



# Szoftvertchnológia

## 5. Tervezés, implementáció

BSc kurzus

Dr. Balla Katalin



# Tartalom

- Tervezés / design
  - Definíciók
  - A folyamat
  - Szoftvertervezési alapelvek és fontos elemek
  - Architektúrák, döntések, tervezési minták
  - Felhasználói interfész (UI) tervezése
  - Szoftvertervezés a CMMI és az Automotive SPICE modellekben
- Implementáció / kódolás
  - Kódolási szabványok alkalmazása
  - A kód minőségének ellenőrzése
- Becslés a szoftvertervezés során
- Tervezés és implementáció agilis környezetben



# Mit jelent a szoftvertervezés (Design)?

- Az a folyamat, melynek során :A követelményeket kiindulásul használva egyre pontosabban és részletesebben megértjük a rendszert, annak minden elemével együtt
  - Ennek a megértésnek a formalizálása, különböző nézőpontokból, általában:
    - Funkcionalitás
    - Funkciók, funkció csoportok, feature-ok ,
    - Adatok
    - Kommunikáció, az elemek közötti interfészek
  - A formalizálás különböző tervezési modellek felhasználásával történik.
- A követelményeket lefordítjuk / konvertáljuk
  - Funkcionális elemekre
  - Szoftvert struktúrákra



# Tervezés a SWEBOK szerint

- „Design”: mind „az architekturális emelek, komponensek, interfészek és egyéb rendszer elemek definíciója” , mind „ezen folyamatok eredménye” [1].
- Ha folyamatként vizsgáljuk, a szoftvertervezés a szoftverfejlesztési életciklus azon tevékenysége, amelynek során a követelményeket elemezzük abból a célból, hogy elkészítsük a szoftver belső struktúrájának leírását, amely az implementáció alapjául fog szolgálni.
- A szoftver terve (a folyamat eredménye) leírja a szoftver architektúráját – vagyis azt, ahogyan a szoftver elemekre / komponensekre bomlik – és az elemek közötti interfészeket. A komponenseket a szoftvertervnek olyan részletességgel kell leírnia, amely lehetővé teszi azok „megépítését” (a kódolást).
  - [1] ISO/IEC/IEEE., *24765:2010 Systems and Software Engineering—Vocabulary*, ISO/IEC/IEEE, 2010.
  - From: SWEBOK , [http://swebokwiki.org/Chapter\\_2:\\_Software\\_Design](http://swebokwiki.org/Chapter_2:_Software_Design)



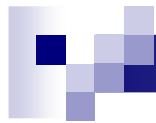
# Design / tervezés

- A tervezés végére a teljes rendszer koherens , összefüggő struktúráját le kell írni!
- A tervezés különbözik a kódolástól. A tervezés a kódolásnál magasabb absztrakciós szinten van!
- A design / tervezés részletezettsége több tényezőtől függ. Különböző modellek különböző alapfogalmakat, elnevezéseket használnak, de általában mindig a következő elemekkel foglalkozik a tervezés:
  - Szoftver architektúra terve (nevezik még magas szintű tervnek / high-level design): a szoftver legfelső szintű struktúráját írja le, és azonosítja a komponenseket.
  - A szoftver részletes terve / Software detailed design: a komponenseket olyan részletességgel írja le, amely lehetővé teszi, hogy implementálni lehessen őket



# Design / tervezés

- A tervezés elemeinek más elnevezései:
  - ☐ Fogalmi terv / Conceptual / high level design
  - ☐ Részletes terv / Detailed design
- A tervezés során alapvetően 2 rendszer-nézetet írunk le:
  - ☐ Statikus nézet – dinamikus nézet

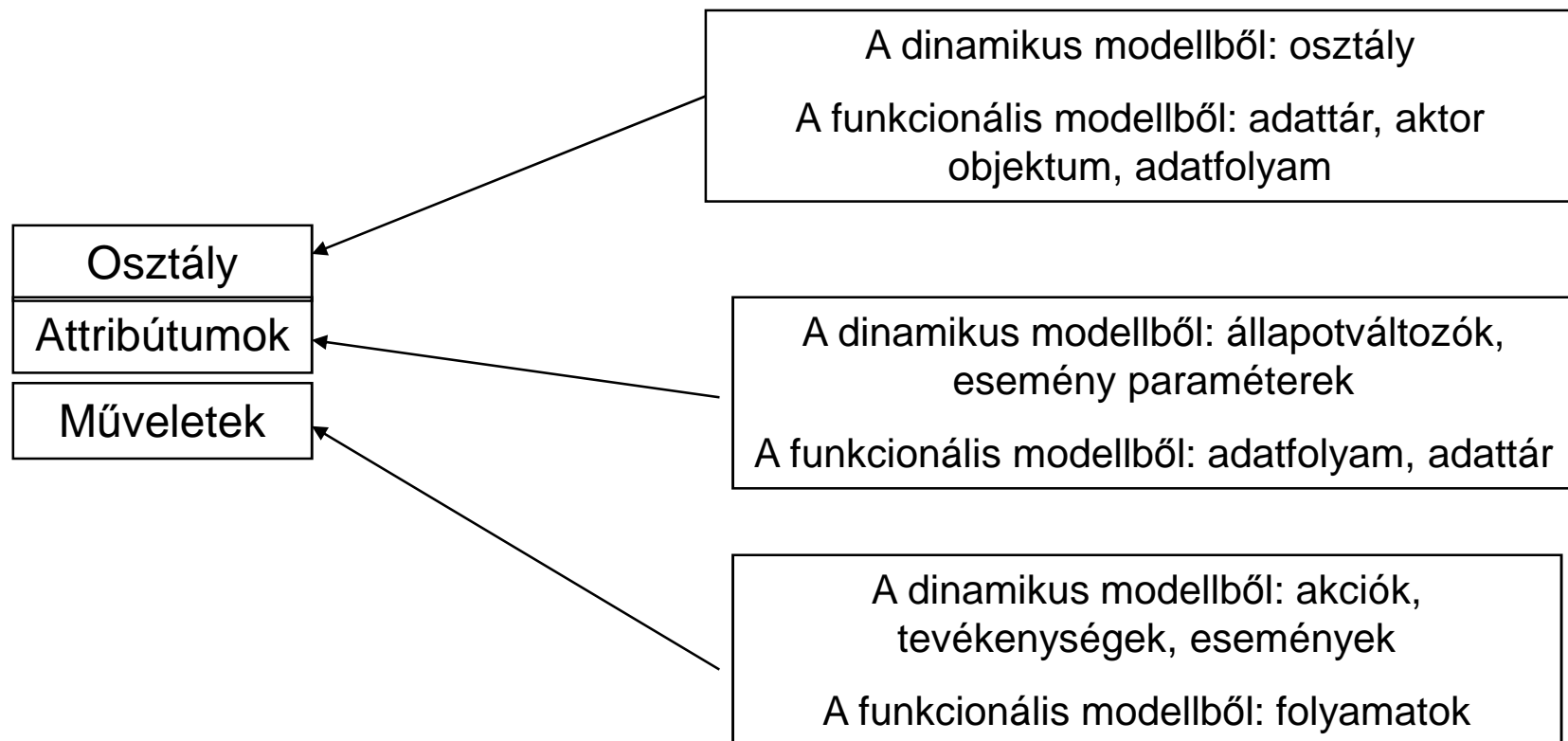


# Design / tervezés

- Vannak olyan modellek, **csakis** tervezésre használhatók? (és nem használhatók pl. követelményelemzésre, tesztelésre...?)
  - Nincsenek. Egy modellt általában több életciklus-fázisban lehet használni, különböző célokkal. Ezzel együtt, kialakultak „jó gyakorlatok”, hogy melyik modell melyik fázisban / milyen tevékenységre használható leginkább.
    - Emlékezzünk az UML diagramokra! Elhangzott, hogy melyik modell melyik életciklus fázisban a leghasznosabb. De több fázisban is használható sok modell!
- Miért van szükség több modellre (strukturális, funkcionális, adat, use-case etb...?)
  - Mert a különböző modellek a rendszer különböző elemeit hangsúlyozzák. Összességükben jobb rálátást biztosítanak a teljes rendszerre
- Eszközökkel támogatható, hogy a modellek egymás között is konzisztensek maradjanak!

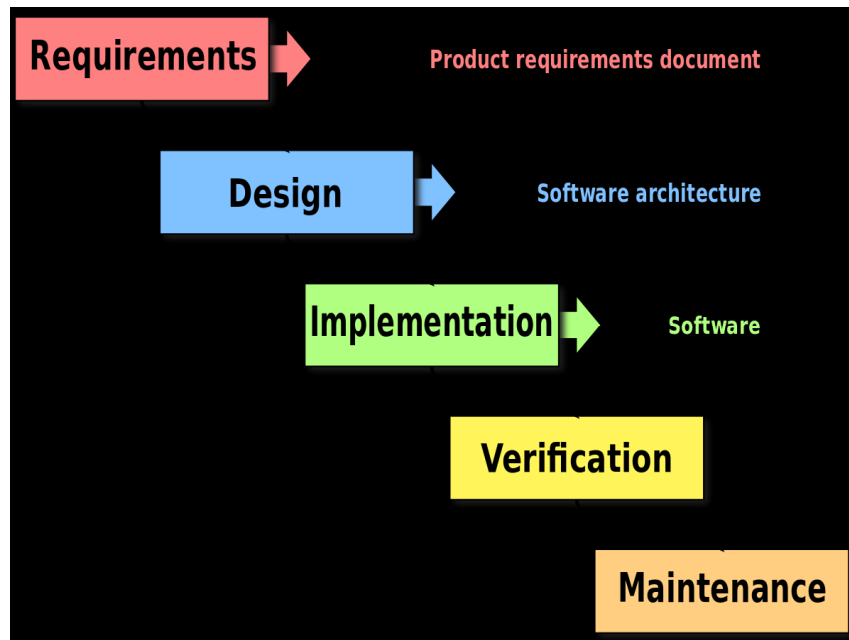
# Hogyan határozzuk meg a tervezésben használatos rendszer-elemeket?

- Példa: OO fejlesztés: elemek meghatározása
- A 3 modell integrálása (OMT)

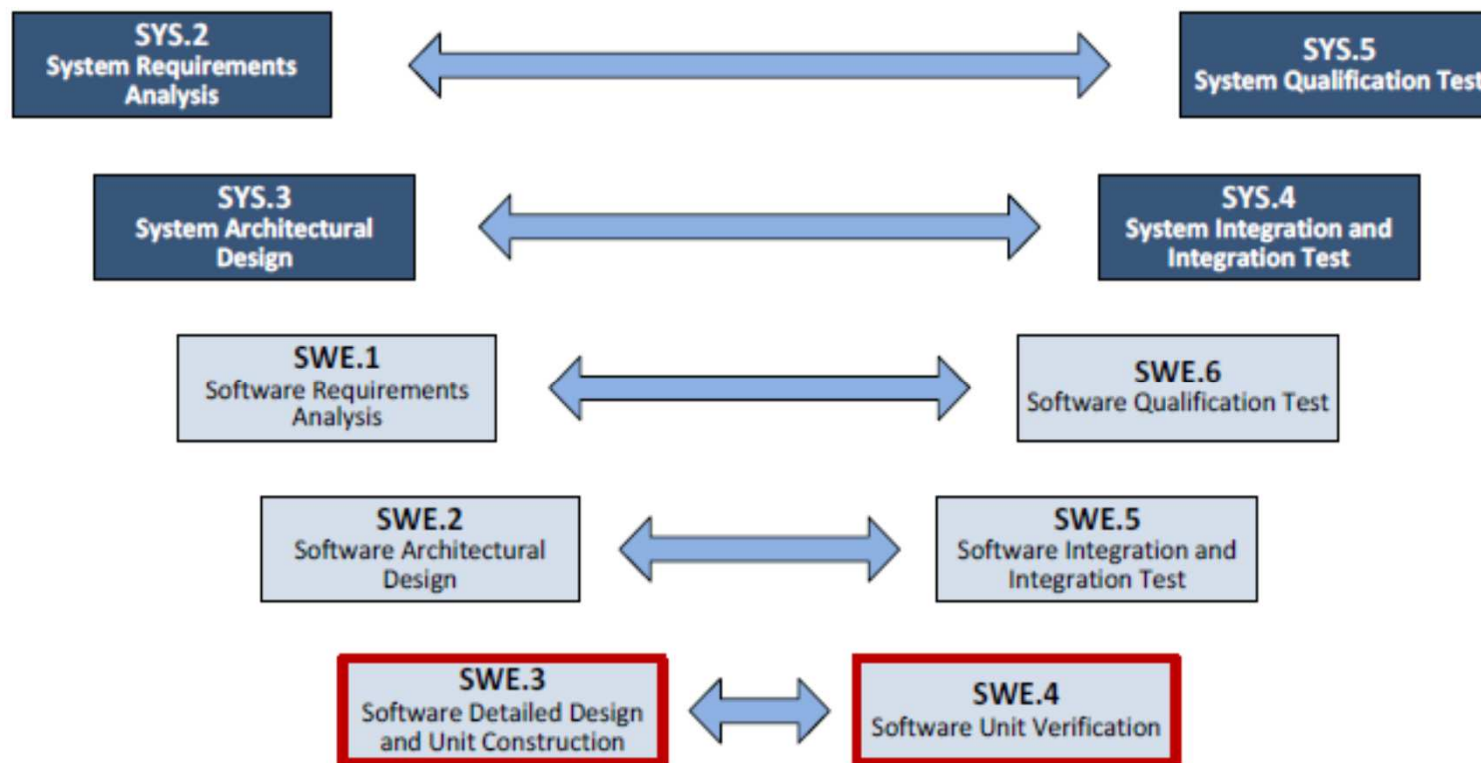




# Tervezés / Design a szoftverfejlesztési életciklusban

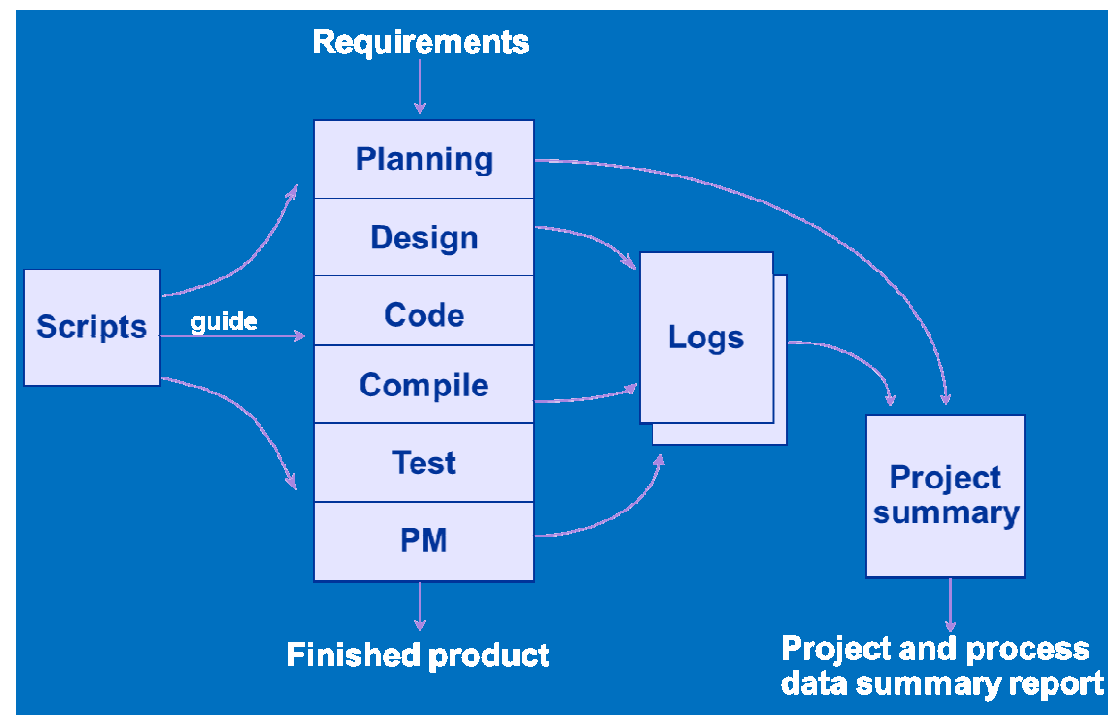


# Tervezés / Design a rendszerfejlesztési életciklusban



From: Automotive SPICE , PAM, v3

# Tervezés / Design a szoftverfejlesztési életciklusban




A PSP alapfolyamata



# Szoftvertervezési alapelvek

- Olyan alapelemek, elvek, amelyeket többféle szoftvertervezési megközelítésben is elfogadnak és alkalmaznak
  - Pl: Single Responsibility Principle (SRP), Open/Closed Principle (OCP), Demeter törvénye(LoD) stb.
  - Szó volt már róluk az OO Design Principles c. előadásban , UML2 kapcsán!)



# Mire kell különösen odafigyelnünk a szoftver tervezése során?

- Néhány **minőségi követelmény**, amelyek minden szoftverre érvényesek: teljesítmény, biztonság, megbízhatóság, használhatóság stb.
  - Úgy tervezzük meg a szoftvert, hogy ezeket a jellemzőket tartsuk szem előtt!
- Tervezés során kell „kitalálni”, hogyan **bontsuk fel, szervezzük, csomagoljuk a szoftver komponenseket. Jól kell kitalálni!**
- A **szoftver működésével** kapcsolatban olyan elemek is figyelmet igényelnek, amelyek nem magában a készülő rendszerben, hanem annak környezetében vannak jelen.
  - Pl. illeszkedni kell meglévő elemekhez vagy más rendszerekhez, figyelni kell a működési környezetre stb.



# Szoftver struktúra és architektúra

- Egy szoftver architektúra azoknak a struktúráknak az összessége, amelyekkel foglalkozni kell / tekintetbe kell venni őket a rendszer fejlesztésekor. Magába foglal szoftver elemeket, a közöttük levő kapcsolatokat , valamint az elemek és kapcsolatok tulajdonságait.
  - Az 1990-es évek közepétől a szoftver architektúrák tanulmányozása önálló tudománnyá fejlődött; általánosabb értelemben vizsgálják, tárgyalják ezeket.



# Architektúra stílusok

- Az architektúra –stílus elemtípusoknak és relációtípusoknak egy speciális halmaza, az alkalmazásukra vonatkozó korlátozásokkal együtt.
- Egy architektúra-stílus a szoftver szerkezetének magas szintű leírását adja.
- Különböző szerzők különböző architektúra-stílusokat említenek, pl.:
  - Általános struktúrák (pl. rétegek, csőrendszerek és szűrők...)
  - Elosztott rendszerek (pl. kliens-szerver, háromrétegű...)
  - Interaktív rendszerek (pl. Model-View-Controller, Presentation-Abstraction-Control)
  - Egyéb (pl. batch, interpreter, folyamat vezérlő, szabály-alapú).



# Tervezési minták

- Egy minta “egy általános megoldás egy adott kontextusban megjelenő általános problémára”.
- Az architektúra stílusok tekinthetők a szoftver-szerkezet magas szintű szervezési mintáinak
- Egyéb , a szoftver részleteire is kiterjedő minták is vannak.
- Ilyenek például:
  - Létrehozási minták (pl. builder, factory, prototype, singleton)
  - Strukturális minták (pl. adapter, bridge, composite, decorator, façade, flyweight, proxy)
  - Viselkedési minták (pl. command, interpreter, iterator, mediator, memento, observer, state, strategy, template, visitor).

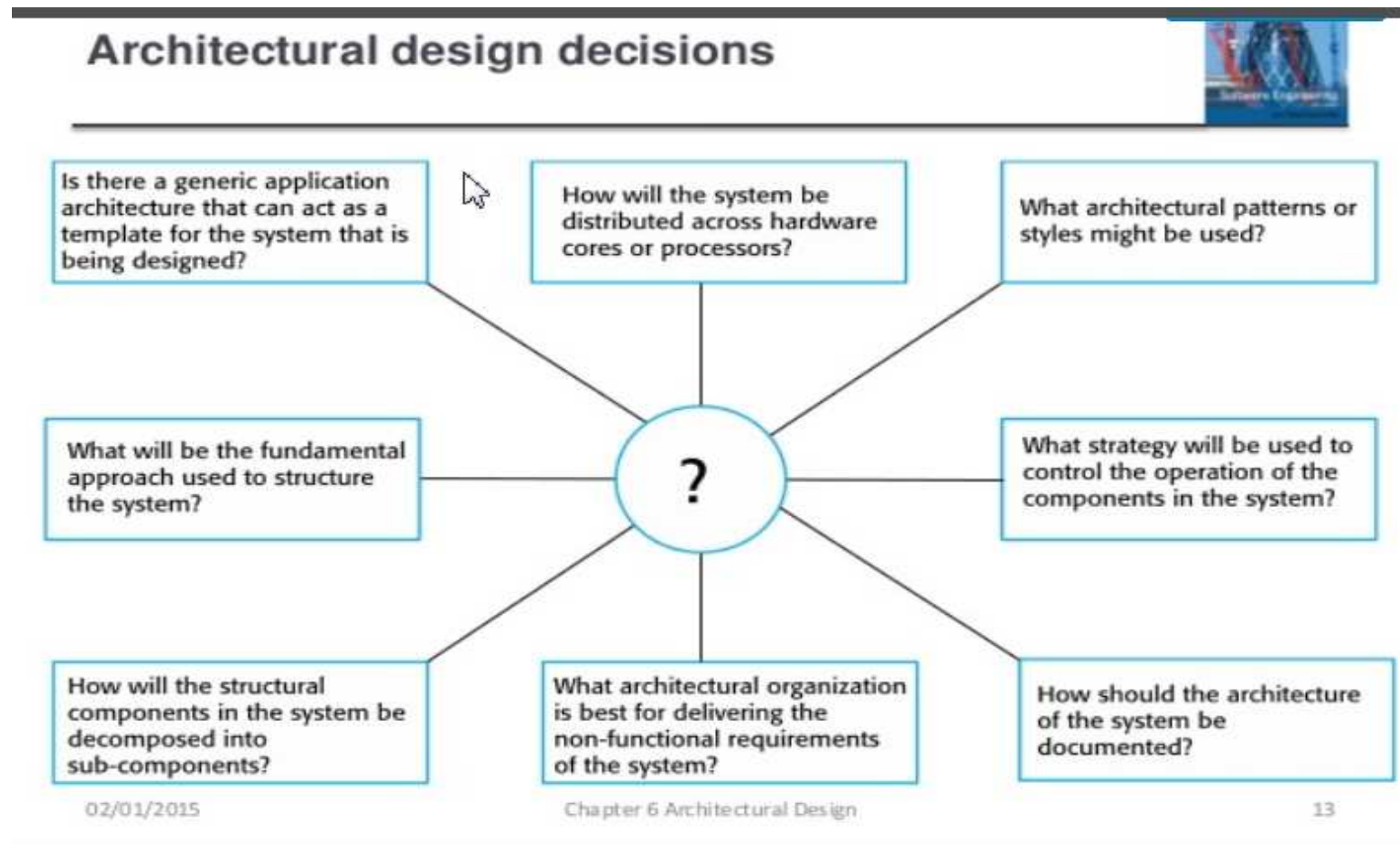




# Tervezési döntések

- A tervezés kreatív folyamat.
- A tervezés során a tervezőknek sok alapvető döntést kell meghozniuk, amelyek a szoftver struktúráját és a teljes szoftverfejlesztési folyamatot döntően befolyásolják.
  - A teljes tervezési folyamatot tulajdonképpen tekinthetjük döntések sorozatának is.
- A döntések során mérlegelni kell a különböző minőségi attribútumokat és vevői igényeket, egyensúlyra törekedve.

# Tervezési döntések





# Architekturális döntések

- Ilyen kérdéseket teszünk fel (magunknak, másnak...):
  - ☐ Van-e olyan általános architektúra, amely mintául szolgálhat a most tervezett rendszernek?
  - ☐ Hogyan fog a rendszer a meglévő hardveren megoszlani?
  - ☐ Milyen architektúra stílusokat, mintákat lehetne alkalmazni?
  - ☐ Milyen stratégiával ellenőrizzük majd a komponensek működését?
  - ☐ Hogyan kell a rendszer architektúráját dokumentálni?
  - ☐ Melyik architektúra a legmegfelelőbb a nem-funkcionális jellemzők megvalósítására?
  - ☐ Hogyan bomlanak alá a rendszer strukturális elemei?
  - ☐ Melyik megközelítés legyen az alapvető a rendszer struktúrájának meghatározásakor?



# Az újrafelhasználás fontossága a tervezés során

## ■ Programcsaládok és keretrendszerek

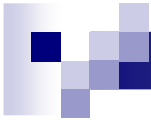
- Az újrafelhasználást támogatja, ha eleve programcsaládokat tervezünk; ezeket terméktípusoknak is nevezik (software product lines).
- Ilyenkor meg kell határozni az elemek közötti kommunikációt, és arra kell törekedni, hogy az elemek önállóak, jól dokumentáltak legyenek.
- Az OO fejlesztésben alapvető elgondolás, hogy „részben kész” keretrendszereket fejlesztenek, amelyekhez később új elemek hozzáadhatók (pl. plug-in-ek)



# Felhasználói interfész tervezése (UI design, GUI design)

## ■ Néhány általános alapelv, amire figyeljünk, ha UI-t tervezünk

- *Tanulhatóság.* Könnyen megtanulható legyen, ösztönözve a felhasználót arra, hogy minél előbb kezdje el használni.
- *Ismerős a felhasználónak.* A felhasználó számára ismerős fogalmakat használjon
- *Konzisztencia.* Az interfész tegye lehetővé, hogy hasonló funkciókat hasonlóan lehessen elérni / indítani.
- *Minimális meglepetés.* A szoftver viselkedése ne lepje meg a felhasználót.
- *Visszaállíthatóság.* Hiba után az interfész tegye lehetővé az újraindítást.
- *Felhasználó támogatása.* Az interfész adjon érdemi visszajelzést a felhasználói hibákra, és nyújtson segítséget a használatban.
- *Felhasználói sokféleség.* A UI tegye lehetővé, hogy a rendszert többféle felhasználó használhassa (pl. vakok, gyengénlátók, színtévesztők stb. ).



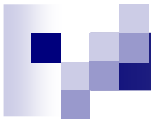
# Tervezés/ design a CMMI-ben

- Nincs egyetlen folyamat „Design” néven
- A Design ezekhez kapcsolódik :
  - Követelményfejlesztés (RD, ML3)
  - Műszaki megoldás (TS, ML3)

# Követelményfejlesztés (RD)

- Célja a vevői, termék, és termék-komponens követelmények felmérése, elemzése és dokumentálása.
- SG 1 Vevői követelmények fejlesztése
  - SP 1.1 Szükségletek felderítése
  - SP 1.2 Érdeelt felek igényeinek vevői követelményekké alakítása
- SG 2 Termékkövetelmények fejlesztése
  - SP 2.1 Termék és termék-komponens követelmények meghatározása
  - SP 2.2 Termék-komponens követelmények allokálása
  - SP 2.3 Interfész követelmények azonosítása
- SG 3 Követelmények elemzése és jóváhagyása
  - SP 3.1 Működési elképzelések és forgatókönyvek meghatározása
  - SP 3.2 Az igényelt funkcionalitás és minőségi jellemzők definiálása
  - SP 3.3 Követelmények elemzése
  - SP 3.4 Követelmények elemzése egyensúlyi állapot eléréséhez
  - SP 3.5 Követelmények validálása
- Ezek a tevékenységek a CMMI 3-as érettségi szintjén szükségesek! Mélyebb szakmai tudást feltételeznek!





# Requirements development

- The purpose of Requirements Development (RD) is to elicit, analyze, and establish customer, product, and product component requirements.
- SG 1 Develop Customer Requirements
  - SP 1.1 Elicit Needs
  - SP 1.2 Transform Stakeholder Needs into Customer Requirements
- SG 2 Develop Product Requirements
  - SP 2.1 Establish Product and Product Component Requirements
  - SP 2.2 Allocate Product Component Requirements
  - SP 2.3 Identify Interface Requirements
- SG 3 Analyze and Validate Requirements
  - SP 3.1 Establish Operational Concepts and Scenarios
  - SP 3.2 Establish a Definition of Required Functionality and Quality Attributes
  - SP 3.3 Analyze Requirements
  - SP 3.4 Analyze Requirements to Achieve Balance
  - SP 3.5 Validate Requirements

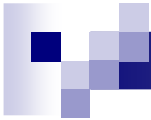






# Műszaki megoldás

- A műszaki megoldás célja a követelmények szerinti megoldások tervezése, fejlesztése és megvalósítása.
- SG 1 Termék-komponens megoldások kiválasztása
  - SP 1.1 Alternatívák és kiválasztási kritériumok kidolgozása
  - SP 1.2 Termék-komponens megoldás kiválasztása
- SG 2 A (műszaki) terv fejlesztése
  - SP 2.1 A termék vagy termék-komponens tervezése
  - SP 2.2 Technikai adatcsomag meghatározása
  - SP 2.3 Interfész-használati kritériumok megtervezése
  - SP 2.4 Elemzés: készítés, vásárlás vagy újrafelhasználás?
- SG 3 Termék fejlesztési modell implementálása
  - SP 3.1 Fejlesztési modell implementálása
  - SP 3.2 Terméktámogatási dokumentáció elkészítése



# Technical solution

- SG 1 Select Product Component Solutions
  - SP 1.1 Develop Alternative Solutions and Selection Criteria
  - SP 1.2 Select Product Component Solutions
- SG 2 Develop the Design
  - SP 2.1 Design the Product or Product Component
  - SP 2.2 Establish a Technical Data Package
  - SP 2.3 Design Interfaces Using Criteria
  - SP 2.4 Perform Make, Buy, or Reuse Analyses
- SG 3 Implement the Product Design
  - SP 3.1 Implement the Design
  - SP 3.2 Develop Product Support Documentation





# Design az Automotive SPICE-ban

- Nagyon fontos a beágyazott rendszereknél!
- Architekturális elemek:
  - Az architektúra rendszer és szoftver szinten kerül megtervezésre
    - A rendszert különböző szintű architekturális elemekre bontják
    - A szoftvert is felbontják különböző szintű architekturális elemekre, míg eljutnak a legalacsonyabb szintű elemhez; ezt szoftver komponensnek nevezik.



# A szoftvertervezés / design dokumentálása

## ■ Design dokumentáció

- ☐ Magas szintű terv / High level design
- ☐ Részletes terv / Low level design
- ☐ Modellek, leírások, döntések alapjának leírása
- ☐ **Verziókezelés a design dokumentumoknál is fontos, elengedhetetlen!**
  - Pl. tesztelés során architektúrális hibát javítunk – de az architektúra leírását elfelejtjük módosítani

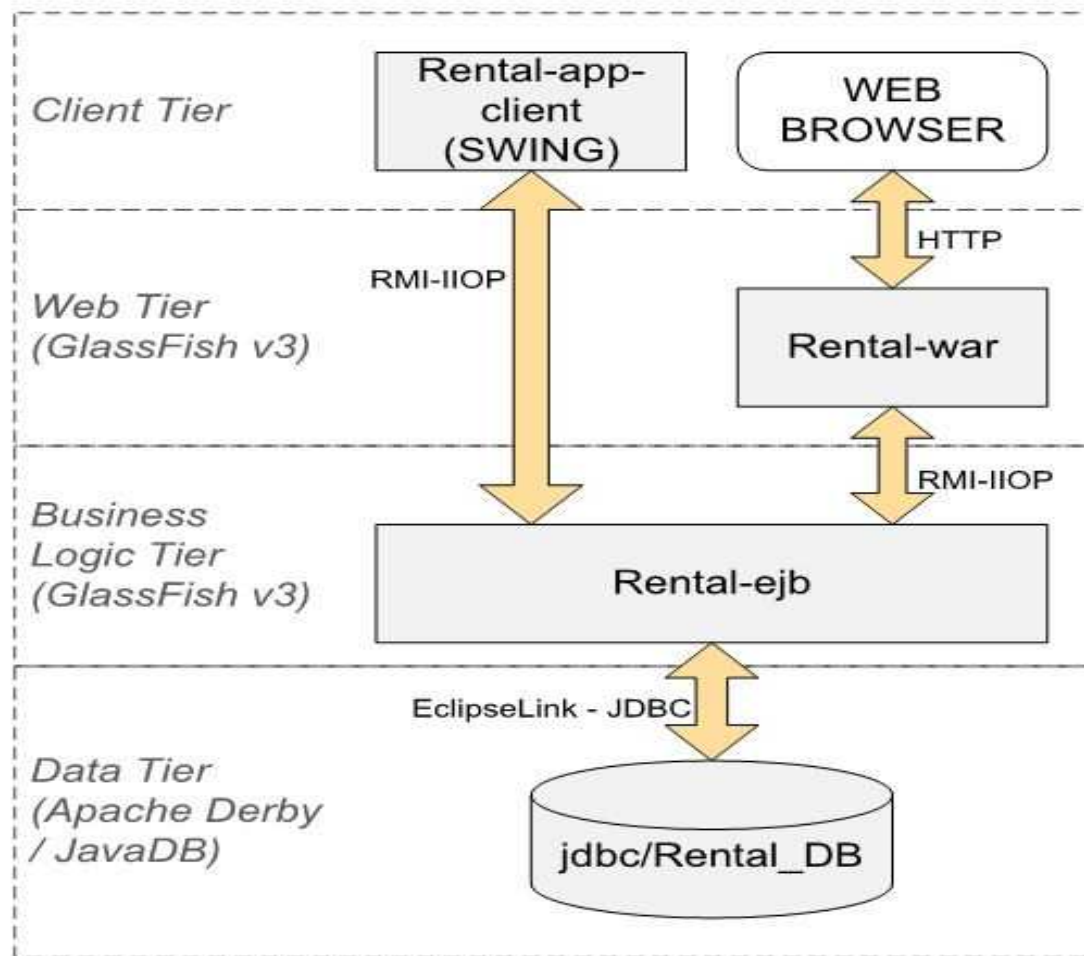


# A szoftvertervezés / design dokumentálása

- A termék és termék komponens terveknek nemcsak az implementálás számára, hanem további fázisok számára is megfelelő tartalmat kell szolgáltatniuk. Például, a módosítás, karbantartás, fenntartás és bevezetés / installáció számára is.
- A **design** dokumentáció segít közös nevezőre hozni a különböző érdekelt felek értelmezését a rendszerről (pl. fejlesztők, tesztelők), és támogatja a későbbi, ellenőrzött módon végzett változtatásokat is.
- A teljes design leírást az un. **műszaki adatcsomag (technical data package)** tartalmazza, amelyben minden fontos információ megtalálható a termékről ( funkciók, interfészek, programozási sajátosságok stb.).

□ Based on CMMI –DEV v1.3

# Architektúra ter. Példa.





# Részletes terv (példák)


- 1016-2009 - IEEE Standard for Information Technology--Systems Design--Software Design Descriptions
- SDD template-ek hozzáférhetőek
  - ☐ A rendszer átfogó leírása
  - ☐ Architektúra terv
  - ☐ Adatterv
  - ☐ Komponensek terve
  - ☐ Humán interakció terve (képernyők...)



# Mikor jó egy szoftver design / terv?

- Szabványos, érthető
  - Önmagával konzisztens
  - Nem redundáns...
- 
- Minőségi attribútumokat lehet megfogalmazni a szoftver tervvel kapcsolatban!





# A szoftver design / terv minőségének megítélése

- Különböző eszközöket és technikákat alkalmazhatunk a szoftver **design** minőségének megítélésében. További részletek a Tesztelés és Minőség előadásokban!
  - Szoftver design review: informális vagy formális technikákkal
  - Statikus elemzés: formális vagy félformális statikus elemzéssel ellenőrizzük, hogy a terve különböző nézőpontjainak elemei egymással konzisztensek-e
  - Szimuláció, prototípus készítés : a design minőségét vizsgáló dinamikus technika



# A szoftvertervezés – mesterség!

- A szoftver tervezése sajátos képességeket és mesterségbeli tudást igényel.
- Szoftvertervezők / elemzők végzik
  - Eredeti szakképesítésük szerint lehetnek:
    - Felhasználó- közeli, üzlet-közeli ( általában a magas szintű tervekhez)
    - IT szakértők (a részletes tervekhez)
    - Az a legjobb, ha van tervezői csapat !



# A szoftvertervezés – mesterség!

- A jó szoftvertervező **több lehetőséget, alternatívát is megvizsgál**, értékkel, pl:
  - A fejlesztés, gyártás, beszerzés, karbantartás, támogatás költsége
  - A kulcsfontosságú minőségi attribútumok várható értéke az egyes esetekben, pl. Elkészülési határidő, biztonság, megbízhatóság, karbantarthatóság
  - A termék komponensek komplexitása és a komponensek elkészítésének élelciklusa
  - A termék robusztussága a működés, használat körülményei, lehetséges működési környezetek viszonylatában
  - A termék várható kiterjesztése, növekedése
  - Technológiai korlátok
  - Mennyire érzékeny a termék a fejlesztés módszereire és a beépített elemekre
  - Kockázatok
  - A követelmények és a technológia fejlődése
  - Forgalomból való kivonás
  - A végfelhasználók és az operátorok képességei és korlátaik
  - COTS termékek jellemzői



# Mint jövőbeli szoftvertervező, jó, ha tudja...

- Nincs egyetlen, egységes tervezési alapelv vagy elem- halmaz, melyet mindenki, mindenütt elfogadna. DE vannak alapelvek és alapelemek, amelyeket figyelembe kell venni szoftvertervezés során.
- A tapasztalat segíti a szoftvertervezőt!
  - Legyen **nyitott** és **türelmes**, ameddig összegyűjt annyi tapasztalatot, amennyi ahhoz szükséges, hogy igazán jó szoftvertervező legyen!
  - Tanuljon meg felismerni mintákat, tervezési dokumentum- elemeket, és használja őket, ha hasonló esettel találkozik!



# Implementációs kérdések

- Nem csak kódolás!
- Hanem még:
  - ☐ Újrafelhasználás
  - ☐ Konfiguráció menedzsment
  - ☐ „Host-target” fejlesztés
  - ☐ ...



# Implementáció

- Kódolás, technikai / műszaki **dokumentáció elkészítése**
- **Kódolási szabványok!**
  - Struktúra, kommentek, változók elnevezése ...
  - Kerüljük el a sokak által elkövetett hibákat
    - Szervezeti folyamat-elemek, jó és rossz kódolási példákkal
  - A programozási jó gyakorlatok ismerete
- Korábbi tapasztalatok újr felhasználása



# Az implementáció dokumentálása

- Forráskód + egyéb műszaki dokumentáció
- Minden döntés, tapasztalatok ...
- Vigyázzunk a kód változásaival!
- Kétirányú követhetőségnek kell lennie a követelmények- design- kód- teszt eset között !
- Konfigurációmenedzsment és verziókezelés fontos!



# Becslések a szoftverfejlesztésben

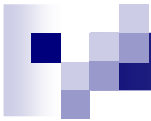
- A becslés segít megérteni, hogy mi is fog történni , illetve „milyen és mekkora” lesz a rendszer
- Szükség van korábbi / historikus adatokra!
- Becslést az alábbiakra szoktunk végezni:
  - ☐ Ráfordítás – a PM előadásban lesz róla szó
  - ☐ Költség – a PM előadásban lesz róla szó
  - ☐ Méret (Size)
- További becslések származtathatók az alábbiakra :
  - ☐ Modulok és interfészek száma
  - ☐ Az egyes feladatok elvégzéséhez szükséges idő
  - ☐ Fázisonként bevitt, kijavított és bennmaradt hibák száma
  - ☐ Hibasűrűség
  - ☐ Stb.





# A szoftvertervezéshez kapcsolódó becslések

- A rendszer méretének becslése
  - ☐ Nem könnyű!
  - ☐ Vannak módszerek – de a tapasztalat elengedhetetlen
- A továbbiakban bemutatunk a tervezésben és kódolásban alkalmazható 2 becslési technikát :
  - PROBE módszer (a PSP-ben használatos)
  - Funkciópont-számolás

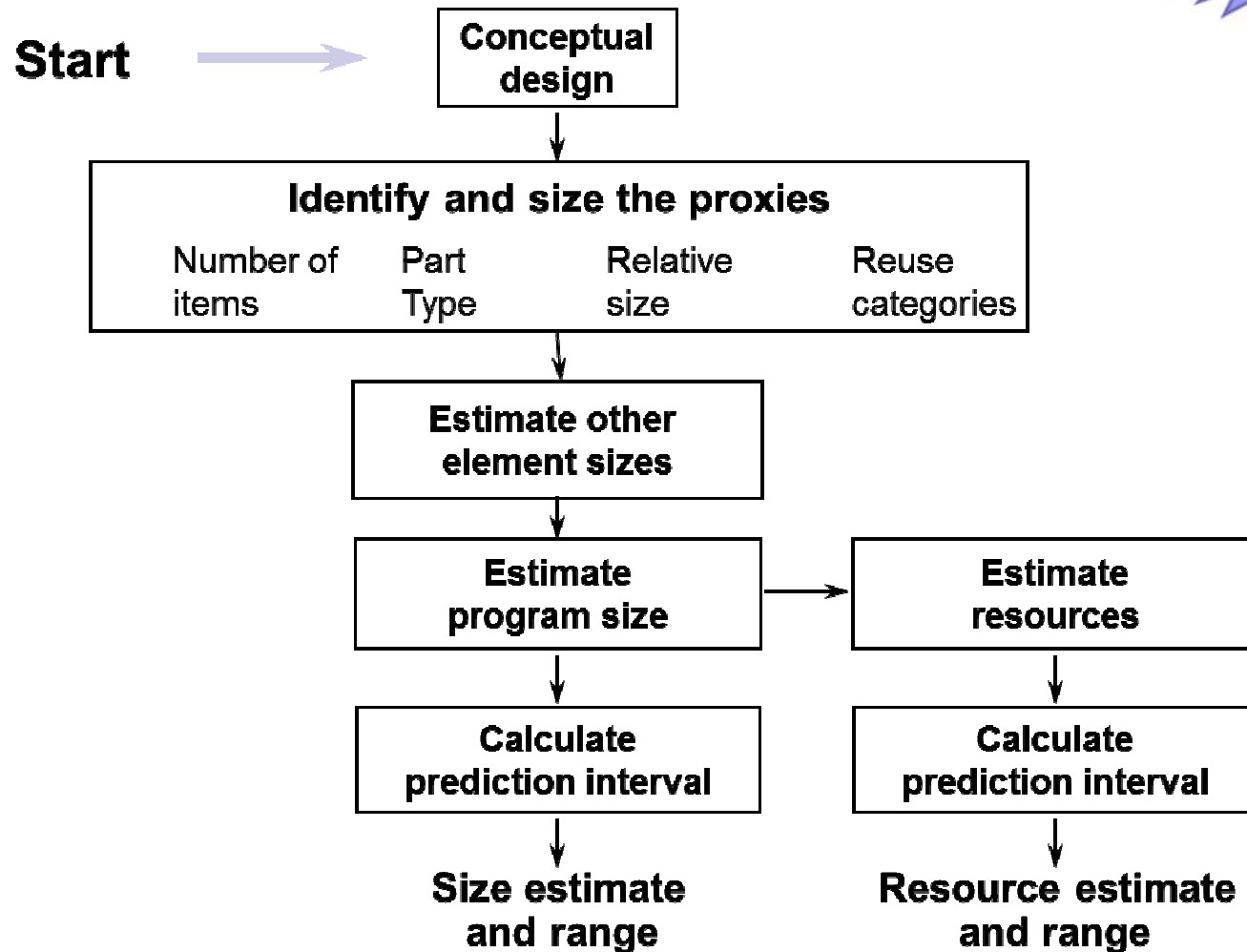


# PROBE módszer - I



- A PSP a PROBE módszert használja becslésre és a projektek megtervezésére
- PROBE = PROxy Based Estimating.
- PROBE proxykat használ (= helyettesítők) a program méretének és a fejlesztési időnek a becslésére
- A jó proxy segít a pontos becslésben.

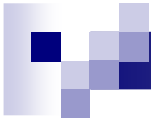
# PROBE módszer - II





# PROBE módszer - III

- Az első lépés egy fogalmi modell készítése
  - A követelményeket rendeljük hozzá a termékhez
  - Definiáljuk azokat a termék-elemeket, amelyek az elvárt funkciókat megvalósítják
- A legtöbb projekt esetében a fogalmi modellt viszonylag gyorsan el lehet készíteni.
- A PSP programok esetében próbáljuk ezt a fázist 10-20 percre korlátozni



# PROBE módszer - IV



- Megfelelő proxy kiválasztása:
  - ☐ A fejlesztési ráfordításra
  - ☐ A proxy tartalma automatikusan számolható legyen
  - ☐ A projekt elején könnyen vizualizálható
  - ☐ A felhasználó szervezet igényeihez igazítható
  - ☐ A különböző implementációs verziókra érzékeny



# PROBE módszer - V



## ■ Lehetséges proxyk:

- ☐ Objektumok → A PSP-ben használatosak
- ☐ Funkciópontok
- ☐ Képernyők
- ☐ File-ok
- ☐ Scriptek
- ☐ Dokumentum fejezetek



# PROBE módszer - VI



- Az első lépés a fogalmi modell elkészítése volt
- A második lépés az objektumok azonosítása :
  - Határozzuk meg az objektum típusát és relatív méretét
  - Azonosítsunk újrafelhasználható kategóriákat
    - Újrafelhasználható könyvtár
    - Újabb újrafelhasznált objektumok

# PROBE módszer - VII



- Határozzuk meg az objektum típusát és relatív méretét
- Példa: **C++** objektumok és méreteik

C++ Object Sizes in LOC per Method					
Category	Very Small	Small	Medium	Large	Very Large
Calculation	2.34	5.13	11.25	24.66	54.04
Data	2.60	4.79	8.84	16.31	30.09
I/O	9.01	12.06	16.15	21.62	28.93
Logic	7.55	10.98	15.98	23.25	33.83
Set-up	3.88	5.04	6.56	8.53	11.09
Text	3.75	8.00	17.07	36.41	77.66





# PROBE módszer - VIII

- A harmadik lépés a becsült objektum forrásokainak száma (LOC)
  - Beleértve a hozzáadott, módosított, törölt, bázis- és újrafelhasznált kódot
  - Egy program módosításakor a „báziskód” a módosítás által nem érintett , létező forráskód
  - Ha a programokat módosítjuk, a nem módosított méretüket a báziskódhoz kell hozzáadni.
- PROBE eszköztámogatás:
  - <https://www.processdash.com/static/help/frame.html?UsingProbeTool>  
=



# Funkciópont számolás

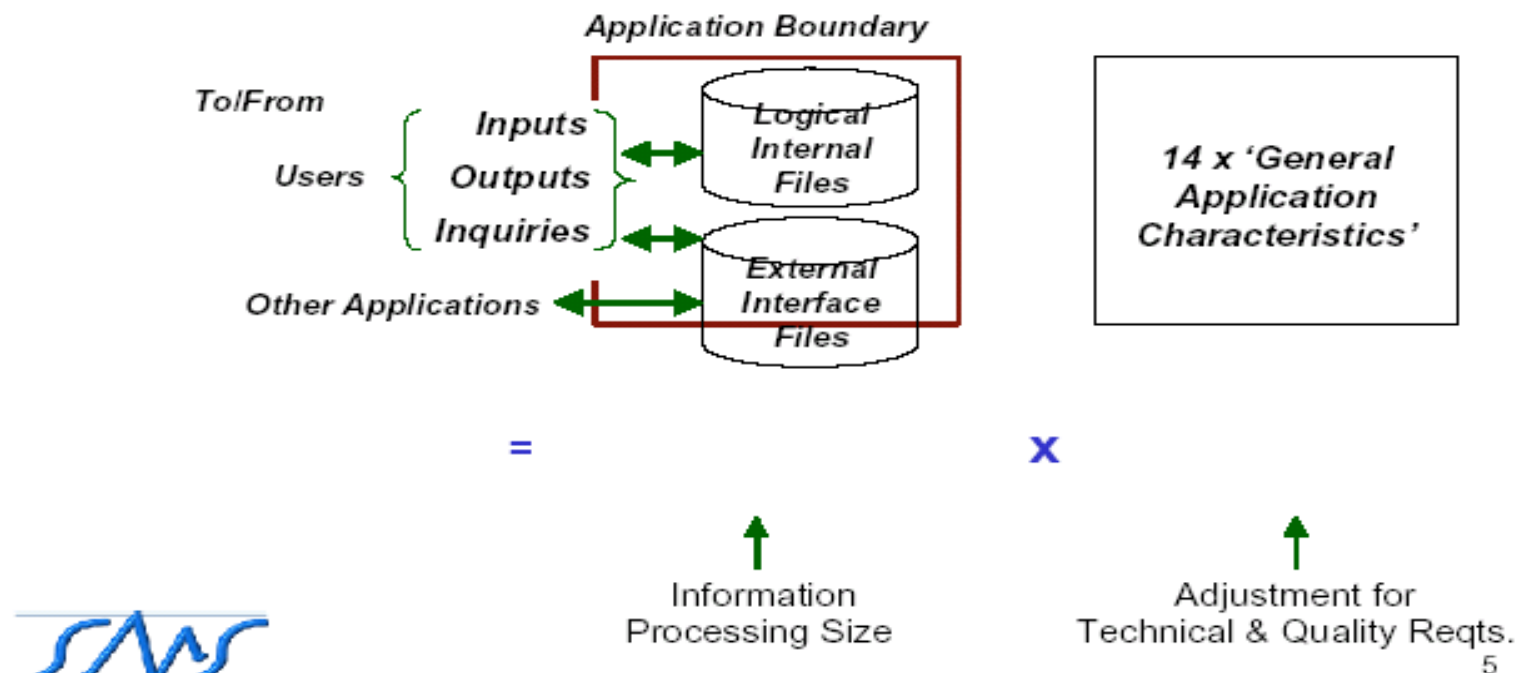
- Kifejlesztésének célja: különböző technológiákkal történő szoftverfejlesztések hatékonyságának összehasonlítása
- Albrecht céljai a funkciópont számolással:
  - a szoftver méretének következetes mértéke legyen
  - legyen független a fejlesztésben alkalmazott technológiától
  - alkalmazása legyen egyszerű, eredménye sokatmondó a végfelhasználónak (is)
- Később rájöttek, hogy a módszer jól alkalmazható a specifikáció alapján történő becsléskor

(Forrás: Charles Symons: Come back Function Point Analysis (Modernised)- all is Forgiven!  
Software Measurement Services  
10th May 2001, FESMA Conference )



# Albrecht funkciópont modellje

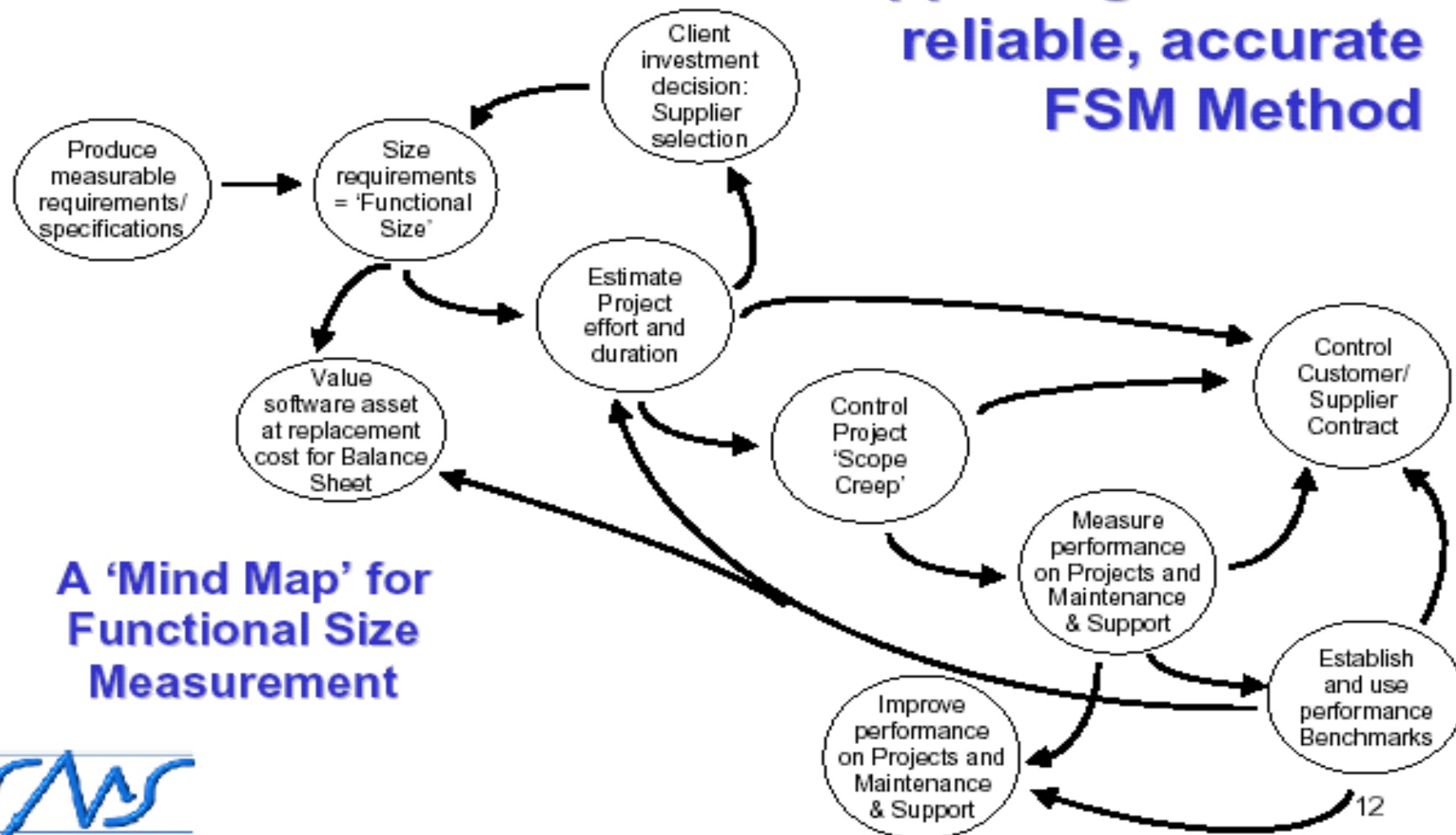
■ 1970-ből



Bizonyos jellemzőket figyelünk, és 0-5 között súlyozzuk jelenlétüket, jelentőségüket.

# Hogyan használható a funkciópont számolás?

**Supposing we had a  
reliable, accurate  
FSM Method**



**A 'Mind Map' for  
Functional Size  
Measurement**



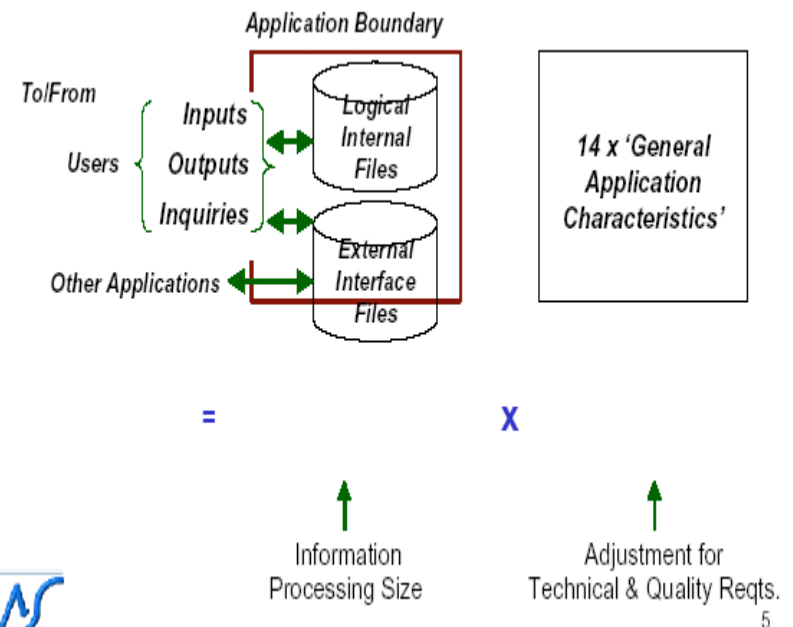


# A funkciópont számolás

- ... legyen egyszerű
- ... kapcsolódjék a cégek egyéb formában mért adataihoz
- ... legyen nemzetközi
- ... kapcsolódjék jobban a (többi) szoftver metrikához

# A funkciópont számolás módszerei

- IFPUG: Albrecht módszere
- Néha nehéz a komponenseket meghatározni, nincsenek pontos definíciók
  - pl. mit jelent „logikai file” az OO környezetben?
  - Hogyan kezeljük azokat a képernyőket, amelyek inputot és outputot is tartalmaznak?
  - Belső komplexitás meghatározása nem definiált (u. az a komplexitása egy egyszerű transzformációnak és egy nagyon bonyolult folyamatnak is, mert csak az I / O-t veszi figyelembe





# A funkciópont számolás módszerei

## ■ IFPUG

- 😊 legtöbbet használták, sok a tapasztalat, esettanulmány
- 😊 nagy nemzetközi szervezeti háttér (képzés, konzultáció, certificate...)
- 😊 adatfeldolgozó rendszerekben jól használható
- 😞 valós idejű rendszereknél nem / nehezen használható
- 😞 csak alkalmazás - típusú szoftverekre használható
- 😞 a módszer struktúrája és a benne használt súlyozások (fontossági sorrendek) kérdéses



# A funkciópont számolás módszerei

## ■ A MKII módszer

### ■ A számolás (mérés) lépései a következők:

- .A számolási nézőpont, cél és típus meghatározása
  - Meg kell határozni, hogy mit fogunk számolni / mérni.
- .A számolás korlátainak meghatározása
  - Meghatározzuk, hogy mi képezi a számolás tárgyát, és mi nem tartozik bele.
- .A logikai tranzakciók azonosítása
  - A logikai tranzakciók a rendszer legalacsonyabb szintű folyamatai (processzei), amelyek még beletartoznak a számolás területébe.
- Az entitások típusának meghatározása és besorolása
  - Jó, ha van egy entity-relationship modell, amelynek alapján ezt az azonosítást és besorolást el lehet végezni.





# A MKII módszer

- A számolás (mérés) lépései (folytatás):
  - A bemenő adat-elemek típusának, a meghivatkozott entitás típusoknak és a kimenő adat-elemek típusának megszámlálása
    - Minden logikai tranzakció esetében meg kell határozni a bemenő adat típusokat ( $N_i$ ), a meghivatkozott entitás típusokat ( $N_e$ ) és a kimenő adat típusokat ( $N_o$ ).
  - A funkcionális méret kiszámolása
    - A következő képlettel történik:
    - $FPI = W_i \times \sum N_i + W_e \times \sum N_e + W_o \times \sum N_o$ , ahol
    - FPI = Function Point Index
    - $W_i$ ,  $W_e$  és  $W_o$  az  $N_i$ ,  $N_e$  és  $N_o$  értékek súlyozott átlaga, éspedig:  $W_i = 0.58$ ,  $W_e = 1.66$  és  $W_o = 0.26$ .
- A számolás (mérés) lépései (folytatás):
  - A projekt ráfordításának meghatározása
    - Határozzuk meg a teljes ráfordítást és az eltelt időt.
  - A termelékenység és egyéb mutatók kiszámolása
    - Példa: Termelékenység =  $FPI / \text{projekt ráfordítás}$



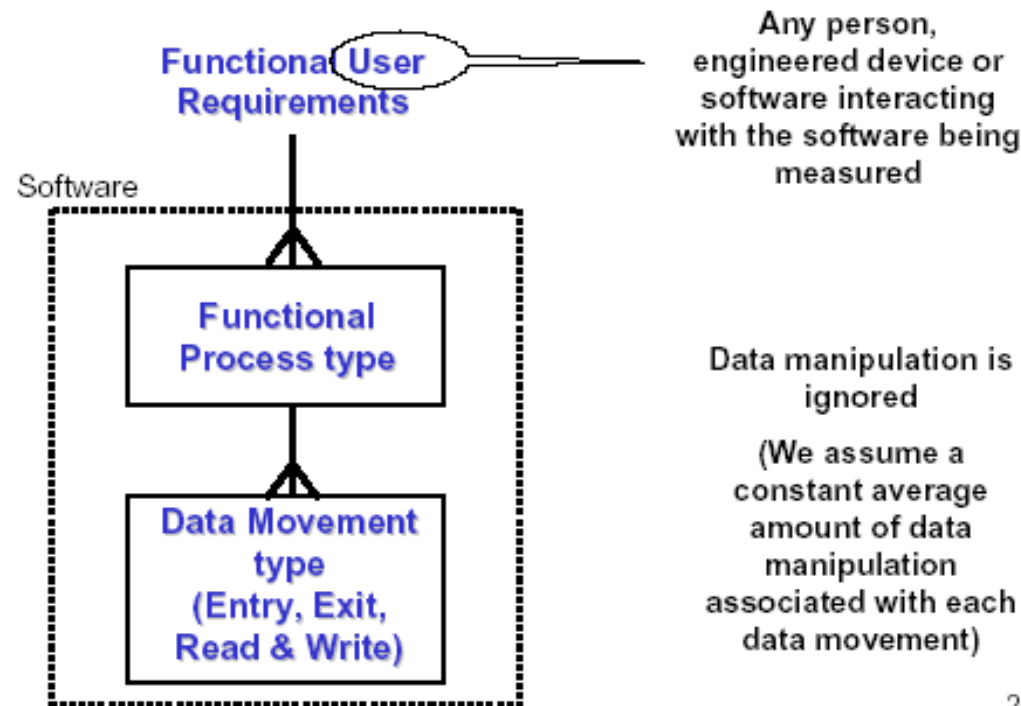
# A funkciópont számolás módszerei

## ■ MkII

- ☺ Menedzsment rendszerekre találták ki, és sokat is használták, jól lehet becslésben alkalmazni
- ☺ Az IFPUG továbbfejlesztése, több adatot tartalmazó rendszerre
- ☺ A strukturált elemzési módszerekkel konzisztens
- ☺ Az életciklus korai fázisában alkalmazható
- ☺ Nagy támogatottság, képzés, certificate...( de kevesebb az IFPUG-nál)
- ☹ használták már valós idejű rendszereknél, de ilyenkor át kell értelmezni
- ☹ csak alkalmazás - típusú szoftverekre használható

# A Cosmic módszer

- <https://cosmic-sizing.org/publications/early-estimating-using-cosmic-ffp/>
- A szoftver funkcionalitásának egyszerű modelljén alapszik





# A Cosmic módszer

## ■ Méretezési szabályok

### □ Egy funkcionális processz mérete

- az adatmozgások számának aritmetikai összege (bemenet, kimenet, írás, olvasás)
- minimális méret: 2
- maximális méret: nincs felső korlát

### □ Egy szoftverelem mérete a funkcionális processzei méretének összege



# A Cosmic módszer

- 😊 Egyszerű és egyértelmű
- 😊 Minden típusú szoftverre alkalmazható (MIR / MIS és valós idejű)
- 😊 Többrétegű architektúrák bármely komponensének esetében is alkalmazható
- 😊 Használták már: az IBM-nél, OO fejlesztési projektek becslésében, távközlésben, repülőgép-iparban...
- 😞 Kevesebb a tapasztalat, kevesebb esettanulmány
- 😞 Kevésbé részletes számolási útmutatók



# Cosmic példa

Egyszerű adatbázis-lekérdezés

Követelmény: meg kell tudni adni egy felhasználó azonosítóját, és a képernyőn meg kell jelentetni az ahhoz a felhasználóhoz tartozó összes információt

Egy **Entry** a Felhasználói ID-re

Egy **Read** a Felhasználó -rekord megtalálására

Egy **Exit** a Felhasználó adatainak megjelenítésére

Egy **Exit** az összes lehetséges hiba- és jóváhagyási üzenetre


---

**Teljes méret: 4 Cfsu**

(Összehasonlítás:

IFPUG: Simple EQ of 3 UFP's, plus allowance for files

MkII FPA: 1 x input DET, 1 x ER, assume 10 output DET's = 5.1 FP)



# Funkciópont számolási szabványok

- [COSMIC: ISO/IEC 19761:2011](#) Software engineering. A functional size measurement method.
- FiSMA: [ISO/IEC 29881:2010](#) Information technology – Systems and software engineering – FiSMA 1.1 functional size measurement method.
- [IFPUG: ISO/IEC 20926:2009](#) Software and systems engineering – Software measurement – IFPUG functional size measurement method.
- Mark-II: [ISO/IEC 20968:2002](#) Software engineering – MI II Function Point Analysis – Counting Practices Manual
- NESMA: [ISO/IEC 24570:2005](#) Software engineering – NESMA function size measurement method version 2.1 – Definitions and counting guidelines for the application of Function Point Analysis



# Tervezés / design agilis környezetben

- Lényeges : **a megoldás korai bemutatása / kipróbálása.**
- A megoldásokat a következőképpen mutathatjuk meg:
  - Funkciók, funkciócsoportok, release – és egyéb komponensek, amelyek támogatják a végső megoldás kifejlesztését
- Ha később az eredeti fejlesztőkön kívül más fogja továbbfejleszteni a rendszert, kiemelten fontos a **tervezési dokumentáció** átadása is.
- A tervezési, fejlesztési döntések, feltételezések alapját is dokumentálni kell



# Agilis becslés

- A cél ugyanaz, mint a hagyományos becsléskor
- Az alkalmazott technikák különbözhetnek



[http://www.codingthearchitecture.com/2010/07/13/estimating\\_a\\_software\\_system.html](http://www.codingthearchitecture.com/2010/07/13/estimating_a_software_system.html)



# Agilis design elemek

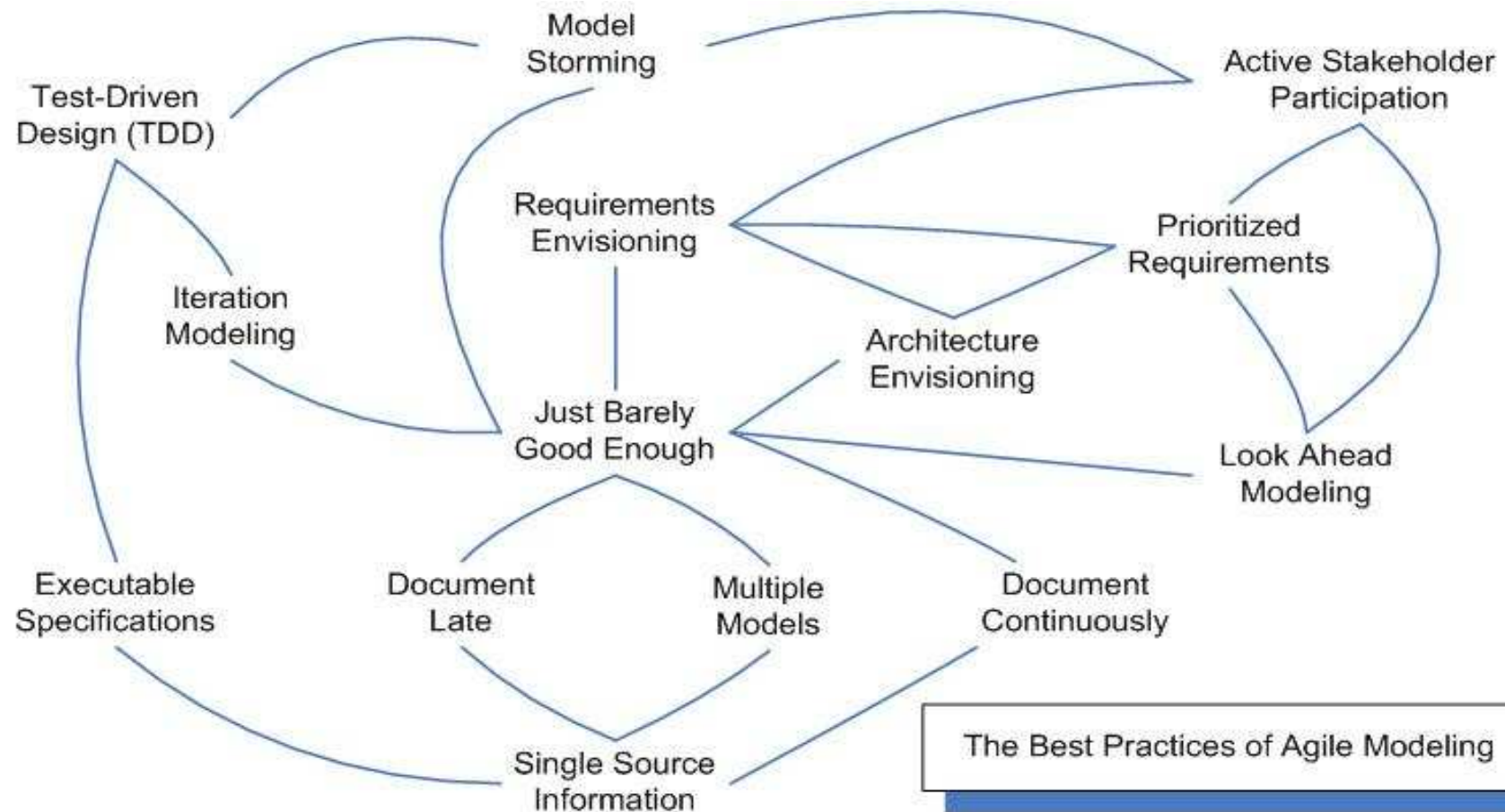
## ■ Magas szintről indulunk

- Elképzeljük a rendszer architektúráját; csak a kritikusnak gondolt elemeket
- Az iterációk elején pár perces modellezést végzünk
- „Model storming” – pár percben , ötletelés, „Just in time” – mindig azt az elemet, amellyel akkor foglalkozunk
- „Test-first design” – írunk egyszerű tesztet, majd a hozzá tartozó kódot
- Refaktorizálás / refaktorálás– kis módosítások a kódon, hogy „jobb, szebb” legyen
- Continuous integration- automatikusan fordítsuk, teszteljük, validáljuk a komponenseket minden változáskor

## ■ Eljutunk a kódig

- Scott W. Ambler alapján, <http://www.ambysoft.com/books/agileModeling.html>

# Agilis design elemek



Copyright 2005-2011 Scott W. Ambler

<http://www.agilemodeling.com/>



# Agilis és hagyományos tervezés

Hagyományos tervezés	Agilis tervezés
Sorozatban történő munkafolyamat; checkout, egyszerre egy valaki tervez	Párhuzamos folyamat, egyszerre többen is terveznek
A feladatokat hetek, hónapok alatt végzik el	A tervezési feladatokat percek , órák, napok alatt végzik el
E-mail-ek, nyomtatás, meetingek	SMS, videohívás...
Íróasztalnál dolgozó, fix csapat	A csapattagok bárhol lehetnek
A tervezési adatokat a tervezők maguknál tartják; a tervezők” csak” terveznek	A tervezési adatok mindenki számára elérhetők, a csapattagok széleskörű tudással rendelkeznek
Az innovációs igények miatt megpróbálnak eszközöket használni	Az eszközök felgyorsítják a munkát és az innovációt
A menedzsment általában hónap végén, már lejárt adatokat kap	A menedzsment folyamatosan tájékozódik, valós időben

<https://www.onshape.com/cad-blog/agile-product-design-requires-a-new-generation-of-cad>



# Agilis design

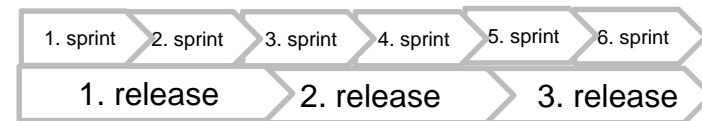
- Az agilis szoftvertervezés szerves része a User Story / felhasználói történet / story point meghatározása
- A követelmények meghatározásánál is láttuk ezeket
- Itt tovább részletezzük őket

# Agilis design

## ■ User Story meghatározása

### ☐ A következő release-re

- 1 iteráció ~ 1-2 hét (sprint)
- 2-5 iteráció = 1 release ~ 1-2 hónap
- 1 projekt ~ 3-6 hónap



### ☐ A követelményeket be kell fagyasztani! (???)

**“Vízen járni és specifikációból szoftvert fejleszteni egyszerű, ha mindkettő befagyasztott állapotban van.”**

[Edward Berard](#)

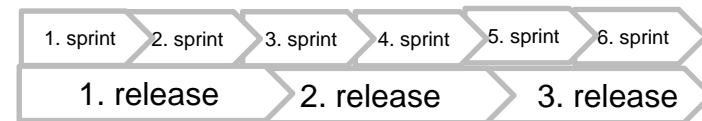
# Agilis design

## ■ User Story meghatározása

## ■ A következő sprintre

- Csak kis változtatások engedélyezettek
- Az ötletek letisztulnak
- Elérjük az implementációs szintet
- Fontos kommunikálni a felhasználóval!

## ■ NB.: a szoftvertervezés (design) és a projekttervezés egyszerre történik !!!!

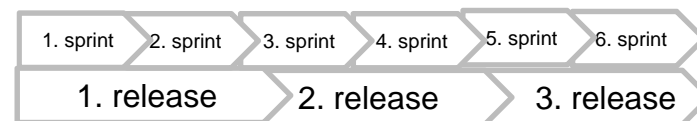


# Agilis design

## ■ User Story meghatározása

### □ A Sprint végén

- „Gyártásra-kész” funkciók
- A fejlesztő csapat mutatja be





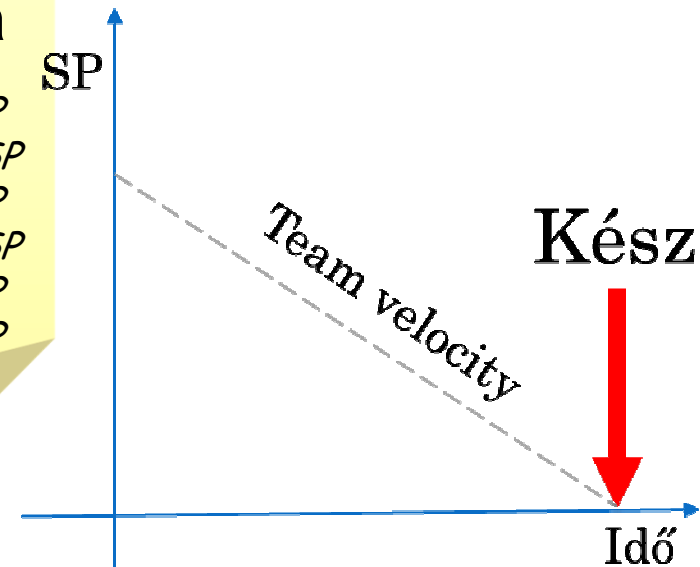
# Agilis design

## Elemek

- ☐ User story
- ☐ Lista
- ☐ Becslések
- ☐ Burndown chart
- ☐ Csapat sebessége /Team velocity

## User Story Lista

<i>Azonosítás</i>	<i>7SP</i>
<i>Regisztráció</i>	<i>10SP</i>
<i>Kijelentkezés</i>	<i>2SP</i>
<i>Felhasználó kezelés</i>	<i>15SP</i>
<i>Felh. Keresés</i>	<i>7SP</i>
<i>Adatlap nyomtatása</i>	<i>5SP</i>



<https://www.agilealliance.org/glossary/>

Minden iteráció végén a csapat összeadja az annak az iterációnak a során elkészült user story-k elkészítéséhez becsült ráfordítást. Ez az összeg **a csapat sebessége**.

A sebesség ismeretében a csapat kiszámolhatja (vagy felülbíráhatja) annak becslését, hogy a projekt mennyi ideig fog tartani; a becslés alapja a még fennmaradó story point-ok, és az a feltételezés, hogy a sebesség nagyjából ugyanaz marad majd.

# Agilis tervezés

## ■ User Story Meghatározás

- ☐ „Követelmény” – Nem túl jó kifejezés
- ☐ Ügyféllel együtt
- ☐ Just In Time

## ■ Fizikai vs. Virtuális

- ☐ Sok-sok fog keletkezni.
- ☐ Érdeemes virtuálisan tárolni
- ☐ Fizikai változat is előnyös

As **who** I want  
**what** so that  
**why**

## ■ Jól működhet: Fizikai reprezentálja a virtuálisat



# Agilis design

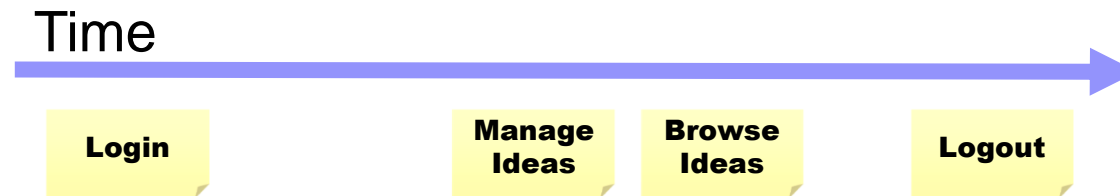
## ■ User Story Mapping

Time



# Agilis design

## ■ User Story Mapping



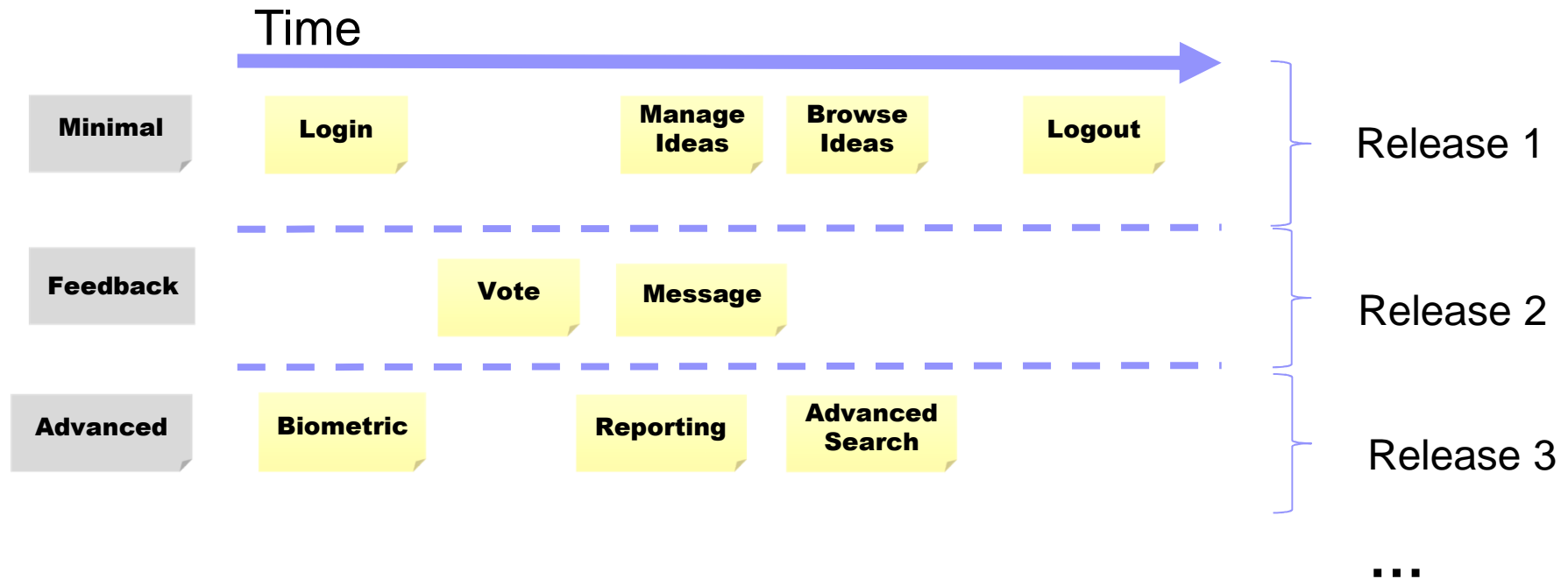
# Agilis design

## ■ User Story Mapping



# Agilis design

## ■ User Story Mapping



# Agilis design

## ■ User Story Mapping

### □ Eredmény:

- Letisztult életpálya terv
- Még tisztább kép a termékről
- További részleteket nyertünk ki

### □ Látjuk

- **Mi lesz a következő release?**
- A feature-öket ki kell bontani
- Fel kell becsülni
- Prioritizálni kell



# Agilis design

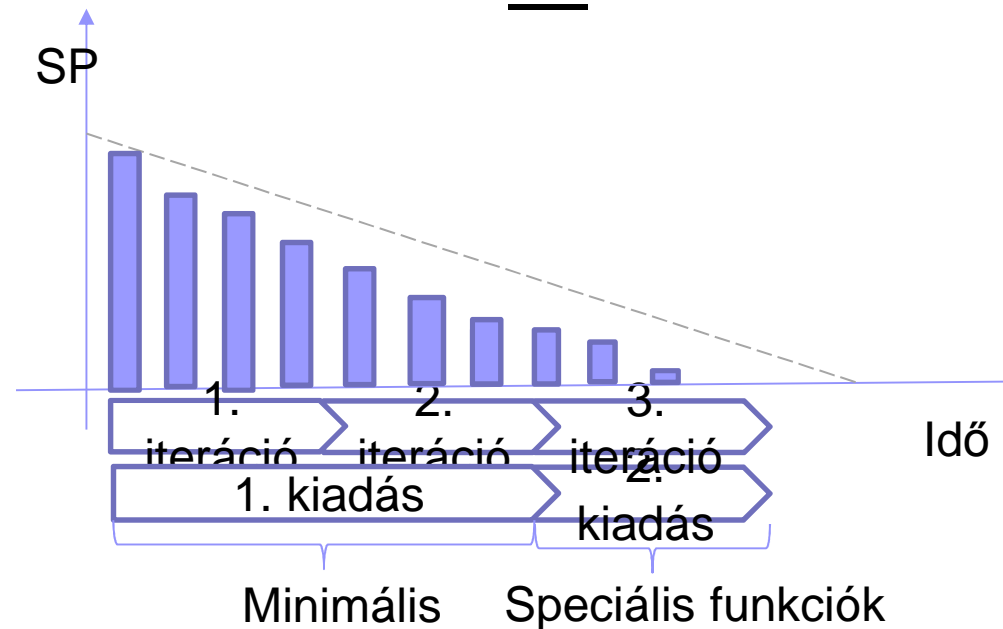
## ■ User Story Mapping





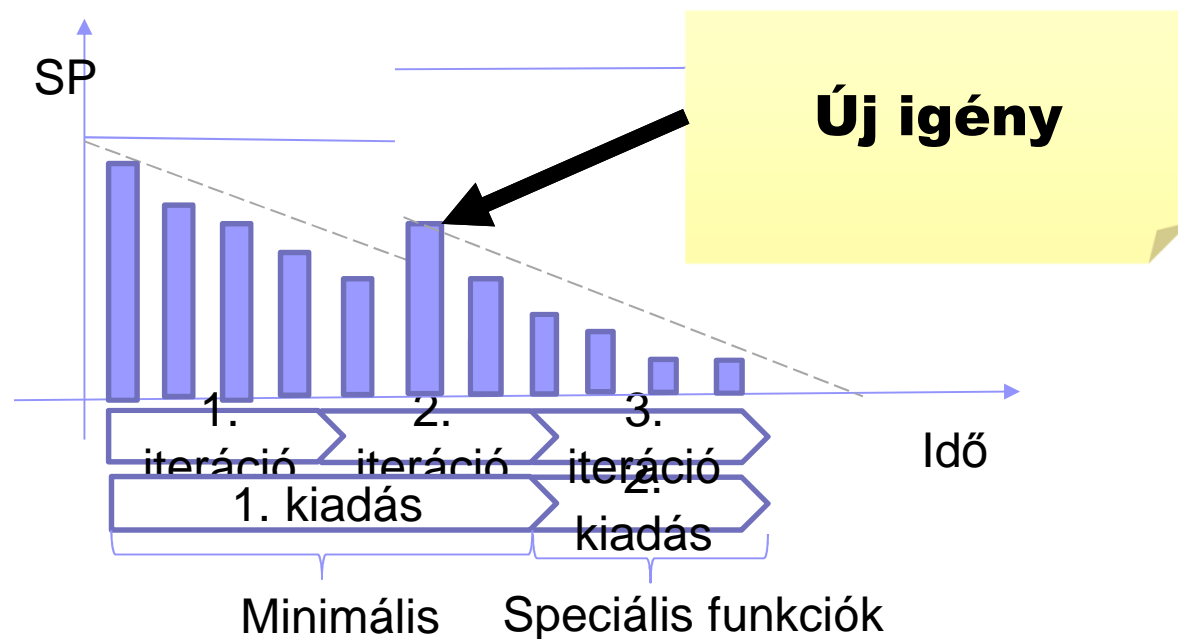
# Agilis design

- A burn-down chart
- Projektmenedzsment és műszaki tervezés!



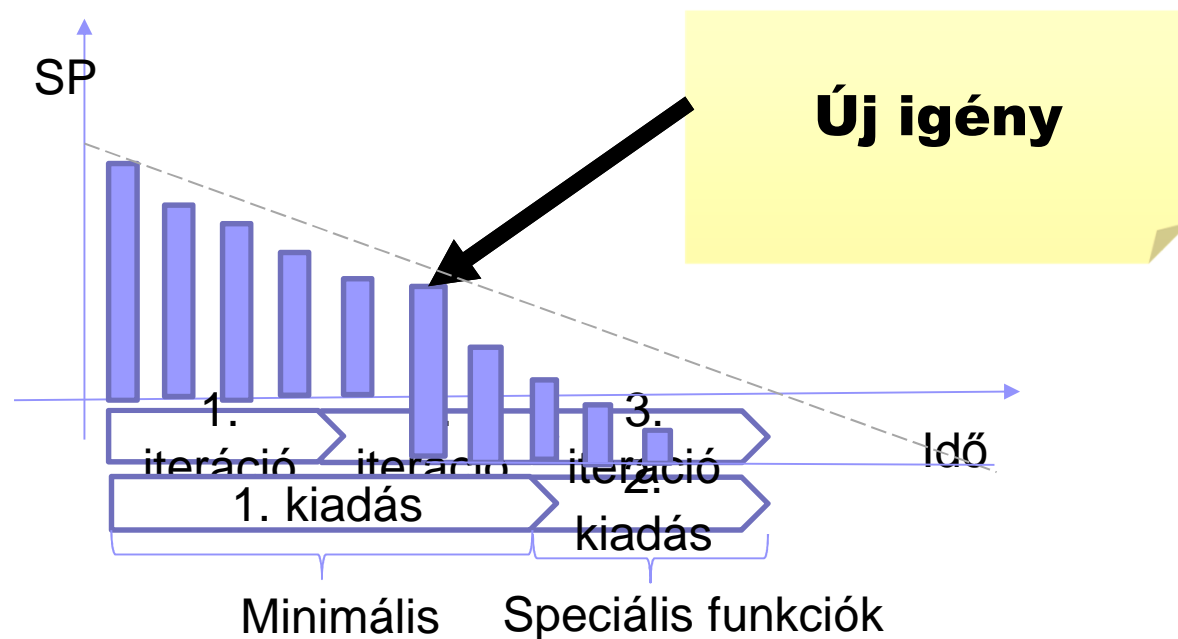
# Agilis design

## ■ A burn-down chart



# Agilis design

## ■ A burn-down chart





# Agilis design és projekttervezés

## ■ Az agilis tervezés

- Számos egyéb változat van a burn up/down-chart-ra
  - Részletkérdés, de legyen konzisztens
- Reális terv kell
- Ne számítsunk csodára.
- Kész
  - **Definicion Of Done**
    - Elemzett, tesztelt, implementált, stb...

**PLAN FIRST!**



# Agilis design

**I**ndependent

**N**egotiable

**V**aluable

**E**stimable

**S**mall (Sized appropriately)

**T**estable

(Független, egyeztethető,  
értékes, becsülhető,  
kicsi (megfelelő méretű),  
Tesztelhető)

„INVEST” azon kritériumok listája, amelyek a user story minőségét segítenek megítélni.

<https://www.agilealliance.org>



# Agilis design

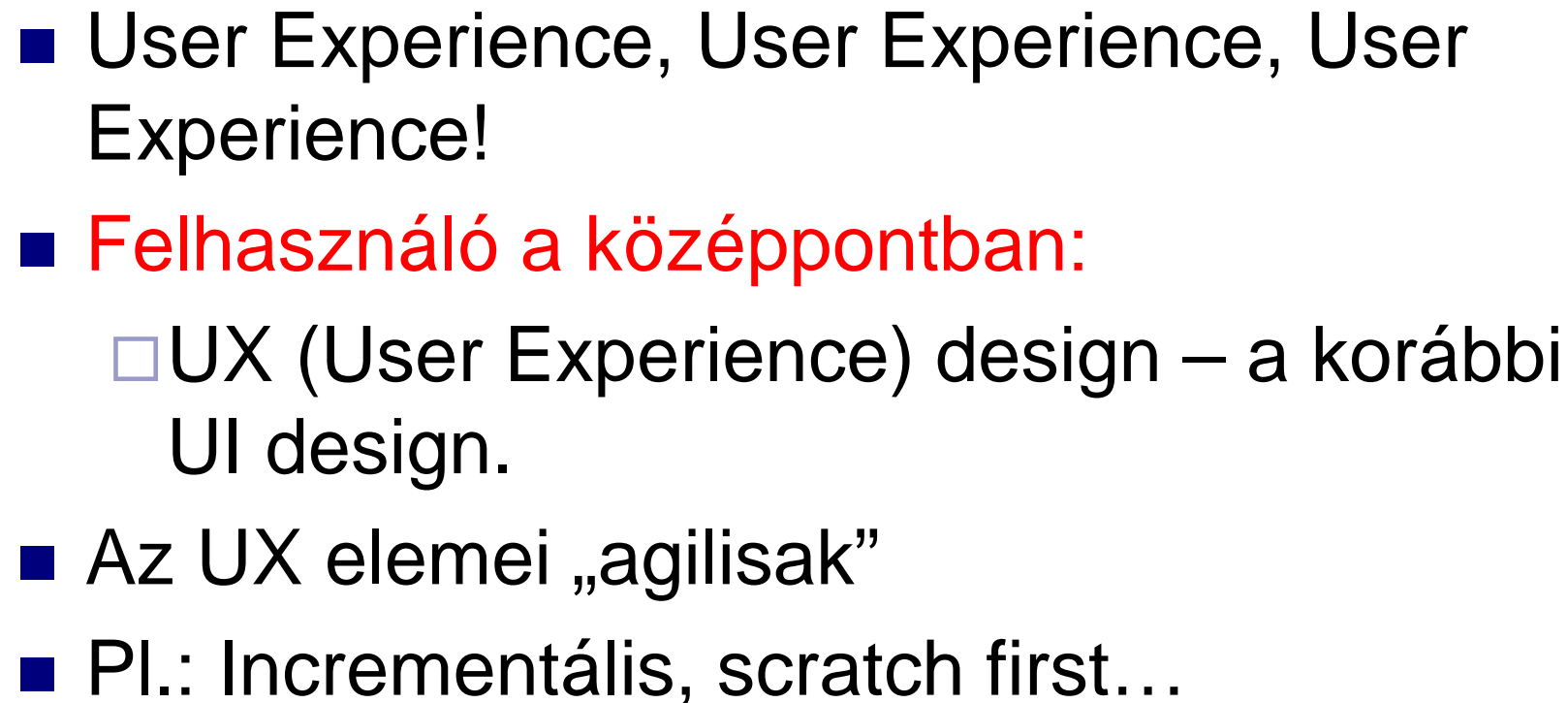
## ■ Definition of Done:

- Egyeztetett / mindenki által elfogadott listája a termék inkrementum elkészítéséhez elengedhetetlenül szükséges tevékenységeknek
- A csapat megegyezésre jut a DoD-t illetően, és ki is függesztik ezt a teremben egy jól látható helyre. Egy sor követelménynek teljesülni kell ahhoz, hogy a termék adott inkrementumát (általában egy user story-t) elkészültnek tekintenek.
- Ha a kritériumok nem teljesülnek a sprint végére, az adott tevékenységeket nem számolják bele a csapat sebességébe.



# Agilis design

- Vegyük észre, hogy a „szoftvertervezés” (mint műszaki, mérnöki munka) és a „projekttervezés” (mint menedzsment feladat) átlapolódnak az agilis környezetben!





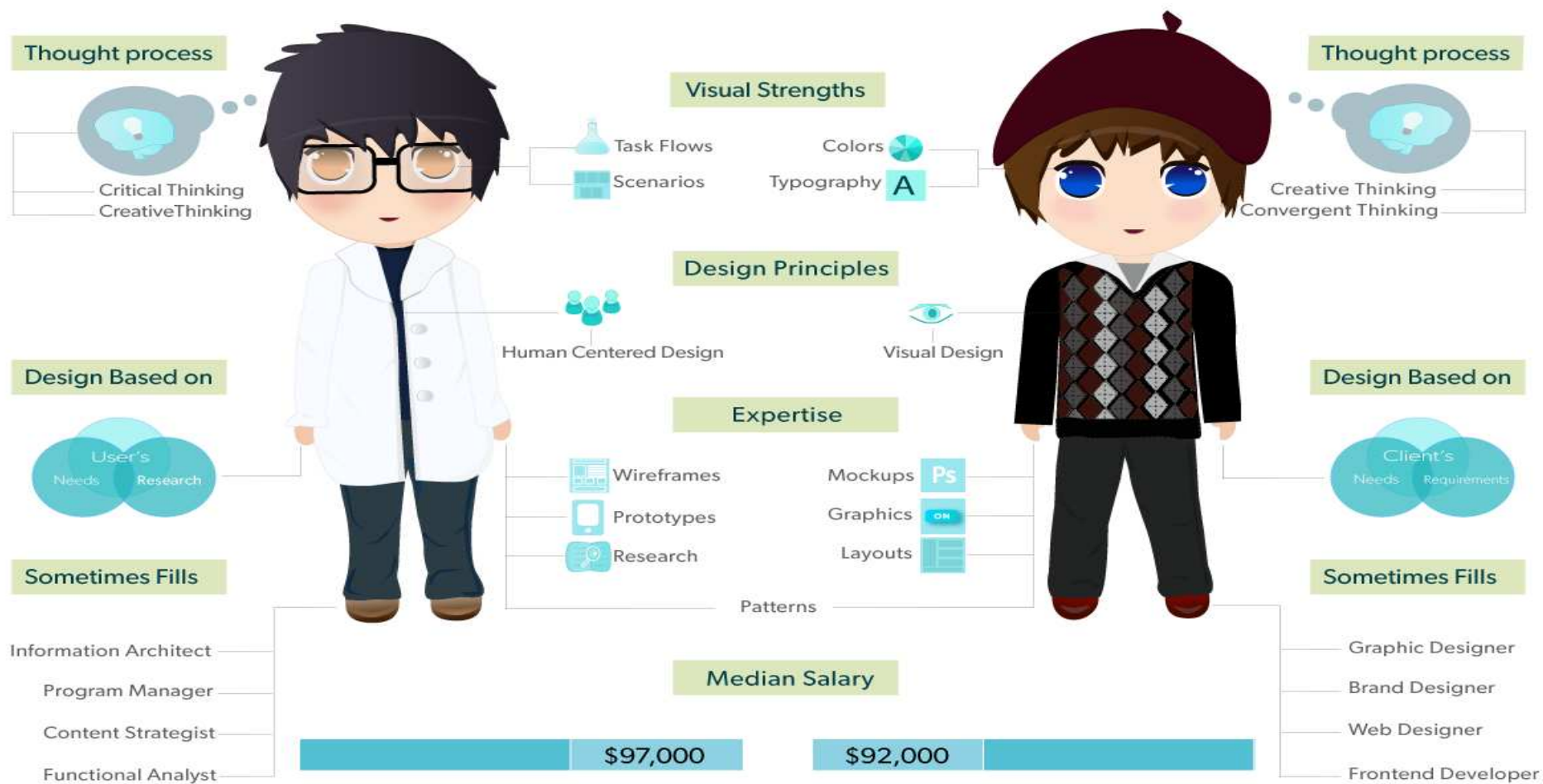
# Agilis szoftvertervező



# UX Designer

# VS

# UI Designer



Infographic created by Ana Harris

# Agilis „business analyst” (üzleti elemző)

- User Story-k rögzítésében segít
- A részletek kidolgozásában segít
- **Just In Time** - Fontos



# Agilis „business analyst” (üzleti elemző)

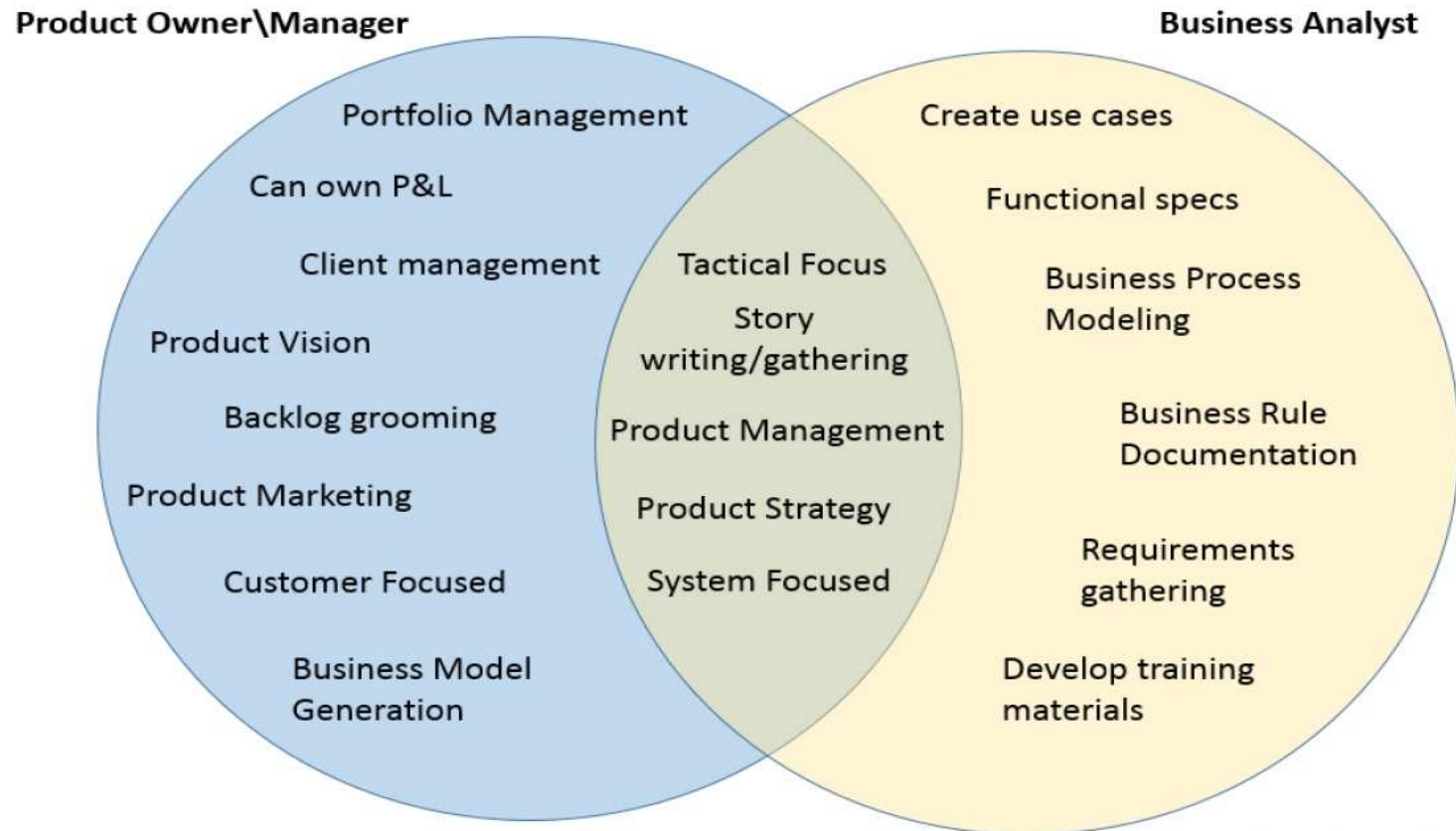


Chart by Jeff Gothelf

<http://masteringbusinessanalysis.com/mba045-the-agile-ba-myths-and-misconceptions/>

# Agilis fejlesztési technikák: Extreme programming





# Extreme Programming

- Extreme Programming (XP) – eredetileg Kent Beck által bevezetett fogalom - **a szoftverfejlesztés agilis módja**, melyet bizonyos értékek, elvek és fejlesztési technikák írnak le.
  - Az XP **öt legfontosabb értéke** a szoftverfejlesztés irányítására:  
**kommunikáció, egyszerűség, visszajelzés, bátorság és tisztelet.**



# Extreme Programming

- További útmutatók, alapelvek:
  - emberség, gazdaságosság, kölcsönös előnyök, hasonlóság, fejlődés, sokféleség, reflektáció, flow, lehetőség, redundancia, minőség, kis lépések és vállalt felelősség.



# Extreme Programming (XP)

- XP **tizenhárom** elsődleges **gyakorlatot** ír le:
  - Üljünk együtt, teljes csapat, informatív munkakörnyezet, energikus munka, páros programozás, user story, heti ciklusok, negyedéves ciklusok, csúszás, tízperces build, continuous integration, „test first” programozás, és inkrementális design.

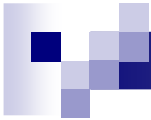




# Agilis fejlesztési technikák

## ■ Páros programozás

- Két programozó ugyanazon a munkaállomáson dolgozik (egy képernyő, egy billentyűzet, egy egér összesen a két embernek)



# Agilis fejlesztési technikák

## ■ Refaktorálás / refaktorizálás

- Egy meglévő forráskód belső struktúrájának fejlesztése / jobbá tétele úgy, hogy közben a program működése ne változzék.



# Agilis fejlesztési technikák

## ■ Test Driven Development (TDD)

- ☐ Tesztvezérelt fejlesztés
- ☐ Ciklikus
- ☐ Rövid ciklus idő ~ percek
- ☐ Automatizált tesztek

## ■ Általában XP-ben és agilis fejlesztésben használják

- Lásd még a 7. és 2. előadást
- ☐ A programozók előbb a tesztekét írják meg
- ☐ A tesztek kezdetben nem futnak le
- ☐ Rendre megírják a kódot a tesztekhez
- ☐ A tesztekét újabb elemekkel egészítik ki

# Agilis fejlesztő

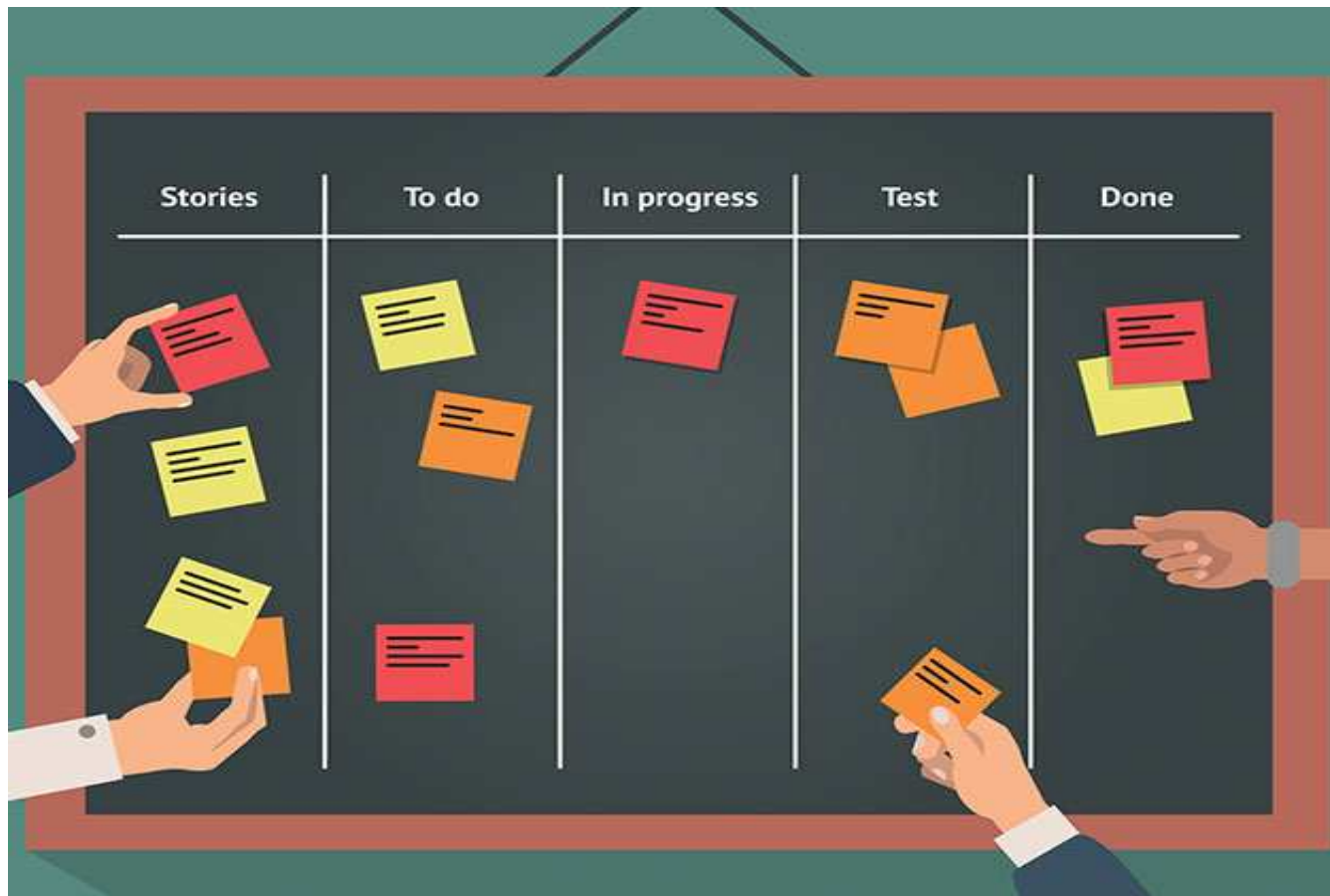
- Műszaki jellegű döntések
- Sok-sok tesztet ír
- Implementálja a user storykat
- Minőség centrikus gondolkodás
- Folytonos refaktorálás, fejlesztés
- A csapat egy tagja!

<http://www.developer.com/design/cartoon-of-the-week-are-you-an-agile-developer.html>



"Yes, you are a developer and yes, you're agile but that doesn't necessarily make you an agile developer."

# Agilis fejlesztő





# Miről volt szó...

- Tervezés / design
  - Definíciók
  - A folyamat
  - Szoftvertervezési alapelvek és fontos elemek
  - Architektúrák, döntések, tervezési minták
  - Felhasználói interfész (UI) tervezése
  - Szoftvertervezés a CMMI és az Automotive SPICE modellekben
- Implementáció / kódolás
  - Kódolási szabványok alkalmazása
  - A kód minőségének ellenőrzése
- Becslés a szoftvertervezés során
- Tervezés és implementáció agilis környezetben