területek kihívásainak kell megfelelni:
Tervezgetés
Kommunikáció
Koordináció
Megoldás:
• Folyamatok és módszerek az elemzéshez és tervekhez
• Formálisabb leírás a kommunikációhoz és koordinációhoz
• A módszertant támogató, automatizáló eszközök
Szoftvertechnológia és -technikák
5


Vízesés modell
A vízesés modell minden szoftverfejlesztési modell őse.


Változás kezelés
Szoftvertechnológia és -technikák
32


RUP
Inception
Elaboration
Construction
Transition
Business Modelling
Requirements
Analysis & Design
Implementation
Test
Deployment
Configuration & Change Mgmt
Project Management
Environment
Szoftvertechnológia és -technikák
57

# Szoftverkészítés: mérnöki munka

A sikeres együttműködéshez az alábbi területek kihívásainak kell megfelelni:

Tervezgetés


Kommunikáció

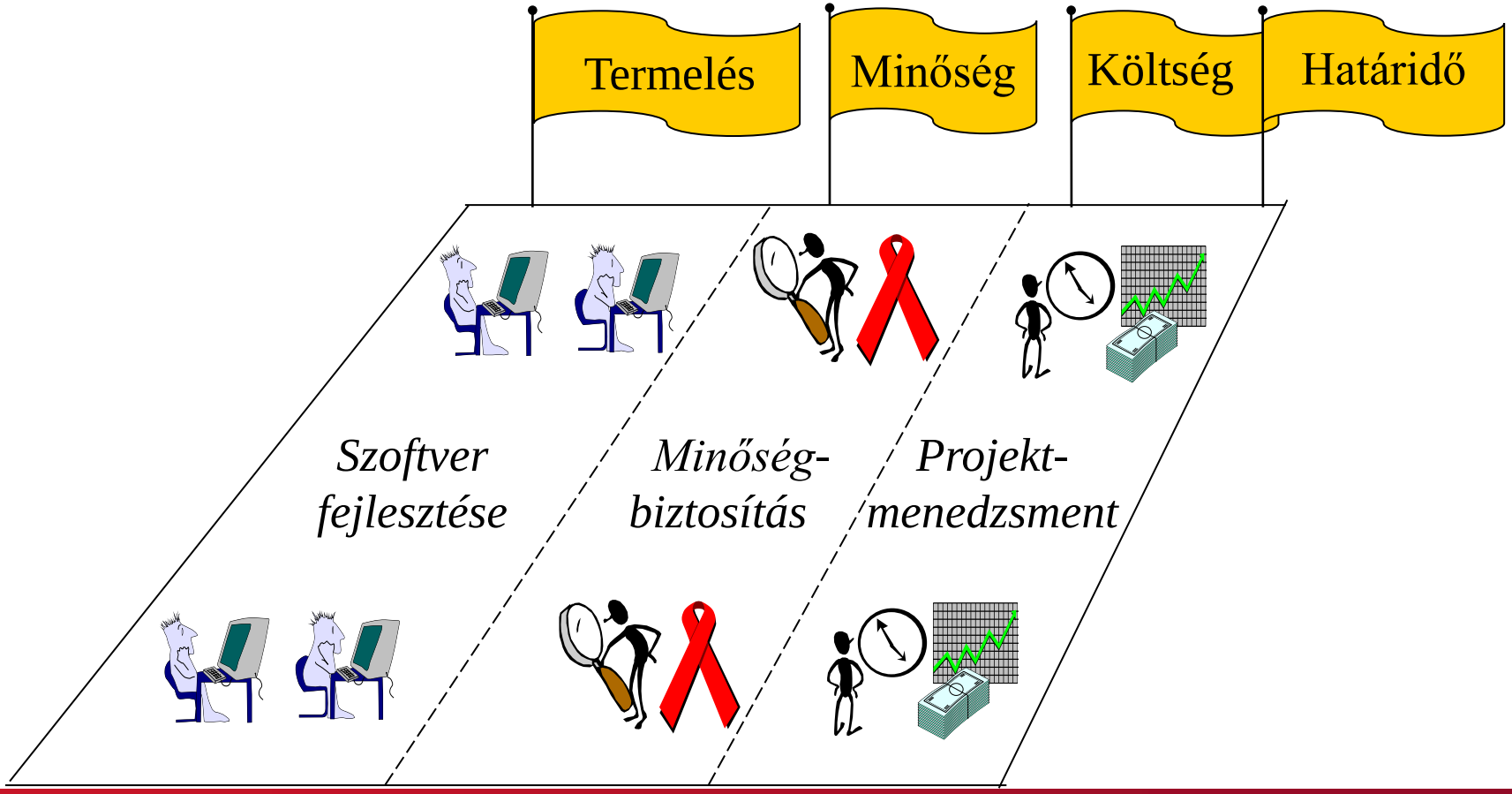
Koordináció

Megoldás:

- Folyamatok és módszerek az elemzéshez és tervekhez
- Formálisabb leírás a kommunikációhoz és koordinációhoz
- A módszertant támogató, automatizáló eszközök

# A szoftverfejlesztés elemei

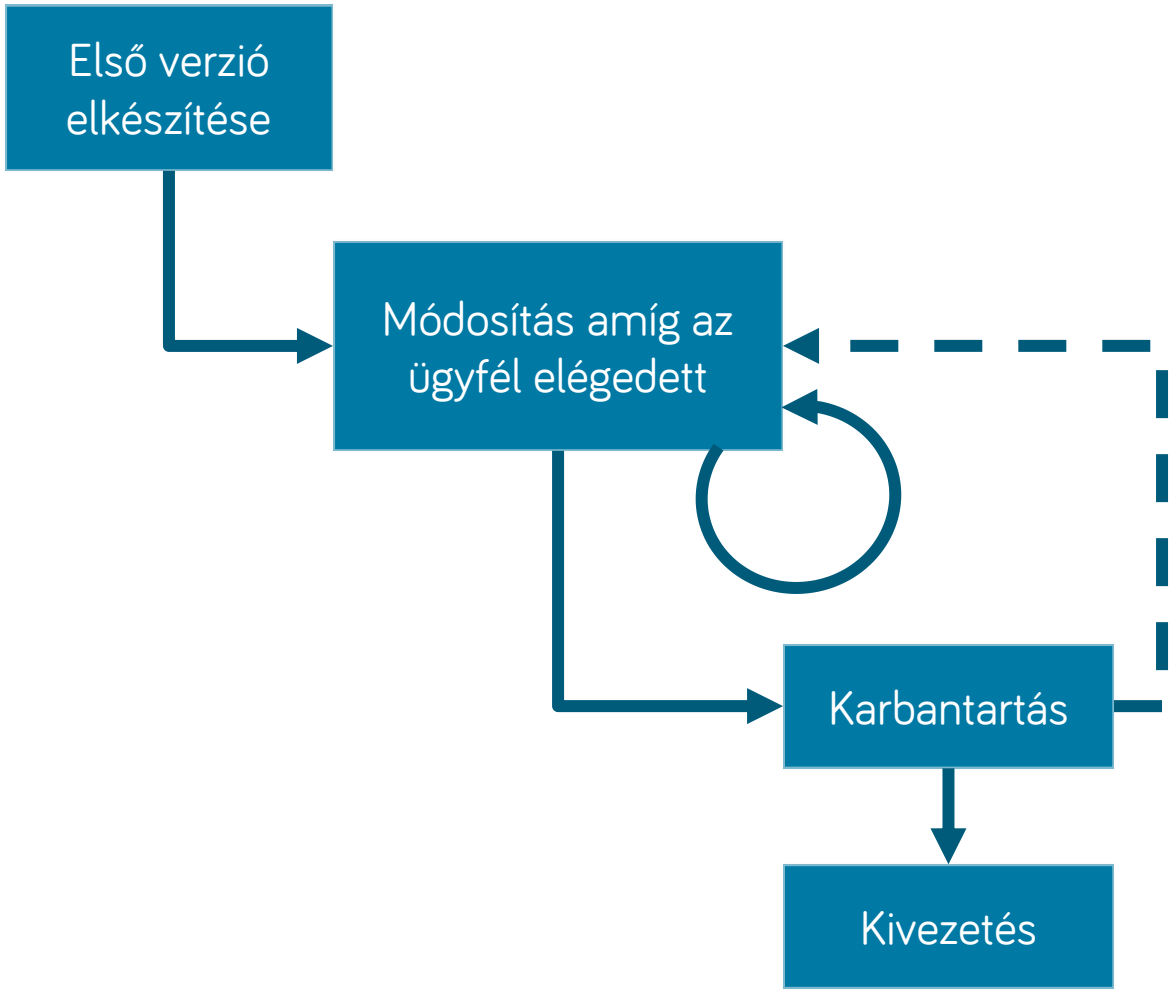
- A szoftverfejlesztési folyamat az alábbi aktivitásokból áll:



# Szoftverfejlesztési folyamat modell

- **Tevékenységek strukturált halmaza**, amelyekkel létrehozunk egy szoftver rendszert
  - specifikálunk, tervezünk, kódolunk és tesztelünk
- A szoftver folyamat modell a folyamat absztrakt leírása, ami segít eldönteni a résztvevőknek:
  - Milyen munkát csináljunk meg
  - Milyen sorrendben
  - A modell gondolkodni segít, nem merev előírások halmaza
  - Minden projekt saját egyedi folyamat mentén halad

# Build and Fix



# Build and Fix

- Nem kielégítő modell értelmes méretű projekt esetén
- A költségek jelentősen megnőnek nagy projekt esetén
- Tipikusan csúszik a szállítás
- Nincs dokumentáció
- Karbantartás nagyon nehéz specifikáció és tervek nélkül
- Kis projektek esetén működik: kis menedzsment overhead

# A szoftverfejlesztési folyamat

- Sokfajta különböző folyamat, de mind tartalmazza az alábbi tevékenységeket:
  - Specifikáció – mit kell tudnia a rendszernek
  - Terv és implementáció – a rendszer elkészítése
  - Validáció –ezt akarja-e a megrendelő?
  - Evolúció – a megrendelői igények változásakor változtatjuk a rendszert
- A folyamat leírások kiterjedhetnek az alábbiakra is:
  - Termékek, amelyek egy folyamat eredményeként megszületnek
  - Szerepek, amelyek az emberek felelősségét tükrözik
  - Aktivitások elő-és utófeltételei

# Terv alapú és agilis modellek

- Terv alapú folyamatok esetén az összes tevékenység előre meg van tervezve és a haladást ehhez a tervhez mérjük
- Agilis esetben a tervezés inkrementális és könnyebb igazítani a folyamatot a változó megrendelői igényekhez
- Gyakorlatban a legtöbb projekt mindkét megközelítésből tartalmaz elemeket
- **Nincs kategorikusan jó vagy rossz folyamat**

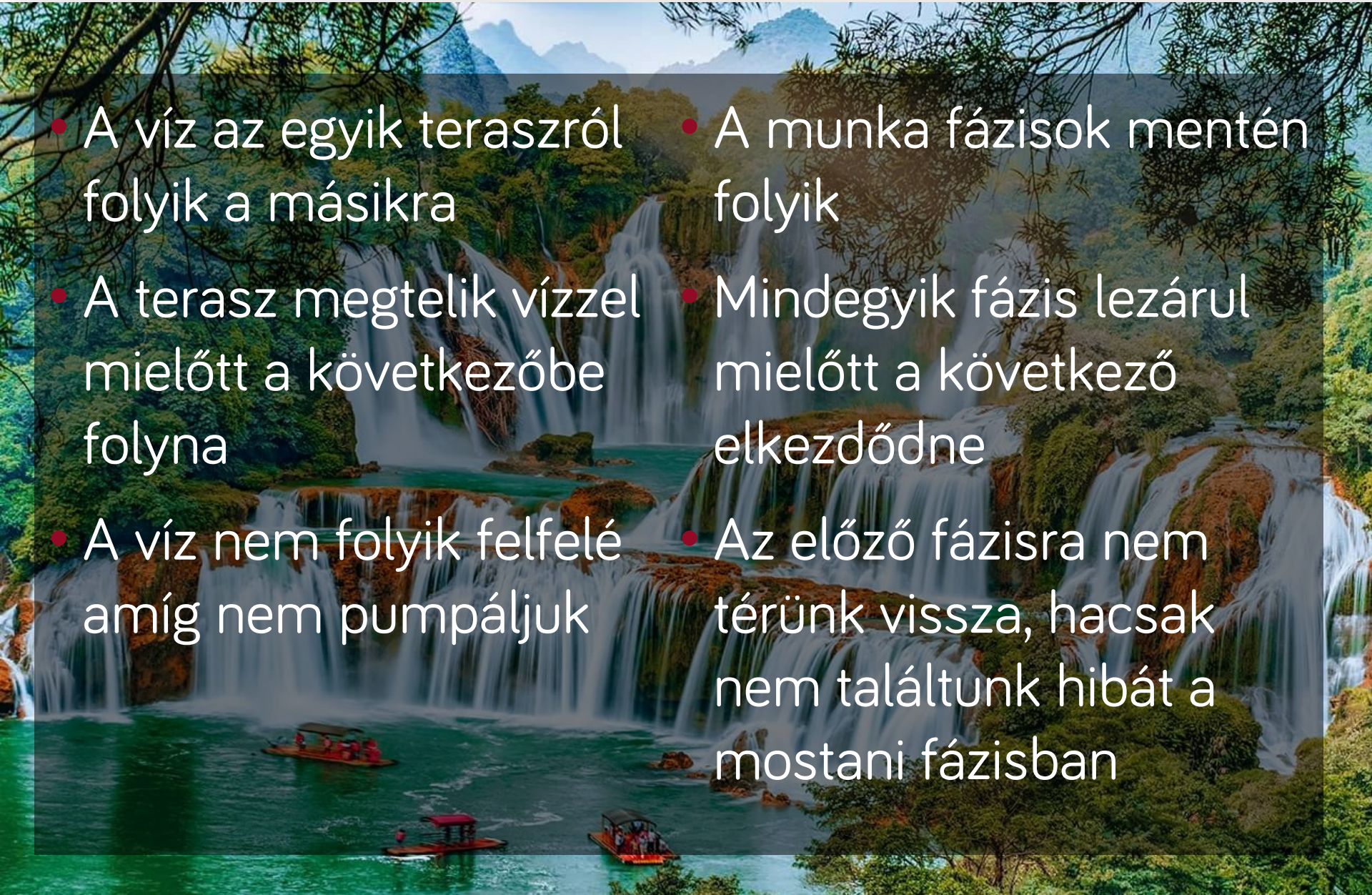
# Különböző megközelítések

- Vízesés modell
  - Terv alapú modell, különálló specifikációs és megvalósítási fázisok.
- Inkrementális modell
  - Átfed a specifikáció, fejlesztés és validáció. Lehet terv alapú vagy agilis.
- Újrafelhasználás központú módszer
  - A rendszert meglévő komponensekből építjük össze. Lehet terv alapú vagy agilis.
- A gyakorlatban a legtöbb nagy rendszer mindegyik megközelítést használja

# Vízesés modell

A vízesés modell minden szoftverfejlesztési modell őse.

# Vízesés modell



- A víz az egyik teraszról folyik a másikra
- A terasz megtelik vízzel mielőtt a következőbe folyna
- A víz nem folyik felfelé amíg nem pumpáljuk
- A munka fázisok mentén folyik
- Mindegyik fázis lezárul mielőtt a következő elkezdődne
- Az előző fázisra nem térünk vissza, hacsak nem találtunk hibát a mostani fázisban

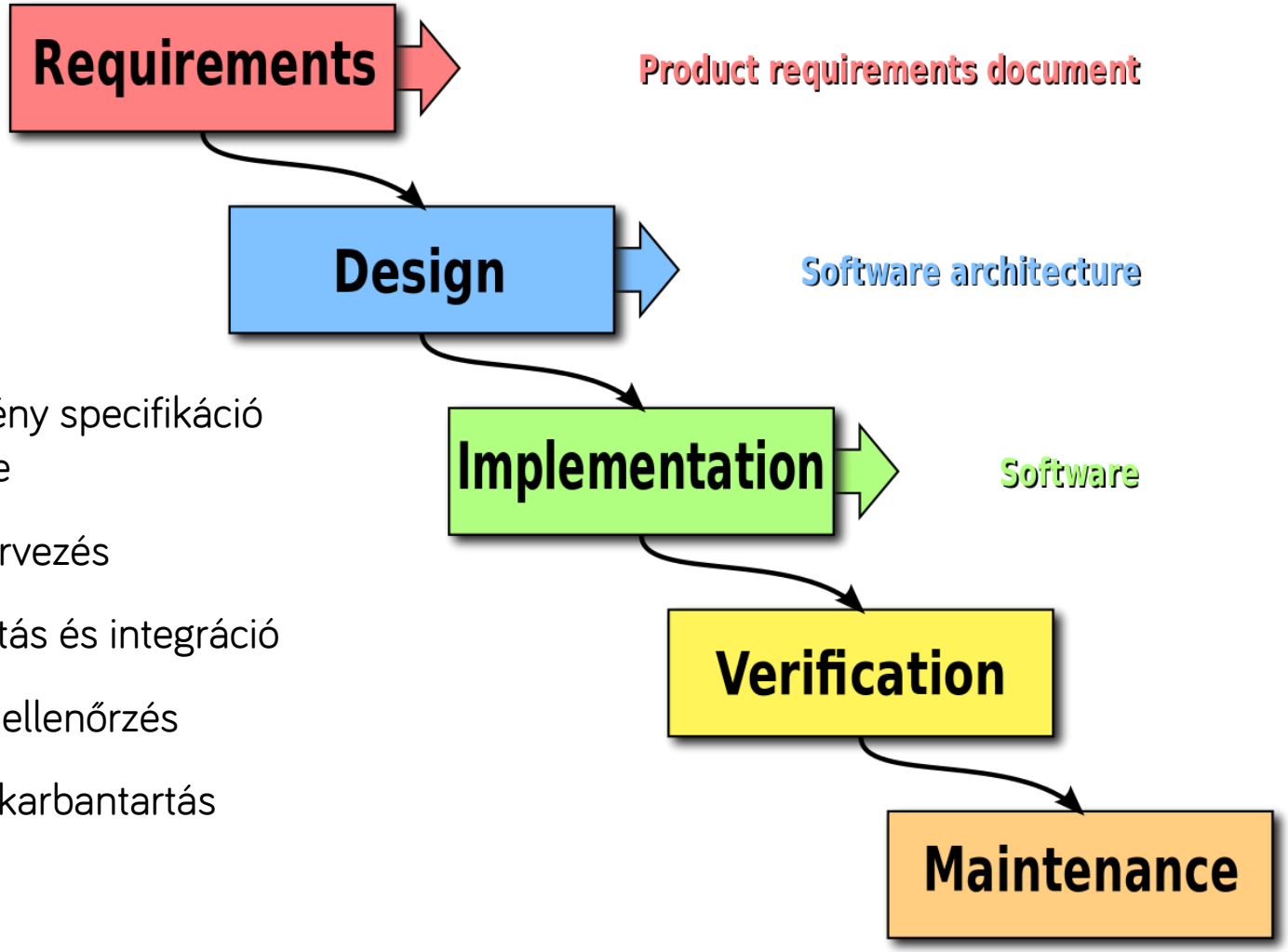
# Vízesés modell

- Az elmúlt 50 év leggyakrabban használt modellje, ma is nagyon gyakori, főleg hazánkban
- Lineáris modell
  - Egymást követik a lépések
  - Akkor kezdünk el egy új fázist, ha az előző befejeződött
- **Iteratív vízesés** modell: van visszalépés az előző fázisra, ha hibát látunk
- Megközelítése minden modellben megjelenik

# Vízesés modell

- **Példa:** A BKV megbízza a szoftverfejlesztő csapatunkat egy digitális jegyrendszer fejlesztésével
- Jól rögzített határideje van:
  - A következő év december 31-re tűzik ki.
- Alfától omegáig minden ránk van bízva:
  - Mindent nekünk kell kitalálni, megvalósítani, telepíteni, átadni, karbantartani
- A megrendelővel a projekt elején és végén találkozunk

# Vízesés modell lépései




1. Követelmény specifikáció elkészítése
2. Szoftvertervezés
3. Megvalósítás és integráció
4. Tesztelés, ellenőrzés
5. Telepítés, karbantartás

# 1. lépés: Követelmények 1

- Részletes közös képet alakít ki a Megrendelő és a Szállító között az elkészítendő termékről
- Akkor lépünk tovább, amikor elfogadásra került a specifikáció

# 1. lépés: Követelmények 2

- Egy specifikáció elemei:
  - A projekt célja, helye a vállalatnál
  - Terminológia, definíciók
  - Felelősök
  - Felhasználói csoportok
  - Feltételezések, kényszerek, elő-követelmények, függőségek
  - Funkcionális követelmények – folyamatok, interfészek stb. tételes felsorolása és részletezése
  - Nem funkcionális követelmények – teljesítmény, biztonság, bővíthetőség, formátum, …
  - Becsült határidők
  - Átadás / átvételi tesztek

# 2. lépés: Tervezés 1

- Koncepcionális tervek:
  - Architektúra, felhasznált keretrendszer
  - Interfészek, modul határok
  - Részletes adatbázisterv
  - Folyamatok leírása
  - Képernyőtervek, navigációs térkép, stb.
  - Használati minták
  - Biztonsági, tesztelési tervek

# 2. lépés: Tervezés 2

- A tervezés fázisa után a további módosítások már költségesek!
- A vízesés modell alapja a jó terv.
  - Megrendelő által validálható tervek – például képernyő képek, folyamat ábrák!
  - Megrendelői oldalon valódi ellenőrzés a munkát végzők által (kulcsfelhasználók)!
  - Nehézséget és kockázatot jelent, ha a megrendelő közben egy transzformációban van benne.

# 3. lépés: Megvalósítás és Integráció

- Ebben a lépésben indul el a fejlesztés, kódolás
- Munka dekomponálása, fejlesztési tervek
- Fejlesztői, unit tesztelés
- Többnyire a leghosszabb idő a elkészítési folyamatban

# 4. lépés: Tesztelés

- A minőség ellenőrzők megvizsgálják a szoftvert, hogy a specifikáció szerint működik-e?
  - > Verifikáció
- A hibák visszajutnak a fejlesztőkhöz és javítás után a tesztelés elölről indul
- Gyakran használunk átmeneti állapotot: béta verzió, meghívott felhasználók stb.
  - > Validáció

# 5. lépés: Telepítés, karbantartás

- Telepítés, terjesztés
- Karbantartási / továbbfejlesztési fázis
  - Hibajavítás, tesztelés, …
- A karbantartás a legdrágább fázis
  - A hibajavítás kiemelten költséges lehet

# A vízesés modell előnyei 1

- Egyszerű, könnyen betartható kezdő csapat számára is
- A változások a dokumentumokon keresztül jól követhetőek
- A megrendelő és a szállító a projekt elején megállapodnak, tisztázzák a fejlesztés mérföldköveit
- Nem szükséges a megrendelő folyamatos jelenléte
- Hamarabb megtalálható hibák -> olcsóbb megoldás
- A követelmények változatlanok

# A vízesés modell előnyei 2

- Menedzsment oldalról jól kontrollálható
- A projekt fejlődése könnyen mérhető és nyomonkövethető
  - Melyik fázisban vagyunk éppen
  - Tudjuk, jó becslésünk van arról, hogy mennyi van hátra
- Nem kell mindenkinek folyamatosan a projekten dolgoznia
  - Pl. A tesztelőknek a tesztelés fázisában kell dolgozniuk, előbb nem vagy kevesebbet

# A vízesés modell hátrányai 1

- A „követelmények nem változnak” nem realisztikus elvárás
- Nem az készül el, amire szükség van
  - **Nem kapunk visszajelzéseket a termékről** addig, amíg az teljesen el nem készült
  - A papíron létező specifikációt az ügyfél nem vagy félre érti
- A megjelenő új követelmények koncepcionális változásokat indukálhatnak, ami újratervezést igényelhet
  - Jelentőssé válhat a kidobott idő, energia

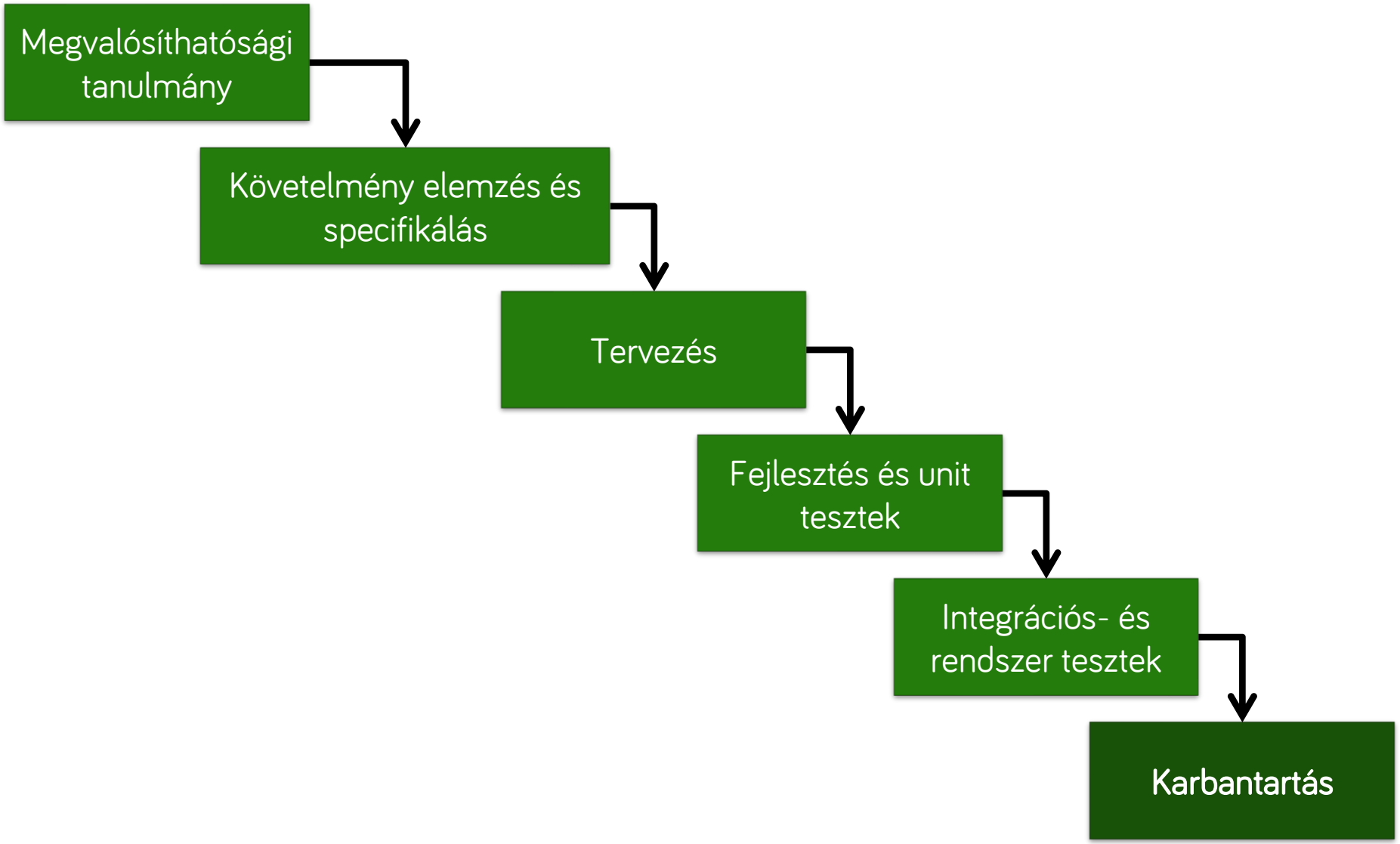
# A vízesés modell hátrányai 2

- Nehéz az összes erőforrást pontosan becsülni
- Könnyen csalóka lehet a haladás látszata
- A tervezők gyakran nincsenek tisztában az implementációs részletekkel
  - Néha egyszerű tervváltoztatás helyett bonyolult implementáció készül el
- A tervező eszközök nem integrálódnak jól az implementációt segítő eszközökkel
- Túl hosszú lehet a teljes fejlesztési fázis, nincs részfelhasználás, visszajelzés
- Az integráció egy nagy feladat a munka végén

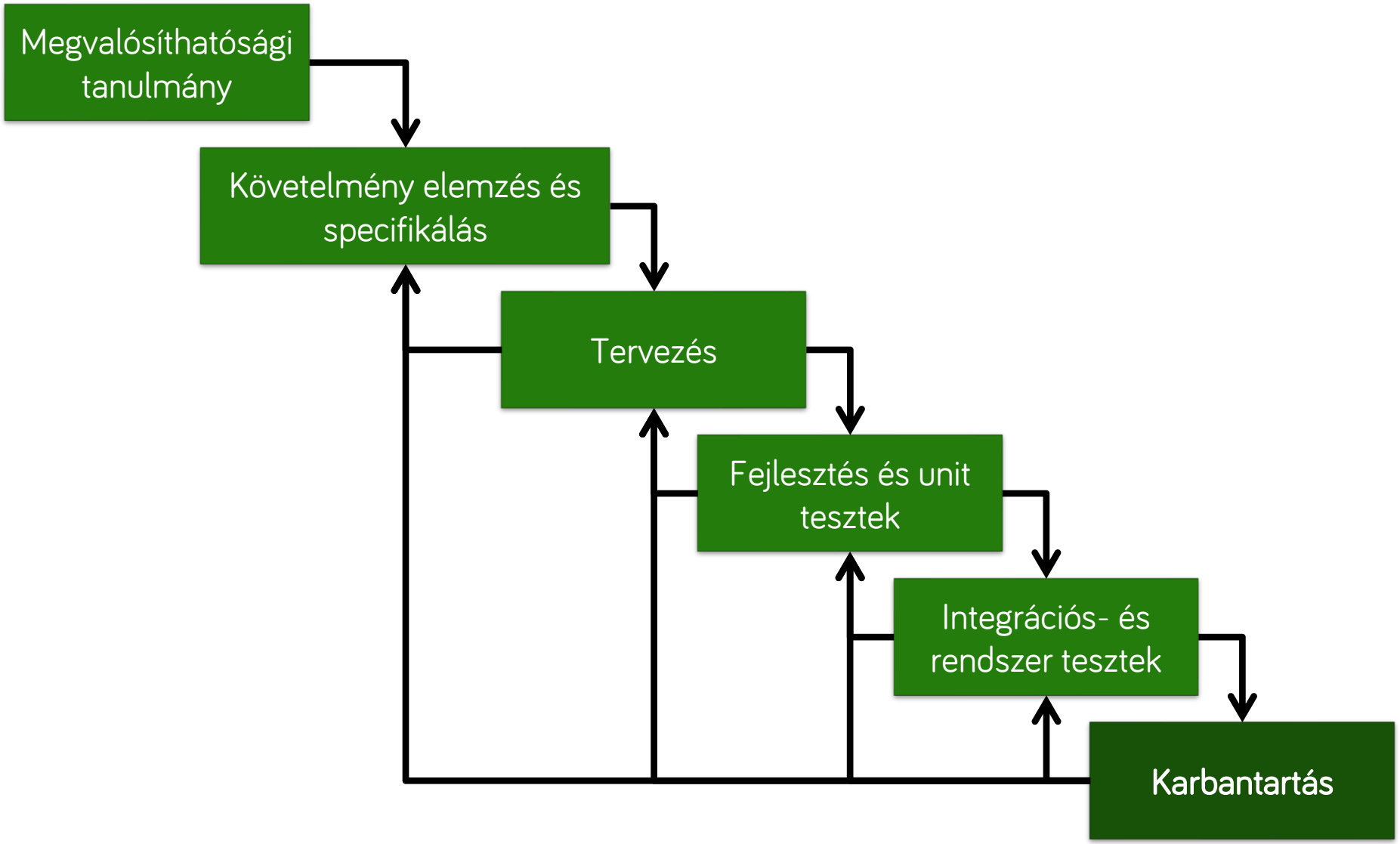
# Mikor használjuk?

- Új verzióját készítjük egy meglévő alkalmazásnak
  - Funkcionális bővítés vagy platform váltás
- Ismertek és változatlanok a követelmények
- Jól ismert technológiákkal dolgozunk
- Fix projekt alapú a megállapodásunk
- A megrendelő csak a mérföldköveknél van jelen
- Pontos határidővel dolgozunk
  - Például törvényi előírások

# Klasszikus vízesés modell



# Iteratív vízesés modell



# Változás kezelés

# Változások kezelése

- A változás elkerülhetetlen a legtöbb projektben
  - Az üzleti változások új követelményeket szülnek
  - Új technológiák új lehetőségeket teremtenek
  - A platform váltás rendszer változást hoz magával
- A változás újraírást jelent, aminek költsége kettős
  - A követelmények újra elemzését
  - A változás implementálását

# Az újraírás költségének csökkentése

- A változtatás elkerülése, ahol a folyamatba beépíthető olyan tevékenység ami előre jelzi a lehetséges változtatást
  - Például prototípusokon keresztül hamarabb kaphatunk visszajelzést az ügyféltől
- A változtatás költségének csökkentése a folyamat ilyen irányú optimalizációjával
  - Tipikusan inkrementális fejlesztés segítségével

# Prototipizálás

- A prototípus a rendszer egy első verziója, ami koncepciók, dizájn alternatívák bemutatását célozza
  - MVP: Minimum Viable Product
  - Segíti a követelmény felmérést és validációt
  - UI dizájn kialakításában az alternatívák felfedezésében
  - Megmutatja a fejlesztők kompetenciáját

# A prototipizálás lehetséges előnyei

- Jobban használható szoftver
- Közelebb áll ahhoz, amire a felhasználónak szüksége van
- Jobb dizájn
- Jobb karbantarthatóság
- Kisebb fejlesztési költség

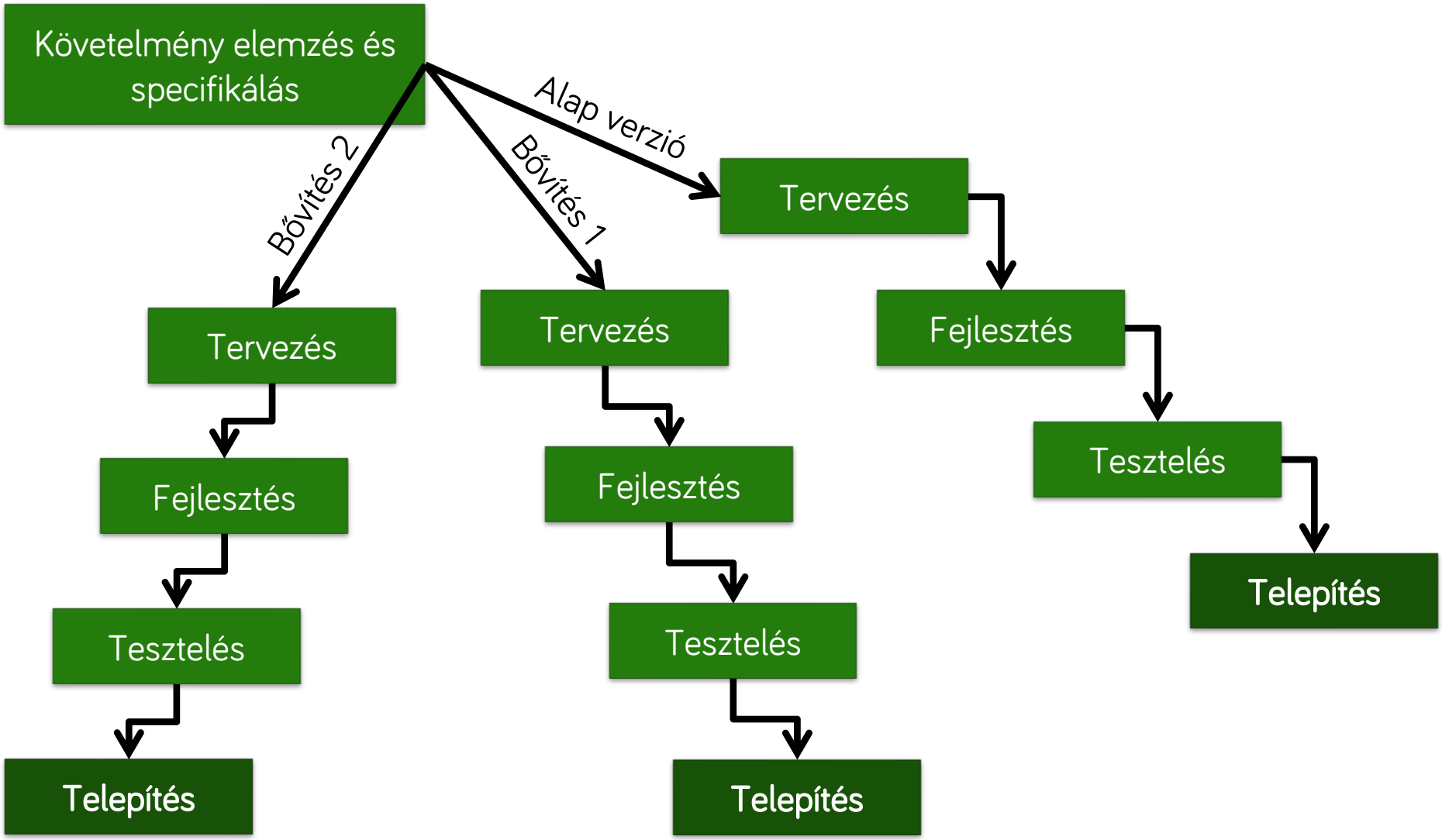
# Prototípus fejlesztés módja

- Gyors, erre kifejlesztett eszközöket, nyelveket használunk
- Kihagyhat funkcionalitást, például
  - A rendszer kevéssé értett területére fókuszál
  - Hibakezelés, helyreállítás stb. nem része a prototípusnak
  - Funkcionális követelmények vannak a középpontban

# Eldobható prototípusok

- A prototípusokat el kell dobni a valódi rendszer fejlesztése előtt
  - Nem alakítható át úgy, hogy a nem-funkcionális követelményeknek megfeleljen
  - A prototípusok általában nem dokumentáltak
  - A prototípus felépítése nem megfelelő a sok változtatás hatására
  - Általában nem felel meg az elvárt minőségi követelményeknek

# Inkrementális fejlesztés



# Inkrementális fejlesztés

- **Megismerjük a teljes követelményrendszert**
- Ahelyett, hogy a rendszert egy lépésben szállítanánk le, azt **darabokra bontjuk** és inkrementumokban szállítjuk
  - Az **inkrementumokat iteratív vízesés** modellben valósítjuk meg
- A követelményeket priorizáljuk, a magas prioritásúakat szállítjuk elsőnek
- Ahogy egy inkrementumot elkezdünk fejleszteni, a követelményeit már nem módosítjuk

# Inkrementális fejlesztés és telepítés

- Inkrementális *fejlesztés*
  - Inkrementeket fejlesztünk és értékeljük azt mielőtt tovább lépnénk
  - **Minimális változtatásokat eszközlünk**
  - Az értékelést a **kulcs felhasználó**, üzleti elemző végzi
- Inkrementális *telepítés*
  - A **végfelhasználó** számára telepítjük az inkrementumot
  - Valódibb visszajelzést kapunk a szoftverről
  - Nehéz egy rendszer kiváltása esetén alkalmazni, mert eleinte kisebb a funkcionalitása mint a kiváltandó rendszernek

# Inkrementális fejlesztés előnyei

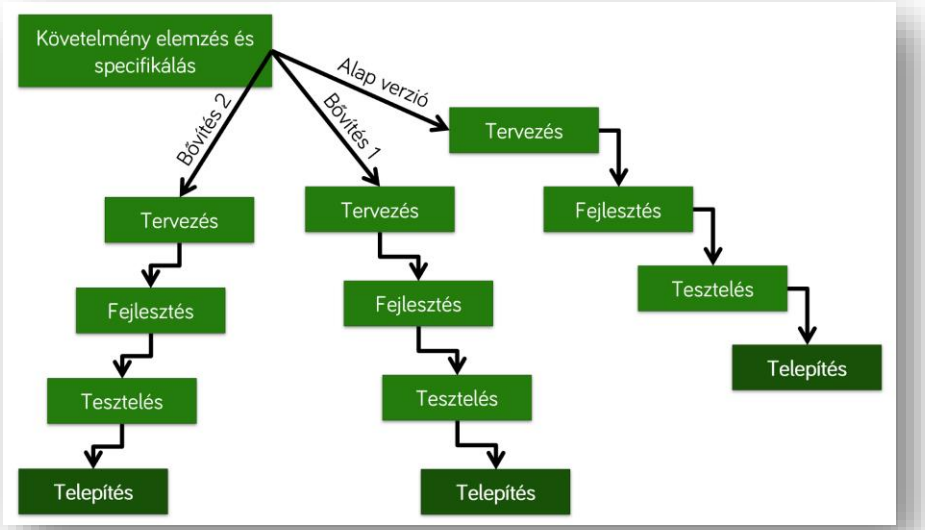
- A változtatások implementálásának **költsége csökken**, a visszajelzések a többi modul tervezésébe beépíthetők
  - A kapcsolódó elemzés, dokumentálás kevesebb, mint a vízesés modellben, mert hamarabb kapunk visszajelzést
- Könnyebb a felhasználóktól érdemi visszajelzést kapni az elvégzett munkáról
- A megrendelő **hamarabb jut** a felhasználók által **használható szoftverhez**
  - A szoftver hamarabb teremt értéket, mint vízesés modellben
- Összességében kisebb az esély a bukásra
- A magasabb prioritású modulokat többet tesztelik

# Inkrementális fejlesztés hátrányai

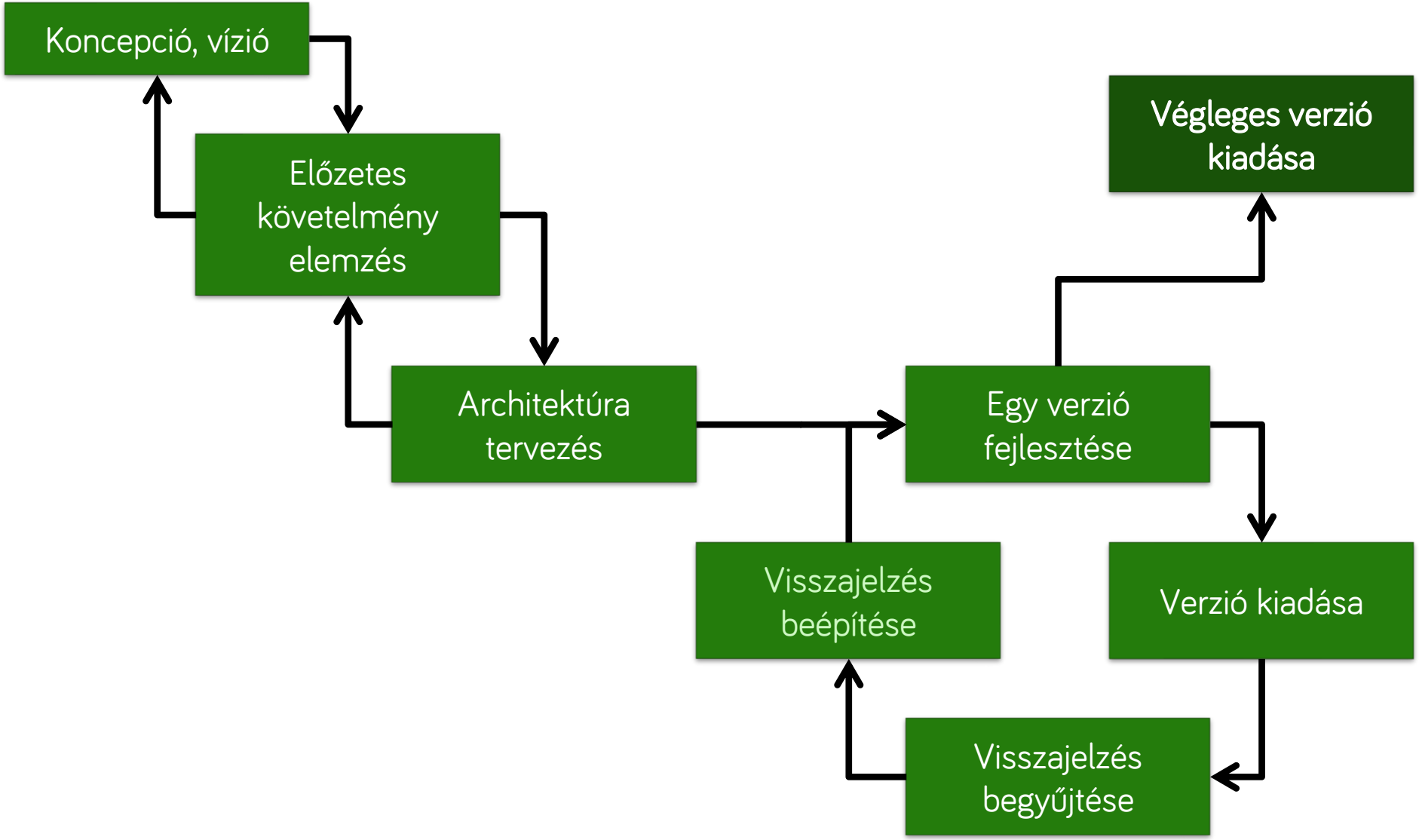
- Az inkrementumok változásai nehezen becsülhetőek
  - Nehezen becsülhető a teljes költség, határidő
  - Nehéz több, egymástól függő projekt összehangolása
- A **változtatások befogadása potenciális konfliktus forrás**
- A rendszer **közös elemeinek felépítése erodálódik** az új inkrementumok fejlesztésénél
  - Új funkciók bevezetése a közös részbe aránytalanul megdrágulhat
  - Hacsak időt és pénzt nem fordítunk a folyamatos refaktorra

# Mikor válasszuk?

- A **követelmények többé-kevésbé tiszták** a megrendelő oldalán
  - Az eltérésből adódó kockázatot úgy kezeljük, hogy visszajelzést kérünk és a változtatásokat beépítjük a következő inkrementumba



# Evolúciós modell



# Evolúciós modell

- **Induláskor nem ismert a teljes követelményrendszer**
- Közös vízió és az architekturális alapok letétele után indul a folyamat
- Mindig a **legfontosabb követelmények** kerülnek kidolgozásra, lefejlesztésre, tesztelésre és átadásra – ez egy iteráció
- **Átadás után a visszajelzések** bekerülnek a követelménylistába
- **A szoftver minden iteráció után használható**

# Evolúciós modell előnyei

- Még kevesebb eltérés van aközött, amire a felhasználónak szüksége van és aközött, amit fejlesztünk
  - **Lényegében addig fejlesztjük, amig pont jó nem lesz**
  - … vagy amíg a pénz el nem fogy
- A megrendelő folyamatosan kap használható szoftvert
- A felhasználó folyamatosan ad visszajelzést

# Evolúciós modell hátrányai 1

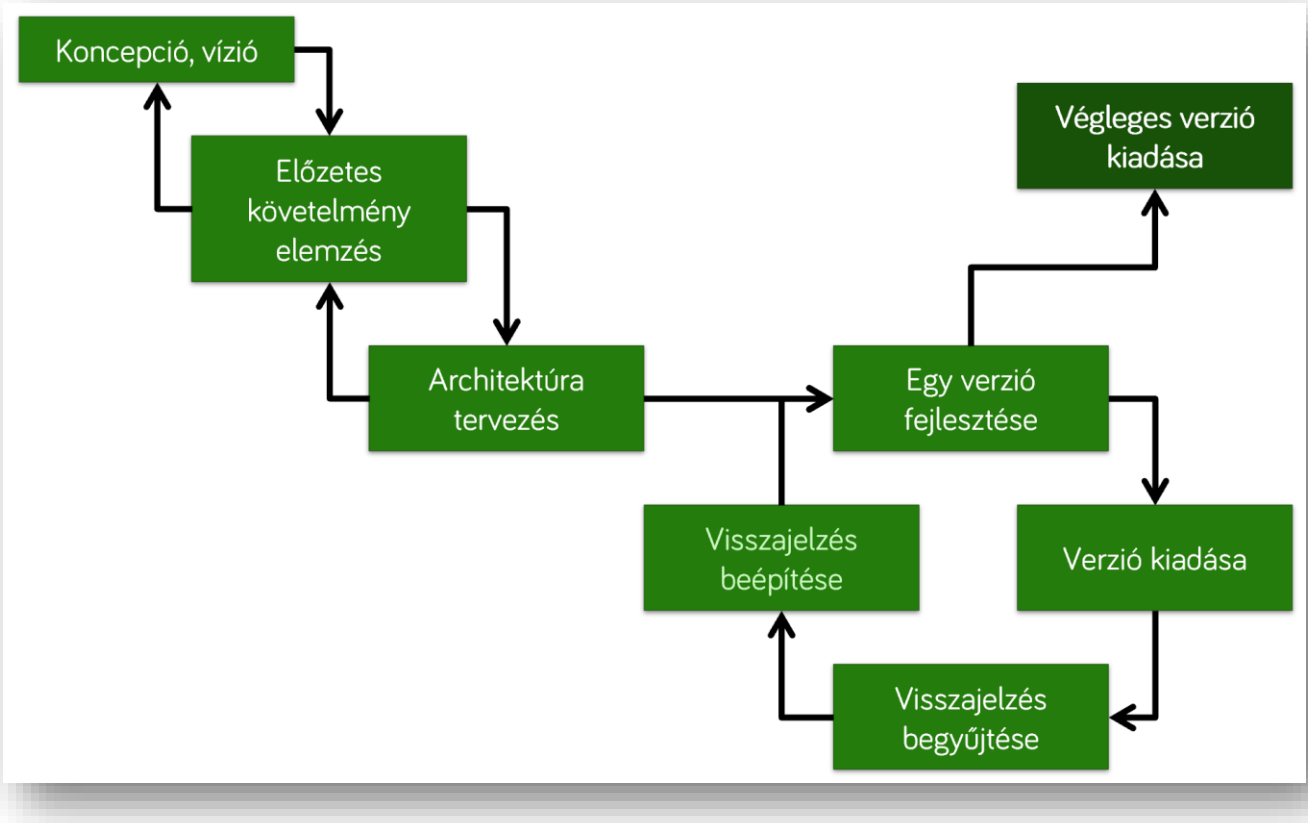
- A legtöbb rendszernek szüksége van alap infrastruktúrára, amit több modul használ
  - Ezeket a közös részeket nehéz azonosítani és specifikálni, amíg a teljes követelményrendszer össze nem áll
- Az iteratív fejlesztés lényege, hogy a specifikálás és fejlesztés együtt halad
  - Sok szervezetben ez ellentmond a beszerzési folyamatnak, ahol a teljes követelményrendszer a szerződés része

# Evolúciós modell hátrányai 2

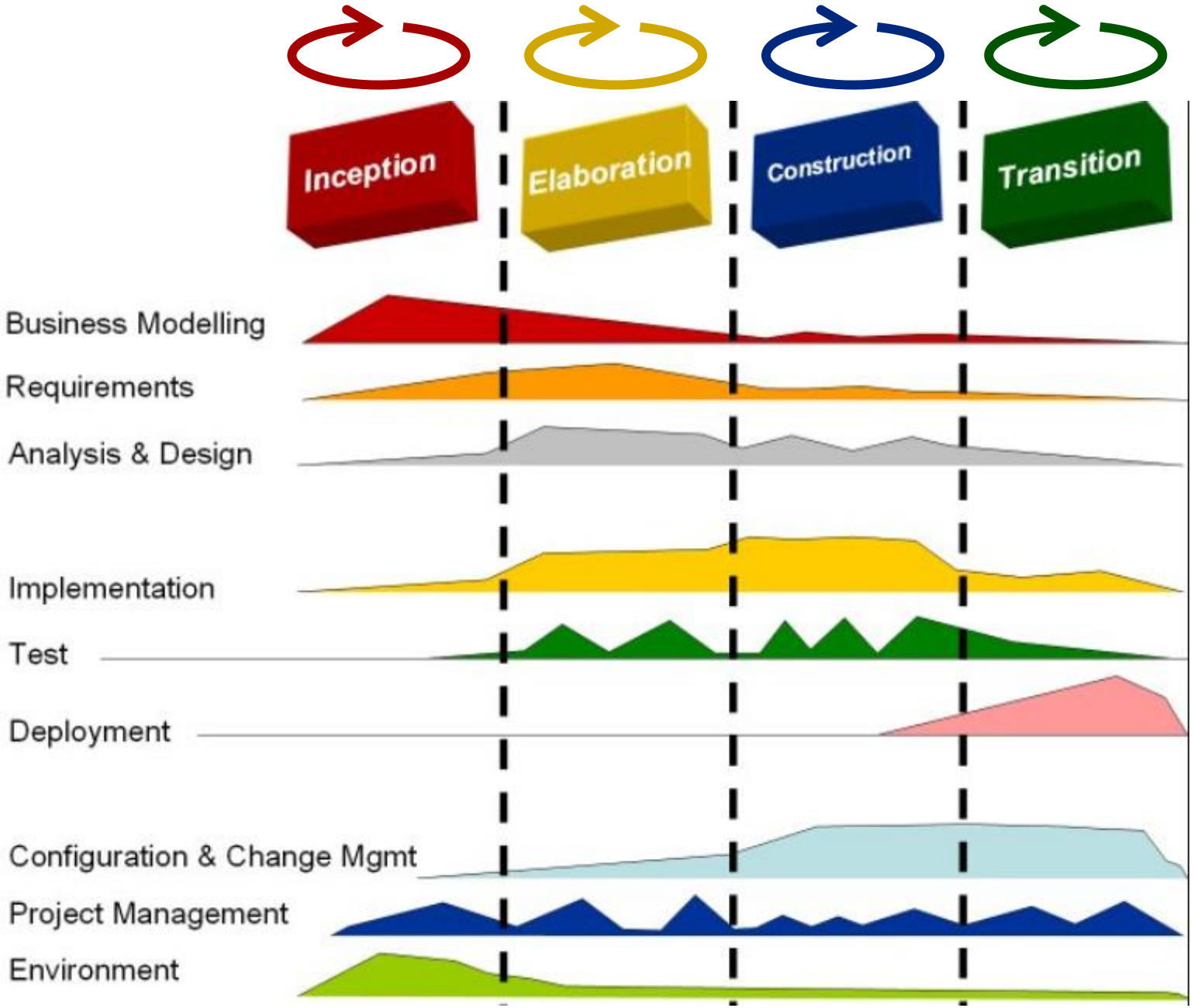
- Nincs pontos terv a teljes folyamat lefutására
  - Nehezen becsülhető a teljes költség, határidő
  - Nehéz több, egymástól függő projekt összehangolása
- A rendszer felépítése erodálódik új inkrementumok fejlesztésénél
  - Új funkciók bevezetése aránytalanul megdrágulhat
  - Hacsak időt és pénzt nem fordítunk a folyamatos refaktorra
- A folyamatnak csak a részei láthatók
  - A haladás felméréséhez rendszeres eredménytermékekre van szüksége a menedzsmentnek – gyors verzió váltások esetén nem hatékony az összes dokumentum összeállítása

# Mikor válasszuk?

- Ha a követelmények előre nem ismertek elég részletesen



# RUP


Inception
Elaboration
Construction
Transition
Business Modelling
Requirements
Analysis & Design
Implementation
Test
Deployment
Configuration & Change Mgmt
Project Management
Environment

# RUP

- IBM **R**ational **U**nified **P**rocess (RUP), 2003
  - Elődje a UP, azt azok készítették, akik az UML-t
- Átfogó folyamat keretrendszer
- A **vízesés**, **inkrementális** és **újrahasznosító** modellek ötvözése
- Iparban alaposan tesztelt legjobb gyakorlatok szoftverrendszer fejlesztéshez

# RUP jellemzői 1

- **Adaptálható**nak tervezték: a projekthez igazítható
- Folyamat-keretrendszer
- Use-case vezérelt
- Architektúra központú
- Javasolja az UML használatát
- **Formálisabb**, előíróbb, mint más agilis keretrendszerek
  - Jobban illeszkedik a klasszikus nagyvállalati kultúrába

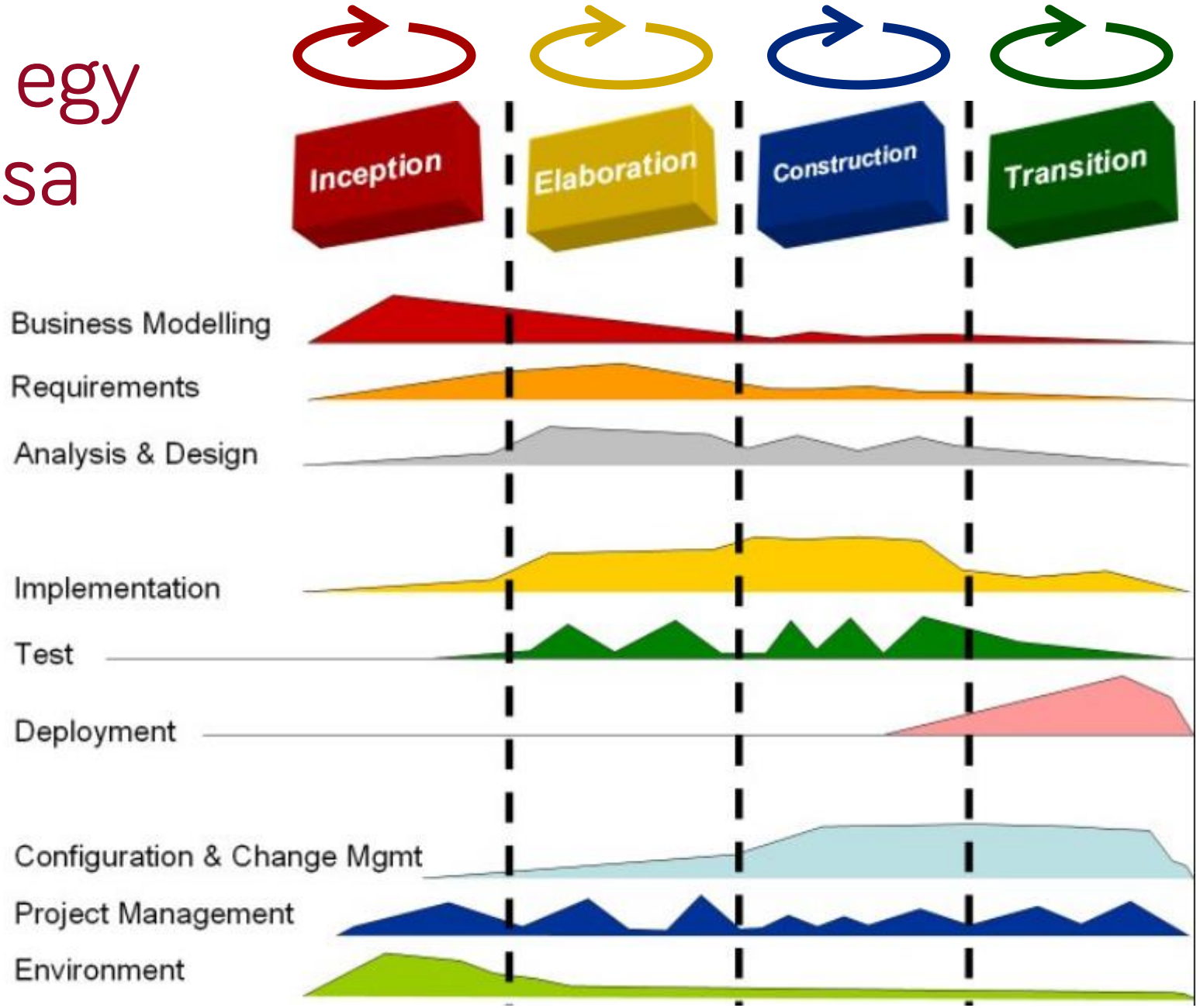
# RUP jellemzői 2

- **Kockázatok** folyamatos kezelése
- Biztosítja, hogy **érték**et szállítunk az ügyfélnek
- A **működő szoftverre** fókuszál
- **Képes a változásokat korán befogadni**
- Komponens alapon építkezik
- **Csapatmunkában** gondolkozik
- A **minőség** a folyamat része, nem útógondolat

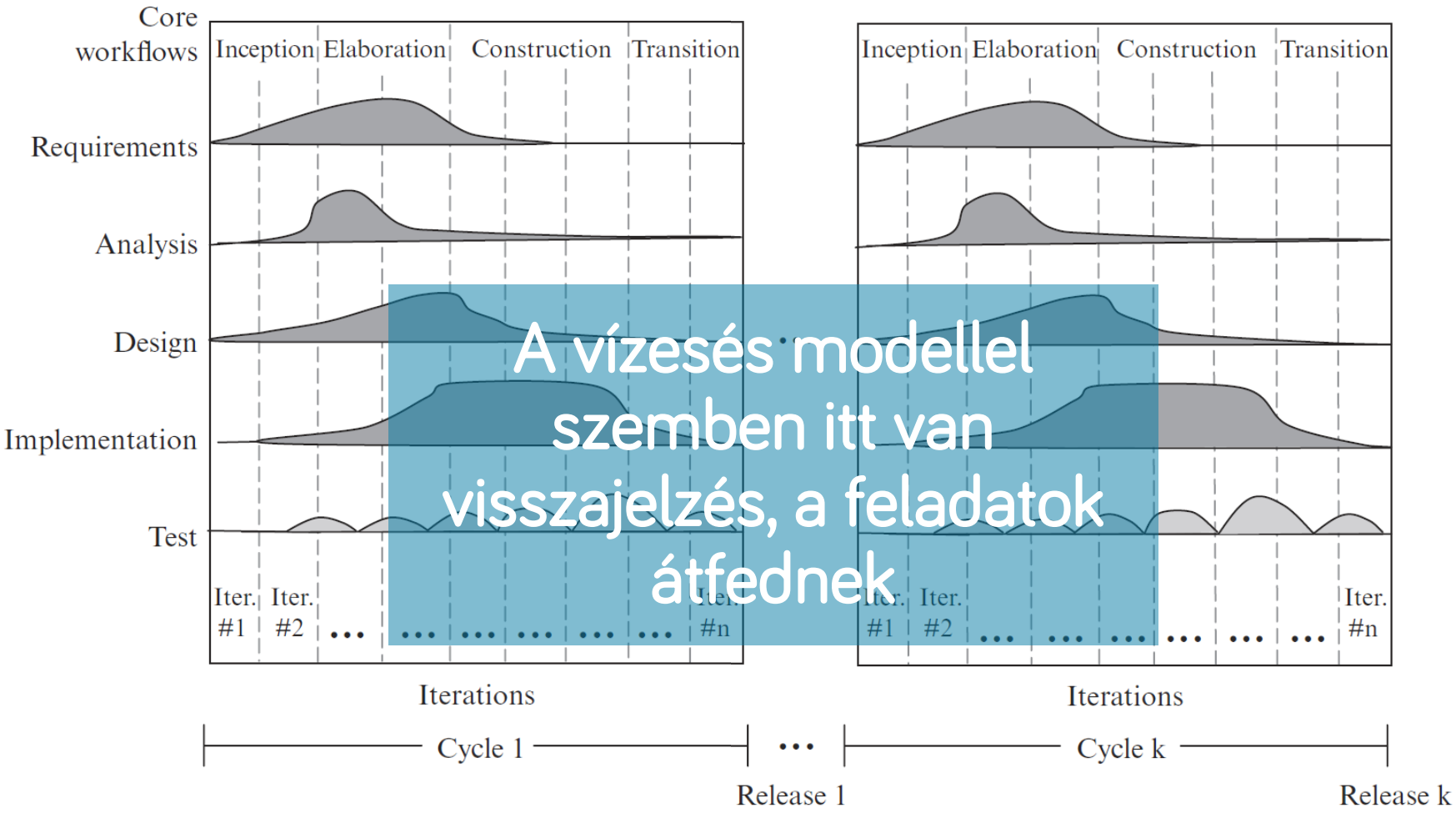
# RUP terminológia

- **Ciklus**: a szoftver egy kiadásához tartozó tevékenységek időbeli kerete
- **Fázis**: minden ciklus négy fázisra osztott
- **Iteráció**: a fázison belül több iteráció lehetséges, amely eredménytermékek kiadásával zárul. Az iterációk célja a kockázatok folyamatos csökkentése.
- **Mérföldkő**: egy iteráció vége, formális pont
- **Workflow**: tevékenységek sorozata, amely látható értéket teremt

# RUP egy ciklusa



# RUP iterációi és ciklusai




A vízesés modellel szemben itt van visszajelzés, a feladatok átfednek

# RUP építőelemek 1

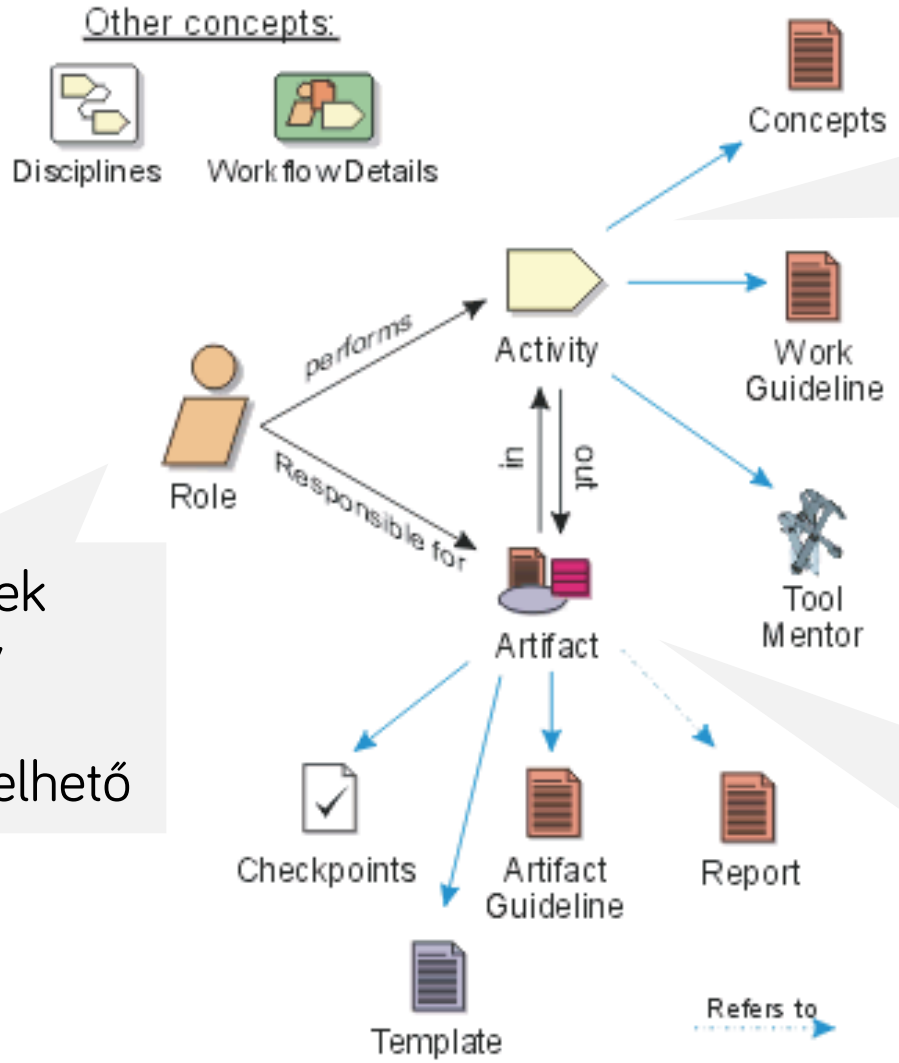
# RUP építőelemek 2

- **Szerepkörök** (who) – A szerepkör képességet, kompetenciát és felelősséget definiál.
- **Eredménytermékek** (what) – Az eredménytermék (artifact) a feladatok eredményeit jelentik, beleértve a teljes folyamat során keletkezett modelleket és dokumentációt, például:
  - Vízió dokumentum
  - Prototípus
  - Use case
  - Üzleti leírás
  - Terv
  - Forráskód
  - Telepíthető állomány

# RUP építőelemek 3

- **Feladatok** (how) – A feladat egy szerepkörhöz rendelt munkaegységet definiál, melynek használható eredménye van
- Minden iteráción belül a feladatok kilenc területre oszlanak fel:
- Hat mérnöki területre
  - Üzleti modellezés
  - Követelmények
  - Elemzés és tervezés
  - Implementáció
  - Tesztelés
  - Üzembe helyezés
- Három támogató területre
  - Konfiguráció és változáskövetés
  - Projektmenedzsment
  - Környezet

# RUP építőelemek 4

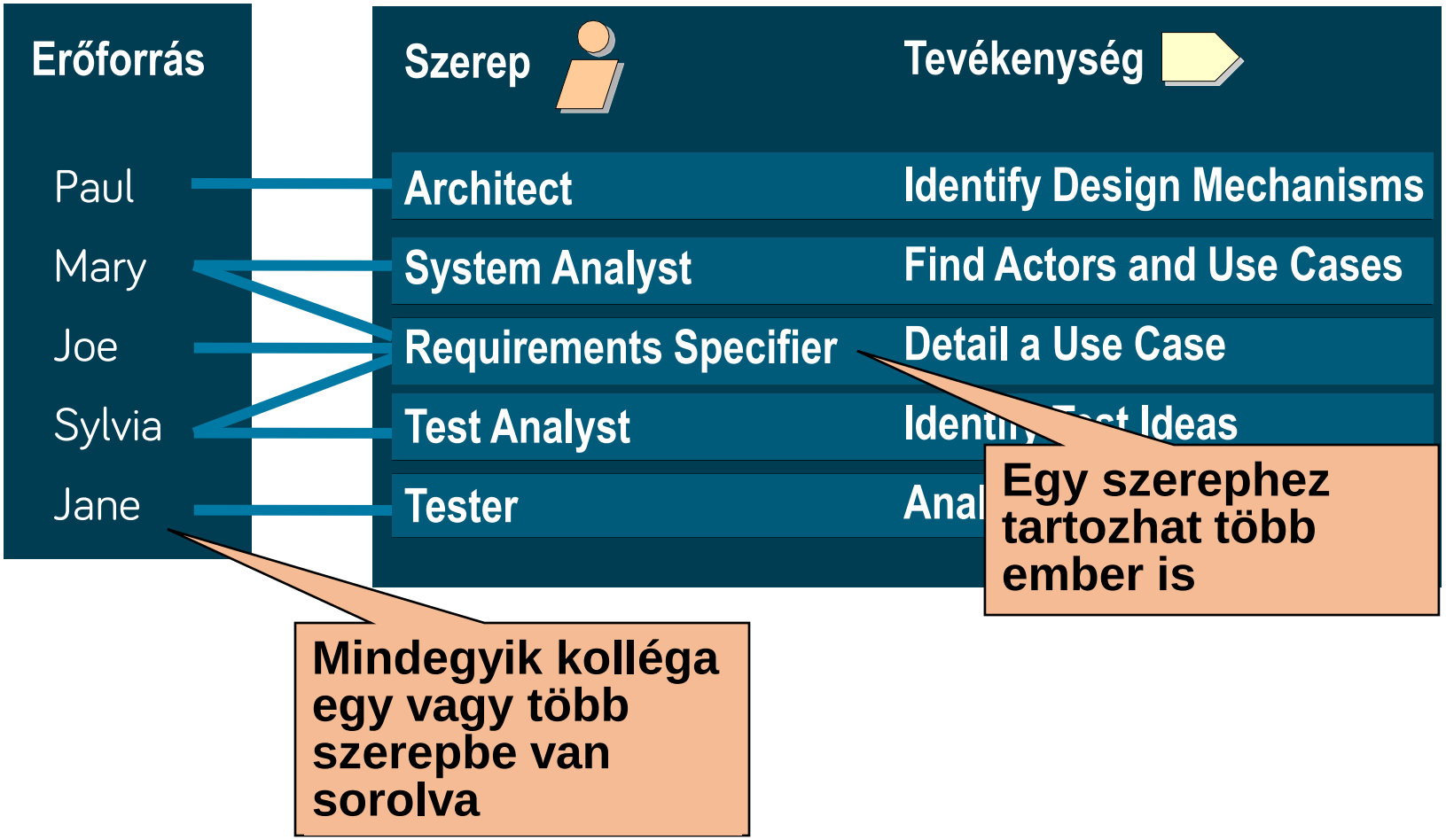



**Activity**: a munka egysége amit egy szerep végez el

**Role**: felelősségek halmazat amely egyénhez vagy csapathoz rendelhető

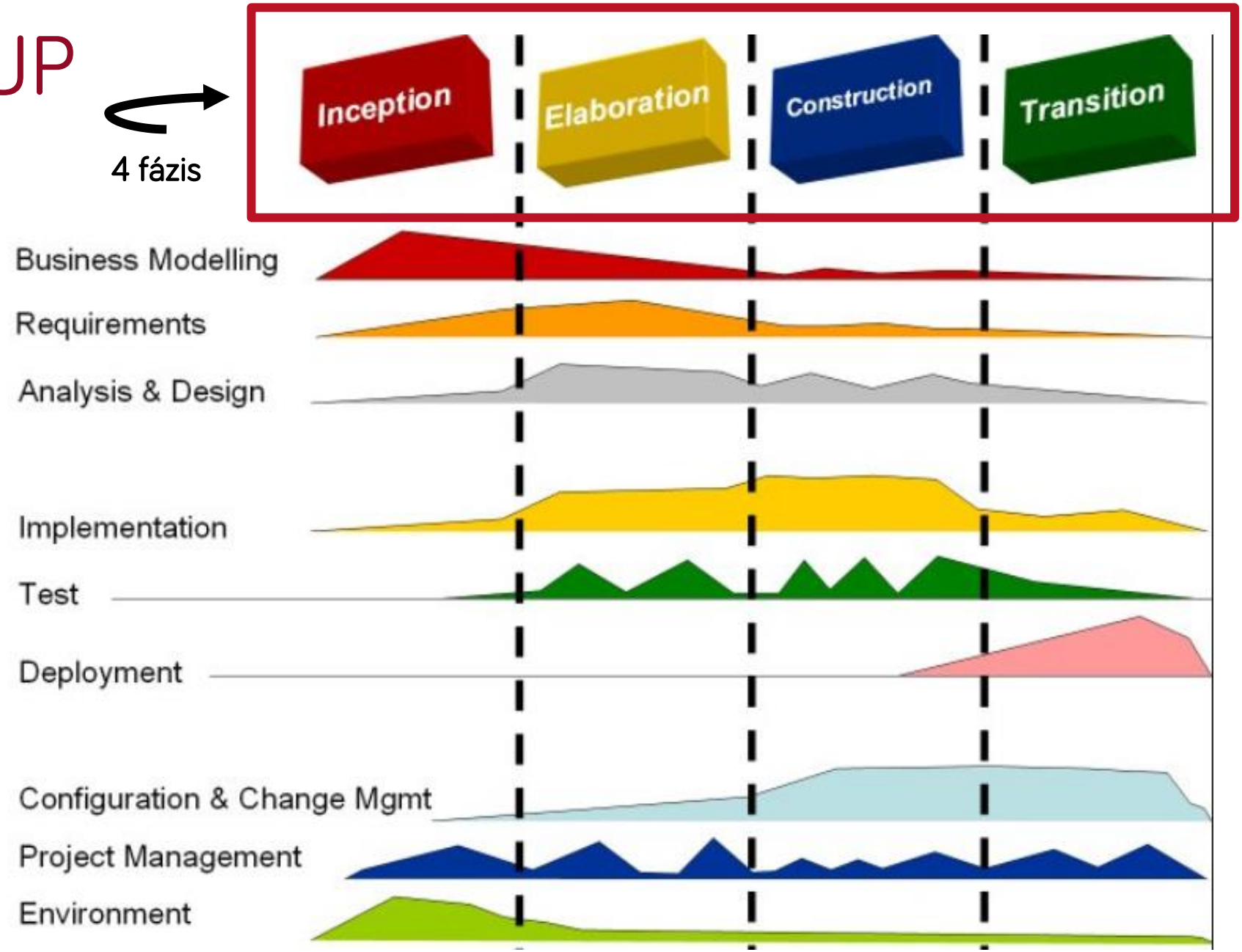
**Artifact**: információ darab, amit egy tevékenység használ, módosít vagy állít elő

# Roles Are Used for Resource Planning



# RUP

4 fázis


Inception
Elaboration
Construction
Transition
Business Modelling
Requirements
Analysis & Design
Implementation
Test
Deployment
Configuration & Change Mgmt
Project Management
Environment

# RUP – A projekt életciklus négy fázisa

- Az iteratív fejlesztési folyamatot négy fázisra osztja
- Minden fázis egy vagy több iterációt tartalmaz
- Minden fázis alatt minden terület megjelenik, de más mértékben
- Tipikusan ciklusonként kötünk szerződést, egy ciklus néhány hónap hosszú
- Egy iteráció tipikusan 2-6 hét hosszú
- A bővített modell tartalmaz plusz egy fázist: üzemeltetést és a hozzá tartozó folyamatot

# RUP – Inception (Kezdet)

- A projekt terjedelem kialakítása
  - Milyen hozzáadott **érték**e van a felhasználó szempontjából vs mennyi feature-t valósítunk meg
- A csapat eldönti, hogy érdemes-e a projektet elindítani, illetve folytatni
- Elindul a tervezés
  - Költség elemzés, erőforrástervek
  - Kockázatok
  - Folyamatok, eszközpark kialakítása
  - Magas szintű architektúra javaslat

# RUP – Elaboration (Kidolgozás)

- A probléma terület megértése
- Prototípusok készítése
- Funkcionális és nem-funkcionális követelmények rögzítése, finomítása
  - Use-case-ek és kiegészítő dokumentáció
  - Szekvencia és kollaborációs diagrammok
- A projekt architektúra pontosítása
- Részletes projekt terv kialakítása
  - Erőforrás igények, függőségek stb.
- Kockázatok felülvizsgálata

# RUP – Construction (Megépítés)

- Működő szoftver készítése
- A hagyományos fázisok közül itt történik:
  - A szoftver tervezése
    - További UML és egyéb mérnöki tervek, kevesebb szöveges dokumentáció
  - A szoftver implementációja/fejlesztése
  - A szoftver tesztelése

# RUP – Transition (Átadás)

- A szoftver publikus kiadása, átadása
  - Alfa, béta tesztelés, pilot időszak
  - Végső simítások
  - Visszajelzések alapján történő korrekciók
- Dokumentáció véglegesítése
- A fejlesztő csapat mérete csökken
- Az irányítást átadjuk az üzemeltetésnek

# A 6 best practice

1. Develop iteratively (iteratív fejlesztés)
   - Megrendelői prioritás alapján tervezzük a következő iterációt
   - Minden fázis végén go/no go döntés: költség csökkentés
2. Manage requirements (követelmény kezelés)
   - Mindig tartsuk szem előtt, hogy a követelményeket a felhasználók határozzák meg
   - A követelményeket és változásokat dokumentáljuk

# A 6 best practice

3. Use components (komponenseket használunk):
   - A projekt komponensekre bontása segíti a tesztelést, **újrafelhasználást**, átlátást
4. Model visually (Vizuálisan modellezünk):
   - Használjunk diagramokat a szoftver statikus és dinamikus nézetének leírására

# A 6 best practice

5. Verify quality (minőségtudatosság):
   - Biztosítsuk, hogy a szoftver  a szervezet minőségi követelményeinek megfelel
   - A tesztelés mindig kapjon kitüntetett szerepet a projekt minden fázisában
   - A tesztelési feladat folyamatosan nő a projekt előrehaladtával, de konstans faktornak kell lennie!

# A 6 best practice

6. Control changes (Kövessük és ellenőrizzük a változásokat):
   - Kövessük és dokumentáljuk a változásokat, használjunk erre eszközöket!
   - Számos projektet több különböző csapat fejleszt, gyakran különböző helyszínen, más-más platformok használatával
   - Folyamatosan gondoskodjunk a változások szinkronizálásáról és verifikálásáról. (Continuous integration)

# RUP előnyei

- Rendszeres visszajelzés a megrendelőtől
- Iteratív folyamat
- Végül azt szállítjuk, amit a megrendelő szeretne
- Az erőforrások hatékony felhasználása
- A folyamat korai fázisában megjelennek a kezelendő kihívások
- Beépített kockázat kezelés

# RUP hátrányai

- Bizonyos helyzetekhez túl komplex folyamat
- Működtetéséhez jelentős erőforrásokra, szakértelemre van szükség
- A fejlesztés „elszabadulhat”

# Mini videók – perc:másodperc ☺

- Waterfall – 1:55
  - https://www.youtube.com/watch?v=5A5XCuWMG4o
- Iterative and incremental – 1:14
  - https://www.youtube.com/watch?v=gcpOZi6Hz38
- Spiral – 1:25
  - https://www.youtube.com/watch?v=mp22SDTnsQQ
- RUP – 2:30
  - https://www.youtube.com/watch?v=YgkhFH8g0J4

# Köszönöm a figyelmet!