

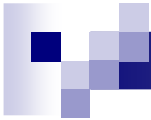


Szoftvertchnológia

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
 - 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



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

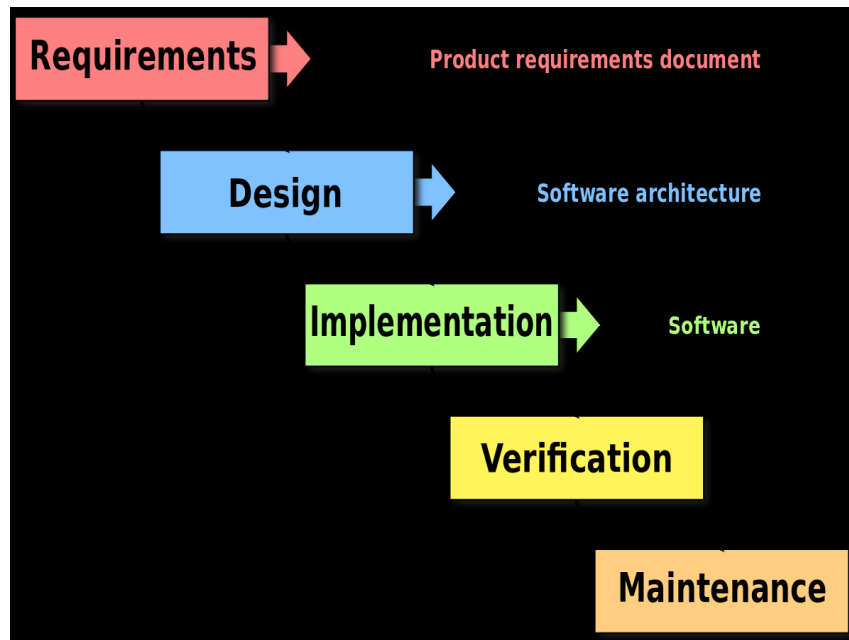
- A tervezés elemeinek más elnevezései:
 - Fogalmi terv / Conceptual / high level design
 - Részletes terv / Detailed design
- A tervezés során alapvetően 2 rendszer-nézetet írunk le:
 - Statikus nézet – dinamikus nézet



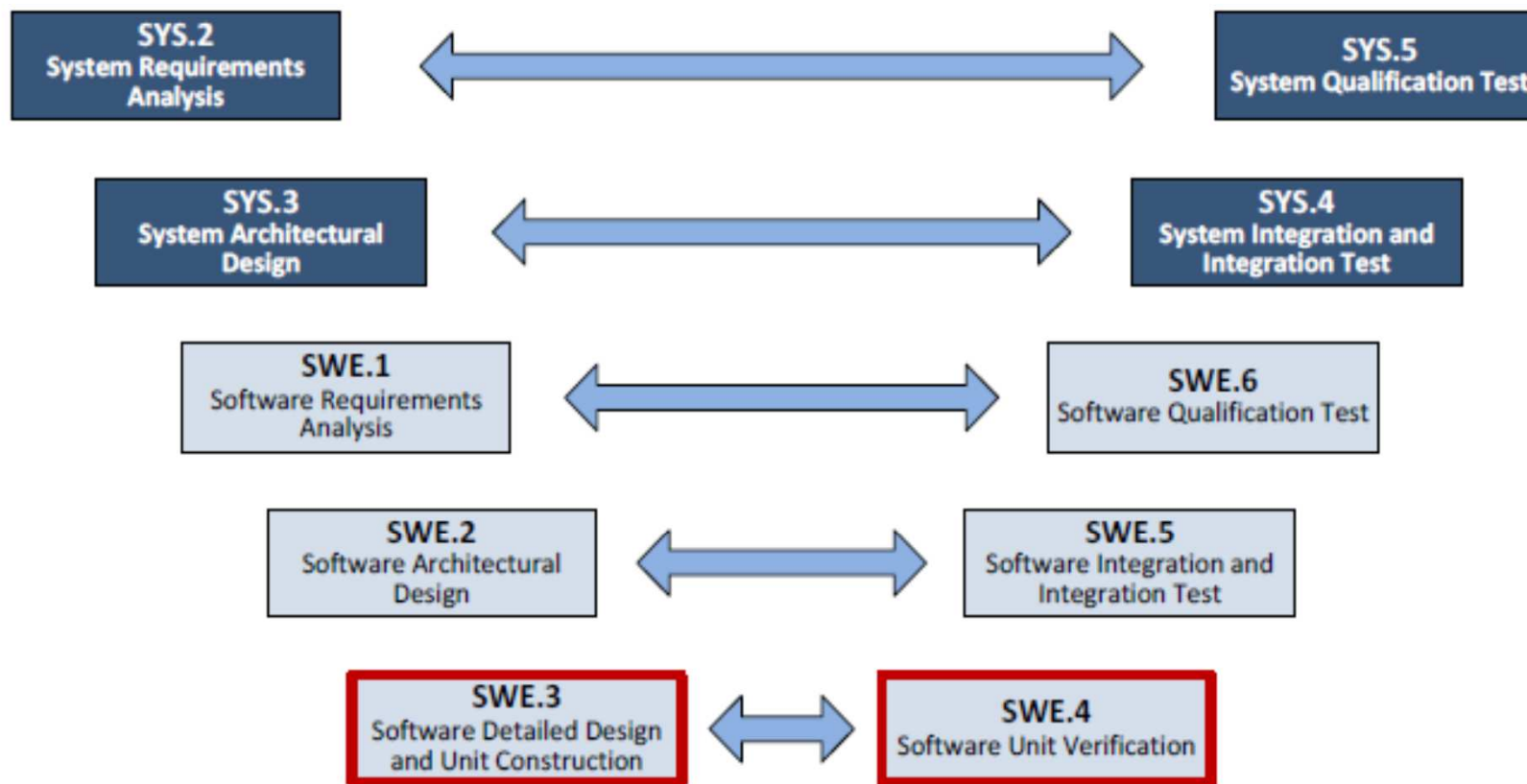
Design / tervezés

- 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!

Tervezés / Design a szoftverfejlesztési életciklusban



Tervezés / Design a rendszerfejlesztési életciklusban



From: Automotive SPICE , PAM, v3

2018.10.17.



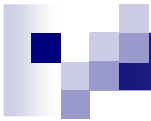
Szoftvertervezési alapelvek

- 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.



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!





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



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

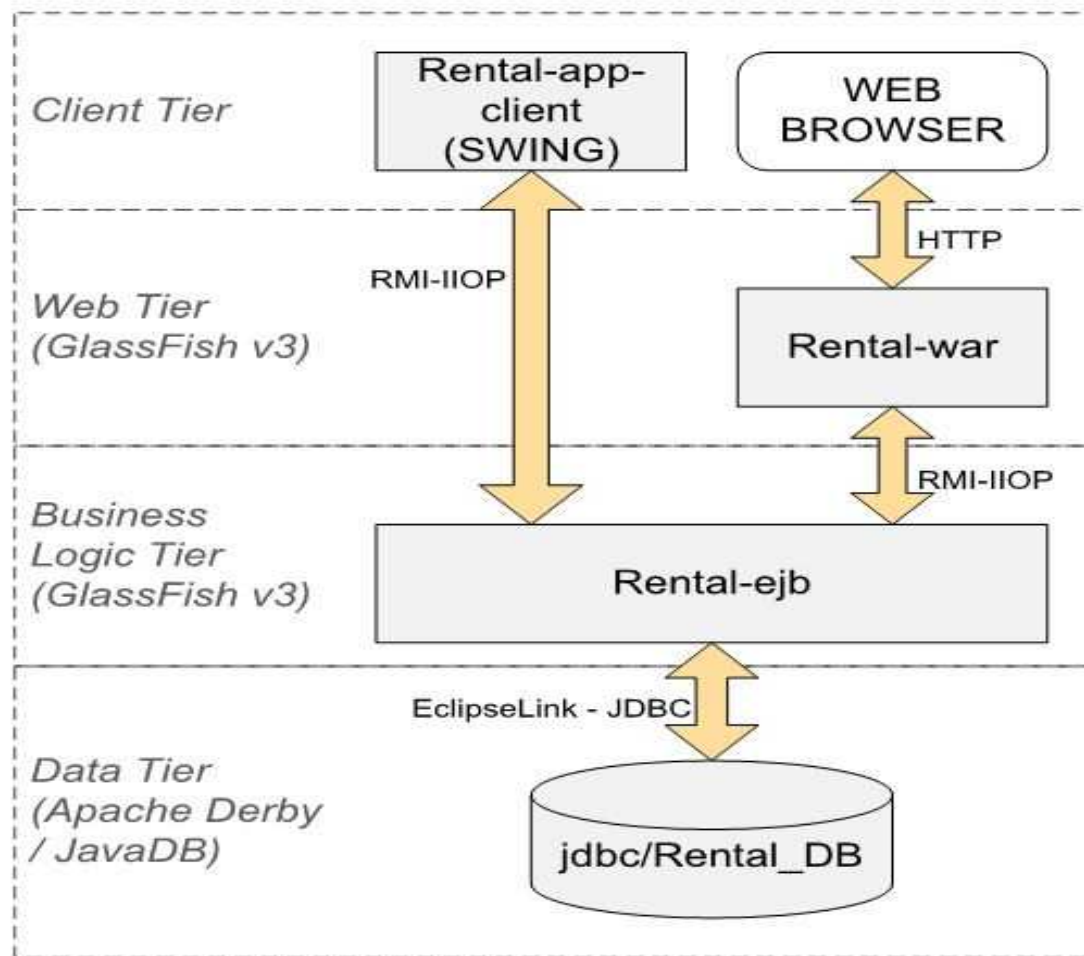
- ☐ Magas szintű terv / High level design
- ☐ Részletes terv / Low level design
- ☐ Modellek, leírások, döntések alapjának leírása
- ☐ **Verziókezelés a design dokumentumoknál is fontos, elengedhetetlen!**
 - Pl. tesztelés során architekturális hibát javítunk – de az architektúra leírását elfelejtjük 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.).
 - A CMMI –DEV v1.3 alapján

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 elemhalmaz, 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
 - ☐ Konfigurációmenedzsment
 - ☐ Változásmenedzsment
 - ☐ Verziókezelés
 - ☐ ...



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 újrafelhasználása



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 kiegészítő anyagban bemutatunk a tervezésben és kódolásban alkalmazható 2 becslési technikát :
 - PROBE módszer (a PSP-ben használatos)
 - Funkciópont-számolás



Funkciópont számolás

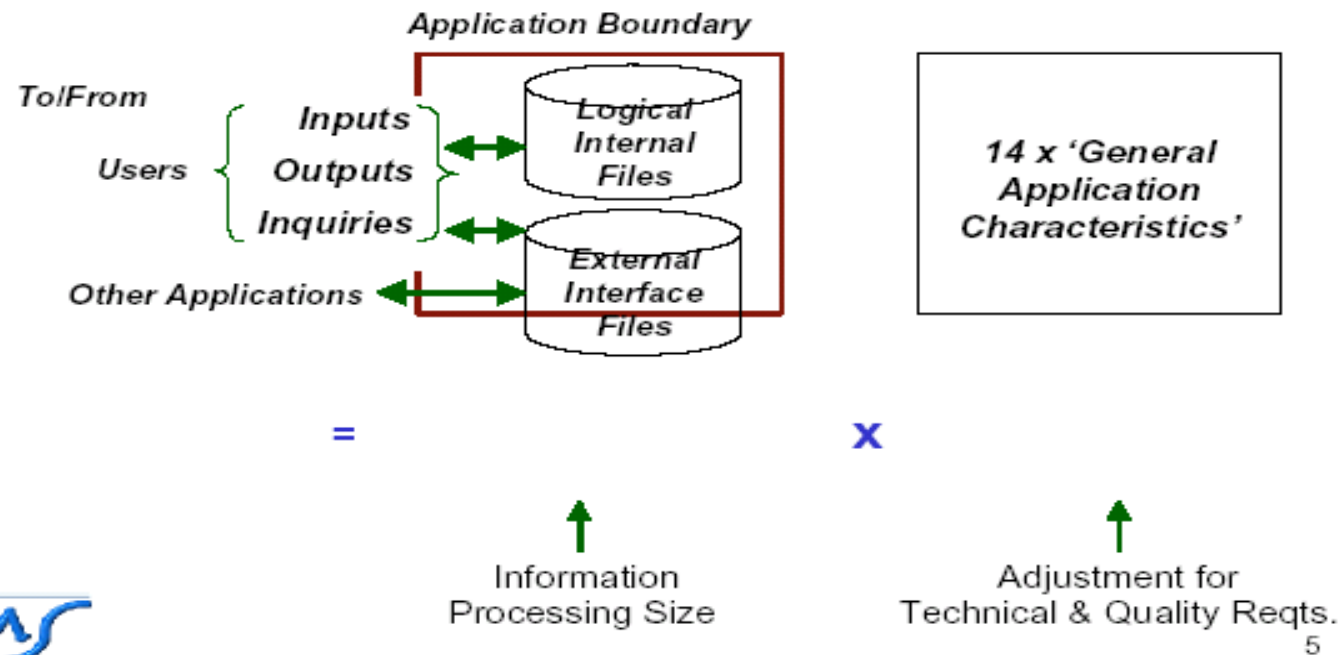
- Kifejlesztésének célja: különböző technológiákkal történő szoftverfejlesztések hatékonyságának összehasonlítása
- Albrecht céljai a funkciópont számolással:
 - a szoftver méretének következetes mértéke legyen
 - legyen független a fejlesztésben alkalmazott technológiától
 - alkalmazása legyen egyszerű, 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

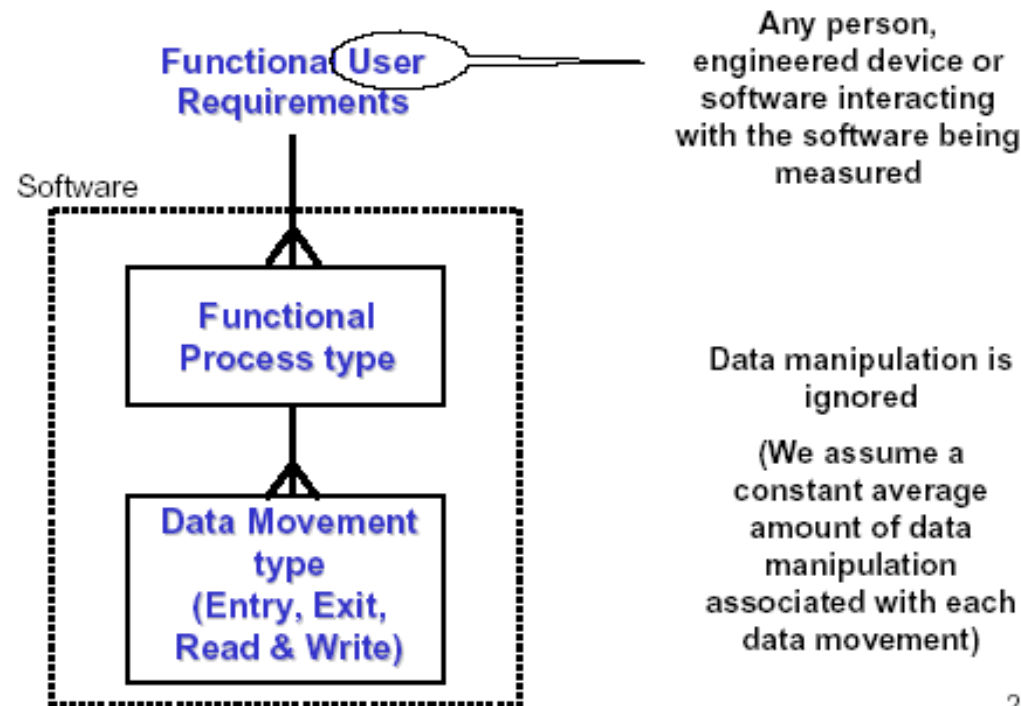
■ 1970-ből

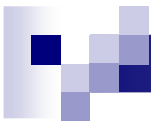


Bizonyos jellemzőket figyelünk, és 0-5 között súlyozzuk jelenlétüket, jelentőségüket.

A Cosmic módszer

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





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



Agilis design elemek

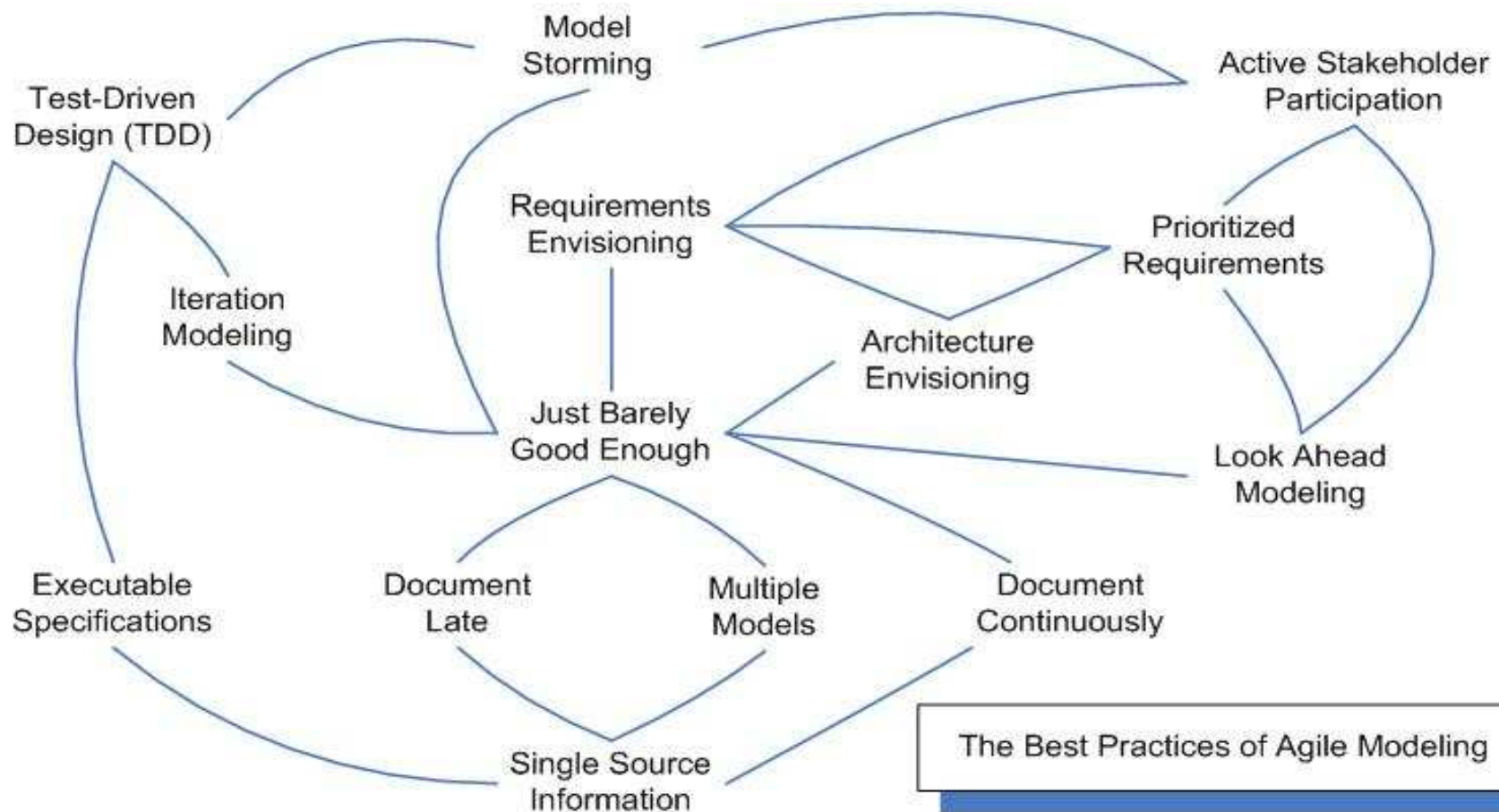
■ Magas szintről indulunk

- 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

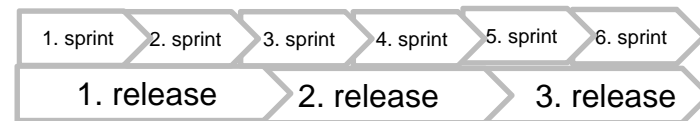
- Az agilis szoftvertervezés szerves része a User Story / felhasználói történet / story point meghatározása
- A követelmények meghatározásánál is láttuk ezeket
- Itt tovább részletezzük őket

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](#)

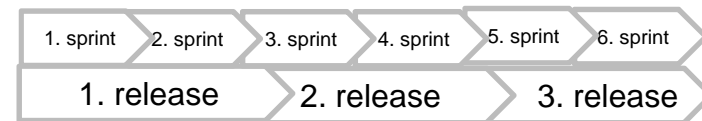
Agilis design

■ 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 !!!!

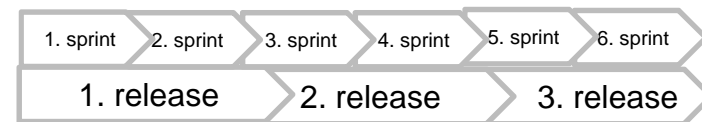


Agilis design

■ User Story meghatározása

□ A Sprint végén

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



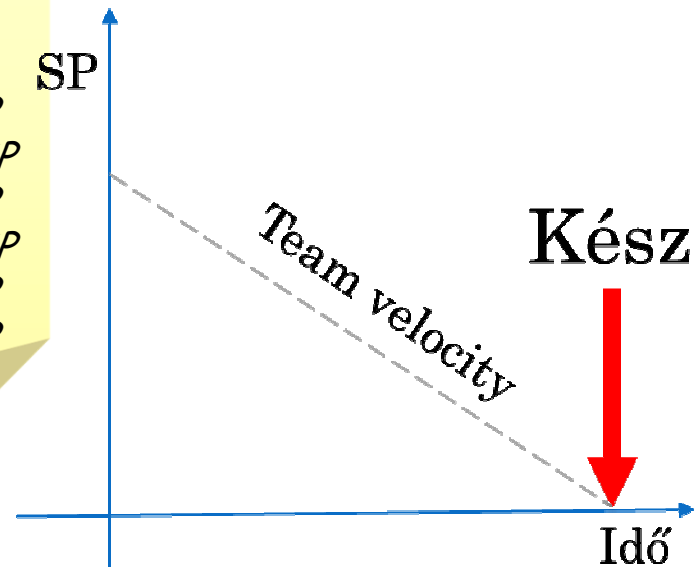
Agilis design

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.

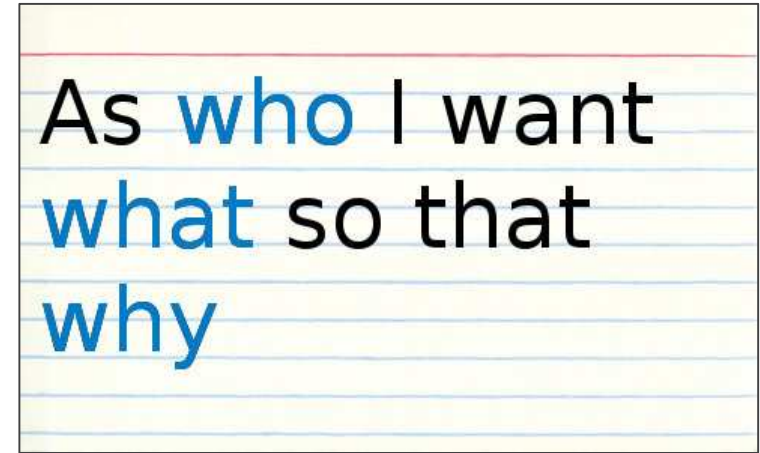
Agilis tervezés

■ 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

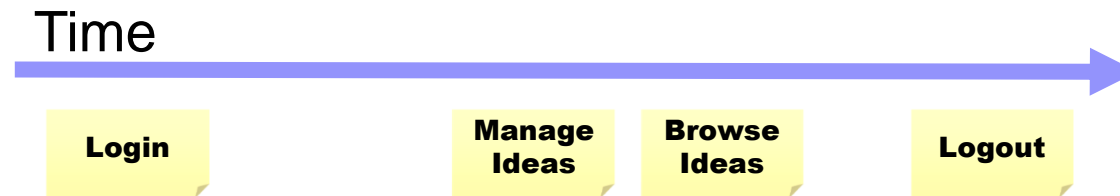
■ User Story Mapping

Time



Agilis design

■ User Story Mapping



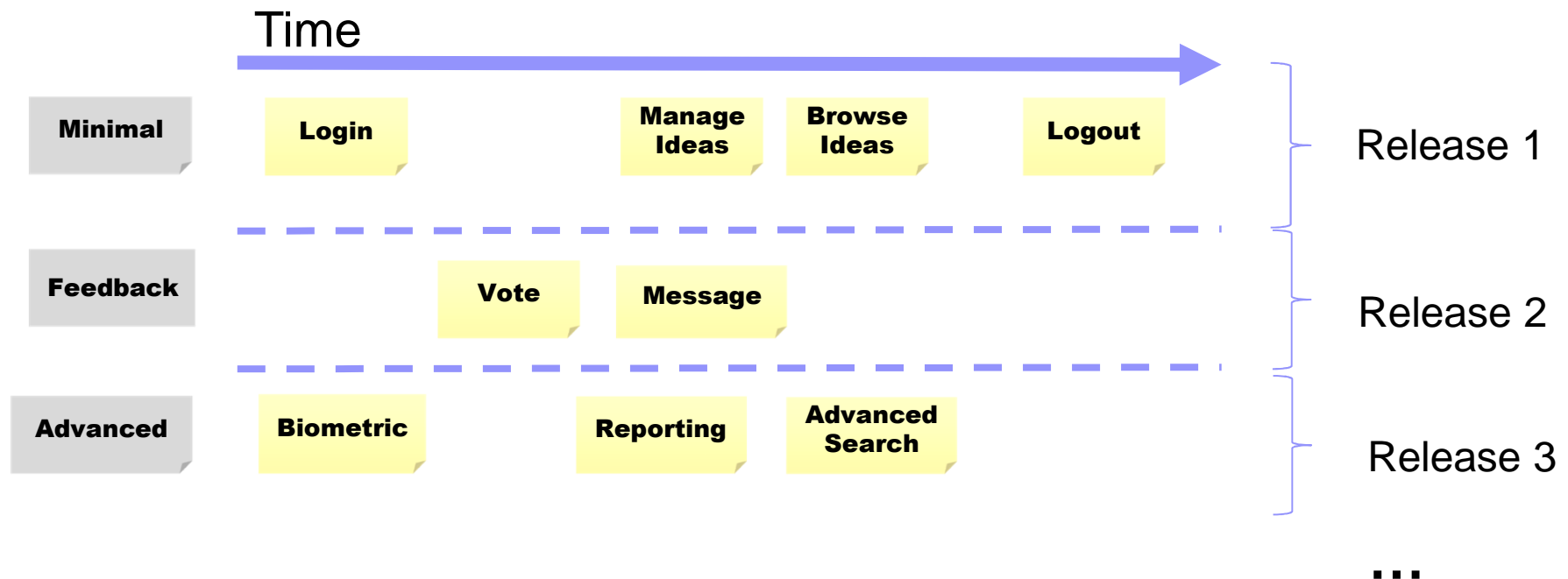
Agilis design

■ User Story Mapping



Agilis design

■ User Story Mapping



Agilis design

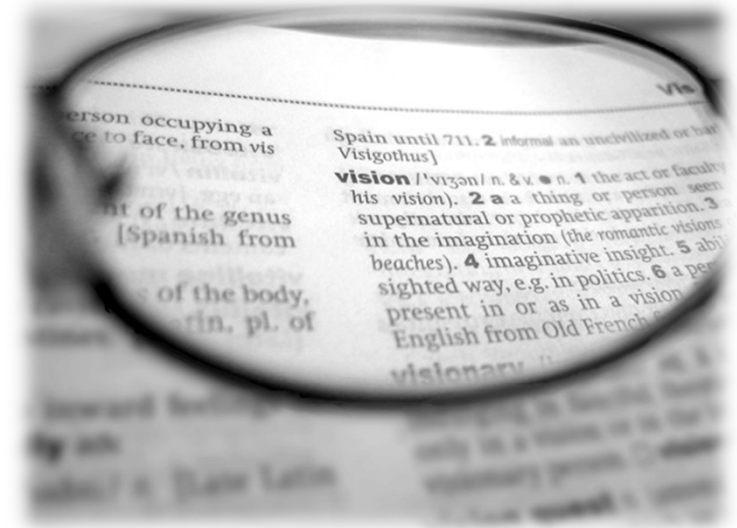
■ User Story Mapping

□ 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



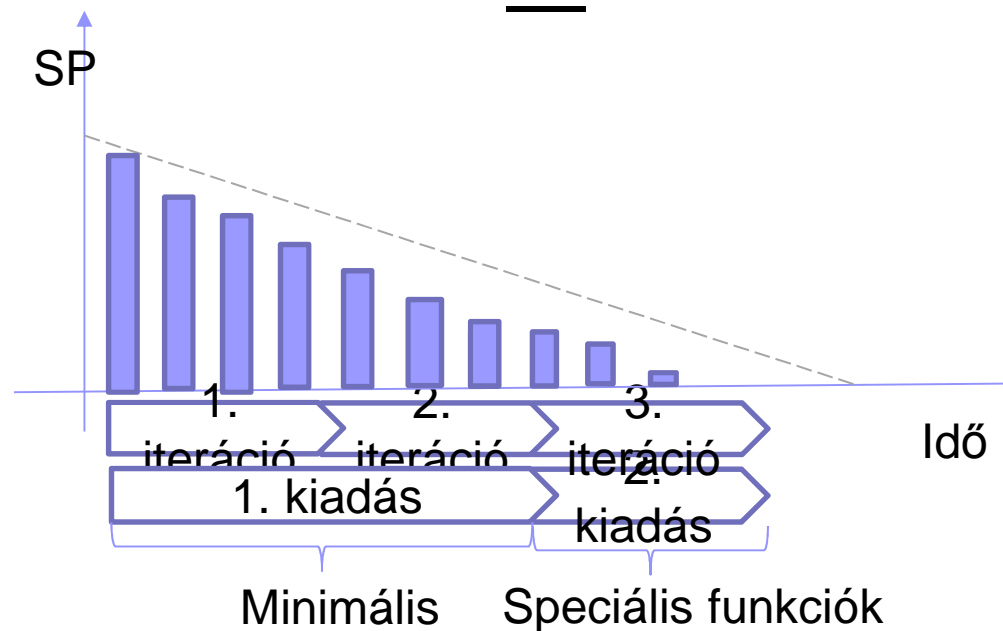
Agilis design

■ User Story Mapping



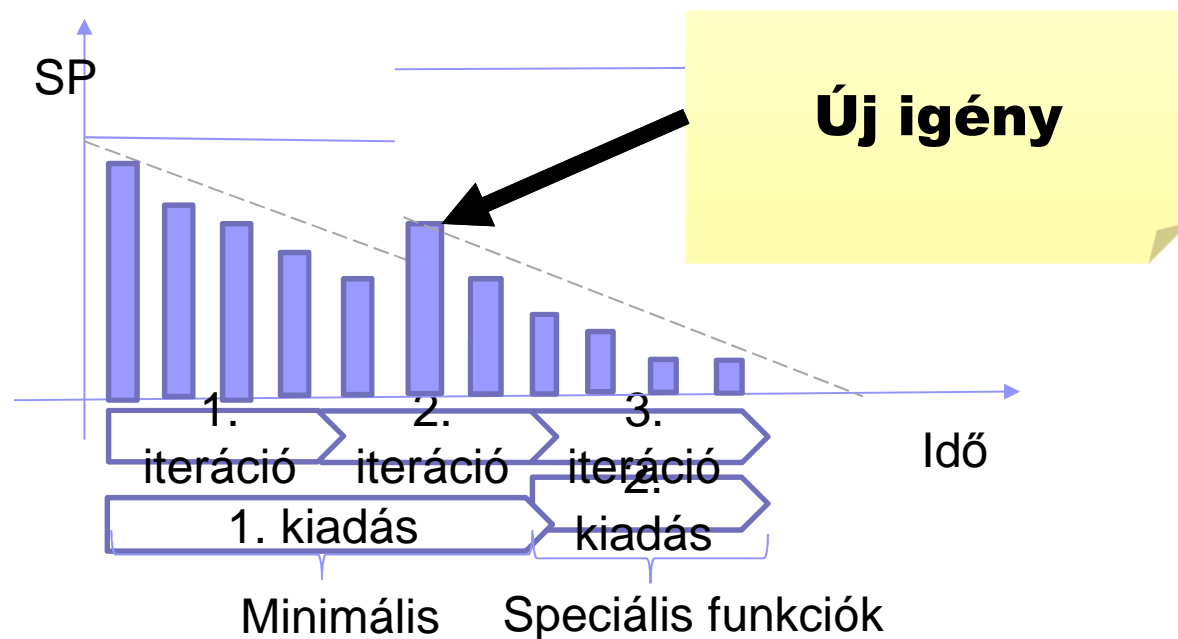
Agilis design

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



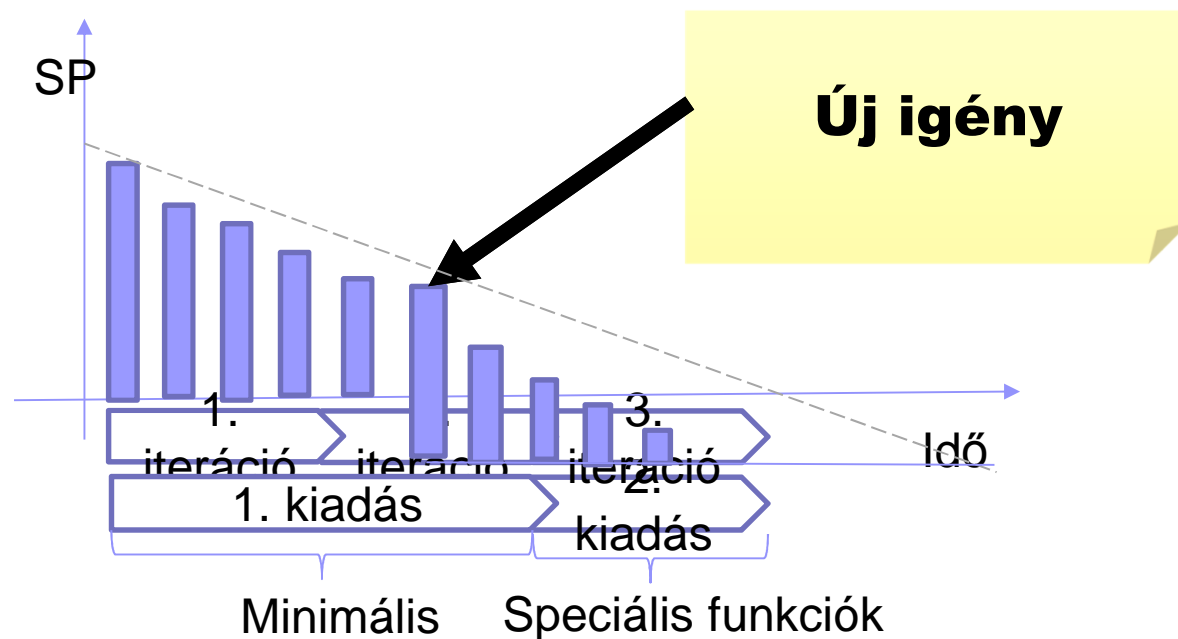
Agilis design

■ 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
 - **Definion Of Done**
 - Elemzett, tesztelt, implementált, stb...

PLAN FIRST!



Agilis design

Independent

Negotiable

Valuable

Estimable

Small (Sized appropriately)

Testable

(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:

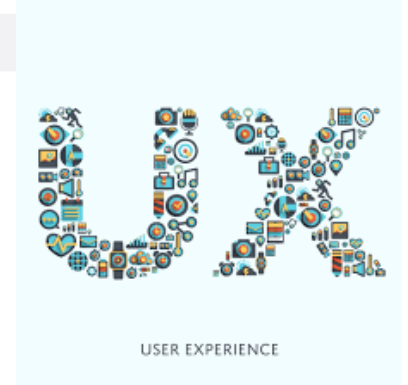
- Egyeztetett / mindenki által elfogadott listája a termék inkrementum elkészítéséhez elengedhetetlenül szükséges tevékenységeknek
- A csapat megegyezésre jut a DoD-t illetően, és ki is függesztik ezt a teremben egy jól látható helyre. Egy sor követelménynek teljesülni kell ahhoz, hogy a termék adott 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



- User Experience, User Experience, User Experience!
- **Felhasználó a középpontban:**
 - UX (User Experience) design – a korábbi UI design.
- Az UX elemei „agilisak”
- Pl.: Incrementális, scratch first...

Agilis szoftvertervező



Agilis „business analyst” (üzleti elemző)

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



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.



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

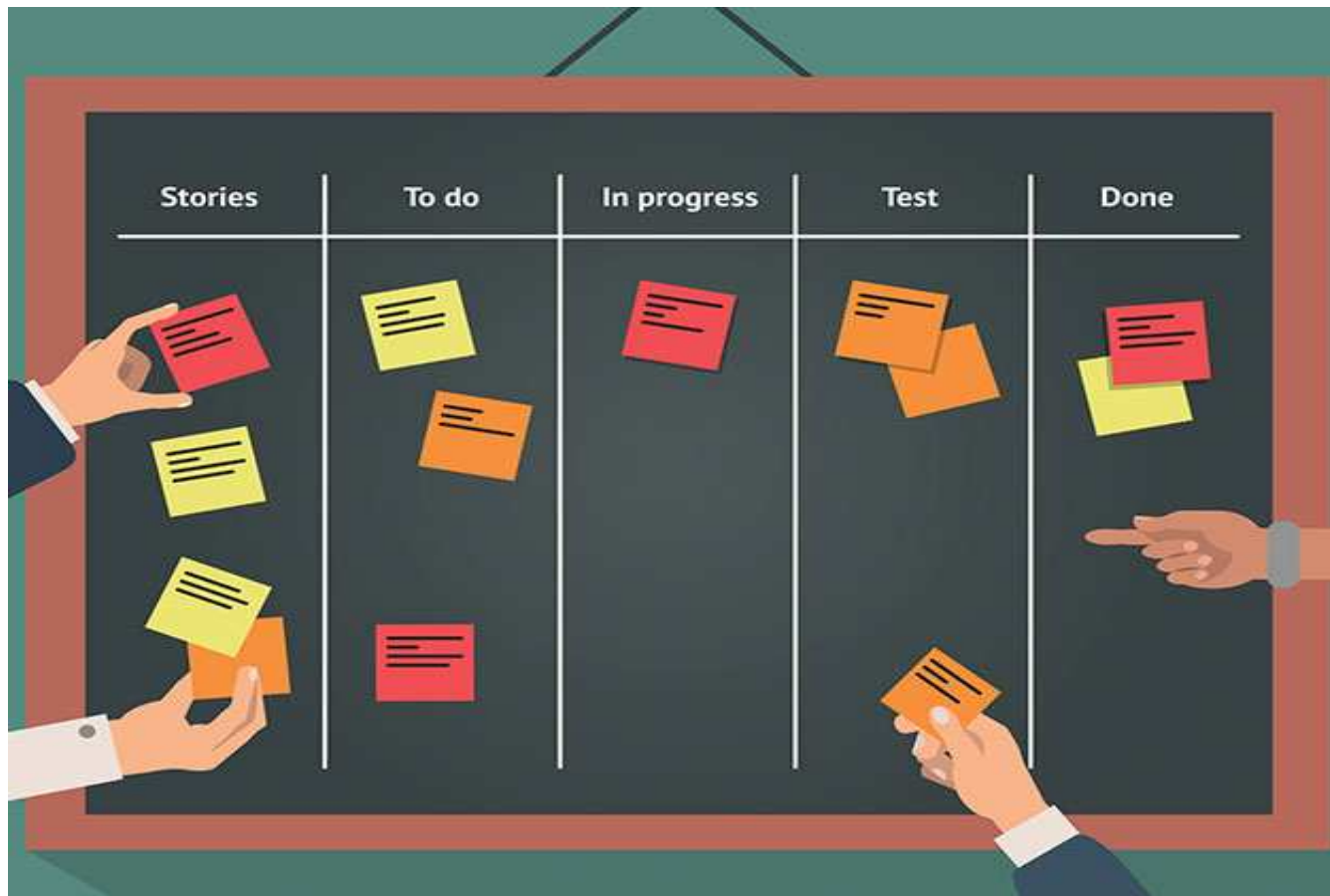
- Műszaki jellegű döntések
- Sok-sok tesztet ír
- Implementálja a user storykat
- Minőség centrikus gondolkodás
- Folytonos refaktorálás, fejlesztés
- A csapat egy tagja!

<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