

A formális módszerek szerepe

dr. Majzik István

dr. Bartha Tamás

dr. Pataricza András

BME Méréstechnika és Információs Rendszerek Tanszék

Mik azok a formális módszerek?

Formális módszerek

- Matematikai technikák

- elsősorban diszkrét matematika
- és matematikai logika

használata arra, hogy elkészítsük és ellenőrizzük hardver és szoftver rendszerek

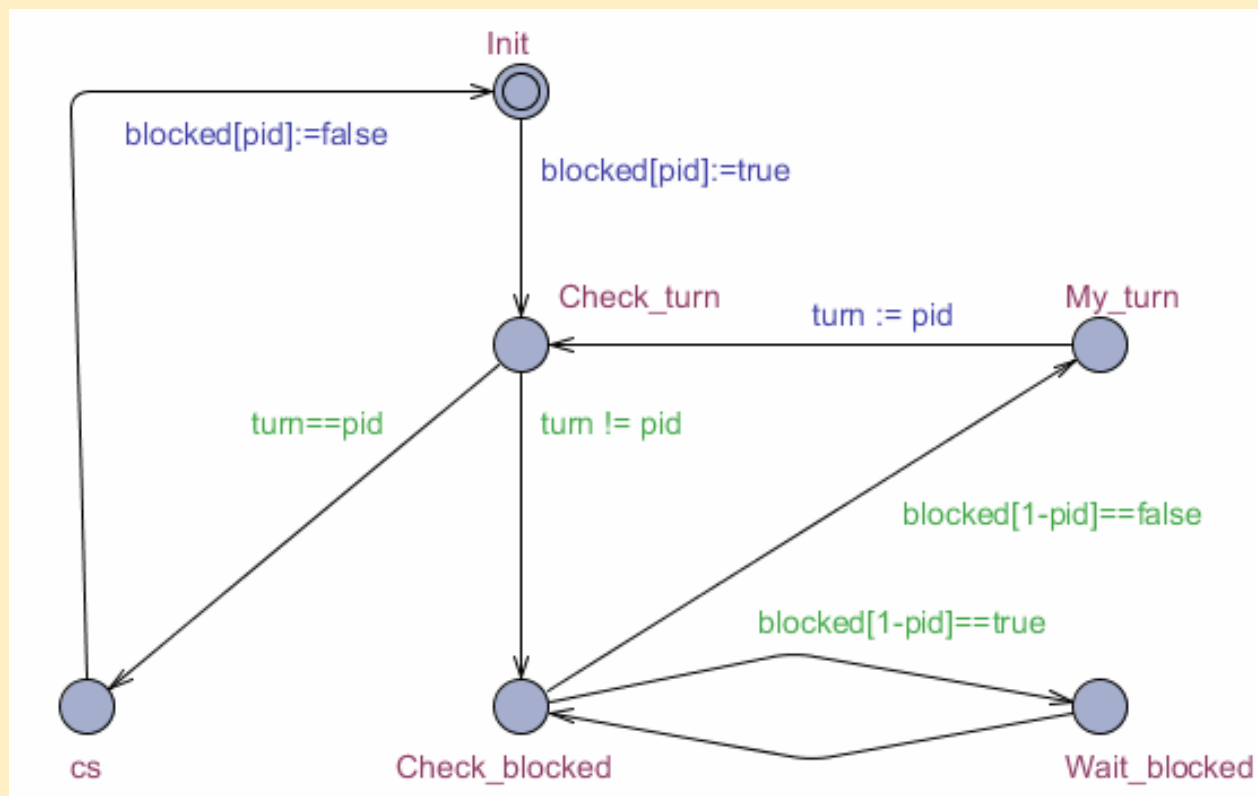
- specifikációját,
- terveit (modelljeit),
- implementációját (viselkedését),
- dokumentációját.

Első lépés: Formalizmus

- Cél: **Specifikációs** illetve **modellezési** (tervezési) nyelv
 - Az elvárt tulajdonságok illetve a modellek leírására
 - Matematikai precizitással
- Formális nyelvek felépítése
 - **Formális szintakszis**
 - Jelölésmód: milyen nyelvi elemek és kapcsolatok vannak
 - **Formális szemantika**
 - A jelölésmód interpretációja: mit értek alatta
- Mit szeretnénk leírni formális nyelvekkel?
 - Funkcionalitás (viselkedés, feltételek, elvárások, ...)
 - Struktúra, interfészek
 - Extra-funkcionális aspektusok is: Teljesítmény, megbízhatóság, ...
- A formális nyelv használatának előnyei
 - Egyértelműség, ellenőrizhetőség
 - Automatikus feldolgozhatóság

Egy egyszerű példa

- Automata formalizmus:
 - Állapotok és állapotátmenetek
 - Változók
 - Változókon kiértékelhető feltételek az átmenetek végrehajtásához
 - Értékadás akciók az átmenetek végrehajtása során



Formális szintakszis (áttekintés)

- Matematikai eszközök:

$KS = (S, R, L)$ és AP , ahol

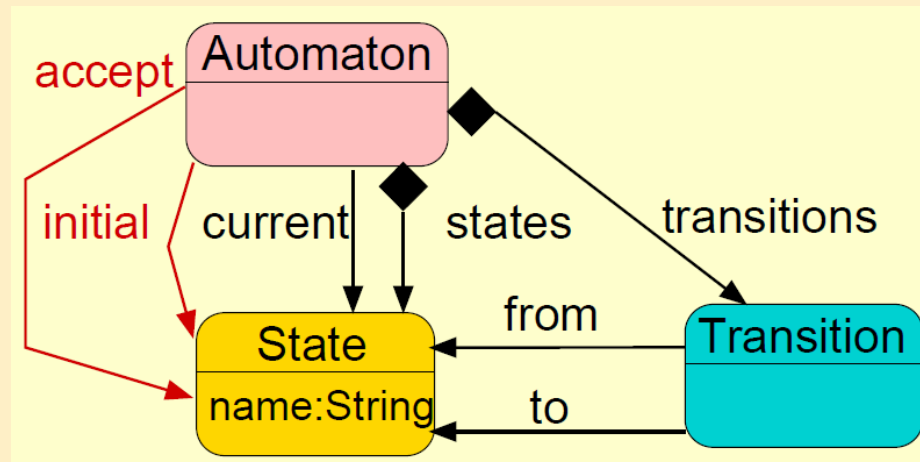
$AP = \{P, Q, R, \dots\}$

$S = \{s_1, s_2, s_3, \dots, s_n\}$

$R \subseteq S \times S$

$L: S \rightarrow 2^{AP}$

- BNF: $BL ::= \text{true} \mid \text{false} \mid p \wedge q \mid p \vee q$
- Metamodell:



- Absztrakt szintakszis (nyelvtani szabályok)
- Konkrét szintakszis (megjelenítés)

Formális szemantika (áttekintés)

- **Műveleti (operációs) szemantika:** „Programozóknak”
 - Megadja, mi történik a számítások során
 - Egyszerű elemekre épít: pl. állapotok, akciók
- **Axiomatikus szemantika:** „Helyességbizonyításhoz”
 - Állítás nyelv + axiómakészlet + következtetési szabályok
 - Elő- és utófeltételek adják meg a jelentést
 - Pl. automatikus tételbizonyító rendszerekhez
- **Denotációs szemantika:** „Fordítóprogramokhoz”
 - Szintaxis által meghatározott „jel -> jelölt dolog” leképezés
 - A „jelölt dolog” tipikusan mint matematikai domén adott
 - Számítási szekvencia, vezérlési gráf, állapothalmaz, ...
és ezeken definiált műveletek (összefűzés, unió, ...)
 - A modellek vizsgálata a mögöttes matematikai objektum vizsgálatára vezethető le

Továbbblépés: Formális módszerek

Formális módszer:

A formális modelről ismeretet adó matematikai eljárás
Eszközökkel támogatható

- A formális modell **végrehajtása**
 - Szimuláció
- A formális modell **ellenőrzése**: Formális verifikáció
 - „Önmagában való” vizsgálat
 - Konzisztencia, ellentmondás-mentesség
 - Teljesség, zártság
 - Megfelelés vizsgálata
 - Modellek között
 - Modellek és elvárt tulajdonságok között (implementáció $\leftrightarrow$ specifikáció)
- A formális modell alapján történő **szintézis**:
 - Szoftver (programkód, konfiguráció) generálása
 - Hardver tervek generálása

A formális módszerek használatának lépései

- Valós probléma formális vizsgálata:
 1. Fogalmi tér felépítése <- feltételezések
 2. Probléma formalizálása <- absztrakció
 3. Formális modell analízise <- automatikus lehet
 4. Eredmények értelmezése, felhasználása
- Szükséges feltételek:
 - Feltételezések teljesülésének ellenőrzése
 - Modell validálása
 - Eszközök helyességének belátása

Mire szeretnénk használni a formális
módszereket?

Milyen problémákkal nézünk szembe?
Miben segíthetnek a formális módszerek?

Egy tanulságos történet...

- Vasa svéd hadihajó, 1628:
Elsüllyedt közvetlenül
a vízrebocsátás után
- Problémák:
 - Változó követelmények
(II. Gusztáv Adolf király)
 - Hiányzó pontos specifikáció
(Henrik Hybertsson építő)
 - Ellenőrizetlen tervek
(Johan Isbrandsson alvállalkozó)
 - Figyelmeztetések figyelmen kívül hagyása
(Fleming admirális)
- Dokumentáció:
 - The Vasa: A Disaster Story with Software Analogies. By Linda Rising. The Software Practitioner, January-February 2001.
 - Why the Vasa Sank: 10 Problems and Some Antidotes for Software Projects. By Richard E. Fairley and Mary Jane Willshire. IEEE Software, March-April 2003.



Komplex rendszerek tervezése



- Minimális tervezés
- Implicit folyamat
- Egyszerű eszközök



- Tervezés
- Definiált folyamat
- Hatékony eszközök



- Ellenőrzött tervek
- Meghatározott folyamat
- Automatikus eszközök

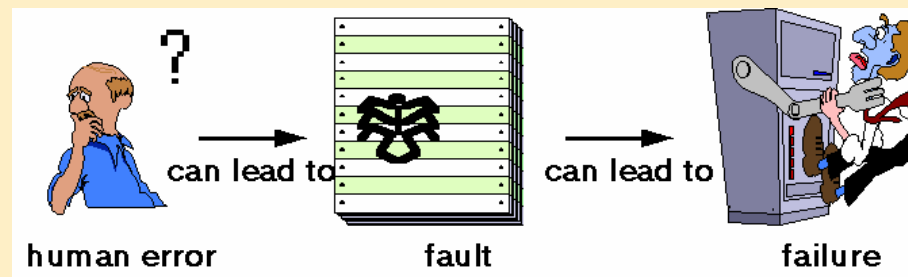
Szoftver minőségi krízis

- Tipikus kódméret:
 - 10 kLOC ... 1000 kLOC
- Fejlesztési idő:
 - 0,1 - 0,5 mérnökév / kLOC (nagy méretű szoftver)
 - 5-10 mérnökév / kLOC (kritikus szoftver)
- Hiba eltávolítás (ellenőrzés, tesztelés, javítás):
 - 45 - 75% ráfordítás
- Hibasűrűség változása:
 - 10 - 200 hiba / kLOC jön létre a fejlesztés során



Ellenőrzés, debuggolás, javítás

- 0,01 - 10 hiba / kLOC maradhat az üzembe helyezésig



Egy „földhözragadt” példa: Kölcsönös kizárás

- 2 résztvevőre, 3 megosztott változóval (H. Hyman, 1966)
 - **blocked0**: Első résztvevő (**P0**) be akar lépni
 - **blocked1**: Második résztvevő (**P1**) be akar lépni
 - **turn**: Ki következik belépni (0 esetén P0, 1 esetén P1)

```
while (true) {  
    blocked0 = true;  
    while (turn != 0) {  
        while (blocked1 == true) {  
            skip;  
        }  
        turn = 0;  
    }  
    // Critical section  
    blocked0 = false;  
    // Do other things  
}
```

P0

```
while (true) {  
    blocked1 = true;  
    while (turn != 1) {  
        while (blocked0 == true) {  
            skip;  
        }  
        turn = 1;  
    }  
    // Critical section  
    blocked1 = false;  
    // Do other things  
}
```

P1

Helyes ez az algoritmus?

IT alkalmazások fejlesztésének kihívásai

- Jó minőségű specifikáció és tervek készítése
 - Teljes
 - Ellentmondás-mentes, egyértelmű
 - Ellenőrizhető
- Tervek ellenőrzése
 - Tervezői döntések igazolása
 - Bizonyítottan helyes tervek a továbblépés alapjai
 - Hibák elkerülése vagy korai felderítése
 - Minőség ↔ költség ↔ fejlesztési idő optimalizálás
- Bizonyított helyességű tervezői eszközök használata
 - Forráskód, konfiguráció, teszt és monitor szintézis

Ezek alapjait a formális módszerek adhatják!

Klasszikus példa: Cleanroom Software Engineering

- Eredete:
 - IBM javaslat (80-as évek),
 - US katonai fejlesztések (90-es évek)
- Célkitűzés:
 - Hibaelkerülés a hibaeltávolítás helyett
 - Formális modelleken alapuló ellenőrzés
- Alapelvek:
 - Formális modellek használata és ellenőrzése
 - Inkrementális megvalósítás minőségellenőrzéssel (komplexitás fokozatos növelése)
 - Statisztikai alapú tesztelés a formális modellek alapján
 - Reprezentatív trajektóriák kiválasztása
 - A modellezés kézi validációja



Biztonságkritikus szoftverek fejlesztése

- IEC 61508:
 - Functional safety in electrical / electronic / programmable electronic safety-related systems
 - Szakterület-specifikus szabványok alapja
- Követelmény-specifikáció készítésének előírásai:

Table A.1 – Software safety requirements specification (see 7.2)

Technique/Measure*		Ref.	SIL1	SIL2	SIL3	SIL4
1	Computer-aided specification tools	B.2.4	R	R	HR	HR
2a	Semi-formal methods	Table B.7	R	R	HR	HR
2b	Formal methods including for example, CCS, CSP, HOL, LOTOS, OBJ, temporal logic, VDM and Z	C.2.4	---	R	R	HR
NOTE 1 – The software safety requirements specification will always require a description of the problem in natural language and any necessary mathematical notation that reflects the application.						
NOTE 2 – The table reflects additional requirements for specifying the software safety requirements clearly and precisely.						
* Appropriate techniques/measures shall be selected according to the safety integrity level. Alternate or equivalent techniques/measures are indicated by a letter following the number. Only one of the alternate or equivalent techniques/measures has to be satisfied.						

Biztonságkritikus szoftverek fejlesztése

- IEC 61508:
Szoftver
tervezés
és
fejlesztés
előírásai

Table A.2 – Software design and development:
software architecture design (see 7.4.3)

Technique/Measure*		Ref	SIL1	SIL2	SIL3	SIL4
1	Fault detection and diagnosis	C.3.1	---	R	HR	HR
2	Error detecting and correcting codes	C.3.2	R	R	R	HR
3a	Failure assertion programming	C.3.3	R	R	R	HR
3b	Safety bag techniques	C.3.4	---	R	R	R
3c	Diverse programming	C.3.5	R	R	R	HR
3d	Recovery block	C.3.6	R	R	R	R
3e	Backward recovery	C.3.7	R	R	R	R
3f	Forward recovery	C.3.8	R	R	R	R
3g	Re-try fault recovery mechanisms	C.3.9	R	R	R	HR
3h	Memorising executed cases	C.3.10	---	R	R	HR
4	Graceful degradation	C.3.11	R	R	HR	HR
5	Artificial intelligence - fault correction	C.3.12	---	NR	NR	NR
6	Dynamic reconfiguration	C.3.13	---	NR	NR	NR
7a	Structured methods including for example, JSD, MASCOT, SADT and Yourdon.	C.2.1	HR	HR	HR	HR
7b	Semi-formal methods	Table B.7	R	R	HR	HR
7c	Formal methods including for example, CCS, CSP, HOL, LOTOS, OBJ, temporal logic, VDM and Z	C.2.4	---	R	R	HR
8	Computer-aided specification tools	B.2.4	R	R	HR	HR

NOTE – The measures in this table concerning fault tolerance (control of failures) should be considered with the requirements for architecture and control of failures for the hardware of the programmable electronics in IEC 61508-2.

* Appropriate techniques/measures shall be selected according to the safety integrity level. Alternate or equivalent techniques/measures are indicated by a letter following the number. Only one of the alternate or equivalent techniques/measures has to be satisfied.

A formális módszerek használatának várt eredményei

- Csökken a rendszerben maradó koncepcionális és tervezői hibák száma, javul a minőség
 - Szolgáltatásbiztonság növelhető
 - De garanciát nem ad a használhatóságra, a felhasználói elvárások teljesítésére
 - Validációt nem helyettesíti
- Tervezők (nem csak matematikusok) által is használható, amennyiben a specifikus tudást **eszközökbe** integrálják
 - Modellező eszközök
 - Modell ellenőrzők
 - Automatikus tételbizonyítók, ekvivalencia ellenőrzők
 - Szintézis eszközök
 - Kódgenerátor, konfiguráció generátor az ellenőrzött modellek alapján
 - Teszt generátor (validáláshoz)

Formális módszerek lehetséges szerepe a szoftver fejlesztési életciklus modellekben

A fejlesztés életciklus modelljei

- Miért van szükség életciklus modellre?
 - Komplexitás kezelése
 - Fázisokra osztás
 - Elosztott fejlesztés, integráció
 - Ellenőrzések ütemezése
 - Mérföldkövek
 - Ellenőrizhetőség, számonkérhetőség
 - Tervezhetőség (pénz, idő)
 - Visszalépés lehetősége
 - Változások kezelése
 - Követelmények módosulása, hibajavítás
 - Új megoldások és technológiák bevezetése

Fejlesztési tevékenységek

Követelményanalízis

Mi a megoldandó probléma?

Probléma

Konceptcionális tervezés

Milyen megoldási módszerek/
eszközök léteznek?

Rendszertervezés

Hogyan oldható meg a
feladat?

Implementáció

Implementáció

Hogyan valósítható meg
a feladat megoldása?

Tesztelés

Megoldottuk a problémát?

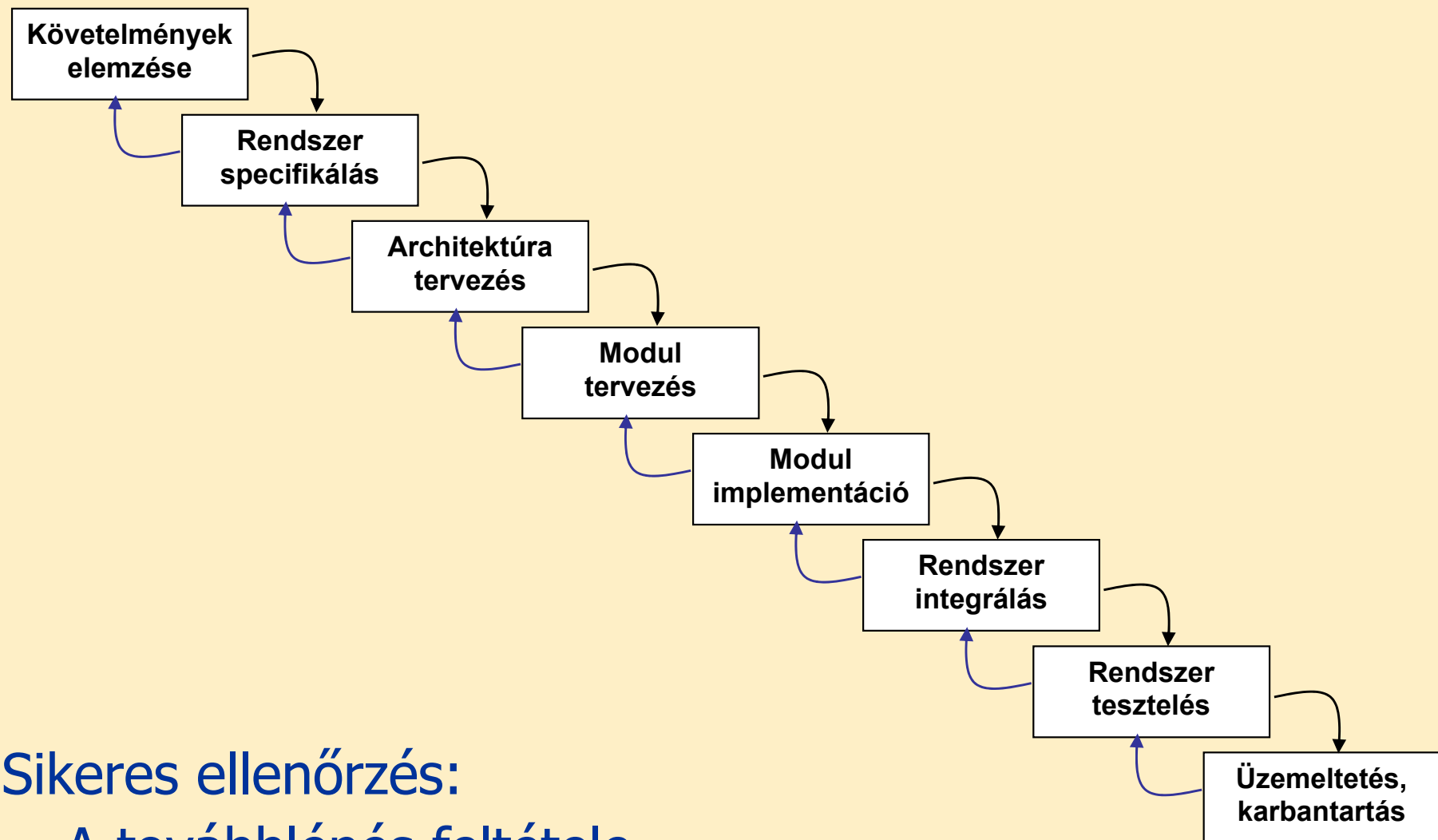
Üzembehelyezés

A megrendelő megfelelőnek tartja
a kész rendszert?

Üzemeltetés, karbantartás

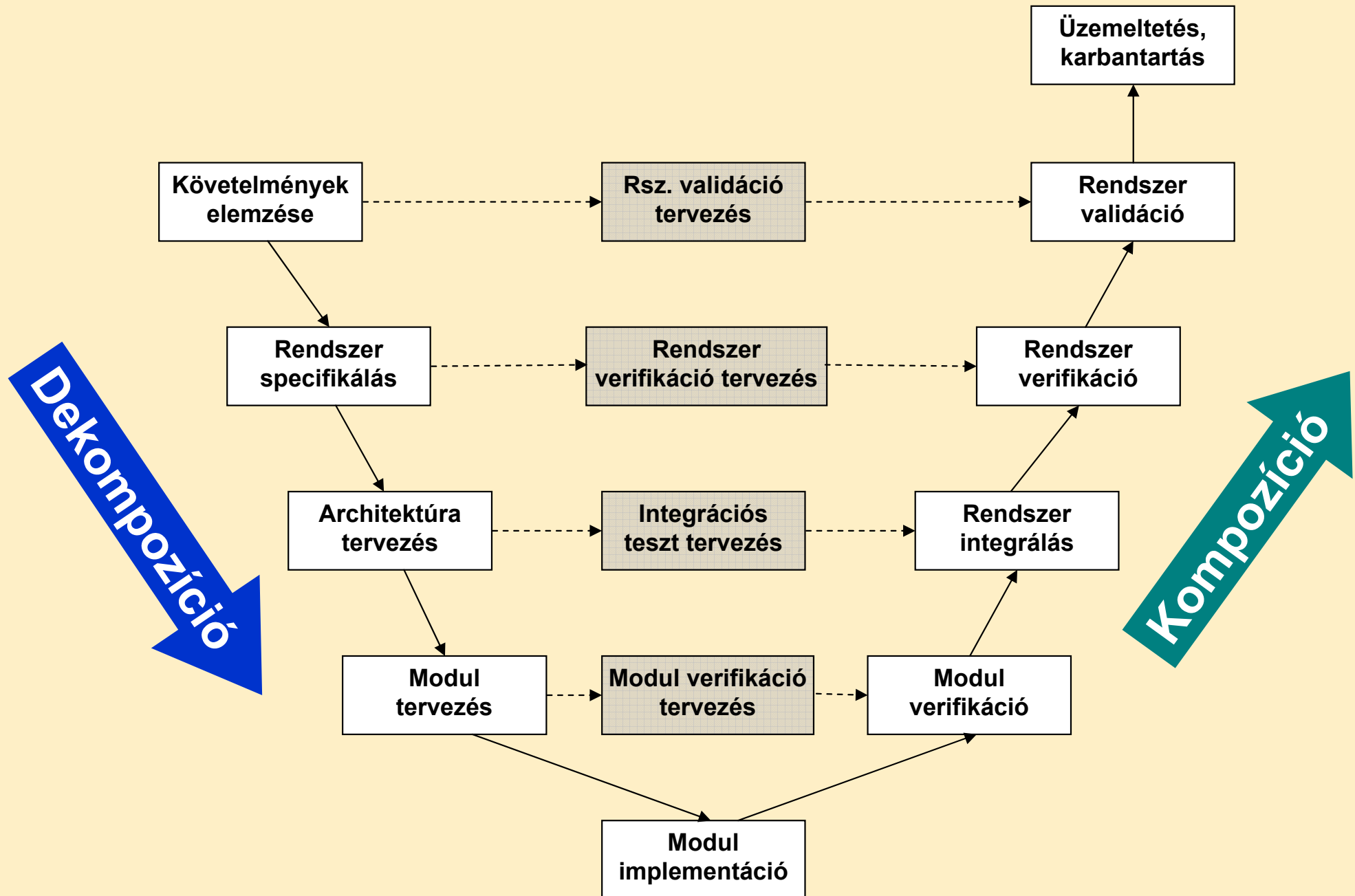
Továbbfejlesztés szükséges?

Vízesés modell (visszalépéssel)



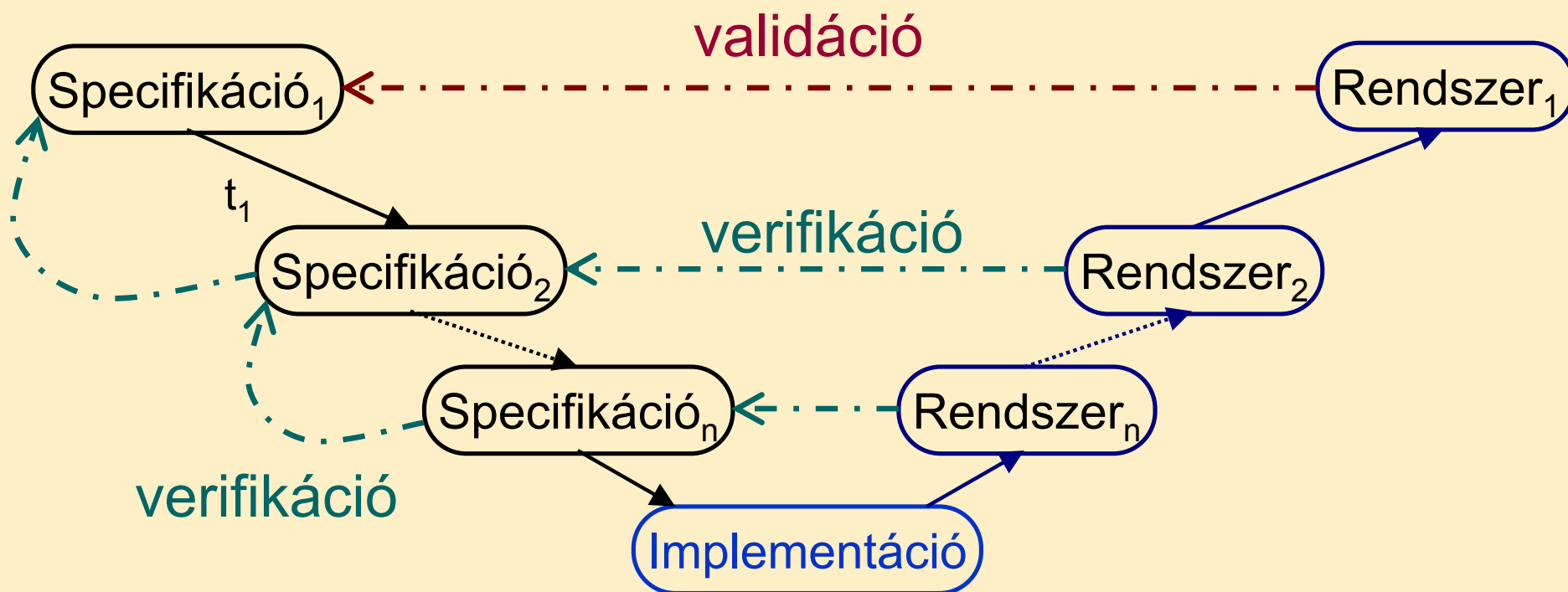
Sikeres ellenőrzés:
A továbblépés feltétele

V-modell



Verifikáció és validáció szerepe

- **Verifikáció** (igazolás):
 - Jól építjük-e a rendszert?
- **Validáció** (érvényesítés):
 - Jó rendszert építünk-e?



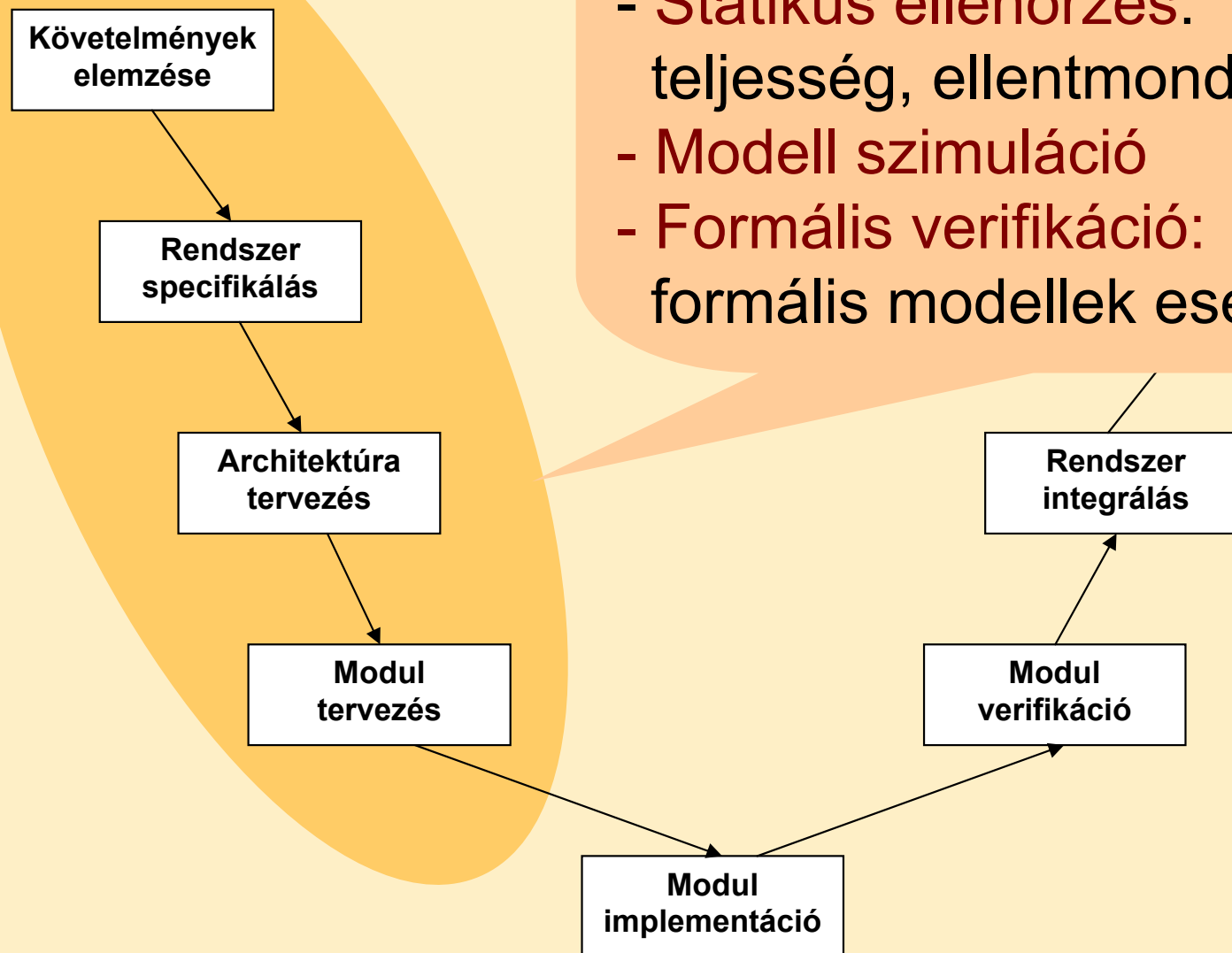
Verifikáció és validáció összehasonlítása

Verifikáció (igazolás)	Validáció (érvényesítés)
„Jól építjük-e a rendszert?”	„Jó rendszert építettünk-e?”
Összhang ellenőrzése a fejlesztési fázisokban, illetve ezek között	A fejlesztés eredményének ellenőrzése
Fejlesztési lépések során használt tervek (modellek) és specifikációjuk közötti megfelelés ellenőrzése	A kész rendszer és a felhasználói elvárások közötti megfelelés ellenőrzése
Objektív folyamat; formalizálható, automatizálható	Szubjektív elvárások lehetnek; elfogadhatósági ellenőrzés
Felderíthető hibák: Tervezési, implementációs hibák	Felderíthető hibák: Követelmények hiányosságai is
Nincs rá szükség, ha automatikus a leképzés követelmény és implementáció között	Nincs rá szükség, ha a specifikáció tökéletes (elég egyszerű)

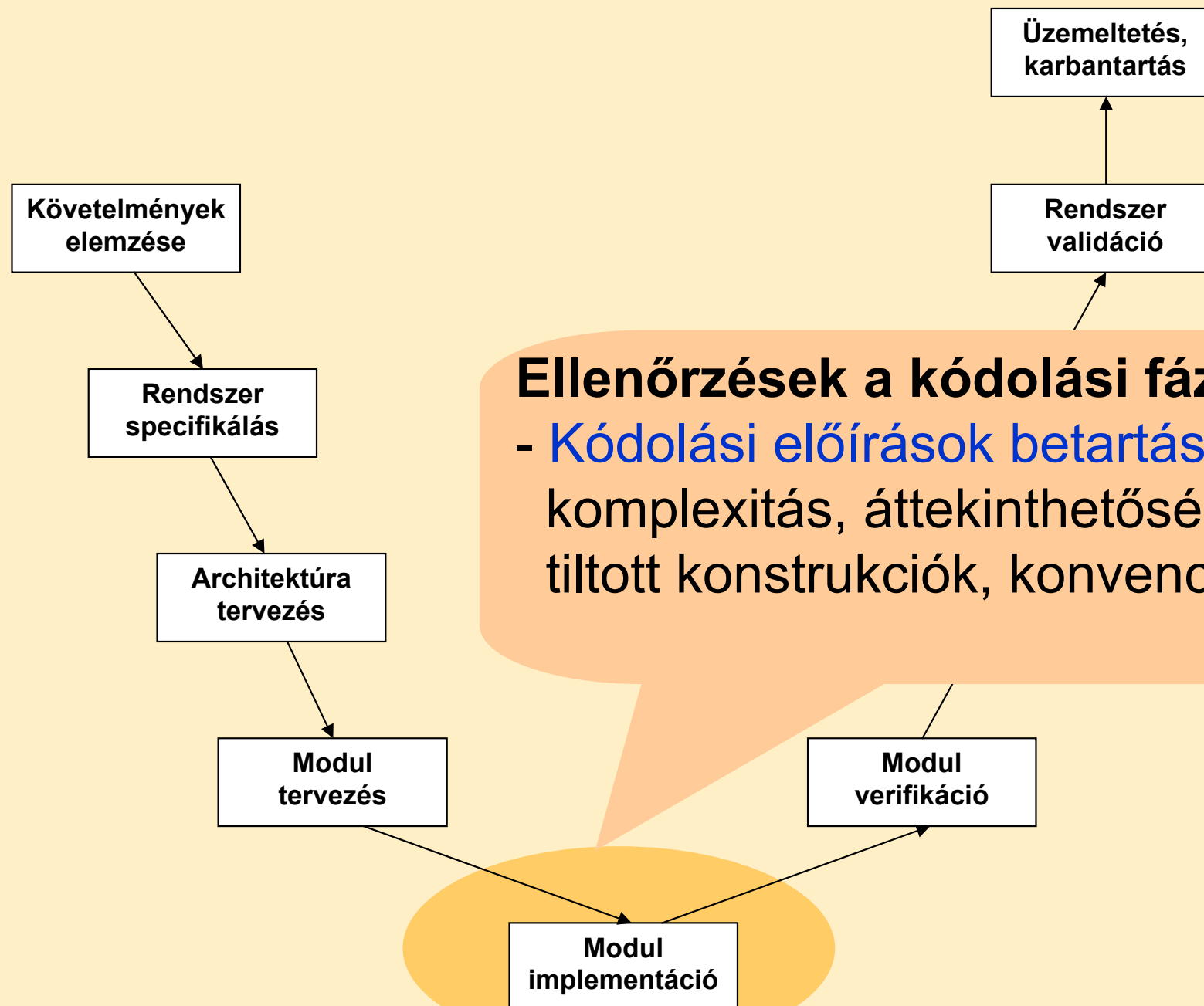
Verifikáció és validáció a V-modellben

Ellenőrzések a specifikálás és a tervezés során:

- Statikus ellenőrzés:
teljesség, ellentmondás-mentesség
- Modell szimuláció
- Formális verifikáció:
formális modellek esetén



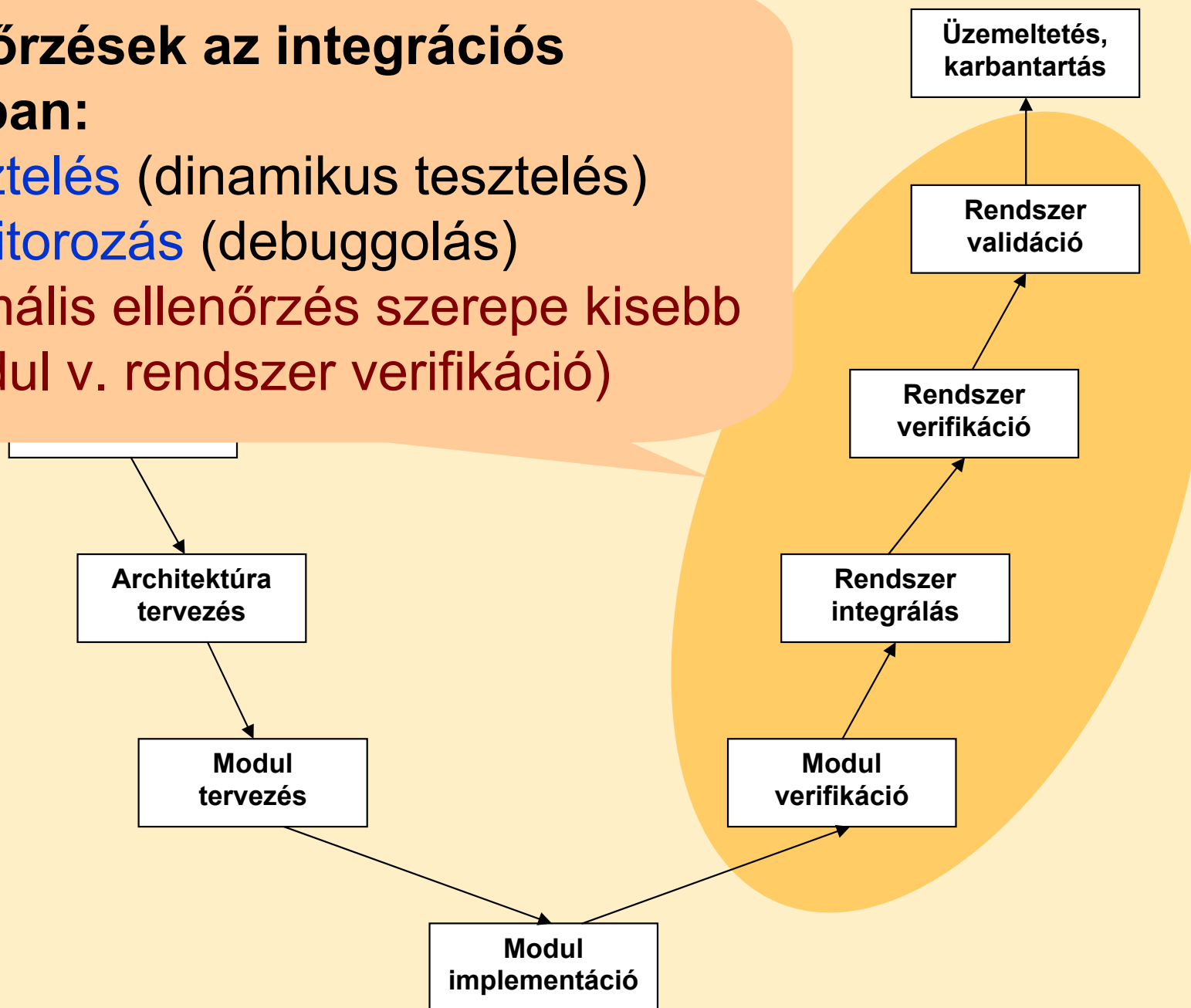
Verifikáció és validáció a V-modellben



Verifikáció és validáció a V-modellben

Ellenőrzések az integrációs fázisban:

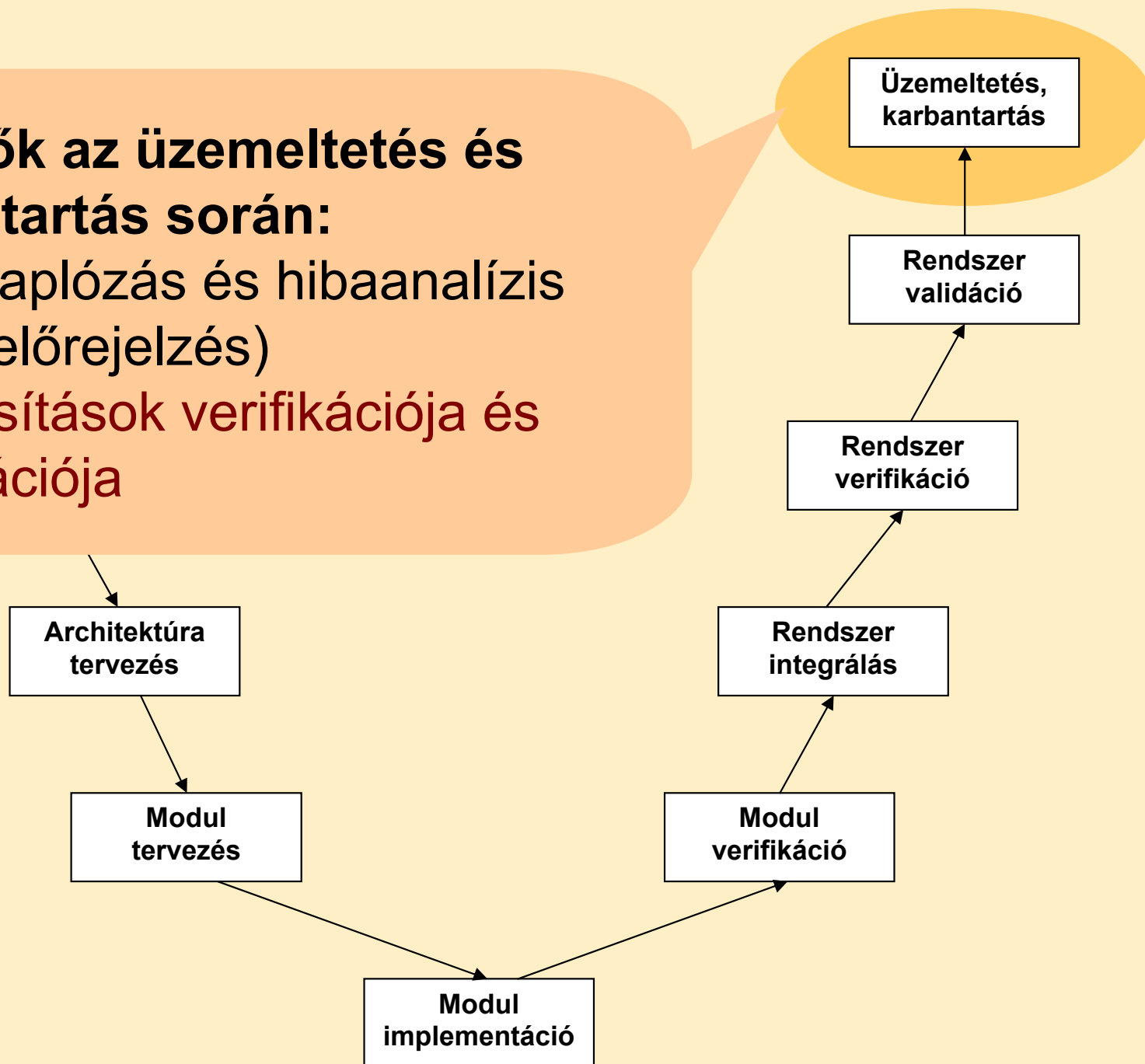
- **Tesztelés** (dinamikus tesztelés)
- **Monitorozás** (debuggolás)
- **Formális ellenőrzés szerepe kisebb** (modul v. rendszer verifikáció)



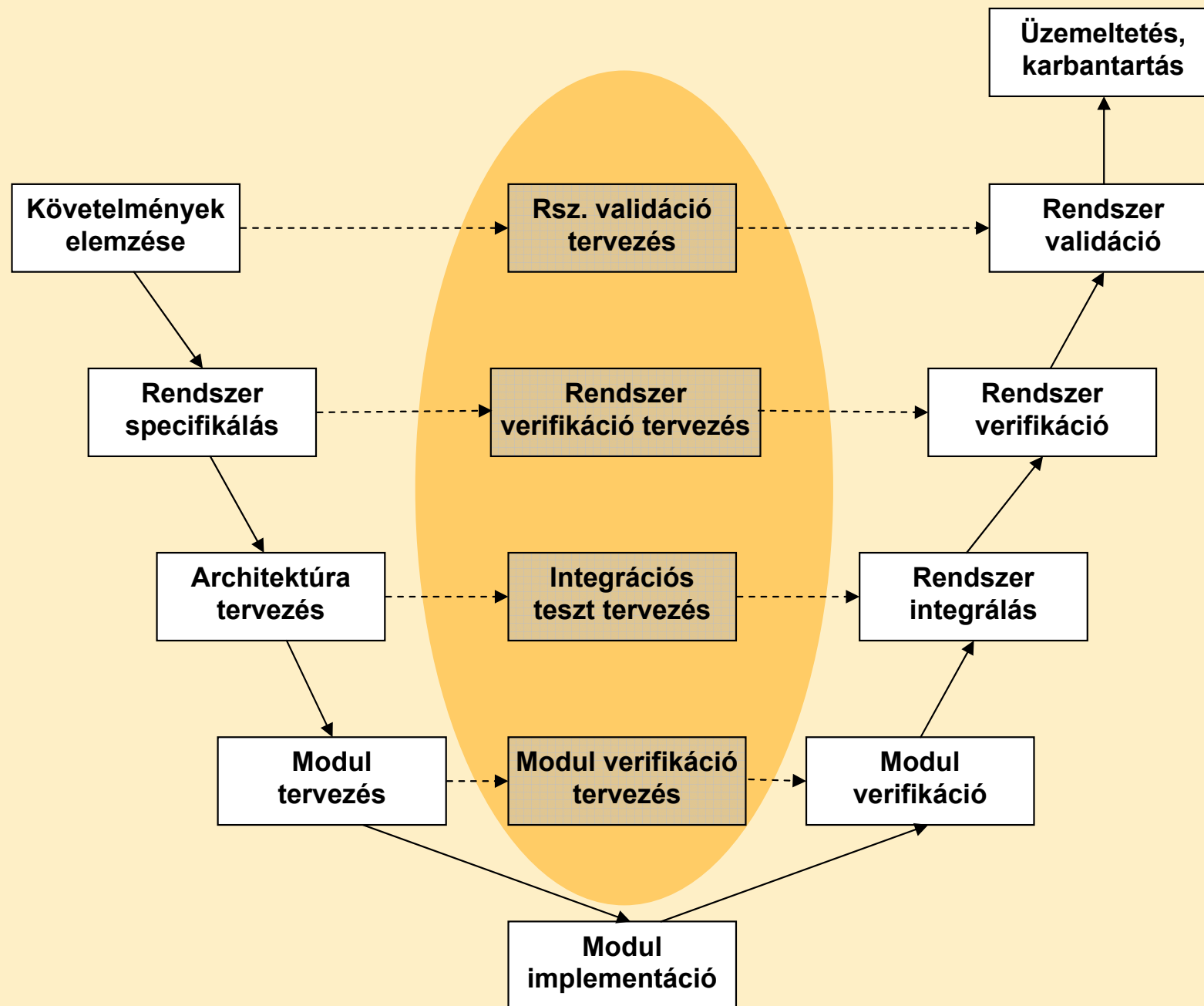
Verifikáció és validáció a V-modellben

Teendők az üzemeltetés és karbantartás során:

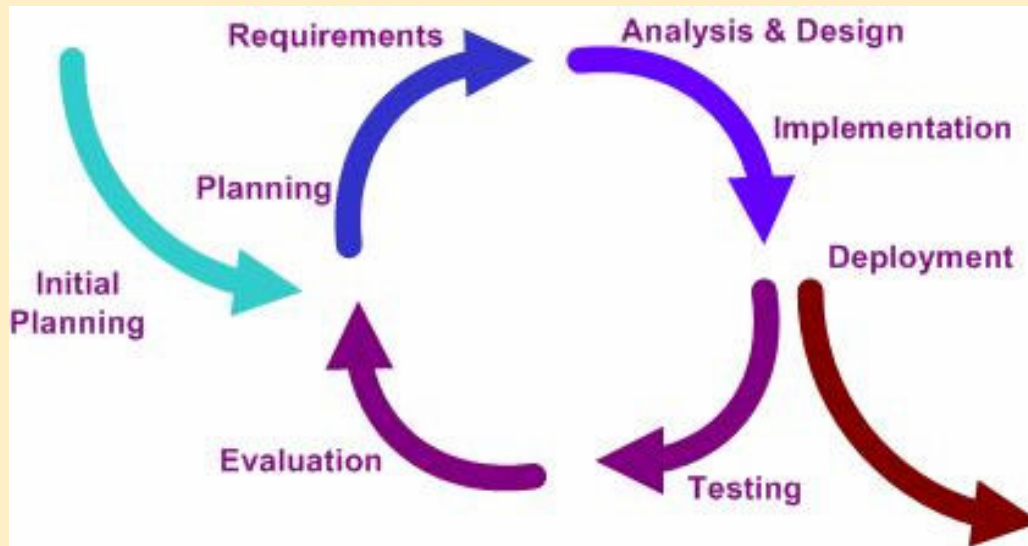
- Hibanaplózás és hibaanalízis (hiba előrejelzés)
- **Módosítások verifikációja és validációja**



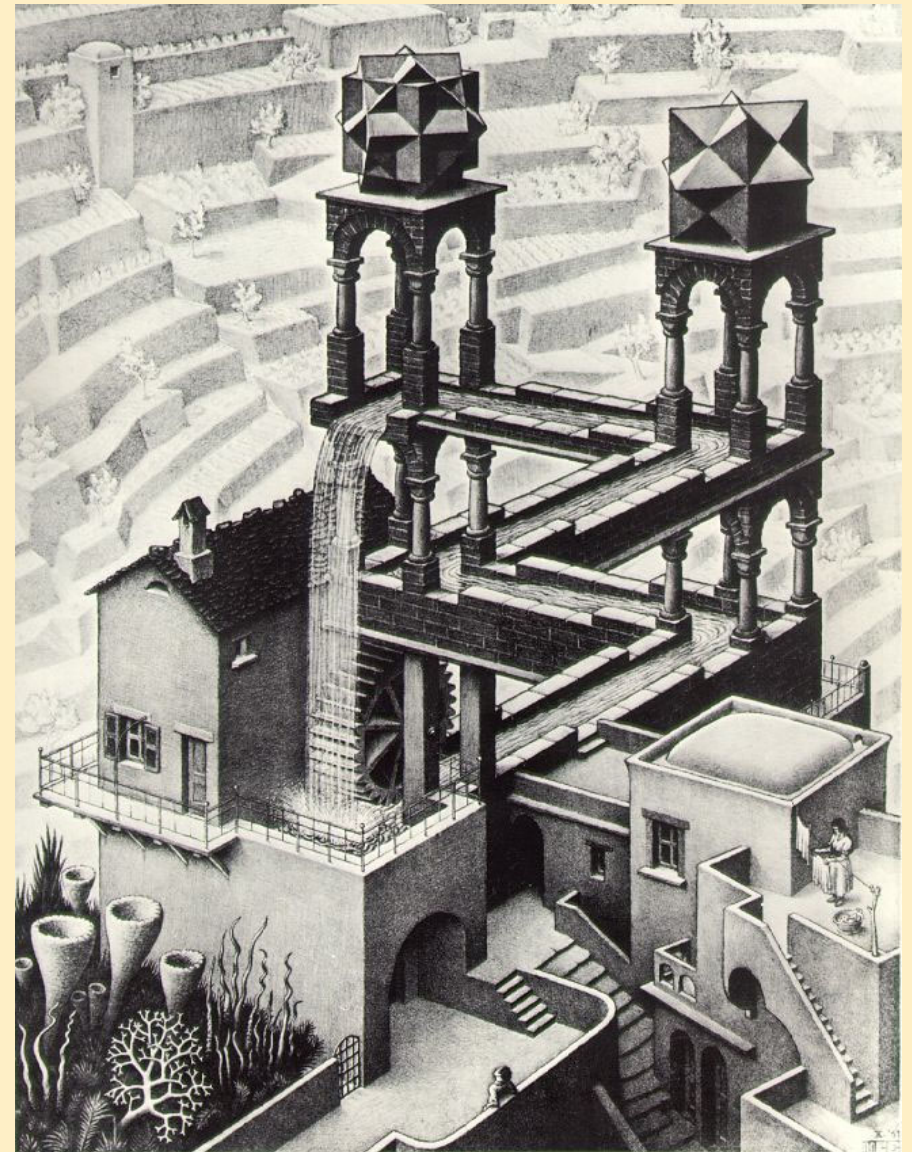
Verifikáció és validáció a V-modellben



Iteratív fejlesztési modellek



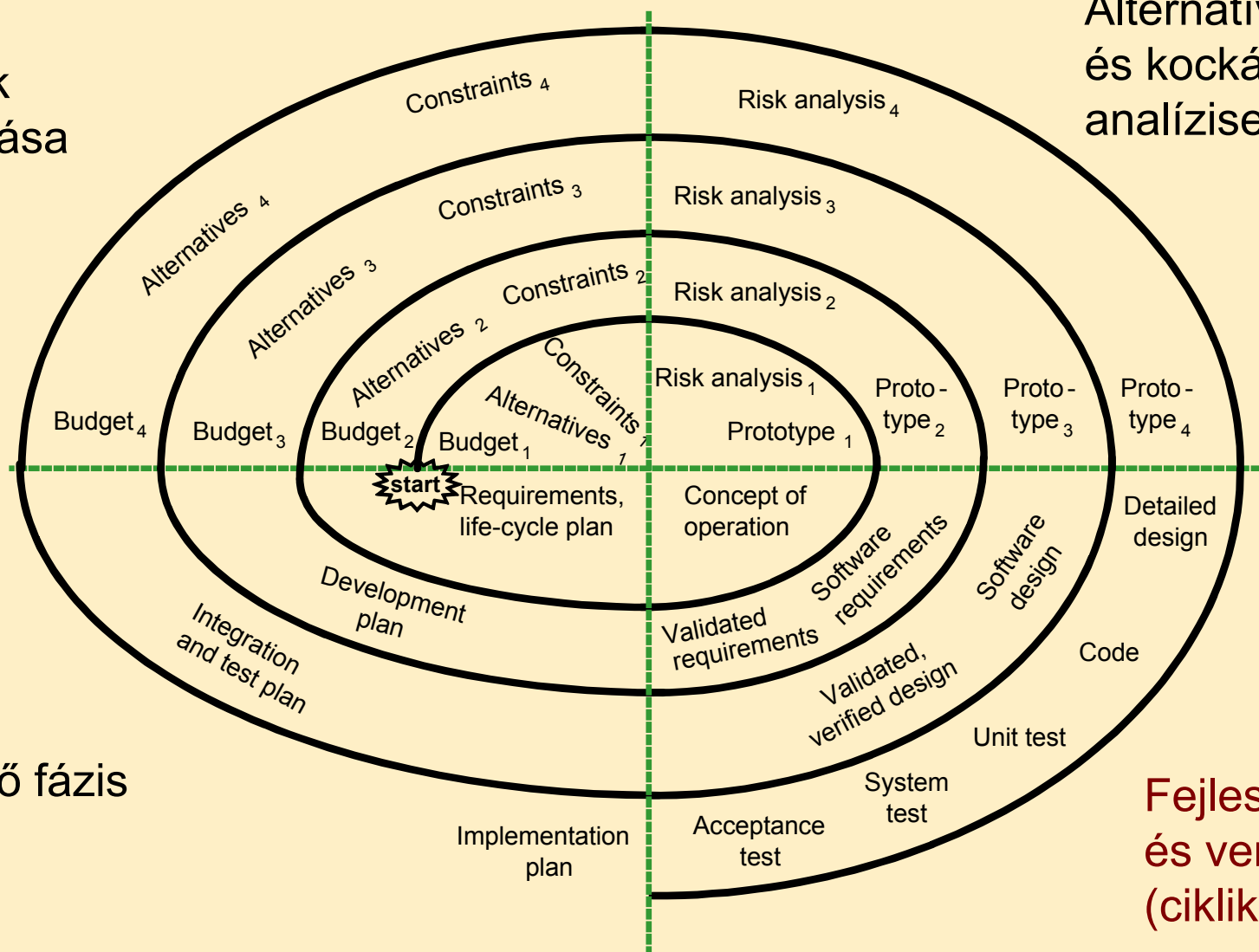
- Eddigi modellek „egy-verziós” életciklust írnak le
- Rendszerfejlesztés **iteratív**:
Prototípus → Felfedett hibák →
Továbbfejlesztés → Új prototípus
- Spirál modell



Spirál modell (Boehm)

Célok,
alternatívák,
korlátozások
meghatározása

Alternatívák
és kockázatok
analízise



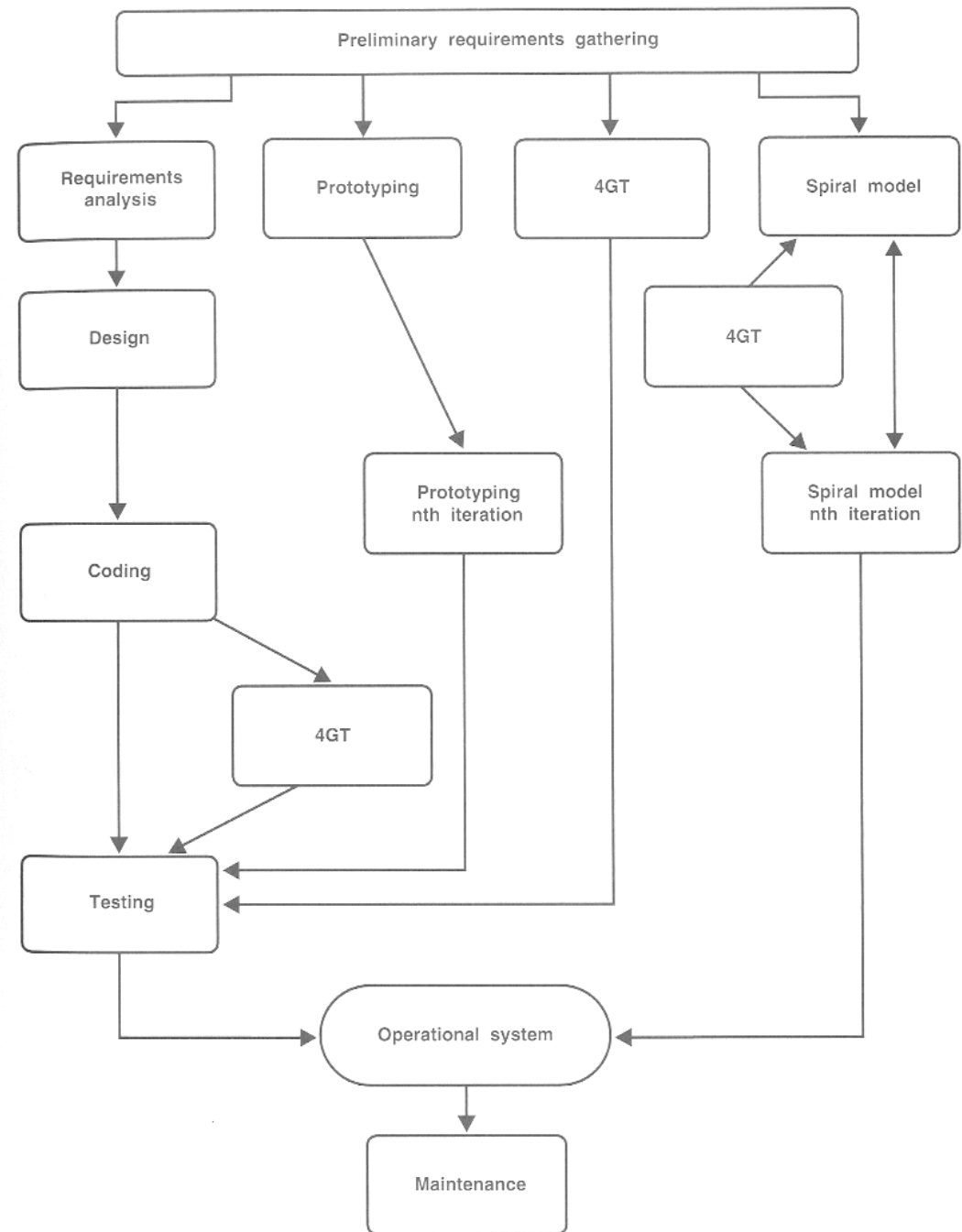
A következő fázis
tervezése

Fejlesztés
és verifikáció
(ciklikusan)

„Negyedik generációs” modell

Több ágon haladhat:

- Jól megfogható követelmények:
Hagyományos fejlesztés
- Rosszul specifikált követelmények:
Gyors prototípus fejlesztés
- Formalizálható követelmények:
Modell alapú fejlesztés
(kódgenerálás, CASE eszközök)
- Iteratív is lehet



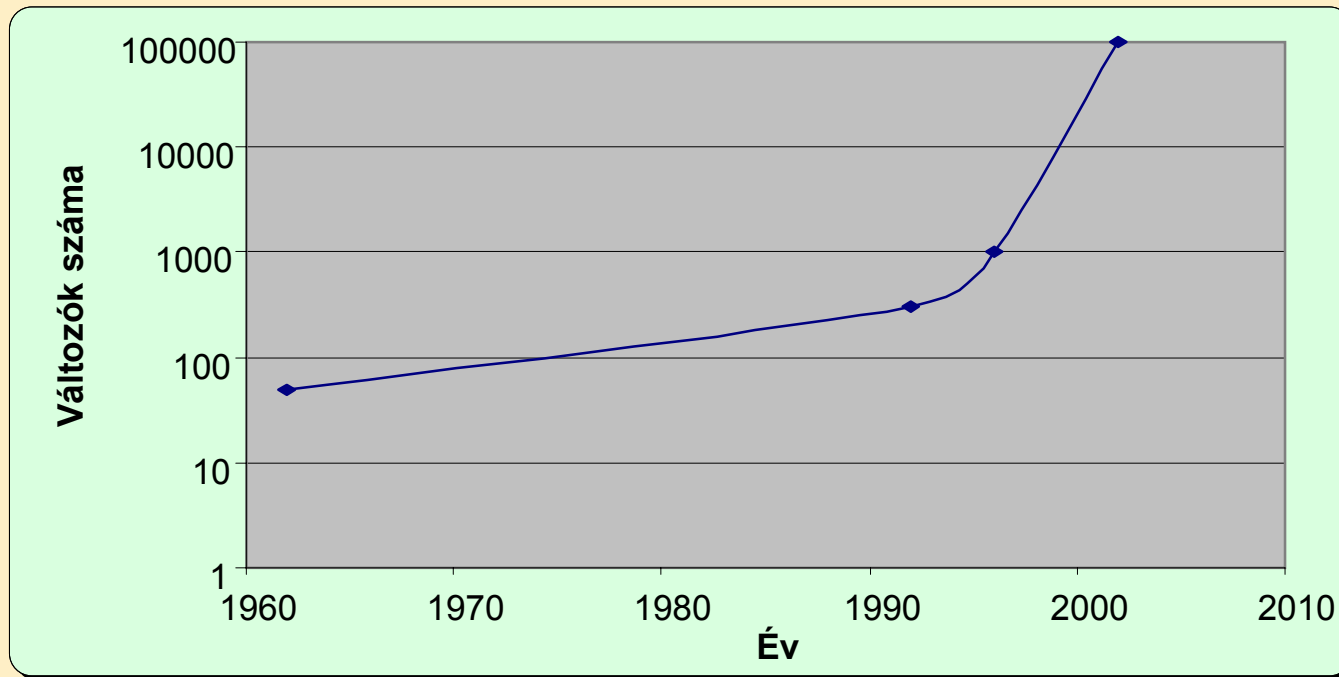
Formális módszerek értékelése

Amik a formális vizsgálatot nehezzé teszik...

- Valósághű modellezés
 - Ismeretek hiánya, feltételezések (pl. a környezetről)
 - De: Formális módszerek használatától független ez a probléma
- Speciális ismereteket igényel a felhasználótól
 - Nehéz matematikai modellek, jelölésrendszer
 - De: Mérnöki modellezési nyelvek eltakarhatják
- Bonyolultak az ellenőrzés és szintézis módszerei
 - Algoritmusok, technikák korlátait ismerni kell
 - Kézi beavatkozásra lehet szükség (pl. tételbizonyító rendszerek)
 - De: Terjednek a „gombnyomásra működő” eszközök
- Csak „kisméretű” problémákra alkalmazható
 - Modell, állapottér kezelhető-e a meglévő erőforrásokkal
 - De: Eszközök hatékonysága folyamatosan nő

A formális verifikáció fejlődése

- A SAT eszközök (kielégíthetőség) lehetőségei:



- Modellellenőrző eszközök képességei:
 - $10^{20} \sim 2^{66}$ méretű állapottér (ROBDD-vel, 1990)
 - $10^{100} \sim 2^{328}$ méretű állapottér speciális esetben

Jelen helyzet

- Megkötések

- Diszkrét állapotú
- Diszkrét idejű
- Diszkrét eseményterű

} rendszerek (DES)

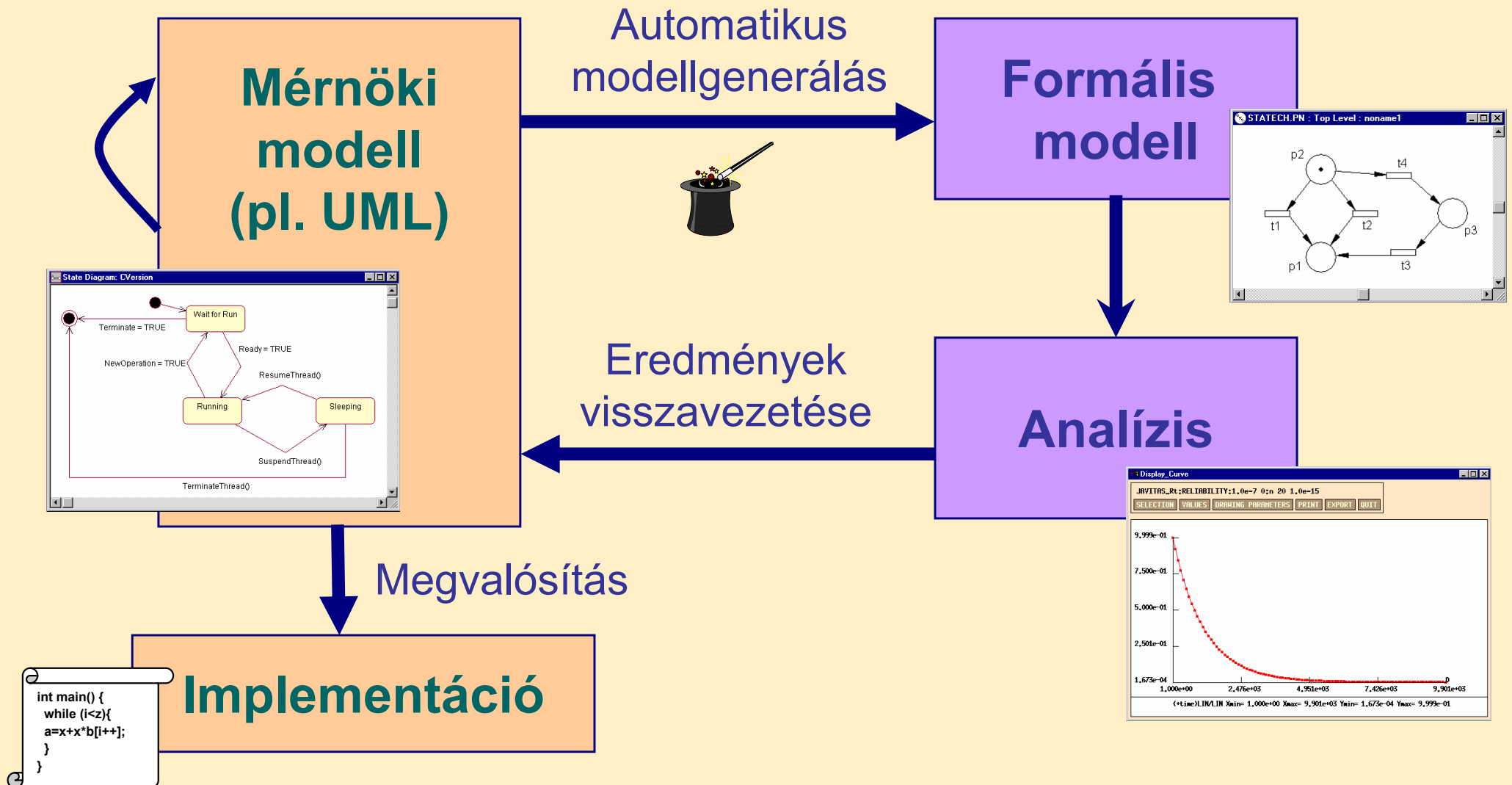
- Kulcskérdések

- Matematikai algoritmusok hatékonysága és szintje
- Modell osztályok korlátai (pl. időzítés)
- Modell készítés nehézsége (pl. absztrakció)
- Nyelvek kifejezőképessége (pl. VHDL)
- Nyelvek nem pontos definíciója (pl. UML)

Sikeres megközelítés

Inf. rendszer tervezése

Formális ellenőrzés



Modellek a formális ellenőrzéshez

- Rendszermodellek

- Mérnöki modellek

- Pl. UML diagramok (fél-)formális szemantikával

- Magasabb szintű modellek:

- Vezérlés orientált: Automata, Petri-háló
 - Adatfeldolgozás orientált: Adatfolyam háló
 - Kommunikáció orientált: Processz algebra

- Alapszintű matematikai modellek:

- KS, LTS, KTS, automata, Büchi automata

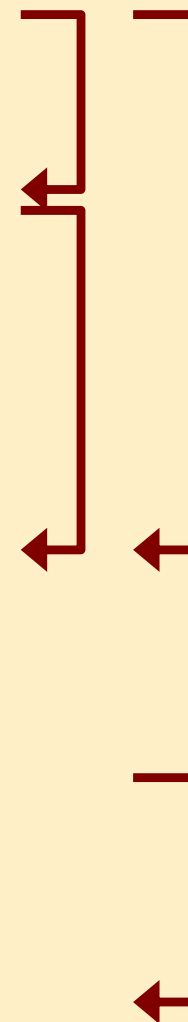
- Tulajdonság leírások

- Magasabb szintű:

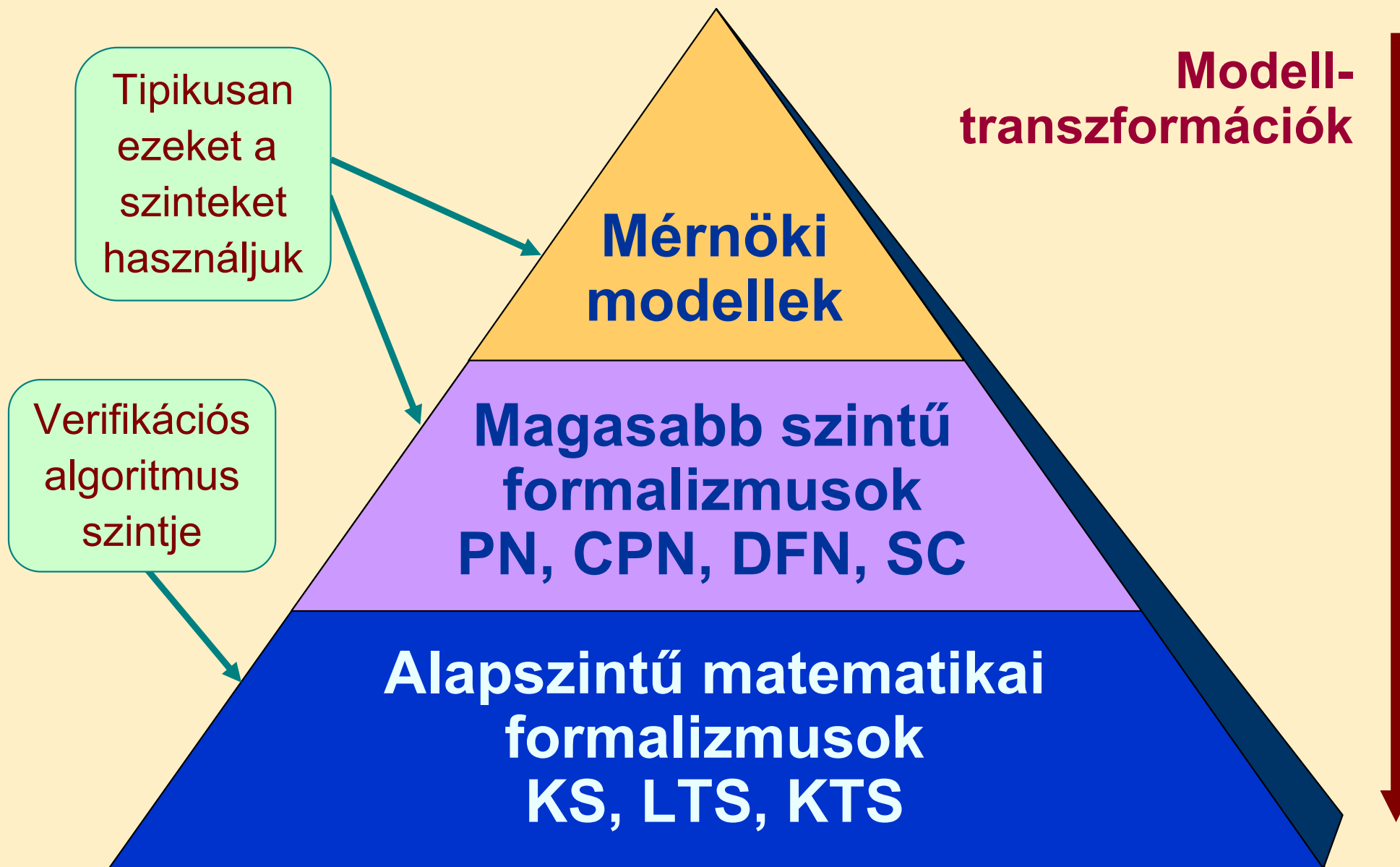
- Idődiagram, szekvencia diagram (MSC)

- Alapszintű:

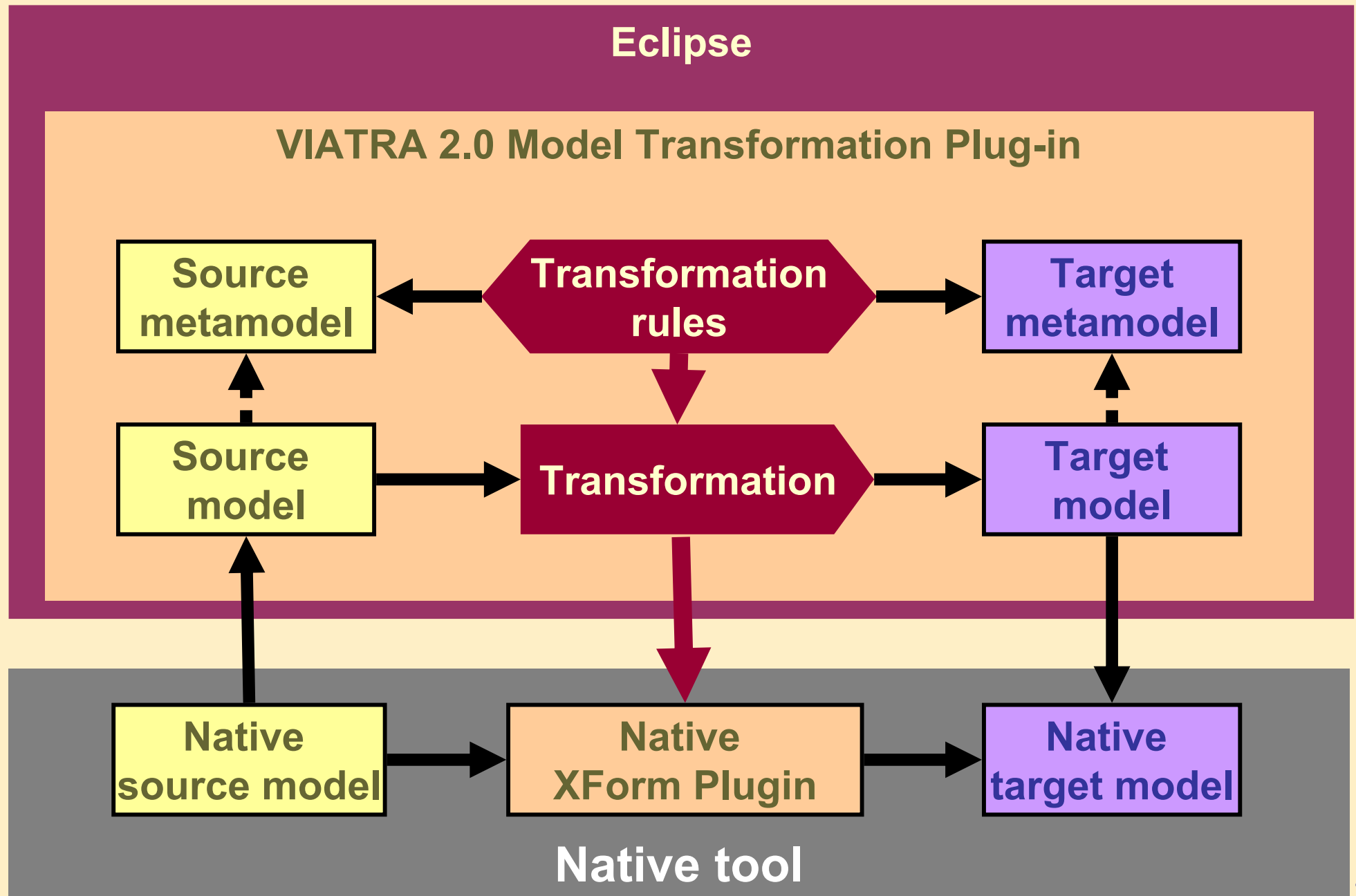
- Elsőrendű logika, temporális logika, referencia automata



Visszaautalás: A tárgy felépítése



Egy modelltranszformációs keretrendszer



Modelltranszformációval elérhető eszközök (BME MIT)

Specifikáció fázisa:

- UML állapottérkép specifikáció ellenőrzése:
Teljesség és ellentmondásmentesség

Architektúra tervezés és részletes tervezés fázisa:

- Hibamodellezés adatfolyam hálókkal:
Hibaterjedési és tesztelhetőségi analízis
- Megbízhatósági modellezés Petri-hálókkal:
Rendszerszintű megbízhatóság jellemzői
- Modell ellenőrzése állapottérképeken:
Biztonsági tulajdonságok (állapotok elérhetősége)

Prototípus és megvalósítási fázis:

- Automatikus forráskód generálás:
Futási idejű monitorozás és ellenőrzés beépítve

Sikertörténetek

Klasszikus alkalmazások

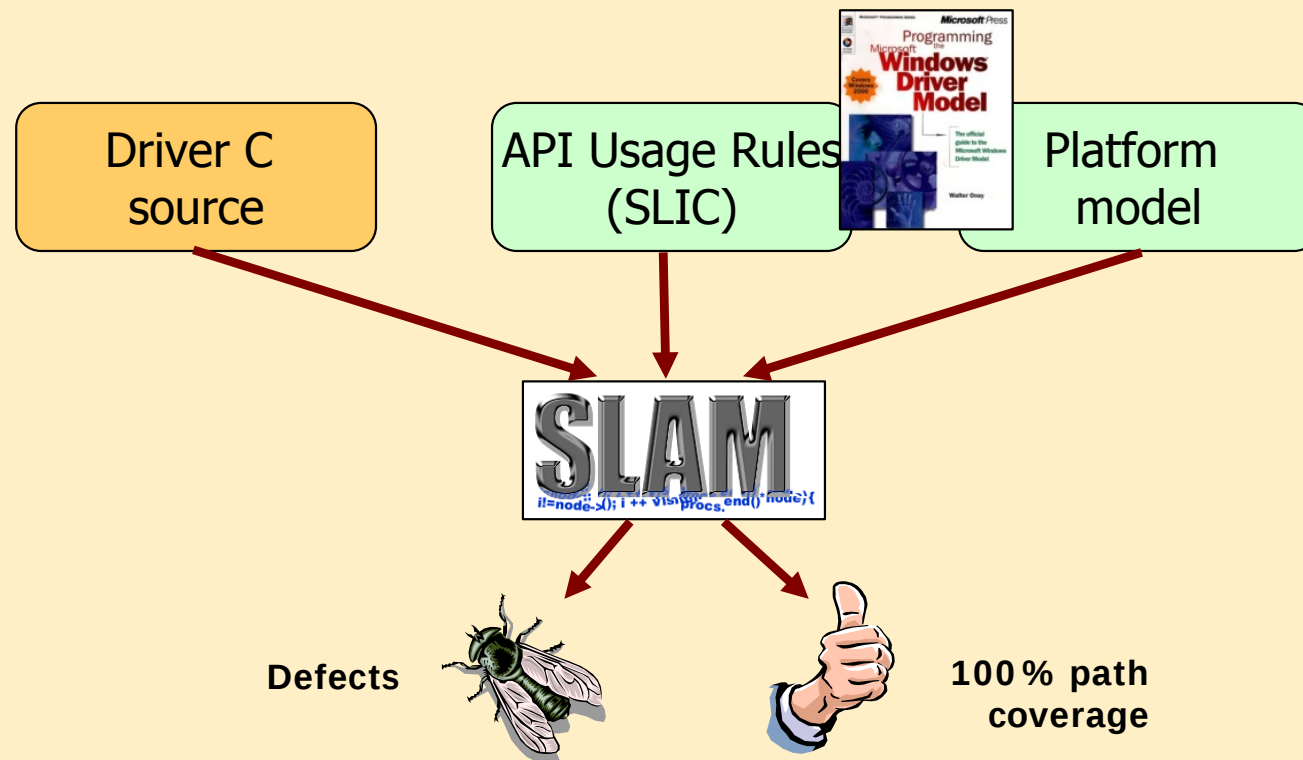
- USA TCAS-II forgalomirányító rendszer
 - RSMIL nyelven specifikált; teljesség és ellentmondás-mentesség ellenőrzése
- Philips Audio Protocol
 - 1994: manuális verifikáció, majd 1996: automatikus ellenőrzés (HyTech)
- Lockheed C130J repülési szoftvere
 - Programfejlesztés helyességbizonyítással (CORE spec. nyelv + Ada)
 - Költség nem nőtt a tesztelés egyszerűsödése miatt
- IEEE Futurebus+ szabvány
 - Carnegie Mellon SMV: cache koherencia protokoll hibájának kiderítése
- Hardver projektek: ACL2 automatikus tételbizonyító
 - Motorola DSP Complex Arithmetic Processor mag (250 regiszter): DSP algoritmusok ellenőrzése
 - AMD 5K86 processzor: Lebegőpontos osztás algoritmusának ellenőrzése
- Intel i7 processzor
 - *„For the recent Intel Core™ i7 design we used formal verification as the primary validation vehicle for the core execution cluster”*
 - Szimbolikus szimuláció az adatutak teljes vizsgálatára (2700 mikroutasítás, 20 mérnökévnyi munka) – Binary Decision Diagram alkalmazása
- Modell alapú szoftverfejlesztéshez kapcsolódó eszközök
 - IBM, Esterel, Prover, Mentor, Telelogic, ...

Forráskódhoz illeszkedő formális verifikáció

- Java
 - Bandera, PathFinder: modell absztrakció
 - Java VM formalizálása: Abstract State Machine
- Ada
 - SPARK Ada verification condition generator tételbizonyítóhoz
- C
 - BLAST: Szoftver modellellenőrző C programokhoz (absztrakció)
 - CBMC: C alapú korlátos modellellenőrző
- C#, Visual Basic .Net
 - Zing (MS Visual Studio-hoz): Konkurens szoftver modellellenőrzése
- Spec# (C# superset)
 - MS Research Boogie 2: Specifikációs nyelvi kiterjesztések
 - Helyességi kritériumok ellenőrzése: program absztrakcióval és tételbizonyítóval (Z3)
- Microsoft Windows Driver Kit (WDK)
 - Static Driver Verifier Research Platform, SLAM 2 eszköz
 - Windows API használati feltételeinek statikus ellenőrzése

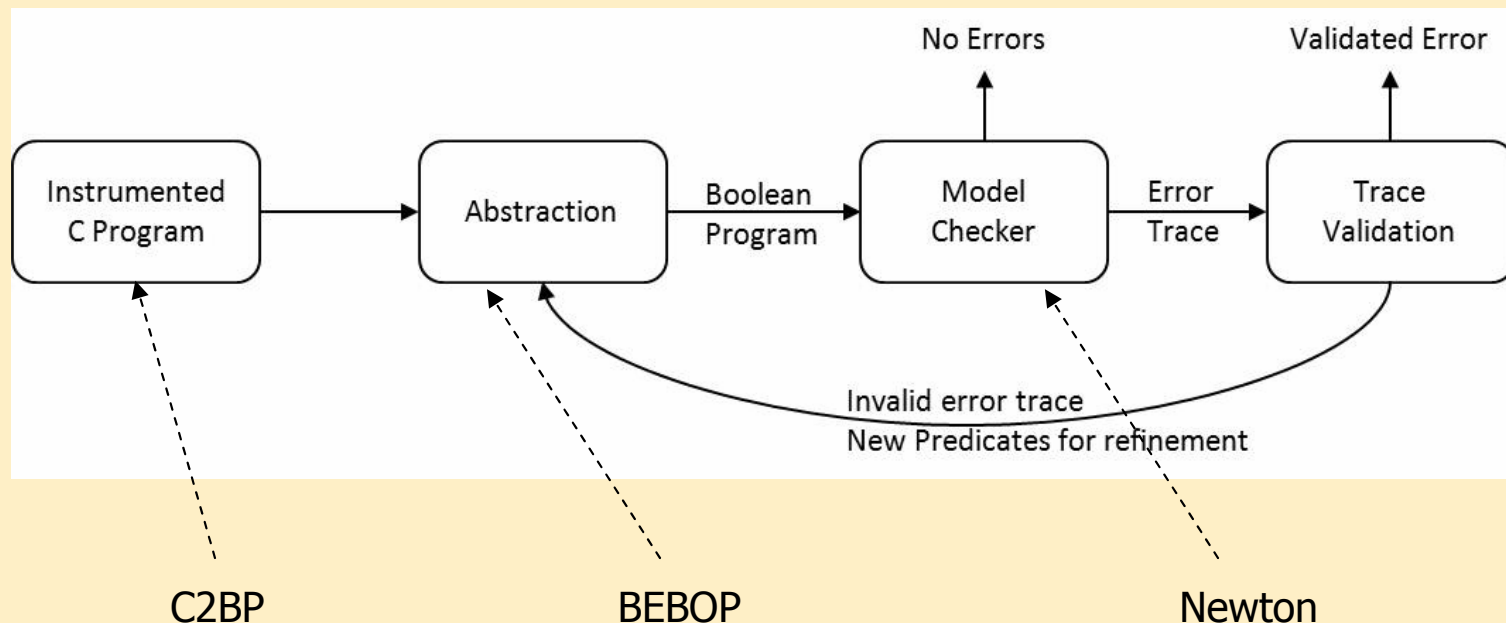
Példa: SLAM

- Motiváció: Hibás meghajtók megzavarhatják az OS működését
 - Pl: hibás zárolások erőforrásokon (nincs zárolva, nincs felszabadítva, ...)
- Megoldás:
 - Szabályokkal leírható a helyes használat (pl. zárhasználat „állapotgépe”)
 - Ennek teljesülését ellenőrzi a SLAM a forráskódon
 - Forráskód absztrakciót használ: Boole-program állapotainak vizsgálata
 - Megtalált hibás (szabályokhoz nem illeszkedő) utakat vissza kell ellenőrizni



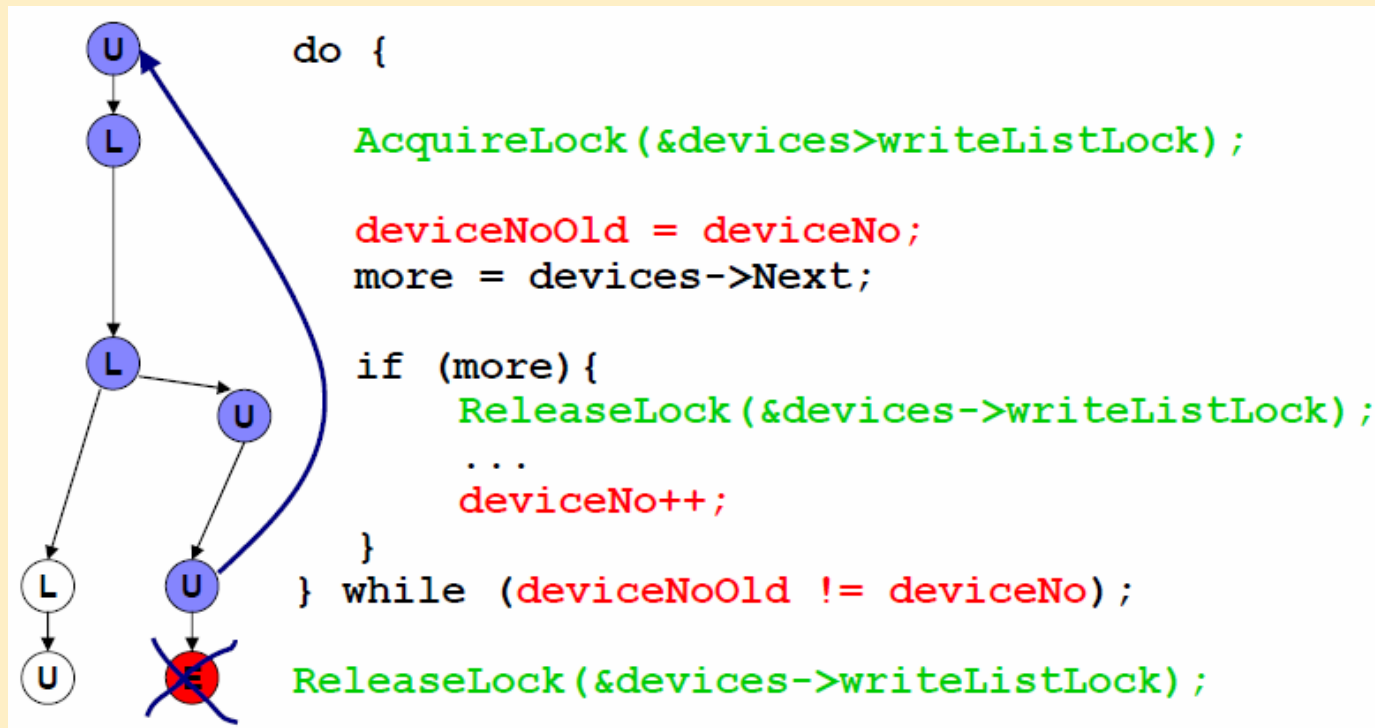
Példa: SLAM

- Motiváció: Hibás meghajtók megzavarhatják az OS működését
 - Pl: hibás zárolások erőforrásokon (nincs zárolva, nincs felszabadítva, ...)
- Megoldás:
 - Szabályokkal leírható a helyes használat (pl. zárhasználat „állapotgépe”)
 - Ennek teljesülését ellenőrzi a SLAM a forráskódon
 - Forráskód absztrakciót használ: Boole-program állapotainak vizsgálata
 - Megtalált hibás (szabályokhoz nem illeszkedő) utakat vissza kell ellenőrizni

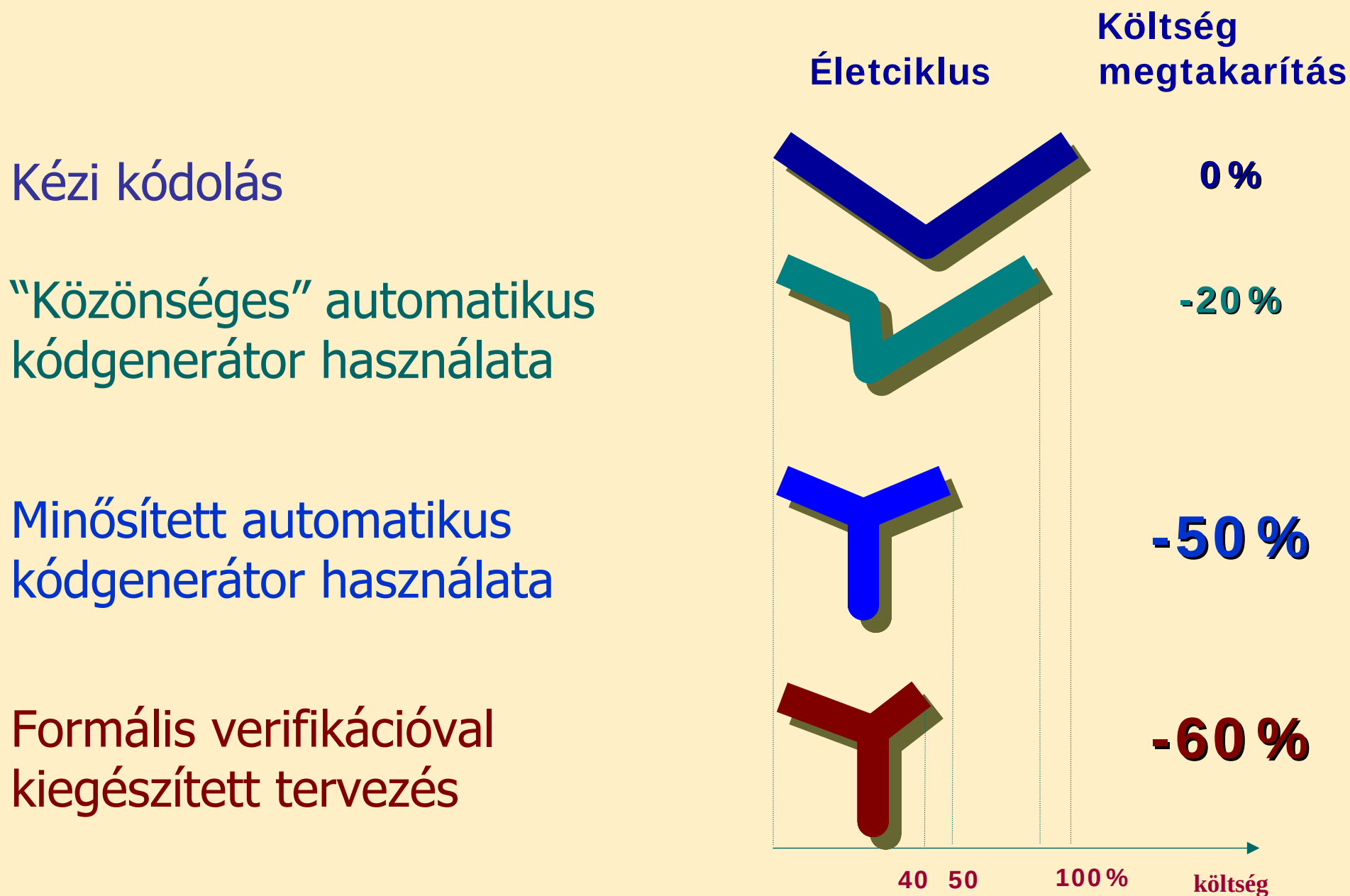


Példa: SLAM

- Motiváció: Hibás meghajtók megzavarhatják az OS működését
 - Pl: hibás zárolások erőforrásokon (nincs zárolva, nincs felszabadítva, ...)
- Megoldás:
 - Szabályokkal leírható a helyes használat (pl. zárhasználat „állapotgépe”)
 - Ennek teljesülését ellenőrzi a SLAM a forráskódon
 - Forráskód absztrakciót használ: Boole-program állapotainak vizsgálata
 - Megtalált hibás (szabályokhoz nem illeszkedő) utakat vissza kell ellenőrizni



V-től az Y modellig (Esterel Technologies)



Összefoglalás

- Mik a formális módszerek?
 - Formalizmus, formális nyelv
 - Formális módszerek és eszközök:
Szimuláció, formális verifikáció, szintézis
- Mire használhatók?
 - Motiváció: Szoftver minőségi kihívások
 - A formális módszerek lehetőségei
 - Fejlesztési folyamatok
- Mit várhatunk?
 - Korlátok
 - Sikertörténetek