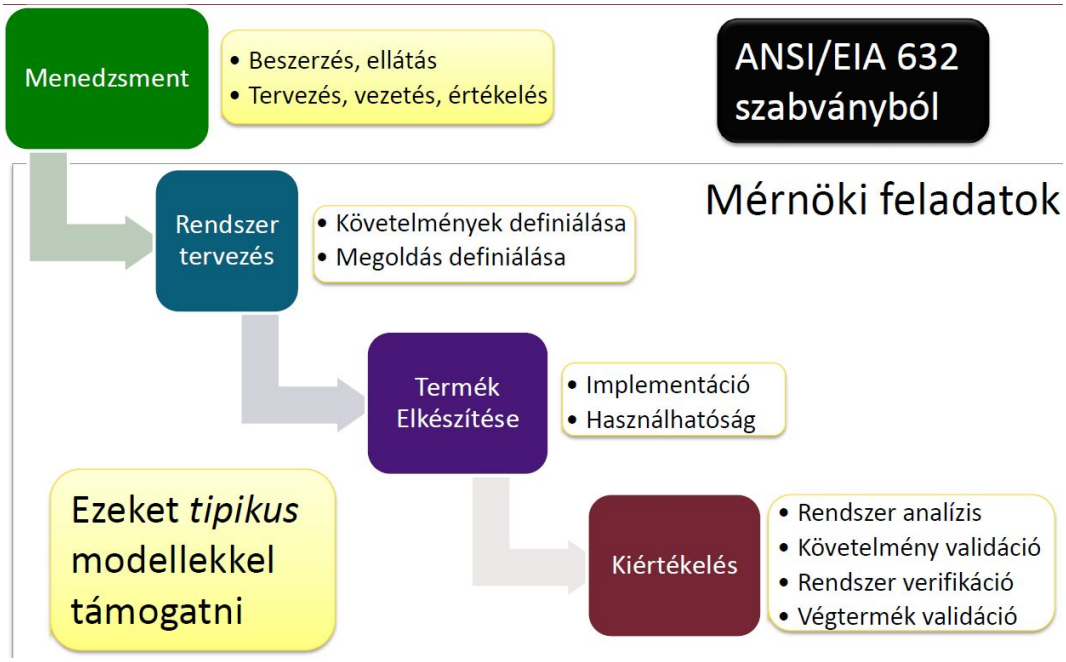
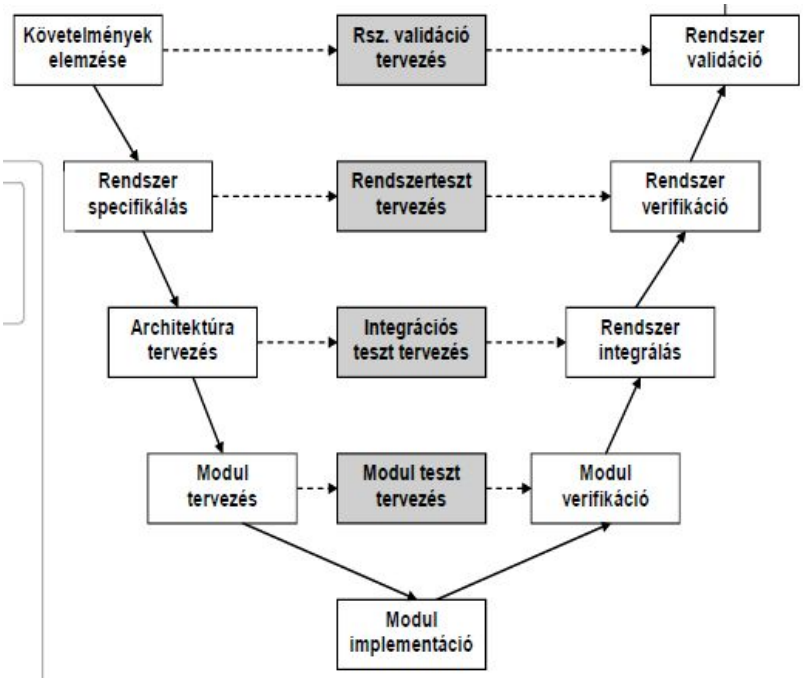
Rendszertervezési folyamat (36.)

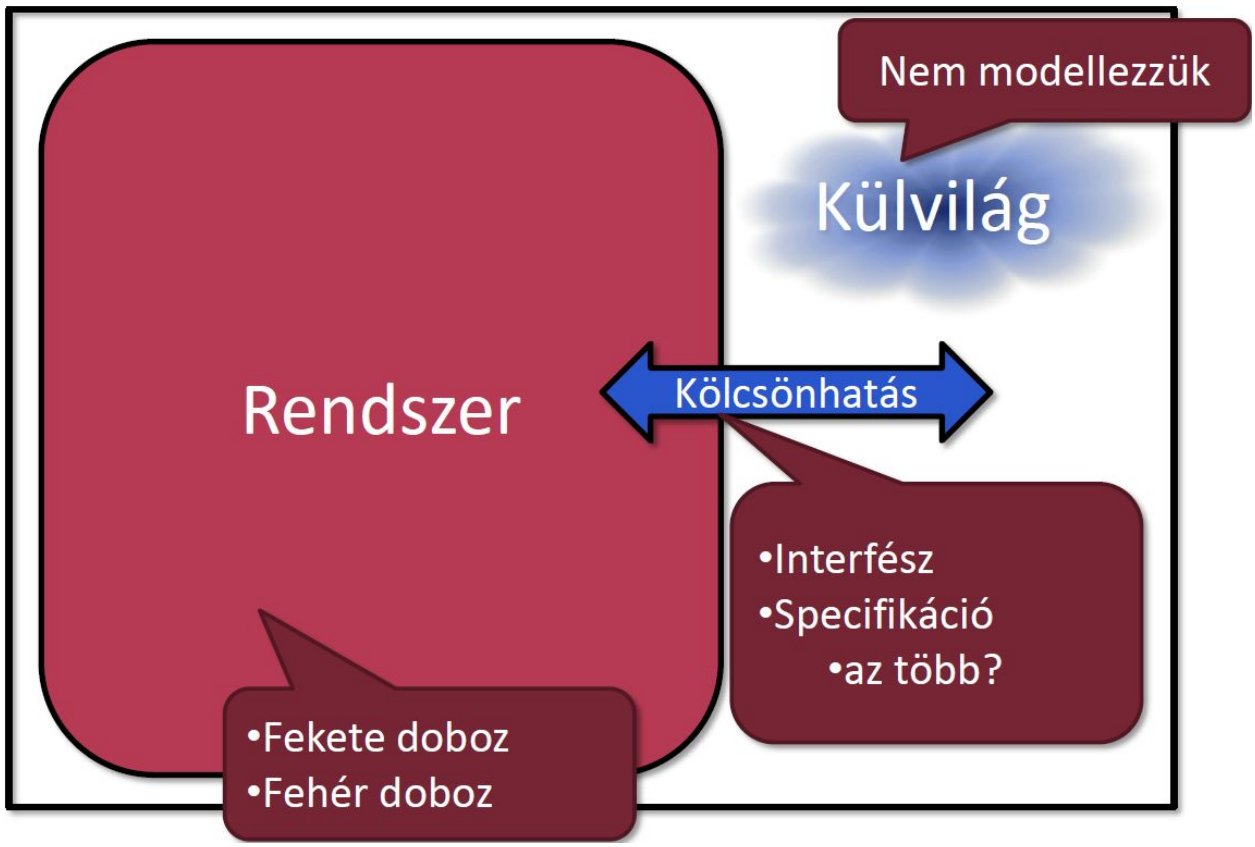


Rendszer tervezés folyamatai (47.)



rendszer és külvilág (52.)

fekete doboz - csak a kimenetet ell.
fehér doboz - magát a megvalósítást is tudjuk



Osztályozási szempontok (61-74)

- Felépítési vs. viselkedési modellek
  - (felépítési:) Statikus
  - (felépítési:) rész-egész, összetevők
  - (felépítési:) kapcsolatok, összeköttetések
  - (felépítési:) PÉLDÁK:

struktúra, keretszerkezet, Tartalmazási hierarchia, Szervezeti felépítés (ld. tartalmazási hierarchia), Architektúra modell (blokkdiagram), Taxonómia jellegű, Ontológia

  - (viselkedési:) dinamikus
  - (viselkedési:) időbeli lefolyás
  - (viselkedési:) állapot, folyamat
  - (viselkedési:) reakciók a külvilágra

- (viselkedési:) PÉLDÁK:

szekvencia diagram, activity diagram, állapottérkép, vezérlési folyam, adatfolyamháló, Petri-háló, Sztochasztikus (Markov-) modellek

- Matematikai-formális vs. informális
  - mennyi matematikai kifejezést tartalmaz?
  - nem válik szét teljesen
  - nem biztos, hogy mindig a szigorúbb a jó

- Folytonos vs. diszkrét
  - Értékben
  - Időben

- Végrehajtható vs. deklaratív (tipp: sorra kevésbé végrehajtható)
  - Teljes eseménysor determinisztikusan rekonstruálható
  - Eseménysor sztochasztikusan definiált
  - Nemdeterminisztikusan végrehajtható
  - Részben korlátozza a lehetséges eseményteret

Pl. csak ellenőrizhető kritériumok

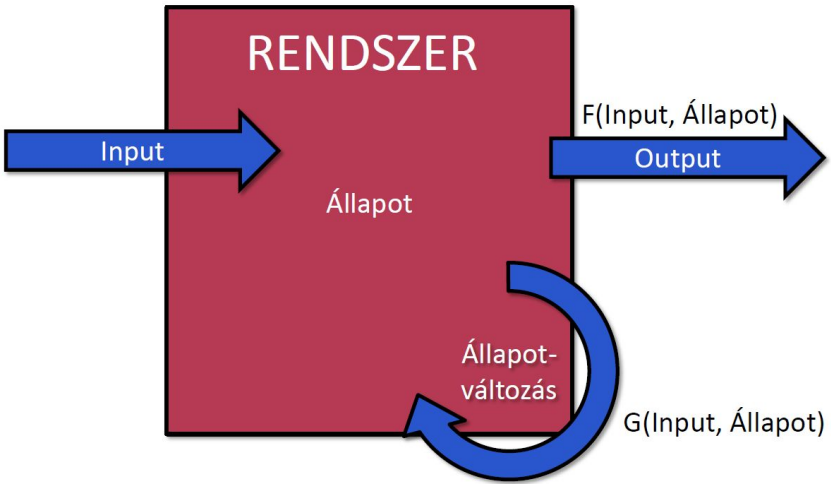
  - Nem is viselkedési modell

Viselkedésmodellek ábra (79, 89) részei: esemény alapú modell

folyamat alapú modell

állapot alapú modell

Klasszikus rendszerelméleti automata-modell



diszkrét eseménysor (91)

- **Esemény**
  - pillanatszerű változás (a rendszerben vagy ki/bemeneten)
- **Eseményfolyam**
  - Pl. input/output adatforrás
- **Eseménytér** – {megengedett események}
  - Beolvasható input értékek, kibocsátható output értékek
- Pillanatszerű események sora, egyszerre ≤ 1

Eseményfolyam: toronyóra ütései

Eseménytér: {*éjfél*, *dél*}

(94-100)
állapottér - egymástól megkülönböztetett rendszerállapotok halmaza; minden időpontban pontosan egy eleme jellemzi a rendszert az állapottér egy halmazának.
pillanatnyi állapot - egy adott időpontben a rendszer ~-a az állapottér azon egyetlen eleme, amelyik abban az időpontban jellemző a rendszerre.
teljesség - mindig legalább az egyik állapot fennáll
kölcsönös **kizárólagosság** - egyszerre csak egy állapot állhat fenn

Állapotfinomítás/absztrakció - az állapottéren végzett halmazfinomítás/absztrakció
Állapotfinomítás - tervezés előrehaladása, több megvalósítási részlet
Specializáció / Kiegészítés
Több rendszer együttes vizsgálata
Több információ, több tudás (nem mindig jó)
Állapotabsztrakció - hasznos, ha az absztakt állapotok

- "egységesek" majdnem ekvivalensek
- valamilyen szempontból egyformák az összevont állapotok
- Ld. "helyettesítési tulajdonságú partíció ->Digit

bizonyos feladatokra kevesebb információ is elég

- kisebb, egyszerűbb állapottérrel könnyebb tervezni

állapotmentes modell (|S|=1)
néha csak kevesebb információt szabad felfedni
gyakori formája: dekompozíció

Állapotterek (direkt) szorzata (101)

- Két állapottér együttes figyelembevétele
  - $S_1 = \{teljes\ fokozat, olvasztó\ mód, kikapcsolva\}$
  - $S_2 = \{nyitva, zárva\}$

$S_1 \times S_2$	*nyitva*	*zárva*
teljes	*teljes* és *nyitva*	*teljes* és *zárva*
olvasztó	*olvasztó* és *nyitva*	*olvasztó* és *zárva*
kikapcsolva	*kikapcsolva* és *nyitva*	*kikapcsolva* és *zárva*

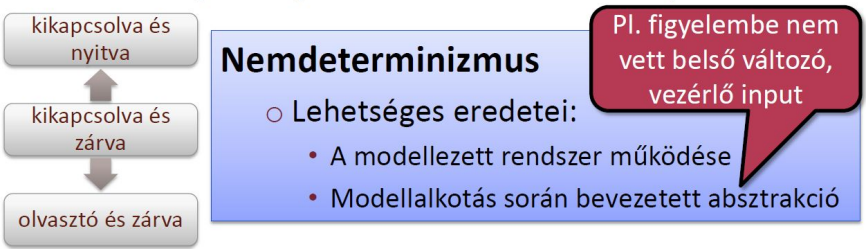
állapotabsztrakció

állapotabsztrakció (vetítés)

Észrevétel: $|S_1 \times S_2| = |S_1| \cdot |S_2|$

Állapotátmenetek típusai (115)

- Lehetséges, hogy…
  - … teljes gráf → minden átmenet engedélyezett
    - Pl. $S_{\text{kismacska}}=\{alszik, játszik, iszik\}$
  - … nem minden állapotból érhető el az összes többi
    - Pl. $S_{\text{pohár}}=\{üres, teli, törött\}$ → nincs *törött* ⇝ *üres* út
  - … bizonyos állapotoknak több rákövetkezője is van



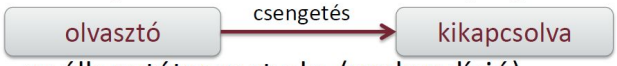
Eseménytér: állapotátmeneti reláció $R \subseteq S \times S$

Állapotgépek, kiterjesztéseik (117)

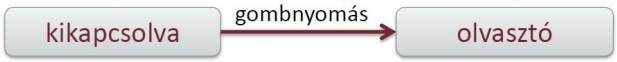
Állapotátmenet címkézése eseménnyel

- Átmeneti szabály címkéje

  - Pillanatszerű esemény
  - Az átmenet csak az eseménnyel együtt következhet be
- Különféle értelmezés: lehet az esemény…
  - … az állapotátmenet következménye (posztkondíció)

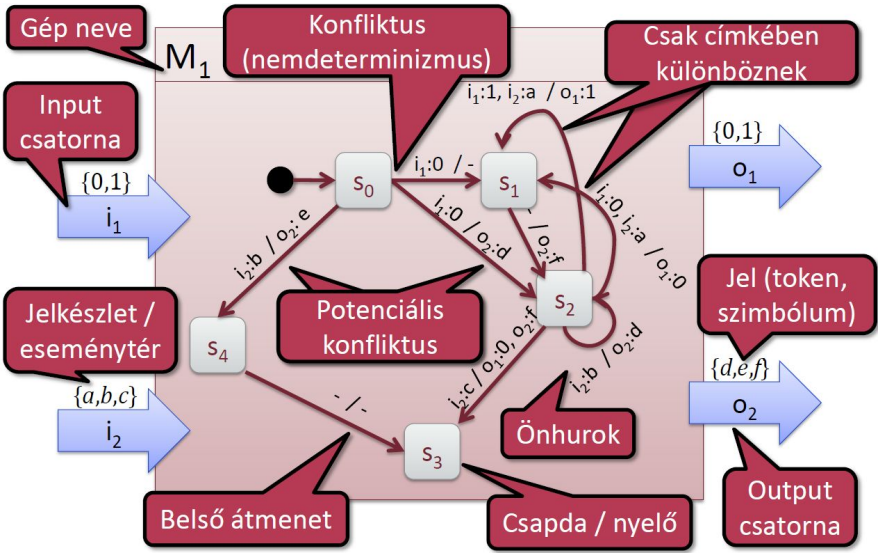


  - … az állapotátmenet oka (prekondíció)



- Egy szabály címkézhető több eseménnyel is
  - input beolvasás / output kiírás

Mealy kiterjesztett állapotgépe (122)

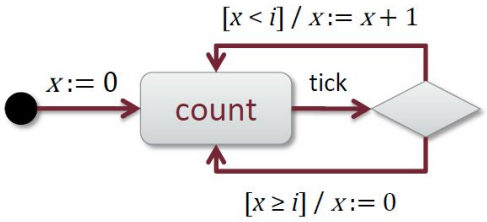


Specifikálatlan input

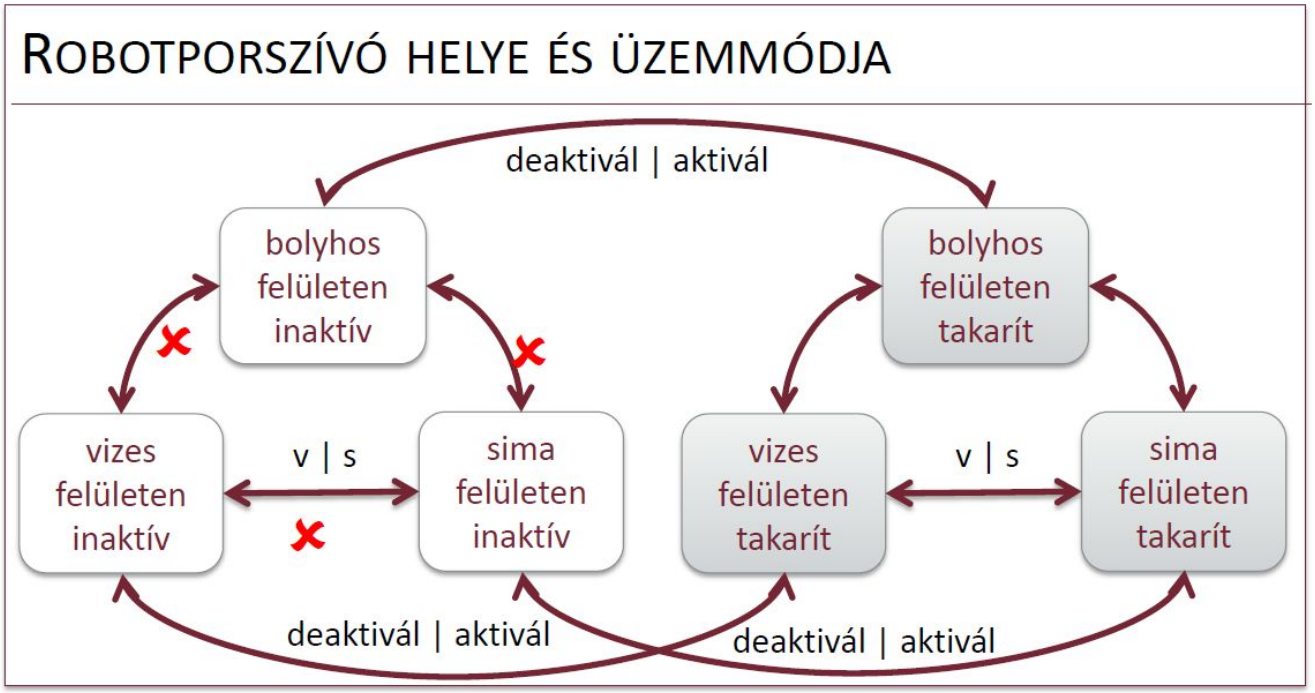
- Minden állapotban minden inputra kell szabály?
  - Ha nincs szabály → esemény nem történhet meg
    - Matematikailag így van, de **input**okra nem reális feltételezés
    - Mi van, ha a gyakorlatban mégis megtörténik?
  - Ha nincs szabály → láthatatlan hurokél
    - „Minden egyéb” input beolvasva, figyelmen kívül hagyva
  - Ha nincs szabály → érvénytelen a modell
    - Pl. kritikus beágyazott rendszereknél
    - Az is érvénytelen, ha több szabály van (determinizmus kell)

Pszeudállapot: (145)

- Szemantikailag nem állapot:
  - Nincs olyan időpillanat, amikor a rendszert jellemezné
- Szintaktikailag állapot:
  - Lehet tranzíció kezdő- vagy célállapota

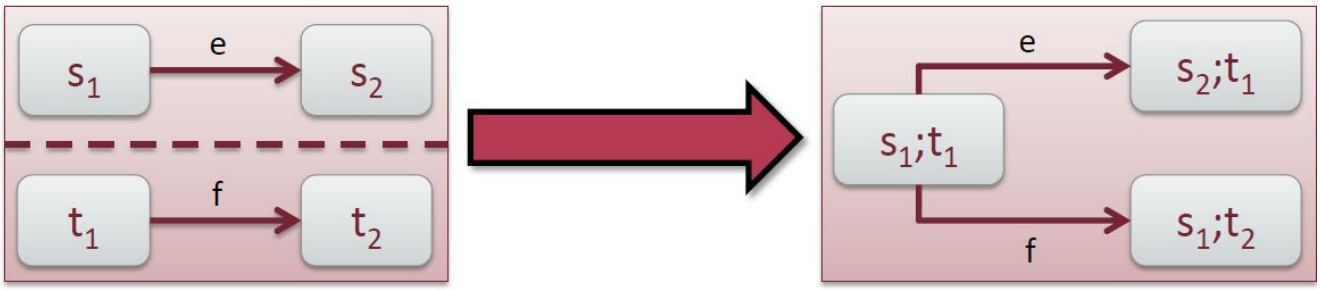


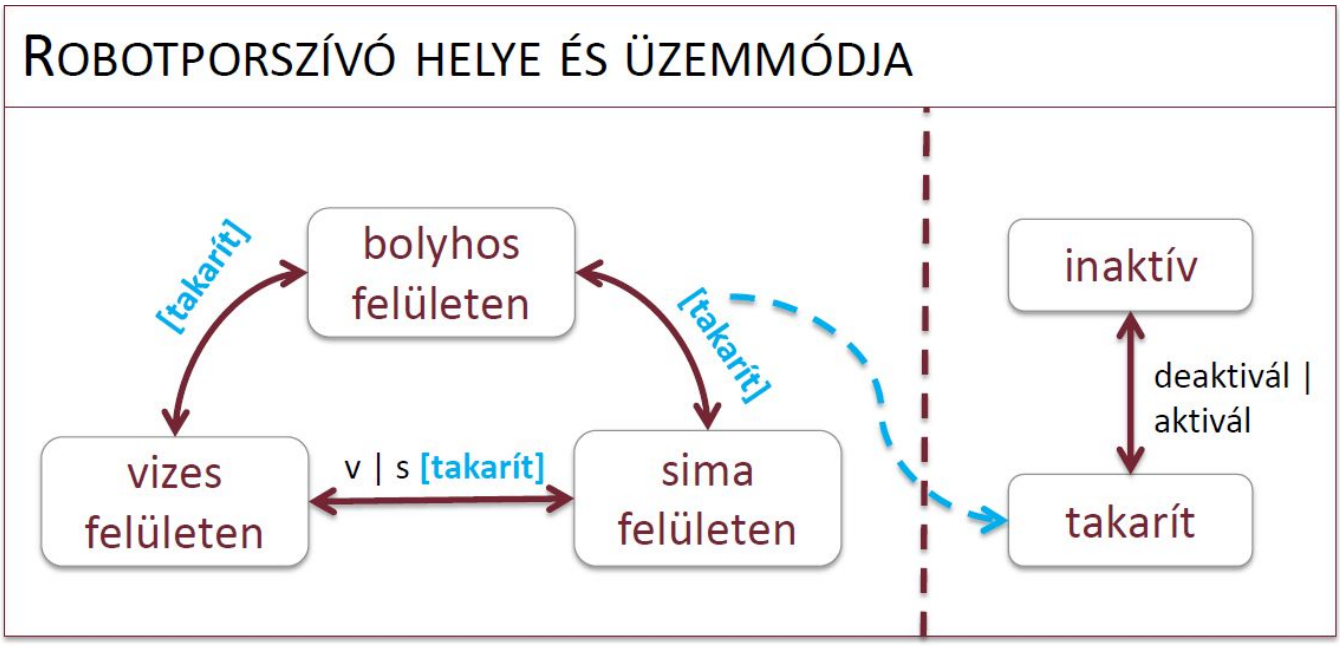
Kooperáló állapotgépek
Aszinkron szorzat (147)



- További finomítást igényel: átmenetek kieshetnek
  → Állapotok így elérhetetlenné válhatnának

- Átmenetek
  - **Aszinkron szorzat**: egyszerre egy állapotrégió lép
    - Az átmenetek a szorzatba „másolódnak”

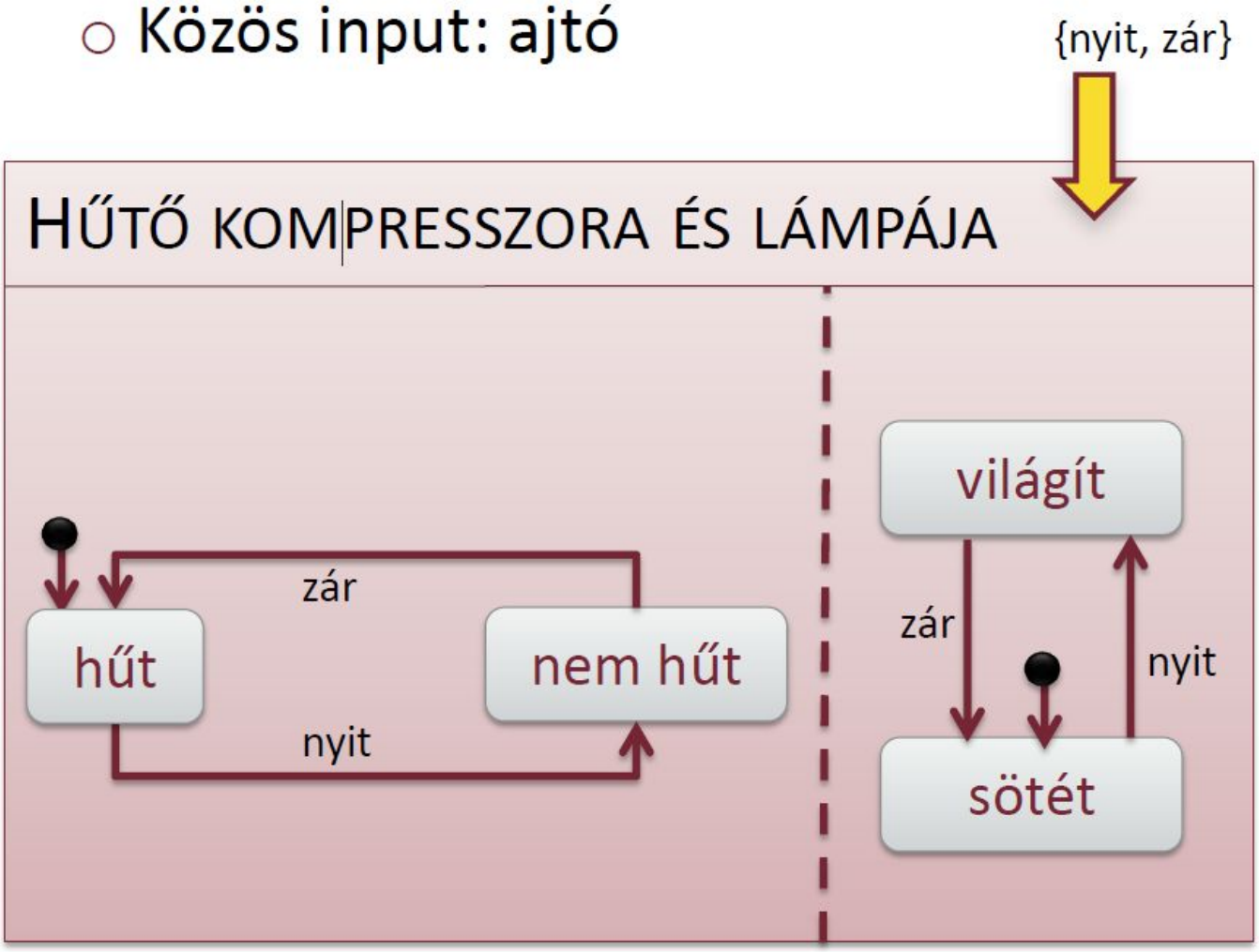




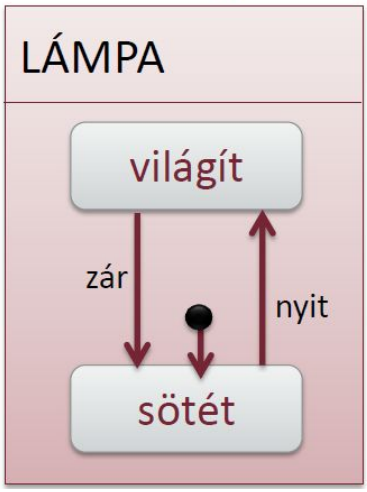
- Kooperáció **őrfeltétel**lel
  - Egyik régióban átmenet feltétele másik régió állapota

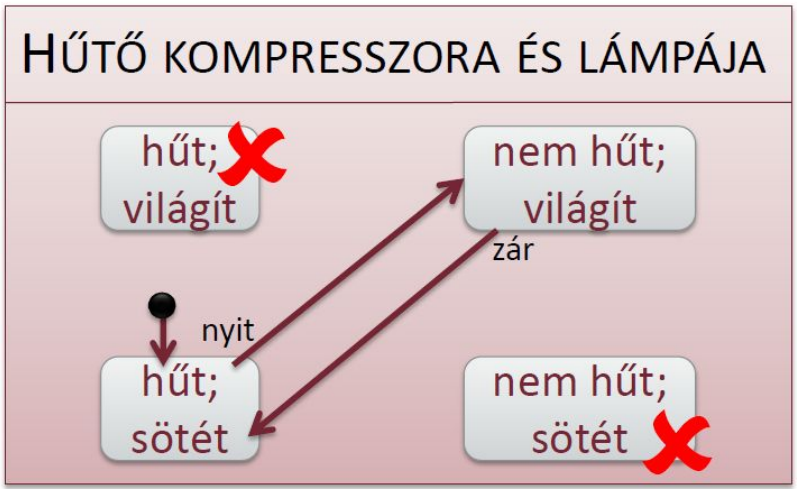
Szinkron szorzat példa

- Hűtőszekrény kompresszor vs. lámpa
  - Közös input: ajtó

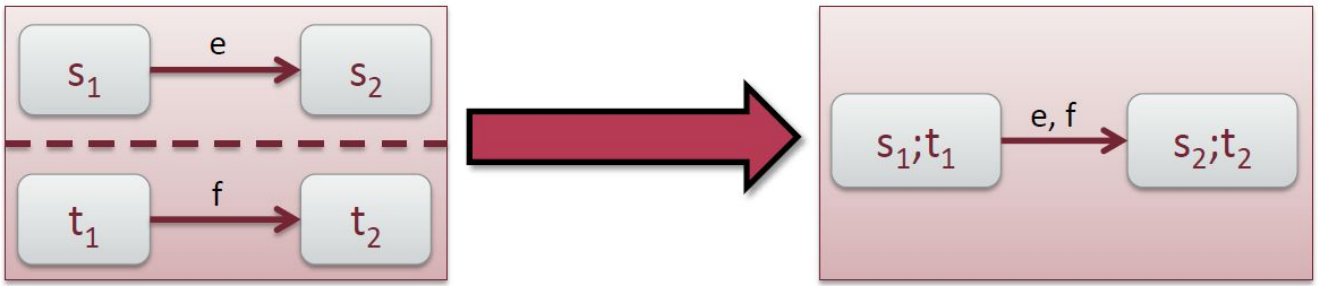


- Közös inputolvasás → szinkron lépés
  - (A kezdőállapotból elérhetetlenné válhatnak állapotok)





- Átmenetek
  - **Szinkron szorzat**: mindkét állapotváltozó egyszerre lép
    - Egy-egy átmenet „összeragasztása”, címkék uniója



vegyes szorzat

- **Vegyes szorzat**: helyenként szinkronizáció történik
  - Alapvetően aszinkron kompozíció…
  - …de bizonyos esetekben együtt lépnek a régiók

Szinkronizáció egyszerű esete: van közös input (is)

Fejlett kooperáció: randevú (belső szinkronizáló esemény)

Szorzatok áttekintése

- Állapotterek szorzata
  - **Direkt szorzat**: $S_1 \times S_2 \times \ldots \times S_n$
    - Összetett állapot: komponensek állapotaiból képzett n-es
- Állapotgépek szorzata
  - Az állapottér mindig a direkt szorzat (finomítása)
  - **Szinkron szorzat**
    - Mindig egyszerre lépnek a komponensek / régiók
  - **Aszinkron szorzat**
    - Mindig csak egy komponens / régió lép
  - **Vegyes szorzat**
    - Helyenként egy, helyenként több komponens / régió lép

Kooperáció módjai

Őrfeltétel

Randevú

----------176.diáig kigyűjtve, ha vmi fontosat gondolsz még, add hozzá----------

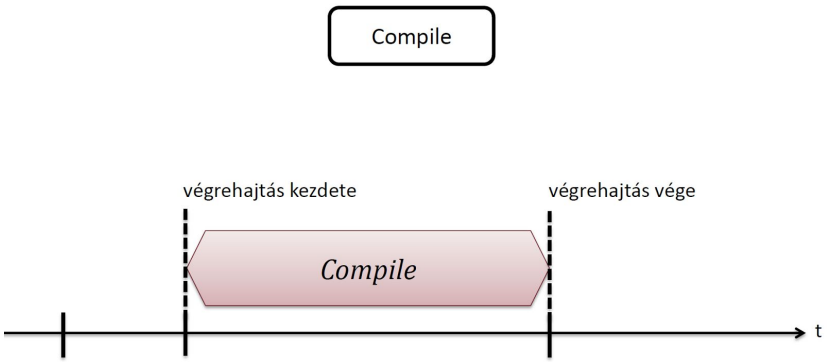
folyamat alapú modell

Folyamat - lépések sorozata, melyek sorrendben történő végrehajtása valamilyen célra vezet.

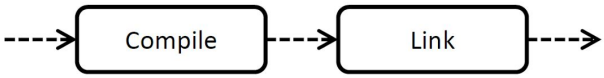
Folyamatmodellezés célja (184)

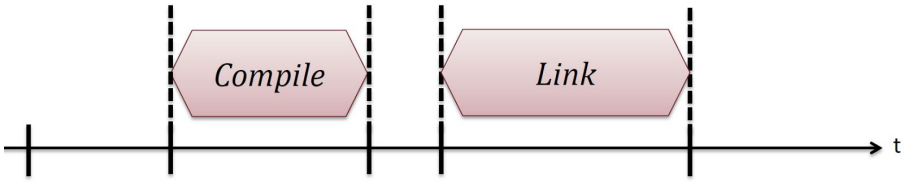
- Dokumentáció
- Implementáció
  - Végrehajtható modellek
  - Kódgenerálás
- Modell szintű ellenőrzés (Verifikáció)
  - Szimuláció
  - Monitorozás
  - Automatizált modellellenőrzés

Elemi tevékenység

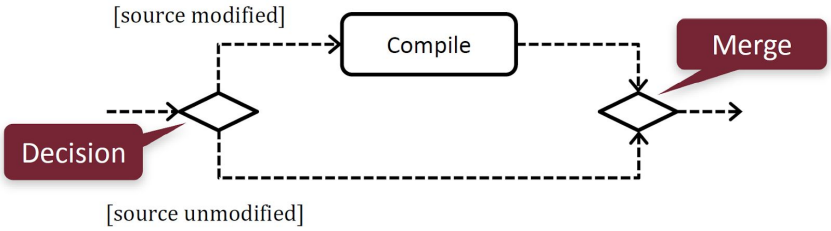


Szekvencia, vezérlésfolyam

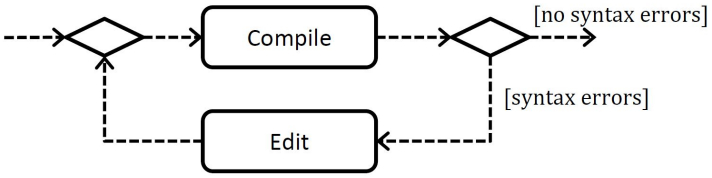




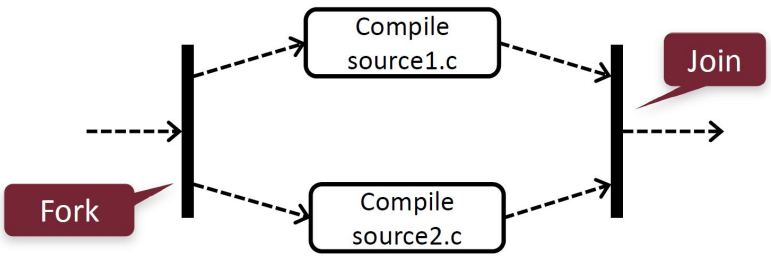
Őrfeltételek, elágazás



Ciklus

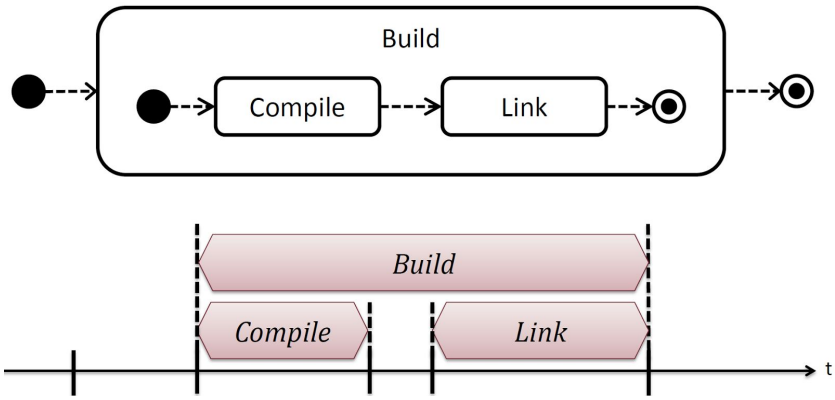


Fork/Join

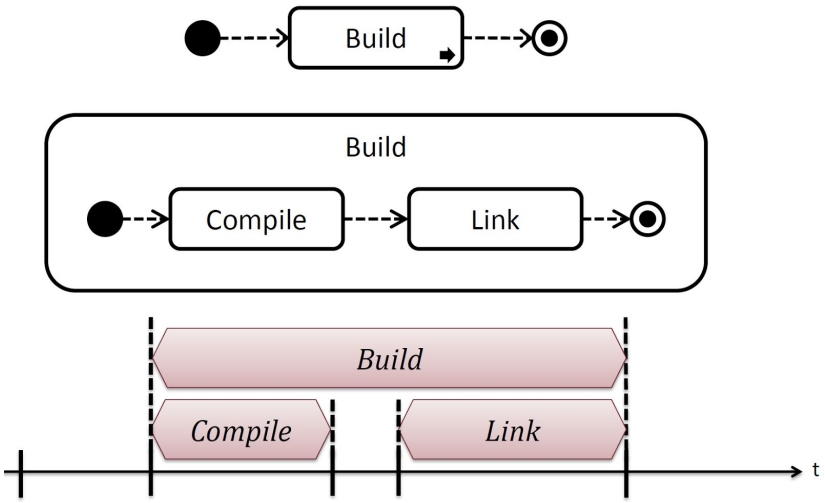


Flow begin / flow end

Hierarchia



Hivatkozás / hívás

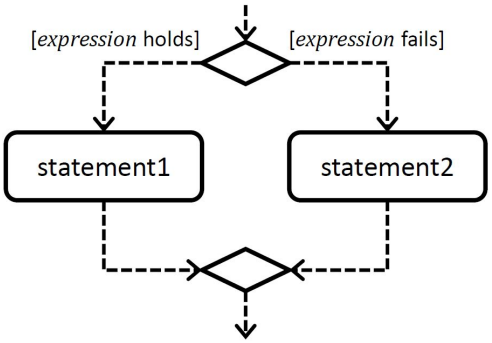


Flowchart / döntési diagram

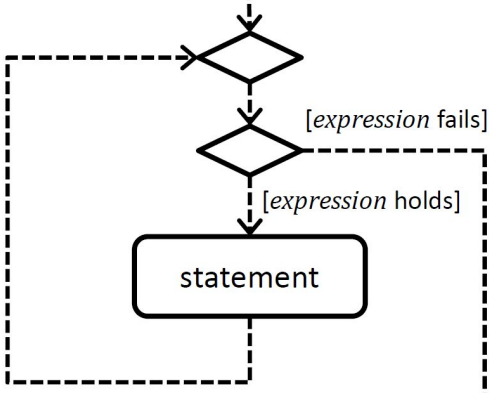
o Döntéshozatali gondolatmenetet ír le
o Konklúzióhoz vezet
o Nem fejez ki időbeli szekvenciát

Vezérlési folyam

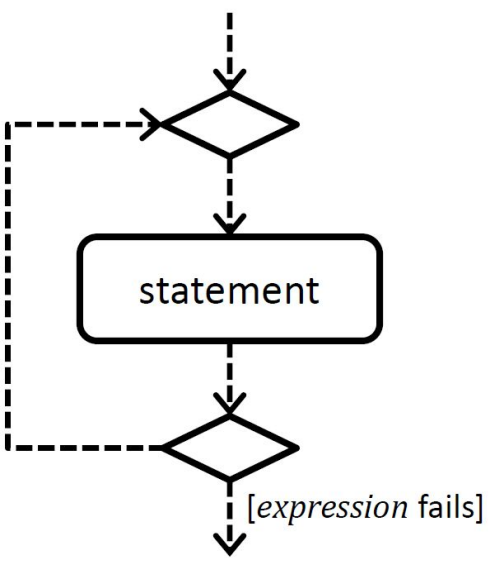
if (<expression>)
  <statement1>
else
  <statement2>



while (<expression>)
  <statement>



do
  <statement>
while (<expression>)

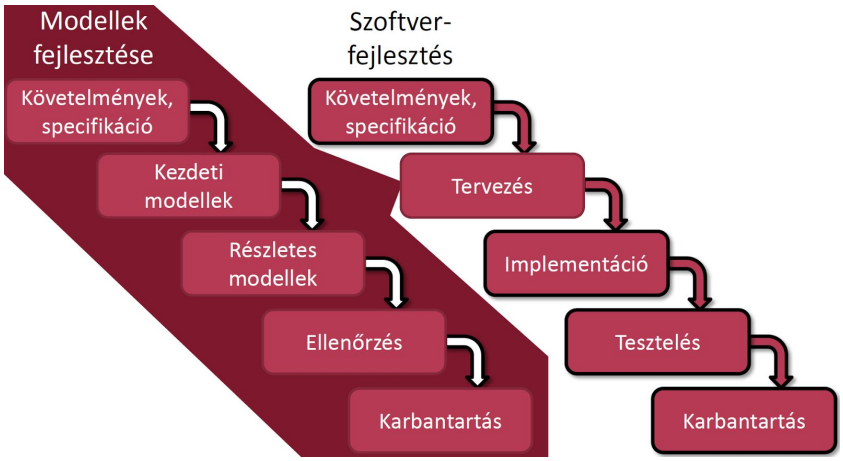


Ciklomatikus komplexitás
M = E – N + 2

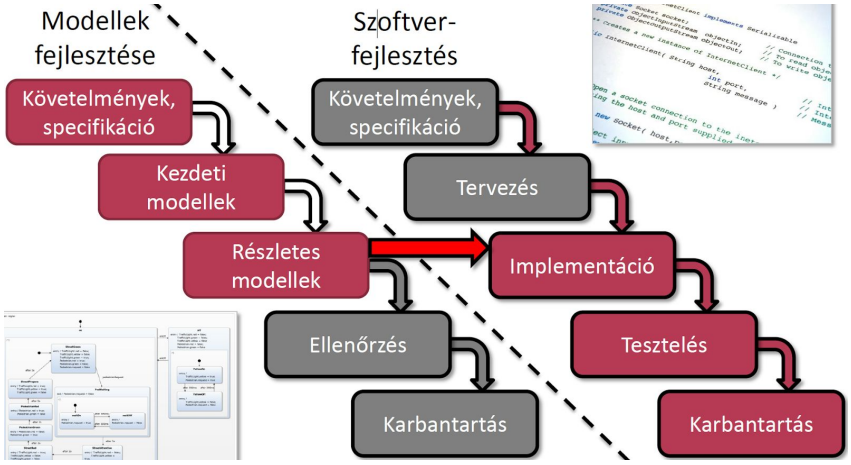
Jólstrukturált folyamatok

Vezérlési blokkokból építkezünk
o Egy bemenet, egy kimenet, közte jól strukturált blokk
o Szekvencia, decision-merge és fork-join blokk, ciklus
o (üres vezérlési szakasz)

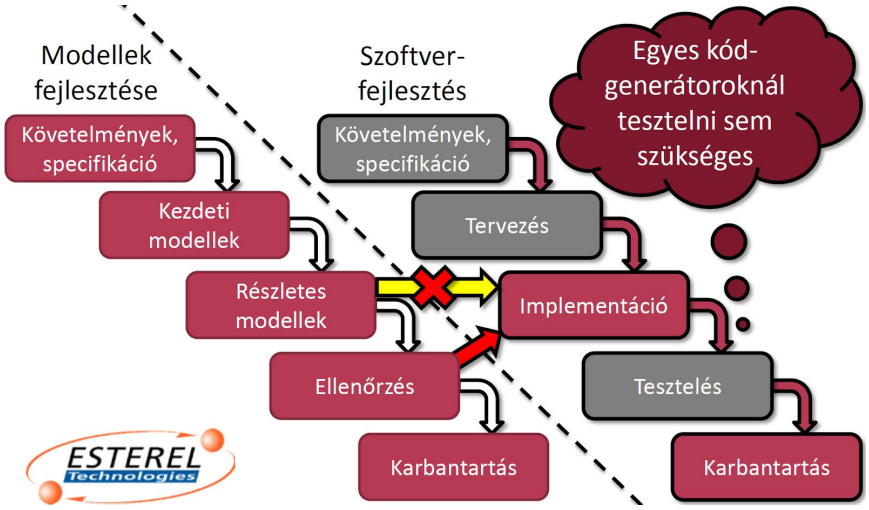
Vízesés modell



modellalapú szoftverfejlesztés



Automatikus kódgenerálás - hitelesített kódgenerátor

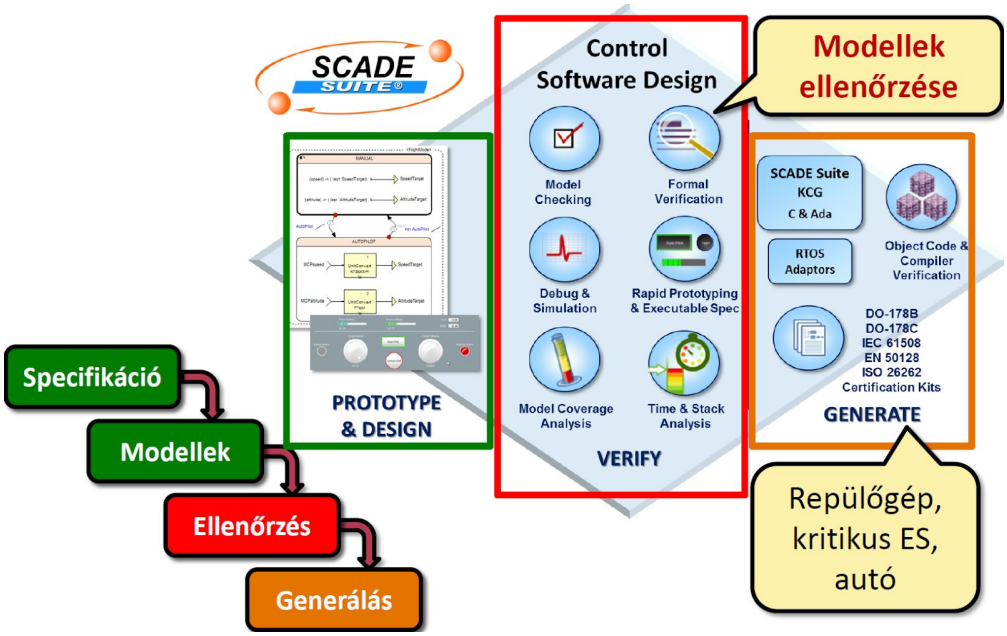


- Szintézis:
  Specifikációnak megfelelő modell?

- Analízis:
  Modell viselkedése?

- Vezérlés:
  Kívánt állapot hogy érhető el?

(Kritikus Rendszerek) Modellvezérelt tervezése



Példa: Esterel SCADE

Előnyök:

- átlagosan 10 ellenőrzött kódsor/nap(kézi kódolásnál 5)
- kódolás, ellenőrzés és tesztelés költsége 70-90% csökkenés
- módosításhoz szükséges idő: 65-75% csökkenés
- automatikus modellellenőrzés: nincsenek kódolási hibák vagy alacsony szintű tesztek

Specifikáció

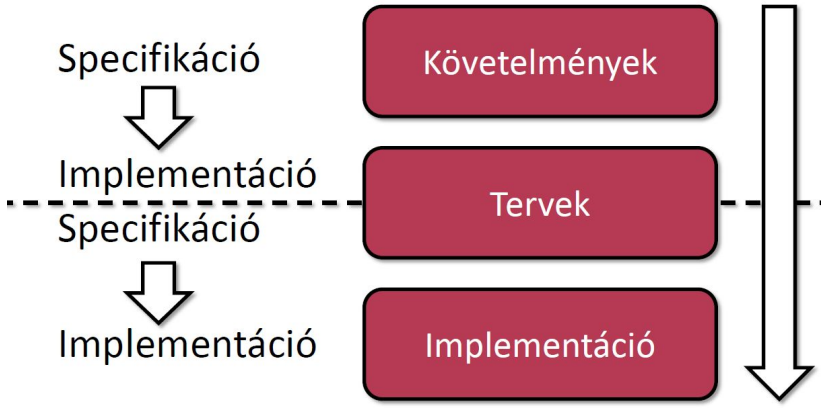
Dokumentált követelmények halmaza, amelynek egy adott terméknek, dizájnnak vagy szolgáltatásnak meg kell felelnie.

Fajtái:

- informális vs. formális
- deklaratív vs. végrehajtható
- funkcionális vs. nemfunkcionális

Specifikéció>reguláris kifejezés>állapotgép

Implementáció>Állapotgép>C++ kód



Informális specifikáció
o Leginkább kommunikációra használt
o Jellemzően a fejlesztési folyamat elején
o Többféle értelmezés, pontosítás szükséges
o Pl. szöveges leírás, folyamatábra, pszeudokód, stb

Formális specifikáció
o Pontos matematikai szemantika
o Egyetlen lehetséges értelmezés
o Ellenőrzés és kódgenerálás alapja lehet
o Pl. Mealy automata, BPMN, logika, stb

Deklaratív specifikáció
o Az elvárt végeredményt írja le, de nem mond semmit az előállításáról
o Pl. logika, reguláris kifejezés, stb.

Végrehajtható specifikáció
o Legalább „ennyit kell tudnia” az implementációnak
o Pl. állapotgép, folyamatmodell, adatfolyamháló, stb.

Funkcionális követelmény
o Mit kell csinálnia a rendszernek?
Többnyire a rendszer egy részére vonatkozik
o Pl. Autót lehessen vezetni

Nemfunkcionális követelmény
o Hogyan csinálja a rendszer?
o Pl. Autó legyen gyors
Többnyire a teljes rendszerre vonatkozik

pl.:

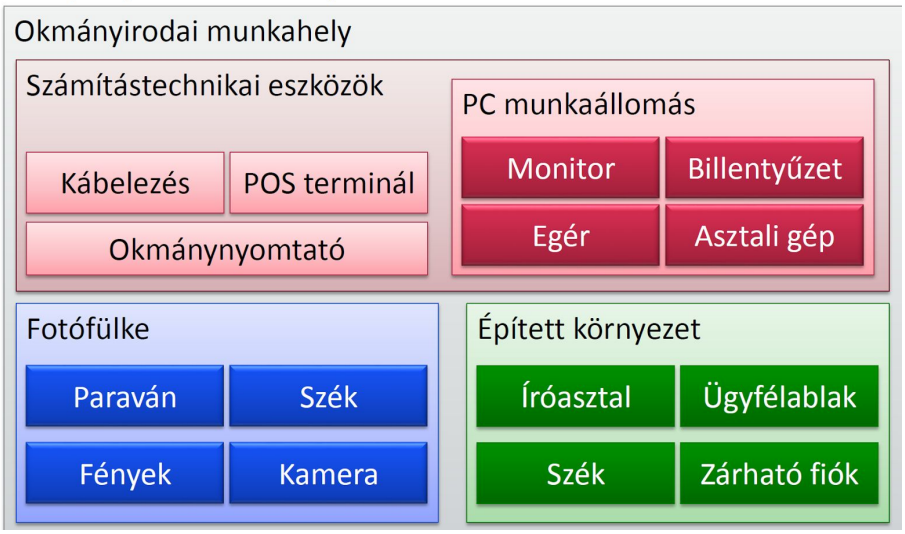
- időzítés
- átbocsátóképesség
- teljesítmény
- megbízhatóság
- rendelkezésre állás
- védelem ártó szándék ellen
- biztosnságosság
- karbantarthatóság
- használhatóság

Szintézis/finomítás:
A specifikációt teljesítő implementációk felderítése
Választás a lehetséges megoldások közül, valamilyen szempont szerint: tervezői döntés > felelősség
Új információ kerül a tervekbe

Fordítás/generálás:
Jellemzően különböző formalizmusok között
Nincs új információ, automatizálható

Top-down tervezés:

- Alaplépés: dekompozíció



Előny:

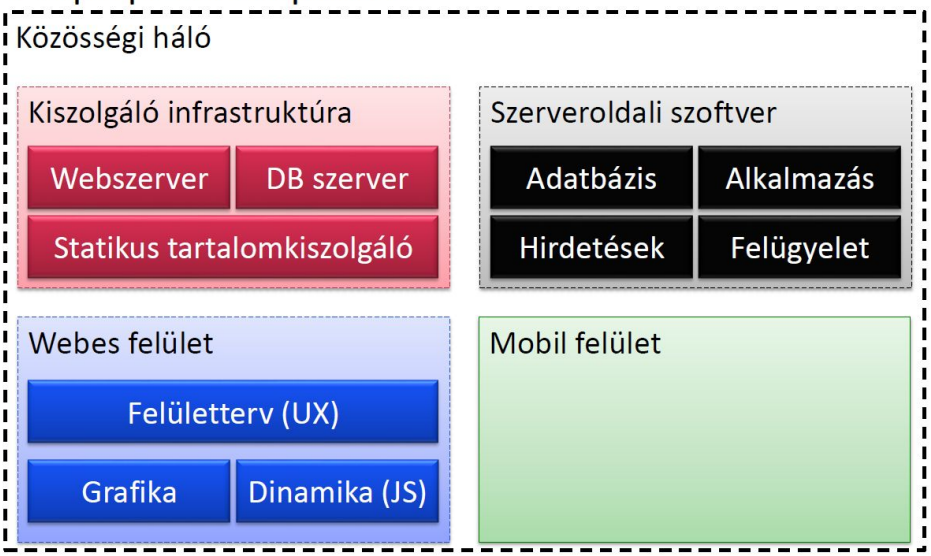
- részrendszer tervezésekor a szerepe már ismert

Hátrány:

- félidőben még nincsenek működő részek
- részek problémai, igényei későn derülnek ki

Bottom-up tervezés:

- Alaplépés: kompozíció



Előny:

- alrendszer önmagában kipróbálható, tesztelhető
- részleges készültségnél is összeépíthető valami

Hátrány:

- nem látszik előre a rész szerepe az egészben

Dekompozíció(„faktoring”)**:**
Egy összetett probléma vagy rendszer kisebb részekre bontása, amelyek könnyebben érthetők, fejleszthetők és karbantarthatók.

- Funkcionális dekompozíció: ~blokkséma
- Hierarchikus dekompozíció: rész-egész viszony

Követelménymodellezés
Cél: egyértelmű, kezelhető specifikáció
Terminológiai egységesítés
Nyomonkövethetőség: A követelménytől a kódig, legyen követhető melyik követelményből származik.
Szabvány: OMG ReqIF (Requirements Interchange Format)
Eszközök:

- Excel
- IBM DOORS
- ECLIPSE RMF
- FormalMind

Strukturális finomítás: A rendszer egyes tevékenységei részletezhetők
Tokenfinomítás - adattcsere (adattípusok) részletesebb modellezése (tehát többféle lesz)
Állapotfinomítás - belső működést részletezhetjük

Tervezéskor, implementáláskor finomabb modell (Finomítás)
o Új információ beépítése
o A lehetséges megvalósítások száma csökken
o Tervezéskor, specializációkor, együttes vizsgálatkor

Teszteléskor, ellenőrzéskor absztraktabb modell (Absztrakció)
o Információ elvesztésével jár
o A lehetséges megvalósítások száma nő
o Elemzéskor, teszteléskor, dekomponáláskor

----------313.dia----------

Modellek ellenőrzése:

Helyesség:
modell vagy kód megfelelése a követelményeknek.
o Funkcionális helyesség:

megfelelés a funkcionális követelményeknek
o Nemfunkcionális követelmények ellenőrzése:
lásd teljesítménymodellezés előadás <link?>

Szempontok:
o Mindig képes teljesíteni a feladatot
o Hibamentes
o Nincs tiltott viselkedés

Funkcionális követelmények csoportosítása:

Megengedett viselkedés:
o Milyen állapotokban lehet/nem lehet a rendszer
o Milyen viselkedés tilos
o Univerzális követelmények
• Mindig igaznak kell lenniük

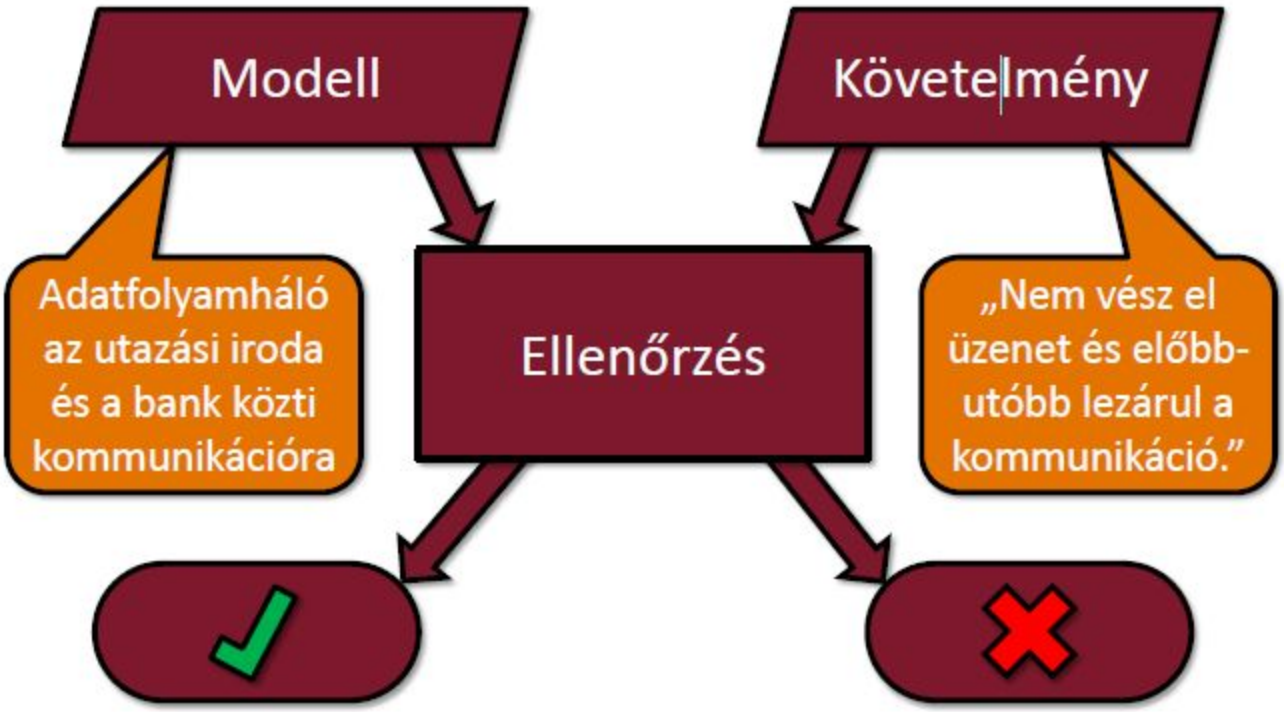
Elvárt viselkedés:
o Milyen állapotokba kell tudni eljutni
o Milyen funkciókat kell tudnia a rendszernek
o Egzisztenciális követelmények
• Lehessen lehetőség a teljesülésükre

Holtpont
Beragadt állapot:
a rendszer csak külső segítséggel képes kilépni.
o Pl. egymásra várakozó folyamatok miatt

Gyakori tervezési hiba párhuzamos rendszereknél
o Sokszor nehéz elkerülni, feloldani
• A jónak hitt megoldás is problémát okozhat
o Nehéz tesztelni, látszólag véletlenszerű is lehet
o „Többmagos CPU krízis”

Végtelen ciklus (livelock)
Beragadt állapothalmaz:
a rendszer csak külső segítséggel képes kilépni.
o Pl. éppen a holtpont feloldása miatt

Modellek ellenőrzése



Vizsgálatok fajtái:

Cél szerint:

o Verifikáció: jól csinálom a rendszert?

• Az implementáció megfelel a specifikációnak?

o Validáció: jó rendszert csinálok?

• A rendszer teljesíti a felhasználói követelményeket?

Módszer szerint:

o Statikus ellenőrzés

o Dinamikus ellenőrzés

• Szúrópróbaszerű (tesztelés, szimuláció)

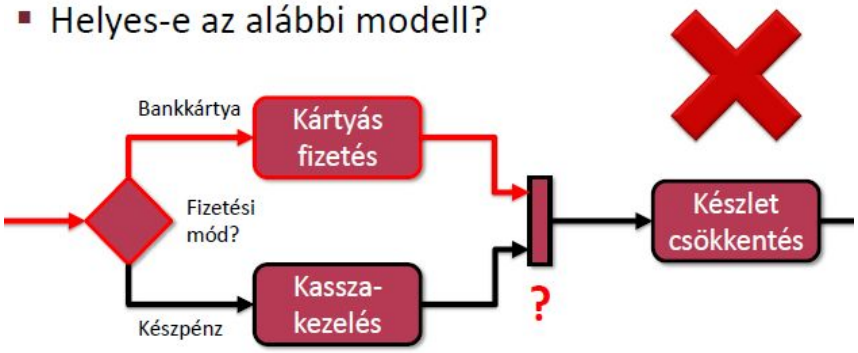
• Kimerítő/teljes (modellellenőrzés)

Statikus ellenőrzés:

pl.: HIBÁS modellekre

Decision és Join!

- Helyes-e az alábbi modell?

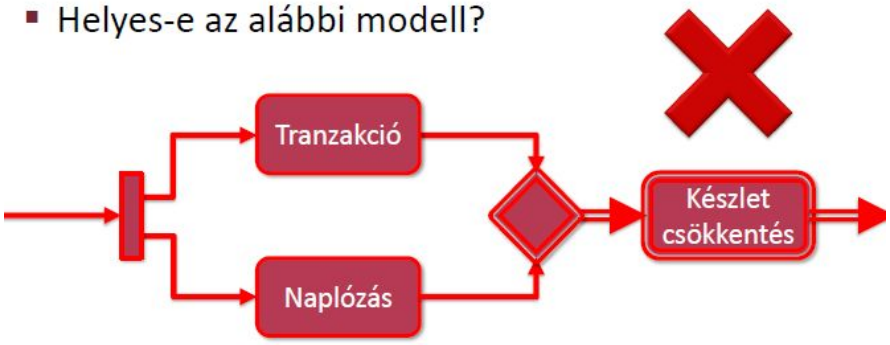


- Join: csak akkor léphet tovább, ha mindegyik bemeneten érkezett token

→ **DEADLOCK**

Fork és Merge

- Helyes-e az alábbi modell?

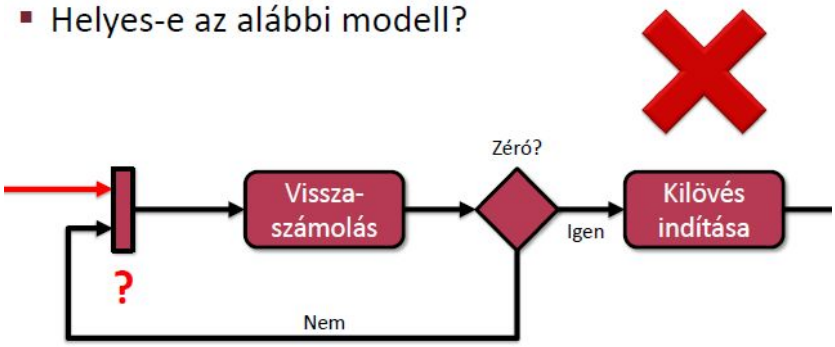


- Merge: bármelyik ágon érkező tokent átengedi
  - Nem szinkronizálja a szálakat

→ **„Készlet csökkentés" kétszer fut le**

Ciklusok

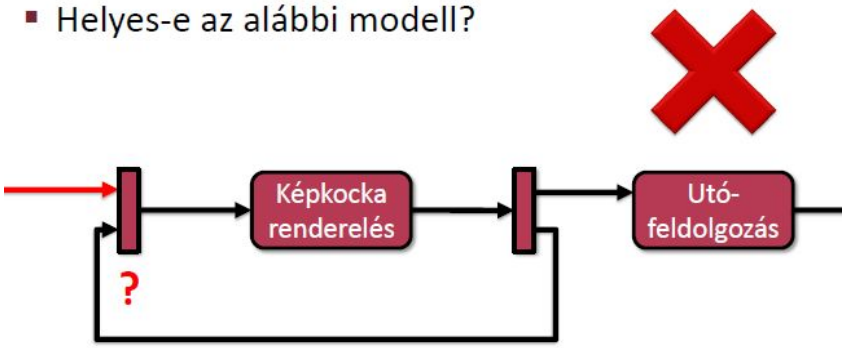
- Helyes-e az alábbi modell?



- Join: csak akkor léphet tovább, ha mindegyik bemeneten érkezett token

→ **DEADLOCK**

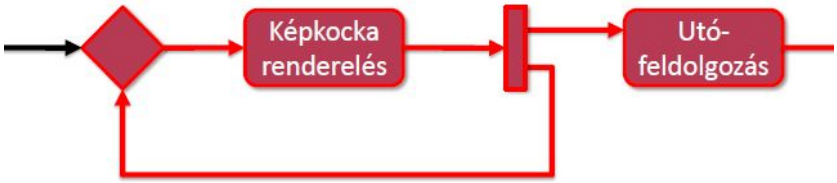
- Helyes-e az alábbi modell?



- Join: csak akkor léphet tovább, ha mindegyik bemeneten érkezett token

→ **DEADLOCK**

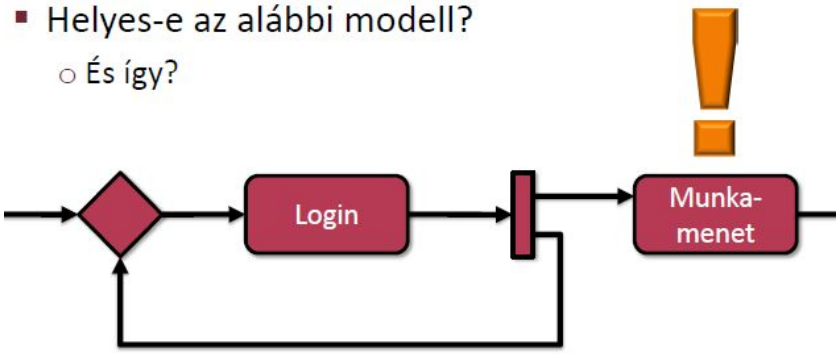
- Helyes-e az alábbi modell?



- Minden iterációban egy új képkocka
  - Mindegyikre utófeldolgozás (sokszor – hányszor?)

Határeset…

- Helyes-e az alábbi modell?
  - És így?

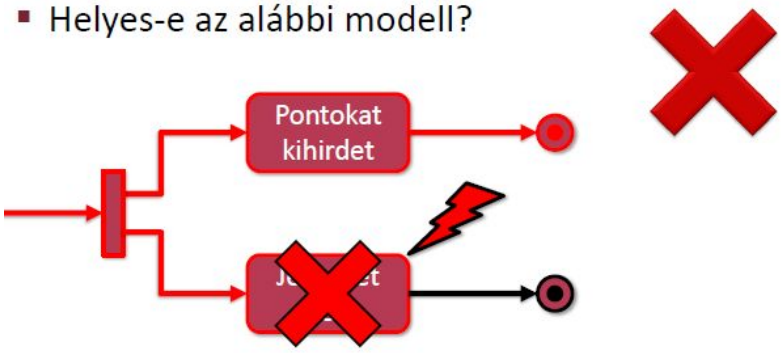


- Minden login után újabb login…
  - …és egy munkamenet…?

→ **Szálakat „termel” a hibás implementáció**

Termináló csomópont:

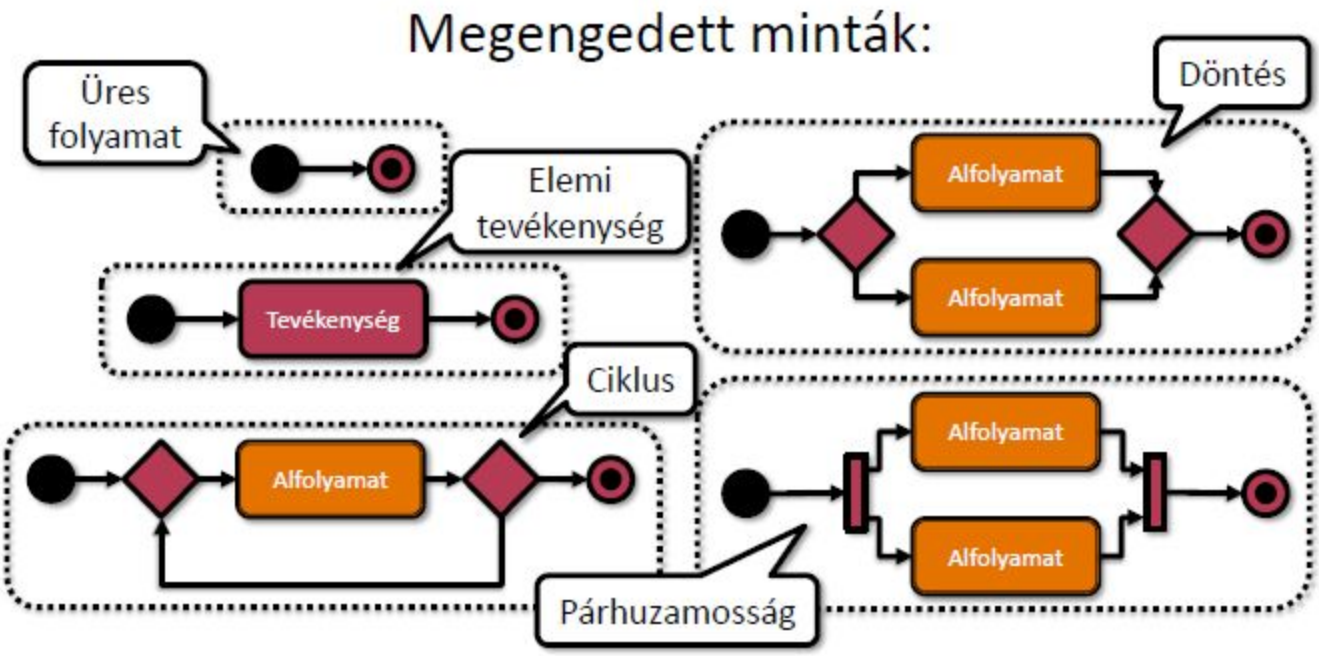
- Helyes-e az alábbi modell?



- Termináló csomópont: azonnal leállítja az **egész** folyamatot

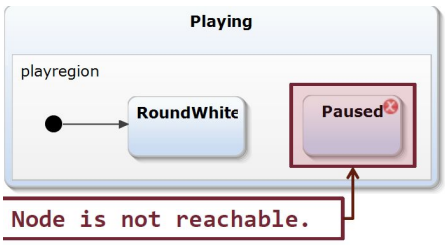
→ **A másik tevékenység nem fog lefutni**

Jól strukturált folyamatmodellekkel ezek a hibák elkerülhetőek.



Adatkezelés statikus ellenőrzése:
Szimbolikus végrehajtás:
Konkrét értékek helyett értékhalmazokkal hajtjuk végre a programot
o Érdekes bemenetek meghatározhatóak
• Pl. ahol belső elágazások vannak
Milyen bemenettel érhetőek el az ágak?

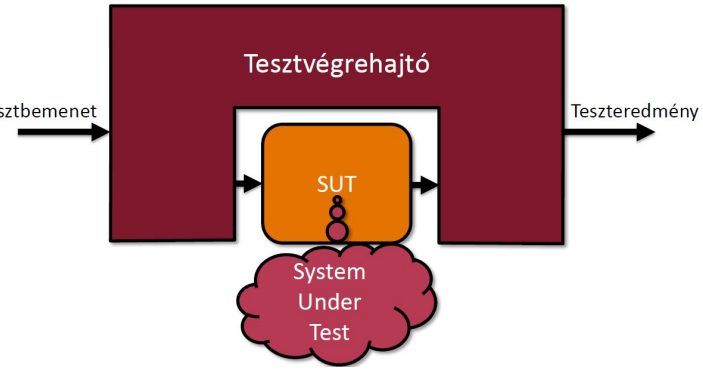
Statikus ellenőrzés:
szintaxisellenőrzés
szintaktikai ell.: modellező eszközök összekötik a logikailag egymásra épülő modellelemeket
szintaxisvezérelt szerkesztő
kód és diagram együtt
programozás : szerkesztés közben **hibás**
modellezés : szerkesztés közben **helyes**

strukturális helyesség
strukturális ell.: modell gráf vizsgálata
hibaminták keresése szekesztés közben
pl.: elérhetetlen  állapot:

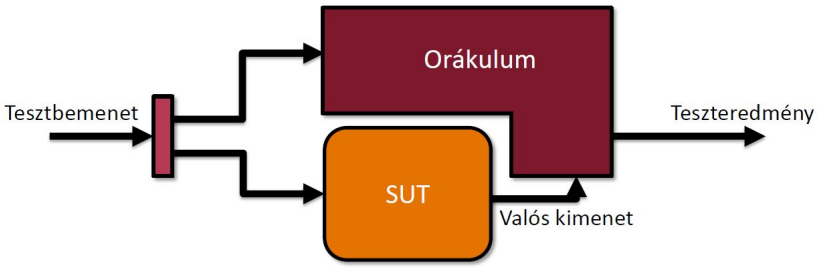
lehet még:
hiányzó kezdőállapot,
holtpont,
változó értékadások...

Tesztelés

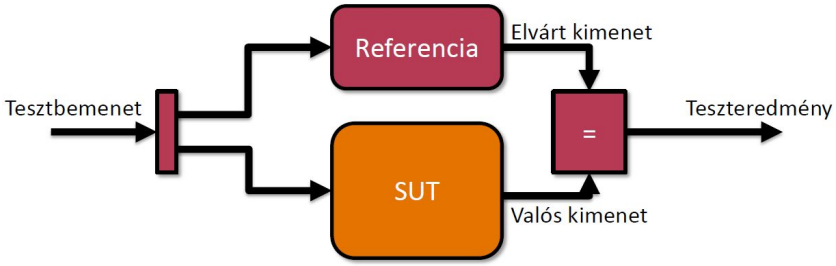
Modellek tesztelése:



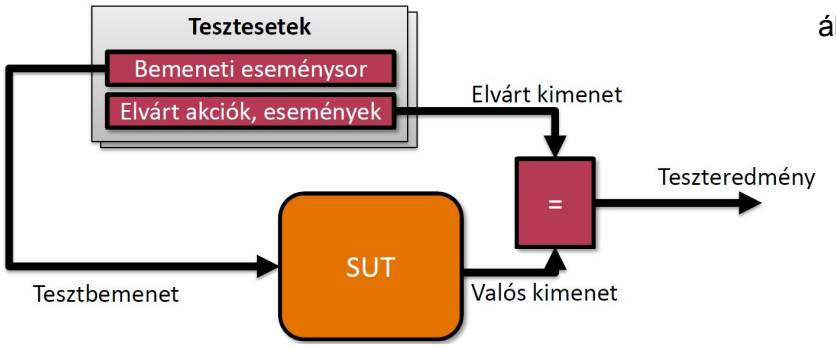
Orákulum:
elvárt eredmények előállítása és összehasonlítása



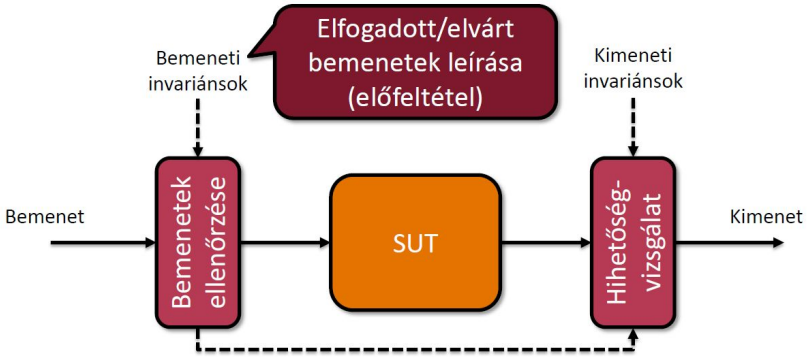
Referencia:
tesztbemenet alapján elvárt kimenet



pl.: Yakindu állapotgép

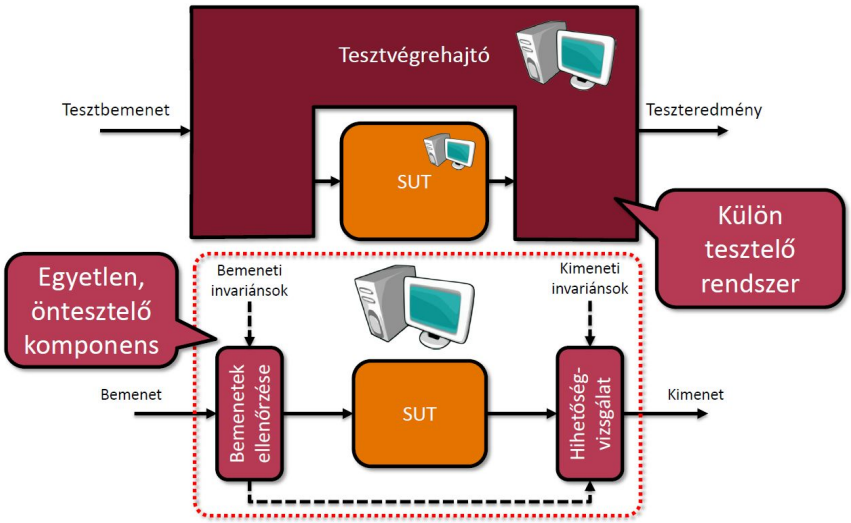


Öntesztelés (monitor):
Invariáns tulajdonság: folyamatosan igaznak kell lennie



Elfogadható/ígért kimenetek leírása (utófeltétel) -> kimeneti invariánsok

Öntesztelés vs. Külső tesztelés



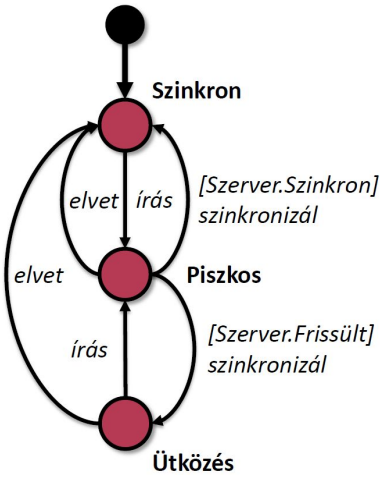
Exception(kivétel): A normálistól eltérő, váratlan helyzet, aminek a *kezelését máshol valósítjuk meg.* (InvalidInput)
Oka: hibás használat
Assert(feltételezés): Hibás állapot, aminek a kezelésére a kód *nincs felkészítve*. (Error)
Oka: Hibás implementáció vagy futásidejű hiba.

Szimuláció: A modell futtatása
Adott bementekre vizsgált viselkedés

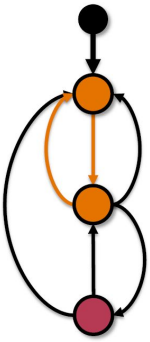
Fedettség:
Adott tesztkészlet esetén a tesztek futtatása alatt érintett részek aránya a modellben.

Állapot lefedettség (állapotgépben): érintett állapotok / összes állapot
Átmenet lefedettség (állapotgépben): érintett átmenetek / összes átmenet

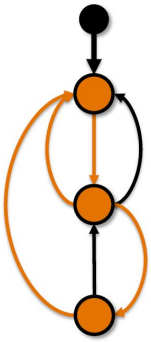
Utasítás lefedettség (vezérlési folyamatban): érintett tevékenységek / összes tevékenység



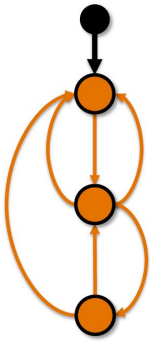
1. Teszteset után:
Állapotfedés: 2/3=66%
Átmenetfedés: 2/6=33%



2. Teszteset után:
Állapotfedés: 3/3=100%
Átmenetfedés: 4/6=66%



3. Teszteset után:
Állapotfedés: 3/3=100%
Átmenetfedés: 6/6=100%



Tesztelt modellek használata:

Szoftvertesztelés: futás előtt
o (100%-osan fedő) tesztkészlet újrafelhasználása
o Fedő tesztbemenetek (bemenet)
o Modell által adott kimenetek (elvárt kimenet)

Monitorozás: futás közben
modell szimulálása a szoftver futtatása mellett
o Azonos bemenetek a modellnek és a programnak
o Kimenetek összevetése -> hibadetektálás

Logelemzés: futtatás után
Monitor futtatása naplózott bemenetek/kimeneten

Tesztdokumentáció: A teszteseteket és teszteredményket is dokumentálni kell!
Nyomonkövethetőség: teszteletlen kódrészletek és ellenőrizetlen követelmények felderítése

Tesztek fajtái, szakaszai:

Modultesztelés: Egy komponens leválasztása és tesztelése.
Integrációs tesztelés: Több komponens együttes tesztelése.
Rendszertesztelés: A teljes rendszer együttes tesztelése.
Regressziós tesztelés: Változtatások utáni (szelektív) újratesztelés

Formális verifikáció: modellek/programok helyességének bizonyítása matematikai eszközök segítségével
Eszközök:

- *Modellellenőrzés*: lehetséges viselkedések kimerítő vizsgálata
- Automatikus helyességbizonyítás: axiómarendszerek alapján automatikus tételbizonyítás
- Konformancia vizsgálatok: megfelelőség ellenőrzése modellek között

Modellellenőrzés:

A modell lehetséges viselkedéseinek kimerítő (teljes) vizsgálata adott követelmények alapján
Tesztelés vs Modellellenőrzés:

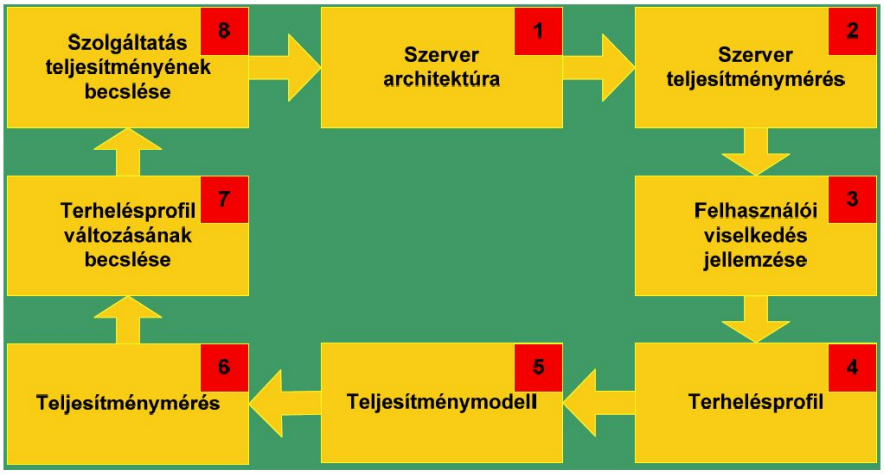
Tesztelés	**Modellellenőrzés**
szúrópróbaszerű	kimerítő/teljes
elvárt kimeneteket ellenőriz	állapotok sorozatát ellenőrzi
kisebb számítási igényű	nagy számítási igényű
nem bizonyítóerejű	formálisan bizonyít

Kapacitástervezési metodika

Kapacitéstervezés:

- Annak *becslése*, hogy a rendszer mikor *telítődik* a terhelés hatására.
- A leginkább *költséghatékony* módszer megtalálása, mellyel a rendszer *túlterhelése* a lehető legjobban *késleltethető*.
- figyelembe veszi a nyújtani kívánt szolgáltatás szintjét

Mennyiségi analízis lépései:



Elvárt kapacitás függ:

- **Service Level Agreement (SLA):** A vezetés által meghatározott mérőszámok

pl.: válaszidő, elérhetőség, átbocsátóképesség

- **Használt technológiák, szabványok**
- **Pénzügyi lehetőségek**
- **Kapacitéstervezés:** kvantitatív szemlélet

cél: ne kelljen sűrűn változtatni a rendszeren

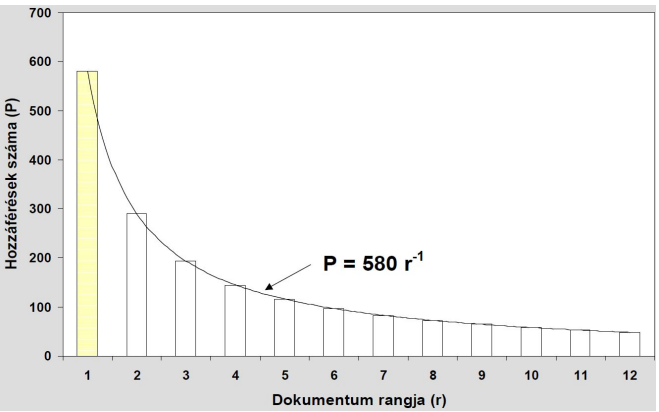
A terhelés nőhet:

- nő az átlagos terhelés (rendszer változatlan)
- új alkalmazások/szolgáltatások
- változik a felhasználók viselkedése

<u>Terhelés modellek:</u>

A kérések tartalma: Pareto-elv: a kérések többsége az adatok kis részére irányul
műszaki hatása van, a gyakori kéréseket másképp kezeljük

Zipf törvénye:



Eredetileg: korpuszokban előforduló szavak száma és gyakorisága jellegzetes eloszlást mutat
(több tudományban is felhasználható)

Példák:

- slágerlisták

- városok populációja rangsoruk szerint
- internetes forgalom karakterisztikája
- weboldalak aloldalainak népszerűsége
- nyílt forrású rendszerek evolúciója

Képlet:

$$R_i \sim \frac{1}{i^{\alpha}} \qquad f \sim \frac{1}{p}$$

Ri – az i. szó előfordulási gyakorisága
α – a korpuszra jellemző 1 közeli érték
Egyszerűsítve:
o f (frequency) gyakoriság
o p (popularity): a szöveg ,,rangja'' (csökkenő sorrendben)
Zipf törvénye: web dokumentumokra:

$$P = \frac{k}{r}$$

P – hivatkozások (elérések)
r – rang (1 = leggyakoribb)
k – pozitív konstans

Felhasználói viselkedés:

- statikus: oldalak szerkezet alapján
- HTTP logok alapján
- hasonló viselkedésű felhasználók csoportosítása
- hogyan változik a modell az új funkciók bevezetése után?

Felhasználói modell - CBMG (customer behavior model graph):
- navigációs minták: felhasználhatóak a későbbi terhelés előrejelzéséhez
- eszköz:
  - **Customer Behavior Model Graph (CBMG),**
  - **Customer Behavior Model Statechart (CBMS),**
  - **Customer Visit Model (CVM):** kevésbé részletes, nem használható előrejelzésre

CBMG:
irányított gráf az oldalak közti lehetséges átmenetek és valószínűségeik ábrázolására
N csúcs, ahol:
- 1. csúcs: Entry állapot: absztrakt belépési pont, minden felhasználó innen indul
- n. csúcs: Exit állapot, absztrakt kilépési pont, a folyamat vége
- A többi csúcs megfelel a felhasználó által elérhető szolgáltatásoknak

élek: az oldal szerkezete határozza meg a lehetséges átmeneteket

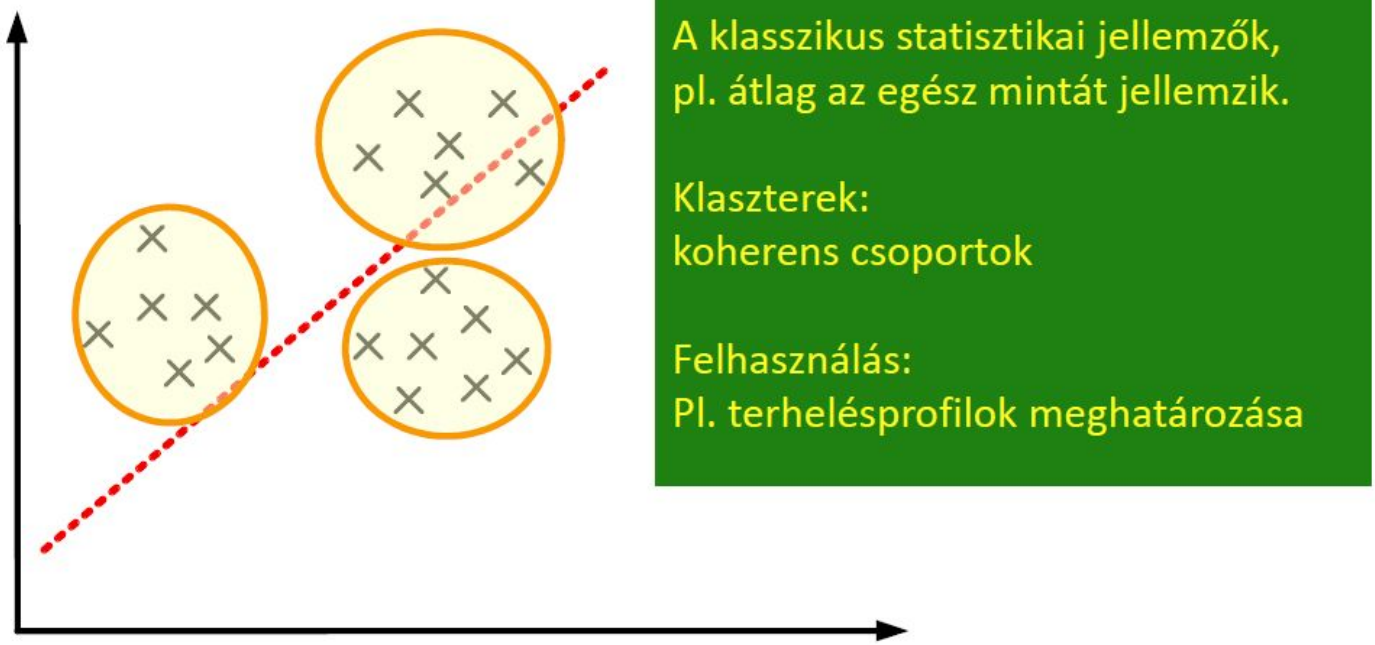
Statikus CBMG: oldalak és köztük lévő lehetséges átmenetek

CBMS: UML állapotdiagram, hasonló információt hordoz, mint CBMG, de mindezt szabványos módon jeleníti meg
Állapotok: az oldal funkciói (állapotai)
Lehetséges átmenetek a rendszer modellje alapján, az átmenetekhez valószínűség tartozik

Felhasználói viselkedés vizsgálata, mérőszámok:
Revenue throughput: az oldal által elért átlagos bevétel
Potential Loss Throughput: mennyibe kerül a szolgáltatás kiesése egy adott időszakban
Visit Ratio: átlagosan hányszor vesznek igénybe egy adott szolgáltatást
Buy to Visit Ratio: átlagosan hányszor vásárolnak egy session alatt (tényleges eladási tranzakció)
Average Session Length: egy session átlagosan hány szolgáltatást vesz igénybe (nem időt mér)

CVM: különböző típusú felhasználók viselkedését jellemző vektorok (melyik állapotban hányszor járt)
A felhasználók csoportosítása klaszterezési technikákkal történhet
nem jelzi az egyes állapotok közti átmenetek gyakoriságát, kevésbé alkalmazható előrejelzésre

Klaszter technikák:



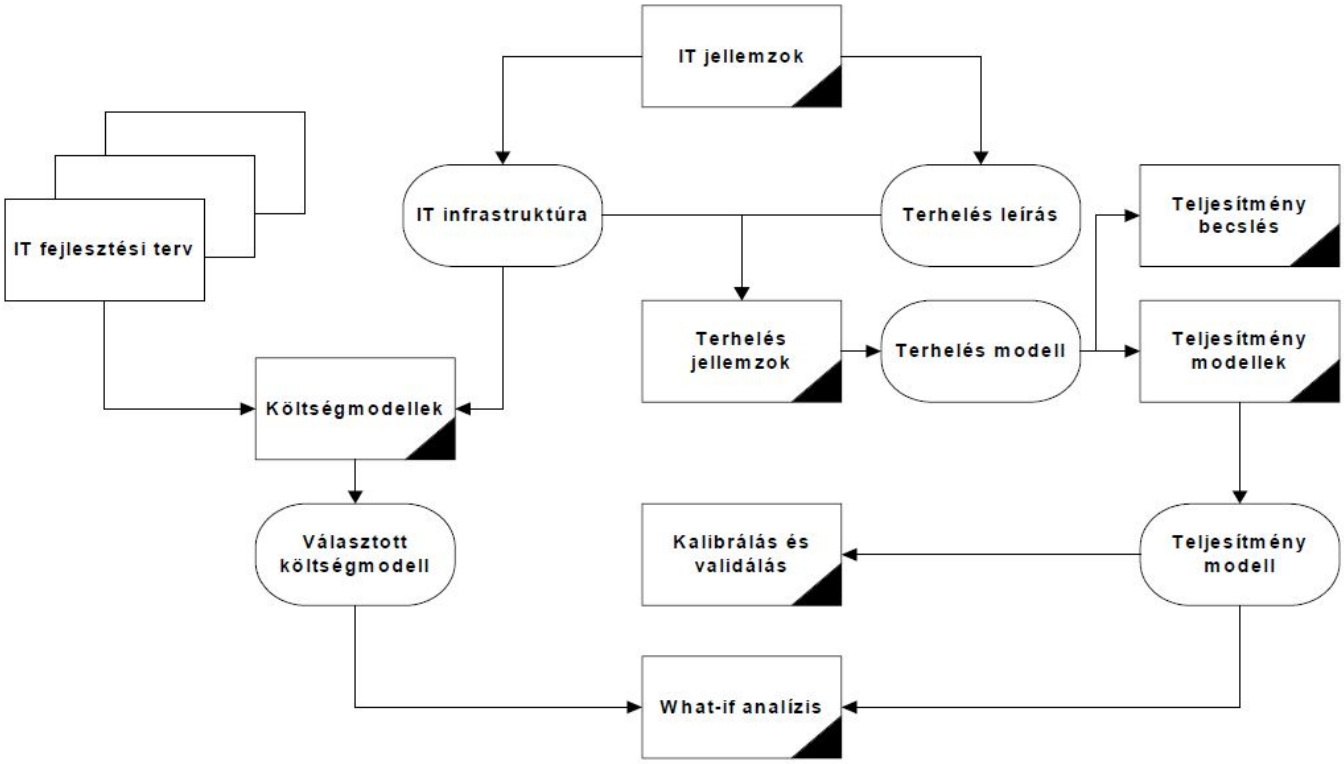
Klaszter: összetartozó, hasonló értékek csoportja
Klaszter közzépontja: centroid: ezzel helyettesíthető az adott csoport
Különböző távolságfogalmak: nem csak euklideszi
Algoritmus+távolság definíció: klaszter technika

Session azonosítás:

- cookie-k hsználatával: a kliens mellett az alkalmazás állapotát is tárolhatja
- autentikációs mechanizmusok, rejtett mezők HTML oldalak formjaiban, dinamikus URL-ek
- szerver logok alalpján, várakozási küszöbérték (threshold) használatával: ha ennél hosszabb a szünet két kérés (request) között akkor két külön session

C/S Interaction Sequence Diagram (CSISD)
szekvenciadiagramok minden lehetséges esetre
objektumok: kliens és a különböző szerverek
nyilak: üzenetküldés
üzenetek: [p,m] párok, ahol p az üzenet küldésének valószínűsége, m a mérete (byte)
minden lehetséges végrehajtási szekvencia kliensből indul ki és kliensben végződik
CSID: C/S Interaction Diagram: a szekvenciák összessége

Erőforrás szintű kapacitástervezés lépései:



Erőforrás szint:
Felhasznált IT erőforrások konkrét meghatározsa az előző modellek alapján.
Informatikai környezet meghatározása:
o infrastruktúra és

o terhelés leírása
o felhasználói funkciókhoz tartozó programok meghatározása

Infrastruktúra
o hardver (szerver gépek, diszk farmok, routerek, tűzfalak...)
o szerverek (Web ~, alkalmazás ~, adatbázis ~, DNS...)
o szoftverek (OS, middleware, adatbáziskezelő...)
o Hálózati kapcsolat, hálózati protokoll
o Fizetési szolgáltatás (Payment service)

Erőforrás szintű terhelés meghatározása
R: adott erőforrás
F: funkciók halmaza
E[terhelés$_R$] = $\sum_{|F|}$(gyakoriság$_F$*E[terhelés$_{F,R}$]) <- felhasználói, informatikai mérték

(csal, mert a terhelés nem additív, lehetnek kiugró erőforrásigények)

Informatikai környezet meghatározása:
Cél:meghatározni a funkciókhoz tartozó tranzakciókat (elemi lépéseket) és az ezek által használt erőforrásokat

Terhelés leírása
melyik szolgáltatás milyen erőforrásokat használ
Alapvető komponensek: tranzakciók
CSISD (szekvenciák) felírása, minden szerveroldali objektumhoz tranzakciónév
Komponensekhez gyakoriság (érkezési ráta) és erőforrásigények meghatározása
cél: szolgáltatások erőforrásifényeinek meghatározása

----------sdfg----------

TERHELÉS MODELLEK, ELŐREJELZÉS

Terhelés modell meghatározása:
- Adatgyújtés: benchmarkok, ökölszabályok, “best practices”, mérések
- Monitorozás: belső (szerver), külső (kliens, hálózat)
- Adatok rendszerezése: klaszterezési technikák stb.

Mérés:
- felhasználó szemszögéből
- üzemeltető szintjén
- alkalmazástól függően

Terhelésgenerálás:

- Felhasználói viselkedés rögzítése: cookie, SSL
- Visszajátszás tipikus paraméterei: párhuzamosan futó kliensek száma, lekérdezések száma, timeout, cache használata, stb.
- Emulált kliensek: Web Performance Tools, ASP.NET - ANTS Profiler, Rational Performance Tester, Jmeter, Selenium, stb.

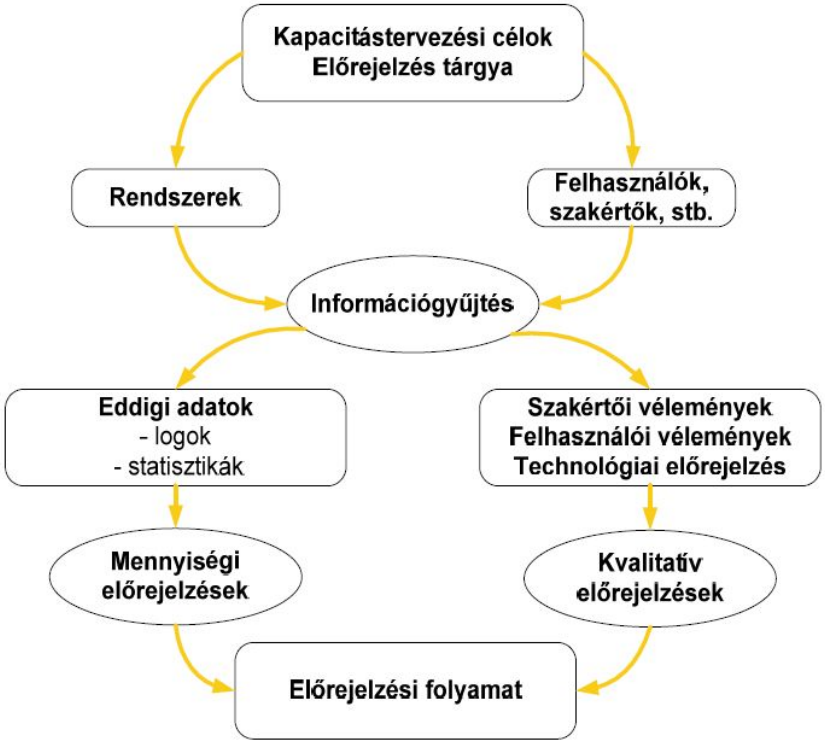
Windows teljesítményszámlálók:

- Átlagszámlálók (Average counters)
- Különbségszámlálók (Difference counters)
- Pillanatnyi érték számlálók (Instantaneous counters)
- Százalékszámlálók (Percentage counters)
- Gyakoriság számlálók (Rate counters)

Konkrét teljesítményszámlálók:

Processor(_Total)\% Idle Time
o A processzor szabad erőforrásának mértéke
Processor(_Total)\% Processor Time
o A processzor aktuális kihasználtsága
ASP,NET v2,0,50727\Request Execution Time
o Web kérések végrehajtási ideje
SQLServer:Databases(_Total)\Transactions/sec
o Adatbázis tranzakciók másodpercenként
SQLServer:General Statistics\Logins/sec
o Adatbázis bejelentkezések száma
Memory(_Total)\Available Memory
o Rendelkezésre álló memória
ASP,NET\Requests Current
o Aktuális kérések száma

Terhelés előrejelzési stratégia



Hierarchikus előrejelzési modell:

Terhelés előrejelzés: pl. lineáris regresszió

Terhelés típusai:

- Trend: állandó jellegű változás
- Időszakos: visszatérő, periodikus hatások
- (Statikus)
- (Véletlenszerű)
- Előrejelzési módszer a konkrét mintához

Terhelés előrejelzési technikák:

- Regresszió: lineáris, exponenciális, logaritmikus
- Mozgó átlagok
- Exponenciális csúszóablak
- Könnyen automatizálhatók: Matlab, R, MS Excel

Regressziós módszerek

- Alapelv:

$$Y_t = f(\bullet) + \varepsilon_t$$

Véletlen változó — Közelítés — Hiba

$$Y = f(X_1, X_2, ..., X_n)$$

Jósolt esemény — Megfigyelhető változók

- Átlagos hiba (mean error)

$$ME = \frac{\sum_{t=1}^{n} (Y_t - F_t)}{n}$$

Becsült érték — Mért érték

TELJESÍTMÉNYMODELLEZÉS

Már tervezési időben készüljünk a terhelésre!

<u>Terhelésmodellezés:</u> Az eddigi modellek kiegészítése nemfunkcionális követelményekkel: időzítések, erőforrások, stb.
Célja:
• A rendszer teljesítményének értékelése a tervezési fázisban
• Kritikus részek azonosítása
• Skálázás, méretezés

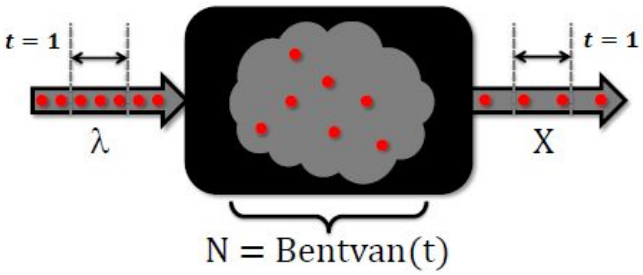
Alapmodell: Bejött(t) - Kiment(t) = Bentvan(t)

Ráták:
Érkezési ráta: egységnyi idő alatt érkező kérések

$$\lambda = \frac{Bejött(t)}{t} \qquad [\lambda] = \frac{1}{s}$$

Átbocsátás: egységnyi idő alatt feldolgozott kérések

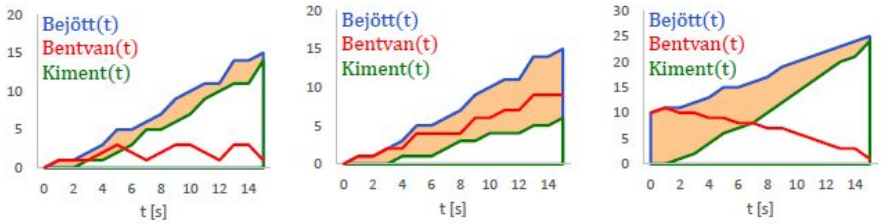
$$X = \frac{Kiment(t)}{t} \qquad [X] = \frac{1}{s}$$



Egyensúlyi állapot: Bentvan(t) közel állandó
Átlagos értékek csak ilyenkor használhatóak.

$$\lambda = X$$

$\lambda = X$ $\lambda > X$ $\lambda < X$

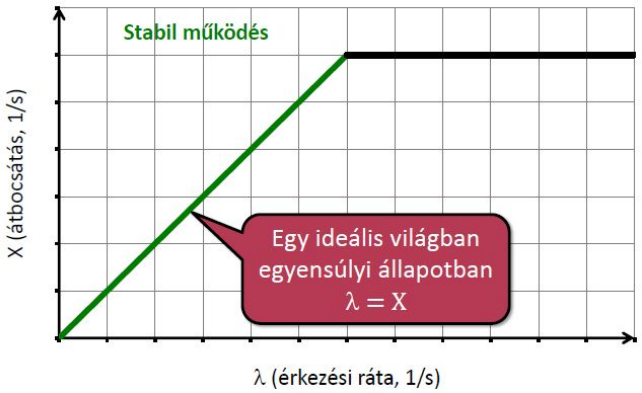


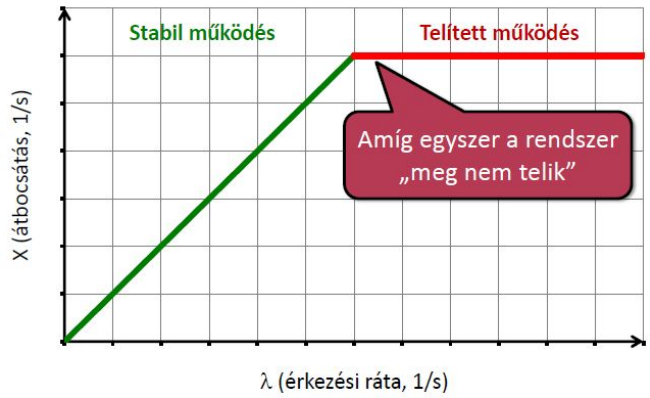
DDoS - Denial of Service Attack

- tömeges kérésgenerálás > rendszer lefoglalása
- a leterhelt rendszert könnyebb támadni
- teljes szolgáltatások leállíthatóak

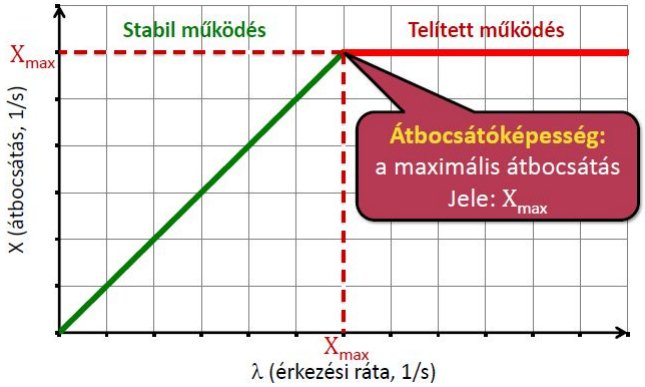
Terhelési diagram

Az érkezési ráta és az átbocsátás kapcsolata, átbocsátóképesség

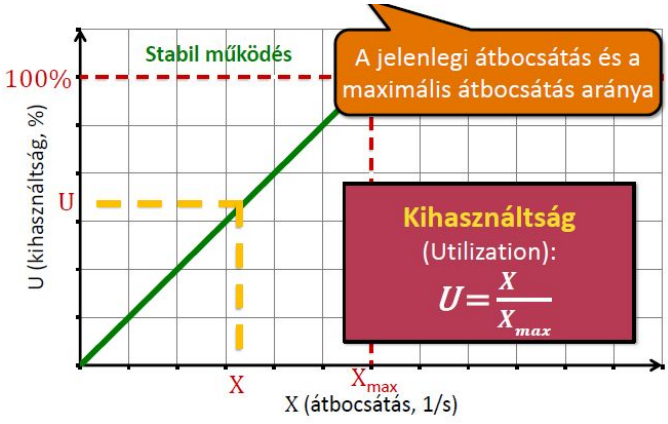




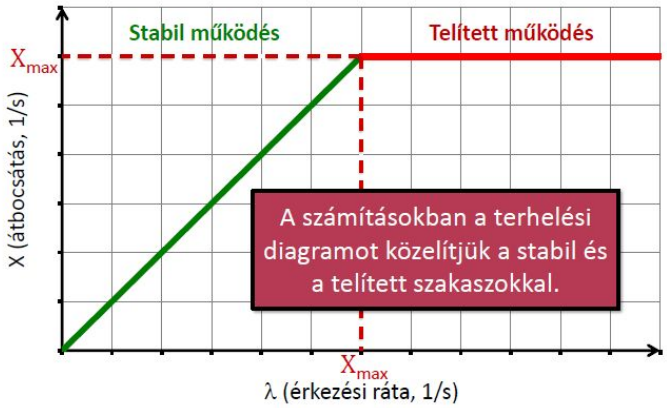
Átbocsátóképesség:



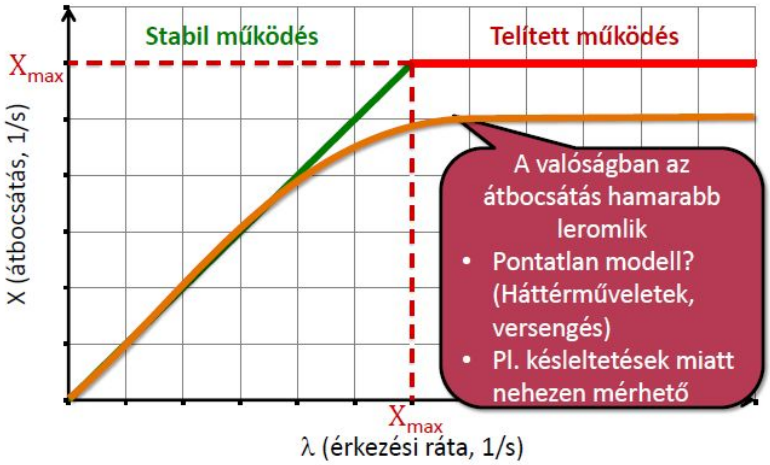
Kihasználtság:

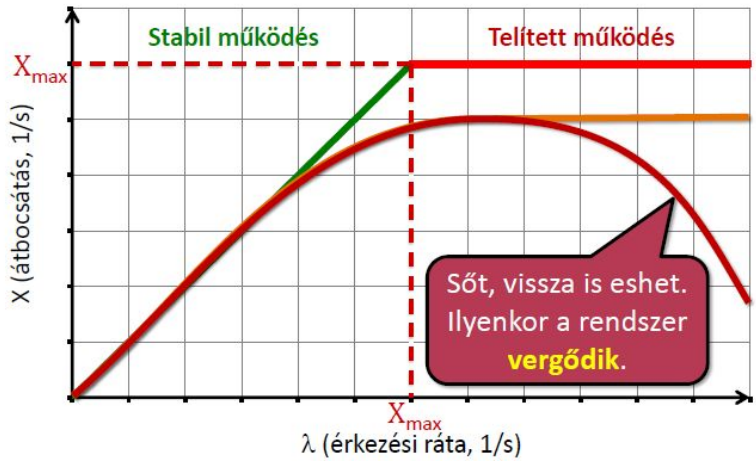


Közelítő terhelési függvény



Valós terhelési diagram:





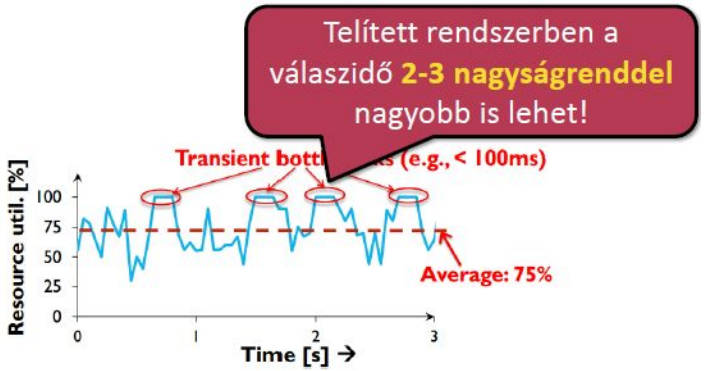
A modellben elhanyagolt hatások:

- Taszkváltások számításigénye: takarítás, előkészület

- Erőforrásváltás számításigénye
- Többszörös telítés: egyszerre több erőforrás is telített

Elriasztott kérések esete: Igények csak a fanatikus klienseknél függetlenek a sor hosszától.

A terhelésingadozás hatása: átlagos értékek vs valós terhelés



Következmény: nem érdemes 40-60%-nál nagyobb kihasználtságra tervezni
A túlterhelt rendszer teljesítménye drasztikusan leeshet, akár össze is omolhat

A létszámkorlátos megoldás célja egyensúlyi állapotban tartani a rendszert!

- ez az alapja a DDoS elleni védekezésnek is
-

<u>Erőforrásmodellezés:</u> Miből származik az átbocsátás korlátja?

Kizárólagos erőforrás törvénye: Ha a kérések közül egyszerre legfeljebb egy futhat, a többi sorban áll
Ekkor: átlagos végrehajtási idő mellett

$$X_{max}^{(1)} = \frac{1}{T}$$

Sorbanállási idő: várakozás erőforrásra
Végrehajtási idő: kérés feldolgozása
Válaszidő: sorbanállási + végrehajtási

Kizárólagos erőforrás kihasználtsága: Az átbocsátás és az átbocsátóképesség aránya

$$X_{max}^{(1)} = \frac{1}{T} \quad \Rightarrow \quad X_{max}^{(1)} \times \mathrm{T} = 1$$

$$X_{max}^{(1)} = \frac{1}{T} \quad \Rightarrow U = {}^{X}/_{X_{max}^{(1)}} = \mathrm{X} \times \mathrm{T}$$

A kihasználtság képlete:

$$\mathbf{U = X \times T}$$

Az egységnyi idő alatt beérkező X taszk végrehajtása T végrehajtási idővel az egységnyi idő hányadrészét teszi ki?

Skálázás:
Vertikális skálázás(Scale-up):

- A feldolgozóegység teljesítményét növeljük
- erősebb CPU, több RAM
- egyszerű és nagyszerű
- technológiai korlátok

Horizontális skálázás (Scale-out):

- A feldolgozóegységek számát növeljük
- több CPU mag, több szerver
- elvileg korlátlanul növelhető
- plusz bonyolultság

K erőforráspéldány használata
Ha az összes kérés közül egyszerre legfeljebb K futhat, a többi sorban áll

Ekkor: (T: átlagos végrehajtási idő)

$$X_{max}^{(K)} = K \times X_{max}^{(1)} = \frac{K}{T}$$

Több erőforráspéldánnyal és párhuzamosítással a rendszer skálázható

K erőforráspéldány kihasználtsága:

$$X_{max}^{(K)} = \frac{K}{T} \Rightarrow X_{max}^{(K)} \times T = K \; ?$$

$$X_{max}^{(K)} = \frac{K}{T} \Rightarrow U = {}^{X}/_{X_{max}^{(K)}} = \frac{X \times T}{K}$$

A kihasználtság képlete:

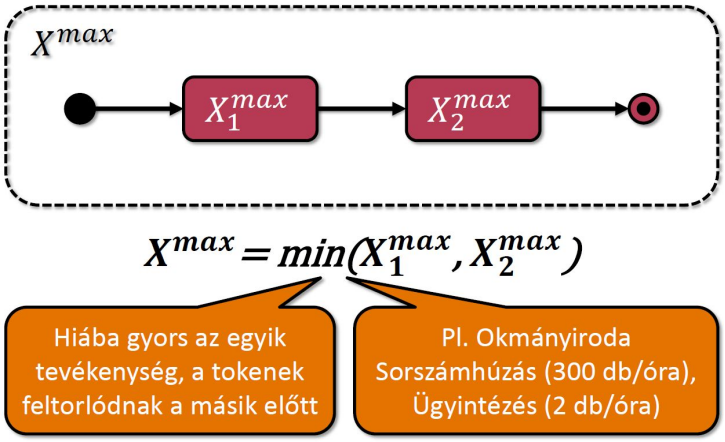
$$U = \frac{X}{K} \times T$$

K időegység hányadrészében dolgozna 1 példány?
Az egy példányra eső átlagos átbocsátás mellett mekkora egy erőforráspéldány kihasználtsága?

Egyensúlyi állapot:
o Átlagos értékekkel számolunk
o $\alpha = X$ (érkezési ráta = átbocsátás)
Átbocsátóképesség:
o Az elérhető maximális átbocsátás
o Xmax= K/T(K erőforráspéldány esetén)
Kihasználtság:
o Az átbocsátás és az átbocsátóképesség aránya
o $U = X/K \times T$ (K erőforráspéldány esetén)

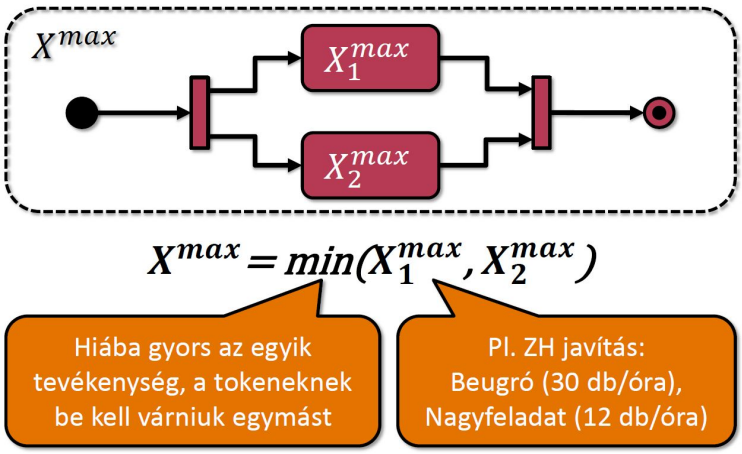
Folyamatmodellek elemzése: Hogyan számoljuk ki egy összetett folyamat átbocsátóképességét? Általában tevékenységekhez rendelünk erőforrást, átlagos végrehajtási ideje is adott

Szekvenciális komponálás:



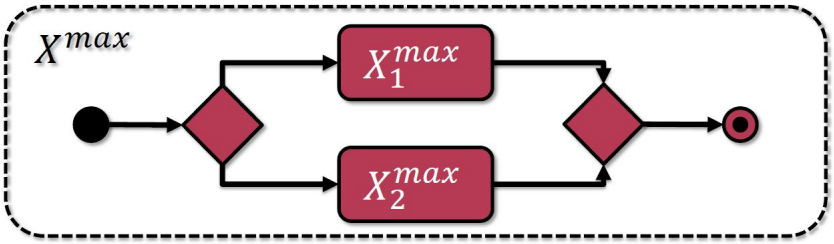
Szűk keresztmetszet: A minimumhelyet adó tevékenység (vagy az ahhoz rendelt erőforrás)

Párhuzamos komponálás:



Szűk keresztmetszet: A minimumhelyet adó tevékenység (vagy az ahhoz rendelt erőforrás)

Komponálás szabad választással:



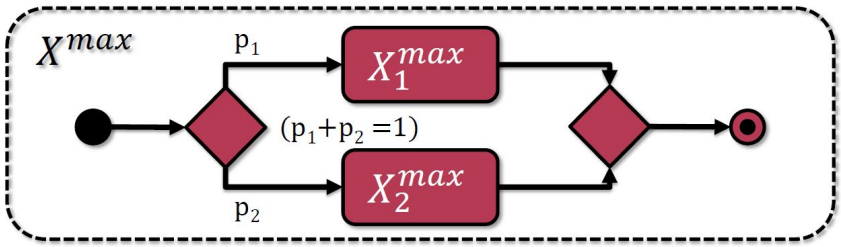
$$X^{max} = X_1^{max} + X_2^{max}$$

A tokenek mindkét irányba mehetnek: ha az egyik tevékenység telítésben van, a másik még fogadhat tokent.

Pl. Áruház: K db pénztár, mind 10 db/óra

Megj: van, amikor a lépéshez rendeljük majd az erőforrások számát, és nem ugyanazt a logikai lépést tüntetjük fel többször a modellben (ld. Szimuláció előadás).
Feltétel: minden erőforráson ugyanannyi ideig tartson

Komponálás kötött arányú választással

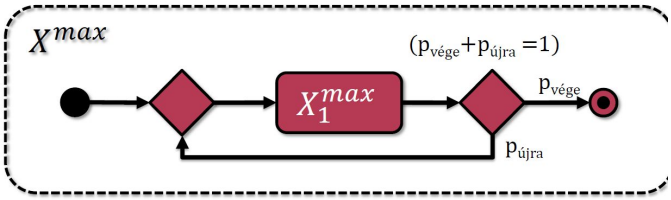


$$X^{max} = min\left(\frac{1}{p_1} \times X_1^{max}, \frac{1}{p_2} \times X_2^{max}\right)$$

A tokenek p_1 és p_2 valósínűséggel választják az első ill. második tevékenységet. A teljes folyamatból tehát $\frac{1}{p_1}$ ill. $\frac{1}{p_2}$ tokenből egy jut az első ill. második tevékenységre.

Szűk keresztmetszet:
A minimumhelyet adó tevékenység (vagy az ahhoz rendelt erőforrás).

Komponálás ciklussal:



$$X^{max} = \frac{1}{\frac{1}{p_{vége}}} \times X_1^{max} = p_{vége} \times X_1^{max}$$

Az $\frac{1}{p_{vége}}$ érték az iterációk várható száma
(lásd később a Valószínűségszámítás tantárgyban).

Pl. Felhasználó a rendszerben:
10% eséllyel kilép, 90% eséllyel új kérés 15 kérés/s, átlagosan 10 kérés/munkamenet

Vizitációs szám: megmutatja, hogy a folyamat végrehajtása során átlagosan hányszor fut le az adott tevékenység/alfolyamat

- választás esetén maga a döntési valószínűség
- ciklus esetén a várható iterációk száma

Választás: $$X^{max} = min\left(\frac{1}{p_1} \times X_1^{max}, \frac{1}{p_2} \times X_2^{max}\right)$$

Ciklus: $$X^{max} = \frac{1}{\frac{1}{p_{vége}}} \times X_1^{max} = p_{vége} \times X_1^{max}$$

Átbocsátóképesség a vizitációs szám ismeretében:

$$X^{max} = \frac{1}{v} \times X_1^{max}$$

Átbocsátóképesség a vizitációs szám ismeretében:

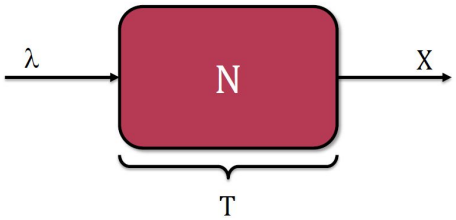
$$\frac{1}{X^{max}} = v \times \frac{1}{X_1^{max}}$$

Végrehajtási idő a vizitációs szám ismeretében:

$$T_{folyamat} = v \times T_{taszk}$$

A Little-törvény (FONTOS!)

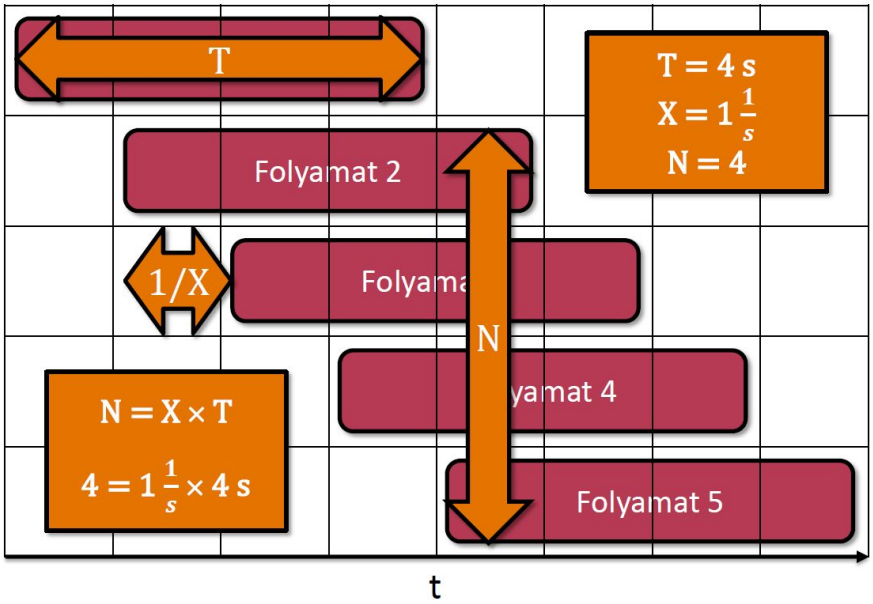
- **λ**: érkezési ráta
- **X:** átbocsátás
- **T:** rendszerben tötött idő
- **N:** rendszerben lévő tokenek száma



Egyensúlyi állapotban (λ = X) igaz a Little-törvény:

N = X x T

A törvény szemléltetése:



K erőforráspéldány: maximum K folymatpéldány állhat végrehajtás alatt

A Little-törvény megadja a végrehajtás alatt álló folymatpéldányok számát (N)

$$U = \frac{X}{K} \times T = \frac{X \times T}{K} = \frac{N}{K}$$

Kihasználtság K erőforráspéldányra

Little-törvény ($N = X \times T$)

Little-törvény gyakorlati példák:

1. <u>példa:</u>

Erőforrás: diszk

40 kérést szolgál ki másodpercenként (nincs átlapolódás)

1 kérés kiszolgálása átlagosan 0,0225 másodpercig tart

<u>Kérdés: Mekkora a kihasználtság?</u>

U = X × Tdiszk = 40 kérés/mp × 0,0225 mp = 90%

Sorban állás is van a diszk előtt,

Kérések átlagos száma a rendszerben: 4

Kérdés: Átlagos rendszerben tartózkodási idő (Trendszer)?
Rendszer
N = X × T → Trendszer = 4 kérés / 40 kérés/mp = 0,1 mp

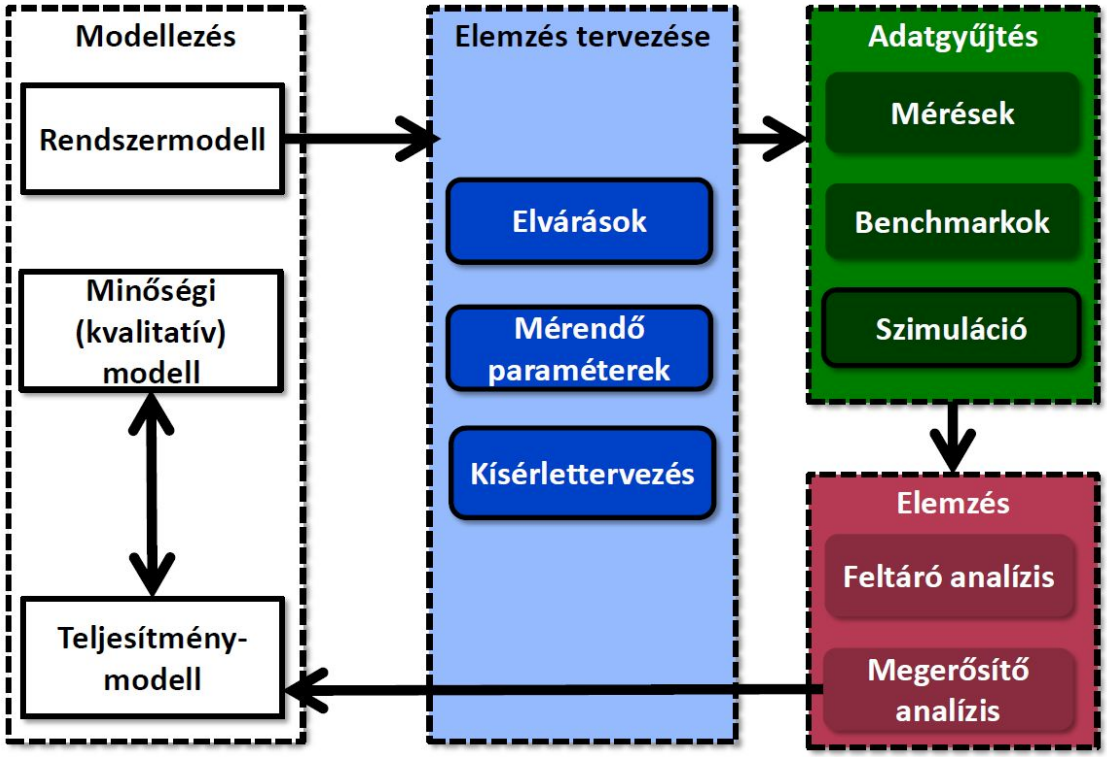
Kérdés: Átlagos sorbanállási idő(Tvárakozás)?
(Trendszer – Tdiszk) 0,1 mp – 0,0225 mp = 0,0775 mp

Kérdés: Kérések átlagos száma a sorban?
(Nrendszer– Ndiszk)
4 kérés – 0,9 kérés = 3,1 kérés

Terhelés változása:
Valójában nem feltétlenül kiszámítható a terhelés.
Valójában a rendszer viselkedése időben változik.
>műszaki hatásai vannak

VIZUÁLIS ADATELEMZÉS

Rendszermodelltől a teljesítménymodellig



A folyamat alapja az interakció
1. Adatvizualizáció
– több ábra együttes vizsgálata
2. Vizuális kiértékelés
– emberi kognitív képességek használata
3. Vizuális kiválasztás és manipuláció
4. Interpretáció, korreláció más modellekkel, kiértékelés

Numerikus és kategorikus változók
Numerikus (numerical): az alapvető aritmetikai műveletek értelmesek
Katorikus(categorical): csak megkülönböztetés miatt, már reláció sincs

Numerikus változók:
Folytonos: mért - tetszőleges értéket felvehet (adott tartományon belül, adott pontosság mellett)
Diszkrét: számolt - véges sok értéket vehet fel adott tartományban

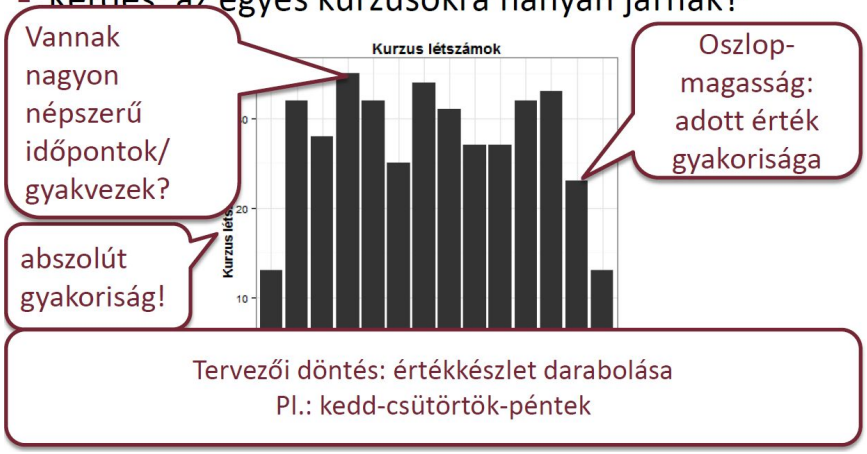
Kategorikus változók:
- Rendezett (ordinális): teljes rendezés az értékeken, pl.: szállodai csillagok
- Nem rendezett (reguláris)

9. Ajánlanád-e a tárgyat másoknak?
- ○ Mindenkit rábeszélnék
- ○ Nyugodtan ajánlanám
- ○ Esetleg ajánlanám
- ○ Inkább lebeszélném róla
- ○ Feltétlenül lebeszélném
- ○ Nem kívánok válaszolni

1 változó - eloszlásokra:

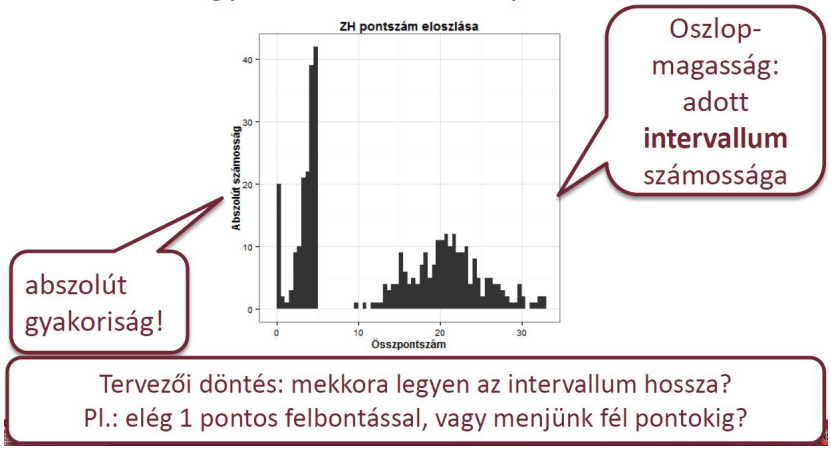
Oszlopdiagram:

- Bemenő változó: kurzus kód
- Kérdés: az egyes kurzusokra hányan járnak?



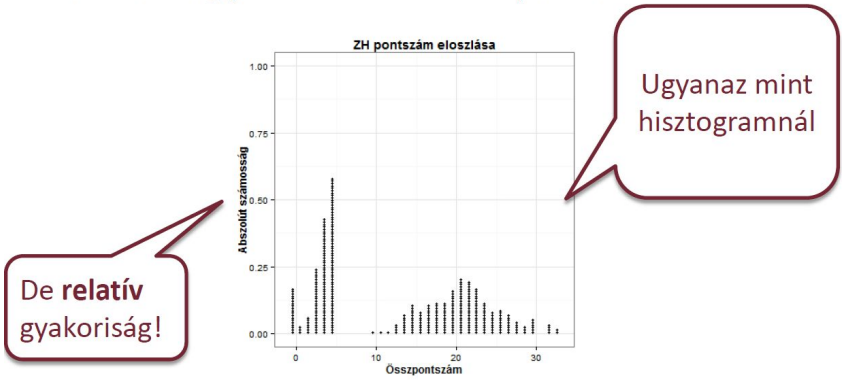
Hisztogram:

- Bemenő változó: ZH összpontszám
- Kérdés: hogyan alakultak a ZH pontszámok?



Relatív gyakoriságok:

- Bemenő változó: ZH összpontszám
- Kérdés: hogyan alakultak a ZH pontszámok?

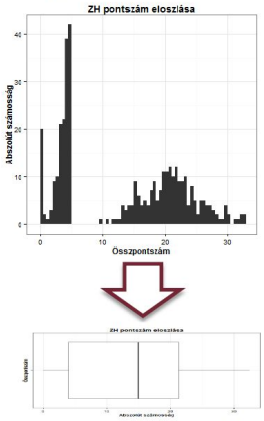


Egyszerű statisztikai jellemzés:

- Hol van az adatok “közepe”?
- Mennyire “szórtak” az adatok?
- Vannak-e kilógóak?

Boxplot:

- Bemenő változó: ZH összpontszám
- Kérdés: hogyan alakultak a ZH pontszámok **úgy nagyjából**?



Egyfajta absztrakció itt is: legyenek intervallumok, felesleges minden pontot kirajzolni

EZ WTF?

Folytonos megfigyelések jellemzése:

A “központ” jellemzése

- átlag, medián, módusz
- {3,4,4,5,5,6,10,20}
    - átlag: ~7,125
    - medián: 5 (középső elem)
    - módusz: 4 és 5 (leggyakrabban előforduló elem)

A “terjedelem” jellemzése

Percentilis

- Az n-edik perentilisnél az adatok n%-a kisebb
- {3,4,4,5,5,6,10,20}
    - 50. percentilis: 5
    - 25. percentilis: 4
    - 75. percentilis: 6

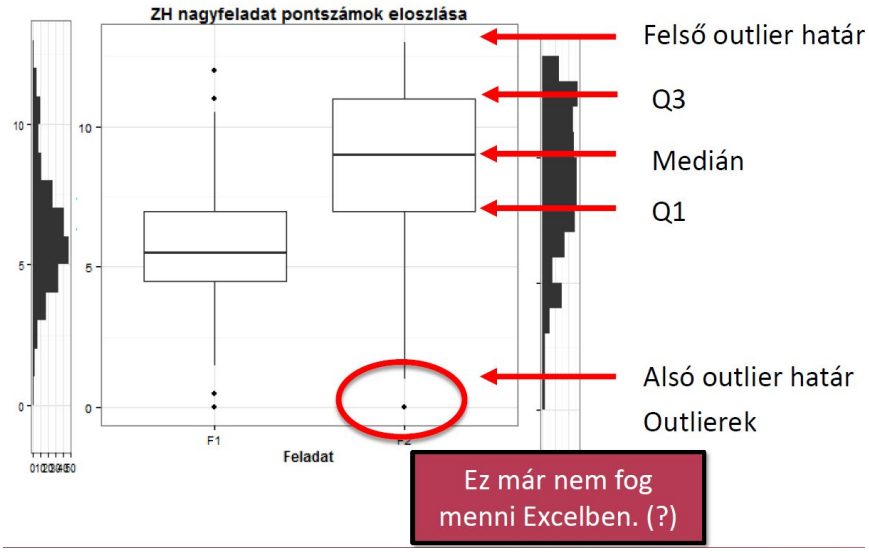
Kvartilis:

Q1: 25. percentilis

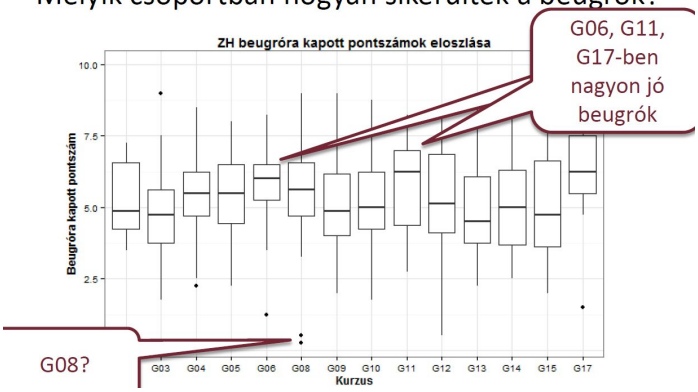
Q3. 75. percentilis

Q2: medián

Boxplot



- Melyik csoportban hogyan sikerültek a beugrók?

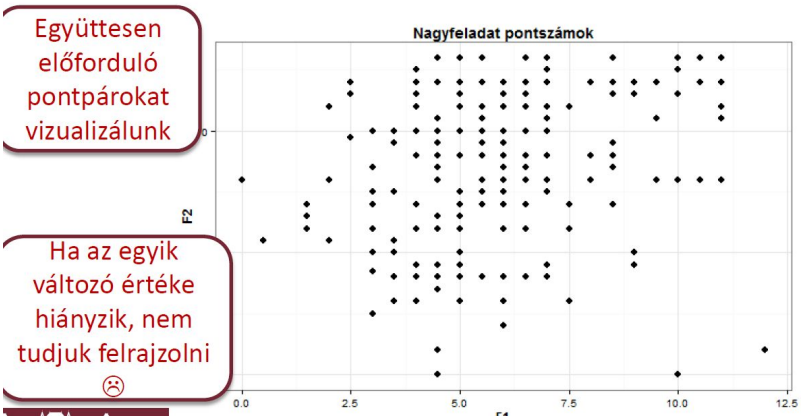


Absztrakció: A boxplottal fontos információt is veszíthetünk

2 változó kapcsolata:

Pont - pont diagram (scatterplot)

- Bemenő változó: nagyfeladatokra kapott pontok
- Kérdés: hogyan viszonyulnak egymáshoz?



Overplotting:

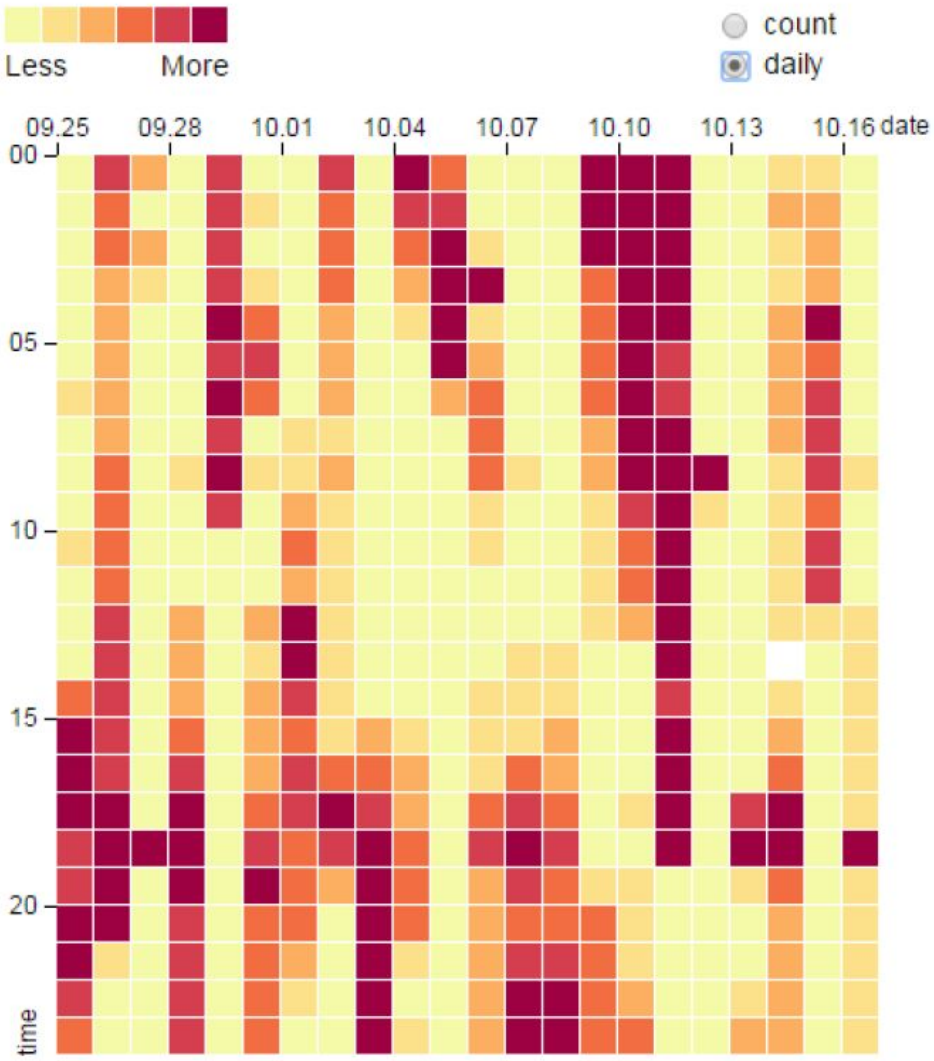
1. **Jitter**
2. **Átlátszóság**
3. **Méret**

Sok változó (>=3)

A grafikai objeltumok atrribútumait változtatjuk

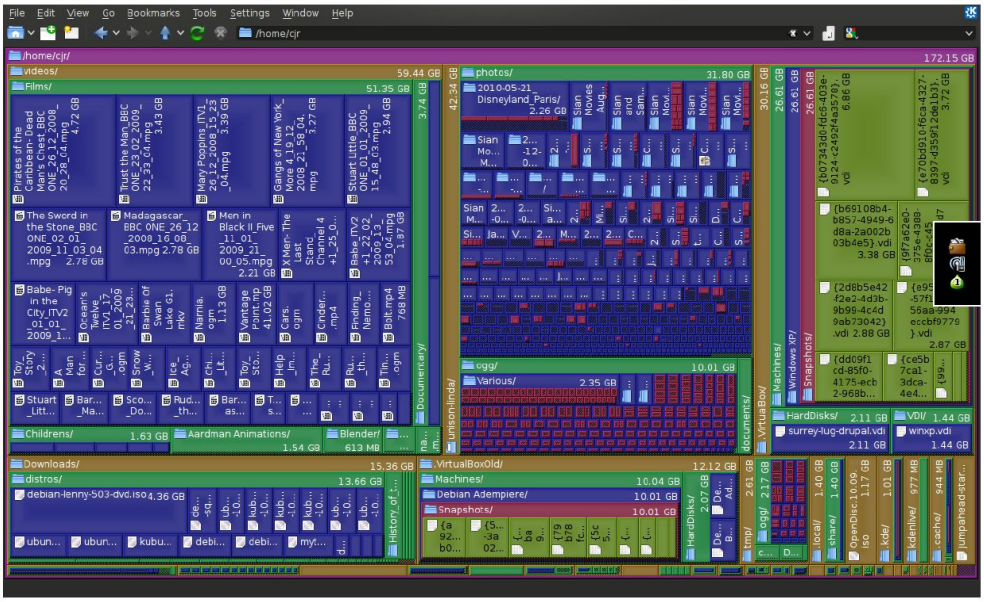
- szín
- méret
- textúra
- hely - ez triviálisnak tűnik, de a treemapnél van jelentősége
- Pl.: heatmap, bubble chart, treemap

Heat map: napi hőmérséklet

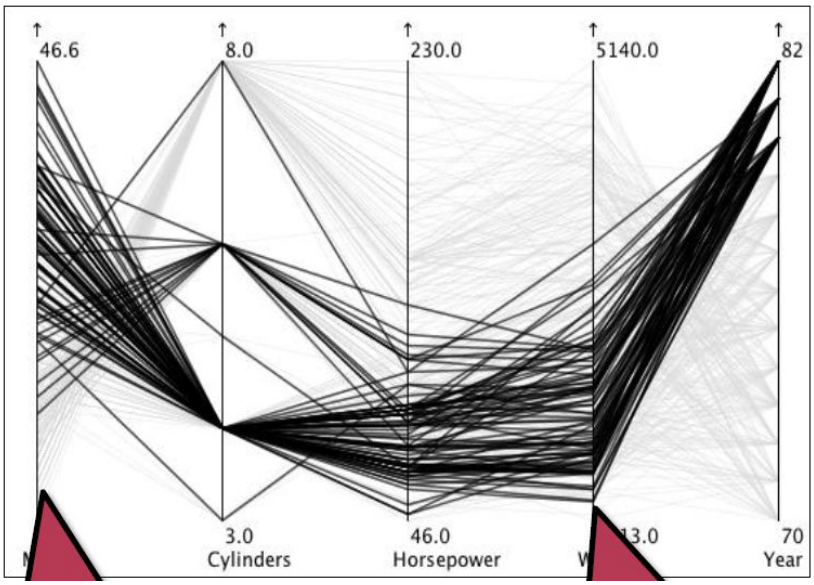


Bubble chart: átlagéletkor változása régiónként

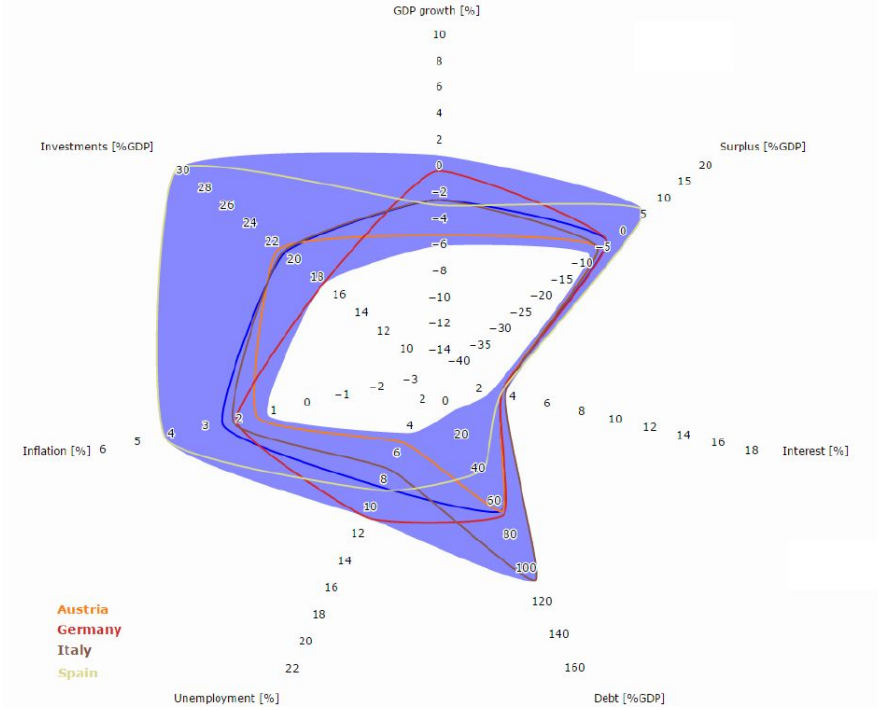
Treemap: állományrendszer



Párhuzamos koordináták:
Kodimenziós
Kompakt és skálázható
Koordináta sorrend
Ábrázolt összefüggés: rekordok/attribútumok hasonlósága
Adategysége: törött vonal - az egyes attribútumtengelyeken felvett értékek rendezett sorozata
Korlátok: tengelyek (attribútumok) más mértékegysége/nagyságrendje stb. torzíthat



Radar chart: egy párhuzamos koord. kiterjesztés



616 és 617 diákat mire véljem?

ADATELEMZÉSEK ÉS MÉRÉSEK

- regresszió, átlagolás
- következtető elemzés
- benchmarking
- kísérlettervezés

Regresszió:
f függvény,
bemenet: az attribútumok értéke
kimenet: megfihgyelések legjobb közelítése
“ökölszabály”
Példa: testtömeg/magasság együttes eloszlás egyenesre illeszthető

Regressziós módszerek:

- Alapelv:

$$Y_t = f(\bullet) + \varepsilon_t$$

Véletlen változó — Közelítés — Hiba

$$Y = f(X_1, X_2, \ldots, X_n)$$

Jósolt esemény — Megfigyelhető változók

- Átlagos hiba (mean error)

$$ME = \frac{\sum_{t=1}^{n} (Y_t - F_t)}{n}$$

Becsült érték — Mért érték

Lineáris regresszió: Egszerű függvény illesztése az adatokra, nem vár alapvető változást a rendszer viselkedésében

Y = a + bX
legkisebb négyzetek módszere: keressük azokat az a,b paramétereket (itt: a: eltolás, b: meredekség), amelyekre:

$$SSE = \sum_{t=1}^{n} \varepsilon_t^{\,2} = \sum_{t=1}^{n} \left(Y_t - F_t\right)^2$$ minimális (Sum of Squared Errors)

cél: $$\sum_{t=1}^{n} \left(Y_t - F_t\right)^2 = \sum_{t=1}^{n} \left[Y_t - \left(a + bX_t\right)\right]^2$$

Lineáris regresszió: legjobban illeszkedő egyenes

$$min(\sum_{i=1}^{n} |Y_i - \hat{\mu}(x_i)|^2), \text{ ahol } \hat{\mu}(x) = ax + b$$

DE:
Ansacombe's quartet: minőségileg különböző adatok, azonos regressziós egyenes

Itt valami wild black magic történik több dián keresztül…kurvára nem tudom, mit és hogyan kellene kiírni
de szerintem ez talán annyira nem is fog kelleni, mert nem rémlik, hogy lett volna ilyen gyak

Korrelációs együttható (négyzete)

$$R^2 = \frac{\sum_{t=1}^{n} \left(F_t - \overline{Y}\right)^2}{\sum_{t=1}^{n} \left(Y_t - \overline{Y}\right)^2}$$

- o a változó becsült és tényleges értékének kapcsolata
- o 0 és 1 közti érték
- o 0: nincs kapcsolat
- o 1: függvényszerű kapcsolat

Nemlineáris módszerek:

Exponenciális közelítés: Yt = axb^t

Exp. terhelés példa: Yt = axe^bt

Mozgó átlagok módszere

- rövid távú előrejelzésre jó
- egyszerre egy értéket ad meg
- a becsült érték az utolsó n érték átlaga

$$F_{t+1} = \frac{\sum_{i=t}^{t-n+1} Y_i}{n}$$

ahol Y_t a t. időpontban mért érték

F_{t+1} a becsült érték

n tipikusan 3 és 10 között van

(becslés hibája ne legyen túl nagy)

Exponenciális csúszóablak:

- egy értéket ad meg az előző méréseket átlagolva
- későbbi mérés nagyobb súllyal
- súlyozza a mérési hibát
- rövid távú előrejelzésre

$$F_{t+1} = F_t + \alpha\left(Y_t - F_t\right)$$

ahol F_t :a t. időpontra becsült érték

Y_t : t. időpontban mért érték

$Y_t - F_t$: mérési hiba a t. periódusban

α : súlyozás ($0 \leq \alpha \leq 1$, gyakorlatban $0.05 \leq \alpha \leq 0.3$)

eddig tartott a borzalom…

BENCHMARKING

Célok: szoftver/hardver eszközök teljesítményének összehasonlítása

- Döntéstámogatás
- Teljesítménytesztelés

Elvárások:

- Ismételhetőség
- Reprodukálhatóság
- Relevancia
- Szabványok/megállapodások betartása
- Általánosított felhasználói eset

Benchmark terhelési modellek:

- Tudományos/műszaki rendszerek: nagy mennyiségű adat feldolgozása
- Tranzakciókezelés (OLTP): kliens-szerver környzet, sok gyors, párhuzamos tranzakció
- Batch jellegű adatfeldolgozás: riport készítés nagy mennyiségű adatból
- Döntéstámogatás: kevás, bonyolult lekérdezés, ad hoc műveletek, sok adat (OLAP)
- Virtualizáció

Mérendő paraméterek

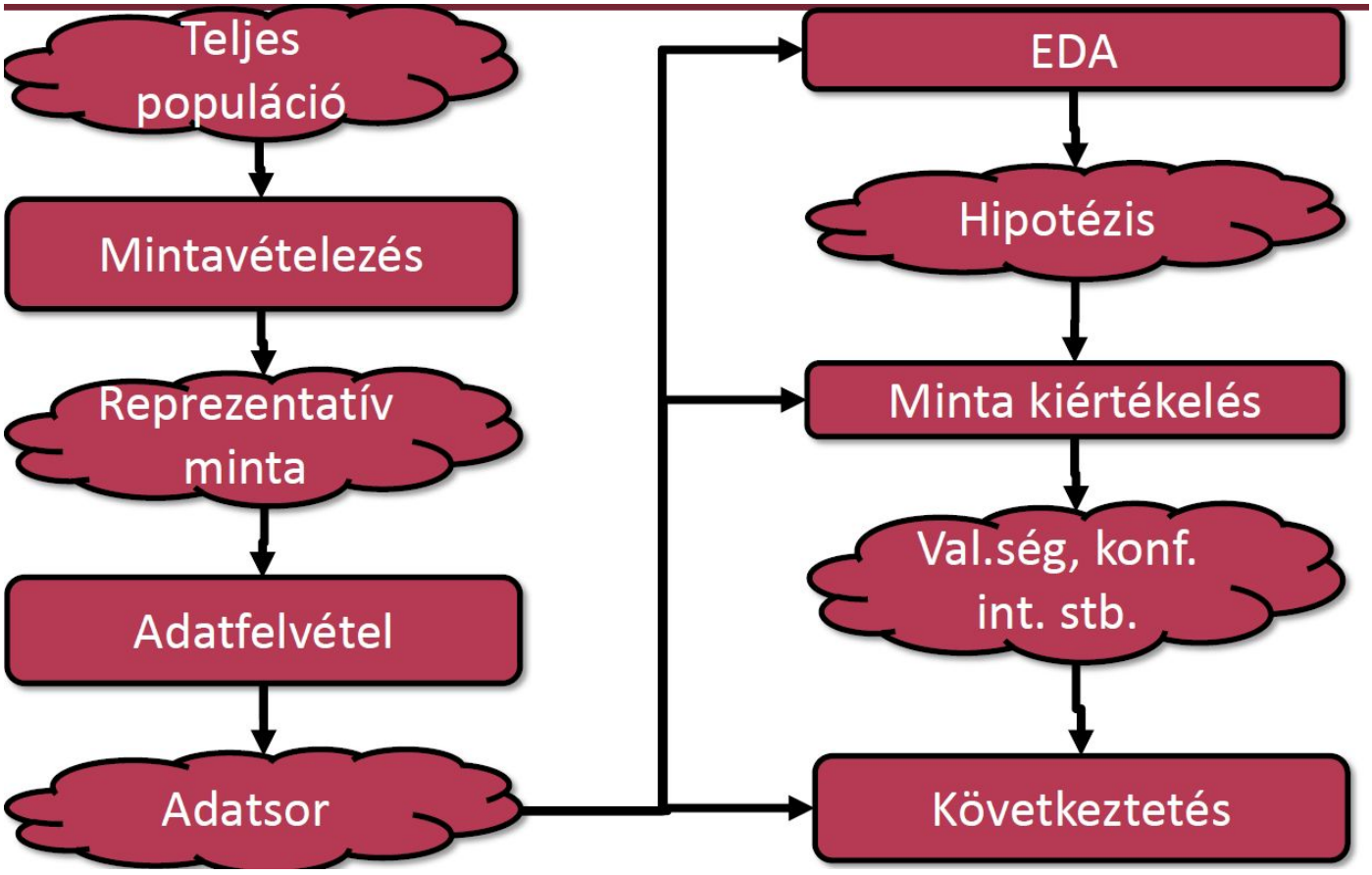
- Futási idő
- Tranzakciósebesség
- Áteresztőképesség(terhelés függvényében): feldolgozott adatmennyiség/futási idő
- Válaszidő(terhelés fóggvényében): felhasználók, tranzakciók száma
- X-Percentil: egy adott halmaz X százaléka ez alatt az érték alatt van

Benchmark elvégzése:

- **Relevancia biztosítása:**
  - tényleg aut az alkalmazást mérjük amit kell
  - terhelésgenerálás jellege közelítse a valósi terhelést
  - minimalizáljuk a zavaró tényezőket

PL: SPEC(CPU intenzív), TPC(adatbáziskezelő rendszerek mérése)

<u>Következtető statisztika:</u>



Mi az az EDA amúgy???

Ökölszabályok

LLN (Law of Large Numbers): ha a kísérletek száma tart a végtelenhez, az előfordulási gyakoriság az elméleti valószínűséghez konvergál

Döntési bemenet
- o Valami küszöbérték

Adatsor típusa
- o Megfigyelési tanulmány (observational study)
  - • A köztes változók kiléte bizonytalan
  - • Csak korreláció, kauzális következtetések nem
- o Kísérlet (experiment)
  - • A köztes változókat kiszűrtük (mintavételezés!)
  - • Kauzális következtetések is

Kísérlettervezés

Cél: a modell paraméterezése a valóság alapján

Információt kísérletek révén szerzünk

- Pontosan mit szeretnénk tudni?
- Ehhez milyen megfigyelést, hányszor kell elvégezni?
- A kapott eredményekből mire lehet következtetni?
- (statisztikai) **kísérlettervezés (Design of Experiment, DOE)**
    - hatékony eljárás a kísérletek tervezésére és elemzésére
    - valós és objektív konklúziók levonásához

Kísérletterv: még a kísérlet elvégzése előtt

Haszna:

- alternatívák közötti választás
- érzékeny paraméterek, kulcsfaktorok
- megfelelő célérték, változékonyság csökkentése
- robosztussá tétel

Fontos:

- világos cél, egyértelmű eredmények
- kis méret, alacsony költség
- valós viszonyok

Tapasztalati átlag, szórás

Valószínűségi változó: E (vizsgálandó jelenség)

- Várható érték: $\mu = \mathrm{E}(X)$ átlagos viselkedés)
- Szórás: $\sigma = \sqrt{\mathrm{E}(X-\mu)^2}$ (eltérések mértéke)

Tapasztalai átlag:

Módszer: számtani átlag képzése

- ismételt megfigyelések
- egymástól függetlenül
- azonos feltételek mellett

Kérdés:

- hány megfigyelés kell végezni?
- A tapasztalati átlag mennyire jellemzi a valódi várható értéket?

Először tisztázandó:

A tapasztalati átlag eloszlása

Tapasztalati átlag eloszlása
kísérlet = megfigyelések sorozata
Megfigyelések sorozatának tapasztalati átlaga:

- egy jellemzőt t db független megfigyeléssel mérve
- majd a mért értékeket átlagolva kapott eredmény

Centrális határeloszlás tételéből következik:

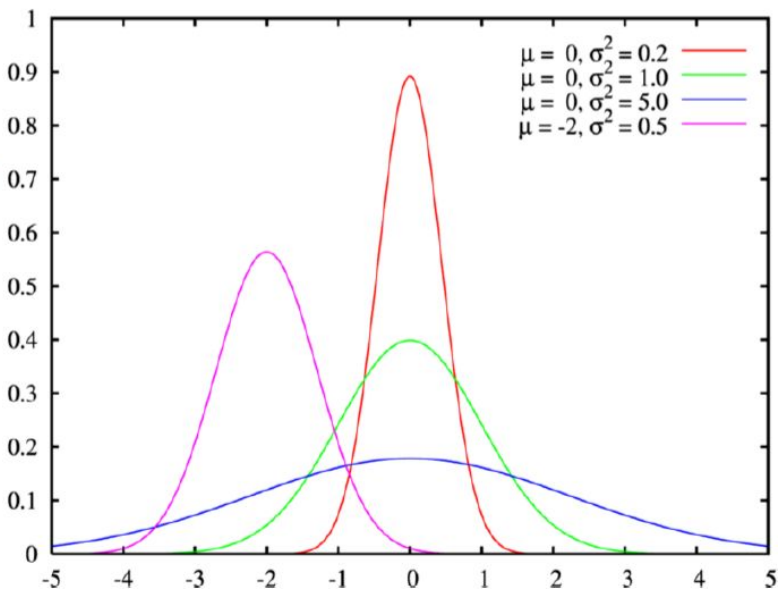
- tetszőleges eloszlású jellemző (de legyen véges m várható értékű és s szórású)
- tapasztalati átlaga t-->∞ esetén közelítőleg
- normális eloszlású
- μ = m várható értékkel és σ = s/√t szórással

Ökölszabály:
ismert szórásnál t>30
ismeretlen szórásnál t>100
után kezd elfogadható lenni a közelítés

A normális (Gauss) eloszlás

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

- Paraméterek
  - Várható értéke μ
    - → **m** a mi esetünkben
  - Szórása σ
    - → $^{s}/_{\sqrt{t}}$ esetünkben



A várható érték körül koncentrálódik

A normális eloszlású változó…

- az esetek **68%**-ában legfeljebb **1**σ messze kerül μ-től
- az esetek **95%**-ában legfeljebb **2**σ messze kerül μ-től
- az esetek **99,7%**-ában legfeljebb **3**σ messze kerül μ-től

Konfidenciaintervallumok:
Ha tehát a tetszőleges eloszlású, s szórású vizsgált jellemzőről t db (>30) megfigyelést végzünk
A tapasztalati átlagáról…
o 68% biztonsággal kijelenthető, hogy legfeljebb s/√t pontatlansággal becsli m értékét
o 95% biztonsággal 2s/√t sugarú intervallumba esik
o 99,7% biztonsággal 3s/√t sugarú intervallumba esik
És t növelésével gyökösen szűkül az intervallum

Kísérlettervezés példa:

- A várható értékre 30 megfigyelés
  - Tapasztalati átlag: 2,3 s (jó-e ez? kell még mérni?)
  - Tapasztalati szórás: $\boldsymbol{s}$ = 1,1 s
- Cél
  - 99,7%-os konfidenciaintervallum 0,6 s széles legyen
- Kísérlettervezés
  - Elvárt sugár (félszélesség) = $3\sigma = {}^{3\boldsymbol{s}}/_{\sqrt{\boldsymbol{t}}} < 0{,}3$ s
    - (ez a σ az átlag szórása, nem az eredeti mért jellemzőé!)
  - Ezért t = 121 megfigyelés kell legalább
- Hol a csalás?

Korrekció:

- Többnyire a tényleges eloszlás paraméterei *a priori* ismeretlenek (különben minek mérnénk?)
- Így nem használható fel a tényleges ***s*** szórás
- Csak a tapasztalati szórás használható → Gauss/normális helyett Student t-eloszlás
  - (más konfidenciaintervallumok)
- $t \rightarrow \infty$ esetén Student → normális
- Ökölszabály: $t>100$ esetén használható a Gauss

HIBAMODELLEZÉS

A szolgáltatásbiztonság fogalma

Motiváció: hibamentes működés
Szolgáltatási szint szerződések (SLA): ügyfél által elvárt jellemzők
Biztonságkritiukus rendszerek:

Szabvány előírások a hibák gyakoriságára
Biztonságintegritási szintek (Safety Integrity Level)

Elkerülhetetlen: Hibahatások:
Fejlesztési folyamat: tervezési hibák, implementációs hibák > verifiáció és validáció a tervezés során
Jellemzői:

- jobb minőségbiztosítás, jobb módszertanok
- növekvő bonyolultság, nehezebb ellenőrzés
- szokásos becsült értékek 1000 kódsorra
  - jó kézi fejlesztés és tesztelés: >10 hiba marad
  - automatizált fejlesztés: ~1-2 hiba marad
  - formális módszerek használata: <1 hiba marad

Működő termék: harder hibák, konfigurációs hibák, kezelői hibák >hardver hibák, konfigurációs hibák
Technológia korlátai:

- jobb paraméterek, jobb anyagok

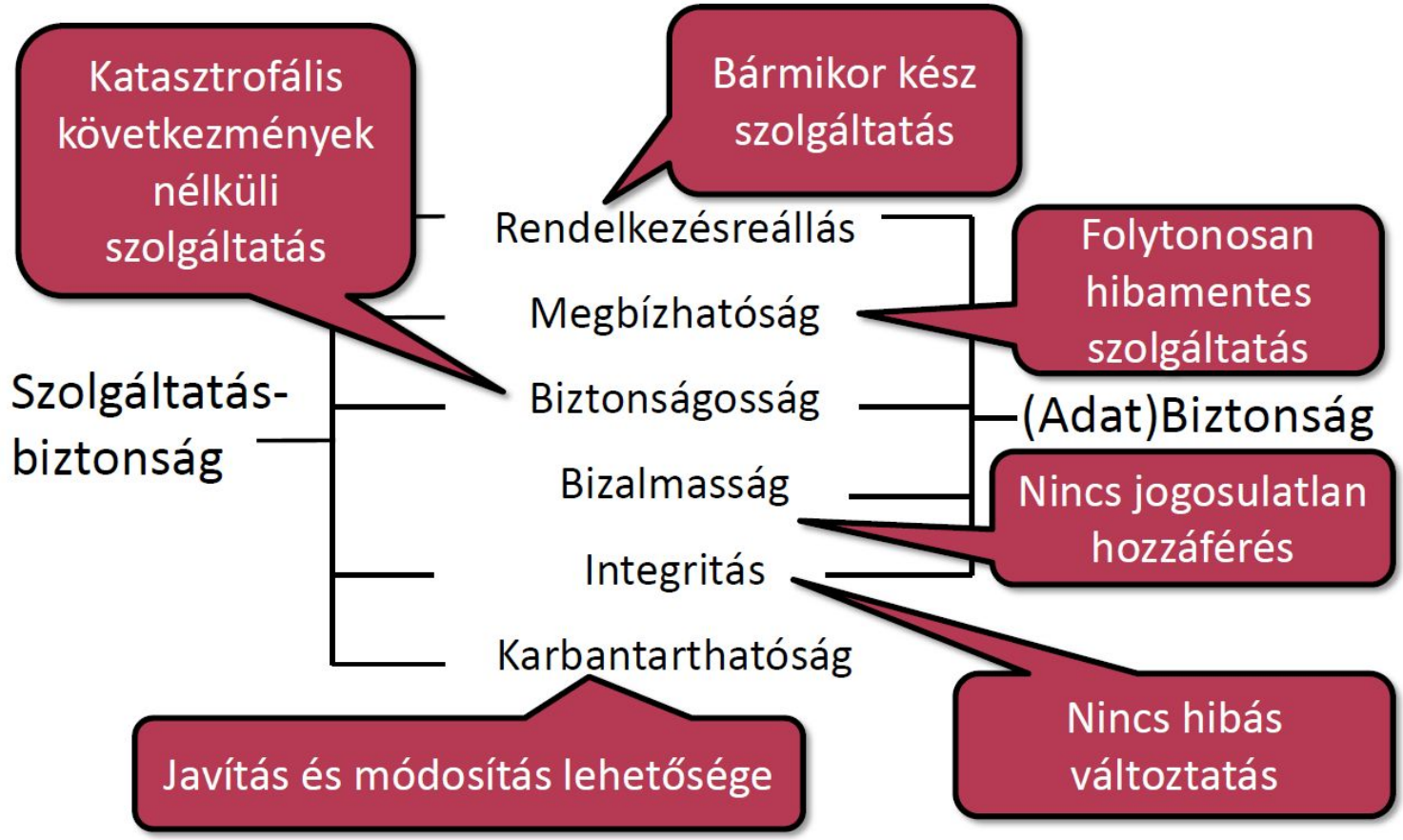
- de növekvő bonyolultság (érzékenység)

Szokásos becsült értékek:

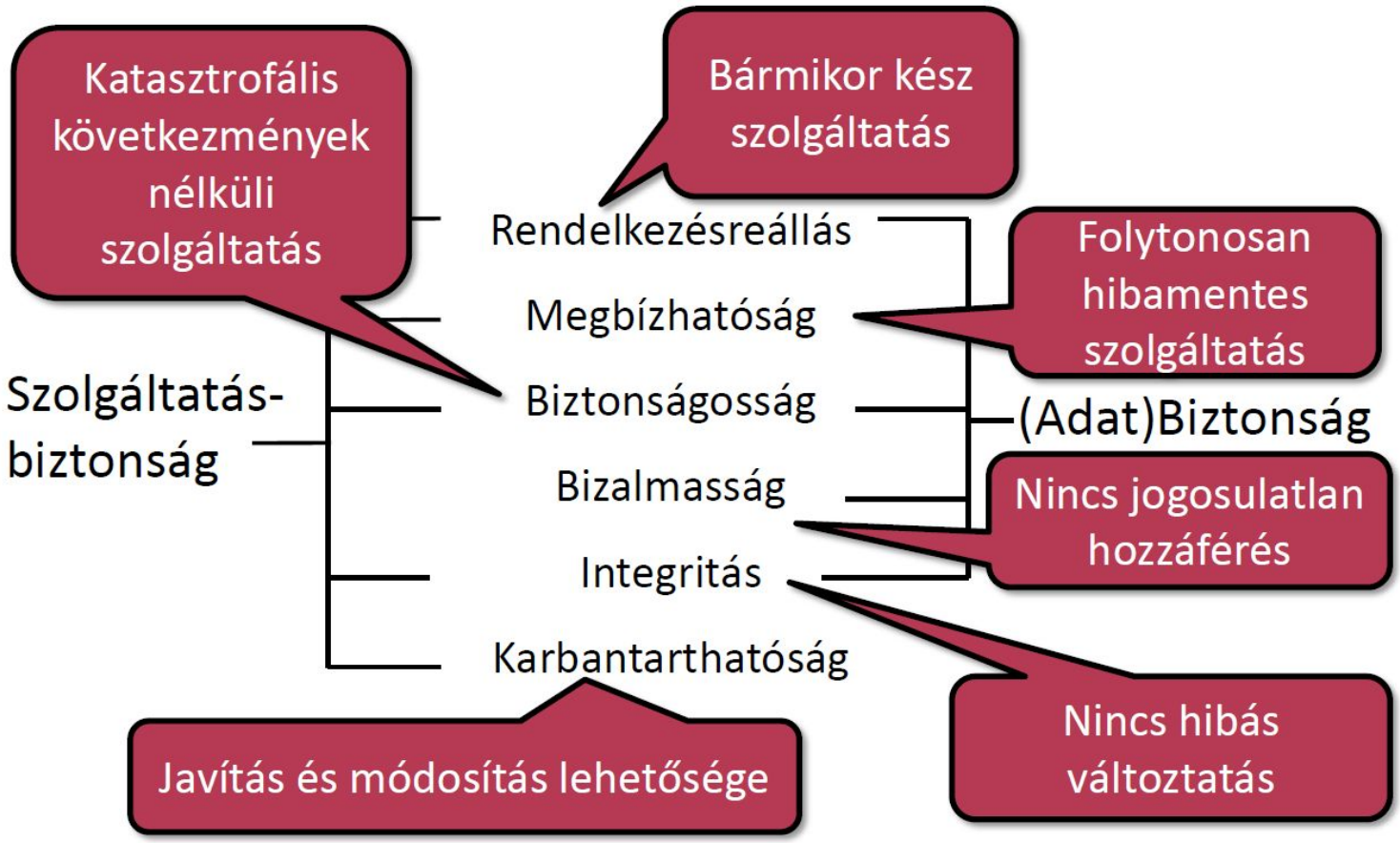
- CPU: 10-5…10-6 hiba/óra
- RAM: 10-4…10-5 hiba/óra
- LCD: ~ 2…3 év élettartam

Szolgáltatásbiztonság (dependability): nemfunkcionális követelmény
a képesség, hogy igazoltan bízni lehet a szolgáltatásban
igazoltan: elemzésen, méréseken alapul
bizalom:szolgáltatás az igényeket kielégíti

Szolgáltatásbiztonság jellemzői:



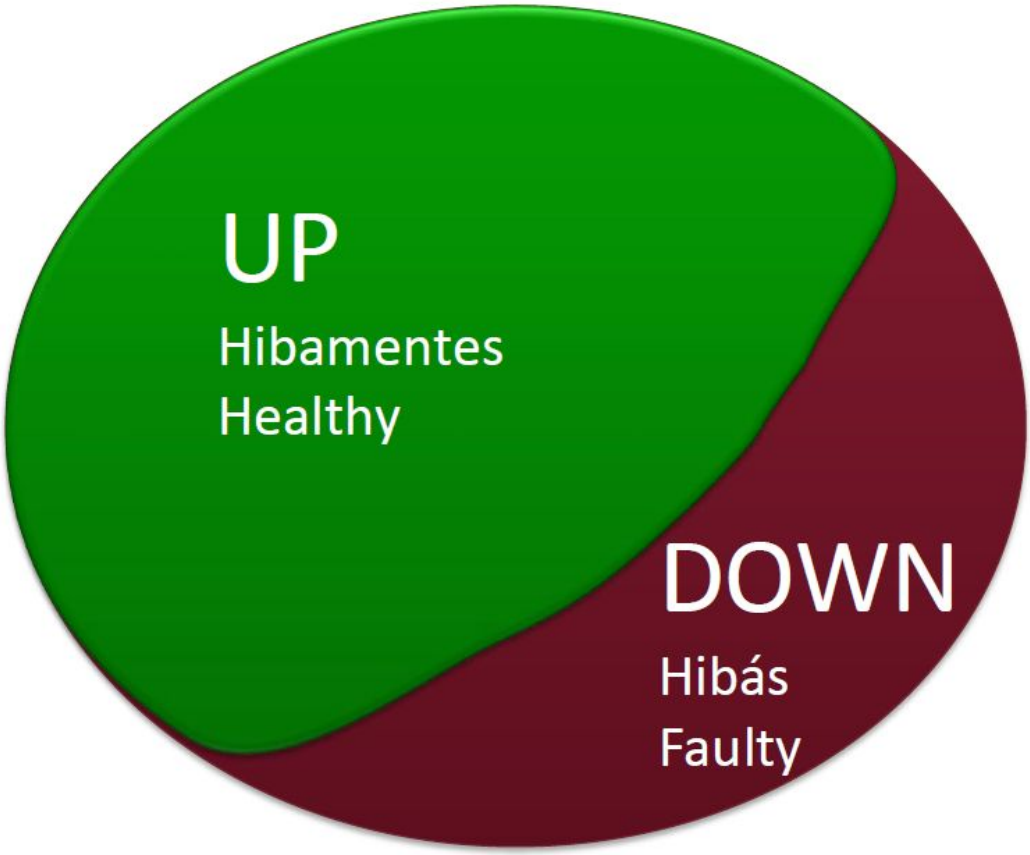
Laprie et. al.: Basic Concepts and Taxonomy of Dependable and Secure Computing



Laprie et. al.: Basic Concepts and Taxonomy of Dependable and Secure Computing

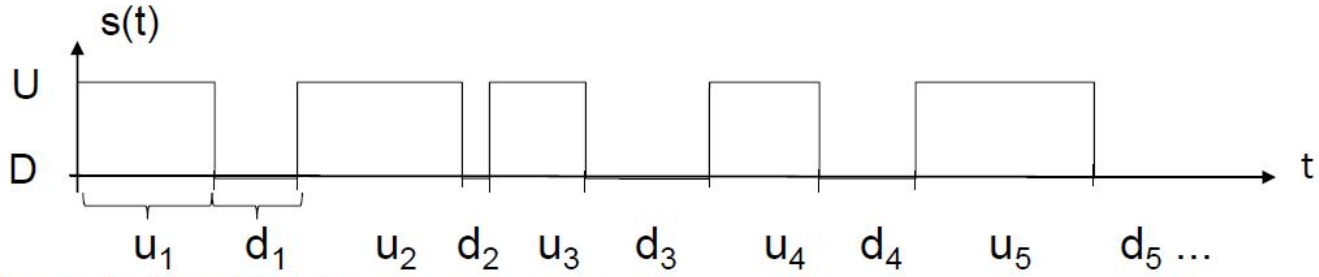
Állapotpartícionálás:

- S: a rendszer teljes állapottere



Megbízhatósági mértékek:

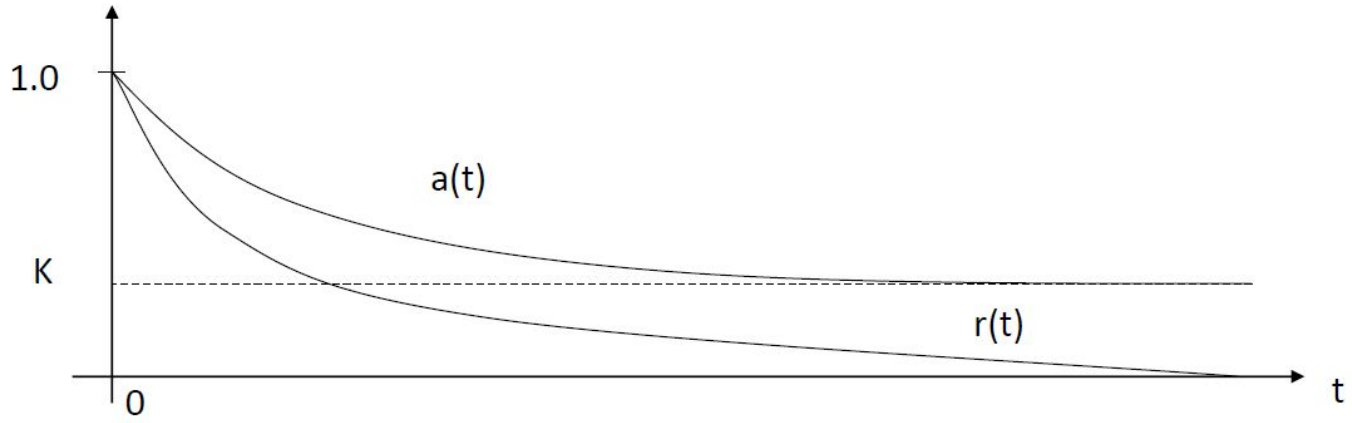
- Állapotparticionálás → s(t) rendszerállapot
  - Hibás (D) - Hibamentes (U) állapotpartíció



- Várható értékek:
  - Első hiba bekövetkezése: MTFF = $E\{u_1\}$ (mean time to first failure, néha MTTF)
  - Hibamentes működési idő: MUT = $E\{u_i\}$
  - Hibás állapot ideje: MDT = $E\{d_i\}$
  - Hibák közötti idő: MTBF = MUT + MDT (mean time between failures)

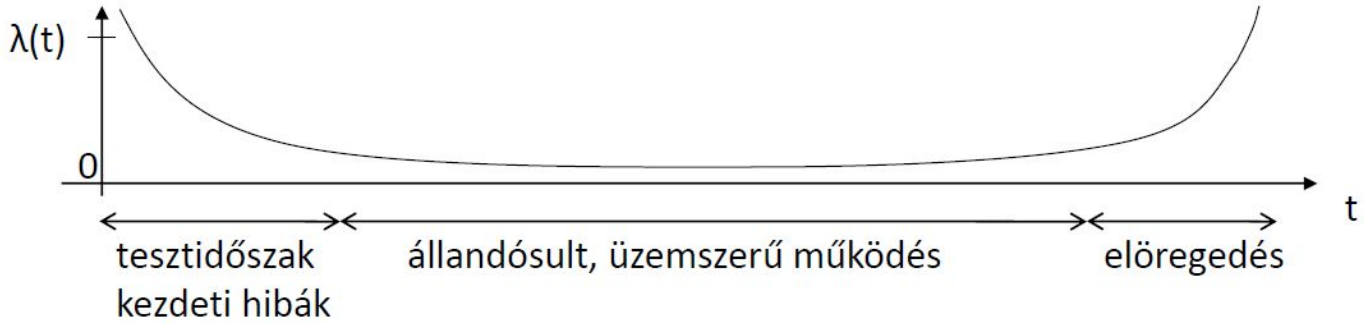
Valószínűségi időfüggvények:

- megbízhatóság: (reliability)
  $r(t) = P\{\forall\ t' < t: s(t') \in U\}$ (nem hibásodhat meg)
- rendelkezésre állás: (availability)
  $a(t) = P\{ s(t) \in U \}$ (közben meghibásodhat)
  - készenléti tényező: $K = \lim_{t\to\infty} a(t) = \frac{MUT}{MUT+MDT}$



Megbízhatóság:

- megbízhatóság: $r(t) = P\{ s(t') \in U, \forall t' < t \}$
- (első) meghibásodások gyakorisága: $-r'(t)$
  - Ez az „első meghibásodás ideje” valószínűségi változó eloszlásának sűrűségfüggvénye!
  - Várható érték: MTTF
- meghibásodási ráta: $\lambda(t) = -r'(t) / r(t)$ (meghibásodás valószínűsége időegységenként és darabonként)
  - „kádgörbe”:



- Közelítés: állandósult állapot, λ(t) = λ (konst.)
  - „Örökifjú” / „memoryless” tulajdonság
  - Jól tesztelt informatikai rendszerre igaz lehet: előbb avul el, minthogy elöregedne
- Következmény: $r(t) = e^{-\lambda t}$
- -r’(t): meghibásodás ideje exponenciális eloszlás lesz
  - λ paraméterrel
  - 1/λ várható értékkel
  - Vagyis: MTTF = 1/λ !

A szolgáltatásbiztonságot befolyásoló tényezők

Hibajelenség (failure):
A specifikációnak nem megfelelő szolgáltatás: értékbeli/időzítésbeli, katasztrofális/”jóindulatú”
Hiba(error):
Hibajelenséghez vezető rendszerállapot: lappangó->detektált
Meghibásodás (fault):
A hiba feltételezett oka

- hatás: alvó -> aktív
- fajta: véletlen vagy szándékos, időleges vagy állandósult
- eredet: fizikai/emberi, belső/külső, tervezési/működési

Szoftver hibák: Állandósult, tervezési hiba
Aktiválás a működési profil függvénye: bementi tartományok, trajektória
Megbízhatóság arányos: tesztelés után bennmaadó hibák száma
Bennmaradó hibák száma arányos: időegység alatt detektált hibák a tesztelés végén
Statisztikai tesztelés: megbízhatóság mérése
Statisztikai módszerekkel becsülhető, meddig kell a tesztelést folytatni adott megbízhatóság eléréséhez

Hatáslánc: Meghibásodás -> Hiba -> Hibajelenség
o pl. szoftver:
• meghibásodás: progr. hiba: csökkentés helyett növel
• hiba: vezérlés ráfut, változó értéke hibás lesz

• hibajelenség: számítás végeredménye rossz

o pl. hardver:

• meghibásodás: kozmikus sugárzás egy bitet átbillent
• hiba: hibás pozícióérték a memóriában
• hibajelenség: robotkar a falnak ütközik

Rendszer hierarchiaszint függvénye: alsó szintű hibajelenség felsőbb szinten meghibásodás

A hatáslánc befolyásolása:

meghibásodási tényező csökkentése
• jobb minőségű komponensek
• szigorúbb fejlesztési folyamat (ellenőrzés, tesztelés)
A meghibásodás-mentesség nem garantálható
(csökkenő chipméretek, bonyolultabb programok)
hibajelnség kialakulásának megakadályozása
• rendszerstruktúra kialakítása: redundancia

Hibatípusok:

o előre figyelembe vehető hibák: optimális kezelés a tervezési folyamat során
o előre figyelembe nem vehető hibák: megfelelő rendszerstruktúra kialakítása szükséges

Meghibásodások kategorizálása:

Hardverhibák

o alaprendszer (alaplap, processzor, memória)
o tápellátás (tápegység, szünetmentes táp)
o adattároló alrendszer
o hálózat

Emberi hibák

o rendszergazdai hibák
o illetékes felhasználók nem rosszindulatú hibái
o illetékes felhasználók rosszindulatú hibái
o illetéktelen felhasználók támadásai

Környezeti hatások

o üzemeltetési környezet rendellenességei, például a légkondicionálás leállása, bombariadó, csőtörés
o természeti katasztrófák

Szoftverhibák

o az operációs rendszer hibái
o alkalmazáshibák
o illesztőprogram-hibák

A szolgáltatásbiztonság eszközei

Hiba megelőzés: meghibásodás megakadályozása

- fizikai hibák: jó minőségű alkatrészek, árnyékolás
- tervezési hibák: verifikáció

Hiba megszüntetése: hibaállapotból kimozdítás

- prototípus fázis: tesztelés, diagnosztika, javítás
- működés közben: monitorozás, javítás

Hibatűrés: szolgáltatást nyújtani hiba esetén is

- működés közben: hibakezelés, redundancia

Hiba előrejelzés: hibák és hatásuk becslése

- mérés és "jóslás" megelőző karbantartás

Hibatűrő rendszerek:

Akármilyen jó is az ellenőrzés a tervezés során,a szolgáltatásbiztonság nem garantálható:

- o időleges hardver hibák (ld. zavarérzékenység)
- o teszteletlen szoftver hibák
- o figyelembe nem vett komplex interakciók

→ Fel kell készülni a működés közbeni hibákra!

Hibatűrés: Szolgáltatást nyújtani hiba esetén is

- o működés közbeni autonóm hibakezelés
- o beavatkozás a meghibásodás hibajelenség láncba
- o rendszertechnikai megoldások (+ megbízható alkatrész)

Alapfeltétel: Redundancia (tartalékolás)

- o többlet erőforrások a hibás komponensek kiváltására

Redundancia megjelenése:

1. Hardver redundancia
   - o többlet hardver erőforrások
     - • eleve a rendszerben lévők (elosztott rendszer)
     - • hibatűréshez betervezett (tartalék)
2. Szoftver redundancia
   - o többlet szoftver modulok
3. Információ redundancia
   - o többlet információ a hibajavítás érdekében
     - • hibajavító kódolás (ECC)
4. Idő redundancia
   - o ismételt végrehajtás, hibakezelés többlet ideje

Együttes megjelenés!

Redundancia típusai

Hidegtartalék (passzív redundancia):
- o normál üzemmódban passzív, hiba esetén aktiválva
- o lassú átkapcsolás (elindítás, állapot frissítés,...)
- o pl. tartalék számítógép

Langyos tartalék:
- o normál üzemmódban másodlagos funkciók
- o gyorsabb átkapcsolás (indítást nem kell várni)
- o pl. naplózó gép átveszi a kritikus funkciókat

Meleg tartalék (aktív redundancia):
- o normál üzemmódban aktív, ugyanazt a feladatot végzi
- o azonnal átkapcsolható
- o pl. kettőzés, többszörözés

1. Hardver redundancia

Többszörözés
- o Kettőzés
  - • hibadetektálás: pl. master-checker kialakítás
  - • hibatűrés csak diagnosztikai támogatással és átkapcsolással
- o TMR: Triple-modular redundancy
  - • hiba maszkolás szavazással
  - • szavazó kritikus elem (de egyszerű)
- o NMR: N-modular redundancy
  - • hibamaszkolás többségi szavazással
  - • MTFF kisebb, de missziós idő túlélése nagyobb esélyű
  - • repülőgép fedélzeti eszközök: 4MR, 5MR

Tipikus: nagy rendelkezésre állású fürtök

Többszörözés szintje:

Számítógép (szerver) szint: Lazán csatolt
- o nagy rendelkezésre állású fürtök
- pl. Sun Cluster, HA Linux, Windows Failover Cluster
- o szoftver támogatás: állapotszinkronizálás, tranzakciók

Kártya szint:
- o futásidőbeli átkonfigurálás, “hot-swap”
- pl. compactPCI, HDD
- o szoftver támogatás: konfigurációkezelés

Alkatrész szint: Szorosan csatolt

- alkatrész szintű többszörözés

pl. TMR, önellenőrző áramkörök

2. Szoftver redundancia

Használat:

1. Szoftver tervezési hibák esetén:
   - ismételt végrehajtás nem segít…
   - redundáns modulok: eltérő tervezés szükséges

variánsok: azonos specifikáció, de

- eltérő algoritmus, adatstruktúrák
- más fejlesztési környezet, programnyelv
- elszigetelt fejlesztés

2. Időleges (hardver) hibák esetén:
   - ismételt végrehajtás esetén a hiba nem jelentkezik
   - hibahatások kiküszöbölése a fontos

3. Információ redundancia

Hibajavító kódolás

- memóriák. háttértárak, adatátvitel
- pl. Hamming-kód, Reed-Solomon kódok

Korlátozott hibajavító képesség

- hosszú idejű adatstabilitás rossz lehet (“felgyűlnek” a hibák)
- háttértárak: “memory scrubbing” folyamatos olvasás és javítva visszaírás

Redundáns (többpéldányos) adatbázisok:

- hozzáférések konzisztenciájának biztosítása
- egypéldányos sorosíthatóság

4. Idő redundancia

Tiszta eset: utasítás újrapróbálás (retry)

- alacsony hardver szinten: processzor utasítás
- időleges hibák esetén hatásos

Idő redundancia “velejárója” a többi típusnak

Valósidejű rendszerek: tervezési szempont mennyire garantálható a hibakezelés ideje

- állandósult hardver hibák: maszkolás, meleg tartalék
- időleges hardver hibák: előrelépő helyreállítás
- szoftver tervezési hibák: N-verziós programozás

a(t), válaszidő SLA fontos tényezője

Hibák kezelése

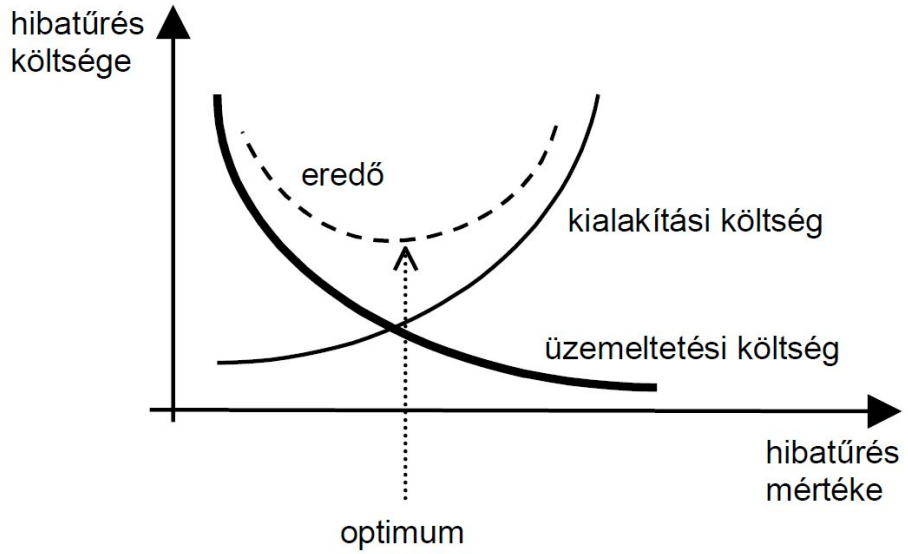
Hardver tervezési hibák (< 1%):
- nincs figyelembe véve (ld. jól tesztelt komponensek)

Hardver állandósult működési hibák (10%):
- hardver redundancia (pl. tartalék processzor)

Hardver időleges működési hibák (70-80%):
- idő redundancia (pl. utasítás újravégrehajtás)
- információ redundancia (pl. hibajavítás)
- szoftver redundancia (pl. állapotmentés és helyreállítás)

Szoftver tervezési hibák (10-20%):
- szoftver redundancia (pl. eltérő tervezésű modulok)

Költségoptimalizálás



<u>Szolgáltatásbiztonság analízise</u>

szükséges: a redundancia költséges, csak a jól tervezett redundancia éri el a célját

- mennyiség
- hideg/meleg
- javítás

feladatok: hibamódok, meghibásodások azonosítása
analízis: **kvantitatív és kvalitatív**

módszerek:
ellenőrző listák
táblázatok: FMEA: Failure Mode and Effect Analysis
hibafák
állapot alapú módszerek (pl. Petri-hálók)

1.Ellenőrző lista
Technika:
- o Tapasztalatok rendszerezett összegyűjtése
- o Alkalmazás mint „ ökölszabályok”

Biztosítja:
- o Ismert hibaforrások nem maradnak ki
- o Kipróbált módszereket alkalmaz

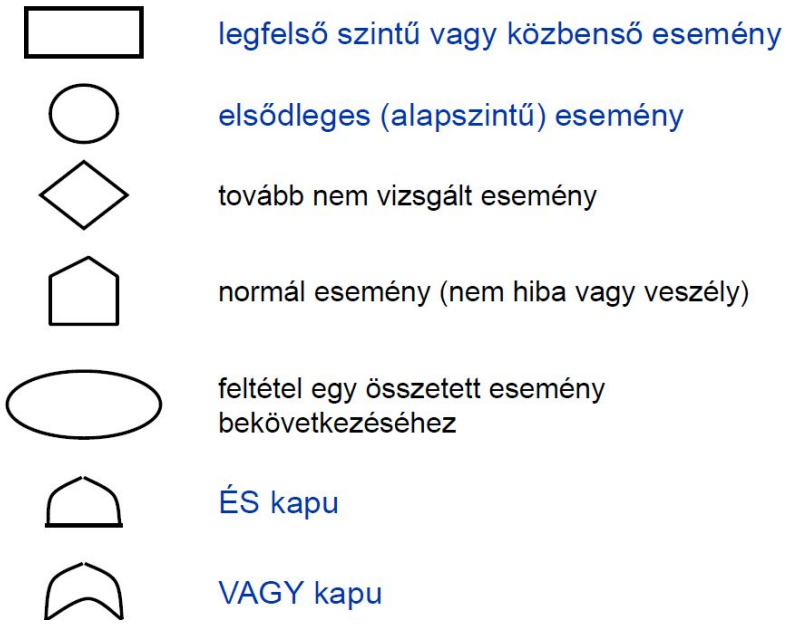
Hátrányok:
- o A lista nem teljes és nehezen kezelhető
- o Téves biztonságérzetet ad
- o Más környezetben az alkalmazhatóság kérdéses

2.Hibamód és hatás analízis (FMEA): meghibásodás és hatásaik felsorolása

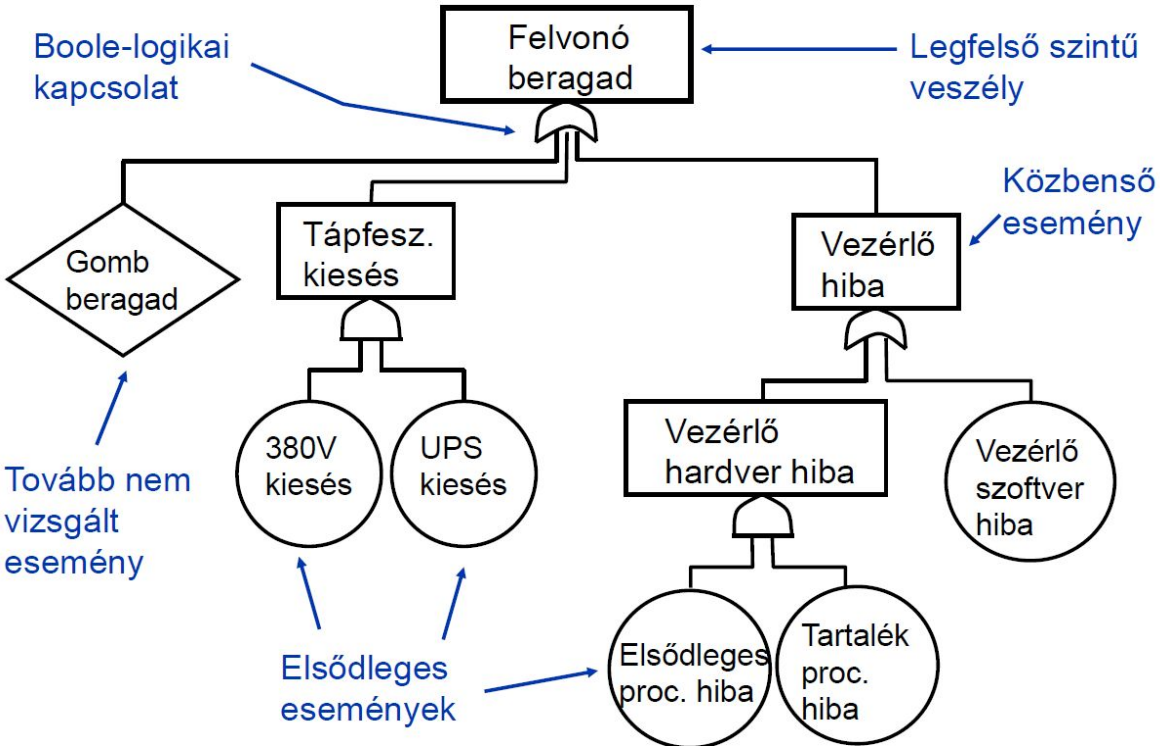
3.Hibafa- analízis
Kvalitatív:
- egyszeres hibapont (SPOF) azonosítása
- kritikus esemény: több úton is hibajelenséget okoz

Kvantitatív:
- alapszintű eseményekhez valószínűség rendelése (nehéz: honnan lesznek jó adataink?)
- gyökérelem jellemzőjének (pl. megbízhatóság)
- Problémák: nem független események...ű

Hibafa grafikus elemkészlet



Hibafa példa: felvonó



Hibafa példa: Szoftver minta

<u>Minőségi (kvalitatív) analízis</u>

Hibafa redukció: közbenső események ls pszeudo-események feloldása

→ diszjunktív normálforma (OR a legtetején)

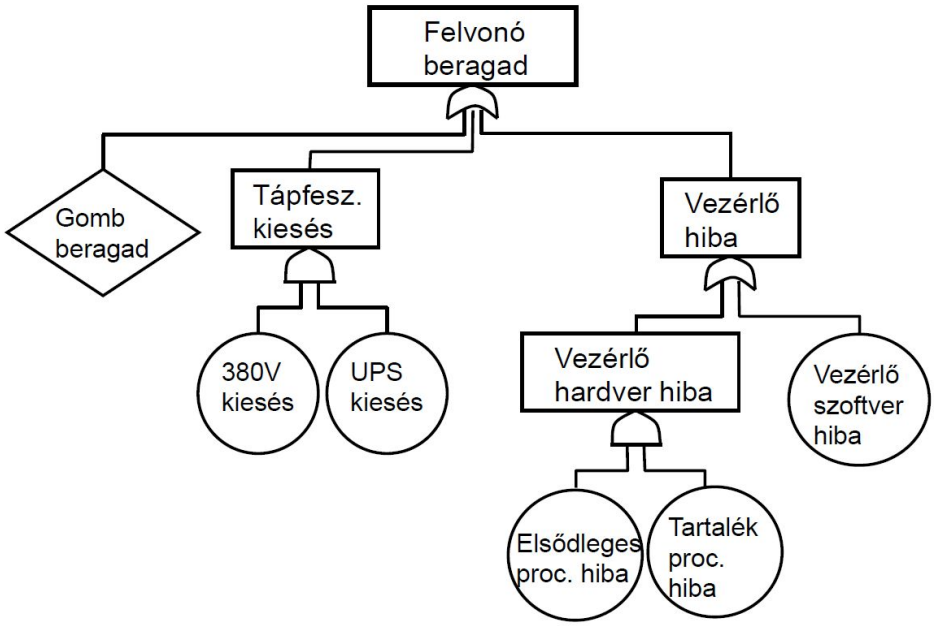
Vágat: AND kapuval összefogott elsődleges események

Minimális vágathalmaz: nem redukálható, nincs olyan aminek a részhalmaza is megtalálható

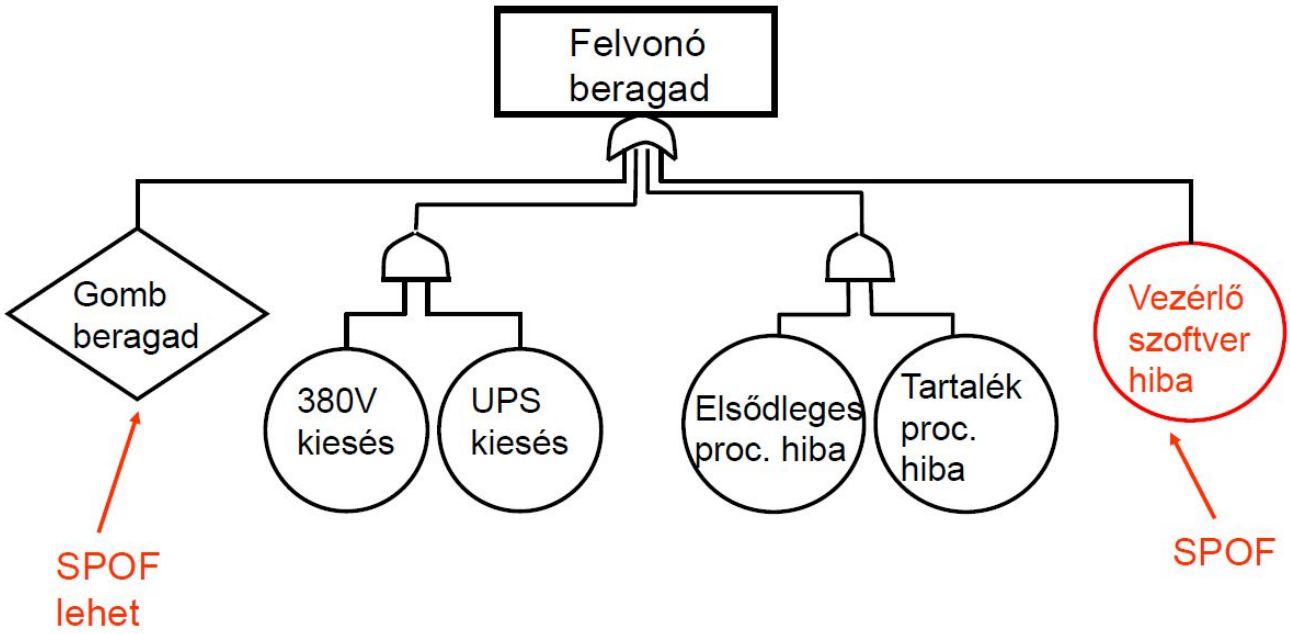
Azonosítható:

- egyszeres hibapont (SPOF)
- kritikus esemény (több vágatban is szerepel)

Hibafa példa: felvonó



Redukált hibafa példa:



Mennyiségi (kvantitatív) analízis

Alapszintű eseményekhez rendelt valószínűségek

- komponens-adat, tapasztalat, becslés

Rendszerszintű veszély valószínűség számítása

- AND kapu: szorzat (ha független események)
  pontos: $P\{A \text{ és } B\} = P\{A\}P\{B|A\}$
- OR kapu: összegzés (felső becslés)
- pontos: $P\{A \text{ vagy } B\} = P\{A\}+P\{B\}-P\{A \text{ és } B\}<=P\{A\}+P\{B\}$

Problémák:

- korreláló hibák
- időbeli (hiba) szekvenciák kezelése

Meghibásodási adatok

analízis alapja: meghibásodási valószínűségek
honnan lesznek jó adatok:

- becslés
- saját monitorozó rendszer
- külső tanulmányok, számok (hihetőség, pontosság)

Pl.: Cisco switch MTBF ~ 200 000 óra (=22,8 év)
IBM S/390 mainframe MTTF 45 év
Windows XP MTTF 608 óra
webszerver MTTF ~16 nap

4. Állapot alapú módszerek

Hibák kvalitítv leírása: diszkrét viselkedésmodell

- állapotgép, adatfolyamháló, folyamat, Petri-háló
- pl. folymatmodellben: hibakezelési ágak

Kvantitítív leírás:

- időzítés, valószínűség az állapotátmenetekhez
- determinisztikus (nincs választási pont, csak véletlen)
- valószínűségi eloszlás alapján: folytonos vagy diszkrét idejű, markovi sztochasztika

Példa: Milyen meghibásodások esetén nem lesz elérhető a szolgáltatás (webáruház)?
áramkimaradás, HW hiba, hálózati elem/kábel hiba, alaklmazás hiba, frissítés telepítése, túlterhelés, támadás, félrekonfigurálás, verzió inkompatibilitás, vírus, stb.

Hibatűrés beépítése: pl.: másodlagos DNS szerver alakalmazása, 2. ISP használata, terheléselosztó fürt, hálózati utak duplikálása, melegtartalék SQL szerver

így is marad sok kiesési lehetőség pl.: adatok törlése, teljes szerverterem pusztulása, adminisztrátori hibák, OS hotfix miatti újraindítás

Mindig tudni kell, hogy mi ellen védekezünk, milyen módszerek vannak rá, és megéri-e védekezni.

Hibafa rajzolás: SHARPE eszköz
Petri-háló: TimeNet eszköz

Hiba hatásának becslése
Alapelv:

- hibaokok/hibamódok (fault) erőforrráshoz rendelése
- hiba útjának követése (ERROR)
- érinti-e a szolgáltatást(FAILURE)

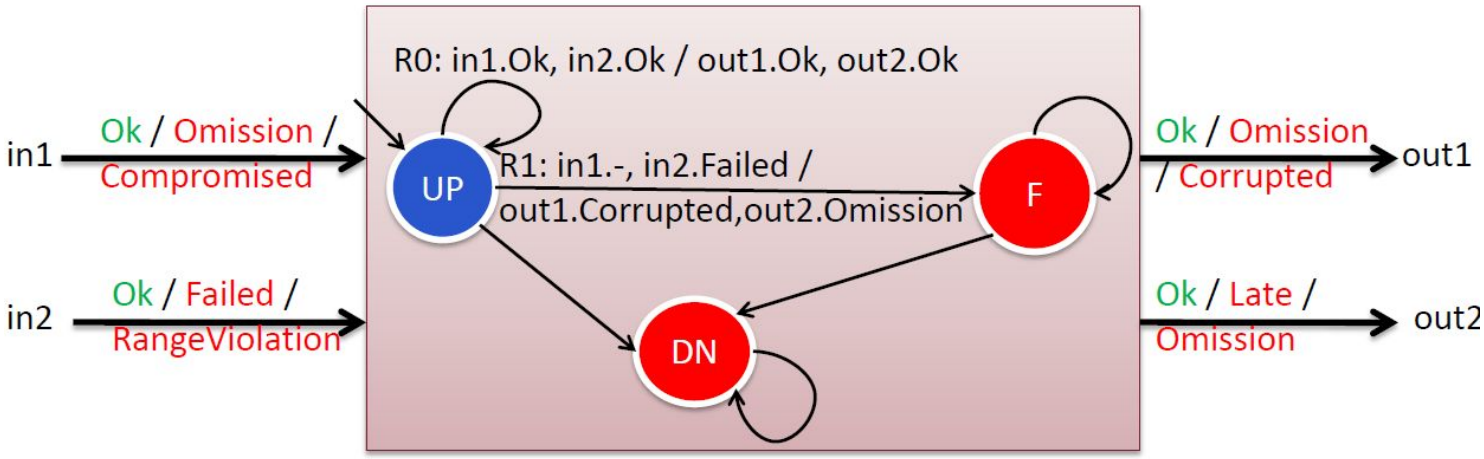
Hibaterjedés modellezése
tokenek: jó/ hibás / gyanús - tokenek színezése

- egy potenciálisan hibás komponens utáni lépések utáni lépések legalább gyanúsak
- kvalitatív közelítés
- statikus: kárbehatárolási tartomány (Damage Confinement Region)
- Dinamikus ellenőrzés?
    - jobb közelítés
    - nehezebb meghatározni

Hibamodellezés adafolyamhálóval
Kvalitítv hibamodell → adatfolyamháló

- komponens → adatfolyam csomópont
- belső hibamódok → csomópont állapotai

- komponens kapcsolatok → csatornák
- kommunikációs hibamódok → csatorna tokenek



MODELLELLENŐRZÉS, PROGRAMVERIFIKÁCIÓ

Modellellenőrzés != modellek ellenőrzése

Modellellenőrzés:
Adott egy (nmdeterminisztikus) viselkedés modell és formális követelmények
A modell kielégíti-e a követelményeket? → a teljes állapottér felderítésével eldönthető

<u>Motiváció</u>
- biztonságkritikus beágíazott rendszerek: pl.: repülőgép/autó
- modellalapú fejlesztés és kódgenerálás: pl:: SCADE - adatfolyam alapú fejlesztőeszköz
- protokollhibák kresése
- programverifikáció

DE:
- nehéz, lassú folyamat
- magas képzettséget igényel

Ellenőrzés:
- naiv-módszer: állapottér-bejárás
- minden elérhető állapotra igazak a kritériumok
- ha igen: *siker*
- ha nem: *ellenpélda*
    - rossz állapot az odavezető úttal együtt
    - segít megtalálni a hibát
    - elhárítása után új ellenőrzés

Problémák:

Az állapottér túl nagy lehet!
Mit kezdünk a nem állapotra felírt kritériumokkal?

→ MC (Model ckecker)

Kritériumok leírása:
Egy gyakori formalizmus: *temporális logikák*
Állapotokra vonatkozó állítások
Logikai operátorok
Útvonalkvantorok

- ∀ (néha A) - minden lehetséges esetben
- ∃ (néha E) - legalább egyféle lefutásban

Temporális operátorok

- □ (néha G) – végig, állandóan
- ◊ (néha F) - legalább egyszer

Programverifikáció

Dinamikus programverifikáció
szúrópróbaszerű analízis: szoftvertesztelés
kimerítő dinamikus analízis

- hagyományos ellenőrzés:modell absztrahálása

JPF: Java Pathfinder: Java VM, bájtkódot futtat, teljes állapottár bejárásra képes

Statikus programverifikáció
“futtatás” állapotbejárás nélküli ellenőrzés

- típusellenprzés, null, stb: legtöbb modenr nyelvben

“design by contract”: programkód annotálása állításokkal (Hoare logika)

- Prekondíció: aminek a művelet előtt teljesülnie kell
- Posztkondíció: ami a művelet után garantált
- Invariáns: állandóan teljesülnie kell
- Cilkusvariáns: indulciós feltevés

Nyelvek:

- eiffel

- ADA → SPARK
- Java+JML (java Modelling Language)
- .NET Code Contracts, C# → Spec#
- Verifikáció a megfelelő eszközökkel
    - pl. ESC/Java2 a JML-hez

Proof-carrying code: viszi magával a bizonyítást

Microsoft Research: **Singularity OS** koncepció

- Spec# → Sing# kommunikációs csatorna protokollokkal
- telepítéskor statikusan verifikált programok
- minden program mehet egyetlen memóriatérbe
- context switch olcsóbb lesz, teljesítménynövekedés