Szoftvertechnológia videó-jegyzet időbélyegzőkkel

Az alábbi jegyzet a video.bme.hu-n is megtalálható, a 2010/2011-es tanév őszi félévében felvett szoftvertechnológia előadásokhoz készült. A jegyzetet magát Horváth Tamás (herrht@gmail.com) készítette, az időbélyegeket Horváth ;DÁvidÁM (kottalovag@gmail.com) tette hozzá.

Fontos, hogy ez alapvetően csak gyorsan össze lett firkálva, tehát nem túl rendezett, nem teljes, vannak elütések, stb., nem igazán volt idő ezekre is figyelni (talán érthető módon), de egy kis kiegészítésnek talán jó. Pl. betájolható, melyik rész melyik videóban, merre van, kereshetők a kulcsszavak.

Verzió: 2011. június 3. 22:15

Szofttech video jegyzet, 6-os bontás.

1. előadás

4: Kettős célkitűzés:
- átfogó kép a szofttechről: hogyan csináljunk szoftvert nagyban?
- OO modellek, fejlesztés, módszerek

10: Mi a SW?
- Két definíció! Inkább a második.
- Mérnöki munka
- Nem anyagi
- Komplex, bonyolult, fő kérdés: hogyan uraljuk ezt a bonyolultságot?
  Gyorsan változik, ez is növeli a bonyolultságot
- Nagy üzlet

11, 12:
Csekkoljuk: http://www.forrester.com/rb/Research/us_and_global_it_market_outlook_q2/q/id/56880/t/2
Min gondolkodik az IT Világ, mire mennyit költenek? Másik: Gartner Group

13,14,15:
SW Ipar hatalmas, nagy részük beágyazott rendszer, mindenhol megtalálható
Komoly munkalehetőségek, van pénz és állás is.

16: Problémák
- Igény meghaladja a gyártás képességeit -> gagyi szoftverek is kikerülnek.
- Minőség
- legacy: nagyon régi szoftverek (30éves), működnek, de érdemes lenne korszerűsíteni, ami irdatlan pénz, munka -> honnan kezdjük?

19: Minőség
- korrektség, dokumentációnak megfelelő
- megbízhatóság: működik akkor is, ha a körülmények nem pontosan megfelelők
- teljesítmény: memória/időbeli korlátok
- hordozhatóság: különböző környezetek

20: Minőség
Minimális hibaszázalék is hatalmas kihatással van.
Hibát kizárni nem lehet, hibák lesznek, a lényeg, hogy ezek egymásra hatását, egymás utáni sorozatát meg tudjuk állítani.

(therac tanulmány)

2. előadás

SW Oktatás Dilemmája:
SW Nagyban -> nagy szoftver kell -> szakismeret kell.
Ehelyett kis problémákra hozunk nagy megoldásokat.

1diasor/35: Karbantartás
- Régi programok, régi fordítók: hogy viselkednek ma?
- hordozók meddig lesznek üzemképesek?
- óriási karbantartási költségek
- **Készítsük úgy, hogy könnyű legyen karbantartani!**

36: célok
- Termelékenységet és minőséget javítani
- Fejlesztés és karbantartás költségeit csökkentsük
- precíz becslések
- automatizált gyártás

37,38:Mérnökség, engineering
- **Költséghatékony** megoldást kell készíteni (milyen tényezők, súlyozás)
- **gyakorlati** problémák megoldása (tömegszerűség, nagyban gyártás)
- **tudományt** alkalmazunk (megismételhető, ellenőrizhető, reprodukálható eszközök)
- **építünk**
- az emberiség szolgálatában (szabályokkal, szabványokkal, előírásokkal kell foglalkoznunk)

39-42: Fejlődés
- Craft, erőből, tapasztalatból megoldott – egyszerű gyártás, egy cél, mellékes egyéb dolgok (pl. piramis)
- Commercial: kereskedelmi forgalom, termék átadása használónak -> ismerni usert, adott eljárásokat (céhek), inasok, gazdaságossági megfontolások, kísérleti megoldások
  Előbb termelünk -> eladható?
- Tudomány
- Professzionális termelés, mérnökök osztálya, tervezett az előrehaladás, a célcsoport, minden lépés; piacszervezés
-
- Termelékenység és Erő találkozása: rómaiak útépítési közbeszerzést írtak ki, tender. Hogy megy oda az építő? Mit eszik? Hogy viszi az anyagot? -> Logisztika
- Tudomány:
  1750 Pápai tudósok, stabil-e a kupola szerkezete? -> Előbb volt kupola, utána mérés!
  1850 Hidat kell építeni -> Előbb mérések maketten, és csak utána építkezés!
- **Előre kell mérni, nem utólag!**
-
- ne írja meg mindenki ugyanazt: könyvtárakba szervezzük, ami már megvan

44,45: Történelem
- 60 évek vége: batch: indít, lefut, vége, van eredmény. Nincs elosztottság, home-made
- 60-70: multiuser, valós idejű (adott időkorlátok), adatbázisok, adatok-programok külön, sw termelés indítása
- 70-80: beágyazott rendszerek (pl. órák), korlátozott elosztottság (hálózatok), olcsó hardver, felhasználók igényei
- 80-90: pc, ExpertSystems (szakértői rendszerek: tudásbázisok létrehozása), ObjektumOrientáltság, neurális hálók, Web-computing, párhuzamosítás

- 90-: metaprogramozás (programkészítő program), Pervasive computnig (bullshit, mindenhova elér a programozás)

46: 60évek, tipikus problémák
- Módszer: olyan, amilyen, kis programok
- Adat, elkülönül a programtól?
- Vezérlés, melyek?
- Specifikáció, nincs (maga a program)
- Állapot, nem igazán ismert fogalom
- Menedzsment, nincs
- Eszközök: minimálisan (pl. debug: core dump -> memória kép)

47: 70évek, programozás kicsiben
- Szerkezetek ismereet
- Továbbra is batch jelleg
- Van állapot
- Menedzsment, hogyan fogjuk össze az embereket? (páran könyvtárakat írnak)
- Compilerek,

48: 80évek, programozás nagyban
- fő probléma az interfészek
- adatbázisok megjelenése
- folytonos végrehajtás (leállításig fut)
- komplett specifikáció
- hatalmas állapottér
- csoportmunka (sok száz programozó munkáját összehozni)
- fejlesztői környezetek

49: 90évek, extra nagy
- nagy elosztott rendszerek
- multimédia adatbázisok (mindent tárolhatunk)
- vezérlés elosztott (mi oszthatatlan, mi eosztott?)
- biztonságilak kritikus rendszerek (repülők, erőművek)
- extrém méretű állapottér
- menedzsment minőség javítására megy rá
- hatalmas keretrendszerek

3. előadás

1.diasor/
51: SW Krízis
- Ami elkészül, csak határidő után
- Költségek átlépik a határokat
- Nem megbízható
- Nincs doksi

Okok:
- Lassú hw
- gyorsan növekvő elvárások
- karbantartási problémák

Megoldás:
- Szabványosítás

52: SzoftverTechnológia
- Tudományos ismeretek gyakorlati alkalmazása a tervezésnél és a program készítésénél, valamint az ezekkel kapcsolatos dokumentációk fejlesztőknek, illetve a karbantartás és üzemeltetés.
- Technológiai és menedzseri fegyelem, a szoftver ide tartozó gyártási és karbantartási fázisaiban, melyeket idővel fejlesztik és változtatják a becsült költségeken belül. wtf?

53 – 54: SzofTech témák

2.diasor
3: Minőségképek
- Transzcendentális: nem mérhető a minőség, de érezhető
- User: Jó, ha a felhasználó elégedett -> amíg nincs kész, nem mérhető
- Gyártás: ha betartjuk a gyártási eljárást, akkor jó
- Termék: minőségellenőrző berendezések, stressz teszt -> nem szoftverre
- Érték-Arányosság: drágább: jobb minőségű (?)

4:
- Process: emberi tevékenységek (és módszerek, eszközök, berendezések) azon célból, hogy az alapanyagot számunkra hasznos termékké transzformáljuk át.
    - Mi az alapanyag?
- Process Menedzsment:
    - Minőséget meghatározza a gyártási folyamat

5, Szoftver folyamat minőségének javítása
- minőség és hatékonyság nő, költségek csökkennek, folyamat flexibilitás nő -> használható másik termékhez; elégedett stáb, fejlesztők

6, SW Probléma

- Absztrakció: Információ rejtés mértéke
  Magasabb absztrakció, ha sok a rejtett információ.
  Alacsony absztrakciós szint, kevés rejtett információ (konkrétabb)
- Formalizáltság: szabályok betartása. Ha elegendően formális, akkor transzformációval előállítható egy másik leírás. Csúcsa a matematika.

7, Fejlesztéstér

- Indulásnál informális, nagyon absztrakt (csak elképzelés, körülírás, mi a probléma?)
- Vége egy konkrét, formális út.
- De mi az útvonal?
- Vízszintesen (egyre formálisabban), majd alacsony absztrakció (konkretizálás)
- Formalizálunk: egyre szigorúbb szabályokkal írjuk le (specifkálunk)
- Abszt.Csökkentése: hiányzó részletek hozzáadása
  - Elméletben, de a valóságban máshogy működik

8, Fejlesztési modell

- csak egy darabig tudjuk növelni a formalizáltságot, utána kénytelenek vagyunk belemenni a részletekbe, majd megint specifikálunk, tervezünk, speci, terv…
- csökkentett absztrakciós szinten csökken a formalizáltság is (több információ, így kevésbé pontos speci)
- általában 2 lépésben: elsőben egy PlatformIndependentModel, majd absztrakció csökkentésével platformspecifikus dolgok hozzáadása.

9, Ward-Mellor

- Absztrakció (függőleges) és implementációfüggőség (vízszintes)
- Pénz, Adó (egyéb pénzügyi fogalmak)
  -> leképezés ->
  Fájl, adatbázis (implementációfüggő fogalmak)
- Spirál vonalon, magas absztrakciós szintről indulunk, és hozzárendelünk implementálás függő dolgokat

10,11, ICOM modell: InputControlOutputMechanism

- Folyamat, a bemenetből állít elő kimenetet, megkötések és erőforrások hozzáadásával
- Input: követelmények
- Output: Kód és dokumentációs
- Kontrol és korlátozás: pénzügyi korlátok, ütemezés, szabványok (betartása/megvált)
- Erőforrások, mechanizmusok: stáb, eszközök

12, ProcessSpeci

- **Mi a cél? Mi-miért van?**
- **Milyen szerepek, kik a szereplők, milyen felelősségük van?**
- **Elkezdés kritériumai: nem feltétlenül kell az alapanyag.**
- **Bemenetek, alapanyagok, folyamat aktivitásai (mit csinálnak? azt ki csinálja?)**
- **Kimenet, mi lesz a vége?**
- **Kimeneti kritériumok: mikor hagyjuk abba (kimenet után mit kell még)?**
- **Metrika (hogyan ment végbe, az jó volt-e, mit mérünk, milyen mértékek?)**

13, SW Process

- extrém (bonyolult)
- embereket kell mozgatni, számonkérni
- staff, doksi, kód
- oszd meg és uralkodj (bonyolult problémát kicsikre bontjuk)

pl

- Építkezés (1:05p körül):
    - Mit is akarunk építeni? Saját életterünk -> milyen az életünk? Milyen életforma? Mik az elvárások?
    - Az építész tervez egy elvárásoknak megfelelő rajzot -> Formális modell
    - Önkormányzatok, rajzok elfogadtatása -> Standard, szabványok
    - Kiviteli tervek, kotta: mikor, mit, kinek, mivel stb. -> pénzügyi tervek, logisztika, eszközök, lépések, méretek, ütemterv
    - Implementáció: a terv végrehajtása (esetleg változtatás)
    - Kész a ház -> hatóságok ellenőrizzék, megfelel-e az elfogadott terveknek -> használatba vételi engedély
    - Használat, karbantartás

4. előadás

14, SW élete
- Készítési folyamatról beszélünk, ez meghatározza a minőségét
- Milyen a SW életciklusa?
- Követelmények tisztázása, mi a kihívás?
- Specifikáció, adjunk a követelményekre egy megoldást, formális leírás
- Design, tervezés: hogyan lesz a semmiből a specifikált dolog? Pénz, anyag, stb
- Kotta, tevékenységlista -> végrehajtás, implementálás
- Validáció, érvényesség ellenőrzése, megfelel-e a követelményeknek összességében?
- Karbantartás

15, Életciklusmodellek, lineáris szekvenciális
- előző lépések lépcsős szerkezete

16,
- Nagy hibák már a kódírás előtt
- Felfedezés-javítás között hosszabb idő, költségesebb javítás

17, Vízesés modell:
- Vannak visszacsatolások, visszalépések
- Következő lépés előtt megnézzük, hogy a részeredményeink megfelelők-e, végrehajtható-e a továbblépés? Ha nem, visszalépünk -> Több kis korrekció

18, Deliverables
- Mikor érünk a speci végére?
- Követelményfázis vége, ha elkészült a **rendszerdefiníció és a projekt terv** -> de mik ezek? Szabványok írják elő, van tartalomjegyzék. PFR: ellenőrzés
- Specifikáció: formális leírás
    - RequirementSpecification: Követelmények technikai része
    - Preliminary user's manual: felhasználói dokumentumok, felületek!
    - Verifikációs terv (előzetes változata): hogyan fogunk ellenőrizni? Mi a feltétele, hogy átadjuk? Hogyan ellenőrizzük a terméket? SSR
- Architektúrális tervezés: milyen elemekből épül? Azok hogyan kapcsolódnak? Nagy komponensek leírása, PDR
- Részletes tervezés: megmondjuk a programozónak, mit csináljon -> munka lebontása elemi lépésekre, melyek kiadhatók, ellenőrizhetők: WBS.
  Felhasználói kézikönyv és verifikációs terv véglegesítése: CDR
- Implementálás: folyamatos SourceCodeReview (kód felülvizsgálás, SCR), AcceptanceTestPlan, vagyis az Átadási teszt tervei, ATR
- Validáció: minden doksi felülírt változatát végigvezetni, érintett részeket is változtatni
  PPR: Product Release Review (kibocsátás előtti végső áttekintés)
  PPM: Project Post Morten (?) (halál után)

19, V model
- Sok programozó dolgozik, sok kis program -> hogyan rakjuk össze? Hogyan ellenőrizzük?
- Unit teszt: önálló kis részlet tesztelése ←→ Részletes tervezés (önálló kis részekre bontás)
- Kis részleteket szépen finoman összerakni: integráció ←→ Architektúrális tervezés (fő komponensek, és azok együttműködése)
- System Teszt: az egész rendszer tesztelése (szembeállítása a specifikációval)
  ←→ Validáció ←→
  Követelmények, Specik

20, Problémák az életciklus-modellekkel

- Ritkán követi a valóságot
- Nem tudjuk eleinte összefoglalni a követelményeket (még ez meg az kéne)
- Ügyfél türelmetlen, kell pár gyors és randa megoldás, amit meg lehet mutatni
- Fejlesztők csúsznak

00:38:55

21, Prototípus + iteratív inkrementális

- Készítünk egy prototípust, majd a tapasztalatok alapján a követelményeket felülvizsgáljuk, kiegészítjük (inkrementáció)

00:41:10

22, Gyors Alkalmazásfejlesztés (RAD: Rapid Application Development)

- Iterációs módszer alváltozata lehet
- Eszközök kész megoldások létrehozására
- Komponálható, összedobálható rendszerek

00:42:21

23, Spiral model

- Bal felső sarok (Planning), belső pontjából indul (initial requirements: követelmények meghatározása)
- Kockázatelemzés (adott követelmények kielégítése mivel jár: mennyi pénzt veszíthetünk, nyerhetünk -> Go/NoGo?)
- Engineering, prototípus összerakása
- Prot. kiértékelése (akár a userrel), mit érdemes változtatni?
- Új követelmények, új kockázatelemzés, megint döntés, újabb proto, kiértékelés -> nogo, kész

00:45:58

00:48:40

00:51:24

24,25,26, CMM

- Cégek (szoftverfolyamatainak) összehasonlítása, minőségellenőrzése
- Mérések, minősítés -> rendben, de hogyan legyen jobb?
- Fokozatok, elvárások, minősítés menetrendje

- 5 szint van: 1 kezdetleges, 5 legjobb
- 1: gyakorlatilag bárki
- 2: van valamiféle ismétlődő technika, fejlesztési folyamat, nagyvonalakban definiált dokumentációs minták. Nem szabványosak
- 3: definiált: van egy meghatározott fejlesztési folyamat, szabványok szerinti folyamatok
- 4: mérés van: folyamat állapota jósolható a mérési eredményekből
- 5: optimalizált: magát a folyamatot tudjuk precízen hangolni, a termékhez igazítani.

00:55:30
00:56:00
00:56:25
00:56:44
27,31,

- 1 szint: mint egy zsák: valami bemegy, majd kijön: nem látunk bele, nem látjuk a hibákat, mikor jön ki eredmény, használható-e egyáltalán
- 2 szint: van ismétlődés, vannak töréspontok, amiket lehet vizsgálni, de nem szabványos
- 3 szint: szabványos, vagyis látjuk a részelemeket is, struktúrák ismertek
- 4 szint: méréseket végzünk, ki-mit dolgozott, milyen doksi készült
- 5 szint: optimalizálás, két elem helyett egy, vagyis magát a folyamatot tudjuk javítani

3. diasor/

01:05:36
01:08:12
3, 4.dia: **Menedzsment**

01:09:54
5, Csoportszervezés

- kevesebb, de jobb ember
- hozzá illeszkedő feladatot kapjanak
- csoportok tagjai között harmónia
- aki nem fér csapatba, azt el kell távolítani
- **Brooks törvénye**: késésben lévő projekthez újembert adunk, csak tovább késik (bele kell tanulniuk, a többieket ez hátráltatja)
  MagicSeven: kb ~7vel érdemes csapatot alkotni

01:18:57
6, Tervezni kell a pojektet

- kompromisszumok a lehetőségek között (emberek, pénz)

01:19:51
7, proj. menedzser:

- erőforrások újraosztása, ütemterv/ütemezés hangolása, követelmények lazítása, tréningre küldeni

01:20:22
8, Project terv

- LongTerm, hosszú táv:
- ShortTerm, rövidtáv, akár napi szintre (melyik nap mit kell csinálni)

5. előadás

Menedzser

EA4/01:21:11 == EA5/00:06:00

9, Erőforrások

- Emberi, emberek együttműködtetése, költséges (fizetés)
- Hardver, fejlesztő-célgép (cél rendszer) elválik, speciális hardver (mosógép-pc között pl.)
- Szoftver, SW-t megvenni, menedzsment-támogató sw

00:10:54

10,

- idő: mindig legyen munka, dolgozzanak
- információ: csak a lényegi információt átadni (effektivitás +titkosítás, hozzáférés)
- szervezet fenntartása
- minőség
- pénz

00:14:06
00:18:00
00:19:20
00:19:44
00:20:44

11,12,13,14,15 Ütemezés

- Munkát időben szétosztani -> elemi lépésekre bontás (WBS: Work Brakedown Structure)
- Melyek hajthatók végre konkurrens módon?
- Taszk hálózat kialakulása
- SW követelmények -> Elemi lépések (BWS) -> Függőségek (pl. 14.o) -> Erőforrások (idő eszköz) becslése -> erőf. feladathoz rendelése -> Gant-diagram (vagy Pert-diagram)

00:24:26
00:25:44

16,17, Kockázatok

- Nem kívánatos esemény bekövetkezése
- Termék, pénzügyi, stáb, hardver
- Mese (28-30p körül)
- Követelmények változtatása
- Technológiaváltás (újabb -> jobb?)

00:34:08

18, Kockázat-menedzsment folyamatai

- Azonosítás (előfordulható kockázatok listára: tapasztalat, szakirodalom alapján)
- Elemzés (minden kockázathoz 2 paraméter)
- Tervezés (hogyan kerüljük el, vagy minimalizáljuk, ha bekövetkezik)
- Monitorozás

00:36:14

19, Azonosítás

- Technológiai, emberi, szervezeti, becslések
- egyéb tapsztalatok, irodalom

00:36:40

20, Elemzés, **paraméterek**:

- Mennyire valószínű a bekövetkezés (1-5: alacsony, nagy, közepes, stb)
- Mennyire komoly a kockázat (4-5: komoly, elfogadható, semleges, katasztrofális)

00:38:12

21, Tervezés, **stratégia, 3 féle**

- Milyen stratégiával fordulunk a kockázat felé (hozzárendeljük)
- 1, Elkerülés: Mi kell, hogy ne következzen be?
- 2, Minimalizálás: Ha biztos bekövetkezik, hogyan csökkenthetem?
- 3, Folytatás: Hogyan tudunk továbbmenni, kimászni a problémából?

00:40:11

22, Monitorozás

- Nem elég egy tervet készíteni: adott gyakorisággal áttekinteni, fejleszteni, felülvizsgálni a kockázatelemzést

00:41:58

23, Becslés

- Jövőre vonatkozik, mi alapján?
- Kis projekt – jó becslés
- Nagy projekt – csökkenő pontosság
- Az igény fordított! Alapellentmondás

00:43:54
00:47:08

24,25 Becslés

- Indokolhatónak kell lennie
- Projekt végén Tény van, adatok -> legyen X.
- Becslés hogy viszonyul X-hez? Negyede és Négyszerese közé lőtt becslés jó
- Becslés egyre pontosabb, ahogy több adat lesz ismert

00:48:27
00:57:17
00:58:35
00:59:45
01:00:50
01:03:03

26-31, Mérések

- Fejlesztéshez mekkora **erőfeszítés** szükséges? Hány ember, mibe kerül? Milyen nagy/bonyolult?
- SW **méret**: Hogyan mérjük a bonyolultságot? LineOfCode és FunkcióPontElemzés (kódsorok száma és Adott funkciók meghatározása)
- **Komplexitás** hogyan mérhető?
- Mese: 50.p körül (pályázat, balesetek)

01:04:20
00:05:10

32,33 Project Plan

- Bevezetés, becslések, kockázatok, ütemezés, stáb, követés, ellenőrző mechanizmusok

01:06:00

34, Bevezetés

- **Hatáskör**: mi tartozik a projektbe, hol vannak a határok?
- SW funciók
- Teljesítmény és viselkedés
- Menedzsment és technikai korlátozások (erőforrások)

01:07:16

35, Becslések

- Történelmi adtok (becslésünk alapjai)
- Becslési technikák leírása
    - Technikák táblázatba vétele
- Becslési módszerek összevetése, kiegyensúlyozása
- Becslések leképzése emberekre, gépekre, eszközökre, konkrétumokra

01:08:50

36, Kockázatok

- Kockázat leírása, Táblázatba foglalása (valszín., hatás), Kezelés és monitorozás

01:09:30

37, Ütemezés

- WBS, lépésekre bontás
- Funkcionális bontás
- Hálózat létrehozása
- Idő diagram, pl. Gant-diagram)

00:09:55

38, Stáb

- Milyen csapatok
- Milyen menedzsment jelentési útvonalak, ki-kinek jelent, milyen riportok, infok és doksik áramlása, továbbítása, kik tárgyalhatnak

01:10:10

39, Követési és Irányítási mechanizmusok

- Milyen szervezeti háttere van a minőségbiztosításnak (SQA: Sw Quality Assurance)
- Változtatás menedzsment (hogyan mennek végbe változtatások, hogy vannak nyilvántartva)

6. előadás, 4.diasor

00:14:33
00:15:11
00:15:18
00:16:34
2-5, Követelmények
- Mik az igények, milyen korlátozások vannak?
- Minél később vesszük észre a hibát, annál többe kerül.

00:19:36
6, Követelmények
- Definíció: Kevésbé formális
- Specifikáció: Komoly formalizáltság
-
- Követelmények meghatározzák, hogy
    - Hogyan viselkedik, mit csinál a rendszer (funkcionális)
    - A viselkedés attribútumai

00:20:56
7, Nem funkcionális:
- Termék használhatósága (adott környezetben használható-e),
- Megbízhatósága (mikor, mennyit kell mennie)
- Hatékonyság, helykihasználás, hordozhatóság
-
- Szervezeti (terjesztés, szabványok, implementáció)
-
- Külső (személyiségi jogok, adatok hozzáférése, etikai, erkölcsi kérdések)

00:33:03
00:37:41
00:38:40
00:39:48
8-11, Követelmény Definíció
- Természetes nyelven, mindenki által olvasható
- Szerkezetben felépítve
- Legyenek ellenőrizhetők („Legyen gyors” ←→ „Legyen kész 5sec alatt”)
-
- Korrekt: tényleg teljesíteni kell
- Egyértelmű: egy értelmezési lehetőség
- Teljes
- Verifikálható, igazolható (módszer az ellenőrzésére)
- Megérthető
- Módosítható (akár menet közben is),
- Követhető (változások, mit-miért csináltunk, melyik követelménynek teszünk eleget), a követelményhez tartozó kódot meg tudjuk találni (főleg változtatások esetén)

00:52:04
12,
- Információ tartalma: adatok mit jelentenek?
- Info hogyan mozog, áramlik?
- Mi az Info struktúrája?
- Mese 55p

00:53:00
01:02:47
01:14:23
13-15, Info szerzés

- Interjú az érdekeltekkel
- Kérdőívek, űrlapok kiadása adott személyeknek
- Csoportokkal találkozás
- Érintettek közreműködése fontos
- 1:55p Mese

01:14:47
16, Rendszer Definíció

- **Bevezetés**
- **Funkciók**
  - **Rendszer architekrúra, mik a nagyobb/alacsonyabb rendű funkciók, rendszer határai**
  - **Adatok felírása, hogyan függenek össze**
- **Interfészek**
  - **User interfészek, kapcsolat külső rendszerekkel,**
- **Alrendszerek leírása**
  - **Hatáskör, folyamatok, milyen adatok, teljesítmény**
- **Rendszermodellek, szimuláció, prototípus**
- **Projektre vonatkozó hatásai**
  - **Költségek, ütemezés**
- **Függelékek**
  - **Termék stratégia**
  - **Üzleti folyamat, szabályzat alakítása**

7. előadás 5.Diasor

00:01:00

3, Specifikáció: formális leírása egy sw-nek, ami kielégíti a követelményeket

- Mese 1:18
- Tevékenységek leírása, ÉS azok sorrendisége: kik, milyen viszonyok, mit csinálnak, milyen sorrendben

00:01:35

4,

- Funkcionális kép: DFD, DataFlowDiagram + Folyamat specifikáció
- Szerkezeti: ERD, EntitásRelációsDiagram és adatspecifikációk: kik a szereplők, köztük mik a viszonyok
- Időzítés: STD, StateTrans..Diagram, Állapotgép, időben hogyan hajtjuk végre ezen funkciókat

00:04:09

00:04:49; 00:07:26

5,6, Funkcionális kép, DFD

- Inputból Outputot transzformálunk -> bontsuk részfolyamatokra
-
- Adatfolyamok: nyilakkal (Data-flow)
- Folyamat: gombóccal (bubble) (Process)
- Adattár: párhuzamos vonalpár (Store)
- Terminátor: téglalappal (Terminator)

00:05:45; 00:08:38

7, FunkModel, Context diagram

- Környezeti diagram
- Mi van a rendszeren belül, és mi van kívül?
- Processnek nincs emlékezete, csak transzformál, function
- Adattár: pl. változó, fájl
- **Egy** nagy process: Jegyeladás -> bontsuk kisebb elemekre!

00:09:00

8,

- Felül ContextDiagram, Két bejövő terminátor, Egy kimenő terminátor, Egy gombóc
- Mi van a nagy Gombócban? -> Önálló DataFlow Ábra -> Lehetnek Adattárak
- Új ábrán a Be- és Kimenetek száma megegyezik! -> Level Balancing
- Ezen Gombócok is felbonthatók DataFlow Ábrára-> Fa szerkezet
- Felbontás vége: PrimitívGombóc, egyszerű processz, nem kell tovább bontani
- Átlagos mélysége 1 felbontás
-
- Legelső szinten a folyamat: 0.Gombóc
- Ehhez tartozik a 0.DFD (a felbontása), amin van 1., 2., 3., Gombóc, stb.

00:13:51

9, Felbontás

00:15:58

10, Store, Adattárak

- Mindig valamilyen processhez kapcsolódik
- Ugyanaz a Store különböző szinteken is megjelenhet
- Időalap is egy Store
- **Jelölések** (írás, olvasás, mindkettő -> nem részletezzük, hogy sikerül-e)

21p: Feladatok

Bal fentről jobbra, majd le:

1, H: Két terminátor nem köthető össze, hogy lenne adatmozgatás? Kell Process

2, H: Nincs kimenete a folyamatnak

3, H: Két Store közötti adatáram…

4, H: Van a gombóc felé nyel (olvas a tárból, annak kell legyen kimenete)

5, I

6, H: Store-ból csak Processzel lehet kivenni

26p: Feladat, Mozijegy eladás

- Követelmény leírás -> formalizálni, specifikációvá alakítani
- Nincs explicite szó a vevőről! -> Beleértjük, hozzáadunk tartalmat.

Alapszabályok

- Először mindig Context Diagramot kell rajzolni: ki van kívül, belül
- Process: igék, ezeket kell valakinek megcsinálni (kívül vagy belül)
- Mozi, előadás (jellemzőinek…, definiált és be nem fejezett -> csak állapotok, attól még az előadásról van szó), vezető, pénztáros, készpénz, jegy (egyben számla), alapadat, szék, telefon
- Megadás, definiál, fizet, foglal, előjegyez, ad
-
- Egy gombóc: jegyeladás
- Mozi: maga az egész, kívül nincs, belül mit keresne? Kihúzzuk.
- Pénztáros: kezel dolgokat, egy gép, processzor -> de pl. egy webes rendszerben nincs, vagyis a tevékenység (jegyvásárlás, fizetés) független tőle. Jelenleg processzor, vagyis folyamatot hajt végre, de ez minket nem érdekel, kihúzzuk
- Telefon: szintén mindegy
- Marad Bemenő oldalon a Vevő és a Vezető, Kimenőn a Vevő.
- Ha van külső adat, akkor azzal belül is kell foglalkoznunk, pl. definiál -> processek
- „jegyeladás” -> igazából „széket adunk bérbe”, a jegy ennek csak bizonyítványa

- Műsorterv definiálás: eladható székek definiálása -> kimenet: eladandó székek

8. előadás

5.diasor/
11, Specifikáció

- Formális leírás készítése, mely eleget tesz a követelményeknek
- Egész szigorú szabályrendszert kell követnünk
- Funkcionális kép (mit kell csinálni), strukturális kép (kik a résztvevők), dinamika (időbeli viselkedés)
- Funkcionalitás lényege, hogy adat transzformációként képzeljük el (bejövőből valamilyen kimenő adat), formális leírása a ContextDiagram (ebbe beleugorva szétszedtük elemi folyamatokra, adatáramlásra, adattárolásra)
-
- Process speci: minden process felbontható DataFlow ábrává, míg primitív nem lesz -> ekkor szoktunk process specit írni:
    - Milyen paramétereink vannak (InputOut)
    - Tevékenységek leírása struktúrált angol nyelvvel + szerkezeti konstrukció a tevékenységek összefűzésére, kapcsolatuk szerint:
        - szekvencia, egymás után
        - szelekció, választás: vagy ezt, vagy azt
        - iterálunk, adott tevékenységet hajtjuk végre adott feltételszer/ig

00:07:45
13, Szerkezeti leírás, adatmodell

- Az adat (gyűjtőnév): mint a struct, benne a mezők az attribútumok
- Vannak attribútumai, jellemzői, melyek együttesen definiálják az adatelemet (entitás, egyed)
- Entitások kapcsolata

00:09:17
00:12:51
14,15, Kapcsolatok, ERD (EntitásRelációsDiagram, adatkapcsolati diagram)

- Adat, Kapcsolat, Kardinalitás (Multiplicitás), Modalitás (Opcionalitás)
- Kardinalitás: Két összekapcsolt adat közül az egyik 1 példányához a másik hány példánya tartozhat. Pl. egy embernek hány autója. Jel: csirkeláb
- Modalitás: Kapcsolat kötelezősége – Kötelező-e mindenkinek autót birtokolnia. Jel: karika.
- Adat, entitás: doboz
- Kapcsolat, reláció: vonal (vagy rombusz, lsd.:30p)

00:15:15
00:16:55
00:30:12
16-18: Példák (15p körül)

- Általában **páronként**i kapcsolatokra bontunk -> általában megtehető, ez a VeszteségmentesDekompozíció
- Pl. 33perc, Termék, Szállító, Alkatrész

00:35:05
19,

- Szekvencia (struct): data = a + b + c
- Selection (enumeráció): data = [a | b | c]
- Iteráció (Tömb, ismétlődés): data = {a} 2{door}5
- Összetett struktúrák
- Kulcs (1 v. több, ami 1 adatelemet azonosít): @ vagy # teszünk attribútum elé., pl SzemélyiSzám

Feladat 39p

- Fel kell ismerni az entitásokat (mely fogalmak írnak le létező dolgokat, embereket, eszközöket)
    - A szöveg is jól felülírja
- Ezek hogyan kapcsolódnak egymáshoz
-
- DVD, példány, szállító, ügyfél, kölcsönöz (határidő miatt), profil

00:55:10

20, XML, 55p

- Adatok leírása
- HTML -> szöveg (hyper text), XML tartalmaz grafikus megjelenítési infót is
- Tetszőleges tag-ek definiálása, de szigorúbb a html-nél
- Szintaktika (pl. elkezdek egy tag-et, azt be is zárom)
- Szemantika (Értelmes-e, tudunk-e vele mit kezdeni), leírása XML sémával vagy DTD-vel (DocumentTypeDefinition) (önleírók)
- **W3C társaság -> tutorialok**

01:02:26

21, XML Példa

01:04:28

22, Szintaktika

- Mindig kell záró tag
- CaseSensitive
- Megfelelően skatulyázott
- Root elem (ami között van minden)
- Space-ek megőrzése
- CR/LF konvertálása LF (írógépes példa… nem fontos?)

01:07:02

23, XML elemek

- Nyitó/Záró tag + Benne mindenféle
- Szülő - Gyerek
- Tartalom: Más elemek, Kevert, Egyszerű (text), Üres
- Pl 1.08p
- Lehet egyben nyitó és záró tag (üres): <email /> ugyanaz, mint <email></email>
- **Megkötések** nevekre: lehet betű, szám, egyéb; nem kezdődik számmal; XML szóval nem kezdünk; Nem tartalmaz Space-t

01:10:45

24, Attribútumok

- Megnevezés=”Érték”
- PL: <note date=”131313”> … </note>
- Attribútumban nincs belső szerkezet, nincs gyermek elem, csak egy szöveg
- Attribútum szerkezetének előírása DTD-vel

01:13:38

25, Elem vs. Attribútum

- Alapszabály: ha egy attribútum a szerkezetben levő dologra vonatkozik, akkor legyen **eleme**
- **Attribútum**, ha nem magának az ábrázolt dolognak a lényege, hanem az XML leírással (adatszerkezettel) függ össze (pl. személy sorszáma)

01:17:07

26,

- Név konfliktusok: ugyanolyan tageken belül másmilyen adat tartozna (pl.: name: vezetéknév, vagy kereszt?)
- Prefix: <a:person>
- Névtér: xmlns=”namespaceURI”
- (URI: Unified Resource Indentifier, Egységesített Erőforrás Azonosító)

01:20:20

27, Validáció

- XML doksi **jól formált**, ha megfelel az XML **szintaxisnak**
- **érvényes**, ha megfelel a DTD/XML Sémában megadott formának**, szemantikailag helyes**
- <![CDATA[„……”]]> CodeData, figyelmen kívül lesz hagyva

9. előadás

Speci – formális leírás
3 szempont: funkcionális kép (mit kell tenni), **szerkezeti** (elemek, adatok, kapcs), dinamikus (időbeli)

EA8/01:21:52
28, DTD – Document Type Definition
- Doksi struktúrát adja meg
- inline: XML fájlba szúrjuk a DTD-t
- external: fájlnév megadása (include)

EA9/00:06:05
29, Példa
- #PCDATA: Parse Code Data, lényegében text, amit fel fogunk dolgozni.
- Elem, Tag, Attribútum, Entitás (speciális xml karakterek), pcdata, cdata

00:08:25
30, Elemek
<!ELEMENT element-name category>
category = EMPTY, (#PCDATA), ANY
- Szekvencia (egyes elemek, más elemekből állnak)
  - <!ELEMENT element-name (child-name, child-name,..)>
- Iteráció (benne lévő elemek)
  - (child-name+) – 1 or more
  - (child-name*) - 0 or more
- Szelekció
  - Kérdőjel: 0 vagy egy: (child-name?) - 0 or 1
  - Felsorolás (enumeráció): (choice1|choice2|choice3)
- Vegyes
  - <!ELEMENT note (#PCDATA|child-name)*>

00:11:14
31, Attribútumok
Megnevezés – Érték páros
<!ATTLIST element-name attribute-name
attribute-type default-value>
- Attribútum típus
  - CDATA
  - (en1|en2|..)
- default-value
  - value, #REQUIRED, #IMPLIED, #FIXED
- Példa:
  - DTD: <!ATTLIST person number CDATA #REQUIRED>
  - XML: <person number="5677" />

Példák, 13p – 23p

Dinamikus Viselkedési leírás, elmélet (23p)

- Alapja, hogy a szoftverünket egy reaktív rendszernek tekintjük.
- Reaktív rendszer: külső hatásra ad valamilyen reakcíót.
- Külső hatás: event, esemény
- Esemény: valami történik, nem tevékenység, 0 idő (oszthatatlan)
- A rendszer reagál erre az eseményre (ami vagy látható kívülről, vagy belül hoz létre változást) -> tevékenységet végzünk
- A tevékenység is oszthatatlan: esemény, reagálás, esemény, reagálás
  (leglább modell, elmélet szinten)
- Aktivitás adatfolyamábrán -> process, reakció eseményekre -> viselkedés kapcsolata a funkcionalitással
- Esemény -> Reagálás. Hogyan írható le? Ugyanúgy kell reagálni az eseményre?
  Lift másodikra -> fel vagy lefelé indul? Mi a korábbi állapot, oda hogyan kerültem, stb…
- -> Szekvencia, sorrendi hálózat, eseménysorrendek -> állapotok, állapottábla
-
- **Példa 33p**
- Szöveget beolvasni, számot találunk, körülötte szeparátor, pl: 53 blbablb 10
- Minden sor végén írjuk ki a sorbna lévő számok összegét:
  alma 24 körte 73 → 97
- Beolvasás karakterenként -> új karakter egy esemény.
- Osztályozzuk az eseményeket! (következőállapot/tevékenység)

(állapot)	Szeparátor	Számjegy	Sorvége	Egyéb
Accept (elfogadunk)	Accept/	Get/X=f(c) (BejöttSzám Felvesz)	accept / print; Y = 0;	Wrong
Wrong (rossz adat)	Accept/	Wrong	accept / print; Y = 0;	Wrong
Get (SzámJött)	Accept/Y=Y+X (SUM+=Szám)	Get/X=X*10 + f(c)	accept / Y=Y+X; print; Y = 0;	Wrong

- **Jön esemény, Osztályozzuk, ez alapján lesz oszlopunk, vagyunk épp egy állapotban -> Állapot+Esemény helyen van egy tevékenység, Végrehajtom, Majd következő állapot**
- **54p:** DataFlow ábra a fentiekhez
- STD: State Transition Diagram

00:23:20; 01:06:40
01:07:16
01:08:18
32,33,34, Esemény, állapot

- Vezérlőjelek az adattranszformációhoz
- 34: Kérdőjel helyére jöhet az állapotgép

01:11:13
35, Példa, 01:11p

01:12:34
36, Kombinált DFD

01:12:53
37, Szintaxis gráf

- Főleg kifejezések megadására lehet használni
- Átjutni balról jobbra

00:18:45

38, Backus – Naur

10. Előadás

Specifikáció: formális leírás sw-re, mely kielégíti a követelményeinket.
Funkcionális, szerkezeti, időbeli kép

5.Diasor/

01:19:47
39, Algebrai axiómák, Absztrakt adatszerkezetek

- Adatszerkezetek: tömb, bináris fa, láncolt listák…
- Adatszerkezet felvételénél két elképzelés
    - Hogyan implementáljuk? Hogyan lehet felépíteni pl. Bin.Fát?
    - Legyen rejtve, absztrakt valami, amin lehet műveletet végezni (pl. beszúr)
      Nem foglalkozunk a megvalósítással, csak a műveletekkel, szabályokkal
- Milyen szabályok érvényesek a műveletre? Pl. mi történik beszúrás után?
-
- Adatszerkezet: Értékek halmaza
- Elem tulajdonsága, hogy a szerkezetbe felvehető
- Műveletek, van szignatúrája: pl. függvény prototípusa (név, paraméterek, visszatérési érték)
- Axióma: bal oldal = jobb oldal

00:11:32
40, Példa, Stack, 12p

- stack, item (amit bele tehetünk), boolean
- Szignatúrák: new, push, stb
- Axiómák: alap igazság, bal oldali helyettesíthető a jobboldalival.
  Két művelet egymás után oszthatatlanul végrehajtva:
    - Empty(New()), vagyis újat hoz létre, majd megnézi üres-e = true
    - Empty(Push(s,i)), vagyis egy strackbe elemet teszünk, majd megnézi üres-e: false
    - Stb
- Rajzolt Példa, 22p

00:14:25
41, Műveletek, 29p

- Három kategória: Konstruktor, Modifier, Behavior
- Konstruktor: minimális adathalmaz, szükséges ahhoz, hogy az összes lehetséges adatszerkezetet elő tudjam állítani. Pl (stacknél) : new, push -> Új stack
- Modifier: módosító, szintén új adatszerkezetet állít elő, de csak módosítja az adatszerkezetet, nélküli is tudunk létrehozni. Pl.: pop, empty
- Viselkedés: adatszerkezet jellemzője az eredmény
-
- Majd a Behaviort és Mofidiert alkalmazzuk a konstruktoron.

00:28:00
00:32:55
00:36:30
42-44, Példa, tömb, 33p

00:42:03
00:45:05
45-46, Példa, lista, 42p

00:56:30
01:03:25
47-48, Petri hálók 58p

- Konkurencia: versenyhelyzet (párhuzamosság?)
- Ezen viselkedés leírására jó a petri háló, irányított gráf.
- Egyik csomópont: place (karika), hely
- Másiok csomópont: transition (vonal), átmenet
- Place csak Transitionnal, és Transition csak Place-szel lehet összekötve
- Zöld pötty a karikán, placen: token
- Végtelen sok token (zseton) állhat
- Transition tud tüzelni: akkor, ha valamennyi InputPlace-en van zseton.
- Tüzel: elveszi a zsetonokat, és valamennyi KimenőPlace-re tesz +egyet.
- 48.dia: M = markerezés: Minden place-hez hozzárendelünk M term.számot, megadja, adott Place-en hány zseton van. Tüzelés megváltoztatja a markerezést

01:04:20
01:05:50
01:10:19
01:12:30
49-52, Példa, 1:04p

01:18:00
53, Kiterjesztések

- Tüzelési szabályok megváltoztatása (zseton elhelyezésre pl.)
- Tiltó bemenet (ha van zseton, nem szabad tüzelni)
- Prioritás tüzelésre (bal/jobb oldalra pl.)
- Ugyanaz a zseton vándorol, elfárad -> x tüzelés után eltűnik
- Token saját jelentéssel: Példa: könyvtár

01:23:02
54, Nagyon szép ábra, 1:23p

11. Előadás

00:00:41
00:01:00
00:01:20
00:02:50
00:05:40
00:06:25
00:07:08
00:07:44
00:09:20
56-64, Specifikáció

- Bevezetés
- Hatáskör (kapcsolat a világgal, kívül/belül, korlátozások),
- Felhasználói forgatókönyvek, profil (ki-mihez ért, milyen a felkészültségük), esetek
- Adatleírások: Adat objektumok, kapcsolatok, adatmodell (erd), adatszótár
    - Adatszótár: szómagyarázat (jellemző fogalmak kiemelése, magyarázata, felhasználva más adatszótárakat vagy közismereti szavakat (szinonímák))
- Funkcionális leírás, szöveges leírás, DFD, interfészet, transzformációk
  Egyéb követelmények hozzáadása, vezérlés
- Viselkedési modell: események és reagálás (event, állapot)
- Egyéb korlátozások
- Validációs kritériumok (hogyan tesztelünk, tesztesetek, elvárt eedmények)
- Függelék
    - TracebilityMatrix: programban van egy sorom, akkor arról tudnom kell, hogy mihez tartozik (melyik követelménynek felel meg?)
    - Metrikák, hogyan mérünk?

00:12:40
00:13:08
65-66, UsersManual, 13p körül

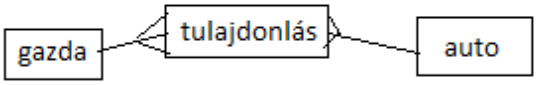
- Bevezetés, RendszerÁttekintés, GettingStarted, Rendszerhasználat, Összetettebb műveletek, Error üzenetek, How-to
- Bevezetés: általános áttekintés a projektre, milyen elérési pontok
- Rendszer áttekintése!
- GettinsStarted: install, login, főmenü, help
- Rendszer használata
- Extraszolgáltatások részletezése
- Error: jól értelmezhetők legyenek

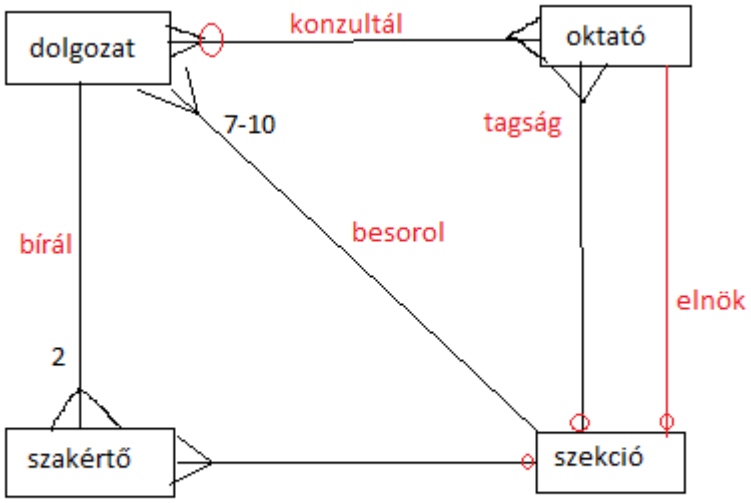
---------- Specifikáció vége ----------

PÉLDÁK megoldása

Nyilvántartás, adott időpillanatban ki volt az autó tulajdonosa. Egy autónak egyszerre csak egy tulaja van.

Autó, tulaj, idő





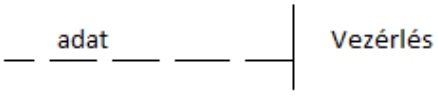
Állapot \ Esemény	Tudor	Vidor	Kuka
a	a/	b/get	-/
b	b/proc	a/write	a/write

A/B kitöltés véletlenszerűen (a és b is csak tipp, lehetne c is…)
A megadott tevékenységeket meg csak úgy be kell szórni valahova (get, proc, write)

---

	beta	X	C3
A1	A1/alfa	S2/	A1/ubul
S2	S2/	S2/X2	A1/alfa

adat =[beta, x, c3]
process: alfa, ubul, x2
adattárak: nem tudjuk



Algebrai axiómák
MBR(CRT(),i) = false
MBR(INS(s,i),j) = If (size(s) < 10 and i == j) true else MBR(s,j)
RMV(CRT(),i) = CRT()
RMV(INS(s,i),j) = If (size(s) < 10 and i != j) INS(RMV((s,j),i) else RMV(s,j)
SIZE(CRT()) = 0
SIZE(INS(s,i)) = If (size(s) < 10 and ~MBR(s,i)) size(s)+1 else SIZE(s)
//Halmaz, vagyis 1 elem csak egyszer szerepelhet!

---

12. előadás

Egy karakter, és az 1 karakterből álló string nem ugyanaz!!!

LAST(CRT()) = undefinied
LAST(ADD(s,x)) = x

END(CRT(),CRT()) = true
END(ADD(s,x),CRT()) = true
(üres sztring áll minden string végén) → end (s, crt()) = true, helyettesíti a fenti kettőt

END(CRT(),ADD(s,x)) = false
END(ADD(s1,x), ADD(s2,y)) = (x==y) && end(s1,s2)

DUPLO(CRT()) = false
DUPLO(ADD(s,x)) = igaz, duplo(s), vagy ha az utolsó karakter = x, az utolsó hozzáadottal egyezik

---

Petri háló, Elérhetőségi gráf: markerezés. 34p körül.
1100 → 0110 vagy 1001 → … → 0020 → 0011 → 0020 → 0011…

6.Diasor, 37p.

00:36:55
00:38:34
00:40:18
00:42:14
2-5, **Tervezés**: Mit kell csinálni ahhoz, hogy a speciben leírt programunk legyen.

- Meg kell érteni a SW struktúráját, meg kell határozni, hogy ez hogyna hozható létre
- Tervezői döntéseket dokumentálni (indokolni)
- Hogyan fogunk implementálni, teszteni?
- Architekturális, Részletes tervezés
- Architekturális: mik a fő komponensek, és azok hogyan kapcsolódnak egymáshoz?
- Részletes: az egyes eljárásokat, metódusokat, osztályokat meg tudjuk határozni

00:45:12
6, Tanácsok

- Ne legyünk csőlátóak – ne csak egy dologra koncentráljunk
- Követhetőség – minden tervezési lépés legyen indokolható
- Ne találjuk fel újra a kereket – használjuk a szabványokat
- Tervezés összhangban az alkalmazási területtel – ne szakadjunk el a megszokottól

- Fel kell készülni a változásra – menet közben is
- Váratlan dolgokra fel kell készülni
- Nem kódolás!
- Van minősége
- Terveket is felül kell vizsgálni

00:53:39

7, Absztrakció csökkentés

- Játék a részletekkel
- Gondoljuk végig, hogy mikor-minek a részleteivel foglalkozzunk. Pl: 55p.
- Részletek finomítása az alkalmazási terület és az informatika szemszögéből

00:57:55
8, Informatikai nézőpontban: Procedurák, Adatok, Kontrol

01:00:46
9, Egységbe zárás, encapsulation
- egymáshoz kapcsoló dolgoz összerakása
- probléma: mit-hova soroljunk? Példa 1.01p.

01:03:14
10, Információ rejtés
- Rakjuk össze úgy a dolgokat, hogy a dobozokba nem kell belelátni a működéshez, nem kell érteni a működését

01:06:12
01:07:00
11-12, Modularitás
- Hogyan szedjünk szét bonyolult dolgokat
- Szészedni jó, mert a rész-bonyolultságok összege kisebb, mint az egész egység bonyolultsága
- 1.08p, mese

01:08:35
13, Diagram

00:03:28
14, 1:21
- Adat alulról felfelé áramlik, akkor Afferens
- Adat fentről lefelé áramlik, akkor Efferens
- Fan-out: 1 modul hány alárendeltnek parancsol (7 körül jó)
- Fan-in: „hány főnökünk van közvetlenül”, erre nehéz számot mondani,
-
- Vezérlés és döntési hatáskör, döntés hasítás fogalma

13. Előadás

Tervezés: a speciből kiindulva megmondjuk, hogy mit kell csinálni ahhoz, hogy legyen szoftverünk. Munkaterv, kotta készítése, napi teendőkre -> részleteket feltárjuk

00:04:18

15,

- Vezérlési hatáskör: önmaga, és az alá tartozó egységek, pl: B,D,E
- Döntési hatáskör: azon modulok, melyeket a hozott döntések érintenek. pl: szemantika kérdése
- Ha a döntési hatáskör szélesebb, mint a vezérlési, akkor az a Döntés Hasítás (Decision Splitting)

00:10:50

16, Csatolás

- SW elemeink vannak (osztály, funkciton, stb). Akkor működik, ha össze vannak kapcsolva
- Elemek közötti kapcsolat erőssége -> minél erősebb, annál nehezebb javítani (12p példa)
- Laza csatolás: elem könnyen kiemelhető, cserélhető, átírható
- → Függőség: egy elem módosítása milyen munkát, veszélyt generál
- Szeretjük a lazán csatolt rendszereket

00:15:54

17, Csatolás Dimenziói

- Tárgya: mit is kommunikálunk
- Mérete
- Kapcsolás ideje, mikor jön létre
-
- pl. 16perc
- **Mit kommunikálunk?**
- Adat: leggyengébb, Ismerjük a paramétereket, primitív paraméterek (pl. int)
- Stamp csatolás: Összetett adatot adunk át (pl. rekord, struktúra). Ugyanazt a szerkezetet kell ismerni a két félnek -> függenek egy harmadiktól
- Kontrol: Vezérlést adunk át, pl. EOF, a másik modul működését befolyásoljuk, vagyis kívülről vezéreljül
- KözösAdatok: pl. globális adatok -> kinek a kezében van?
- Tartalmi jellegű: Egyik programrészből (pl. modul) a másik programrész **kódját** adatkezelem.
-
- **Mérete**
- Minél több adatot adunk át (paraméter), annál nagyobb valószínűséggel lesz baj.
- Legjobb: 1 paraméter.
-
- **Mikor jön létre**
- Program írásakor
- Kompájler (forrásból tárgykód: kereszthivatkozások kielégítése, pl. pointer offset)
- Linker: távoli kapcsolatok kielégítése
- Load, program betöltése: diszk -> memória, környezeti változók
- Futás közben
- -> Korábban: biztonságosabb, pl. kompájler figyel, míg pl. futás közben nincs ilyen

00:38:26
00:41:30
18-19, Kohézió: összetartóerő

- Objektumok mennyire állnak közel egymáshoz? Milyen a kohézió?
- Kohézív, ha egyetlen egy funkciót valósít meg, pl. gyökvonás. Csak azért van ott leírva minden, mert szükséges az adott folyamathoz
- Szekvenciális kohézió: lazább, pl. GetValidInput: Vesz egy inputot, és megnézi, hogy érvényes-e. Gyengébb, hiszen két dolgot végez el. Nem jó, bontsuk inkább két részre.
- Kommunikációs: van egy adatszerkezet, a közös, hogy rajtuk műveleteket végzünk.
- Procedurális: maga a tennivaló típusa tartja össze őket (pl. beolvasás), de típusonként mást kell csinálni (egy switch, és típusonként más végrehajtás)
- Temporális: Az idő tart össze, pl. Execption, Inicializálás alatt összehozott elemek, nem a funkcionalitás tartja össze
- Logikai: valamilyen logika tartja össze az elemeket. Pl. 30 funkcion-t szétrakni három felé – de milyen szempont alapján? Pl. ABC, de nem a legjobb: értelemszerűen kéne, szemantika alapján.
- Koincidenciális: Véletlenszerű

00:56:50
20, Történelem

- Sruktúrált programozás (pl. Pascal), Sruktúrált tervezés, Sruktúrált analízis, Formális Módszerek, OO orientáltság, Újrahasznosíthatóság, Komponens-alapú

00:59:03
21, Strukturált programozás

- Top-Down, fentről lefelé haladunk
- Oszd meg, és urlkodj, bontsuk szét
- Hierarchikus szerkezet

01:00:15
01:05:03
01:11:50
01:12:14
22-25, Strukturált tervezés 1:06p

- Jackson Structured Programming, JSP
    - Szekvencia, Felső elem az Alatta lévő elemek sorrendjéből áll
    - Szelekció, Választás
    - Iteráció, X sok A-ból áll

01:14:44
26, Strukturált analízis

- DeMarco
    - DFD – a problémát adatok áramlásaként ábrázolja
    - Draw Structure Chart – a programot funkcionális komponensek hierarchiájában írja le. A DFD-ből származik

14. Előadás

00:00:42

27, SW Architektúrák

- Milyen komponensekből áll, és ezek hogyan működnek együtt

00:03:26

28,

- Tervezői döntéseket foglalmagában:
- Elemek megválasztása, együttműködési mód
- Hierarchikus rendszer ezekből
- Minták alkalmazása

00:05:27

29, Egy architektúra stílus

- meghatározza a használható mintákat
- definiál egy szótárat, szókészletet
- korlátozások
- szemantikai modell (hogyan függenek, hívják egymást)
- Architektúrális kép: szeletek a különböző modelleken

00:07:44

30, Jó architektúra

- Rugalmas, változtatható
- Egyszerű
- Jól legyenek elválasztva a különböző szempontok
- Felelősségek kiegyensúlyozása
-
- Fő elemek
- Üzleti elemek, fő elemek
- Mechanizmusok, eljárások
- Processzorok és processzek (akik végrehajtják a folyamatokat)
- Hierarchia, rétegek és alrendszerek

00:12:23

31, Architekturális minták

00:14:50

32, Pipes and filters, csövek és szűrők

- pl: Unix vagy pl. dir | sort | more
- Filterek transzformálnak, a pipe-ok meg valahogy összekötik őket
- Előnyök
    - Támogatja az újrahasználhatóságot
    - Egyszerű karbantartani
    - Konkurens megoldás támogatása
- Hátrányok:
    - Batch, sorrendiség: csak sorbaveszi a lépéseket -> mi van, ha az egyik hibázik vagy elágazik?

00:18:54

33, BlackBoard

- pl. Adatbáziskezelő rendszerek
- Középen egy nagy adatbázis (oszlop-sorok)
- Hozzá csatlakoznak processzek
- -> Nagy közös adatszerkezetek, amin egymástól független processzek dolgoznak
- Tranzakciók
-
- Előnye:
    - Felelősségek nagyon jól el vannak különítve
- Hátrány
    - Nehéz tesztelni, pl. átírok valamit a központi adatbázisban, megnézzük, a többi kis rész működik-e továbbra is…
    - Nem épp hatékony, magas overhead, ráadásul gyorsan változik

00:20:54

34, Interpreter, 30p körül

- Egy egyszerű kis virtuális gép (pl. állapotgép, tábla)
- Fogom az Eventet, ez, és az aktuális állapot alapján becímzünk a táblába, a táblából kiolvasom az értéket, és átteszem a következő állapotra, parancskódra egy switch, majd megyek az elejére
- Engine (maga a program), ControlState (aktuális állapot), PseudoCode (lényegében a tábla), ProgramState (Adatelemek, amik kellenek a végrehajtáshoz, inputok)

00:22:34

35, ObjektumOrientált

- Absztrakt adatszerkezetet implementálunk, amik (az osztályok) egymás metódusait hívják meg.

00:18:12

36, Event Based, implicit invocation, 43p.

- „Előfizetői” minta
- Van egy eseményforrás, ahova beregisztrálnak azok az elemek, akik ezekben érdekeltek, és ha változik történik, értesülnek
- Sok overhead és nehézség, de használjuk, pl. ablakkezelő
- Előny: objektum interfész, sok event
- Hátrányok
    - Vezérlést nehéz összerakni
    - Inkább inputhoz köthető
    - Korrektség bizonyítása szinte reménytelen (kik kapták meg, milyen sorrendben -> ezt igazolni)

00:40:48

37, Layered, Rétegelt szerkezet

- Vízszintesen, a hívások között definiál szinteket
- Layerek kicserélhetők egy másik layerrel
- Csak az eggyel alatta lévő használhatja a szolgáltatásait
- Előny: Absztrakció növelése, Reuse, Könnyen karbantartható
- Hátrány: nem minden rendszernél használható, teljesítmény szempontjából probléma lehet a funkciók szétválasztása

00:43:02
00:49:31
00:51:20
00:53:00
00:54:14
00:55:11
00:55:33
38-44, Kliens-Szerver, 54p

- Ált. 3 réteg:
  - GUI, kezelőfelület
  - BOM (Business Object Model)
  - Database
- Sokszor 2 fizikai rétegen helyezzük el.

EA14/00:59:13 == EA15/00:00:40
45, **SOA**: Service Oriented Architecture

- pl. Argep.hu, brókerek (kigyűjti az infókat, és ajánlatot mutat)
- Webes szolgáltatáson keresztül (vagyis nem html) érjük el az oldalt
- WSDL: Webes szolgáltatások leírása
- SOAP: XML-es kommunikáció web szolgáltatások eléréséhez
- UDDI: „yellowpages", szakmák szerint milyen szolgáltatások hol vannak

EA14/01:07:05 == EA15/00:02:03
46

- WSDL: WebServiceDescriptionLanguage
- Interfészek: milyen szolgáltatás kérhető, annak milyen paraméterei vannak
- Message: milyen üzenetet kell küldenem, hogy egy szolgáltatást elérjek
- Service: interfészek fölötti kapcsolat
- Blinding: Összerendelés (fizikailag, url és protokoll szinten)

EA14/01:09:00 == EA15/00:02:37
47,

- SOAP: Simple Object Access Protocol
  - Üzenetformátum, RPC-t takar el (RemotePr…C, távoli eljáráshívás)
  - Különböző Node-okat definiál
    - Sender: küld egy üzenetet
    - Receiver: vesz
    - Intermediary: közbenső elem
  - Header,Body -> Strukturált

- UDDI: UniversalDescription, Discovery and Integration 1:12p
  - Aranyoldalak szerű könyv
  - Előfizethető

EA14/01:12:58 == EA15/00:03:46
48, SOE, Service Oriented Enterprise: Szolgáltatás mérnökség

- Szolgáltatási rendszer
- Szabványosítás
- Folyamatok

15. előadás

- BPEL: Üzleti folyamat leíró nyelv
    - Dokumentum flow jelleg, mikor-milyen lépések (amik esetleg újabb szolgáltatások igénybevételét jelentik), melyek mögött emberi tevékenység járhat
- Végrehejtásra két mód:
    - Orchestration: zenekar vezénylése, egy process ami leírja a folyamatot, tartozik hozzá egy végrehajtógép, ami a process elemeit végrehajtja
    - Choreography: ha elkezdődik a folyamat (pl. érkezik megrendelés), továbbítjuk az első állomásra, ahol az előírt tevékenységet végrehajtják. Majd, hogy hova kell továbbküldeni (szolgáltatást kérni), azt tudja az első állomás.
      Nincs központi rész
- BPEL4WS: BPEL web service-ekhez.
- WS-*: WebService Szabványok, ajánlások
- WS-Sec: üzenet szinten milyen titkosításokat érdemes használni
- WS-Rel.Mes: Hogyan tudunk biztonságosan üzenetet küldeni (elektronikus tértivevény, visszaigazolás): kizárólag egyszer jusson el a címzetthez.
- WS-Tranz: tranzakciókezelés: műveletek együtt kezelendők, mindet végrehajtani (rövid idejű tranzakció kezeléshez)
- WS-Policy: összefogja a különböző szabványok szerinti leírásokat
- WS-Coord: Hosszú távú tranzakcióknál használatos

7. diasor, JSD, Jackson State Development

00:22:55
00:29:10
00:31:52
00:34:35
3-6, Módszertan, ma már nem nagyon használják

- Csónakázós Példa -> hatalmas hiba: nincs szó magáról a csónakázásról
- Felismerjük-e a szereplőket, entitásokat?
- Modellben gondolkodjunk, ne funkciókban

00:39:33
7, Bankos példa:

- minden ügyfélhez ugyanaz a program
- tudom, hogy az idő nagy részében adott ügyfél programja nem fut -> használjunk egy gépet, amin az éppen aktuális ügyfél programja fut
- Elválaszhatjuk a program törzs (text) részét, és az adattartalmát – text rész állanfó, adatrész ügyfélfüggő
    - program text
    - state vector (Adatbázis)
- Rengeteg ügyfél processt hogyan tudnánk kezelni?
    - Elválik a program ütemezésének kérdése (mikor-melyik ügyfélé fut) és maga a program feladatának szerepe
- **Elválasztjuk** a modellt (ügyfél, csónakázás) és az implementálást (hogyan valósítom meg, sok ügyfél-sok process overhead)

00:44:14

8, JSD Lépések

- Modelleket készítünk:
  - Entity Action Step: Kik az entitások, az alapanyagok, ezekkel mi történik
  - Entity Structure Step: Entitáshoz hozzá kapcsoljuk azokat az eseményeket, amik vele történnek, és ezen akciókat időben rendezzük is
- Network Step: Processek hálója
  - Initial Model Step: Entitások történetéből megalkotjuk a processt, számítástechnikai modell
  - Function Step: Ha megvannak a modellek, akkor azokra kell tenni funkciókat
- Implementation Step:
  - System Timing Step: Időzítések, milyen ütemezést készítünk.
  - Implement. Step: Implementáció elkészítése

00:49:16

9, Példa, Bankprogram, 49p

- Menedzseljük a bankszámlát
- Rögzítjük a tranzakciókat (betét és kivét)
- Ha egyenleg minuszba megy, küldünk egy üzenetet
- Nyomtattunk egyenleget

00:50:23

10, Entity Action Step – mik az entitások, és mik az eseményei?

- Entitás: ügyfél (vagy bankszámla, mindegy)
- Események: Számlanyitás, Befizetés, Kivétel, Bezárás

00:51:03

11, Entitás Struktúra Lépés – megnevezett eseményeket rendeljük entitásokhoz, abban a sorrendben, ahogy előfordulhatnak

- Entitás: **Ügyfél**
- Megnyitás, **Törzs** (használat), Bezárás -> egymás mellett, szekvencia
- **Mozgások*** -> Tranzakciók; * mert iteráció, sok ilyen
- Befizetés^{o} Kivét^{o} -> karikás, szelekció

00:52:36

12, Initial Model Step – lerajzoljuk, tulajdonképpen mi történik, **53p**

00:54:45

00:57:48

13-14, Function Step – két funkciónk van

- Küldjünk üzenetet, ha egyenleg mínusz -> csak akkor fordulhat elő, ha pénzt veszünk ki.
- Egyenlegről küldünk infót: Adunk egy kezdeményezést (Enq.Inp), Adott ügyfél adatát elkérem.
- Lekérdező function: 14.dia

00:58:34

15, Időzítés

- Hogyan kezeljük a mínuszba hajtást – azonnal, vagy később?
- Milyen időszakra vonatkozik a lekérés?

01:00:03

16, Implementáció, 1:00p

01:02:29
17, Készítsünk ütemezőt, első változat
- Üzeneteket vesz iterációban (ciklusban)
- Vagy InputÜzenet
    - Maga az üzenet, és vagy
        - kell Exc report,
        - vagy nem
- Vagy LekérdezőÜzenet
    - Venni a megfelelő Inputot és
    - Generálni a megfelelő Outputot

01:03:37
18, Ütemező, másik eset – legyen nap vége, napváltás idejében generáljunk, és reagáljuk azonnal
- Inputnak periódusa van: vagy NapKözben, vagy NapZárás

Lépések részletesebben
01:07:07
19, EntitásAkcióLépés
- Kik az entitások, milyen események
    - Tipikusan főnevek (csónakázás, bankszámla, ügyfél)
    - Elszenvednek, vagy okoznak akcióat
    - Típus/példány probléma: típusokról kell gondoskodni (amikből sok példány van)
- Akciók
    - A külvilágban hajtódnak végre
    - Igék
    - Oszthatatlan

01:08:36
01:10:36
20-21, EntitásStruktúra
- EntityLifeHistory (ELH)
- Leírja az eseményeket
- Esemény csak **levélen lehet!** Ami nem levél, csak arra szolgál, hogy a struktúrálást lehetővé tegye számunkra (pl.:cust_body)
- Pl. 20,21. dia

01:14:40
01:16:25
01:17:30
01:19:01
22-25, InitialModelStep, Modellalkotás

01:20:41
26, Funkciók 1:20p
- A modellbe építhetők (mínuszba hajtás a számlán)
- Lekérdező (egyenleg)
- Iteratív, 27.dia

01:21:01
27, System Timing Step
- Korlátozások a rendszer kimeneteire (mikor értelme, mikorra vonatkozik)
- Állapotlekérdezésnél (ami nem feltétlenül eseményhez köthető), annak a frekvenciáját megadjuk
- Model-process közötti szinkronizálási folyamatok

- Idő-alap lekérdezés
- Hazárd lehetősége

16.előadás

00:13:50
00:18:31
29,30, Implementáció

- Egyik process hívja a másikat – de idővel váltanak, másik hívja az elsőt -> 3 függőleges vonal jelölésben.
- Helyette: Process ütemezés (Timing)

JSD lényege:

- Program és állapotok szeparálása, szeparált állapotvektorok adatbázisok, nagyon fontos modellt készíteni és vigyázni kell a funkcionalitással, funkc. túlhangsúlyozása zsákutcába vezethet

8. Diasor, OO, UML (obejtum orientált modellező nyelv), RUP

00:25:19
00:27:00
00:30:45
00:32:31
00:33:20
00:38:57
3-8, Áttekintés

- Alapja a strukturált programozás,
- Lépésenkénti finomítás
- Inforejtés
- Előnye
    - Tiszta struktúra
    - Nagyon jól dokumentálható
    - Hordozható, réteges
- Problémák
    - Túl széles rétegek
    - Globális változók / közös adat problémái
    - Közös adatoknál láthatósági probléma (27p, rajz)
    - Moduláris programozás
    - Nagy programok szétvágása – kohézív szétválasztás (összetartozó részek)
- Moduláris: adat és rajta operációk, interfészen keresztül férünk hozzá
- Absztrakt adatszerkezetek: Adat absztrakció és Procedurális interfész
- Problémák
    - Vagy név probléma, vagy kód láthatósági probléma (35p körül)
    - Hiányoznak a generikus struktúrák
    - InFix operációk: a+b vagy +(a,b) (-> utóbbi a prefix)
        - Vagyis Infix fenn van tartva meghatározott típusokra
- Paradigma: nézetrendszer
    - Gondolkodásmód beli váltás
    - Az alpvető konstrukció, amire építkezünk azok az objektumok
    - Objektumok vannak, amik együttműködnek egymással
    - Ennek a kialakítására OO módszertan
    - Objektumok (kb.: entitás),

00:44:08
9,

- Kliens: valamilyen szolgáltatást kap az objektumtól
- Objektum: szolgáltat a kliensnek (szerver), felelős ezért a szolgáltatásért, meg kell mondani, mi a felelőssége

00:46:15
10, File-Printer példa
- File: átalakítja magát szabványformátummá
- Printer: felismer szabványformátumokat
- PCL: PrinterControlLanguage

00:51:14
00:55:09
00:57:16
01:00:10
11-14,
- Kliens szolgáltatást kér (request), és visszakap eredményt
- OneWay Req: elküldjük a kérést, nem kapunk rá eredményt
- Error: kérést nem sikerül teljesíteni (pl. exception)
- Kérés, 55p
    - Mi az operáció (szolgáltatás)
    - Ki a szolgáltató objektum
- Request form: szignatúrája egy funkciónak: mi a név, milyen paraméterek, mint pl. egy űrlap
- Érték: bármi, ami egy paraméter helyében állhat
- Objektuk Referencia: érték, megbízhatóen kijelöl egy adatobjektumot, egy objektumra több referencia lehet, pl.: mail cím, mobil szám, pointer, url.
- Szolgáltatás kérés a **kliens nevében** lesz végrehajtva.
- Error: exception
- Nem csak szolgáltatást végrehajtást kérhetünk: keletkezzen új objektum, vagy épp törlődjön

01:00:42
15,
- Típus: állítás, amihez tartozik egy egy-paraméteres matematikai funkció bool eredménnyel, ami definiált ezen a típuson – ami kielégíti ezt, arra igaz a típus. pl: Integer, megfelel: 5, 7, stb
- Típusokat arra használjuk, hogy a paraméterek használhatósági körét leszűkítsük
- Típus extenziója: az egész set, adott esetben végtelen (pl. integerek)
- Object típus: tagjai objektum referenciák

01:02:20
01:06:21
01:07:40
16-18, Interfész
- Deklaráció, mely publikus jellemzők, kötelezettségek halmazaként jelenik meg, lényegében egy csomó lehetséges szolgálatkérés.
- Specifikál egy szerződést, mely megmondja, hogy az interfészt kielégítő, realizáló elemek felelősek azokért a szolgáltatásokért, amik az interfészben le vannak írva.
  Pl. Interfészben: levél kézbesítés (1.04p): az illető a kezébe kap egy levél típusú valamit. Az felel meg ennek, aki a levelet eljuttatja az adott helyre -> ez egy szolgáltatás, megvalósítja, pl. posta, futárok, (-> ezek objektumok)
- Típus: azon objektumtípusok, melyek kielégítik az interfészt
- Kielégíti a **Liskov** féle helyettesítési elvet: ha az A interfész B-ből származik, akkor egy olyan objektum, ami támogatja A interfészt, akkor azt bárhol lehet használni olyan helyen, ahol B-nek van deklarálva. Vagyis: B szupertípus, A származtatott típus. A kompatibilis B. A az egy B.
  Pl. 18. dia

01:08:35
19,

- Operáció
  - Olyan entitás, ami egy oszthatatlan dolgot definiál, amelyik valamilyen szolgáltatáskéréssel elérhető (vagyis egy szolgáltatás meghív egy operációt)
  - Általános szignatúrása:
    oneway (!= void), Visszatérési érték , Identifier (neve, pl. SetX), Paraméterek (5, 6, stb), Exception (milyen kivételt dobhat), Contexek

01:11:04
20,

- Paraméterek
  - Átadás irány (in: klienstől szerverfelé, out: szervertől kliensnek, inout: ide-oda áramlik)
    Általában In típus (a ReturnValue egy megkülönböztetett out)
  - Típusa

01:12:50
21,

- Végrehajtási szemantika: operációhoz van hozzárendelve, két fajtáa
  - At-most-once: max. egyszer
    Kérést megpróbálunk **egyszer** végrehajtani, ha nem megy, nem lesz végrehajtva
  - Beft-effort: request only, egyirányú hívás, leginkább aszinkron esetekben
    Példa: 1:15p,

01:19:07
01:20:17
01:21:30
01:22:26
01:22:38
22-26,

- Megvalósítás:
  - Végrehajtási modell: hogyan lesz a kérés végrehajtva
    Metódus hajt végre, ő az, aki operációt (szolgáltatást) megvalósít
    Végrehajtás: metódus hívás
  - Konstrukciós modell: hogyan definiálunk szolgáltatásokat
    Szerkezet készítés, ami leírja a metódusokat,
    Objektum implementáció
    4 rész:
    - Viselkedés: Objektum által mutatott szolgáltatások összege
    - Structure: belső attribútumainak a szerkezete
    - Állapot: ennek a struktúrának a kitöltöttsége, pl. 1:23:30
    - Identitás, egyediség: minden objektumnak saját élete van (referenciák problémája)

17. Előadás

8.diasor

00:06:33
00:08:40
00:09:40
00:10:43
27-30,

- Class: Leír egy objektum sokaságot, melyek hasonló viselkedéssel, struktúrával, szemantikával rendelkezik. Az osztály egy példénya egy objektum. „Objektumgyár”. UML jelölés.
- Öröklés: Meglévő osztályokat felhasználunk arra, hogy új osztályokat készítsünk.
- Nem csak öröklés során lehet a struktúrát kibővíteni, hanem felül is definiálhatjuk a meglévő (ősosztálybeli) fogalmakat. (29.dia)
- Absztrakt operáció (30.dia): csak deklaráljuk, nincs megvalósítva, de mindenhova öröklődik, ahol viszont saját példányt kell definiálni.

00:13:48
00:18:24
00:20:26
00:27:55
31,32-34, 16p. körül

- Design by Contracts. Kötelezettség és haszon .
- Precondition: előfeltétel (pl. kliens számára, hogy fizessen) (kötelezettség, haszon a szervernek)
- Postcondition: utófeltétel (pl. szolgáltató szolgáltasson) (kötelezettség, haszon a kliensnek)
- Class invariants: osztályre előírt szabályok, pl: két attribútum, az összegük legyen 100 Bármilyen művelet lehet, ha a kettő összege 100.
- Pl. Eiffel prog. nyelv

00:29:15
00:32:18
35, 36, Öröklés

- Figyelni kell az ősosztály elő- és utófeltételeire
- Class öröklés: egyszerűen felhasználjuk az osztály, kód öröklés, óvatosan.
- Interfész öröklés: helyettesíthetőségre építünk, aki az ős típus helyébe áll, szintén úgy fog viselkedni.

00:35:22
00:37:30
00:38:55
37-39,

- Open-Closed szerkezet
- Verziók létrehozása örökléssel?

00:40:15
00:42:30
00:46:04
40-42, Példa, 41p.

- Öröklés és szerződés viszonya
- Invariáns ugyanolyan vagy erősebb
- Előfeltétel ugyanolyan, vagy gyengébb, Utófeltétel ugyanolyan vagy erősebb

00:46:48
43, Öröklés, paraméterezés 48p, példa (állatos)

- Covariant: „A" osztály származtatásánál származtatom a paramétert is, de így együtt változik a paraméter is
- Contravarient: Leszármazott osztály beli művelet attribútuma egy ősosztály, és az ősosztálybeli nem változott műveletnél valamilyen leszármazott lehet. (nincs sok értelme?)
- Példa táblán 51p

00:57:25
44, Változók

- Mint egy zseb, bele tudunk tenni értéket
- Van típusa: milyen értékeket tehetünk a változókba. Statikus típus: Előre megmondjuk (ezt használjuk)
- Binding, kötés: a változón végrehajtott művelet kihez kötődik. A válotóhoz (pl. Integer, nem lehet rajta hatványozni), vagy a bele rakott értékhez.

00:59:46
45, Bindig\Typing, static\dynamic, 60p előtt

01:03:41
46, Law of Demeter (LoD)

- Ne fogadj el cukorkát idegentől
- Ne kössünk össze idegen objektumokat. Ki nem idegen?

01:06:44
01:08:00
01:10:38
47-49, LoD példa, 1:07p

- A-ban túl sokat tudunk, távoli eljárásokat akarunk meghívni -> nem biztos, hogy mindig így marad. Tipikusan: pl. Java osztálykönyvtárak
- Megoldás: kapcsolat rejtése

1:14p 9.Diasor, UML
01:15:27
2, Unified Programming Language, áttekintés

01:16:08

3, OMG.org, UML doksi letölthető

- Specifikálni, Vizualizálni, Konstruálni, Dokumentálni, erre szolgáló nyelv
- Mit? Termékeket, szoftverrendszereket
- Üzleti folyamatok modellezése, nem csak szoftverekre
- Történelem: Booch, Rumbaugh, Jacobson

EA17/01:19:44

EA17/01:23:47 == EA18/00:01:00

4,5, UML:

- Egyesít régebbi módszertaniokat
- Nyitott dolog, kiterjeszthető
- Standard, modellező nyelv, nem folyamat
- Szigorú jelölésrendszer
- Mese 1:25p

00:03:10

6, Nem UML

- Nem programnyelv
- Nem eszköz
- Nem fejlesztési folyamat.

00:04:00

7, Ábra, Struktúra

- Inftrastruktúra (Core).
- Körülötte a SzuperStruktúra, ezt használjuk ténylegesen UML név alatt
- Szétválasztásuk az UML2-nél jelent meg

00:06:15

8, UML Package

- Kompozit (összetett) struktúrák: Komponensek, Telepítés. Szerkezeti leírás
- Viselkedések

00:06:57

9, UML Diagram elemei

- Gráf, topológia nélkül
- Vizuális relációk
    - Connection: modell elemek összekötve (vonalak és 2d ábrák)
    - Containment: Szövegdoboz tartalma
    - Szimbólumok egymás mellett
- Grafikus szerkezet: minimális
    - Ikonok: egyszerű ábrák
    - Vonalak, nyilak, szimbólumok
    - Stringek

00:10:56

10, Elemek

- Name: megnevezés, model elem egyedi elnevezése
- Címke: String, amit grafikus elemhez tudunk kapcsolni
- Note: megjegyzés

00:15:27

11, 3 kiterjesztő mechanizmus (nyitott nyelv, terjeszthető)

- Constraint, korlátozás: egy elemhez hozzá kapcsolok valamilyen szemantikus korlátot
- Sztereotípia: UML készlet lehet nem biztosít elegendő szelekciót (pl. megjelölök minden ami I/O, az onnantól I/O sztereotípia alá tartozik). 16p körül. Speciális TagValue (csak név)
- Tag Value: Név-Érték páros

00:23:52

00:24:22

12-13, Diagram, 24p

- Struktúra és Viselkedés diagram

00:24:30

14, Diagram -> Struktúra -> Class

- Objektum, képes magát példányosítani
- Tartalmaz egyfajta speci, hogyan is nézzen ki az objektum példány
- Vannak Attribútuomk, Operációk
- 3 részből áll a doboz:
    - Név
    - Attribútumok
    - Operációk
- Analízis szintű és Implementáció szintű ábra

00:27:01

15,

- Név doboz
    - Sztereotípus
    - Név (dőlt: absztrakt)
    - Properties
- Standard sztereotípiák
    - Interface!
    - Utility: 29p.
    - MetaClass: osztályok felett áll, példányai osztályok. Ritkán implementálunk (SmallTalk)

00:33:08

16, Attribútumok, 34p körül

- Birtokolt tulajdonság

00:35:28

17,

- elem jelölések
- ha nincs jel: nincs jelölve a láthatóság, de attól még definiálva van! És persze, nem public – de nem is tudjuk, hogy milyen

00:36:52

18, 36p körül

- Ha attribútum sora alá van húzva, akkor class-scope (vagyis static, az osztály hatáskörébe tartozik).
- pl. van a1 és a2, akkor mennyi lesz az „a*" attribútumok értéke. NINCS több „a", csak 1, a végeredmény 5 lesz a példában.

00:38:27

19, Property modifiers

- ReadOnly: csak olvasható a tulajdonság
- SubSets: attribútumok vmilyen részhalmaza szerepel, pl. származtatás során ősosztály attribútumainak részhalmaza
- Union: Fel-származás esetén fordul elő. Pl. leszármazott osztályoknak vannak attribútumai, az ősosztályban pedig ezen attribútumok összessége
- Redefiens: a propertyt újra definiáljuk
- Orderes: rendezett (több elemnél)
- Unique: Egyedi értékek, általában kifejezést adhatunk meg

00:40:50

20,

- Attribútum – Kollekció típusok 41p.
- Set: nem rendezett, de minden elem egyedi (isUnique);
- OrderedSet: rendezett halmaz, egyedi elemekkel
- Bag: nem rendezett, nem egyedi elemekből áll
- Sequence: rendezett, de nem egyedi elemekkel

00:41:48

00:46:36

21-22, Enumeráció

- Képes enumerációt attribútumként kezelni

00:47:22

23, Operációk, 48p

- Amiket metódusként implementálunk az objektumba

00:49:40

24, Operations property

- query: nem változtatja meg magát a szerkezetet
- redefines: operációt felüldefiniálok
- ordered, unique: visszatérési értékekre vonatkozik

- korlátozások

00:50:28

25, Operáció diagram

- Operáció zárójelekkel

00:51:19

26, Végrehajtás szemantikája 3 féle lehet. Probléma: mi van akkor, ha kérés végrehajtás közben még egy kérés fut be? 52p körül példával

- Szekvenciális: kizárt, nem jön végrehajtás közben újabb kérés
- Őrzött: befuthat ilyen esemény, de van egy mechanizmus, amivel ezt kezelem (pl. nem törődünk vele)
- Konkurrens: „felvesszünk a telefont, és valamit csinálunk vele”

00:57:07

27, Active object

- Jelölés: dupla fallal
- Saját szála van. Pl. átlag programvan 1 szál van, sorban fut. Itt behoz egy új szálat, az eredetitől függetlenül fut

00:58:13

28, Template, 59p

- Parametrizált osztályok

00:59:40

29, Classifier: definiál egy meta-osztályt az olyasmi elemek fölé, amik olyanok, mintegy egy osztály (pl. interfész)

EA18/01:01:52
01:04:13
01:08:33
01:14:48
EA19/00:01:50
00:03:44
00:04:26
00:07:54
00:12:08
00:18:10
00:19:22
00:20:38
00:21:15
00:22:47
00:25:22
00:26:53
00:28:51
00:29:12
00:32:47

30-48, Relációk 1:02p. Össze vannak közve UML modell elemek

- Legfontosabb: **Függőség** (Dependency)
    - Alkalmazásom használja a stack osztályt, akkor függünk attól az osztálytól
    - Application - - - - -> Stack
    - Mindenféle használat egyfajta függőség.
    - Csak egy irányba működik, irányított.
- Vannak alváltozatai, pl. **Asszociáció 1.04p**
    - szemantikus relációt definiál (tartalmas kapcsolat)
    - osztály jelleg: példányosodik
    - Folytonos vonallal:
    - Person ---------- Car
    - Két elem közötti kapcsolatot ábrázolunk, végén osztály, classifier
    - Asszociáció két vége: **Role, szerep**
    - Végén multiplicitás: * (0 vagy több)
    - Asszociációs osztály: ha saját attribútumai vannak az asszociációnak, pl. 1:13
    - Van neve
    - /, örökölt asszociáció
    - Property string
    - Célja lehet specializálás, újradefiniálás
    - Constraint (megkötés): Kocsinak vagy Person a birtokosa vagy Company, a kettő között fennáll egy {xor} megkötés
    - **AssociationEnd (role?)**
        - Multiplicitás: 0..2 (alsó..felső korlát)
        - Név
        - Navigálhatóság: hatékony megoldás..? 1:16 Nyíl jelzi irányát
            - Nem navigálható: B-ből Á nem érhető ál (nincs hatékony megoldás), áthúzva.

- Role-nak is van láthatósága, tulajdonoson kívül látja-e más
- Role tulajdonosa lehet az asszociáció (vagy a hozzá kapcs. classifier)

19. előadás

- **Qualifier**, minősítő 36, dia
  - Attribútum, vagy azok egy csoportja
  - pl. Multiplicitás csökkentése (indexelés)

- **Aggregáció**: aggregation rész-egész viszony
  - pl. Autónak van ajtaja
  - Fontos szemantikus tartalom
  - Bináris asszociáció
  - Tranzitív, anti-szimmetrikus (autó része ajtó, fordítva nem igaz)
- **Osztott (shared) aggregáció**:
  - Ritkább, 1 dolog Több egész része
    (fal része 2 szobának, lebontva a két szoba megszűnik)
  - Komponens reláció (autó-ajtó)
- **Kompozíció**
  - Egy egész része
  - Propagációs szemantika
  - Egészen végzett művelet, a részeken végzett műveletek összessége, pl. könyv egy aggregátum (összesítés), fejezetek összessége (ami lapok összessége, ami bekezdések összessége…)
- **Nested Class** (Beágyazott osztály), 40.dia
  - Csak és kizárólag az őt tartalmazó osztályon belül látható
- **Generalization**, öröklés 41.dia
  - Fent álló őselemnek a leszármazottai valamilyen specializációi
  - Shape -> Polygon, Ellipse, stb
- **Interfész**
  - Public features and obligations
  - Ha az interfész attribútumokat deklarál, abból még nem következik, hogy azok attribútumként lesznek elérhetők (hanem pl. műveleteken keresztül)
  - **Példák 29p.**
    - UML 1.? Szaggatott vonal, nagy (üres) nyíllíl: interfész implementálása
    - UML 2: Szaggatott vonal, normál nyíl, ráírve, hogy „realize”
      - A régi használatát várják el
    - Nyalóka: vonal, végén karika: ugyanaz, mint ez előzők
    - Elvárt interfész: Mit várok el; Miközben szolgáltatást végzek kell valaki, aki egy interfészt megvalósít. Jel: Nyalóka, túloldalon félkörrel bevonva

00:34:46
00:34:59
00:36:25
00:39:55

49-52, Object Diagram

- Nem osztályokról beszélünk, hanem objektumpéldányokról
- Jim:Person (jelölésben is aláhúzva)
- 37p Class-Car

PÉLDÁK, gyakorlás, 42p

- 42p. Első
    - Aggregáció: Nincs. Asszociáció egyik végén kéne legyen egy **rombusz**
    - Függőség: Nincs. Szaggatott vonal, sima fejjel
    - Osztály metódus: do(), mert a van húzva
    - Példányosítás: szó sem volt róla…
    - Multiobject: Nincs (nem volt anyag, UML2-ben nincs is?)
    - Realizálás (implementálás): Van, B és C az A interfészt (szaggatott, nagy nyíllal)
    - Kollaboráció: Nincs (Még (!) nem volt róla szó)
    - Navigáció: Van hát, a nyilvégeg (asszóciációknak van)
    - Qualifier: Nincs (asszocióció végén egy minősítő doboz)
    - Absztrakt osztály: Nincs (Italic, dőlttel jelölnénk)
    - Sztereotípia: Van (<<Interface>>)
- 46p Második
    - 10-10, mert csak 1 a van, mival az static (aláhúzott) !
- 48p Harmadik
    - Egy ObjektumDiagram, rajzoljunk osztálydiagramot!
- 54p Negyedik
- 1:02 Ötödik
- 1:08 Hatodik
    - B bárhol helyettesíthető E-vel, mert E a B leszármazottja: +++
    - C bárhol helyettesíthető E-vel, mert D a C leszármazottja: --
    - F set(a:A) függvénye kaphat paraméterül D-t, mert a D megvalósítja az A interfészt: +++
    - D add(b:B) függvénye meghívhatja egy maraméterül kapott E count () metódusát, mert E megvalósítja az A interfészt: B-t kérünk paraméternek, fogalmunk sincs, hogy E micsoda: -+
    - E meghívhatja egy D add() metódusát, mert van közös ősük: ++-
    - E nem hívhatja meg egy D bar() metódusát, mert a metódus protected: D-nek nincs bar() metódusa (B-nek van, de private): +-
    - F meghívhatja egy D add(b:B) metódusát, mert E egygszerre a B és C osztály leszármazottja: -+
    - D add(b:B) metódusából nem hívhatjuk meg D do() metósudát, mert a metódus absztrakt: nem absztrakt a metódus, hanem static, így meghathatjuk: --

20.előadás

PÉLDÁK, feladatok, gyakorlás

00:16:35

54, Package Diagram, 18p

- Package: általános csomagoló eszköz, bele tehetünk UML elemeket
- Elemek bizonyos névtéren
- Minden elem csak 1 package-ben
- Nevet vagy bele írjuk (ha nincs benne több doboz) vagy a fülre (jelölésben)

00:18:15

55,

- Visibility:
  - public vagy private
- Package Import: egyik package tartalma a másikban is látható, két fajta (sztereotípia):
  - Access: Így elért package később már nem látható (pl. felvesszük accessal, majd ha mineket akarnak importálni, akkor nem látják ezeket)
  - Import: Használható később is
  - 20p-nél példa

00:21:20

00:22:11

00:24:42

56-58, Package Merge

- Azonos elemeket megpróbáljuk összehozni. 22percnél példa
- Gyakorlatban ritkán használják

00:24:55

00:30:46

59-60, Komponens diagram

- Fizikailag létező, helyettesíthető része a rendszernek, amely bizonyos interfészeket szolgáltat, és számít bizonyos interfészekre
  - Deployment komponensek: telepíthető komponensek, melyek ahhoz kellenek, hogy egy programot telepíteni tudjak
  - Work Product: azok a sw komponensek, amelyek a fejlesztés során keletkeztek (pl. source)
  - Execution komp.: Végrehajtás közben keletkező komponensek. Pl. program hoz létre fájlt
- Jelölés: doboz, szélén 2 csatlakozóval, alul nyalókákkal (interfészek, logikai kapcsolatok)

00:32:59

61, Port

- Definiált csatlakozópont, melyen keresztül bizonyos interfészek elérhetők, kívánatosak

00:34:46
00:35:57
62-63, Connector

- 62, Assembly C.: két komponens között az elvárt és szolgáltatott interfészek illeszkednek (35p)
- 63, Delegation C.: egy komponens újabb komponensekből áll, az ezen belül megjelent elvárásokat és szolgáltatásokat ki kell vezetni a felső komponens széleire -> erre való a delegációs konnektor

00:38:24
00:39:08
64-65, Komponens diagram

- Komponenseket, interfészeket, köztük lévő kapcsolatokat próbáljuk leírni

00:39:20
00:42:46
00:44:50
66-68, Component Structure Diagram - Kompozit struktúra diagram, összetett struktúra diagram

- Kollaboráció: 39perc. Adott funkcionalitásban érdekelt elemek kiemelése (pl. van egy funkció, ez mely osztályokat érinti, highlight)
- Szaggatott vonalú ellipszissel jelöljük, benne az osztályok, kapcsolat
- Kollaboráció, újabb kollaborációk készítésére

00:46:44
00:46:50
00:48:35
00:49:52
00:51:10
00:51:56
69-74, Deployment Diagram, Telepítés

- Artifact, termék: egy komponenshez tartozik valamilyen termék, megjelenik pl. .jar formában
- Node: Sztástechnikai erőforrás, melyre termék telepíthető végrehajtás céljából. Inkább logikai értelmű szerep. Dobozként jelenik meg (térbelinek tűnik)
- Device: Fizikai eszköz, számítási képességekkel rendelkezik, telepíthető rá az eszköz. Hozzárendelhetünk végrehajtási környezetet (mely szükséges ahhoz, hogy adott elemek végrehajtása megfelelő legyen)
- Deployment diagram: hogyan helyezzük el a device-okon, node-okon az egyes termékeinket, artifactjainkat.. 73,74.dia ábrával, 52p körül

00:53:38
75, Behavior Diagram, Viselkedés Diagram, 53p

00:54:00
76, UseCase (használati eset)

- Valamilyen elemi funkcionalitást jelöl,
- Actor: aki egy usecase funkciót kívülről kezdeményez, és a funkció lebonyolításában érintett.

Pl. Készpénzfelvétel. Actor: mi, emberek. UseCase: az automata használati esete. Leírható, mint egy együttműködés az Actor és a Rendszer között.

00:56:57

77,

- Maga a UseCase egy táblázatos leírás, strukturált szöveg
- Elnevezés (vásárlás), Actors (vevő, eladó), Cél (terméket megvenni), ÁttekintőLeírás (ügyfél odaér, pénztáros rögzíti a termékeket, végén fizetés), KövetelményekreHivatkozás (miért is csináljuk)
- Részletesebb leírás (Actor és a Rendszer közti interakció)
    - Szöveges leírás, dialógus, 1:00p példa

01:01:09

78, Use Case Diagram

- Statikus kép arról, hogy kik az együttműködők és milyen funkcionalitásban
- Actor: pálcikaember (ha belső Actor: 1:16p-nél), lába alá írva a szerepe (vagy Osztályt rajzolunk [doboz], rajta <<actor>> sztereopítia)
- Kapcsolatok felrajzolása (függőségek, asszociáció, generalizáció)

01:03:05

79, Ábra

- Vonalak: asszociációk, általában nem irányítottak (oda-vissza, interakció)
- Multiplicitás: egy ügyfél, több UseCase: pl. több browser (vagy fordítva)

01:07:43

01:08:14

80-81, Kapcsolatok – Példa: 1:08

- Asszociáció: mint az osztályoknál
- Include: Kötelezően tartalmatta az Inlcude-ált elemet.
    - Kötelező beágyazás.
- Extend: Adott elemez adott körülmények között kiegészítjük speciális dolgokkal.
    - Opcionális Kiterjesztés

01:14:19

82,

- Generalizálás: akár Actorok között is (itt is van helyettesíthetőség, pl. Kasszás -> Felügyelő, aki szintén tud eladni, de tud adni hitelt is)
- **BelsőActoros** példa: 1:16: Tetris
    - Az elemeket lehet forgatni
    - Az elem folyamatosan esik lefelé -> BelsőActor, szereplő, aki lépteti lefelé az elemet

01:22:21

01:23:05

83-84, Use Case Jellemzők

- Eseményfolyam
- Speciális követelmények: nem funkcionális (pl. biztonság)
- Pre és Post kondíciók
- Kiterjesztési pontok
- Relációk (extend, include)
- Activity diagram (flow chart)

- Use Case Diagram

21, Előadás

00:03:01

86, Interaction, általános meggondolások, diagramok

- Szekvencia: leggyakrabban használt
- Kommunikációs: szintén sűrűn használt, ugyanazt írja le, mint a szekvencia, a két diagram nézőpontja kicsit más
- Interakciós, Időzítő, Áttekintő, IterationTable, ezekkel nem igazán foglalkozunk
- **Eddig szerkezetettek foglalkoztunk, most dinamikával, hogyan történik a szolgáltatáskérés, milyen sorrendben. Meséket fogunk lerajzolni**

00:05:37

87, Fogalmak

- Interakció: együttműködés, mindig valamilyen Classifierhez van rendelve.
  Pl.: Automata: egy eseménysorozat (kártya, nyelv, funkció, stb)
- Trace: események egy sorozata, szekvencia
- Esemény: u.a., mintkorábban: egy rendszert érő külső hatás, és arra reagálás

00:07:49

88, Események

- Execution event: Végrehajtással kapcsolatos események
- CreationEvent, DestructionEvent: objektum létrehozása, elpusztítása
- Message: üzenet küldés/fogadás, ekkor végre lesz hajtva valamilyen operáció
    - Egy specifikáció, leírja, hogy milyen kapcsolatnak kell lennie a kliens és a szerver között

00:10:59

00:13:08

89-90,

- Signal, jelzés, rokon az üzenettel (üzenet célja, hogy egy operációt hajtsunk végre), a Signal jelentősége, hogy egy állapotváltozást fog generálni. Kliens szempontjából szinte mindegy (őt nem érdekli, hogy oldja meg a szerver)
- MessageEvents
    - SendOperationEvent: Kliens oldalon
    - ReceiveOperationEvent: Kiszolgáló oldalon
    - SendSignalEvent: Kliens oldalon
    - ReceiveSignalEvent: Kiszolgáló oldalon

00:13:16

91, Interakció

- Trace-ek (események egy szekvenciája, eseménysorozatok) halmazaiból egy pár
- Egyik halmazban az érvényes eseménysorozatok, másikban az érvénytelenek
- Pl. Automata: Nem megengedett események is vannak (rossz pin, kártyát bevonni)
- Conpositional, Kompozíció elve: ha van két ilyen eseménysorrendem, akkor ezekből összekombinálhatok újabb összetett eseménysorozatot
- Interleaving Semantics, Átlapoló szemantika elve: Ha össze merge-elek különböző eseménysorokat, azt azzal a feltétellel tehetem meg, hogy 1-1 eseménysoron belül a sorrend változatlan

00:16:05

00:16:15

92,93, Sequence, Szekvencia Diagram, 18p körül

- Doboz, cc:C, olyan, mint egy objektum, DE nem az (ha az lenne, alá lenne húzva) -> inkább osztály, nehéz elválasztani. UML2-ben nincs aláhúzás.
- Függőleges szaggatott vonal: Idő múlása
- Diagram idő ábrázolására: egyes osztályok között menő üzenetek időben hogyan helyezkednek el. Lejjebb, később
    - Függéleges szaggatott vonal: Idő
    - Fej + FüggőlegesSzaggatottSzár: LifeLine, életvonal
    - Rajza van az üzenet, ami eseményeket határoz meg
    - **Szinkron** üzenet: kis, **teli fej**. Lényegében C átadja a vezérlést A-nak
    - **Szaggatott, nyitott fejű vonal: return**
    - Szaggatott vonalon függőleges **doboz**: Execution Specification, Execution Bar
        - Valamilyen **tevékenység** folyik
    - Az egész **körül egy doboz!**
    - Bal felső sarokban **név**

00:23:54

94, Másik szekvencia diagram

- Magát a bar-t nem ábrázoljuk (bár lehetne)
- **Nyitott fejű nyíl: Aszinkron küldés**
- Ferde nyíl: üzenet küldés sokkal korábban van, mint a megérkezése (közben megérkezhet másik üzenet is)
    - Pl. kártya kiadás indítás
    - OK üzenet megjelenik -> gyors, elektronikus
    - Kártya kiadva -> lassabb, mechanikus feladat
- Üzenetküldés a NagyDoboz határára: ha ezt beleraktuk egy másik diagramba, ott felhasználhatjuk

00:28:17

95, Csúnya diagram, 28p.

- szaggatott vonallal osztott rész: régió
- Alt: különböző esetekben, különböző régiókban írt szekvenciák valósulnak meg
    - Feltétel: csak olyat, aminek kiértékelése önmagában interakciót nem igényel
    - Szaggatott vonal és nyitott hegy: új objektumpéldány (->ostobaság, UML2)
        - Aszinkronnak látszik, node az szinkron. Példa, 32p
    - Idővonalon X : meghal
- Opt: Olyan Alt, ahol csak 1 régió van
- Break: Olyan szekvencia, ami valamilyen spéci esetben (exception) hajtódik végre
- Par: Szintén régiók vannak, egyes régiók párhuzamosan hajtódnak végre
- Neg: Olyan szekvencia, ami nem megengedett (Invalid)
- Critical: kritikus régió, pl. kölcsönös kizárás, gyakorlatilag oszthatatlannak kell lennie
- Loop: Ismétlődik, felül feltétel

00:39:52

96, Még egy szekvencia diagram

- Példa máshol definiált szekvenciadiagram beépítésére, 40p

00:43:50

98, Communication, Kommunikációs Diagram

- Ugyanaz, mint a szekvencia diagram, csak másik nézetből, átalakíthatjuk az egyiket a másikba

00:44:28

99, Ábra

- „ledöntött" lapok, életvonal merőleges, „belemegy" a lapba, a tetejét látjuk
- Így nem tudunk időbeliséget jelölni
- Elemek nem síkban, hanem térben
- Üzeneteket számozott nyíllal jelöljük: 46p
- Kapcsolatba hozható egy osztálydiagrammal
- Példa 49p

FELADAT, Gyakorlás, 51p

- UseCase Bankos példa, 79-82 diák, úgy nagyjából
- Síkvektor, Osztály Diagram, Szekvencia Diagram, 1:02
- Doktoros, 1:14
    - A vérétel és a megküldés aszinkron események
    - Ha alami olyasmiről van szó, ami a továbbiakban is részt vesz valamilyen műveletben (pl. beutaló, recept), akkor tűntessük fel

22. előadás

1 Feladat

00:09:00
00:11:05
101-102, Timing Diagram

- Hasonló, mint Digitnél
- Interakciókat ír le, de a hangsúly azon van, hogy az időzítéseket ki tudjuk fejezni
- TimeLine vízszintes vonal
- Több állapotnál több szintet vehetünk fel (nem csak 0-1)
- Lehet egyszerre több állapotban (dont care)
- Nyilak: Események egymáshoz kapcsolódnak, de jelenthet üzenetet is

00:14:46
00:15:06
104-105, Interaction Overview Diagram, 15p körül

- Aktivitás diagramra hasonlít (ami meg egy UML-es tuningolt Flow-Chart)
- Időtartamot is jelölhetünk

00:16:42
00:32:00
00:32:08
00:32:15
00:32:31
00:41:19
00:45:59
00:50:42

107-114, State Machine

- Szintén hasonlít a Digites állapotokhoz és eseményekhez
- Lekerekített doboz, beleírva az állapot neve
- Állapotátmeneten az Event, Esemény
- Event (argumantumok) [feltételek]
    - EntryAction: Tevékenység az állapotba lépéshez kötődik
    - ExitAction: Kilépéshez kötődő feltétel (függetlenek attól, hogy honnan jövünk)
    - On MyEvent: MyAction – saját esemény definiálása, állapoton belül maradok, végrehajtom az akciót, nincs Entry meg Exit
    - Do: Do-Activity, adott állapotban nem csak „van", hanem közben csinál is valamit, számol -> hosszabb távú dolog
- Transition: állapotátmenet
- Composite State: Két fajta, 32p
- History: egy interrupt hatására kilépünk, Except helyen végzünk, visszalépünk -> nem a belépési pontba, hanem egy History-be, ami tudja, hol álltunk. Ha valamilyen ok miatt mégsem üzemel a History, akkor van egy előre definiált érték, ahova ilyenkor lép
    - Deep History: H*, Nem csak annyit őriz meg, hogy B-ben voltam, hanem a B belső állapotait, mindet
- Konkurrens Alállapotok: egymással párhuzamosan definiáltak. Pl. Autónál egy állapottér a sebesség, egy másik az AblaktörlőSebessége (ki-be, gyors, stb), Harmadik állapot a Lámpa például. Ezek egyszerre fennálló állapotok. Példa: 49p előtt
- SynchStates: Felső régióból csak akkor tudjuk átmenni (másik állaptba pl.), ha az alsóban átmentünk egy másikba -> Régiók közti szinkronizálás.
  Alsó szinte amikor átlépünk, egy tokent adunk fel, így a felső szint is ekkor tud lépni

00:52:30

115, Activity Diagram

- Megfelel a FlowChartnak (ami egy spéci állapotgép – FlowChart doboza egy állapot, do activity, majd automatikus továbblépés)
- Nevezik ActionState-nek is

00:52:55
00:54:05
00:54:40
00:57:16
00:58:42
116-120, Elemei

- Fork and Join: párhuzamos működés megengedett
- SwimLanes: úszósáv (vízszintes eltolás)
- Action-Object Flow
- Ábra, 56p körül

FELADATOK, Gyakorlás 1:00

- Első: mentés, állapotot meg kell őrizni, majd visszaállítani
    - Mentés kész – hogyan oldjuk meg, hogy ne legyen módosítható?
        - Pl. setState csak egyszer hajtható végre
        - 1:15p… Játék interfészekkel
- Második: Izidor, Ajándék, Jézuska
    - Borzalmas… 1:22p
- Harmadik
    - Get és Remove nem a változónak megy, hanem a Kollekciónak!

23, Előadás

FELADATOK, Gyakorlás

- …
- Harún Ar-Rasíd, 12p

121, Az UML-en túl, 24p

- MetaModel 4 rétege
    - M3: Típusos elem (meta-meta model)
    - M2: Attribútum (meta-model)
    - M1: Példánya a felette lévő metamodelnek (PlateNumber)
    - M0 User object: specifikus elem (ABC123)

123, MOF, MetaObjectFacility

- OMG Standard
- stb

124, EMF EclipseModellingFramework

125, XMI: XML Metadata Interchange

- Tag-eket arra találtuk ki, hogy velük UML modelleket írjunk le
- Ennek alapján generálhatunk kódot

10.diasor

00:40:54

3, RUP: RationalUnifiedProcess

- Hogyan kell használni az UML elemeket, milyen sorrendben, hogyan érdemes fel fűzni ezeket a technikájat
- Architektúrális process: milyen komponensek, hogyan vannak összekötve?
- Tulajdonságai
    - Iteratív és inkrementális
    - Use-Case vezérelt
    - Architektúra-centrikus

00:43:24

4, Iterative and incremental

- Viszonylag korai visszacsatolás a Usernek
- A Team-nek is van ideje, megismerni az alkalmazási területet
- Visszacsatolások alapján korai módosítási lehetőségek
- Korai tapasztalatok a rendszer integrálásában és tesztelésében

00:46:02

5, UseCase driven

- Követelményeinkből lederiválható zárt funkcióhalmazok, ezekből érdemes kiindulni

- Becslési alapot ad (bonyolultság -> erőforrások mennyisége, stb)
- Ütemezést készíteni

00:47:58

6, Architekrúra centrikus

- Példa: 48p
- Alapjaiben meghatározza a cél -> milyen architektúrán fog elkészülni
- Rétegelt szerkezet
- alacsony csatolás, laza
- robosztus, rugalmas, skálázható

00:50:57

7, Felépítés

- Mikor kell valamit csinálni? -> Időzítés
- Mit kell csinálni? -> Tevékenységek
- Mit kell készíteni? (artifact)
- Ki fogja ezt csinálni? (resource)

00:51:40

8, Lyfe cycle, életciklus, 4 fázis

- Inception phase, Kezdeti fázis: probléma megfogása, feldolgozása, végére legyen egy elképzelés a projektről
- Elaboration phase: Analízis és tervezés fázisok
- Construction, konstrukciós fázis: implementálunk,
- Transition, Átmeneti fázis: elkészítés és használat között (betanítás, éles próbák)

00:57:50

01:02:05

9,10, Ábra, 58p

01:03:33

01:04:57

01:05:49

11-13, Adott modellek, milyen diagramokat használnak

- UseCase model: UseCaseDiagram, SequenceDiagram
- Analysis model: StaticStructureDiagrams (osztály d.), SequenceDiagram
- Design model: StaticStructureDiagrams, CommunicationDiagram, StateChartDiagram, ActivityD
- Implementation m: ComponentDiagrams, SequenceDiagrams
- Deployment m: DeploymentDiagrams, SequenceDiagrams

01:06:25

14, Requirement, követelmény fázis

- Vázlatterv,
- Előzetes vizsgálati jelentés
- Meghatározni a követelményeket
- Rögzíteni azokat valamilyen szójegyzékbe
- Implementálni a prototípust
- Definiálni UseCase-eket (magas szintű és fontos)
- Vázlatos fogalmi modell
- Architektúra vázlatos képe
- Finomítani a terveket

01:08:08

15, Analysis

- Lényeges UseCase-ek meghatározása
- UseCase diagramot felépíteni
- Felépíteni egy használható fogalmi modellt
- Rendszer diagramok
- Definiálni a szerződéseket (elő- és utófeltételek)
- Állapotdiagram, az egyes szerkezeti elemekhez

01:08:57

16, Design, tervezés

- Valóságos UseCase, amik a tényleges eszközökhöz kötődnek
- Nyomtatott reportok, felhasználói felületek, storyboard (forgatókönyv, demo kezelői felület)
- Rendszer architektúrák
- Interakciós diagramok, hogyan működünk együtt
- Class diagram
- Adatbázis sémák

01:10:09

17, Implementáció

- Osztályok, interfészek definiálása,
- Metódusok, User felületek, ablakok, reportok, adatbázis sémák
- Tesztprogramok megírása

01:10:28

18, Deployment

- Mik futnak párhuzamosan, milyen konkurencia modell
- Technikai és User dokumentációk véglegesítése
- Különböző szintű tesztelések
- Tréningeket szervezni
- Support
- Installálás

01:10:05
01:12:37
19-20, Requrements ábra

- Függőségek
- Áttekintés, kik a felhasználók, célok, milyen funkciók, milyen nem-funkcionális követelmények, egyéb feltételezések, kockázatok, szótár

EA23/01:14:07 == EA24/00:00:40
EA23/01:14:27 == EA24/00:05:48
21-22, Use-Case model létrehozása

- Actor-based: Kik a szereplők, milyen felelősség?
- Event-based: Esemény alapú megközelítés, milyen események történnek a rendszerünkkel
- Formátumok, 3 dimenzió szerint
    - Magas szintű, és kiterjesztett Use-Case
    - Logikailag értelmezhető, vagy Valóságosan? 1:17p
    - Use-Case prioritás, vannak első, másodrendűek, stb
- Rendszer határok, System Boundaries
    - Ki van kívül és ki van belül

24. előadás

00:07:07

23, Use-Case készítés

- Rendszertől elvárt funkcionalitások bent vannak a követelmények között; megállapítjuk a határt
- Megpróbáljuk azonosítani az Actorokat és a UseCase-eket
- Írjunk meg minden UseCaset maga szintű formában, és kategorizáljuk őket
- Rajzoljunk UseCase diagramot
- Keressünk UseCase-ek között kapcsolatokat
- Megírjuk a legkritikusabb, legnagyobb hatással bíró UseCase-eket
- Rangsoroljuk a UseCase-eket

00:10:10

24,

- Rangsorolás
    - Időben kritikus
    - Funkcionalitás
    - Kockázatosak
    - Új technológia (oktatás, tanulás) áll mögötte

00:11:49

25, Analízis és Tervezés termékek

- Ábra, 12p

00:13:55

26, Goal of Model

- Meg kell keresni az alkalmazói model szereplőit
- Osztálydiagramokat készítünk
- Tilos implementációs részlet, logikai szinten maradunk
- Inkább túlspecifikáljunk, sok részlet
- Mapmakers principline, a Térképész elv
    - Területen alkalmazott nevet használjunk (nevezzük úgy, ahogy ott használják)
    - Hagyjuk figyelmen kívül az értelmetlen részleteket
    - Ne adjunk hozzá olyan dolgot, ami nincs ott

00:19:40

27, Fogalmi model

- Fogalmakat összegyűjtjük, kategorizáljuk
- Főnévi elemek
- Tartsuk meg a helyes fogalmakat
- Rajzoljunk fel egy fogalmi szintet, kis ábrákkal, ismerjük fel a köztük lévő kapcsolatot
- Fölösleges asszociációkat kidobni
- Adjunk hozzá attribútumokat

00:23:58

28, Kiknek lesz a modelben valamilyen elemük, kiket vegyünk fel erre a listára?

- Fizikai objektum
- Specifikációk, tervek, dolgok leírásai
- Helyek, átmenetek, történések
- **Szerepek**
- Elvont főnévi fogalmak
- Események, szabályok,
- Katalógusok (felsorolások)
- Pénzügyi dokumentumok, szerződések
- Szolgáltatások
- Manuálok, leírások

00:27:27

29,

- Hagyjuk figyelmen kívül a szükségteleneket
    - Redundánsak
    - Irreleváns, értelmetlen fogalmak
    - Körülhatárolhatatlan fogalmak
    - Attribútumok
    - Implementációs konstrukciók
- Asszociációk felfedezése
    - Asszocióciók: elemeknek tudniuk kell egymásról, pl. használni akarom
    - Fogalmi modellek megállapítása fontosabb
    - Mikor derül ki? Kommunikációs diagramnál

00:31:04

30,

- Asszociációs lista
    - Valamilyen tartalmazás, A része B-nek
    - Valaminek a tagja
    - Birtokol (autótulajdonos birtokol…)
    - Next, szomszédja
    - Használ, menedzsel, irányít, kommunikál vele
    - Tranzakció
    - Információt tárol róla, jelent
- Figyelmen kívül:
    - Irreleváns vagy implementációs
    - Redundáns vagy leszármazott asszociációk

00:34:12

31,

- Attribútumok
    - Tulajdonságot fejeznek ki
    - Nincs önálló identitásuk 35p
    - Nem érdemes első körben velük foglalkozni
    - Kétely esetén inkább azt mondjuk, hogy osztály
- Inkorrekt attribútumok elhagyása
    - Származtatott attribútumok
    - Idegen kulcsok

00:38:00

32,

- Type conformance, Típus megfeleltetés
    - Helyettesíthetőség

00:38:41

33,

- Asszociatív osztály
- Delegáció, kompozíció

00:42:20

35, System sequence diagram

- UseCasehez tartozó forgatókönyv**ek**
- Rendszeri események érik kívülről a rendszert
- Response válasz ezekre

00:43:43

00:43:50

36, 37, Contract, Szerződések

- Kinek-mi a kötelessége
- Adott UseCase-t mikor lehet használni
- Milyen eredményekkel kell szolgálnia
- Pre-Prost kondíciók

00:44:35

38, Szerződéskötés

- Kik között?
- Kiknek mi a felelőssége? -> Post kondíciók megfogalmazása
- Pre kondíciók -> mi kell ahhoz, hogy teljesülhessen

00:46:06
00:46:24
39-40, Állapot diagram

- Viselkedés állapotfüggő
- UseCasehez, Osztályokhoz, Metódusokhoz
- Állaptfüggő a rendszer, ablakok, koordonátorok, eszközök
- Mutátorok: képes megváltoztatni, hogy melyik osztályból származik

00:49:43
00:51:11
42-43, Interakciós diagramok

- UseCase realizálásához milyen együttműködés szükséges
- Szekvencia -> Kommunikációs diagram (kapcsolatok leolvasása)
- Doing: amit kell tudni
- Lnowing: amit tud

00:51:22
00:51:25
44,45, Design Class diagram, 51p

- Bele megy a részletekbe

00:52:01
00:52:04
00:53:13
00:56:10
00:58:14
46,47-50 Architektúra

- Rendszert szét kell rakni csomagokba
- Konkurrenciák kezelése (több szál, üzemezők, korlátozások, kizárások)
- Tervezési minták
- Package-ek kihelyezése processzorokra, kik hajtják végre, milyen fizikai kapcsolat
- Storage, perzisztencia: állapotra emlékeznünk kell
- Globális erőforrások kezelése: korlátosak, verseny van, ki a gazda, konkurrencia
- Ütemezés

00:59:13
00:59:18
51, 52, Interfészek, metódusok előállítása

- Algoritmusok részletei

25. Előadás

11. Diasor, Verifikáció, Validáció

00:01:54

3, Bevezetés,

- Verification, Verifikáció: Jól készítettük el a terméket? Előírásnak megfelel-e
- Validation, Validáció: A jó terméket készítettük el? Ez az-e a produktum, amire a user vágyik
- Folyamatosan kell alkalmazni a megfelelő technikákat, nem csak a végén

00:04:18

4,

- SW Review
  - Vizsgálatnak vetjük alá, vannak vizsgálati feltételek
  - Adott Specifikációnak való megfelelősséget kell ellenőrizni
- SW Testing
  - Programot futtatjük, működés közbeni viselkedését vizsgáljuk

00:07:15

00:09:08

5-6, Review Definíciók

- review: felülvizsgálat, áttekintés (anyagot elolvasunk, átnézünk), nincs végrehajtás
- review item: azaz anyag, amit vizsgálunk
- audit: szigorú dolog, cégtől független csoport végzi általában az előírt szabályok, szabványok alapján
- walkthrough, átfutás: anyag készítője tart egy rövid bemutatást az anyagról, ezt megvitatják. Elég informális
- inspection, bevizsgálás: előre kiosztanak anyagokat, ezeket kiadják ellenőröknek (reviewer), akik valamilyen jelentést tesznek erről az anyagról. pl. diplomaterv védés, formálisabb

00:11:07
00:15:07
00:19:24

7-9, Review Process

- Meeting előtt, szerepek
  - Koordinátor: szerző vagy csoportvezető; ő felelős a meetingért, következményeit betartja
  - LeadReader: adott anyag szerkezetével tisztában van
  - Scribe, jegyzőkönyv vezető: a meetinget dokumentálja,
  - Különböző képviselők
- Meeting alatt
  - 15p és 2 óra közötti
  - Nem a személy értékelésére szolgál!
  - Kulcs: hibát megtalálni, nem kijavítani
  - Ki kell jelölni egy felelős személyt, rajta lesz számonkérve a felelős
  - Mit kell csinálni: újabb felülvizsgálat, meg kell javítani, ki kell dobni
  - Milyen súlyos a probléma
  - Anyag értékelése legvégül
    - El van fogadva
    - Feltételesen fogadjuk el (vannak problémák + felelősök)
    - Major action, javítás után újabb review
- Meeting után
  - Jegyzőkönyvek kiosztása
  - Felelős személynek dolgoznia kell
  - Ha kell, új meeting

00:19:57

10, Review Report, Jegyzőkönyv

- Dátum, Csoport, Termék, Vizsgált részek, Résztvevők, Szerepük, Cél, Akciók
- Ki a felelős, mi a probléma, milyen a típusa, nehézsége

00:21:17

11, Reviewers

- Illik felkészülni
- Szerepnek megfelelően kell viselkedni
- Legyen együttműködő
- Időben oda kell menni, és végig ottlenni
- Felelősnek meg kell csinálnia a feladatát

00:24:10

12, Testing, tesztelés – működés közben vizsgáljuk a programot

- Konformancia vezérelt tesztelés: előírásoknak meg kell, hogy feleljen, a tesztekkel ezt igazolom
- Hibadetektálás: pl. lefagy a program, mert hibásan lett felvéve változó, mert valaki rosszul értelmezte a feladatot -> hibasorrend
  - error: valamilyen **emberi** tevékenység hiánya, tévedés
  - fault (bug): nem az van a **kódban**, aminek lennie kéne, kódhiba (debuggolás)
  - failure: kívülről **látható** (pl. lefagy, rossz eredményt ad), jelenség

00:29:08
00:29:44
13-14, Céljai

- Objektumok között interakció
- Komponensek jól vannak-e összerakva
- Kielégítjük-e a követelményeket
- Telepítés előtt egyéb hibák kiszűrése

00:31:50
00:39:10
15-16, Testing

- Kimutatható, hogy vannak hibák – de hiányukat nem (hogy jól működik-e)
- Fejlesztés költségeinek 30-50% tesztelésre megy
- Nehéz jó teszteseteket készíteni
- Becsülhető, hogy a SW-ben hány hiba van: szándékosan hibákat ültetnek be: hány hibát találnak, abból mennyi amit ők raktak bele, és mennyi az amúgy is meglévő (halastavas pl, 38p)
- Minél hamarabb tudjunk adni visszacsatolást

00:39:39
17, Information Flow és test, ábra, kell NAGYON

- Kapott teszteredmények összehasonlítása az elvárt eredményekkel
- Előre kell ismerni az elvárt eredményeket!

00:43:30
00:44:10
18-19, V-model

- Különböző tesztelési fázisok

00:44:24
20, Unit test

- Program teszt, amit egy programozó készít (pl. egy funkció)
- Általában nagyon kis elemek, pl. osztály
- Alacsonyszintű folyamat

00:45:01

21, Integrációs teszt

- Elemi egységekből egy rendszer -> független tesztelők végzik
- Egyes egységek együttműködését vizsgáljuk (interakció, kommunikáció)
- Top-Down:
    - Fentről rakunk össze egy programot
    - Ha van gui, azt rakjuk össze először, majd belerakjuk a funkciókat
    - Inkrementálisan tudunk tesztelni, hozzárakunk funkciókat
- Bottom-Up:
    - Legalsó szintű osztályok, metóduok, funkciók készülnek el, ezekből rakjuk össze a bonyoultabbakat
    - hiányzik az a keret, amiből az egészet tudjuk működtetni
    - kell készíteni egy teszt ágyat, amibe az alsó részeket be tudjuk tenni

00:48:25

22, System Test, Rendszer Teszt

- Független tesztelőcsapat
- Fókuszban a teljes rendszer működése
- Tudjuk-e igazolni az előírásoknak való megfelelést

00:49:09

00:51:22

23-24, Acceptance, átadással kapcsolatos teszt

- SW kész, átadás előtt tesztelünk, van formális módja, hogy miket kell elvégezni
    - Végfelhasználó tudja elvégezni
- Informális megoldások:
    - alfa: kissé ad-hoc, házon belül nekiállunk tesztelgetni
    - beta: tesztelők kiadják valamilyen körben

00:52:43

25, Tesztek típusai, FURPS

- Funkcionality, funkcionalitás
- Usability, használhatóság
- Reliability, megbízhatóság
- Performance, teljesítmény
- Supportability, támogatottság

00:53:35

26, Funkcionalítás

- Funkciók
- Security: valóban csak azokat érik el, amikre a felhasználónak lehetősége van
- Mennyiségi: nagy mennyiségű adattal támadunk, felszínre jönnek-e hibák

00:55:47

27, Használhatóság

- Emberi tényező, user doksik, helpek, varázsló, user interfész

00:59:54

28, Megbízhatóság

- Integritás, robosztusság, kibírja-e az idétlenségeket
- Struktúra, ismerem a belsejét, ennek megfelelően akarom kényszeríteni
- Stressz teszt: mindenféle gyötrés, erőforráskorlátok (mem elfogy)

01:00:54

29, Teljesítmény

- Benchmark: összehasonlításokhoz
- Contention: (stress test változat lehet) több Actor versenyez az erőforrásokért
- Load: működési limit környékén működünk
- Performance Profile: éles működés során gyűjtött profilok -> melyek a legjobban terhelt metósudok -> átírták c++-ról assemblyre, mert gyorsítani akarták

01:04:08

30, Támogatottság

- Hogyan konfigurálható
- Hogyan installálható

01:04:39

01:07:47

01:08:22

01:09:23

31-34, Teszt mértékek, metrikák

- Mikot hagyjuk abba a tesztelést?
    - Ha tudjuk, hogy a követelmények teszteléséhez hány tesztesetre van szükségünk, és tudjuk, hogy hány tesztet futtatunk le eddig, akkor ezek hányadosa megmondja, hol állunk
    - Mérhatünk a kódra, megmondjuk milyen elemeket kell tesztelnünk. Pl. Adott ciklust hányszor kell tesztelnünk -> egy teszteset.
- Riportok (eloszlások, prioritás, mióta létezik, hány ilyen hiba)
- Teljesítmény riportok

26. előadás

00:00:43

35, Test Strategy, Teszt stratégiák

- Megadja az általános megközelítést
- Milyen technikákat fogunk használni
- Siker kritériuma
- Mérőszámok definiálása
- Milyen kiegészítő programok kellenek az eredmények feldolgozásához, összevetéséhez

00:05:57

36, Test Procedure, Teszt eljárások

- Egy adott konkrét tesztelést, tesztcsoportot fognak össze
- Milyen előfeltételek, milyen inputok kellenek, mit kell csinálni, elvárt eredmények
- Tesztelés végrehajtása

00:06:29

37, Test Case, Teszt eset

- 1-1 tesztelendő elem ellenőrzése
- Mikre irányulnak, miket fednek le
- Valamennyi eset lefedése

00:07:23

00:15:52

38-39, Test eset készítés

- Tipikusan unit teszt
- White-box test (glass-box)
    - Látom, hogy mi van benne, ismerem a program szerkezetét, ebből kiindulva fogok tesztelni
    - Pl. Legyen minden utasítássor végrehajtva (pl. exception ágak) -> nem túl összetett
    - Pl. Minden döntési ágon végig kell menni, értelmesen, hurkokat bejárni (hányszor?)
    - Exhaustive testing, Kimerítő tesztelés, 11p Mese. Nem túl gyakori
- Black-box test
    - Nem ismerjük, gyakorlatilag csak interfész
    - Ekvivalencia osztályozás
        - Tudunk azonosítani bizonyos viselkedéseket, osztályozunk (17p)
        - Minden osztályt külön kipróbálunk
        - Speciális érték, robosztusság tesztelése

00:18:28

40, Objektum Orientált tesztelés

- Lazán csatolt objektumok
- Interfész szinten kezeljük
- Zártságot vizsgálni
- Öröklés okoz-e problémát
- Állapotfüggő viselkedések

00:19:36
00:22:40
00:23:58
00:24:27
00:25:20
00:25:47
00:27:10
00:28:23
00:29:32
00:29:45
00:30:06
41-51, JUnit

- Ingyenes tesztelőeszköz, hasznos
- Elvárt eredmény – Kapott eredmény összehasonlítása
- Eredmény 26p
    - Lefut
    - Fail: hiba van, „szándékolt hiba” kiértékeléskor
    - Error: olyan hiba, amire nem számítottunk, nem a tesztelt rész
- Fixture:
    - setUP()
    - assert(something);
    - assert(somethingelse);
    - tearDown();
- Test Suite, Teszt Csomag
- Ábra 51,

00:30:30
52, Dokumentációk

12. diasor, 31p, CM: Configuration Management

00:32:10
3, CM

- Túl sok definició
- Egy komplex rendszer fejlődésének követésének tudománya
- Számítógépes programok és dokumentációk együttese

00:34:05
4, Ábra, különböző nézetek:

- CM: dokumentációk, végrehajtható programok sorozata
- CR: Change Request, kész program utólagos módosítása
- Mérés, miből hány van, hibák száma
- -
- Hogyan azonosítsuk a dokumentumainkat, hogyan nevezzük el őket?
- Milyen állapotok vannak

00:36:54

5, Azonosítás

- Mik a Configuration Item –ek? (Amiket a konfiguráció kezelésbe bevonunk)
- Kommunikáció
    - Hogyan kommunikálunk? Email? Adatkezelés?
    - Címlisták – azok kapjanak értesítést, akit az érint
    - Standars memo-k, emlékeztetők
    - Wiki hasznos lehet
- Dokumentáció
    - Milyen doksik, milyen technikák
- Implementáció
    - Milyen források vannak, különböző változatok, milyen eszközzel mit állítottunk elő
-
- A Konfiguráció egy válogatás, szelekció a különböző Item-ek közül

00:43:30

6, Indentification

- Milyen doksikat tekintünk konfigurációs elemnek

00:45:34

00:50:12

7-8, Management

- Hol tároljuk a Konfigurációs Item –eket?
    - Fizikailag hol
    - Ki a gazdája, ki ment, kinek a felelőssége
    - Ki-hogy fér hozzá
- Change Request, Változás kérések kezelése
    - Ki hagyja jóvá
    - Ki implementálja, ki ellenőrzi az implementálást
- Build Management
    - Hiba felfedezése esetén kiknél kell frissíteni
    - Mik a komponensek, mik az alapanyagok
- Release
    - Mi a release, mikör történik?
- Status, Accounting
    - Hogyan nézzük meg, hogyan halad egy project
    - Hogyan ellenőrizzük, hogy az anyagok időben és a megfelelő módon jönnek létre
    - Mikor auditálunk
    - Mikor van Milestone
    - -> Project követése

00:51:12

00:51:22

10-11, Storage Configuration Items, 51p

- Verzió kontroll, Backup, Tárolás, 3rd party, Interface

00:51:54
00:52:44
00:53:43
00:53:50
00:54:56
12-16, Change Request Management

- Hogyan kezeljük a változást – Change Control Board (bizottság)
- Defect Management – Mikor, Hogyan, Ki javítsa a hibákat

00:55:42
00:58:10
01:01:07
17-19, Build Management

- Milyen alapanyagokból, hogyan rakjuk össze
    - Források, külső komponensek, bináris komponensek, eszközök
- Anyag elnevezése (doksi, doksi2, doksi2-jav, doksi2-végleges, doksi3…)
    - Kell egy rendszer
- Baseline (snapshot):
    - Egy alap kép
    - Ezen vannak inkrementumok
    - BaseLine + AdottInkrementum = Egy változat

01:01:13
01:02:20
20-21, Release Management

- Definíciók
    - Version, verzió: Funkcionálisan is különböző
    - Variant, variáns: Funkcionálisan ekvivalens, Nem-funkcionálisan különbözik
    - Release: Új kibocsátás
    - Revision: Verzió + Variáns, nem szükségképp release
- Release kiegészítések
    - Notes
    - Külső és belső elnevezés
- Számozás
    - Elágazásnál problémás

01:03:12
22, Tools, eszközök

- CVS, Subversion, Bugzilla, Ant, stb

01:08:07
01:04:55
01:05:04
01:05:06
01:05:11
01:05:12
01:05:16
23-29, Verió kontroll

- Check out, check in
- Konkurrens check in megoldása

01:05:20
01:05:54
30-31, CM Plan, konfiguráció menedzsment tervezés

- Milyen eszköz?

FELADATOK, gyakorlás, 1:06p

- 1:23, C függ Xtől és X függ Ctől