Mi a projekt?

- Jól definiált cél
- Előre rögzített határidő és részhatáridők
- Meghatározott erőforrások (költségvetés)
- Tervezett tevékenység
- Alapvető jellemzőiben egyedi

DIN 69901:

„Ein Projekt ist ein Vorhaben, bei dem innerhalb einer definierten Zeitspanne ein definiertes Ziel erreicht werden soll, und das sich dadurch auszeichnet, dass es im Wesentlichen ein einmaliges Vorhaben ist."

Projekt méretek

Megnevezés	Résztvevők száma	Időtartam	Program nagysága	Példa
Mini projekt	1	1-4 hét	500 LOC	egyszerű Pascal pr.
Kis projekt	1	11-12 hónap	1-5 KLOC	féléves feladat
Közepes projekt	2-5	1-2 év	5-50 KLOC	Compiler
Nagy projekt	5-100	2-3 év	50-100 KLOC	Op. rend-szer
Szuper projekt	100-1000	3-5 év	1 MLOC	Nagy Op. rendszer
Mega projekt	2000-5000	5-10 év	1-10 MLOC	SDI

4.1 projekt tervezése

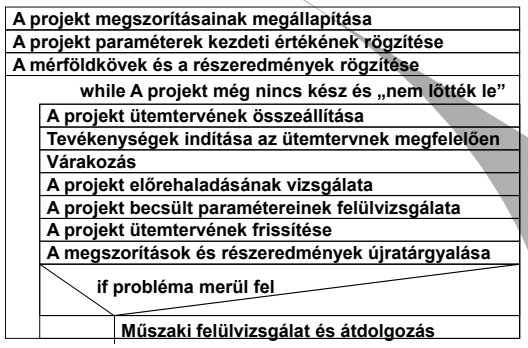
A szoftver projekt előkészítése:

- A megrendelő és a fejlesztő közösen meghatározzák:
  - a projekt céljait
  - a megvalósítandó rendszer fő funkcióit
  - kvantitatív mérőszámokat a funkciók jellemzésére
- Alternatív megoldási módokban állapodnak meg a tervezési tér növelése érdekében
- költség, határidő, erőforrás korlátok tisztázása

A projekt terv típusai

- Minőségi terv (a projektben használandó minőségi eljárásokat és szabványokat tartalmazza)
- Validációs terv (a rendszer validációhoz szükséges megközelítési módot, erőforrásokat és ütemterveket határozza meg)
- Konfigurációkezelési terv (a konfiguráció kezeléséhez szükséges eljárásokat és szerkezeteket tartalmazza)
- Karbantartási terv (a rendszer karbantartásához szükséges követelményeket, költségeket tartalmazza)
- Munkaerő-fejlesztési terv (a projektteam tagjainak szakmai fejlődését tartalmazza)

A projektterv készítésének folyamata



A projektterv felépítése (szakaszai)

1., Bevezetés Meghatározza a projekt célját, alapvető jellemzőit, a menedzselés megszorításait (erőforráskorlátok, időkorlátok)

2., Projekt szervezet Megadja a projekthez rendelt team vagy teamek összetételét, a tagok szerepköreit, feladataikat, felelősségüket

3., Kockázatelemzés Számba veszi a lehetséges projekt kockázatokat, elkerülésükhöz szükséges stratégiákat, bekövetkezésük esetén szükséges tevékenységeket.

4., Hardver és szoftver erőforrás követelmények Rögzíti, hogy a fejlesztéshez pontosan milyen hardver és szoftver eszközökre van szükség. Amennyiben ezeket be kell szerezni, úgy tartalmazza a költségbecslést is.

5., Munka felosztása Tartalmazza a projekt tevékenységekre bontását az egyes tevékenységekhez tartozó részeredményekkel és mérföldkövekkel együtt.

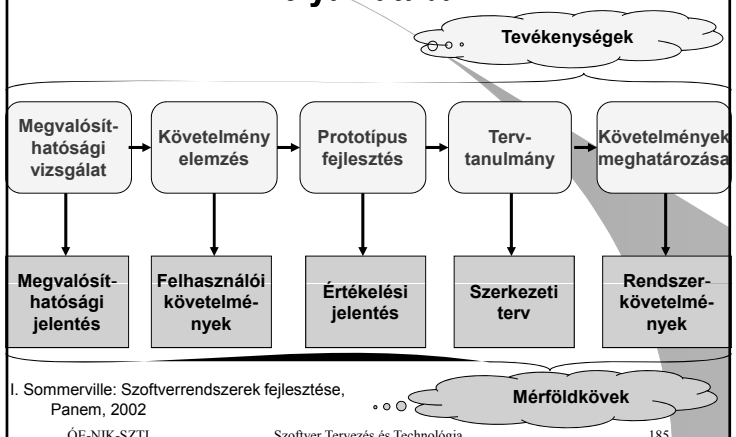
6., Projekt ütemterve Megadja az egyes tevékenységek közötti függőségeket, a mérföldkövek eléréséhez szükséges becsült időt, valamint a tevékenységekhez rendelt becsült emberi erőforrásokat.

7., Figyelési és jelentéskészítési mechanizmusok (projekt követés és ellenőrzés) Meghatározza a menedzser által elkészítendő jelentéseket, azok leadási idejét, valamint az alkalmazandó projekt követési, ellenőrzési technikákat).

Mérföldkövek a projektben

- A mérföldkövek alkalmazása egy eszköz a projekt követésére.
- A mérföldkövek tipikusan a projekt egyes szakaszainak végét jelzik (esetenként ritkább, máskor sűrűbb bontásban).
- A mérföldkövekhez jelentések (report) tartoznak
- A mérföldkövek kapcsolódhatnak belső részeredményekhez, vagy külső, leszállításra kerülő részeredményekhez (pl: specifikáció, stb.)

Mérföldkövek a követelményspecifikáció folyamatában



I. Sommerville: Szoftverrendszerek fejlesztése, Panem, 2002

4.2 Projekt ütemezése

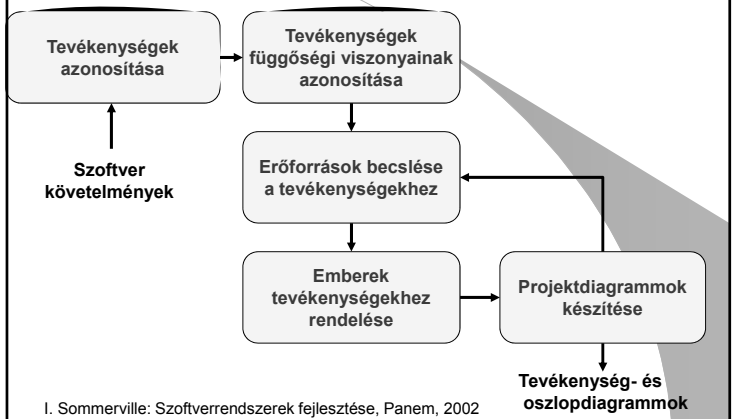
Az ütemezéshez ismerni kell:

- A tevékenység egységeket
- Időigényüket
- Erőforrásigényüket
- Függőségeiket
- A folyamatok, részfolyamatok sorrendiségeit
- A párhuzamosítható tevékenységeket
- Egyéb korlátokat, illetve peremfeltételeket

Ökölszabályok:

- Ha a tevékenység 8-10 hetet meghaladna, szét kell bontani
- Mindig tervezzünk tartalékot a nem várt problémákra (30-50%)

A projekt ütemezés folyamata



I. Sommerville: Szoftverrendszerek fejlesztése, Panem, 2002

A projekt ütemterv ábrázolásának lehetőségei

Oszlopdiagramok (Tevékenység diagramok)

- Tartalmazza az adott tevékenység felelősét (munkavégzőjét) és a tevékenységek kezdési és befejezési időpontjait

Tevékenységhálók

- A tevékenységek közötti függőségeket ábrázolja

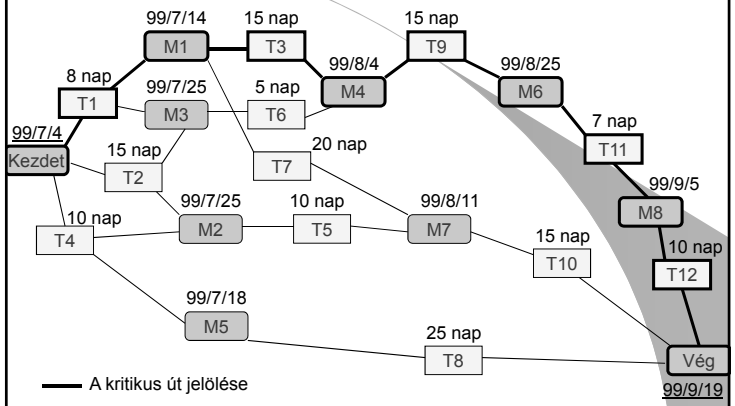
Példa a projekt ütemtervre

Adott egy projekt tevékenységhalmaza: **(T-tevékenység, M-mérföldkő)**

Tevékenység	Időtartam (nap)	Függőségek
T1	8	
T2	15	
T3	15	T1(M1)
T4	10	
T5	10	T2, T4 (M2)
T6	5	T1, T2 (M3)
T7	20	T1 (M1)
T8	25	T4 (M5)
T9	15	T3, T6 (M4)
T10	15	T5, T7 (M7)
T11	7	T9 (M6)
T12	10	T11 (M8)

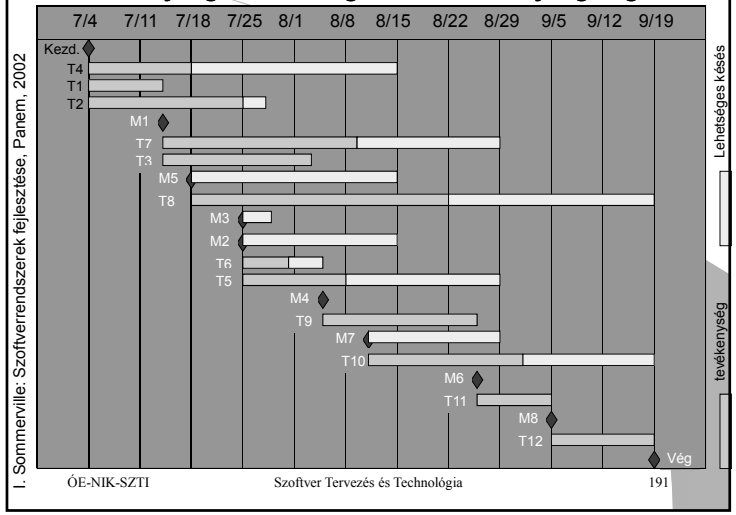
I. Sommerville: Szoftverrendszerek fejlesztése, Panem, 2002

A tevékenységhalmazból generált tevékenységháló:

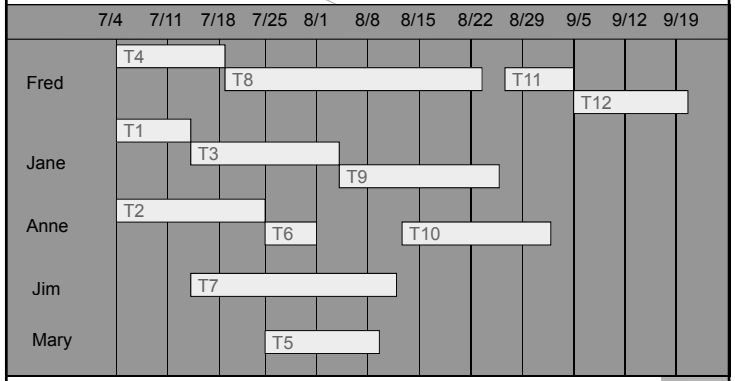


I. Sommerville: Szoftverrendszerek fejlesztése, Panem, 2002

A tevékenységhalmazból generált tevékenységdiagram

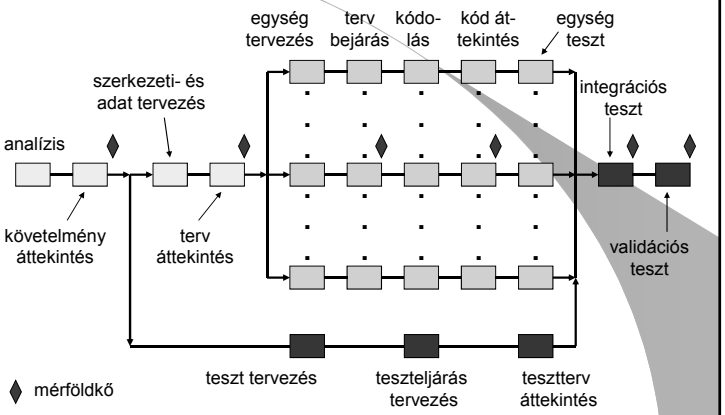


A munkatársak lekötöttsége az idő függvényében



I. Sommerville: Szoftverrendszerek fejlesztése, Panem, 2002

Egy hagyományos szoftver projekt ütemezése



Az emberi erőforrások ütemezésének kérdése

példa:

Adott 4 fejlesztő 5KLOC/év kapacitással

Ha egy csoportban dolgoznak -> 20 KLOC/év !!!

Ebből lejön azonban a kommunikációs veszteség, ami 250 LOC/év !!

(4x5000) - (6x250) = 18500 4625 LOC/fő

Ha hozzáadunk még két embert a csoporthoz:

(6x5000) - (15x250) = 26250 4375 LOC/fő

Az összefüggés nem lineáris! Nem érdemes csoportban dolgozni !?

Érdemes mert:

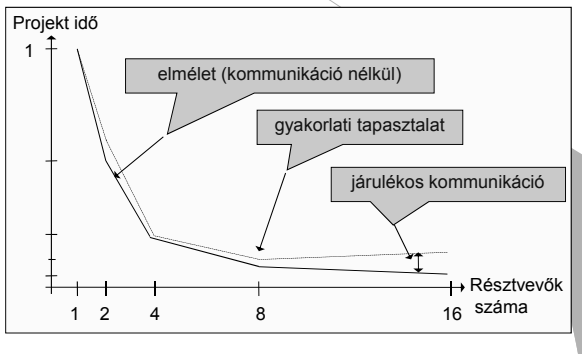
- **sokszor a projekt méretei miatt nem is lehet másként megoldani**
- **jobb a minőség és a megbízhatóság mint a „magányos farkasok” esetében**
- **nem fog mindenki, mindenkivel „beszélgetni”**

Brooks törvénye

Ha egy elcsúszott projekthez plusz embert rendelsz, az tovább fog csúszni !

Prof. Dr. F. P. Brooks
(az IBM-360 és az OS/360 Projektmenedzsere)

A projektidő alakulása a résztvevők számának függvényében



Ütemezési módszerek:

- PERT (Program Evaluation and Review Technique)
- CPM (Critical Path Method)

Lehetővé teszik:

- kritikus út meghatározását
- a feladatok legvalószínűbb idő-ütemezését
- idő-ablakok meghatározását (legkorábbi, legkésőbbi befejezés / kezdés, indítási idő-intervallumok)

4.3 Projekt mérése

A szoftver mérése a szoftver mint termék, illetve a szoftver készítési folyamat szignifikáns jellemzőinek számszerűsítésével foglalkozik (szoftver metrikák).

A szoftver metrikák lehetnek:

- Vezérlési metrikák (a szoftver folyamattal kapcsolatosak, pl: hibák száma, ráfordítások, idők)
- Prediktor metrikák (a szoftver termékkel kapcsolatosak, pl: a modul ciklomatikus komplexitása)

A vezetői döntéshozatalt mindkét típus befolyásolhatja

A mérés célja

A mérések több célt szolgálhatnak:

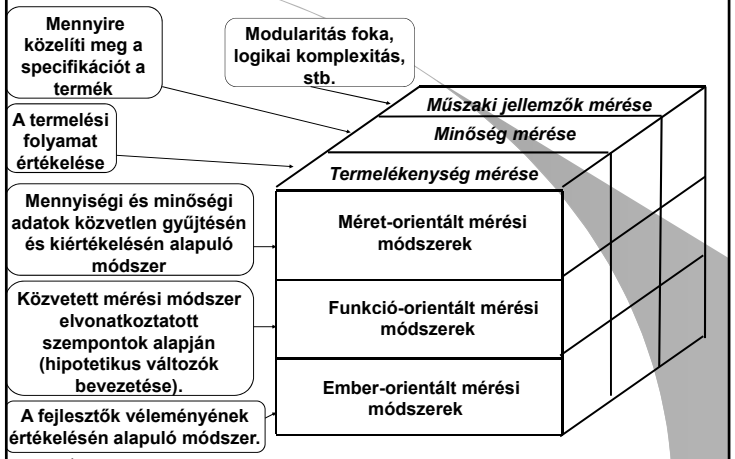
A termék mérése esetén

- a termék minőségének értékelése

A folyamat mérése esetén

- az alkalmazott módszerek, eszközök hatékonyságának értékelése
- a fejlesztők termelékenységének elemzése
- a következő projektek becslési fázisához bázis-értékek kialakítása, a meglévő értékek javítása

A mérések kategorizálása



Méret-orientált mérési módszer

Közvetlen mérési módszer, a mérés tárgya az elkészült termék és a készítés folyamata.

Projekt neve	Ráfordítás [ember/év]	Költség [ezer$]	KLOC	Doku. [oldal]	Hibák száma	Részt-vevők száma
A	30	141	11	500	19	4
B	70	200	30	1500	61	8
C	40	155	21	700	5	6
.	.	.	.	.	.	.
.	.	.	.	.	.	.

$$\text{Termelékenység} = \frac{\text{KLOC}}{\text{Ráfordítás}}$$

$$\text{Minőség} = \frac{\text{Ráfordítás}}{\text{Hibákszáma}}$$

$$\text{Fajlagos költség} = \frac{\text{Költség}}{\text{KLOC}}$$

$$\text{Fajlagos doku} = \frac{\text{Doku}}{\text{KLOC}}$$

A módszer jellemzői:

- mennyiségi mérőszámokon alapuló egyszerű módszer (oldalak száma, sorok száma, emberek száma, stb.)
- nem igényel tapasztalatot, szaktudást a kiértékelés
- nem veszi figyelembe a komplexitást, a szoftver sajátosságait
- csak azonos típusú projektek összehasonlítására, becslésére használhatók fel a mérőszámok

Funkció-orientált mérési módszer

A „LOC számolása" helyett a szoftver funkcionalitása kerül előtérbe. A módszer alkalmas a komplexitás kezelésére.

Albrecht: Function Point Method (1979)

A Function Point (FP) valamilyen számszerűsíthető jellemzőből származtatható.

Tipikus jellemzők, melyeket a módszer alapul vesz:

- felhasználói inputok száma (adat)
- felhasználói outputok száma (adat)
- felhasználói interakciók száma (vezérlés)
- fájlok száma
- külső interfészek száma (pl.: hálózat)

A módszer lépései:

1., A számértékeket tartalmazó táblázat kitöltése.

Jellemzők	Szám	Súlyfaktor			Érték
		Egyszerű	Átlagos	Komplex	
Felhasználói inputok száma		3	4	6	
Felhasználói outputok száma		4	5	7	
Felhasználói interakciók száma		3	4	6	
Fájlok száma		7	10	15	
Külső interfészek száma		5	7	11	
Teljes összeg					

2., A komplexitás értékelése

Módszer:
14 kérdésre adandó 0..5 értékű, osztályzat jellegű válasz összege adja a SUM(Fi), a komplexitást kifejező tényezőt. (max. érték = 70)

Példák a komplexitást kifejező tényezőkre:
- **Mennyire szükséges üzembiztos mentés?**
- **Mennyire szükséges az osztott feldolgozás?**
- **Mennyire legyen a kód újrafelhasználható?**

(lásd: Jones, C.: Programming Productivity, McGraw-Hill, 1986.)

3., A táblázat és a válaszok alapján FP meghatározása

FP= [Teljes összeg] x [0.65 + 0.01 x SUM(Fi)]
A komplexitástól függően a [teljes összeg] 0.65 ... 1.35 -tel módosulhat.

$$\text{Termelékenység} = \frac{\text{FP}}{\text{Ráfordítás}}$$

$$\text{Minőség} = \frac{\text{Ráfordítás}}{\text{Hibák száma}}$$

$$\text{Fajlagos költség} = \frac{\text{Költség}}{\text{FP}}$$

$$\text{Fajlagos doku} = \frac{\text{Doku}}{\text{FP}}$$

Jellemzők:
- **adat alapú, ami azért előnyös, mert az adatok a fejlesztés korai szakaszában már ismertek, míg a LOC csak a végén**
- **programozási nyelv független**
- **részben szubjektív**
- **a LOC alapúval szemben az értékelés szaktudást, tapasztalatot igényel**

Objektum-orientált mérési módszer

Az OO projektek esetén is alkalmazható pl. az FP alapú mérési módszer.

Pontosabb mérési módszerre tett javaslatot: Lorenz, M., Kidd, J.: Object-oriented Software Metrics, Prentice-Hall, 1994

Javasolt paraméterek:

- A szcenáriók száma (a felhasználó és az alkalmazás közti interakciókat írják le – erősen korrelál az alkalmazás méretével)
- Kulcs osztályok száma (az OO analízis korai szakaszában már definiált, erősen független osztályok – korrelál a számuk a rendszer kifejlesztéséhez szükséges ráfordítással, a szükséges tesztesetek számával

- Támogató osztályok száma (nem tartoznak közvetlenül a rendszer probléma specifikus funkcióihoz pl. UI osztályok, szerviz osztályok – számuk jó indikációt ad a rendszer kifejlesztéséhez szükséges ráfordításra, és a reuse lehetőségére
- A kulcs osztályokat támogató osztályok átlagos száma GUI alkalmazás esetén a támogató/kulcs osztály arány 2-3, nem GUI alkalmazás esetén 1-2 (mivel a kulcs osztályok már az analízis korai fázisában ismertek, így következtethetünk a teljes alkalmazás méretére
- Alrendszerek száma – a rendszer komplexitásának jó indikátora

Use-Case-orientált mérési módszer

Potenciális előnyök a use-case-ek metrikákban történő felhasználása esetén:

- A szoftver folyamat korai szakaszában rendelkezésre állnak
- A rendszer funkcionalitását (legalább a felhasználó oldaláról) jól jellemzik
- Programozási nyelvtől, környezettől függetlenek
- Számuk arányos a rendszer nagyságával és a szükséges tesztesetek számával

Problémák, ami miatt még nincs széleskörű használat:

- A use-casek mérete, bonyolultsága
  - nem szabványosított,
  - erősen függ az alkalmazott absztrakciós szinttől
- Nehéz megadni a use-case / ráfordítást

Coming soon ... ? ... !

4.4 Projekt becslése

A becslés tárgya:

- ráfordítás
- költség
- időtartam

A becslést nehezítő tényezők:

- a projekt nagysága, komplexitása
- Időkorlát
- erőforráskorlát
  - emberi (mennyiségi, minőségi)
  - hardver (speciális egységek: analizátor, stb.)
  - szoftver eszközök

A becslési technikák csoportosítása:

- Dekompozíción alapuló technikák
  - LOC vagy FP orientált
  - ráfordítás becslés
- Tapasztalati becslési modellek

Dekompozíción alapuló technikák

Mindegyik technika alapja a bonyolult, nagy rendszer szétbontása kezelhető részekre. A kiindulás minden esetben a feladat vázlatos, lehetőleg számszerűsített leírása (scope).

I. LOC vagy FP orientált technikák

lépései:

A., Meghatározzuk az egyes modulok LOC-ját
(három értéket adunk meg: optimista, valószínű, pesszimista)

B., Képezünk modulonként egy súlyozott átlagot

$$E = \frac{\text{optimista} + 4 \times \text{valószínű} + \text{pesszimista}}{6}$$

C., Kitöltjük a táblázatot

Modul neve	LOC			Számított érték
	optimista	valószínű	pesszimista	
A	1800	2000	2500	2050
B	700	1000	1200	983
C	5000	6000	8000	6166
D	900	1200	1700	1233
Összeg				10432

D., Meghatározzuk a becsült költség és ráfordítás értékeket

Modul neve	Számított LOC	Becsült Ft/LOC	Becsült LOC/hó	Költség [Ft]	Ráfordítás [ember-hó]
A	2050	800	270	1.640.000	7.6
B	983	1200	75	1.179.600	13.1
C	6166	700	300	4.316.200	20.5
D	1233	1500	110	1.849.500	11.2
Összeg				8.985.300	52.4

Az eltérő súlyok a különböző modultípusok komplexitás-különbségeit fejezik ki. Az értékek az előző projektek méréseiből származnak és projektenként finomíthatók.

A módszer jellemzői:

- A LOC orientált részletesebb, finomabb felbontást igényel,
- Az FP orientált nagyobb, komplexebb egységeket képes kezelni.
- A módszer nagy tapasztalatot, jó „érzéket” kíván, hiszen a projekt elején tényszámok nincsenek,
- A becslésnél nagy a bizonytalanság.

II. Ráfordítás becslés

Módszer:

Egy „rutinos öreg róka” megbecsüli az egyes tevékenységek ember-hónap ráfordítás igényeit minden modul esetében.

A kiindulás ennél a módszernél is a projekt vázlatos leírása.

Modul neve	Tevékenységek ráfordításai [ember-hó]				Összeg
	Analízis	Tervezés	Kódolás	Tesztelés	
A	1.5	2.5	0.5	3.5	8
B	2	5	1	4	12
C	3	8	1	8	20
D	2	4	0.5	3.5	10
Összeg	8.5	18.5	3	19	50
Fajlagos ktg. [eFt]	300	200	60	120	
Költség [eFt]	2.550	3.900	180	2.280	8.910

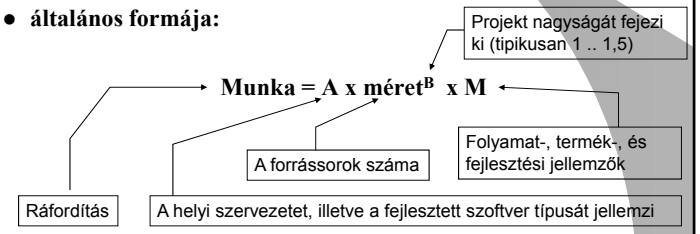
A módszer jellemzői:

- A különböző komplexitású modulok különböző súllyal szerepelnek
- Az egyes tevékenységek súlyozása eltérő
- Igen nagy rutint igényel (indirekt becslés), jól kell ismerni a rendszer típusát és a fejlesztőcsapatot
- Jól használható az előző módszer kontroljaként

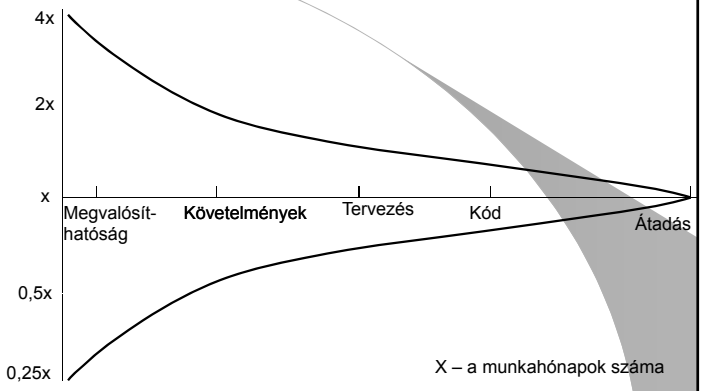
Tapasztalati becslési modellek

Jellemzőik:

- Általában méretből indul ki
- Tartalmaz exponenciális komponenst a méret – komplexitás viszony kifejezésére
- Az algoritmus miatt nem okvetlenül pontos a becslés, nagyban függ a paraméterek jó becslésétől
- általános formája:



A becslési bizonytalanság



A COCOMO modell

COCOMO = Constructive Cost Model
Barry Boehm: Software Engineering Economics, Prentice Hall, 1981

Jellemzői:

- **Jól dokumentált**
- **Szabadon hozzáférhető**
- **Széles körben használt**
- **Hosszú ideje használt, több verziót fejlesztettek ki, sok tapasztalat halmozódott fel**

A COCOMO 81

Jellemzői:

- **Feltételezi a vízesés modell alapú fejlesztést**
- **A szoftver túlnyomó többsége még nem létezik**

A modell szintjei:

Model-1 (alap COCOMO)
Egyszerű, egyváltozós modell, a sorok számán alapszik.

Model-2 (középszintű COCOMO)
A LOC-on kívül szubjektív költségtényezők határozzák meg a ráfordítást.

Model-3 (fejlett COCOMO)
A Model-2 jellemzőit a szoftver technológia fázisaira (analízis, tervezés, stb.) szétbontva határozza meg.

A COCOMO projektosztályai:

Kis méretű projekt
Relatív kis méretű projekt, egyszerű feladat, kis team, jól felkészült szakemberek

Közepes méretű projekt
Közepes bonyolultságú feladat, közepes méretű projekt, vegyes összetételű team.

Nagy méretű projekt
Bonyolult hardver körülmények között, kényszer-feltételek mellett fejlesztendő projektek (pl.: légi -irányítási rendszer)

COCOMO Model-1

$$E = a_b \times (KLOC)^{bb} \qquad D = c_b \times (E)^{db}$$

ahol: E - ráfordítás [ember-hónap]
D - a projekt időtartama [naptári hónap]

Projekt mérete	a_b	b_b	c_b	d_b
kis	2.4	1.05	2.5	0.38
közepes	3.0	1.12	2.5	0.35
nagy	3.6	1.20	2.5	0.32

COCOMO Model-2

$$E = ai \times (KLOC)^{bi} \times EAF$$

ahol: E - ráfordítás [ember-hónap]
EAF - költség befolyásoló tényezők

Költség befolyásoló tényezők:

Termék jellemzők
- Igényelt szoftver megbízhatóság
- Felhasznált adatbázis mérete
- A termék bonyolultsága

Hardver jellemzők
- Run-time kényszerfeltételek
- Memória kényszerfeltételek
- Virtuális gép kényszerfeltételek
- Igényelt válaszidő korlátok

Személyi jellemzők
- Analizáló szakember kapacitás igény
- Szoftver technológus szakember kapacitás igény
- Alkalmazói gyakorlat
- Virtuális gép gyakorlat
- Programozási nyelv gyakorlat

Projekt jellemzők
- Szoftver tool-ok használata
- Szoftver technológiai módszerek használata
- Igényelt fejlesztési ütemezés

Kiértékelés:
1., mind a 15 kérdésre egy 6 fokozatú skála segítségével
2., táblázat alapján EAF megállapítása (tipikusan: 0.9-1.4)

Együtthatók:

Projekt mérete	a_i	b_i
kis	3.2	1.05
közepes	3.0	1.12
nagy	2.8	1.20

A COCOMO 2

Jellemzői:

- **Elismeri a szoftverfejlesztés különböző szemléleteit (prototípus készítés, komponens alapú fejlesztés, 4GL, stb)**
- **A modell szintjei itt már nem a becslések részletesebbé válását jelentik**
- **A szintek a szoftverfejlesztési folyamat tevékenységeihez kapcsolódnak**

A COCOMO 2 modell azonosított szintjei:

- **Korai prototípuskészítési szint (a méretet objektumokkal becsli, a ráfordítás mértéke méret/termelékenység képlettel számítható ki.)**
- **Korai tervezési szint (a rendszerkövetelményekhez valamilyen kezdeti tervezetet készítünk. A becslések funkciópontokon alapulnak, amelyet a forráskód sorainak számává alakítunk át. A képlet alakja marad az egységes formátumú, a szorzóknak egy egyszerű halmaza kapcsolódik hozzá)**
- **Posztarchitekturális szint (a rendszer architekturájának elkészülte után már aránylag pontos becslés tehető a szoftver méretére. Itt a becslések már több szorzót tartalmaznak, ezekkel a személyi kapacitások, a termék és a projekt jellemzői fejezhetők ki.)**

Részletesebben lásd a Sommerville könyvben!

I. Sommerville: Szoftverrendszerek fejlesztése, Panem, 2002

4.5 Kockázat menedzsment, Kockázatkezelés

Mi a kockázat?

Hogyan jellemezhetjük?

Érdemes-e vele foglalkozni?

Mi a „haszna” és az „ára” a kockázat kezelésnek?

> „If you don’t actively attack the risks, they will actively attack you”
>
> Tom Gilb

A kockázati kategóriák csoportosítása (kockázat típusok):

- **Projektkockázat (kihatással van a teljes projektre pl: erőforrások rendelkezésre állása, ütemterv betartása, költségvetés tartása, kollégák kiválnak a projektből, vezetőségváltás)**
- **Termékkockázatok (kihatással van a fejlesztés alatt álló termék minőségére, teljesítményére, pl.: fejlesztőeszköz elégtelensége, elvárások jelentős változása)**
- **Üzleti kockázatok (a szoftver fejlesztését végző szervezetre hat ki, pl: konkurens termék kerül a piacra, megváltozik a cég stratégiája, technológiaváltás)**

A kockázatkezelés fázisai:

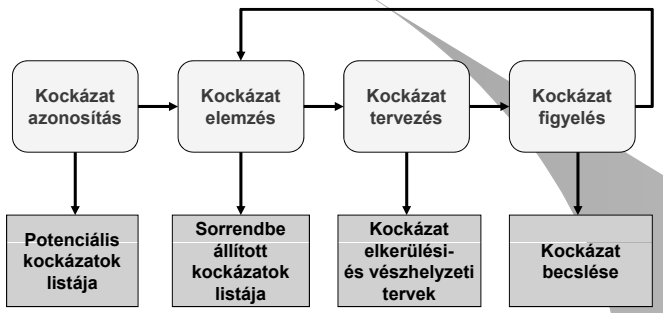
1., Kockázat azonosítás (a lehetséges kockázatok közül meghatározza a valószínű kockázatokat)

2., Kockázat elemzés (megbecsüli a kockázat bekövetkezésének valószínűségét és kihatását)

3., Kockázat tervezés (intézkedési tervek készítése a kockázat csökkentésére, illetve a teendőkre a bekövetkezésének esetére)

4., Kockázat figyelés (állandó monitorozás és terv revízió)

A kockázatkezelés folyamata és dokumentumai



I. Sommerville: Szoftverrendszerek fejlesztése, Panem, 2002

Kockázat azonosítás

Cél: kiválasztani a lehetséges kockázatok közül a potenciális kockázatokat.

Módszer: checklist alkalmazása

Eredmény: Potenciális kockázatok listája

Példák kockázatokra:

Műszaki-, technológiai-, emberi-, szervezeti-, hardver-, eszköz-, követelmény-, becslési kockázat

Kockázat elemzés

cél: a potenciális kockázatok vizsgálata két szempontból:
- bekövetkezésének valószínűsége
- következményei

A kockázat elemzés lépései:

- A bekövetkezés valószínűségének számszerűsítése (alkalmazható skálák: bináris, minőségi, mennyiségi)
- A bekövetkezéskor fellépő következmények leírása
- A kockázat kihatásának becslése
- A kockázat elemzés pontosságának megfogalmazása

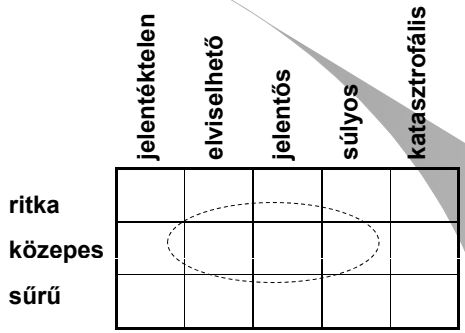
A kihatás mértéke lehet:

- jelentéktelen,
- elviselhető,
- jelentős,
- súlyos,
- katasztrofális

A bekövetkezés gyakorisága lehet:

- ritka
- közepes
- sűrű

Melyikre lehet felkészülni?



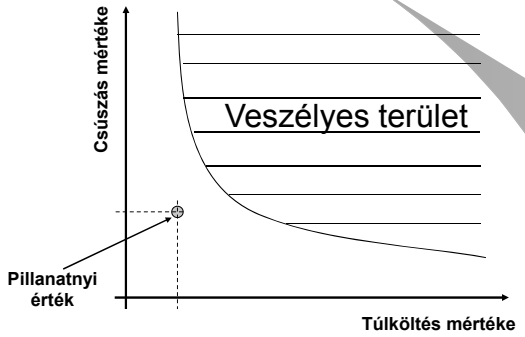
Kockázat tervezés

Kidolgozandó a kockázat elkerüléséhez szükséges stratégiák, és a bekövetkezésekor elvégzendő teendőket tartalmazó tervek.

Kockázat elkerülési stratégiák: azon tevékenységeket és megszorításokat tartalmazza, melyek segítségével csökkenteni igyekszünk a kockázat bekövetkezésének valószínűségét. (pl: bizonytalan elemek lecserélése megbízhatóra)

Kockázat minimalizációs stratégiák: célja a kockázat bekövetkezésekor, annak kihatásának csökkentése (pl: redundanciák, átfedések, tartalékok beépítése)

Vészhelyzeti tervek: a kockázati veszélyeztetettség fokára vonatkozó referencia szinteket, a kockázat bekövetkezésekor szükséges teendőket, a helyzet megoldásához tartalékolandó erőforrások leírását tartalmazza.



Kockázat figyelés, (követés)

A teljes projekt során szükséges tevékenységek:

- az azonosított kockázatok bekövetkezési valószínűségének figyelése
- A kockázat elkerülési terv végrehajtásának ellenőrzése
- Szükség esetén a tervek módosítása
- Kockázati esemény bekövetkezése esetén vészhelyzeti terv végrehajtása, hatás ellenőrzése

Nagy projektek esetén az azonosított kockázatok száma: 30-40 (felkészülés, erőforrás tartalékolás drága)

Általános szabály:

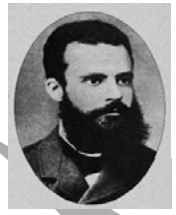
A kockázatmenedzsment költsége ideális esetben a projekt költségvetésének 3-5 százaléka.

Ha a kockázat menedzsment költsége eléri a projektköltség 15 százalékát, akkor meg kell gondolni, hogy szükséges-e egyáltalán (a kockázat menedzsment, vagy a projekt maga)

Pareto 80/20-as szabálya

Alkalmazása a

- A hagyományos Projektmenedzsmentben
- A modern projektmenedzsmentben
- A bürokráciában



Vilfredo Pareto (1848-1923)
olasz közgazdász,
statisztikus, egyetemi tanár

A Pareto 80/20-as szabály alkalmazása a kockázat menedzsmentre

A projekt alatt bekövetkező kockázatok 80%-át lefedi az előzetesen becsült kockázat 20%-a.

4.6 Projekt követés és ellenőrzés

Célja: a projekt előrehaladásának figyelemmel kísérése a sikeres befejezés érdekében.

A projekt követés alapja a tervezés során elkészült ütemezés és az abban elhelyezett mérföldkövek.

Módszerei:

- rendszeres „status report meeting” (pl. minden hétfőn)
- a projektbe beiktatott review-ok kiértékelése
- „mérföldkövek” elhelyezése a projektben
- a feladatok indítási idejének ellenőrzése, összehasonlítás a tervezett értékkel
- informális meeting-ek a „local guru-kkal”

Ezen módszerek egymással párhuzamosan is használatosak.

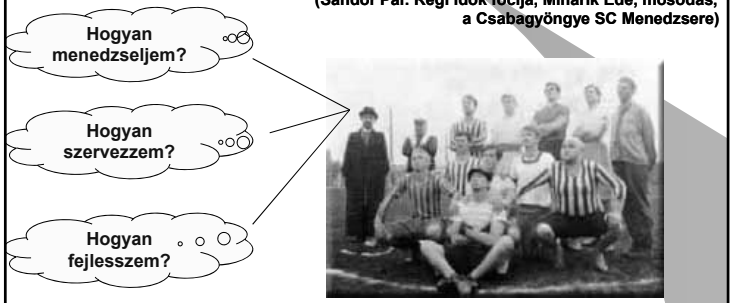
Irányítás:

- ha minden rendben, akkor minimális tevékenység
- ha probléma merül fel:
  - helyzetfelmérés, erőforrás felmérés
  - intézkedési terv kidolgozás
  - esetleges projekt átütemezés

4.7 Team-szervezési megoldások

„Kell egy csapat !”

(Sándor Pál: Régi idők focija, Minarik Ede, mosodás, a Csabagyöngye SC Menedzsere)



Géniusz elv

Jellemzői:

- A programozás „hőskorára” jellemző
- Az átlagos programozók egy kiváló képességű, vagy autokrata egyéniségű programozó (vezető) keze alá dolgoztak
- Nem volt tervezés (feladat szétbontás, integrálás, stb.) mindent a vezető döntött el
- Az ő képességei határozták meg a program minőségét
- Fejletlen információs rendszer (gyakorlatilag nincs)
- A dokumentáció gyakran utólag készül

Homogén csoport elv

Jellemzői

- A programozás egy későbbi fázisára jellemző
- Az elv, hogy nem kell vezető a csoportba (demokratikus működés: a csoport minden tagja egyenrangú)
- Döntéseket közösen hoztak, a modularizálást közösen végezték el
- A programozás egyénileg történt, az integrálásnál jött az „integrációs BUMM”
- A modulok „személyesek” a teljes program személytelen
- Az információs rendszer nem fejlett, információ áramlás nincs

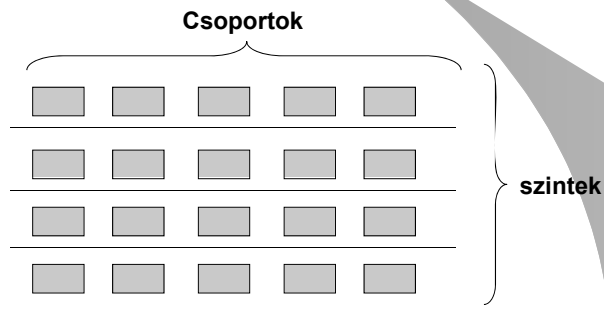
Horizontális szervezési elv

Jellemzői

- A programozási, programtervezési módszerek fejlődésével kialakult a specializálódás, szakosodás (külön tervező szakember, külön programozó szakember, stb.)
- A programozás egyes részfeladataira tehát specializált csoportok rendszere alakult ki:
  - rendszertervezők, rendszerszervezők
  - programozók *
  - kódolók *
  - tesztelők

A *-gal jelölt csoportok minden munkánál, illetve szoftverháznál megtalálhatók, míg a többi esetleges

- A feladatot mindegyik csoport a saját szempontjai szerint feldolgozza és továbbítja az alatta lévő szintekre
- Fejlett információs rendszer és információ továbbítás
- Az egyes csoportok közötti kapcsolat pontosan szabályozott

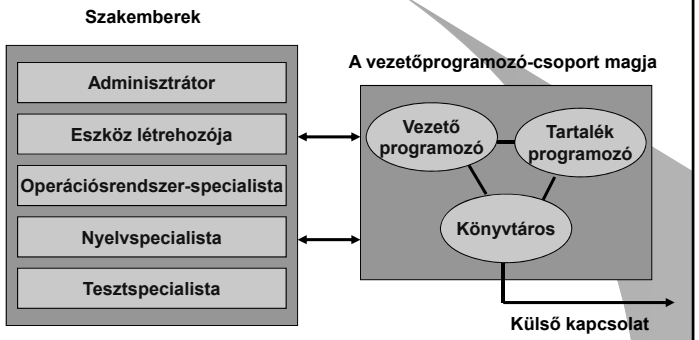


Vezetőprogramozó-csoport elv

Jellemzők:

- Az igen jó képességű programozók hatékonyságának növelése érdekében magas szintű kiszolgálásuk
- A csoportot egy három tagú mag alkotja, amelyhez időlegesen kapcsolódnak egyéb szakemberek
- Dinamikus, a feladathoz jól illeszkedő struktúra

A vezetőprogramozó-csoport felépítése



I. Sommerville: Szoftverrendszerek fejlesztése, Panem, 2002

Nagyvállalati projekt menedzsment
EPM – Enterprise Project Management

Portfolió menedzsment:

A portfolió menedzsment a projektek azonosításának, fontossági sorrendbe állításának és beruházásának állandó folyamata, amely a vállalati stratégiával összehangoltan történik. Az üzleti feltételek és a költségvetések változásakor az „agilis" vállalatok proaktívan értékelik és kiigazítják a portfóliót úgy, hogy a kockázattűrésükkel összhangban optimális megtérülést érjenek el.

A portfoliójukat bölcsen kezelő vállalatok megalapozott kompromisszumokat kötnek a projektek között, és erőforrásaikat megfelelően átirányítva biztosítják, hogy csak olyan tevékenységekre fordítsanak időt, amelyek a végcélhoz szükségesek.

Erőforrás menedzsment:

Az egyének jelentik a szervezet legértékesebb és gyakran a legdrágább tőkéjét. Ez elengedhetetlenül fontossá teszi a megfelelő személyek rendelését a megfelelő projektcsapatokhoz, hogy a termelékenység költség hatékony módon a lehető legnagyobb legyen.

A csapat képzettségi adatainak és rendelkezésre állásának teljes körű ismerete szükséges a szervezet munkaportfoliónak megfelelő stratégiai erőforrás-kölcsönzést, - fejlesztést és - felállítást végezni.

Együttműködés és kommunikáció:

A lehető legnagyobb mértékű koordinálás és részvétel a jobb termelékenység érdekében

A hatékony együttműködés és kommunikáció alapvető a projektek sikeréhez. Világos kommunikációs folyamatok révén a csoport tagjai megoszthatják tudásukat, együtt készíthetik el a feladatokat és a leadandó anyagokat, gyorsan igazodhatnak a változásokhoz.

A projektekben részt vevő csoportok tagjai egyre távolabb vannak egymástól szervezeti és földrajzi értelemben is; ez veszélyezteti a termelékenységet, és olyan technológiát tesz szükségessé, ami értelmesen kapcsolja össze a csoporttagokat a koordináció és a minőség fenntartása érdekében.

Projektmenedzsment

Termékek és szolgáltatások szállítása esetén is a szervezetnek be kell tartania a projektek határidejét és költségvetését. Az ügyfelek elégedettségének fenntartása és elvárásainak teljesítése megköveteli, hogy ne legyenek hibák és késések.

A versenyképességhez a vállalatok növekvő mértékben tesznek kezdeményezéseket a projektek folyamatos javítására a ciklusidők rövidítése, a költségek leszorítása és a minőségellenőrzés révén. Mindehhez szakképzett egyének, szabványos folyamatok és magas szintű technológia szükséges, hatékony projektmenedzsment irányításával.

www.microsoft.com/hun

5. Unified Modeling Language

5.0 Bevezetés
5.1 Az UML fogalma
5.2 Történeti áttekintés
5.3 Az UML jellemzői
5.4 Az UML kritikája
5.5 Az UML építőkövei
5.6 Diagramok

5.0 Bevezetés

Objektum Orientált programozási nyelvek (OOP)

⇩

Objektum orientált tervezés (OOD)

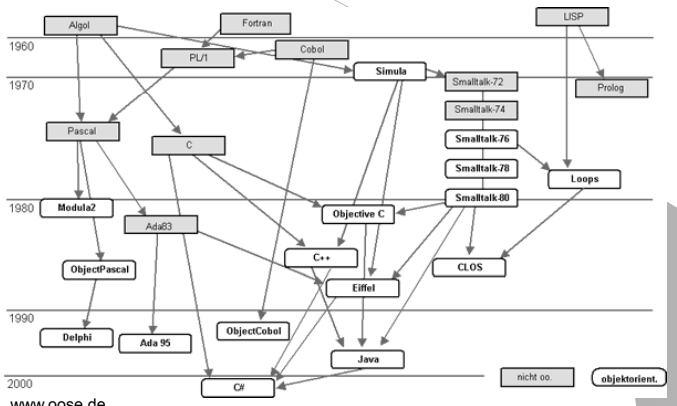
⇩

Objektum orientált analízis (OOA)

⇩

Objektum orientált analízis és tervezés (OOAD)

A programozási nyelvek fejlődése



www.oose.de

Jelentősebb OO módszertanok

1. **Coad-Yourdan: OO analízis és tervezés (OOAD), (1991)**

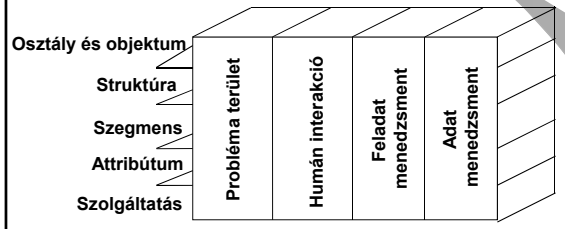
Jellemzője:
- **szerényebb eszközrendszerrel rendelkező,**
- **több szintű, több komponensű fejlesztési módszertan**

az alkalmazott szintek:
- **osztály & objektum**
- **struktúra**
- **szegmens (subject)**
- **attributum**
- **szolgáltatás (service)**

a figyelembe vett komponensek:

- **probléma-terület**
- **humán interakció**
- **feladat (task) menedzsment**
- **adat menedzsment**

A Coad-Yourdan OOAD vázlata



2. Wirfs-Brock et al., Responsibility Driven Design

Jellemzői:

- **Új szemléletű módszer, melyre jellemző az antropomorph megközelítés.**
- **A rendszert egymással együttműködő és „szövetkező" objektumok (ügynökök) alkotják, melyek mindegyikéhez jól meghatározott funkció és felelősség kötődik.**
- **Az objektumok közti kommunikációt „szerződések" (contract) rögzítik.**

Class Responsibility Card

Osztálynév	Absztrakt / konkrét
Ős osztályok	
Utód osztályok	
Felelősségek	Együttműködés

3. Jacobson: Object-Oriented Software Engineering (OOSE) (1992)

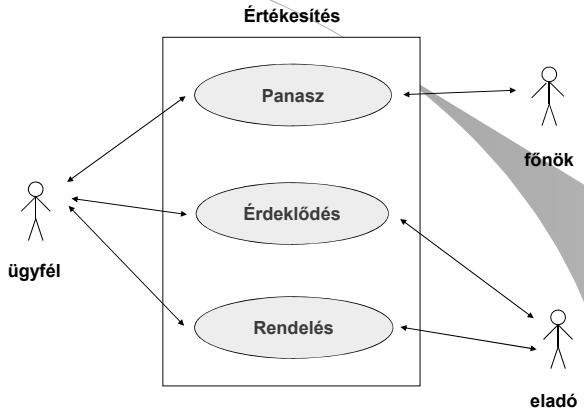
A módszer öt modellt használ az analízis és tervezés fázisaiban:

- követelmény modell
- analízis modell
- tervezési modell
- implementációs modell
- teszt modell

Use Case alapú megközelítés:

- felhasználók azonosítása
- szerepek meghatározása
- interakciók számbavétele
- rendszer viselkedésének

Use Case példa



4. Booch: Object-Oriented Design (OOD) (1991)

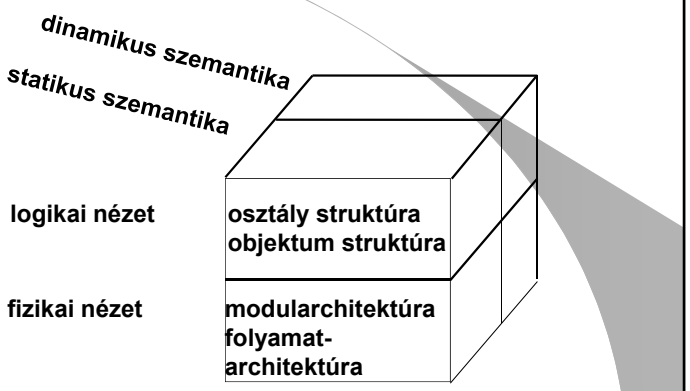
A rendszer statikus leírására használt diagrammok:

- Osztálydiagram
- Objektumdiagram
- Moduldiagram
- Folyamatdiagram

A dinamikus viselkedés leírására használt diagramok:

- Állapotdiagram
- Időzítésdiagram (Timing diagram)

A Booch-féle OOD felépítése

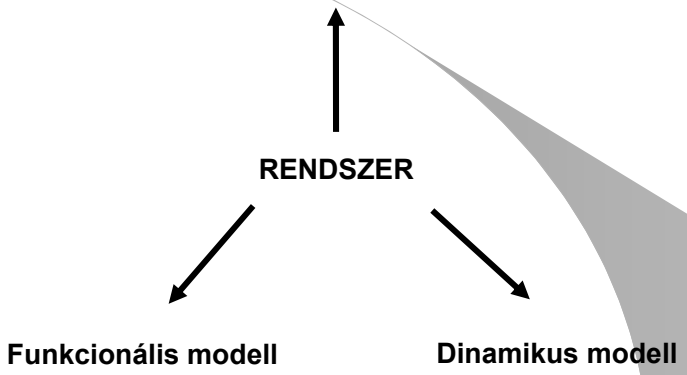


5. Rumbaugh: Object Modeling Technique (OMT)

J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy,
W. Lorensen: Object-Oriented Modelling Technique
Prentice-Hall Internat., 1993

A rendszert három különböző nézetből három modell képezi le.

- Objektum modell
- Dinamikus modell
- Funkcionális modell

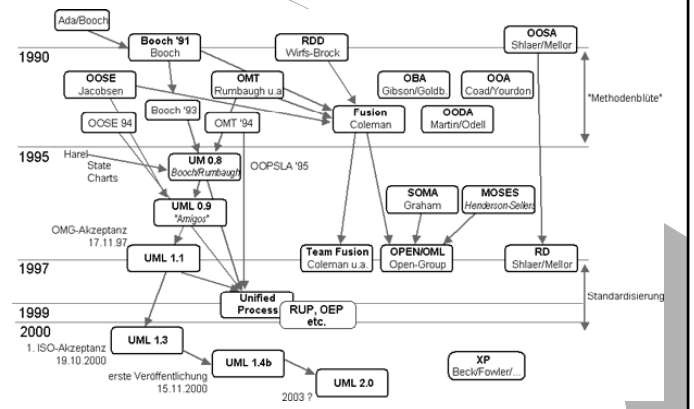


Az *Objektum modell* a rendszer struktúráját, szerkezetét ábrázolja.
- Osztály diagramm
- Objektum diagramm

A *Dinamikus modell* a rendszer viselkedését képezi le.
- Szcenárió
- Esemény diagramm
- Állapotátmenet ábra

A *funkcionális modell* a rendszer funkcionalitását reprezentálja.
- Adatfolyam diagramm

Az OO módszertanok fejlődése



www.oose.de

5.1 Az UML fogalma

„The Unified Modeling Language is a visual language for specifying, constructing and documenting the artifacts of systems. It is a general-purpose modeling language that can be used with all major object and component methods, and that can be applied to all application domains (e.g., health, finance, telecom, aerospace) and implementation platforms (e.g., J2EE, .NET).”[1]

1. Unified Modeling Language (UML) Specification: Infrastructure v2.0 (www.omg.org)

Az UML fogalma

„A Unified Modeling Language (egységes modellező nyelv) rendszerek elemeinek specifikálására, megalkotására és dokumentálására szolgáló vizuális nyelv. Általános célú modellező nyelv, amely minden nagy objektum és komponens alapú módszerrel használható bármely alkalmazási területen (pl. egészségügy, pénzügy, telekommunikáció, légi irányítás) és implementációs platformon (pl. J2EE, .NET).”[1]

1. Unified Modeling Language (UML) Specification: Infrastructure v2.0 (www.omg.org)

5.2 Az UML keletkezése

- Alapvetően három módszer egységesítéséből, összedolgozásából alakult ki:
  - Grady Booch: Booch Methode (BOOCH)
  - Jim Rumbaugh: Object Modeling Technique (OMT)
  - Ivar Jacobson: Object-Oriented Software Engeneering (OOSE)
- Közreműködött az Object Management Group (OMG), az objektum-orientált szakma legjelentősebb szervezete

A kialakulás időrendje

Év	Esemény
1994. október	Fejlesztés kezdete: Booch és Rumbaugh módszerének egyesítése
1995. október	Unified Method 0.8
1995. ősz	Jacobson csatlakozik, módszerét integrálják
1997. január	UML 1.0
1997. szept./1998…	UML 1.1/ UML 1.2 …
2005.	UML 2.0
2008.	2.2

5.3 Az UML jellemzői

- Hamar de-facto szabvánnyá vált
- A szoftveripar domináns modellező nyelvévé emelkedett
- Széles körben sikerrel alkalmazzák az egészségügytől az e-kereskedelemig
- Széleskörű együttműködés eredménye több vezető cég között: pl. Hewlett-Packard, IBM, Microsoft, Oracle, Unisys…

Célok az UML tervezésénél

- **A gyakorlatban jól használható, vizuális, kifejező modellező nyelv kialakítása**
- **A modell bővítését és specializálását lehetővé tevő mechanizmusok biztosítása a nyelvben**
- **Programozási nyelvtől és fejlesztési módszertől független legyen**
- **Formális alap biztosítása a modellező nyelv megértéséhez**
- **Az OO alapú CASE eszközök piacának növekedését ösztönzése**
- **Magasabb szintű fejlesztési koncepciók támogatása**
- **A létező legjobb technikák integrálása**

Az UML alapvető tulajdonságai

- Modellező nyelv, nem fejlesztési módszertan
- Grafikus szemléletet nyújt
- Lehetővé teszi szoftver-intenzív rendszerek specifikálását, konstruálását, vizualizálását és dokumentálását
- Munkafolyamatok, szervezetek, üzleti tevékenységek stb. leírására is alkalmas
- A gyakorlatban jól használható egyedi, osztott vagy konkurens rendszerek kezelésére

5.4 Az UML kritikája

- **Dagályos nyelvezet**
  - **Diagramok közötti redundancia**
  - **Alig használt diagramok**
- **Tanulási és megértési problémák**
  - **Sokan használják, de kevesen értik igazán**
- **A jelölésrendszerek általános problémája**
  - **Bizonyos rendszerek (itt OOP) jobban kifejezhetők, mint mások**
  - **A fejlesztő hajlamos az UML logikáját inkább követő megoldásokat választani**
- **Nehézkes átjárhatóság más modellezési technikákkal**

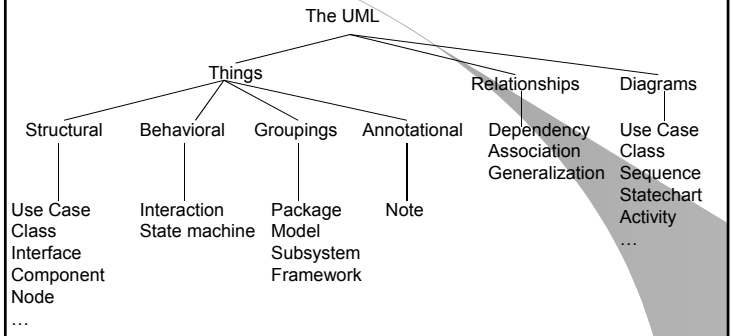
Wikipdia, the free encyclopedia, Unified Modeling Language, 2009. szeptember 10.

5.5 Az UML építőkövei

Az UML szókincse három fő kategóriába sorolható, amelyek tovább bonthatók:

- **Elemek**
  - **Strukturális**
  - **Viselkedési**
  - **Annotációs**
  - **Csoportos**
- **Kapcsolatok**
  - **Függőségi**
  - **Asszociációs (társítási)**
  - **Generalizációs (általánosítási, öröklődéssel kapcsolatos)**
- **Diagramok** …

Az UML építőkövei



Ivar Jacobson, G. Booch, J. Rumbaugh: The Unified Software Development Process, Addison Wesley Longman, 1999

Kapcsolatok

- A modellelemek közötti kapcsolatokat testesítik meg.
- Fajtái:
  - Függőségi (Dependency) ----------->
    - Egyik elem változása hatással van a másik elemre
    - Jele: szaggatott nyíl
  - Asszociáció, társítás (Association) ——————
    - Strukturális, szerkezeti összefüggés
    - Jele: egyenes vonal
  - Generalizáció (Generalization) ——————▷
    - Általános-speciális kapcsolata, öröklődés
    - Jele: üres háromszög fejű nyíl

5.6 Diagramok

- **A modellben lévő információ grafikus reprezentálása**
- **A szoftvert különböző nézőpontból és változó absztrakciós szinten láttatja**
- **Két fő csoportja:**
  - **Struktúrát modellező diagramok**
  - **Viselkedést modellező diagramok**

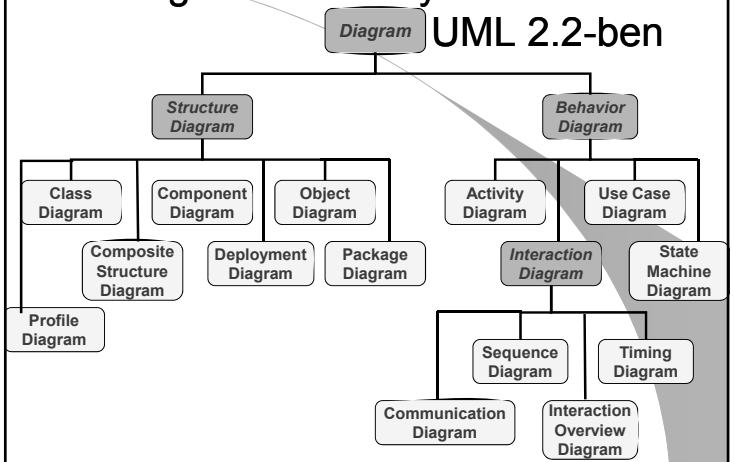
Struktúra diagramok

- A modell statikus architektúráját definiálják
- A „dolgokat” írja le, amelyből a modell felépül
- Időtől független elemek
- Pl.
  - Osztálydiagram
  - Komponensdiagram

Dinamikus diagramok

- A rendszer objektumainak dinamikus viselkedését mutatja
- Az interakciók változatosságát ragadja meg
- Időbeli változások sorozataként írható le
- Pl.
  - Használati eset diagramja
  - Aktivitásdiagram

A diagramok osztályozása az UML 2.2-ben



Wikipdia, the free encyclopedia, Unified Modeling Language, 2009. szeptember 10.

5.6.1 Áttekintés

Feladat:[1]

Liftek vezérlésének logikája többszintes épületben a következő megkötésekkel:

- **Minden liftnek szintenként egy gombja van. A gomb megnyomásra világít, és megáll az adott emeleten. A világítás megszűnik, ha megállt a lift az emeleten.**
- **A legfelső és a legalsó emeleten egy, a többin két gomb található az utazási irány megadására. A gombok megnyomásra kigyulladnak. Elalszanak, amikor egy lift megáll a kívánt emeleten, és a megfelelő irányba folyatatja az útját.**
- **Amikor nincs hívás, akkor a lift az emeleten marad zárt ajtóval.**

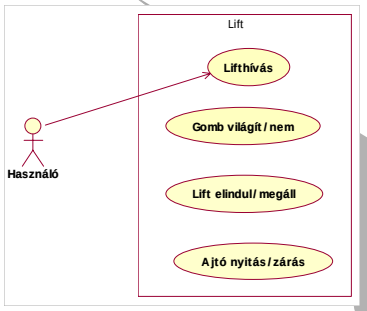
1. http://www.geocities.com/SiliconValley/Network/1582/uml-example.htm 2005. okt.

Áttekintés - UML

- Az UML egy modellező nyelv, amely a szemantikát és a jelölést határozza meg.
- Az analízishez a következő diagramokat használjuk:
  - Use Case diagram
  - Osztálydiagram
- A tervezésnél az alábbiakat használjuk:
  - Szekvenciadiagram
  - Részletes osztálydiagram

Áttekintés - Use Case diagram

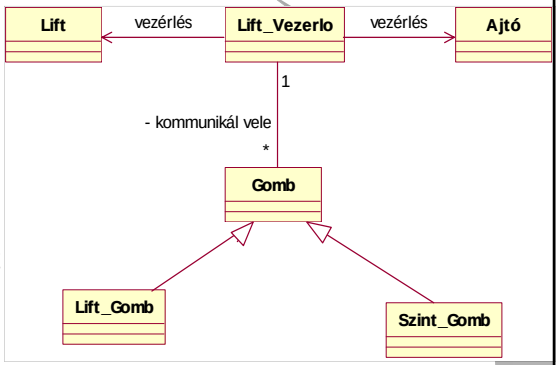
- A rendszer használatának általános leírása
- Áttekintést ad a rendszer tervezett működéséről
- Laikus és szakember számára egyaránt érthető



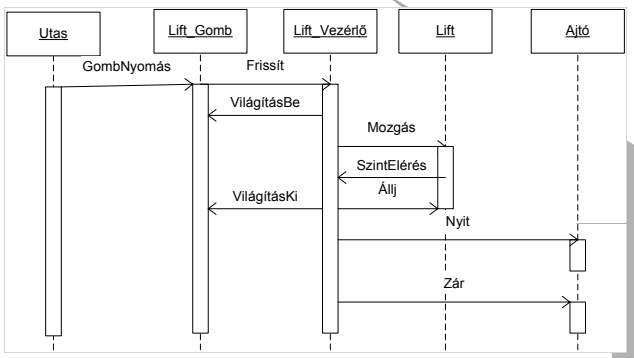
Áttekintés – Osztálydiagram

Az osztálydiagram megmutatja:

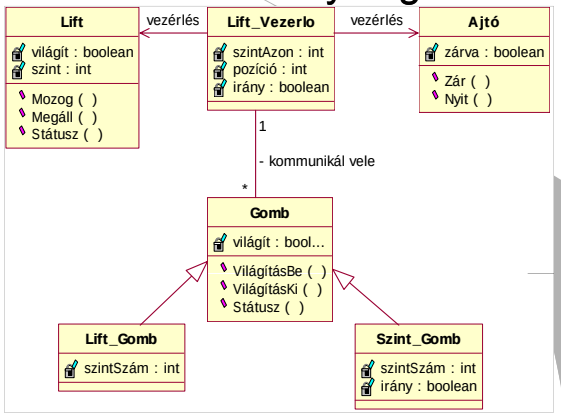
- A rendszer statikus struktúráját
- A belső struktúráját
- Kapcsolatait



Áttekintés - Szekvenciadiagram

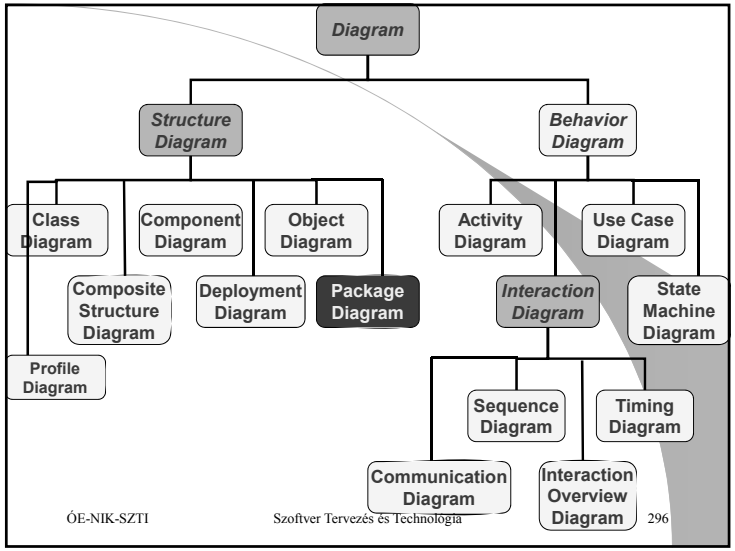


Áttekintés – Részletes osztálydiagram

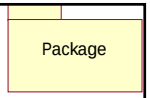


5.6.2 Package Diagram

- Csomagdiagram (Package)
- Használati eset diagramja (Use Case)
- Osztálydiagram (Class)
- Objektumdiagram (Object)
- Szekvenciadiagram (Sequence)
- Kommunikációs diagram (Communication)
- Állapotdiagram (Statechart)
- Aktivációs diagram (Activity)
- Komponensdiagram (Component)
- Telepítési diagram (Deployment)



Package Diagram



Jellemzői:

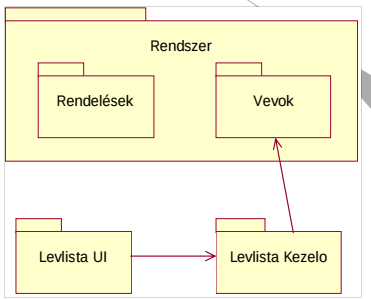
- **A rendszert a legmagasabb absztrakciós szinten bontja fel**
- **A logikailag összetartozó UML elemeket (főleg osztályok) csoportosítja**
- **A csomagok között függőségi kapcsolatokat (pl. egyik elem a másikban változást idéz elő, üzenetküldés) ábrázolja**

Megjegyzés

- **A csomagok egymásba ágyazhatók**
- **Jelölés: négyszög füllel**
- **A csomag megfelelője Java környezetben a package, .NET környezetben a namespace**

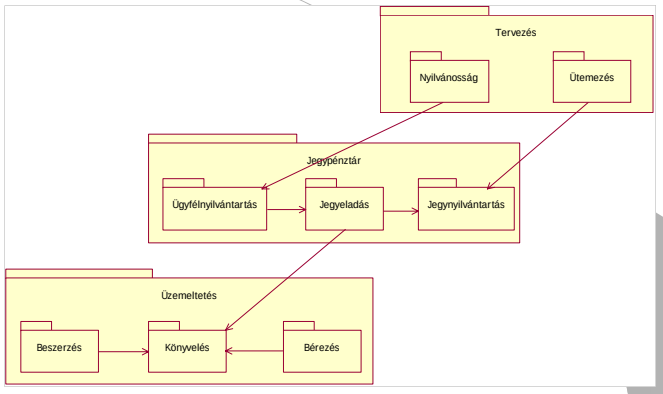
Package Diagram

Példák - Bolt



Package Diagram

Példák - Színház



Rumbaugh, Jacobson, Booch: The Unified Modeling Language Referene Manual, 2004

Package Diagram

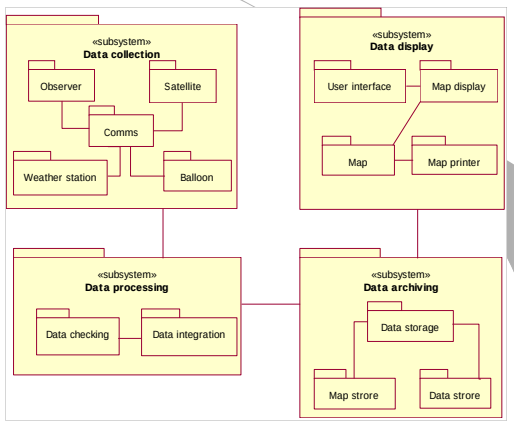
TK Példa

Egy meteorológiai térkép rendszernek meteorológiai térképeket kell előállítania, távoli, felügyelet nélküli meteorológiai állomásoktól és egyéb adatforrásoktól, mint például léggömböktől és műholdaktól összegyűjtött adatok alapján. A meteorológiai állomások adataikat egy körzeti számítógép kérésére megküldik a kérelmező gépnek.

A körzeti számítógépes rendszer ellenőrzi a begyűjtött adatokat és integrálja a különböző adatforrásokból érkező adatokat. Az integrált adatokat archiválja és az archívum, valamint egy digitalizált térképadatbázis alapján létrehozza a helyi meteorológiai térképeket. A térképek egy speciális célú térképnyomtatón kinyomtathatók, majd szétoszthatók, vagy számos más módon megjeleníthetők.

Sommerville: Szoftverrendszerek fejlesztése, Panem, 2002

TK Példa - A meteorológiai térképrendszer alrendszerei

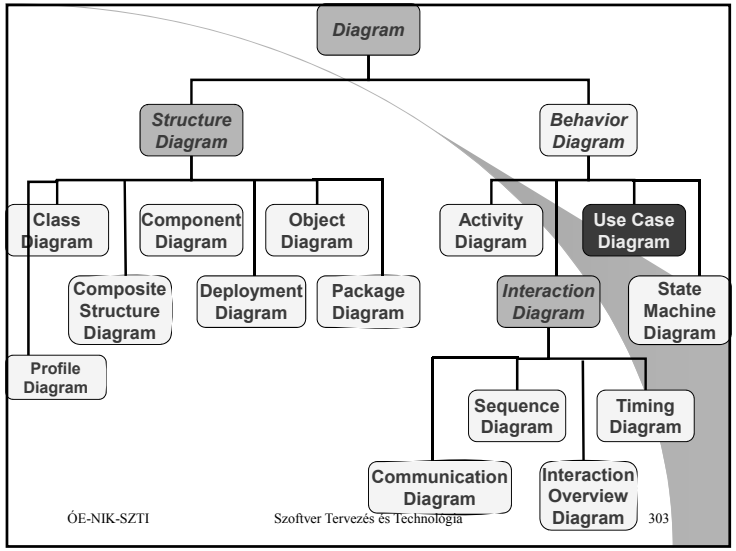


Sommerville: Szoftverrendszerek fejlesztése, Panem, 2002

Package Diagram

5.6.3 Use Case Diagram

- Csomagdiagram (Package)
- Használati eset diagramja (Use Case)
- Osztálydiagram (Class)
- Objektumdiagram (Object)
- Szekvenciadiagram (Sequence)
- Kommunikációs diagram (Communication)
- Állapotdiagram (Statechart)
- Aktivációs diagram (Activity)
- Komponensdiagram (Component)
- Telepítési diagram (Deployment)



Use Case Diagram

- Eszköz a rendszer elvárt működésének specifikálásához
- Tipikusan a követelmények feltárására használják:
  - Mi az, amit a rendszernek tudnia kell
- Kulcsfogalmak:
  - Aktor (Actor)
  - Használati esetek (Use Cases)
  - Tárgy (Subject)
  - Kapcsolatok (Connection)

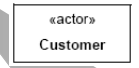
Use Case Diagram

A tárgy

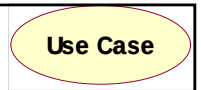
- A rendszer maga, olyan szempontból, ahogy a használati esetek alkalmazzák
- Jelölése: egy téglalap, melynek határa a rendszer határa

Az aktor

- Felhasználók és más rendszerek, amelyek kapcsolatba lépnek a rendszerrel
- Mindig a rendszeren kívül van
- Szerepet definiál
- Nem feltétlenül fizikai entitás
- Jelölése:
  - Pálcikaember
  - Osztályként
  - Egyéb ikonnal, ami nem emberi voltára utal



Használati eset



- A rendszer ajánlott viselkedését írja le
- Nem utal a belső struktúrára
- Eredménye:
  - Megváltozik a rendszer állapota
  - Kommunikáció a környezettel
- Aktorokkal működik együtt
- Speciális esetei:
  - Alapviselkedés variációja (include)
  - Kivételes viselkedés, hibakezelés (extend)
- Jelölése: ellipszis

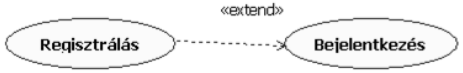
Kapcsolatok

Kapcsolatok	Leírás	Jelölés
Association	A kommunikáció útja az aktor és használati eset között	Egyenes vonal vagy nyíl
Extend	Két használati eset között meghatározza a speciális, kiterjesztett viselkedés idejét és mikéntjét.	Szaggatott nyíl a kiterjesztéstől az alap felé. «extend»
Include	Két használati eset között. A tartalmazott eset az alap egy kiemelt része.	Szaggatott nyíl az alap-tól a tartalmazott felé. «include»
Generalization	Általános használati esetből vagy aktorból származik egy specifikusabb.	Folytonos nyíl üres háromszögű fejjel.

Példák kapcsolatokra

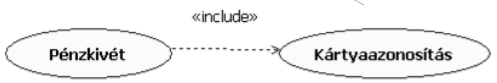
Asszociáció

Extend

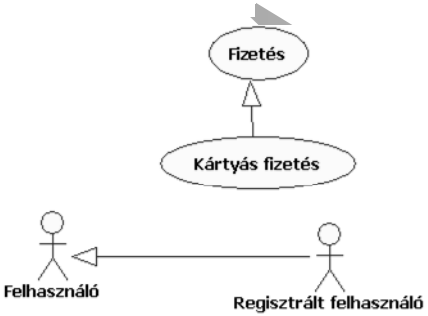


Példák kapcsolatokra

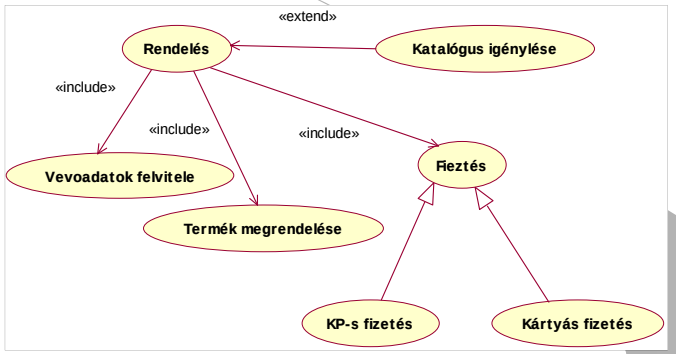
Include



Generalizációk

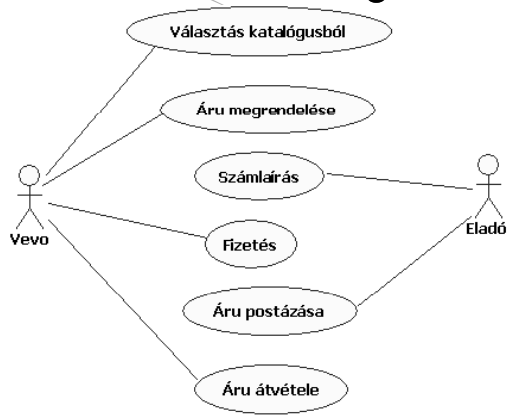


Példák – Place Order



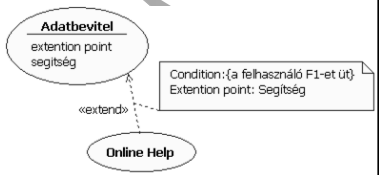
Unified Modeling Language (UML) Specification: Infrastructure v2.0 (www.omg.org)

Áruvásárlás katalógusból



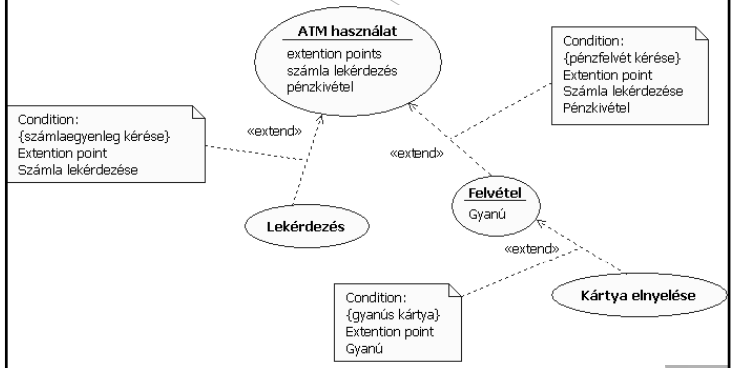
Kiterjesztési pont (Extension point)

- Az extend kapcsolat minden kiterjesztett használati esethez tartalmaz egy kiterjesztési pontot
- Meghatározza azt a pontot, ahol a viselkedés bizonyos feltételinél a használati eset kiterjeszthető egy másik használati esettel.



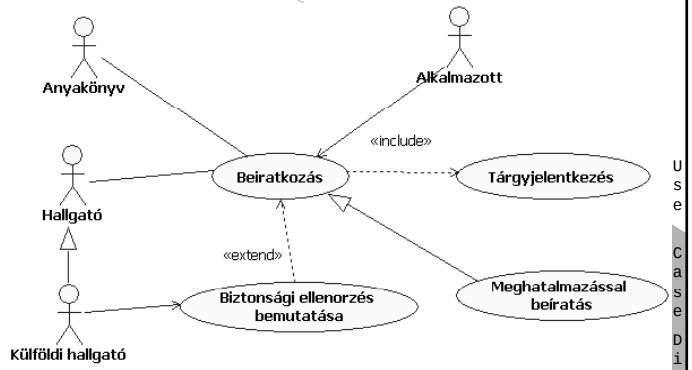
- A feltételt és a kiterjesztési pontot az extend kapcsolathoz jegyzetként kell hozzácsatolni (Note Attachment). (UML 2.x-ben)

Példák - ATM használat



Rumbaugh, Jacobson, Booch: The Unified Modeling Language Referene Manual, 2004

Példák - Iskola

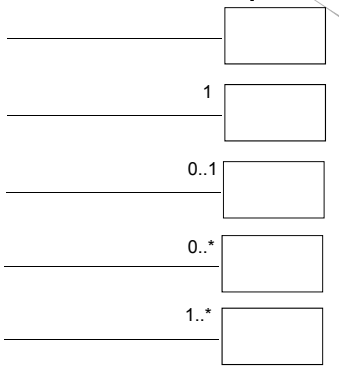


Megjegyzés: nyíllal jelölt asszociációk, a hatás irányát mutatják

http://www.agilemodeling.com/artifacts, 2005. október

Use Case Diagram

A számosság kifejezése az UML-ben (multiplicitás)



Pontosan 1

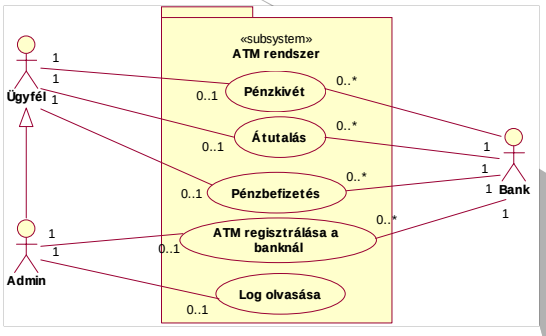
1 **Pontosan 1**

0..1 **Feltételes (0 vagy 1)**

0..* **Feltételes (de több is lehet)**

1..* **Legalább egy (de több is lehet)**

Példák – ATM rendszer



Megjegyzés: a számok a számosságot jelölik, a keret a tárgy határát

Unified Modeling Language (UML) Specification: Infrastructure v2.0 (www.omg.org)

Use Case Diagram

TK Példa

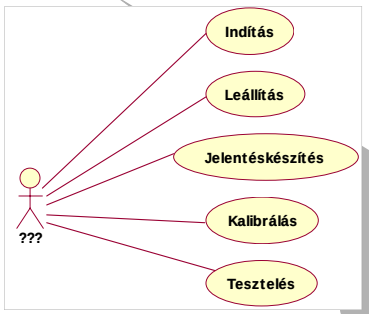
Egy meteorológiai térkép rendszernek meteorológiai térképeket kell előállítania, távoli, felügyelet nélküli meteorológiai állomásoktól és egyéb adatforrásoktól, mint például léggömböktől és műholdaktól összegyűjtött adatok alapján. A meteorológiai állomások adataikat egy körzeti számítógép kérésére megküldik a kérelmező gépnek.

A körzeti számítógépes rendszer ellenőrzi a begyűjtött adatokat és integrálja a különböző adatforrásokból érkező adatokat. Az integrált adatokat archiválja és az archívum, valamint egy digitalizált térképadatbázis alapján létrehozza a helyi meteorológiai térképeket. A térképek egy speciális célú térképnyomtatón kinyomtathatók, majd szétoszthatók, vagy számos más módon megjeleníthetők.

Sommerville: Szoftverrendszerek fejlesztése, Panem, 2002

TK Példa – A meteorológiai állomás használati esetei

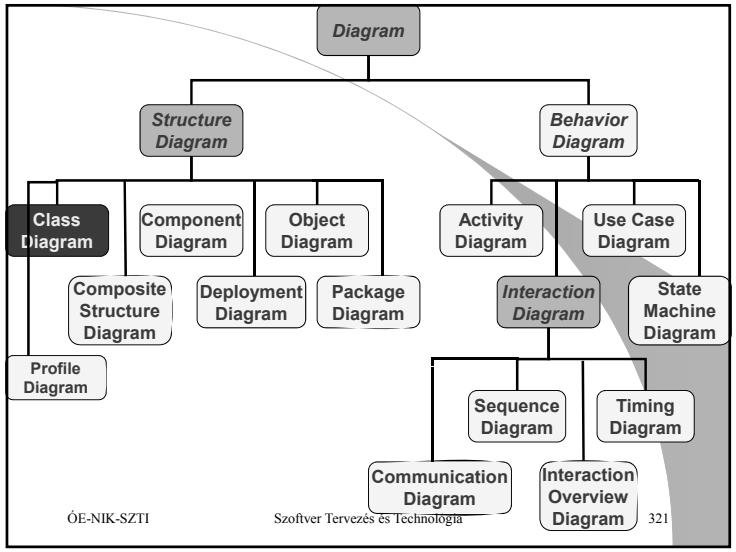
„A meteorológiai állomás külső entitásokkal működik együtt az indítás és a leállítás, az összegyűjtött mete-orológiai adatokból való jelentéskészítés, valamint a műszerek tesztelése és kalib-rálása céljából.”



Use Case Diagram

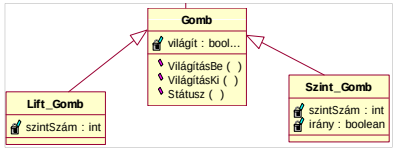
5.6.4 Class Diagram

- Csomagdiagram (Package)
- Használati eset diagramja (Use Case)
- Osztálydiagram (Class)
- Objektumdiagram (Object)
- Szekvenciadiagram (Sequence)
- Kommunikációs diagram (Communication)
- Állapotdiagram (Statechart)
- Aktivációs diagram (Activity)
- Komponensdiagram (Component)
- Telepítési diagram (Deployment)



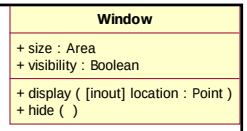
Class diagram

- A modell (egy részének) statikus nézetét írja le
- Objektumorientált rendszer építőelemeit mutatja
- A rendszer osztályait és a közöttük lévő kapcsolatokat írja le



Class Diagram

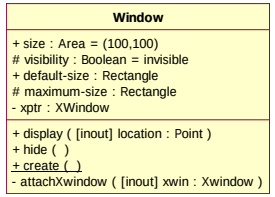
Osztály



- UML-es szóhasználat:
  - Attribútumok
  - Operátorok
- Jelölés: három részből álló téglalap
  - Név rész
  - Attribútum rész (opcionális)
  - Operátor rész (opcionális)
- Láthatóságok: + public, - private, # protected, ~ package

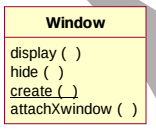
Class Diagram

Osztályok



Részletes osztálydeklaráció

Osztályszimbólum részletek nélkül



Osztálydeklaráció attribútumokkal

Jelölések:

egyenlőségjel: kezdőérték

aláhúzás: osztályszintű operátor

Rumbaugh, Jacobson, Booch: The Unified Modeling Language Referene Manual, 2004

Class Diagram

Az osztályok meghatározásának menete

- **A leírásból (scope) válasszuk ki a főneveket.**
- **Szűrjük ki az egyértelműen nem a rendszerhez tartozó, csak az érthetőséget javító főneveket**
- **Szűrjük ki a szinonimákat, rokon értelmű főneveket.**
- **Keressünk rejtett főneveket (a szövegben explicite nem szerepel, de a kultúrából következik)**
- **Az így kapott jelöltekből válasszuk ki a rendszert alkotó objektumokat.**

TK Példa

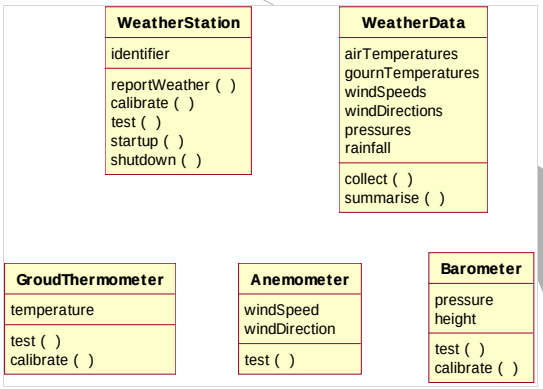
Egy meteorológiai térkép rendszernek meteorológiai térképeket kell előállítania, távoli, felügyelet nélküli meteorológiai állomásoktól és egyéb adatforrásoktól, mint például léggömböktől és műholdaktól összegyűjtött adatok alapján. A meteorológiai állomások adataikat egy körzeti számítógép kérésére megküldik a kérelmező gépnek.

A körzeti számítógépes rendszer ellenőrzi a begyűjtött adatokat és integrálja a különböző adatforrásokból érkező adatokat. Az integrált adatokat archiválja és az archívum, valamint egy digitalizált térképadatbázis alapján létrehozza a helyi meteorológiai térképeket. A térképek egy speciális célú térképnyomtatón kinyomtathatók, majd szétoszthatók, vagy számos más módon megjeleníthetők.

Sommerville: Szoftverrendszerek fejlesztése, Panem, 2002

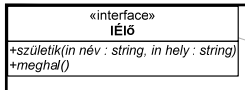
Class Diagram

TK Példa - Osztályok



Sommerville: Szoftverrendszerek fejlesztése, Panem, 2002

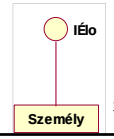
Class Diagram



Interface

- Operációk halmazát adja meg, melyet más osztályok megvalósítanak
- Az interfészt megvalósító osztály és az interfész között realizációs kapcsolat van
- Jelölés:
  - Osztályjelölés + <<interface>> sztereotípia
  - Kör

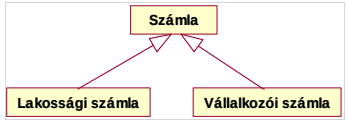
A dőlt betűs operátor- és osztálynevek az elem absztrakt voltát jelölik.



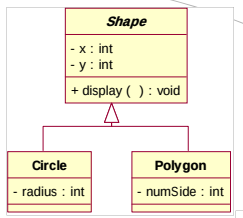
Class Diagram

Generalizációs kapcsolat

- Az öröklődést fejezi ki
- Általános és specifikus osztály kapcsolata

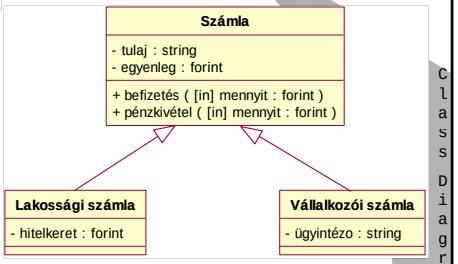


Class Diagram



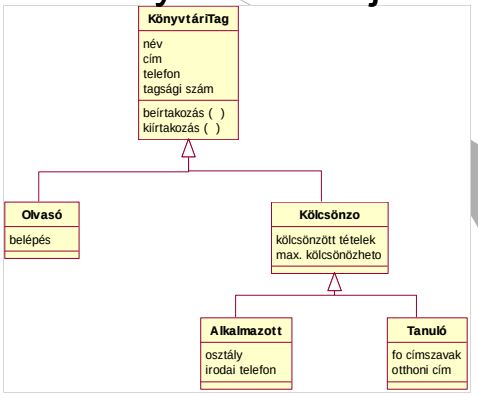
Jelölés:

Operátornál kettőspont után a visszatérési érték van



Class Diagram

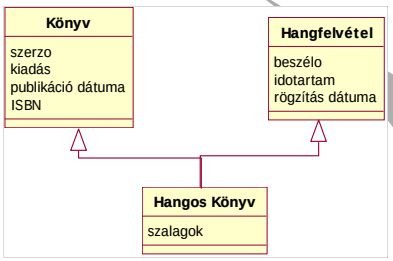
Könyvtári tagok osztályhierarchiája



Sommerville: Szoftverrendszerek fejlesztése, Panem, 2002

Class Diagram

Példa többszörös öröklésre

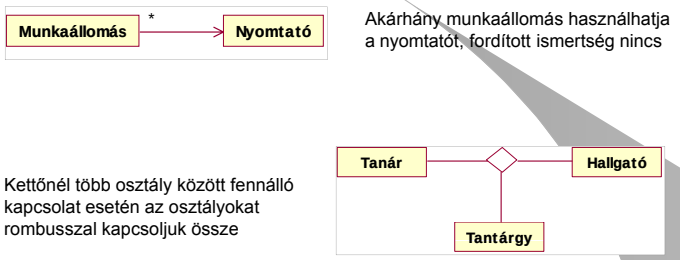


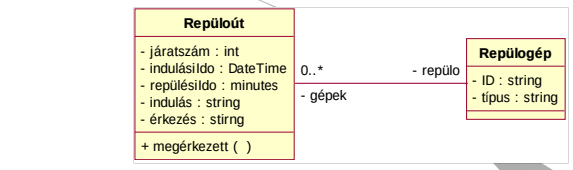
Sommerville: Szoftverrendszerek fejlesztése, Panem, 2002

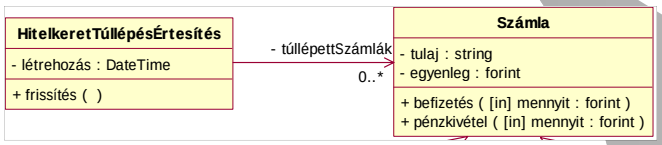
Class Diagram

Asszociációs kapcsolat

- Két osztály összekapcsolása
- A legáltalánosabb reláció ——
- Nyíllal jelölhető a navigálhatósága: ⟶
  - A nyíl irányában ismeri az egyik a másikat
- Multiplicitás jelöli, hogy hány példány vehet részt a kapcsolatban, az alapértelmezett az 1
- Az objektumok szerepét a kapcsolaton jelölhetjük

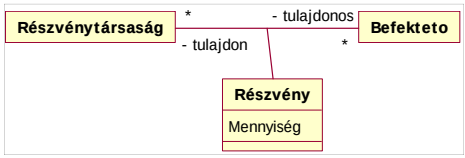




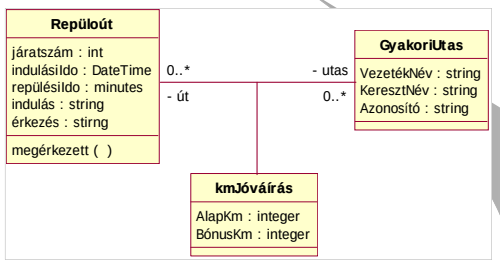


Asszociáció megadása osztállyal

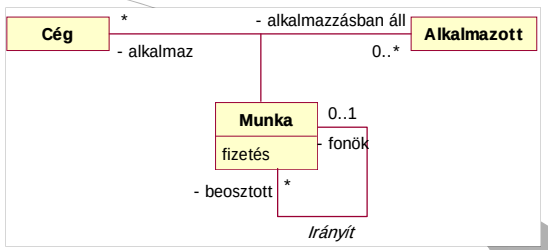
- A kapcsolatra vonatkozó információkat tartalmaz
- Osztályként adható meg, szaggatott vonallak kapcsolódik az asszociációhoz
- Az osztálynak attribútumai vannak



Rumbaugh, Jacobson, Booch: The Unified Modeling Language Referene Manual, 2004



http://www-128.ibm.com/developerworks/rational/library/content/RationalEdge/nov03/t_modelinguml_db.pdf 2005. október



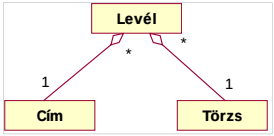
A fizetés nem lehet a cég vagy az alkalmazott attribútuma, mert sok-sok kapcsolat van közöttük. A kapcsolat attribútumának kell lennie.

A beosztott-főnök viszony két ember munkája közötti kapcsolat, nem egyszerűen két ember közötti kapcsolat, ezért ez egy asszociáció az asszociációs osztályon.

Rumbaugh, Jacobson, Booch: The Unified Modeling Language Referene Manual, 2004

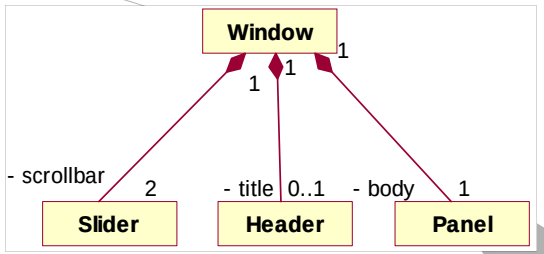
Aggregációs kapcsolat

- Az asszociáció egyik formája
- Az egyik osztály alárendeltje a másiknak
- Tartalmazó-tartalmazott viszony, a tartalmazott túlélheti a tartalmazót



Kompozíciós kapcsolat

- Az aggregáció erősebb formája
- A részek életciklusa egybeesik a tartalmazóéval
- Egy objektum csak egy tartalmazóhoz tartozhat
- A tartalmazó objektum felelős a részek létrehozásáért és megsemmisítéséért
- Osztott aggregáció:
  - olyan aggregáció, ami nem kompozíció
  - a tartalmazók osztozhatnak a részeken

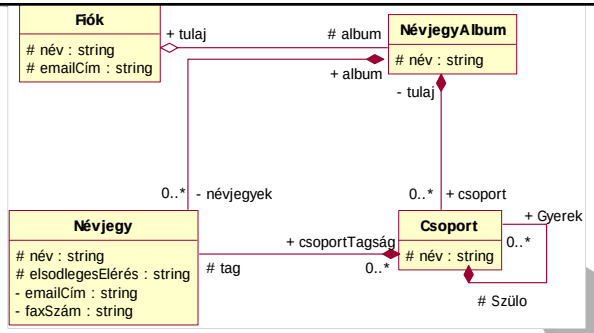


Megjegyzés:

A kompozíció definíciója jól kötődik az automatikus szemétgyűjtés (garbage collection) fogalmához. Ha a tartalmazó objektum megsemmisül, a részre mutató is megsemmisül, és a rész elérhetetlenné válik, a szemétgyűjtő hatáskörébe kerül, visszaállítani lehetetlen.

Rumbaugh, Jacobson, Booch: The Unified Modeling Language Referene Manual, 2004

Class Diagram



Megjegyzés:

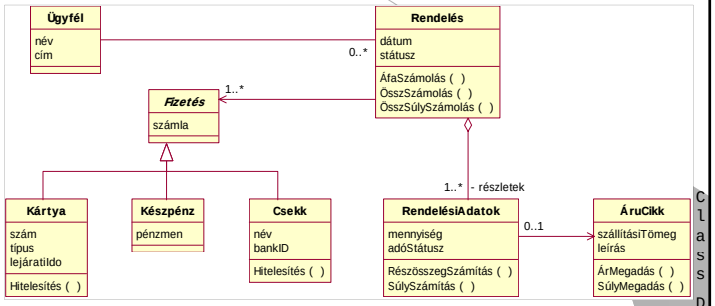
Az aggregáció mutatja, hogy a fiók használja a névjegyalbumot, de nem feltétlenül tartalmaz belőle példányt.

A kompozíció tartalmazást, illetve tulajdonlást fejez ki a két osztály között. Például a névjegy értékeit tartalmazza a névjegyalbum.

http://www.sparxsystems.com.au/resources/tutorial alapján 2005. október

Class Diagram

Összefoglalás - Rendelés

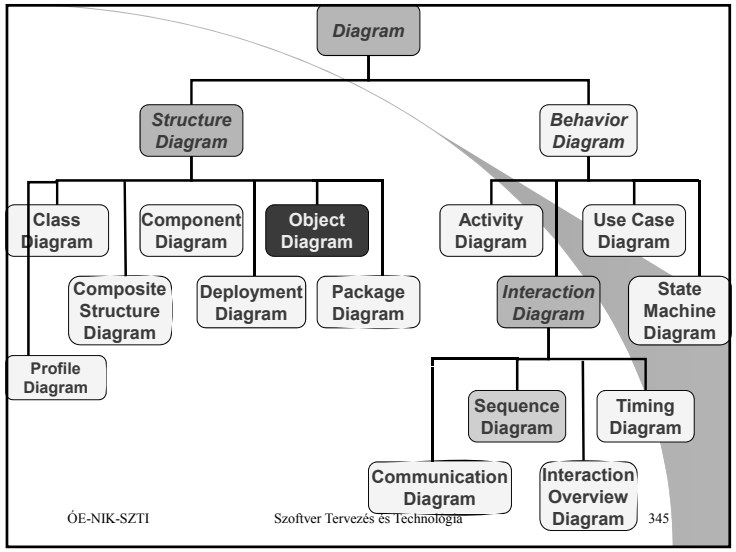


http://bdn.borland.com/article/0,1410,31863,00 2005. október

Class Diagram

5.6.5 Object Diagram

- Csomagdiagram (Package)
- Használati eset diagramja (Use Case)
- Osztálydiagram (Class)
- Objektumdiagram (Object)
- Szekvenciadiagram (Sequence)
- Kommunikációs diagram (Communication)
- Állapotdiagram (Statechart)
- Aktivációs diagram (Activity)
- Komponensdiagram (Component)
- Telepítési diagram (Deployment)



Object diagram

- Objektum: egy osztály egy példánya
- Objektumdiagram: objektumokat és azok kapcsolatait mutatja egy adott pillanatban
- A rendszer adott időben vett állapotát reprezentálja
- Az objektum minden attribútumának van értéke
- Az objektum összeköttetésben áll más objektumokkal
  - Link: az asszociáció egy példánya

Objektumok jelölése

- Hasonlít az osztálydiagramok ábrázolásához
- Két része van:
  - Névrész
    - Objektum vagy az osztály neve aláhúzva
  - Attribútum rész
    - Értékekkel

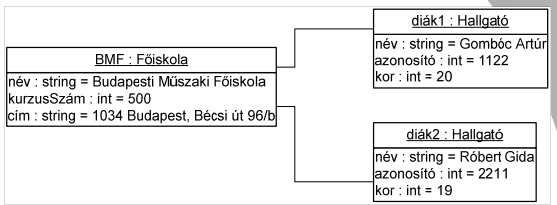
háromszög : Poligon
középpont = (0,0) csúcsok = ((0, 0), (4, 0), (4, 3)) szegélySzín = fekete kitöltőSzín = fehér

A metódusok (operációk) feltűntetésére nincs szükség, hiszen azok minden objektumban azonosak.

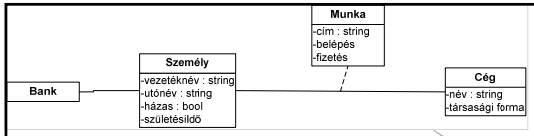
Példa

Osztálydiagram

Az osztálydiagram egy lehetséges objekumdiagramja

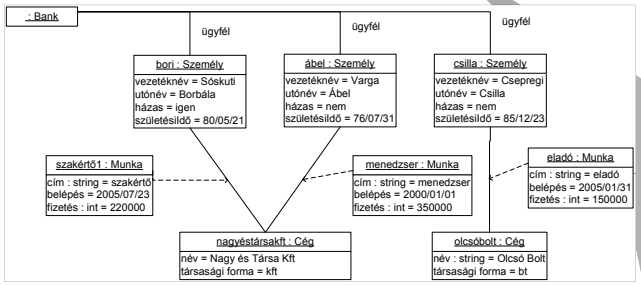


Példa 2.



Osztálydiagram

Az osztálydiagram egy lehetséges objekum-diagramja

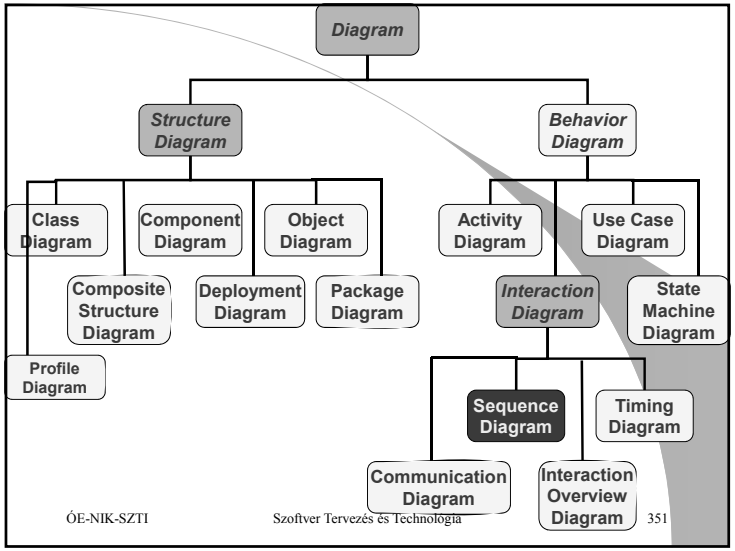


http://universalis.elibel.tm.fr/pub/instances/mars/Bank/imabers/bnkInstances.gif 2005. november

Object Diagram

5.6.6 Sequence Diagram

- Csomagdiagram (Package)
- Használati eset diagramja (Use Case)
- Osztálydiagram (Class)
- Objektumdiagram (Object)
- Szekvenciadiagram (Sequence)
- Kommunikációs diagram (Communication)
- Állapotdiagram (Statechart)
- Aktivációs diagram (Activity)
- Komponensdiagram (Component)
- Telepítési diagram (Deployment)

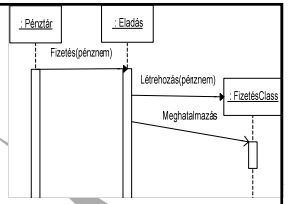


Sequence Diagram

- Viselkedési, interakciós diagram
- Az objektumok interakcióit mutatja az idő függvényében
- Elemei:
  - Objektumok, melyek részt vesznek az interakcióban
  - A kicserélt üzentek sorozata
- Explicit mutatja az időt, de nem jelöli az osztályok kapcsolatát

Sequence Diagram

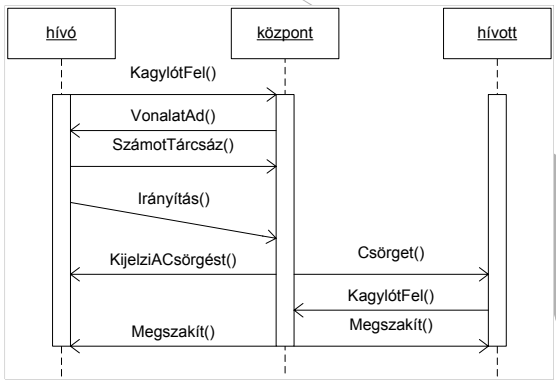
Jelölés



- Függőlegesen jelenik meg az idő
  - Életvonal: az objektum időben való létezését jelenti
  - Aktivációs életvonal: az objektumhoz kapcsolódó műveletek hajtódnak végre
- Vízszintesen az interakcióban részt vevő objektumok vannak
- Egy objektum téglalap jelölése elhelyezkedhet a diagram tetején vagy ahol létrejön
- Az üzenetek különböző típusúak lehetnek
- X jelzi az objektum megszűnését

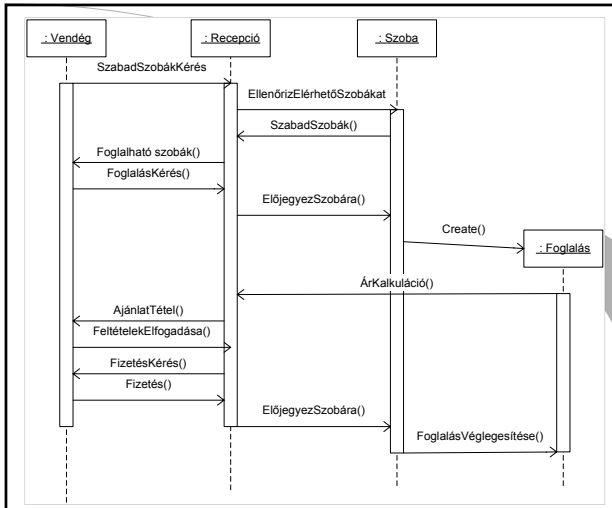
Sequence Diagram

Telefonhívás



Sequence Diagram

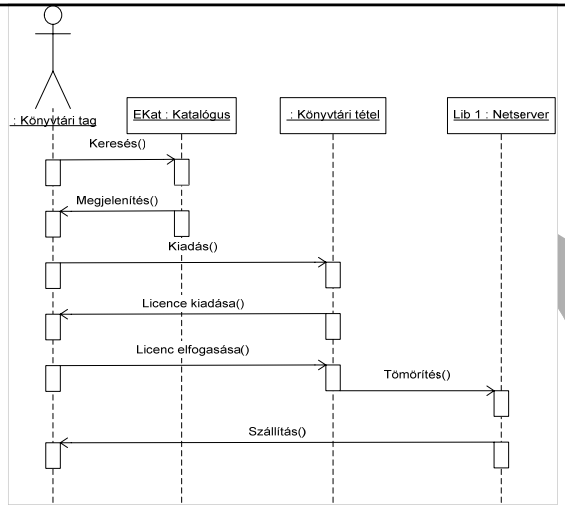
Szobafoglalás



Sequence Diagram

http://government.popkin.com/images/umlsequencediagram.gif 2005. november

Elektronikus tételek kiadása (Sommerville)



Sequence Diagram

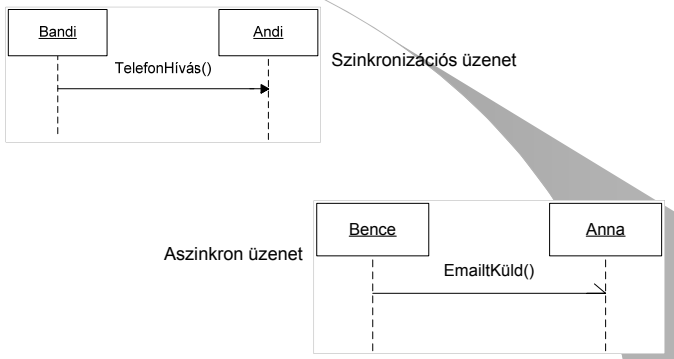
Ian Sommerville: Szoftverrendszerek fejlesztése, Panem, 2002, p. 201

Üzenettípusok

- Egyszerű üzenet: az aktív objektum átadja a vezérlést a passzív objektumnak
- Szinkronizációs üzenet: a küldő blokkolt állapotba kerül, amíg a fogadó nem fogadta az üzenetet
- Randevú üzenet: a fogadó várakozik arra, hogy a küldő üzenetet küldjön neki
- Aszinkron üzenet: a küldő folyamat nem szakad meg, nem érdekli őt, hogy mikor kapta meg a fogadó az üzenetet
- Visszatérési üzenet: amikor az objektum a vezérlést visszaadja , gyakran nem tüntetjük fel

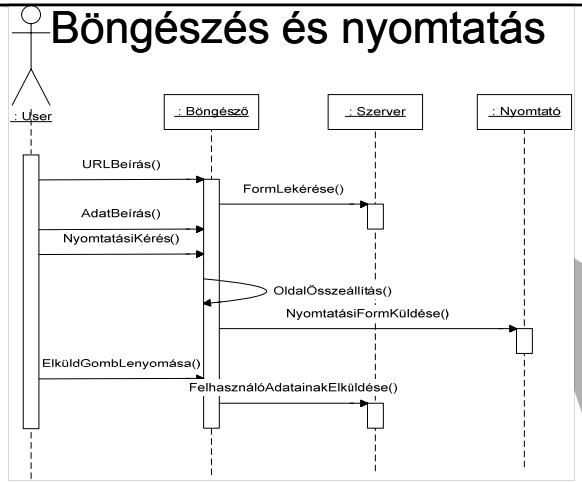
Sike Sándor, Varga László: Szoftvertechnológia és UML, ELTE Eötvös Kiadó, 2003

Példák üzenettípusokra



Sike Sándor, Varga László: Szoftvertechnológia és UML, ELTE Eötvös Kiadó, 2003

Böngészés és nyomtatás

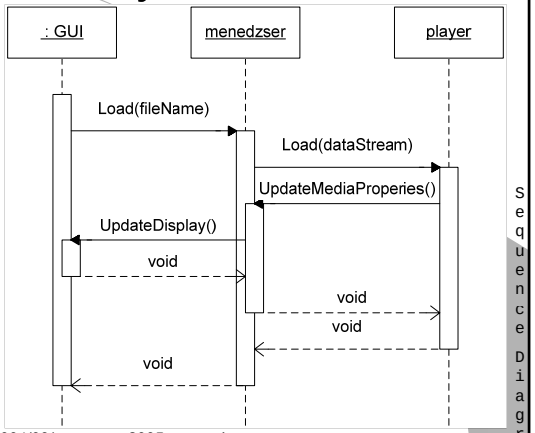


http://www.w3.org/TR/xhtml-print/formsUsageModel.jpg 2005. november

Médiafájl betöltése

Médiafájl betöltése felhasználói felületen keresztül. A MediaPlayer osztály példánya értesíti az őt hívót a betöltött fájl tulajdonságairól, amely frissített felhasználói felületet eredményez.

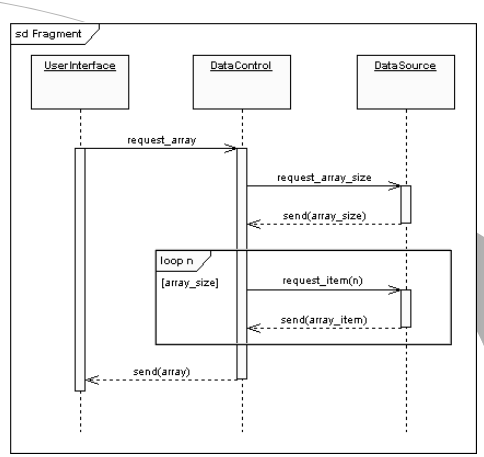
A vezérlés visszaadását külön feltüntettük.



http://dekstpo.de/weblog/2004/03/sequence 2005. november

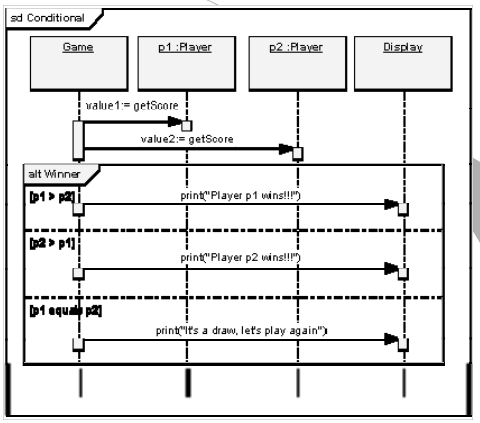
Feltételek, ciklusok…

- A diagramon belül beágyazott diagramrésszel ábrázoljuk
- A részek alrészekre oszthatók (vízszintes szaggatott vonallal), ha szükséges (pl. feltételnél)
- A rész viszonyát az egész diagramhoz a sarkába írt cím jelzi



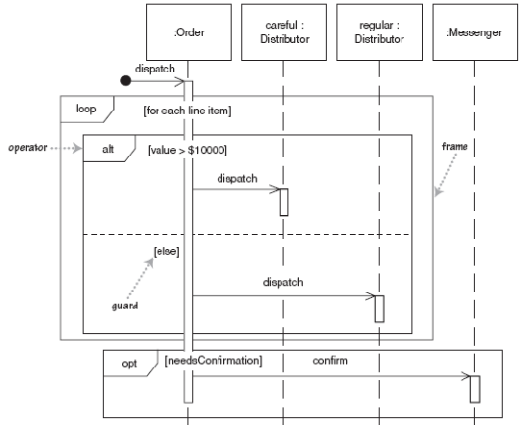
http://sparxsystems.com.au/resources/uml2_tutorial/uml2_sequencediagram.html2005. nov.

Játék



http://www.liemur.co.uk/Articles/UMLInteractionFragments.htm 2005. november

Rendelés



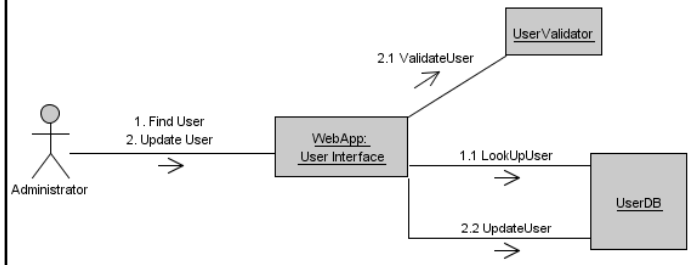
5.6.7 Collaboration Diagram

- Csomagdiagram (Package)
- Használati eset diagramja (Use Case)
- Osztálydiagram (Class)
- Objektumdiagram (Object)
- Szekvenciadiagram (Sequence)
- Kommunikációs diagram (Communication)
- Állapotdiagram (Statechart)
- Aktivációs diagram (Activity)
- Komponensdiagram (Component)
- Telepítési diagram (Deployment)

Communication Diagram

- Objektumok közti interakciót mutatja
- Az objektumdiagram és a szekvenciadiagram keresztezése
- Szabad elrendezést enged az objektumok között (a szekvenicadiagramtól eltérően)
  - Így könnyebb adott objektum interakcióit áttekinteni
  - Az üzenetek időrendben számozottak
- Az UML 1.x-ben Collaboration Diagramnak hívják

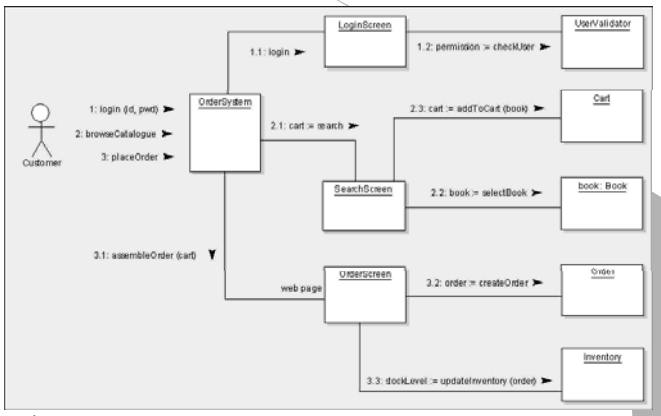
Felhasználók kezelése weben



Communication Diagram

http://www.dotnetcoders.com/web/learning/uml/diagrams/collaboration.aspx 2005. nov.

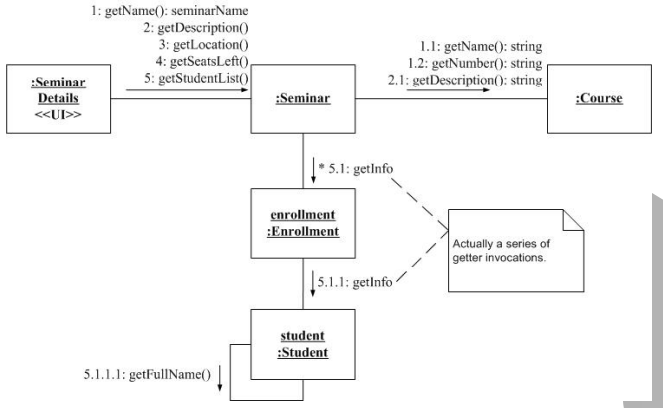
Webes könyvvásárlás



Communication Diagram

http://sparxsystems.com.au/EAUserGuide/communicationdiagram.htm2005. nov.

Tárgyjelentkezés

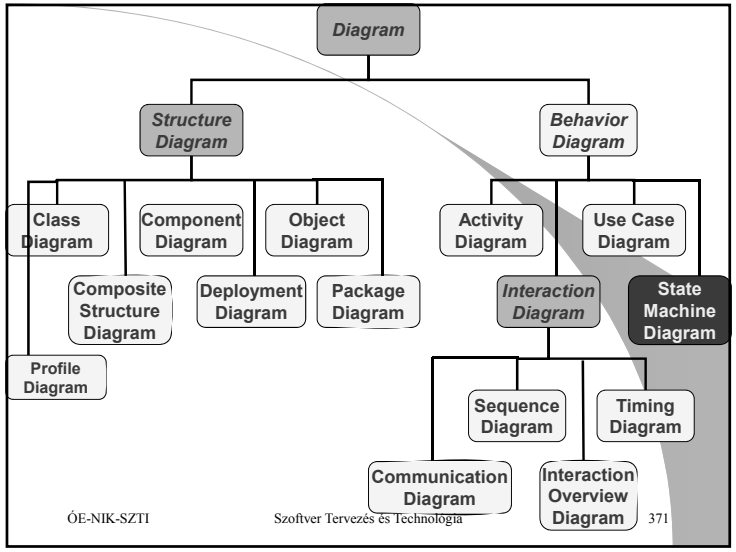


Communication Diagram

http://www.agilemodeling.com/artifacts/communicationDiagram.htm. 2005. nov.

5.6.8 State Machine Diagram

- Csomagdiagram (Package)
- Használati eset diagramja (Use Case)
- Osztálydiagram (Class)
- Objektumdiagram (Object)
- Szekvenciadiagram (Sequence)
- Kommunikációs diagram (Communication)
- Állapotdiagram (Statechart, State Machine)
- Aktivációs diagram (Activity)
- Komponensdiagram (Component)
- Telepítési diagram (Deployment)

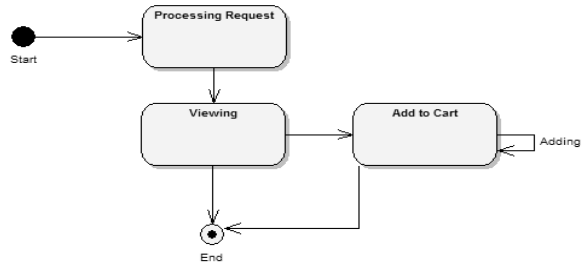


Állapotdiagram

- Elnevezés, eredet:
  - David Harel dolgozta ki a statechart diagramok alapelvét
  - Az UML state machine fogalma ezen alapszik
- Egyetlen objektum vagy interakció állapotainak sorozatát írja le
- Az állapotgép forrásosztályhoz kapcsolódik, és példányainak viselkedését specifikálja

State Machine Diagram

Állapotdiagram – webes vásárlás



http://www.sparxsystems.com.au/resources/tutorial/dynamic_model.html 2005. nov.

Az állapotdiagram elemei

- Állapot
- Belépési pont (entry point) ○
- Kilépési pont (exit point) ◎
- Átmenet: ⟶
  - Állapotok közötti kapcsolat
  - Feltüntethetjük rajta:
    - Az esemény nevét
    - A feltételt, ami az esemény kiváltásához szükséges
    - A végbemenő tevékenységeket

Állapotok

Egy állapot az objektum az attribútum- értékeinek és csatolásainak absztrakciójaként értelmezhető. Egyszerű esetben minden különböző attribútum érték külön állapotnak tekinthető. Általános esetben az attribútum értékek és csatolások halmaza képez egy állapotot.

Azon attribútumok, melyek az objektum viselkedését nem befolyásolják, melyek a vezérlésmintára nincsenek hatással, figyelmen kívül hagyhatók. (A vizsgálat szempontjából nem fontosak!)

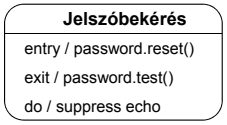
Az objektum viselkedését tekintve az állapot egy időtartamot jelent, mely két esemény között telik el.

Az események és az állapotok összetartoznak. Az események állapotokat határolnak, és fordítva, az állapotok eseményeket terminálnak.

Úgy az állapotok, mint az események az absztrakciós szint függvényei.

Állapot jelölése

- Fontosabb részei:
  - Név
  - Be- és kilépési tevékenység (entry/exit)
  - Belső tevékenység (do)



State Machine Diagram

Események

Az események a külvilágból érkező külső ingerek, melyek a rendszert „stimulálják”, melyekre a rendszernek meghatározott válaszokat kell adnia.

Az események lehetnek belső ingerek, melyeket a rendszert alkotó objektumok bocsátanak ki.

Az objektumok különböző eseményekre adott válaszai függenek azon objektum állapotától, mely az eseményt észleli.

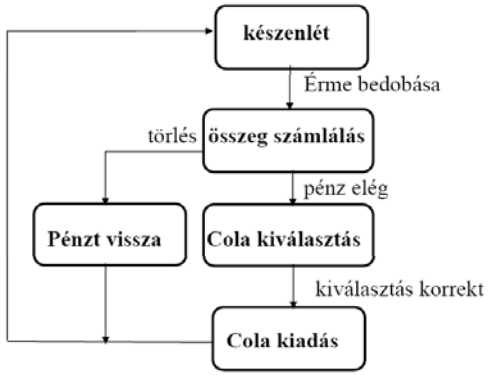
A reakció lehet:

- nincs állapotváltozás
- állapotváltozás
- esemény küldése az eredeti objektumnak
- esemény küldése egy harmadik objektumnak

Az események bekövetkezhetnek egy másik esemény előtt vagy után (okozati összefüggés). Lehetséges azonban, hogy az események nincsenek hatással egymásra (nincs okozati összefüggés). Az okozati összefüggés nélküli eseményeket párhuzamos eseményekként kezelhetjük. (Az események időpontokat reprezentálnak, időtartam hozzájuk nem rendelhető.)

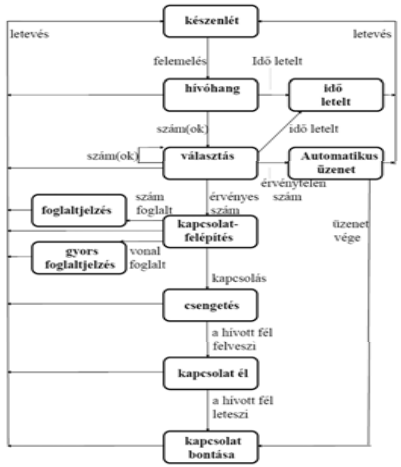
Egy esemény egy információ átvitelként is tekinthető egyik objektumtól egy másik objektumhoz.

Példa: egy egyszerű cola automata



Egy hagyományos példa:

a telefonálás



Feltételek megadása

Egy feltétel egy logikai függvény, mely igaz vagy hamis értéket vehet fel. A feltételek rendelhetők eseményekhez.

Az események feltételei mint „őr” viselkednek. Egy „őrzött” átmenet akkor aktív, ha amikor az esemény megtörténik, azzal egyidőben a feltétel is igaz.

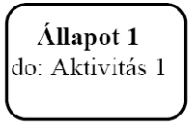
A feltételeket az eseménynév mögött [ ] jelek között adjuk meg.

Műveletek

Az események nem csak átmeneteket, hanem műveleteket is kiválthatnak.

Műveletek:

- Akció (nincs időtartama, eseményhez kapcsolódik)
  Jelölése:
  Esemény-1(Attribútum) [Feltétel-1]/Akció-1 →
- Aktivitás (időtartammal rendelkezik, állapothoz kapcsolódik)
  Jelölése:



Bemeneti és Kimeneti akciók

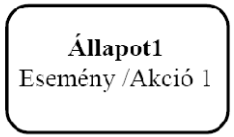
Lehetőség van arra, hogy akciókat egy állapotba való belépéssel, vagy abból történő kilépéssel kössünk össze.

Jelölés:

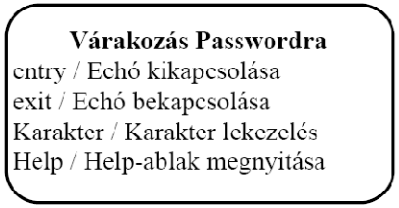
Belső akciók

Egy esemény akció végrehajtását kiválthatja anélkül, hogy az állapotot elhagyná. Ezen akciókat nevezzük belső akcióknak.

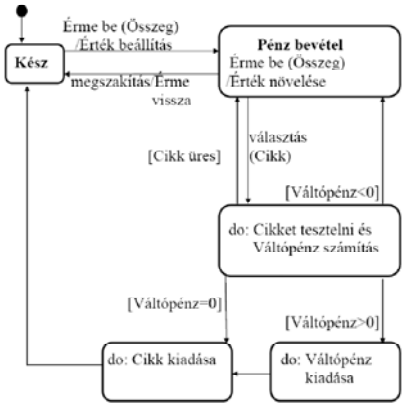
Jelölés:



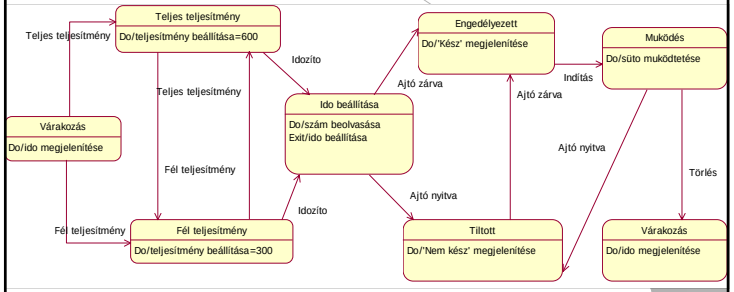
Egy példa: várakozunk a jelszóra



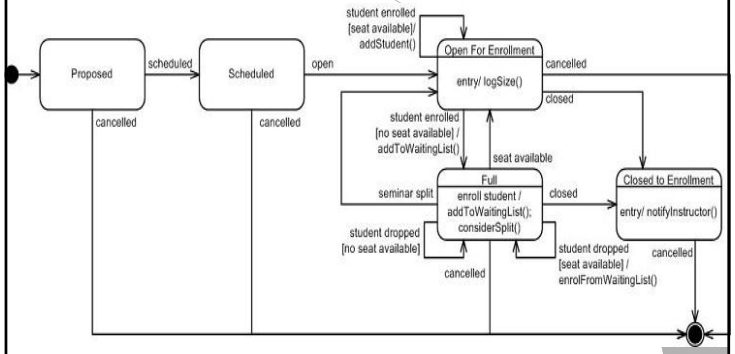
Példa: cola automata, még egyszer



TK példa – mikrohullámú sütő



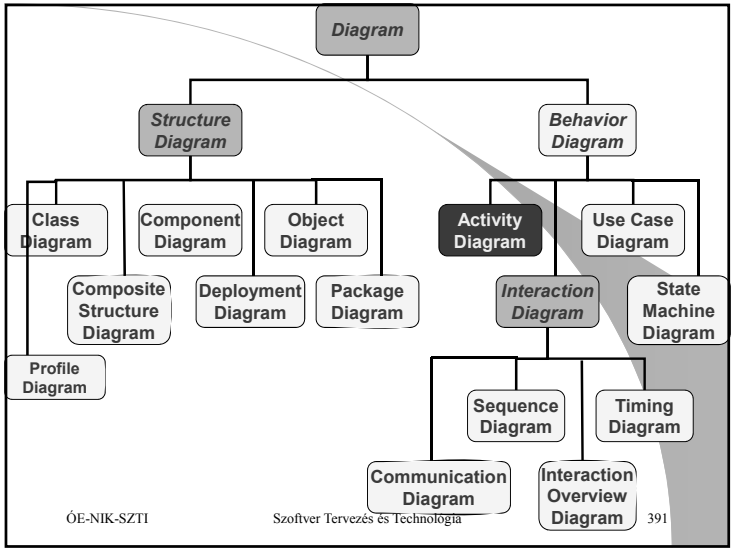
Tárgyjelentkezés



http://www.agilemodeling.com/artifacts/stateMachineDiagram.htm 2005. nov.

5.6.9 Activity Diagram

- Csomagdiagram (Package)
- Használati eset diagramja (Use Case)
- Osztálydiagram (Class)
- Objektumdiagram (Object)
- Szekvenciadiagram (Sequence)
- Kommunikációs diagram (Communication)
- Állapotdiagram (Statechart, State Machine)
- Aktivációs diagram (Activity)
- Komponensdiagram (Component)
- Telepítési diagram (Deployment)



Aktivációs diagram

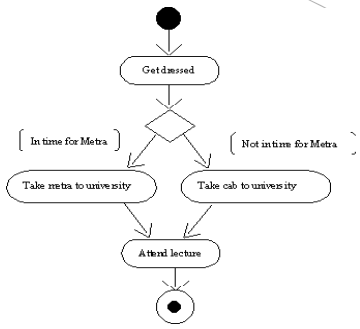
- Tevékenységek procedurális folyamatait modellezi
- Eljárások szekvenciális és konkurens lépéseit írja le
- Egy tevékenység összes lehetséges folyamatát mutatja
- Használható:
  - Használati esetek részletezésére
  - Rendszerszintű funkciók modellezésére
- Fókuszál:
  - A tevékenységek sorrendjére
  - A feltételekre, melyek kiváltják a tevékenységeket

Jelölések

- Tevékenységi állapot: objektum tevékenységét jelöli
- Átmenet ⟶
- Kezdeti állapot ○
- Végállapot ◎
- Szinkronizációs vonal: párhuzamos folyamatok elején és végén használjuk ▬
- A tevékenységek oszlopokba rendezhetők objektumok vagy feladatok szerint

Activity Diagram

Hogyan jutunk a főiskolára

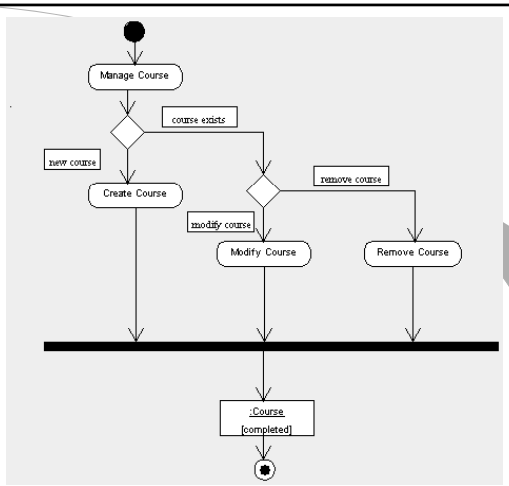


Metra: a HÉV Chicagó környékén

http://www.developer.com/design/article.php/2247041 2005. november

Activity Diagram

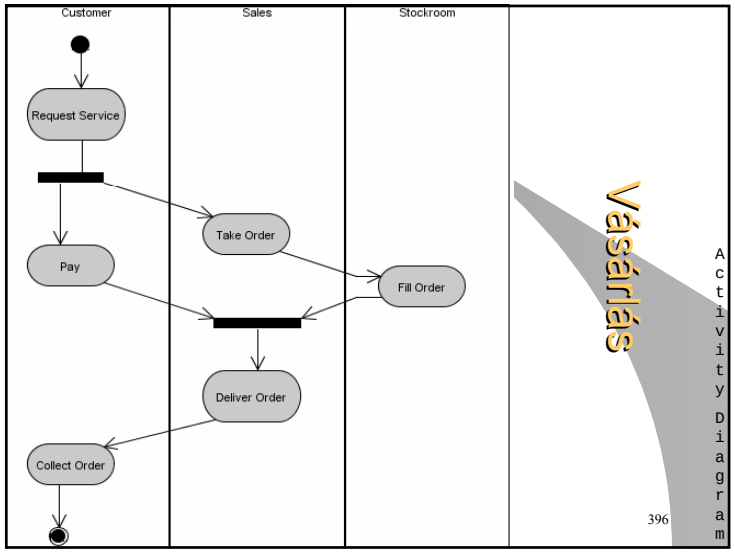
Kurzusmenedzselés



http://www.developer.com/design/article.php/10925_2247041_2 2005. november

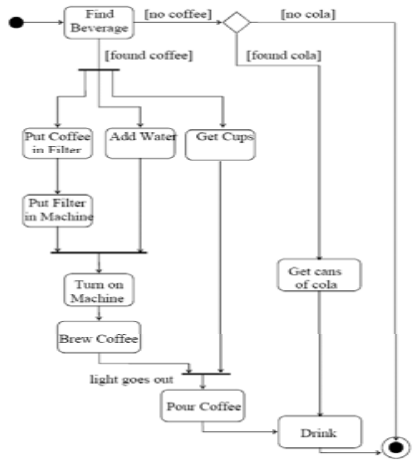
Activity Diagram

Vásárlás



Activity Diagram

Példa: kávé-főzés

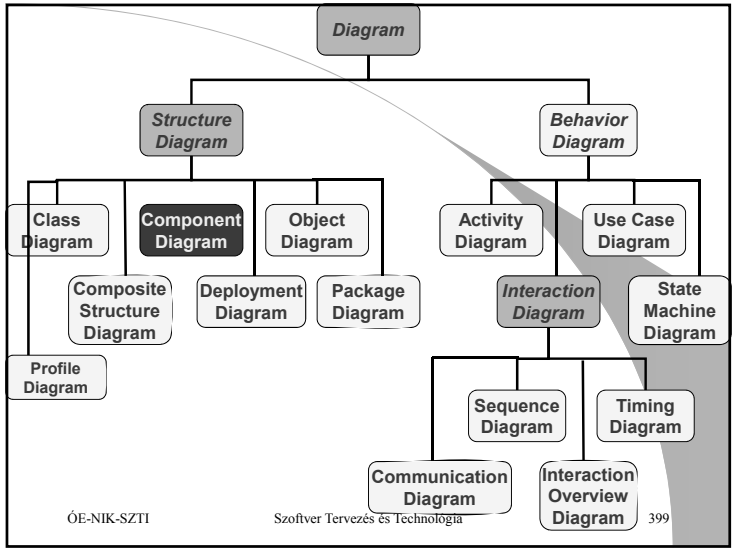


Activity Diagram

http://www.objectmentor.com/resources/articles/cplxtrns.pdf 2005. november

5.6.10 Component Diagram

- Csomagdiagram (Package)
- Használati eset diagramja (Use Case)
- Osztálydiagram (Class)
- Objektumdiagram (Object)
- Szekvenciadiagram (Sequence)
- Kommunikációs diagram (Communication)
- Állapotdiagram (Statechart, State Machine)
- Aktivációs diagram (Activity)
- Komponensdiagram (Component)
- Telepítési diagram (Deployment)



Komponensdiagram

Története:

A komponensek lényege új irányt vett az UML 2.0-ban az addigi fizikai szemlélethez képest. Itt a komponensek az elemek fizikai fogalmától elkülönültek, fogalmi modellként használhatók. Halvány különbség van a struktúrált osztály és a komponens között.

Component Diagram

Rumbaug, Jacobson, Booch: The Unified Modeling Language Reference Manual

Komponens

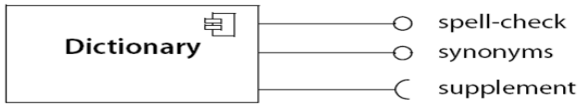
- A fizikai és logikai rendszer azon moduláris részeit írja le, amelyek kifelé látható viselkedése jobban leírható, mint a megvalósításuk.
- A kifelé látható viselkedéseket interfészek halmaza reprezentálja.
- Két nézőpontot mutatnak:
  - Definiálják a rendszer részeinek külső arculatát
  - Megvalósítják a rendszer funkcionalitását

Komponens jelölése

- Két nézet
  - Téglalap és a <<component>> kulcsszó
  - Téglalap benne a komponens ikonnal (az UML 1-es verzióiban ez jelölte a komponenst a téglalap helyett)

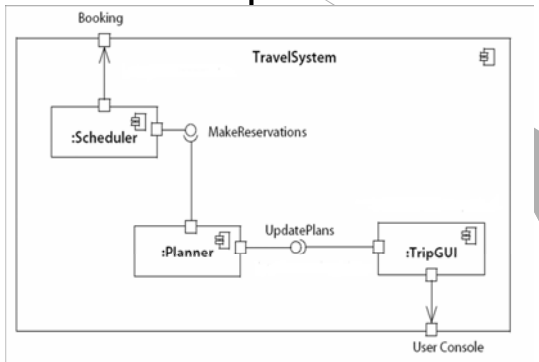
Rumbaug, Jacobson, Booch: The Unified Modeling Language Reference Manual

Interfészek



- Interfészek
  - Provided interface: olyan összetartozó szolgáltatások halmaza, amelyeket a komponens elérhetővé tesz
  - Required interface: a komponensnek meg kell kapnia a szolgáltatást

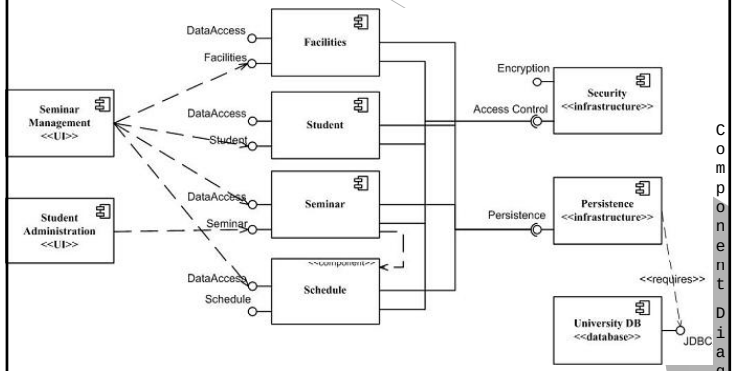
Az utazáskomponens belső komponensei



A kis négyzetek a komponensek határain portot jelölnek.

Rumbaug, Jacobson, Booch: The Unified Modeling Language Reference Manual

Iskolai szoftver komponensei

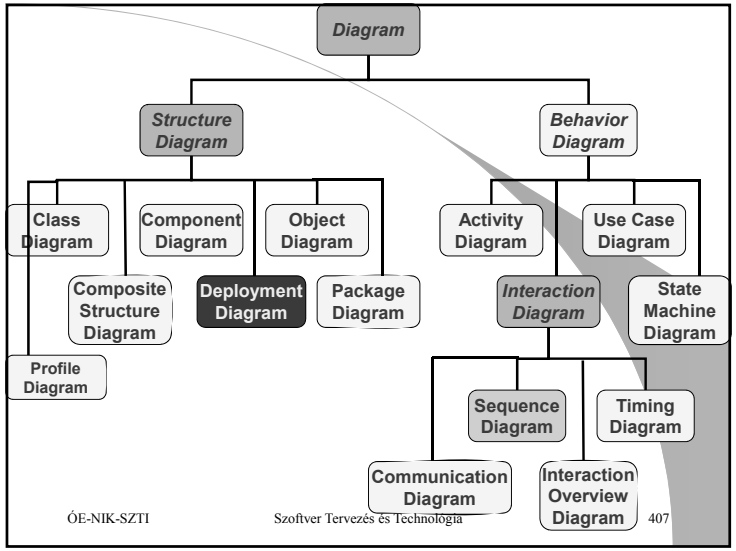


Component Diagram

Rumbaug, Jacobson, Booch: The Unified Modeling Language Reference Manual

5.6.11 Deployment Diagram

- Csomagdiagram (Package)
- Használati eset diagramja (Use Case)
- Osztálydiagram (Class)
- Objektumdiagram (Object)
- Szekvenciadiagram (Sequence)
- Kommunikációs diagram (Communication)
- Állapotdiagram (Statechart, State Machine)
- Aktivációs diagram (Activity)
- Komponensdiagram (Component)
- Telepítési diagram (Deployment)

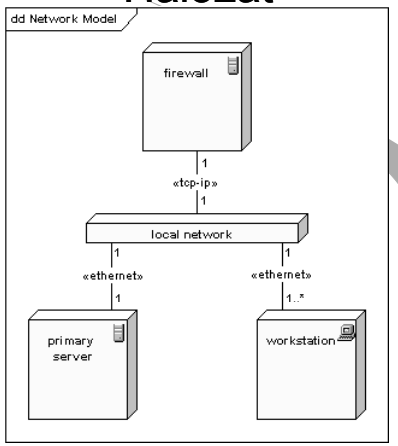


Deployment Diagram

- A rendszer futási idejű architektúráját modellezi
- A hardverelemek (nodes) konfigurációját mutatja, valamint a szoftverelemek leképzését node-okra
- Node:
  - Erőforrást modellez (pl. számítógép, lemezmeghajtó)
  - Eszközök és futtatható környezetek ilyenek
  - Legalább memóriával, gyakran feldolgozó képességgel rendelkeznek

Deployment Diagram

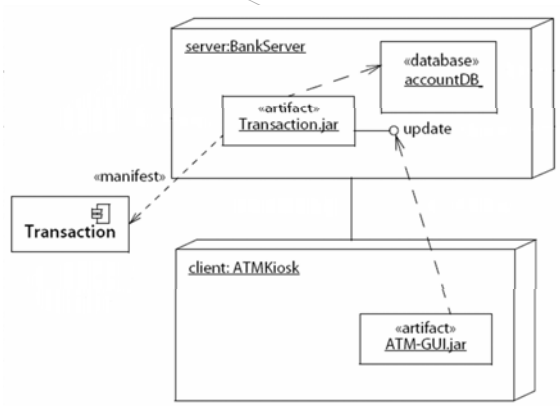
Hálózat



Deployment Diagram

http://sparxsystems.com.au/resources/uml2_tutorial/uml2_deploymentdiagram.html 2005. nov.

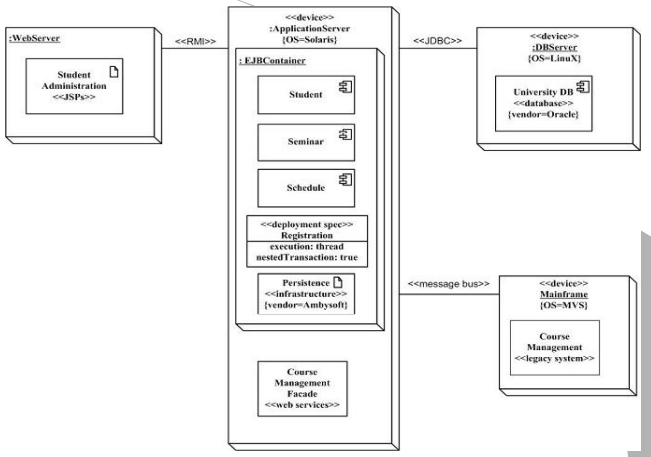
ATM



Deployment Diagram

Rumbaug, Jacobson, Booch: The Unified Modeling Language Reference Manual

Iskolai szoftver



Deployment Diagram

http://www.agilemodeling.com/artifacts/deploymentDiagram.htm 2005. november

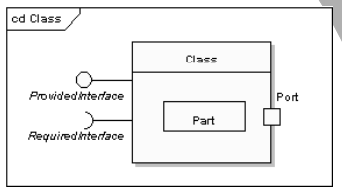
5.6.12 További diagramok

Az UML 2.x-ben újdonság:

- Composite Structure Diagram
- Timing Diagram
- Interaction Overview Diagram
- Profile Diagram

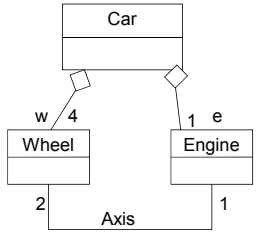
Composite Structure Diagram

- Struktúra diagram
- Az osztály, komponens stb. belső struktúráját és a rendszer más részeihez való kapcsolódását mutatja
- Az osztálydiagrammal ellentétben az osztályt alkotó elemek összetételét mutatja:
  - Interfészek
  - Portok
  - Részek
  - Kollaboráció

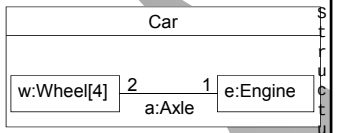


Autó és részei

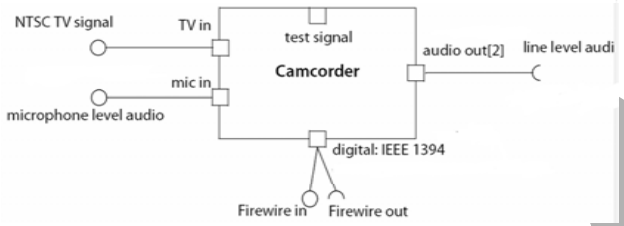
UML1.x-ben osztálydiagrammal



UML2.0-ban composite structure diagrammal



Camcorder strukturált osztály portokkal

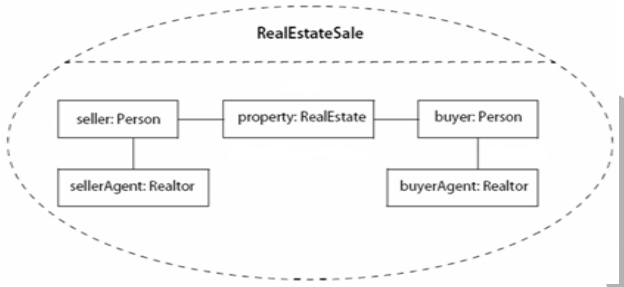


A négyszög belsejében lévő port (test signal) a port rejtettségre utal (a láthatósága private).

Kollaboráció -- Együttműködés

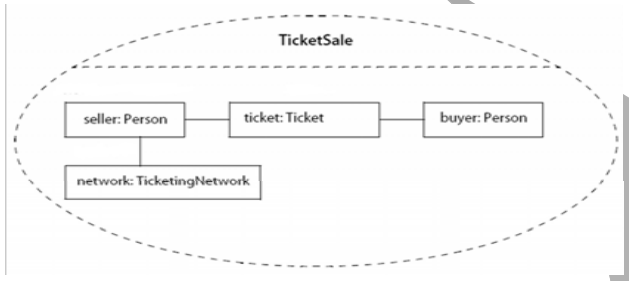
- Semmi köze a Collaboration Diagramhoz, ami az UML1.x-ben volt
- Statikus struktúra
- Együttműködő részek struktúráját mutatja, amelyek együttesen látnak el kívánt feladatokat
- Gyakran patternt valósít meg
- Jelölése: szaggatott ellipszis a kollaboráció nevével

Lakásvásárlás



Composite Structure Diagram

Jegyeladás



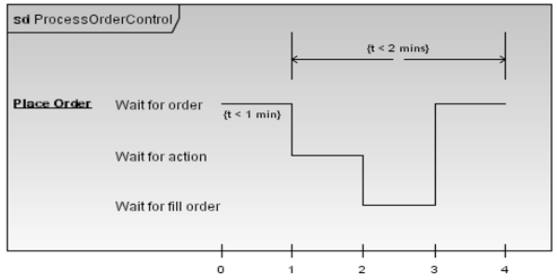
Composite Structure Diagram

Timing Diagram

- Viselkedési interakció diagram
- A szekvenciadiagram bemutatásának alternatív módja
- Explicit mutatja egy életvonalon az állapotokban bekövetkezett változásokat
- Valósidejű alkalmazásoknál hasznos

Timing Diagram

Rendelés

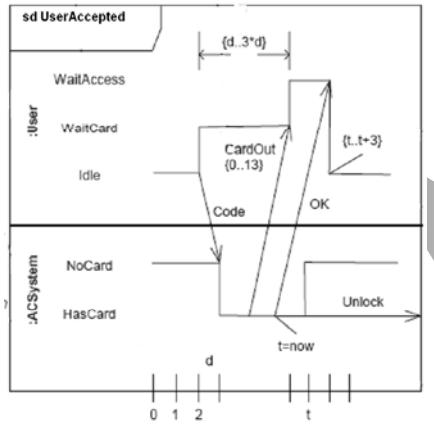


Timing Diagram

Különbség a szekvenciadiagramhoz képest

- A tengelyen balról jobbra az idő nő
- Az életvonalak függőlegesen, elkülönült részekben találhatók
- Az életvonal le-fel ugrál, az állapotok változását mutatva
- Mindenegyes függőleges helyzet külön állapotot jelent
- Az állapotok sorrendje nem feltétlen bír jelentéssel

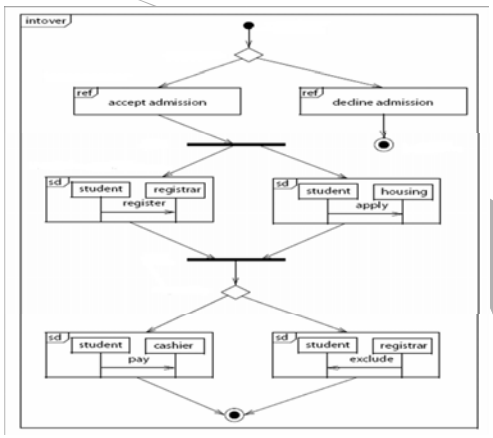
Pénzfelvétel



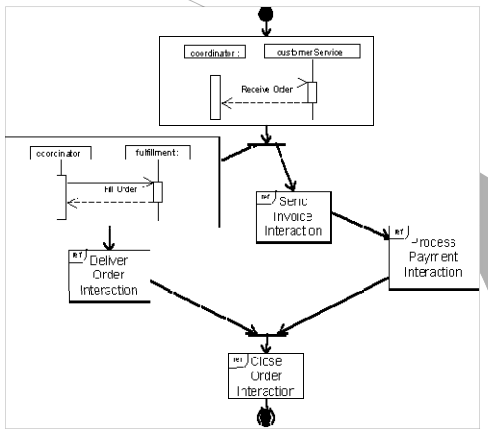
Interaction Overview Diagram

- Viselkedési interakciós diagram
- Az aktivációs diagram variációja, mely magában foglalja a szekvenciadiagramot
- A szekvenciadiagram jelölését használja az aktivációs diagramból vett döntésekkel és elágazásokkal

Felvétel főiskolára



Rendelés



Interaction Overview Diagram

Profile Diagram

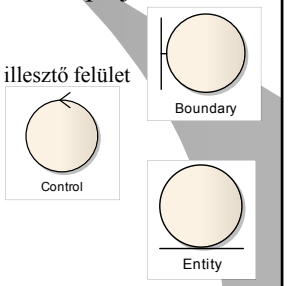
- A profil diagram az UML különböző platformokra való testreszabását segíti elő
- Általános kiterjesztést tesz lehetővé
- A testreszabáshoz
  - sztereotípiákat,
  - tag-eket,
  - megszorításokat definiálhatunk.

Sztereotípia

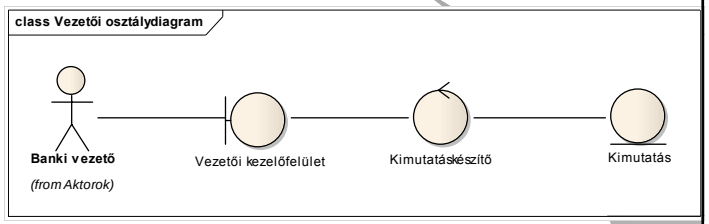
- Lehetővé teszi a tervező számára, hogy az UML nyelvezetét kiterjessze
- Így a már létezőkből új modellelemek definiálhatók
- Amelyek speciális tulajdonságot határoznak meg egy adott probléma megoldásához
- Jelölése: << és >> közé írt szó

Osztályok sztereotípiái

- A sztereotípiák legtipikusabb példája, hogy az analízis fázisában az osztályokat nem részletezve adjuk meg, hanem megadjuk az osztály jellegét, más néven sztereotípiáját:
  - Boundary
    - Határosztály
    - A rendszer és az aktorok közötti illesztő felület
  - Control
    - Vezérlő osztály
    - Más objektumok vezérlése
  - Entity
    - Egyed osztály
    - Információ tárolása

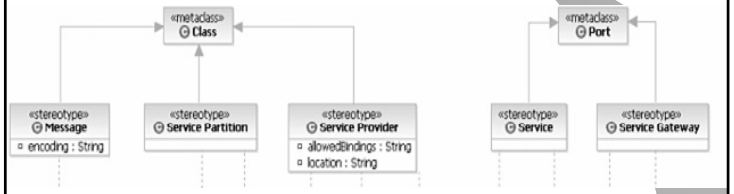


Sztereotípiával megadott osztályok



Profil diagram

- Az UML 2.2-ben a testreszabásokat egy profilban kell definiálni
  - Minden sztereotípiát egy profil elemben határozunk meg



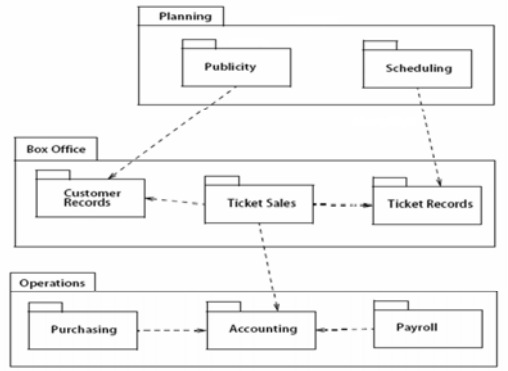
Összefoglaló feladat

- Rumbaugh, Jacobson, Booch: The Unified Modeling Language reference Manual, 2004
  3. UML Walkthrough fejezet alapján
- A fejezet célja egyszerű példán keresztül bemutatni a magas szintű UML fogalmakat
- Az összefoglaló nem törekszik teljeskörű bemutatásra

Feladat

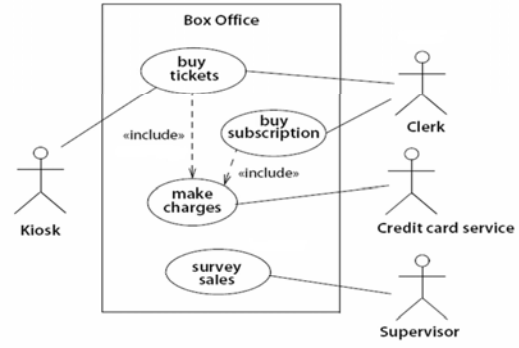
Készítsünk szoftvert egy színházi jegypénztárnak, amely számítógépre szeretné vinni az általa végzett tevékenységeket

Package Diagram

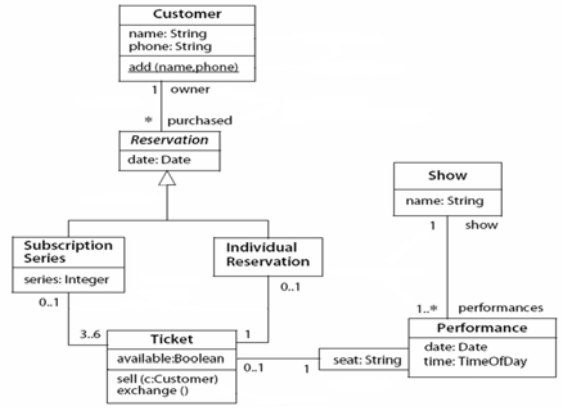


Az ábra a teljes színházi rendszer csomagokra bontását tartalmazza a függőségi kapcsolatokkal együtt.

Use Case Diagram

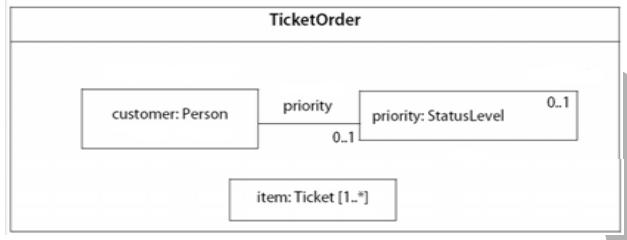


A képen a jegypénztár használati eset diagramja látható.

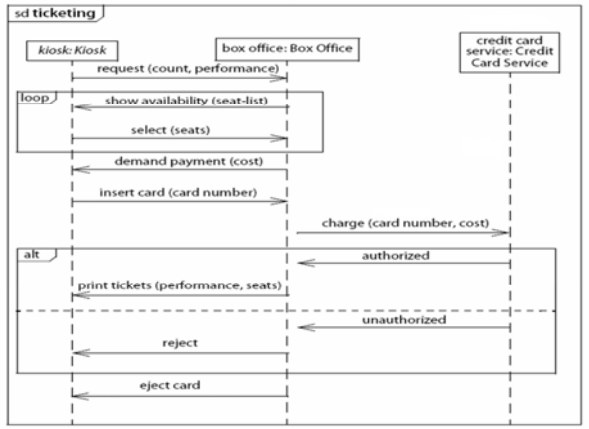


A képen a jegypénztár osztálydiagramja látható.

Composite Structure Diagram

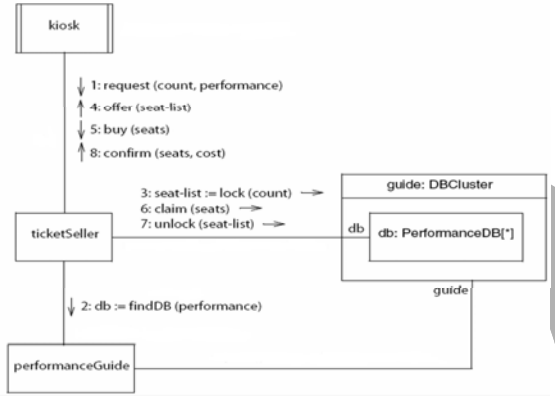


Sequence Diagram



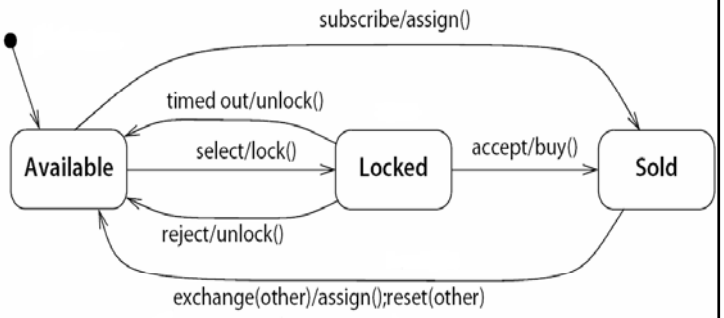
A jegyvásárlás használati eset szekvenciadiagramja

Communication Diagram



A jegyeladás részletének kommunikációs diagramja a fejlesztés egy későbbi szintjén.

State Machine Diagram

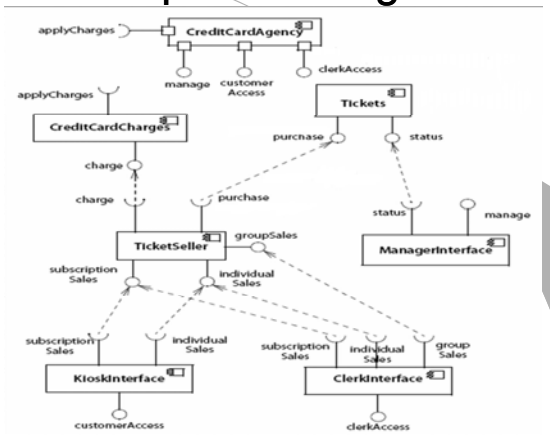


A jegy életciklusának állapotdiagramja.

Activity Diagram

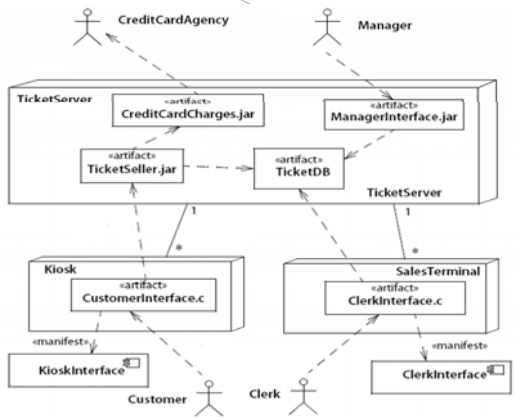
A mű színpadra állításának aktivitás diagramja.

Component Diagram



A hitelkártyával történő jegyvásárlás komponensei.

Deployment Diagram



6. The Unified Software Development Process & Rational Unified Process (RUP)

Tartalom

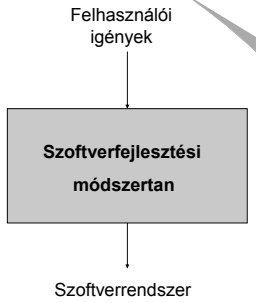
- Mi a szoftverfejlesztési módszertan?
- Unified Process
- Rational Unified Process (RUP)
- Példa

Szoftver = Termék

- Van szolgáltatási funkciója
- Van előállítási költsége
- Van előállítási határideje
- Van minősége

Az előállításához technológiára van szükség

Szoftverfejlesztési módszertan



Szoftverfejlesztési módszertan

- Felhasználói igényeket transzformál szoftverrendszerré
- A jó módszertan
  - Általános (többféle feladat esetén alkalmazható)
  - Fejlődőképes (alkalmazkodik a változásokhoz)

Szoftverfejlesztési módszertan

- Meghatározza, hogy kinek, mit, mikor kell csinálnia ahhoz, hogy elérhesse a megadott célt
- A módszertan alapjai:
  - Technológiák
  - Eszközök
  - Emberek
  - Szervezetszintű munkamenet

Szoftverfejlesztési módszertan

- Technológiák
  - A korra jellemző technológiák meghatározzák az eljárás hatékonyságát
    - Programozási nyelvek
    - Fejlesztői környezetek
    - Operációs rendszerek
    - Hálózati lehetőségek
    - …
- Eszközök
  - A fejlesztéshez használt eszközök befolyásolják a hatékonyságot
  - Az eljárás és az eszközök párhuzamosan fejlődnek

Szoftverfejlesztési módszertan

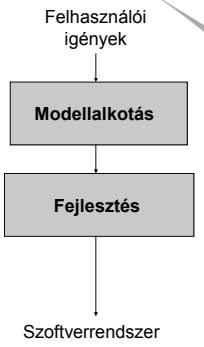
- Emberek
  - A fejlesztési folyamatban résztvevők képzettsége befolyásolja a hatékonyságot
    - Aktuális tudás
    - Rövid távon megszerezhető tudás
  - A fejlesztés bizonyos lépéseinek számítógépes támogatásával csökkenthetjük a képzettség-igényt
- Szervezetszintű munkamenet
  - A szoftverfejlesztők nem egyedül dolgoznak
  - A módszertannak illeszkednie kell a napi valósághoz, a tényleges munkamenethez

Szoftverfejlesztési módszertan

„Az emberi tevékenységek során kialakult az a gyakorlat, hogy a probléma megoldását először egy a probléma számára alkalmas rendszerben konstruáljuk meg, azaz létrehozzuk a megoldás absztrakt modelljét, és csak azután kerül sor a megoldás realizálására. Például lerajzoljuk a megépítendő házat, és csak azután fogunk hozzá annak felépítéséhez.”

Sike Sándor, Varga László:
Szoftvertechnológia és UML

Szoftverfejlesztési módszertan



Unified Process

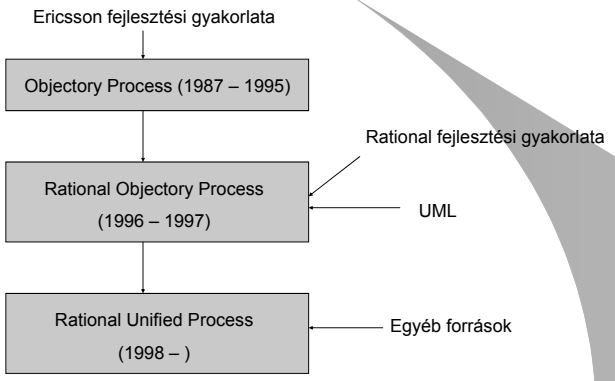
- **Egyesített:** a három legelterjedtebb eljárás egyesítésével jött létre
  - Jacobson
  - Booch
  - Rumbaugh
- **Egységesített:** egységes jelölésmód az egész világon

A tervezés során az UML diagramjait használja.

Rational Unified Process (RUP)

- A Unified Process-re épül
- A Rational (később IBM) cég fejlesztési módszertana
- A módszertan elvek, módszerek és fejlesztést támogató eszközök egysége

Történelmi áttekintés



Történelemi áttekintés

- 1991 – Object Modeling Technique
  (J. Rumbaugh – General Electric)
- 1991 – Booch Method
  (G. Booch – Rational Software)
- 1992 – Object-Oriented Systems Engineering
  (I. Jacobson – Ericsson)
- 1994 – J. Rumbaugh → Rational Software
- 1995 – I. Jacobson → Rational Software
- 1997 – UML
- 1998 - RUP

UML és RUP

- **G. Booch, J. Rumbaugh, I. Jacobson:**
  The Unified Modeling Language User Guide
- **J. Rumbaugh, G. Booch, I. Jacobson:**
  The Unified Modeling Language Reference Manual
- **I. Jacobson, J. Rumbaugh, G. Booch:**
  The Unified Software Development Process

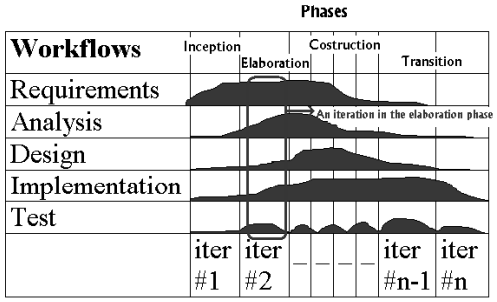
A RUP jellemzői

- **Komponensalapú**
  - a szoftvert komponensekből építi fel
  - a komponensek a megfelelő interfészeken keresztül kommunikálnak egymással
  - a rendszer funkcionalitása különféle komponensek hozzáadásával könnyen alakítható
- **Modellszemléletű**
  - a rendszert különféle modelleken keresztül közelíti meg
- **Használati eset vezérelt**
  - a fejlesztés középpontjában a megrendelővel egyeztetett use case-ek állnak
  - a használati esetek pontos felmérése, majd megvalósítása elengedhetetlen a projekt sikeréhez
- **Architektúra-centrikus**
  - Kiemelt hangsúlyt kap a rendszer architektúrája, az egységbezárás és a laza csatolás általi felépítés.
- **Iteratív és inkrementális**

A RUP felépítése

- Fázisok
  - Előkészítés *(Inception)*
  - Kidolgozás *(Elaboration)*
  - Építés *(Construction)*
  - Átadás *(Transition)*
- Munkamenetek
  - Követelmények meghatározása *(Requirements)*
  - Analízis *(Analysis)*
  - Tervezés *(Design)*
  - Fejlesztés *(Implementation)*
  - Tesztelés *(Test)*
  - ...

A RUP felépítése



Előkészítés / Inception

- Követelmény-feltáráson van a hangsúly
  - Áttekintés (Use-Case modell)
  - Szójegyzék
- Költségek, erőforrások, határidők meghatározása
  - Ütemterv
  - Erőforrás-felhasználási terv
- Kockázatok elemzése
  - Kockázatlista
- Döntés születik a megvalósíthatóságról
  - Prototípus (kezelőfelület)

Kidolgozás / Elaboration

- Az analízisen van a hangsúly
  - Kritikus funkcionalitás
  - Szerkezeti elemek
- A rendszer struktúrájának meghatározása
  - Analízis modell
  - Design modell
  - Telepítési modell
  - Implemetációs modell,
- Az építés fázis menetének megtervezése
  - Iterációs terv
- A tesztelés megtervezése
  - Teszt modell

Építés / Construction

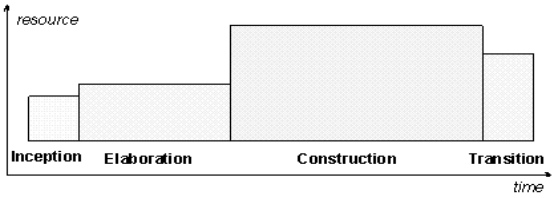
- Az összes funkcionalitás kifejlesztése
  - Működő prototípus
- A program tesztelése
  - Alfa teszt
  - Béta teszt
- Felhasználói leírás készítése
  - Felhasználói kézikönyv
- Az átadás megtervezése
  - Iterációs terv

Átadás / Transition

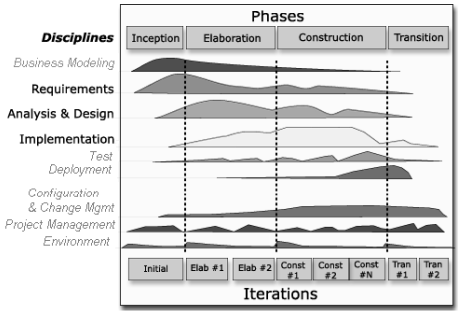
- A program beillesztése a felhasználási környezetébe
- Felhasználói elégedettség mérése
- Költségek ellenőrzése
- A projekt lezárása
- A követés megkezdése

A fázisok idő- és erőforrásigénye

	Előkészítés	Kidolgozás	Megvalósítás	Átadás
Erőfeszítés	**~5%**	**20%**	**65%**	**10%**
Idő	**10%**	**30%**	**50%**	**10%**



A RUP felépítése (teljes üzleti folyamat)



A RUP modelljei

- Use-Case modell
- Analízis modell
- Design modell
- Telepítési modell
- Implementációs modell
- Teszt modell
- (Adat modell)

Use-Case modell

- A felhasználó igényeinek leírása
- Használati eset diagramok
  - Felhasználók (aktorok)
  - Szolgáltatások (használati esetek)
  - Használati esetek leírása (forgatókönyv)
- Az egyes diagramok csoportosítása (csomagok)

Analízis modell

- A Use-Case modell részletezése
- Osztály diagramok
  - Osztályok és kapcsolataik realizálják a szolgáltatásokat
- Kollaborációs diagramok
  - A kommunikációs folyamatok realizálják az egyes forgatókönyveket

Design modell

- A Use-Case modell megvalósítása
- Az Analízis modell alapján készül
- Osztály diagramok
  - Az Analízis modell osztályaiból kiinduló részletesebb felbontás
  - Adattagokkal és műveletekkel bővítve
- Szekvencia diagramok
  - A kollaborációs diagramok alapján készül
  - A Design modell osztálydiagramjaira épül
- Állapotdiagram
  - Állapotok
  - Állapot-átmenetek

Telepítési modell

- A Use-Case modell elosztása
- A Design modell alapján készül
- Deployment diagram
  - Csomópontok
  - Hálózati kapcsolatok
- Az alrendszerek (csomagok) megkönnyítik a modell felosztását

Implementációs modell

- A Use-Case modell implementációja
- A Design modell alapján készül
- Komponens diagramok
  - A Design modell osztályainak megvalósítása az egyes komponensekben

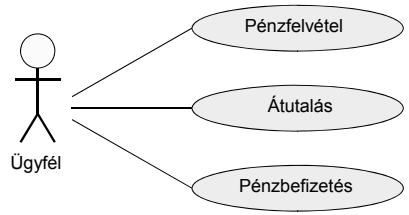
Teszt modell

- A Use-Case modell tesztje
- A Use-Case modell alapján készül
- Use-Case jellegű modell
  - Teszt esetek
  - Teszt folyamatok
- Az egyes szolgáltatásokhoz kell teszt-eseteket készíteni

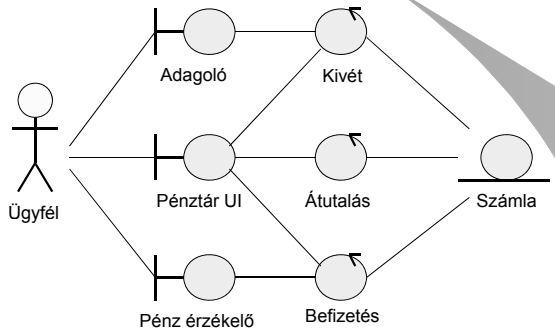
Példa

- Tervezzük meg egy ATM vezérlőszoftverét!
- Funkciói:
  - Pénzfelvétel
  - Pénzbefizetés
  - Átutalás

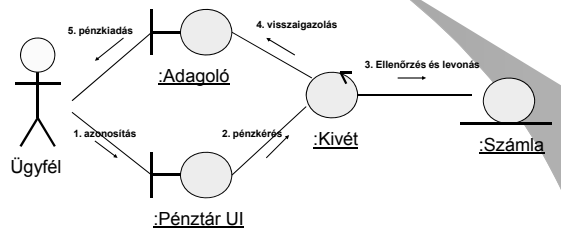
Use-Case modell



Analízis modell (osztály diagram)

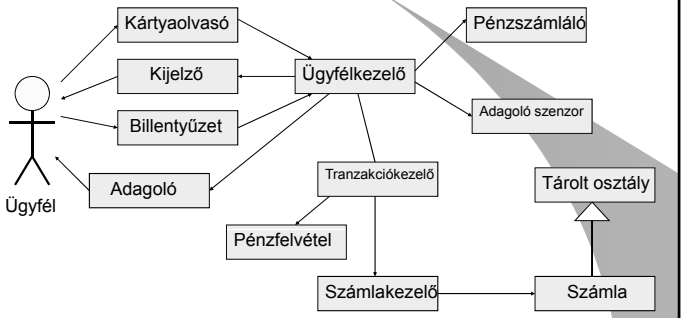


Analízis modell (kommunikációs diagram)



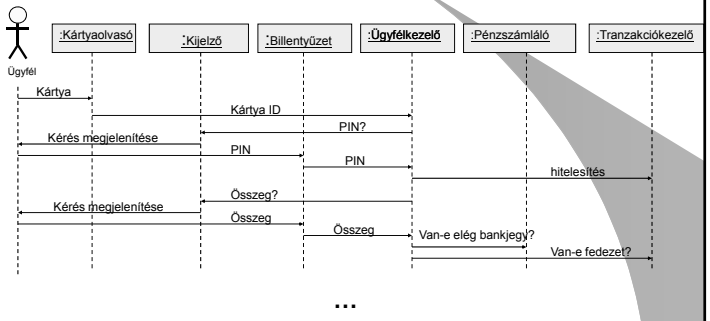
A pénzfelvétel use-case kommunikációs diagramja

Design modell (osztálydiagram)

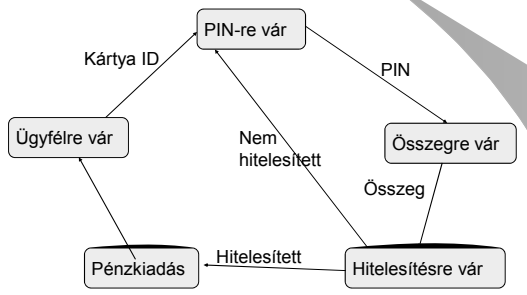


A pénzfelvétel use-case osztálydiagramja

Design model (szekvenciadiagram)

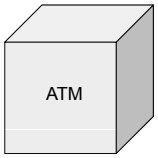


Design modell (állapotdiagram)



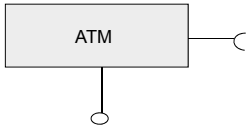
Az ügyfélkezelő osztály állapotdiagramja

Telepítési modell



A program egyetlen csomóponton működik

Implementációs modell



A rendszer egyetlen komponensből áll