Közlekedési automatika

Biztonsági folyamatirányító rendszerek szoftvere

Dr. Sághi Balázs diasora alapján
összeállította, kiegészítette: Lövétei István Ferenc

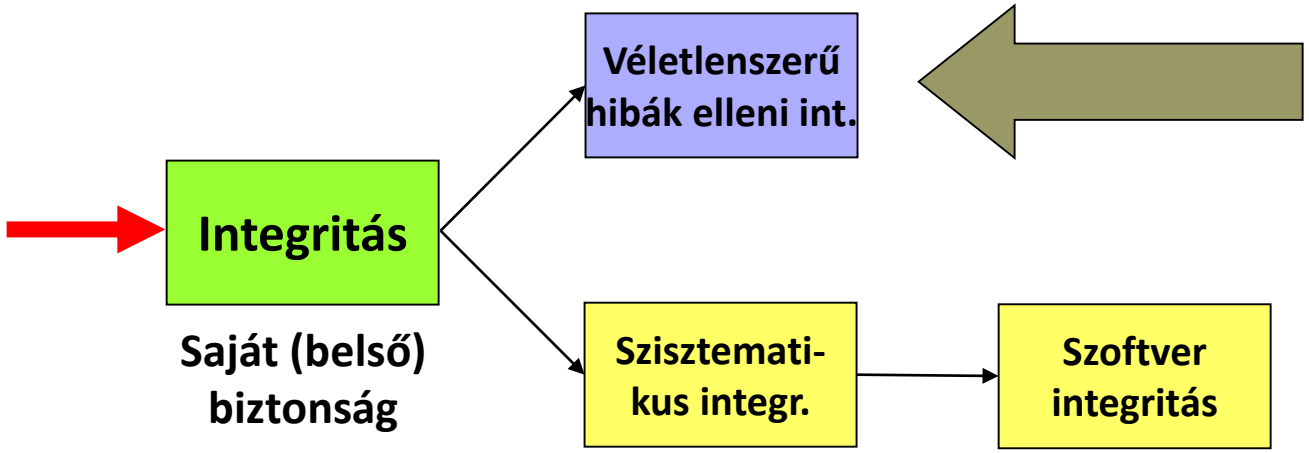
BME Közlekedés- és Járműirányítási Tanszék
2019

Tartalomjegyzék

- Biztonságintegritás, az irányító rendszerek belső veszélyforrásai (ism.),
- Szoftverek szerepe a folyamatirányító rendszerekben,
- Szoftver életciklus modellek,
- Szoftverek architektúrája,
- Szoftverek megbízhatósága,
- Szoftverek tervezése és implementációja,
- Szoftverek tesztelése,
- Szoftverek validációja.

Biztonságintegritási szintek és hibakezelés

Véletlen hardver hibák elleni védelem:
- A biztonsági szintnek megfelelő megbízhatóság elérése, biztonsági stratégiák



Hibadetektálás
Hibatűrés
Biztonsági állapot

A hibák
- **elkerülése**
- **eltávolítása**

Diverz kialakítás

Szisztematikus hibák elleni védelem:

- **Minőségbiztosítás a teljes életciklusban (szervezett folyamat)**
- **A biztonsági szintnek megfelelő fejlesztési módszerek alkalmazása**

A biztonsági rendszerek belső veszélyforrásai

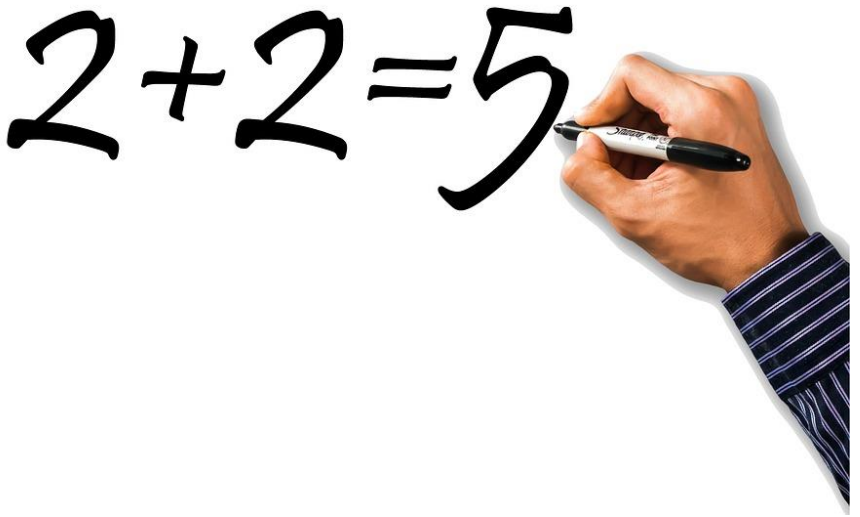
- Szoftverek szisztematikus hibái

- Az SW rendszer **létrehozása során** elkövetett emberi hibák, amelyek

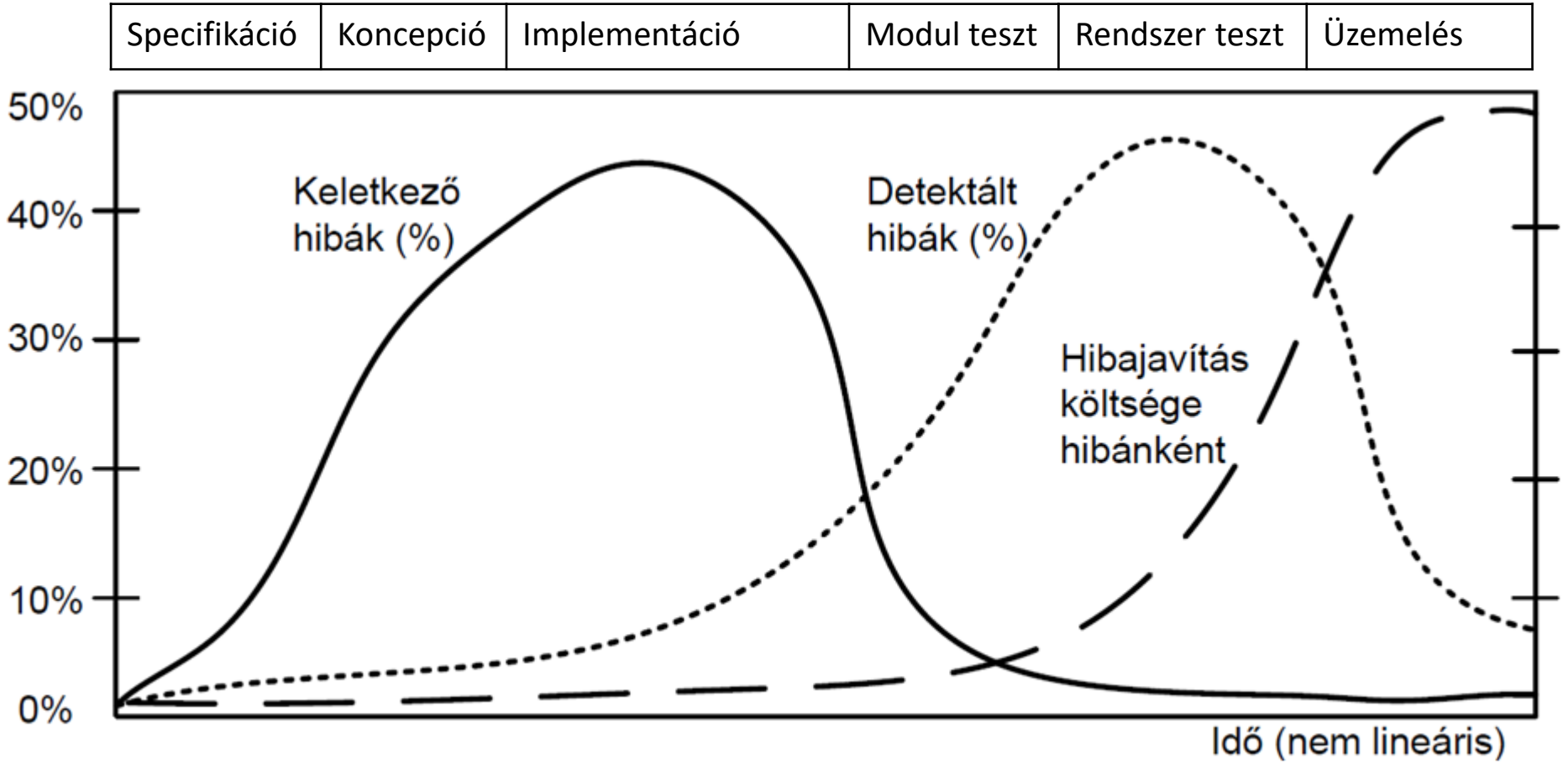
- a rendszer üzemelése során helytelen működést okoznak.

- Specifikációs hibák, tervezési hibák, kódolási hibák, tesztelési hibák, stb.

- Fellépési gyakoriság nem adható meg.



Hibák keletkezése és detektálása



Szoftverek szerepe a folyamatirányító rendszerekben

- Programozott irányítórendszerek:
  - Célgépek:
    - Nincs operációs rendszer,
    - Egyszerű szoftver,
    - Pl. egy-chipes mikrokontrollerek.,
    - Felhasználás pl. járművek elektronikus vezérlőegységei (ECU).

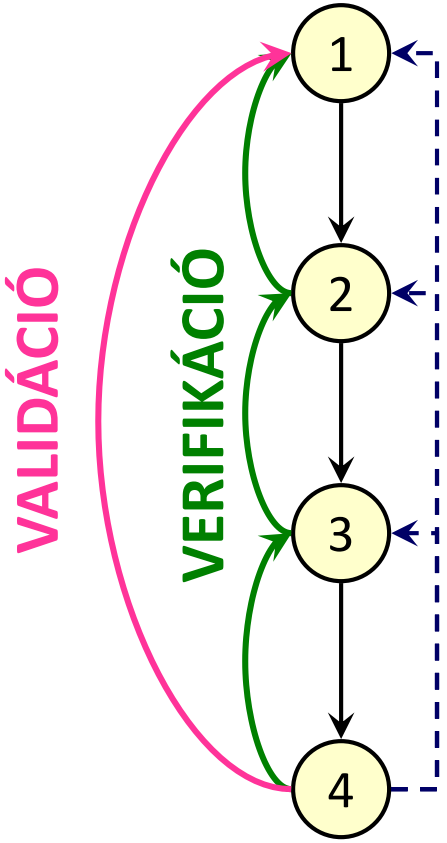
pl. Ranseas V850 MCU for safety applications, link

Szoftverek szerepe a folyamatirányító rendszerekben

- Programozott irányítórendszerek:
  - Univerzális alkalmazású rendszerek:
    - Moduláris hardver (általában kártya rendszerű),
    - Tagolt szoftverfelépítés,
    - Alkalmazástól független HW felépítés, azaz több különböző feladatra is használható.

pl. SIMIS W biztosítóberendezés ECC számítógépe, link

Szoftver életciklus modellek – leegyszerűsítve



SPECIFIKÁCIÓ

A szoftver feladatának meghatározása

SZOFTVER TERVEZÉS

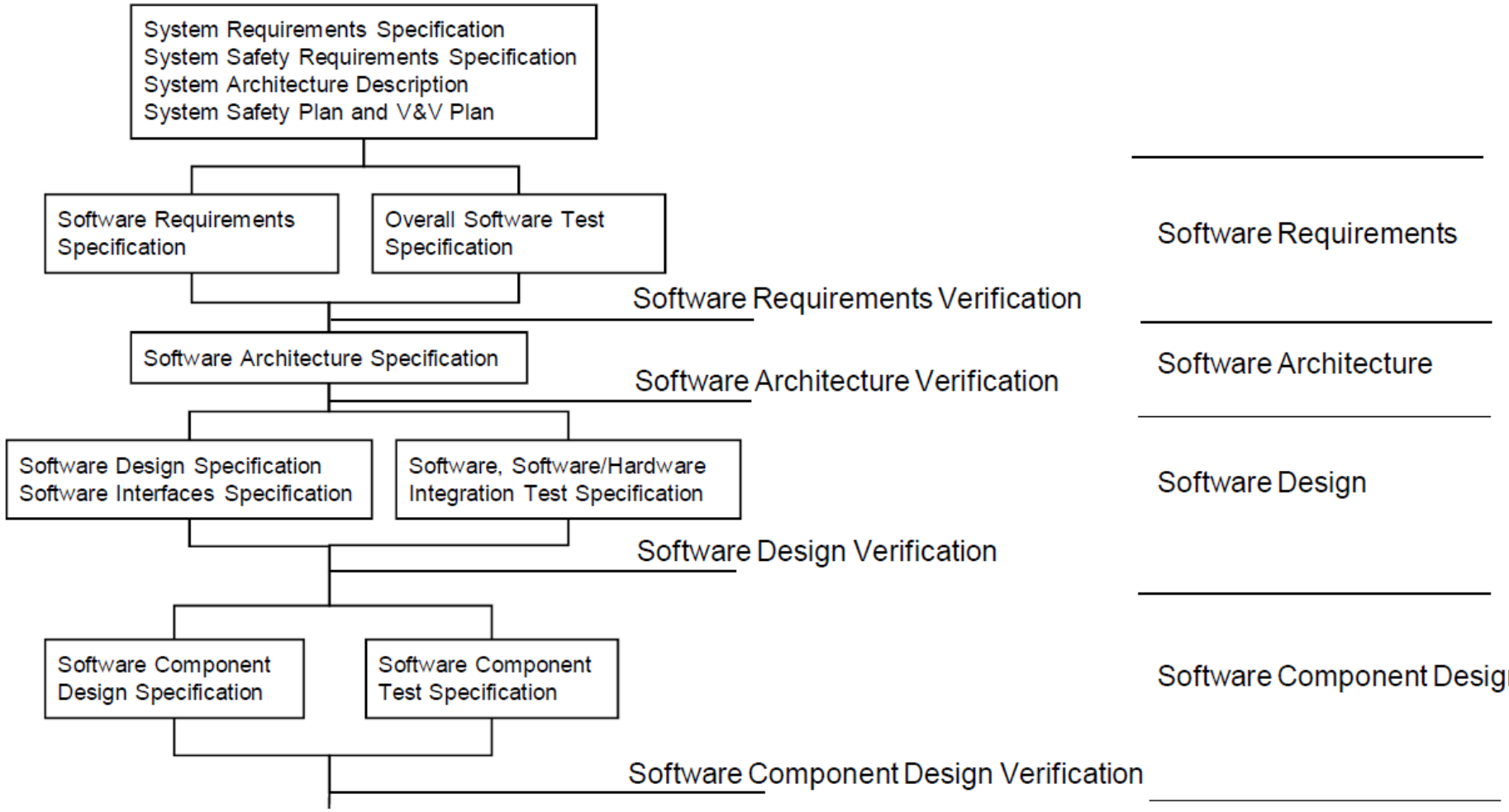
KÓDOLÁS

IMPLEMENTÁCIÓ

KARBANTARTÁS

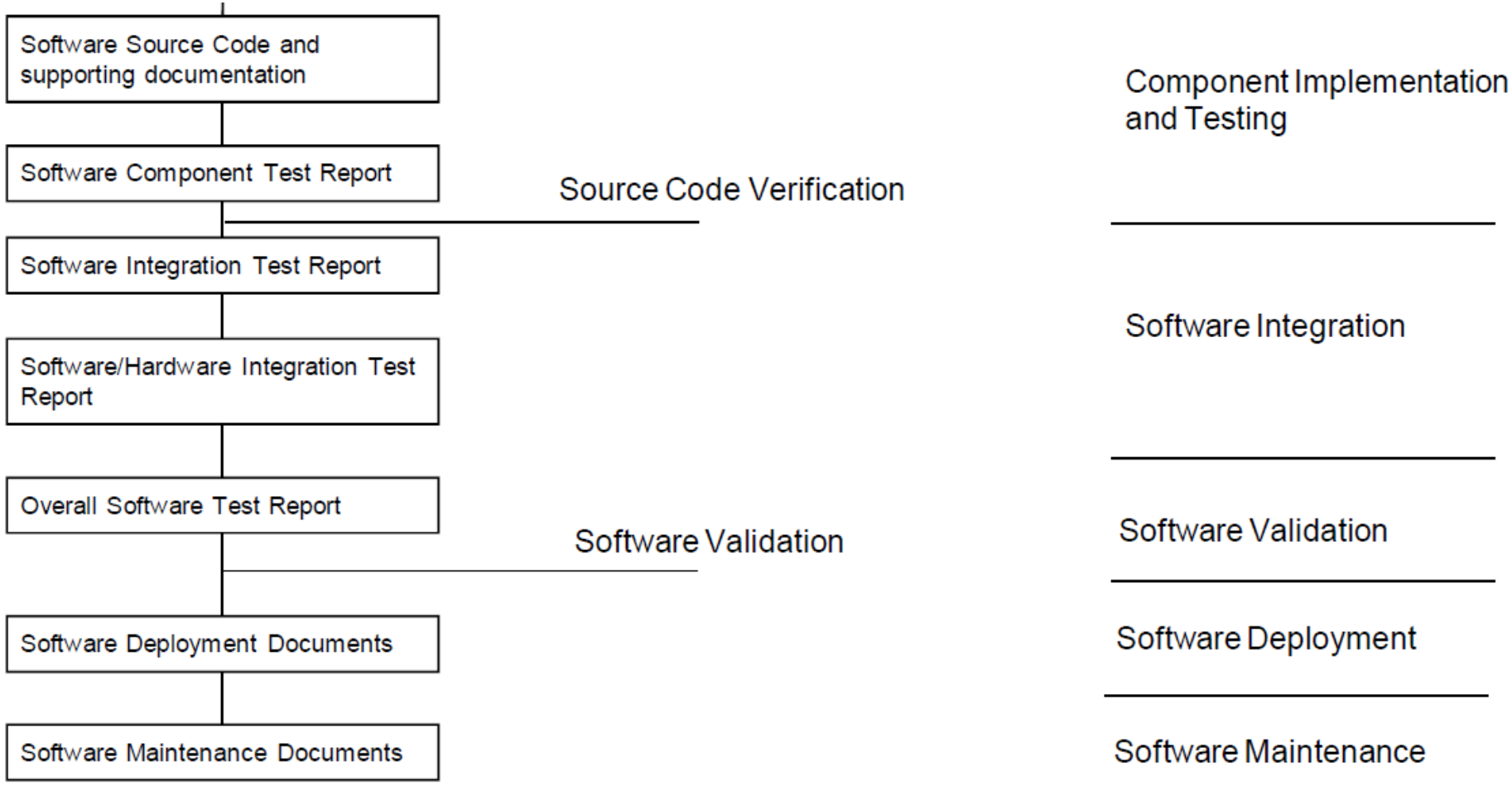
- Hibák javítása
- Módosítások

Szoftver életciklus modellek - példa



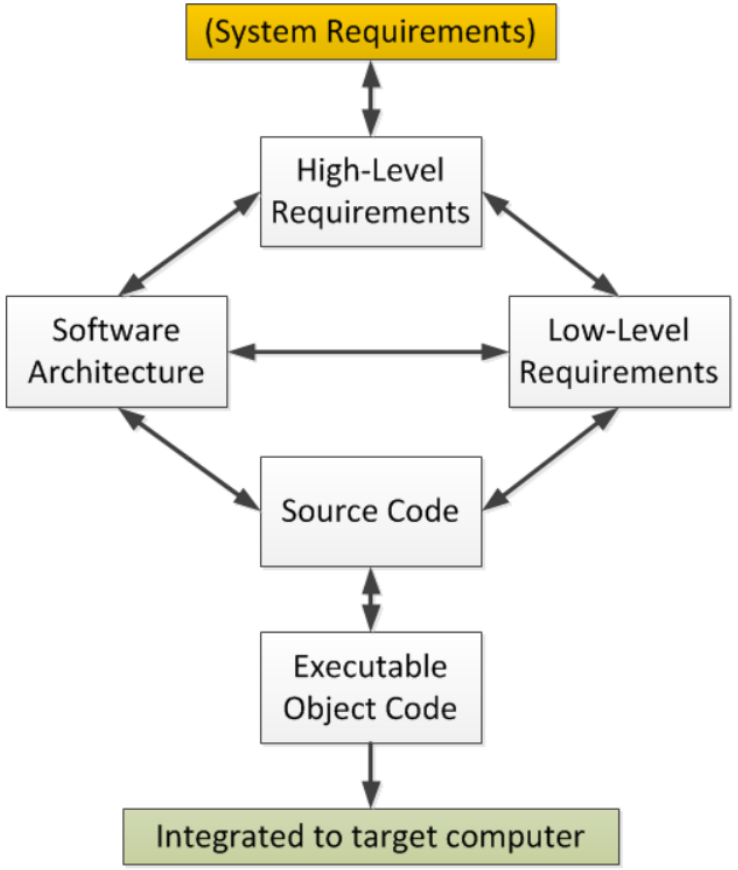
- SW életciklus modell 1. rész - MSZ EN 50128

Szoftver életciklus modellek - példa



- SW életciklus modell 2. rész - MSZ EN 50128

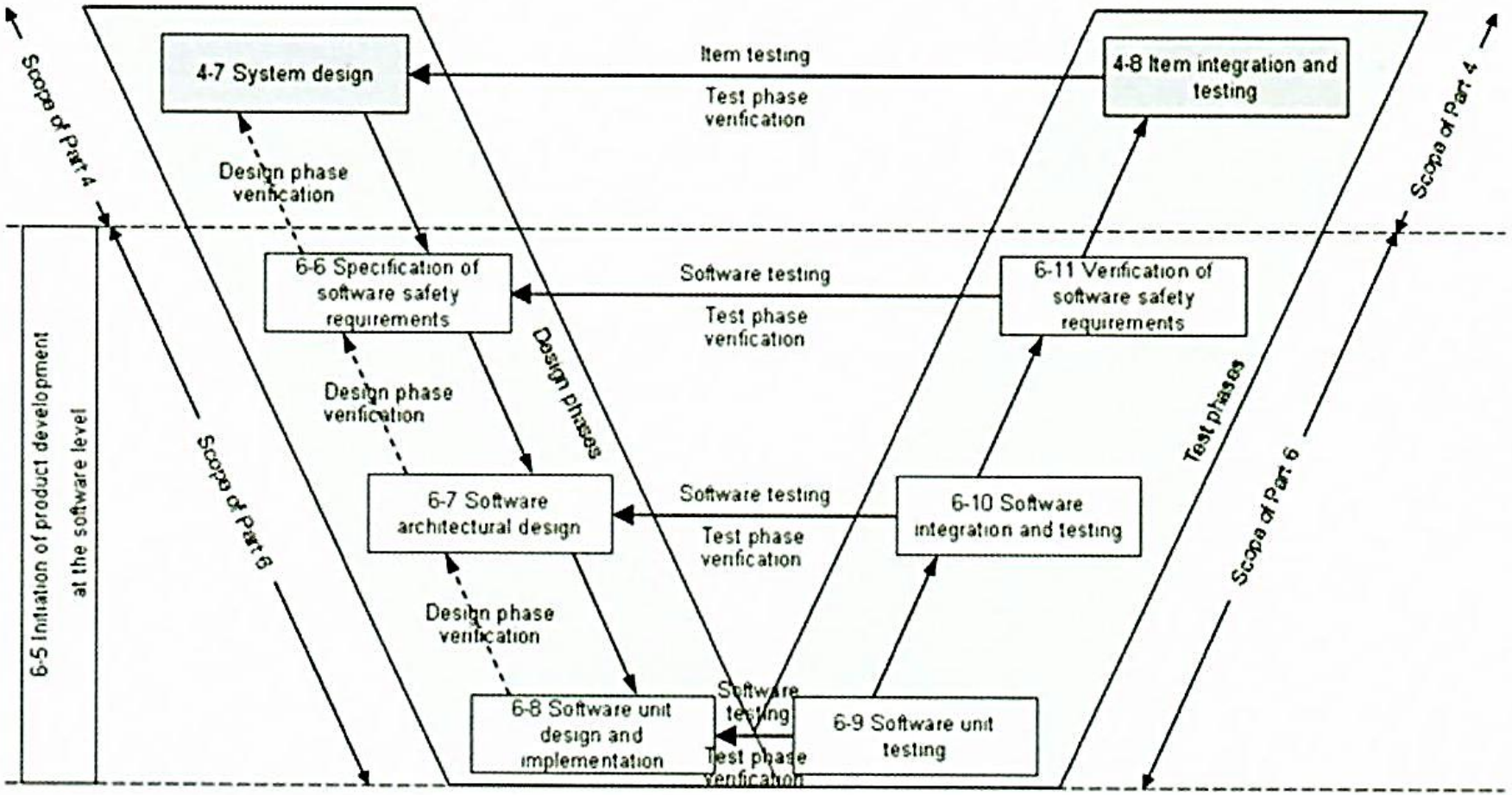
Szoftver életciklus modellek - példa



- A tervezésnek valamennyi fejlesztési folyamaton végig kell mennie:
  - követelménymeghatározás (R),
  - programtervezés (D),
  - kódolás (C),
  - integráció (I).
- Ezek valamennyi szekvenciája lehetséges, pl:
  - R-D-C-I – „V” modell alapján

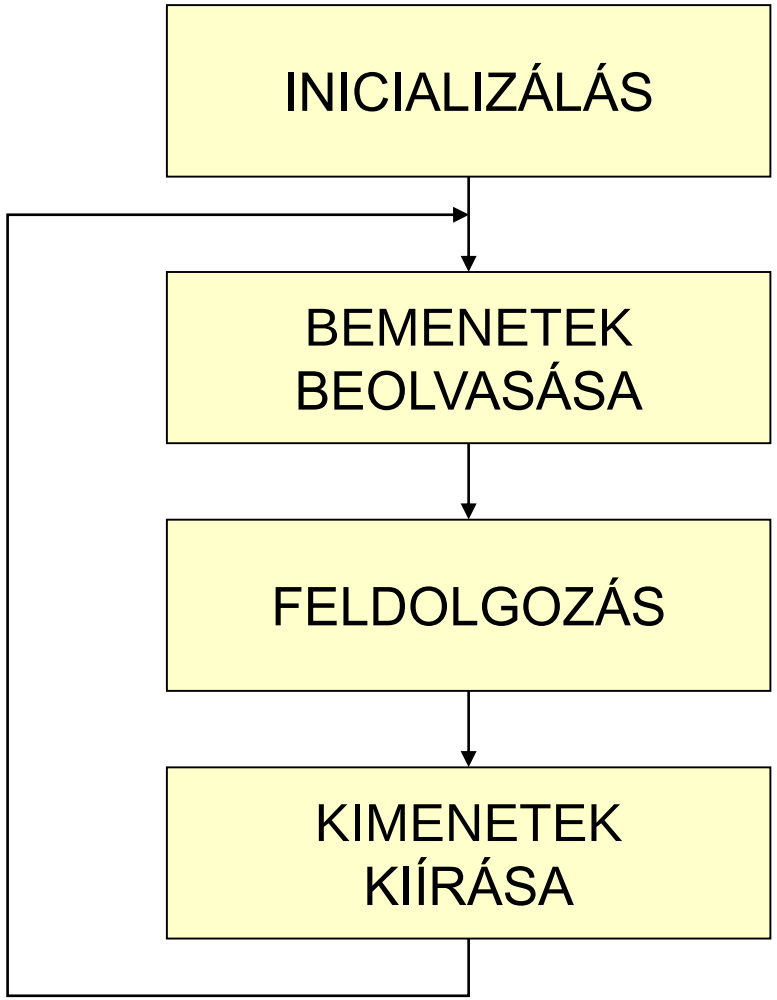
- Ákos Horváth, Standards in Avionics System Developement, (Overview on DO-178B/C), BME-MIT, link

Szoftver életciklus modellek - példa



- Autóipari szabványban definiált SW életciklus modell - ISO 26262

A folyamatirányító számítógépek

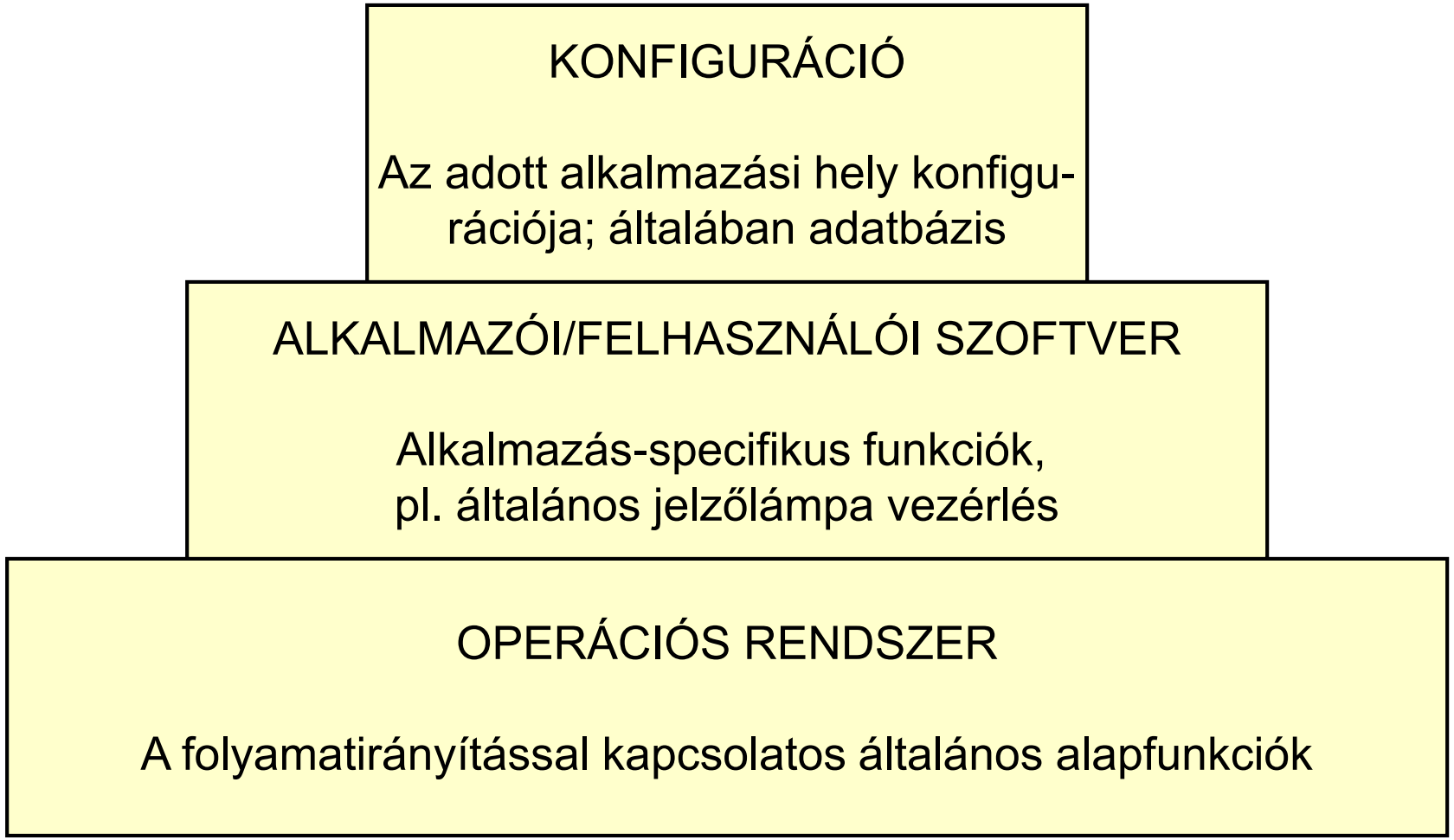


- Operációs rendszer feladatai
  - ciklikus működés,
  - Többcsatornás működés támogatása,
- Futás közbeni tesztek
  - Minden ciklusban v. ritkábban,
  - Kommunikáció tesztelése,
- Cél
  - elsődlegesen hardverhibák feltárása,
  - ritkábban szoftverhibák feltárása.
- A detektált hibákra megfelelően reagálni is kell.

Szoftverek architektúrája

- Olyan architektúra kialakítása a cél, ami lehetővé teszi a szoftverrel szemben támasztott biztonsági követelmények realizációját.
- A szoftver architektúrában az szoftver komponensek és azok kapcsolatai hierarchikus struktúrát alkotnak.
  - mind a statikus részek, pl: interfészek, adattovábbítási útvonalak,
  - mind a dinamikus részek, pl: feldolgozási szekvenciák, időzítéshez rendelt viselkedések.
- Lehet:
  - struktúrált,
  - vagy objektumorientált.

Tagolt SW felépítés



Szoftverek megbízhatósága

- A szoftver akkor működőképes – és így biztonságos –, ha a követelményeket (specifikáció) jól fogalmaztuk meg, és a megvalósítás (implementáció) is helyes.
  - Különösen fontos a dokumentáció.

- A szoftverek megbízhatósága (helyes működésének valószínűsége) **független az időtől** (amennyiben nem változtattuk meg).

- Szoftverek esetében nem beszélhetünk meghibásodásról:
  - A szoftver megbízhatóságát „csak” az eredeti, szisztematikus, specifikációs, tervezési és megvalósítási hibák csökkentik.
  - A szoftvert tároló alkatrész meghibásodása hardver-meghibásodás.
  - DE! Emberi beavatkozással egy jó szoftvert is el lehet rontani, például:
  - **Újra-bekerülési hiba**: átlagosan minden harmadik hibajavítással újabb hibát idézünk elő.

Követelmények a biztonsági szoftverekkel szemben

- Cél: **hibamentesség** (csak vicceltem).
- Szoftver-megbízhatóságot növelő módszerek:
  - Jól strukturáltság,
  - Moduláris felépítés,
  - Áttekinthetőség – szükséges az ellenőrzéshez is:
    - Modulonként kevés be/kimenet (lehetőleg 1-1) → könnyű tesztelni,
    - Jól definiált interfészek,
    - Feltétel nélküli ugrások (GOTO) kerülése,
    - Tesztelhetőség kialakítása – „tesztelés-barát tervezés”,
  - Jól dokumentáltság:
    - Funkciók leírása,
    - Interfészek leírása,
  - Nem-biztonsági részek: arra kell ügyelni, hogy a nem-biztonsági rész semmilyen módon ne legyen hatással a biztonsági részekre – **visszahatásmentesség**.

Szoftverek tervezése és implementációja

- Általánosan használható technikák az implementáció során - példák:
  - Defenzív programozás,
  - Diverz programozás,
  - Hibadetektáló kódok használata,
  - Formális módszerek,
  - Modellezés,
  - Strukturált felépítés,
  - Teljesen definiált interfészek,
  - Szoftver hiba – hatás analízis,
- Egyszerre több módszer egyidejű használata szükséges.

Szoftverek tervezése és implementációja

- **Defenzív programozás**: számítok arra, hogy a programozás során hibákat fogok elkövetni.
  - Passzív ellenőrzések: pl. ellenőrző összegek, hihetőség-vizsgálat, változók csak a saját értékkészletükben definiált értéket vehetik fel (hihetőség vizsgálat), paraméterek ellenőrzése az eljárások elején, kimeneti változók értékének ellenőrzése, bemenő változók és „lokális” változók fizikai jelentéstartalmának ellenőrzése, stb…
  - Aktív ellenőrzések: adatáramlástól független ellenőrzés, így tesztelhetők a ritkán aktív lefutási ágak is.

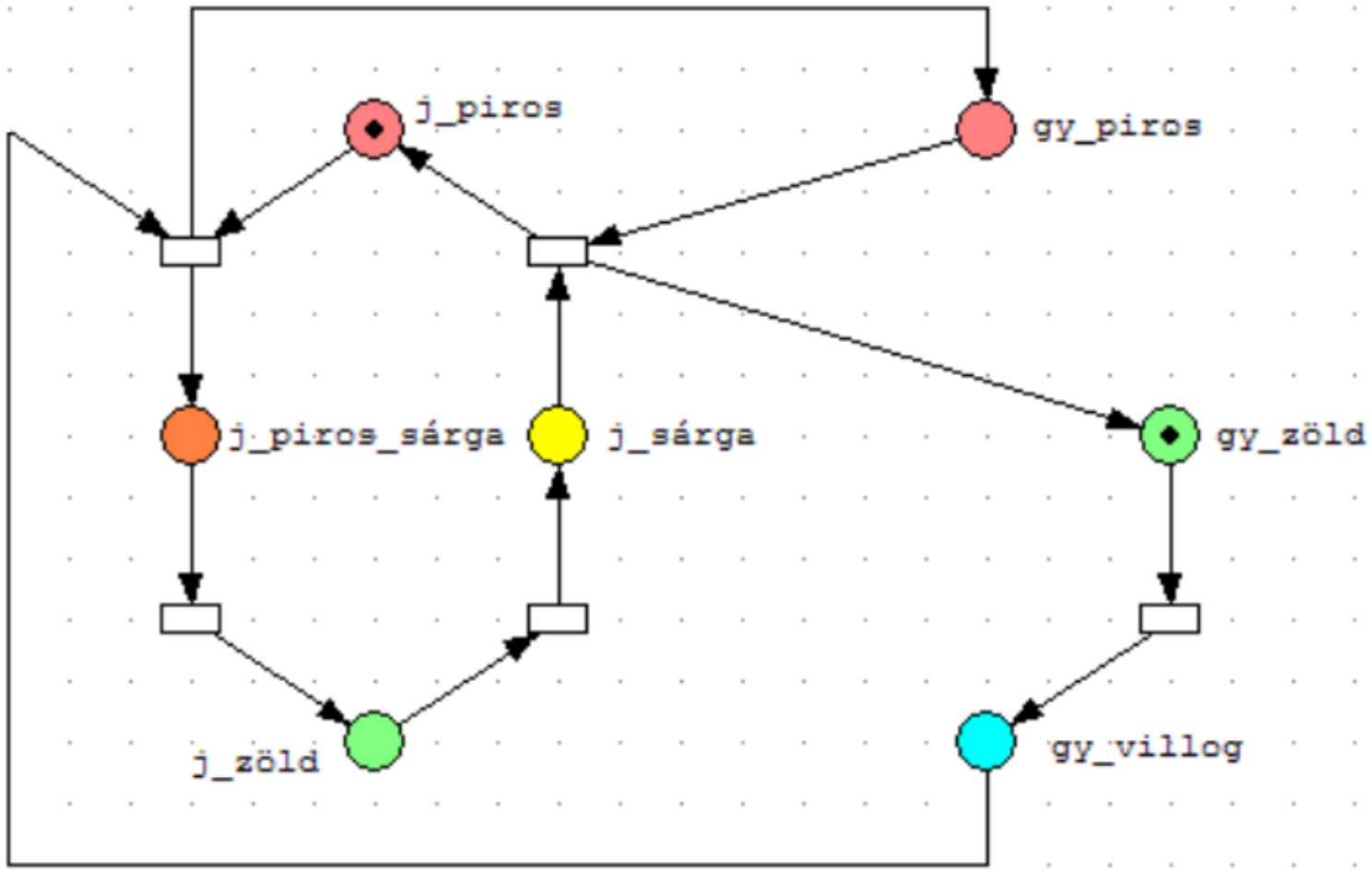
Szoftverek tervezése és implementációja

- **Diverz programozás:** a program több féle képpen, többször is (N verzióban) megírásra kerül, ahol a bemenő adatok ugyanazok, és a kimenetek értékét össze kell hasonlítani a verziók között,
  - a különböző verziók futhatnak ugyanazon vagy különböző számítógépeken egyidőben, vagy ugyanazon számítógépen felváltva,
  - az eredmény szavazás útján áll elő, ahol a **szavazás a biztonság felé téved (pl. ha N=2),**
  - a biztonsági állapot közlekedésben az energiamentes állapot:
    - vasúton: az összes jelző vörös, a vonatok csak speciális szabályok alapján közlekedhetnek,
    - közúton: sárga villogó vagy sötét közlekedési lámpák,
  - a többségi szavazólogika kimenetén a többségi akarat jut érvényre (pl. ha N=3),
  - a módszer nem kerüli el a szisztematikus hibák keletkezését, de detektálhatóvá teszi azelőtt, hogy a biztonságra hatással lenne,
  - lehet tervezési diverzitás vagy funkcionális diverzitás,
  - általában több, különböző fejlesztői és tervezői csapatot igényel – költségek, függetlenség.

Szoftverek tervezése és implementációja

- **Hibadetektáló kódok**: egy n bites információhoz egy k bites kódblokk generálódik, amellyel a hiba detektálható és javítható,
  - pl: CRC, paritás, stb…
- **Teljesen definiált interfészek**: a modul kapcsolatoknak egyértelműnek és korlátozottnak kell lenniük (pl. bemenetek száma), az interfészek csak a minimálisan szükséges paramétereket tartalmazhatják, ami az adott modul funkciójához szükség,
- **Formális módszerek**: matematikai/félmatematikai módszerek, amelyek segítségével a modellezett rendszerek teljes állapottere legenerálható, és valamennyi állapotban az adott követelmények teljesülése ellenőrizhető.

Szoftverek tervezése és implementációja



az ábra forrása: Dr. Bartha Tamás, BME-KJIT, Dr. Pataricza András, BME-MIT, Petri hálók: alapfogalmak, kiterjesztések, egyetemi előadás, Közlekedésautomatika (MSc) , link

Szoftverek tervezése és implementációja

- Programnyelv, fejlesztői környezet:
  - Olyan programozási nyelv szükséges, amely a valósidejű (real-time) működést támogatja.
  - Programozási nyelv szintje:
    - Alacsony szintű programnyelv (pl. Assembler):
      - gépközeli, rugalmas,
      - de a szoftveríró nincs rákényszerítve a strukturált programozásra.
    - Magas-szintű programnyelv:
      - a nyelv szabályai rákényszerítenek a biztonságos programírás szabályaira,
      - de nem gépközeli, ezért a folyamat-vezérlést nehezebb programozni,
      - fordítóprogram (compiler) szükséges: ennek helyes működését is igazolni kell!

Szoftverek tervezése és implementációja

- Programnyelv választásának kritériumai:
  - A programozó mennyire jártas az adott nyelven való programozásban?
  - Mennyi tapasztalat van az adott programnyelvvel?
  - Van-e elterjedt, nagy valószínűséggel hibamentes fordítóprogram?
  - Mekkora az adott programnyelv/fejlesztőkörnyezet támogatottsága (pl. szimulációs háttér)?
- Elterjedt, általánosan használható programnyelvek, pl:
  - C++, C#, ADA, JAVA – magasszintű programnyelvek,
  - Assembler – alacsony szintű programnyelv.

```ada
with Ada.Text_IO; use Ada.Text_IO;
procedure Hello is
begin
  Put_Line ("Hello, world!");
end Hello;
```

ADA

```java
public class HelloWorldApp {
    public static void main(String[] args) {
        System.out.println("Hello World!"); // Prints the string to the console.
    }
}
```

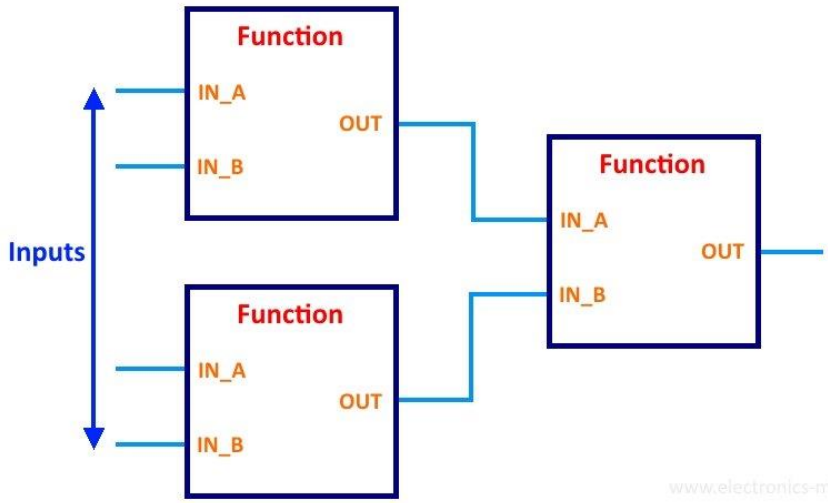
JAVA

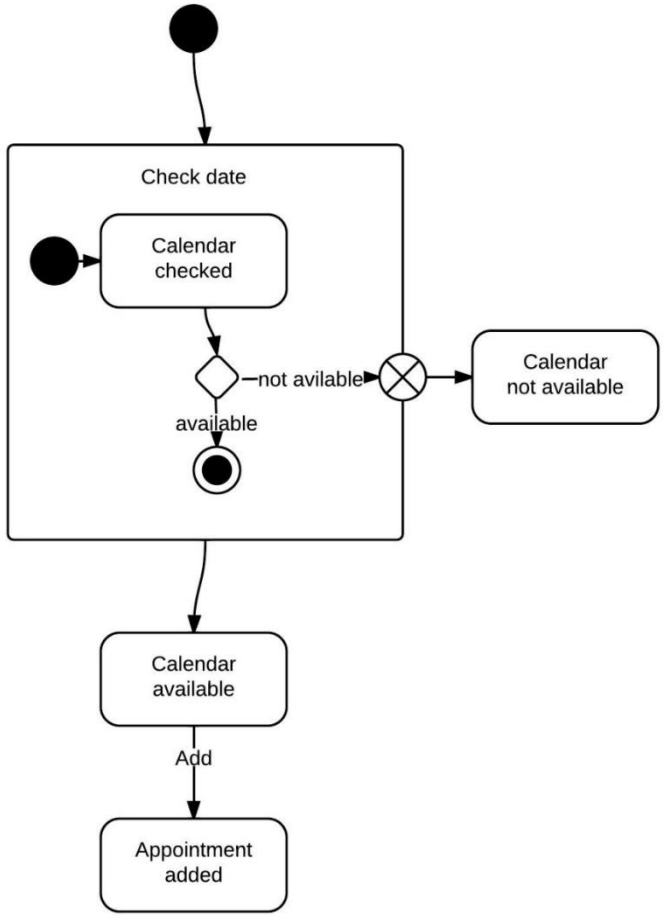
```
loop:
  jnb LEP,vege2
  clr LEP

  mov a,P4
  cpl a

  jb CSERE,felso

  setb CSERE
  swap a
  orl a,#0x0F
  mov P5,a
  sjmp vege2

  felso:
  clr CSERE
  orl a,#0x0F
  mov P5,a

  vege2:
sjmp loop

timer1_do:
  clr TF1
  mov TH1,#THREL1
  mov TL1,#TLREL1
  djnz r2,vege
  setb LEP

  mov r2,#78
vege:
ret
```

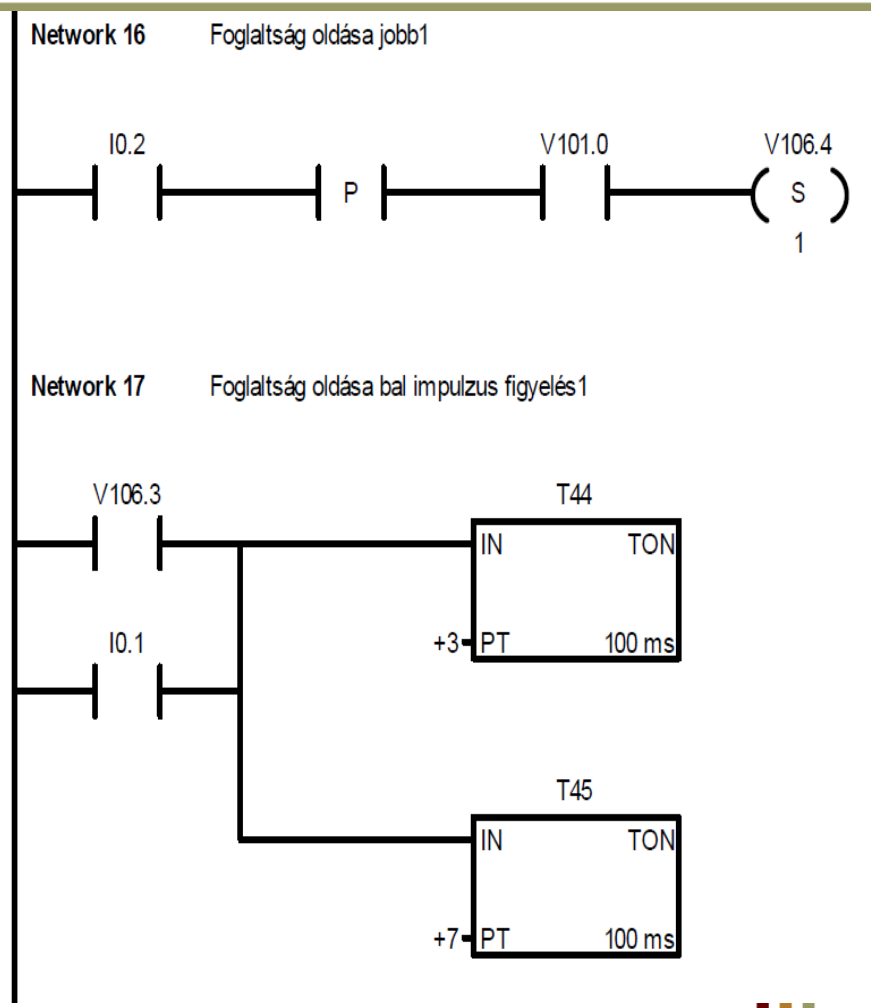
ASM

Szoftverek tervezése és implementációja

- Elterjedt, sematikus programnyelvek, pl:
  - funkcióblokk diagram (FBD),
  - állapotdiagram (State Chart),
  - létra diagram – PLC-k.







Szoftverek tervezése és implementációja

- Szoftverírási módszerek, példák:
  - Kódolási szabványok használata,
    - olyan szabályok és korlátozások egy programnyelvhez, amellyel azok a potenciális veszélyeztetések kerülhetők el, amelyek az adott nyelvhez köthetők.

Not passing code:

```
if (pi_1 > (int *)pc){
    si++;
    }
if (pi_1 == pc){
    si++;
    }
if (pi_1 >= pc){
    si++;
    }
```

Passing code:

```
if (pi_1 == pi_2){
    si++;
    }
if (pi_1 < pi_1+1){
    si++;
    }
if (pi_1 >= pc){
    si++;
    }
if (pi_1 == (int *)pc){
    si++;
    }
```

Starting conditions:

```
extern int *pi_1, *pi_2;
extern int ai_1[10],ai_2[20];
extern char *pc;
```

az ábrák forrása: Vittorio Giovara, Misra C Software Development Standard, 03/10/2008, link

Szoftverek tervezése és implementációja

- Kódolási stílusok használata,
  - szabályrendszerek a formázásra, elnevezésekre (névkonvenció), stb…,
  - több, mint ami csak „segít” olvasni a kódot,
  - segíti a kód megértését és a karbantartást főleg akkor, ha egyszerre több programozó dolgozik ugyanazon a programon.

```
if (expression)
    statement
else
    statement
```

```
if (expression)
    one_statement;
else
{
    statement_1;
    ...
    statement_n;
}
```

az ábrák forrása: C Stlye Guide, NASA Software Engineering Laboratory Series, SEL-94-003, August, 1994

Szoftver tesztelés

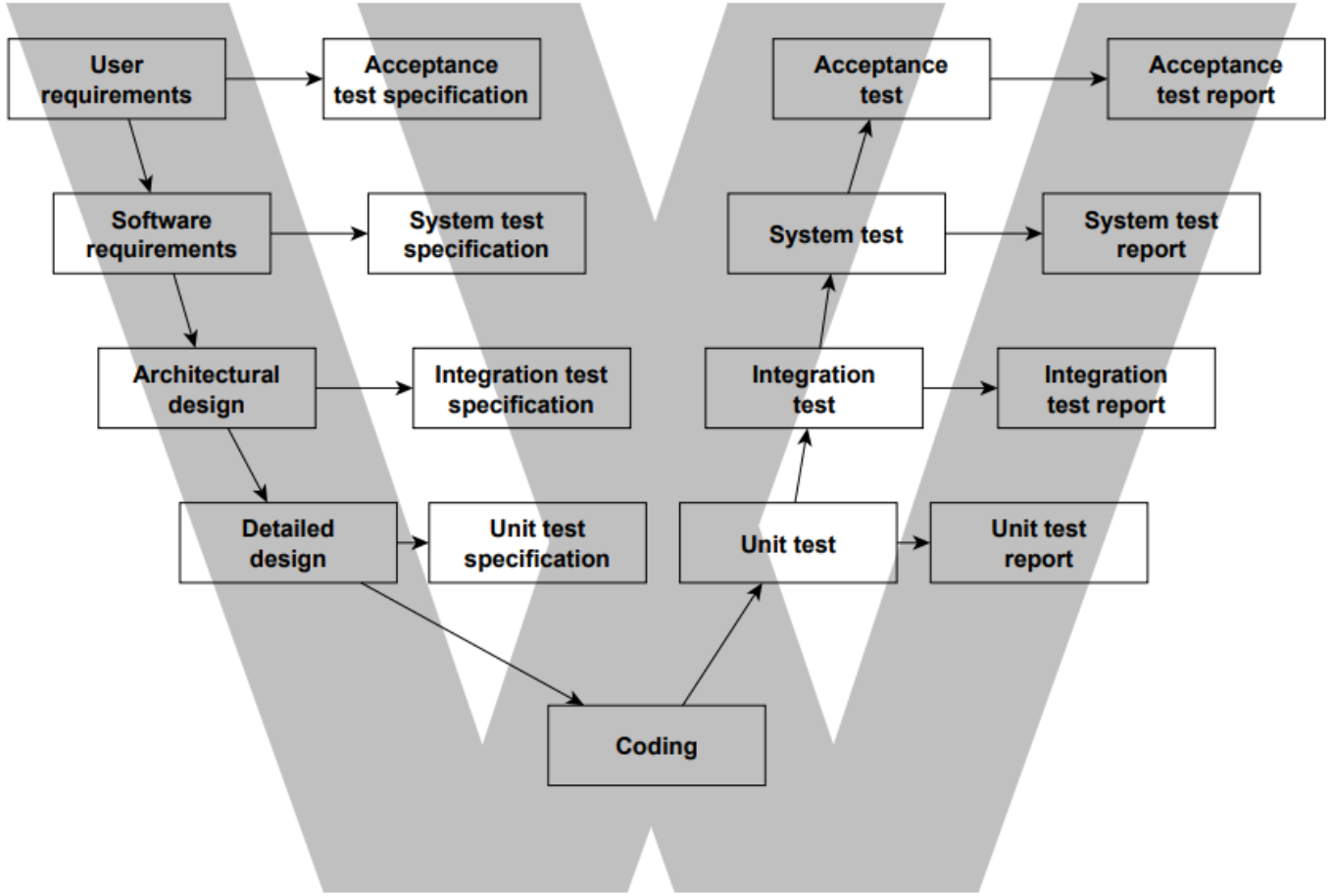
- Az eredeti hibák kiküszöbölésének leggyakoribb eljárása a **hibaeltávolítás**. Lépései:
  - tesztelés,
  - diagnózis,
  - javítás.
- **Tesztelés:**
  - Statikus: a rendszer működtetése nélküli tesztelés. Ez végrehajtható:
    - a rendszeren magán vagy.
    - a rendszer alkalmas modelljén végrehajtott vizsgálatokkal,
    - Formái:
      - statikus analízis (program-átvizsgálás, szimbolikus végrehajtás, statikus kód analízis, adatfolyam analízis stb...),
      - helyességbizonyítás (induktív bizonyítás),
      - modellezés.

Szoftver tesztelés

- Dinamikus tesztelés: a rendszer működtetése révén:
  - Tesztbemenetek kiválasztásának kritériumai:
    - A tesztelés célja szerint:
      - konformitás tesztek: a specifikáció teljesítésének vizsgálata (követelmény tesztek, interfész tesztek),
      - hibakereső tesztek: hibák feltárására,
      - hibainjektálás,- viselkedés vizsgálat,
      - erőforrás felhasználás mérése,
    - Rendszermodell szerint:
      - funkcionális teszt (black box, feketedoboz): csak a be/kimenetek viselkedését vizsgálom,
      - strukturális teszt (white/glass box, „fehér/üveg doboz”): a teljes belső szerkezet figyelembevételével tesztel,
      - kombináció (grey box, „szürke doboz”): nagy vonalakban (nem részletekbe menően) teszi láthatóvá a tesztelendő egység belső struktúráját is,
  - Tesztbemenetek generálása:
    - Determinisztikus tesztek: a tesztmintákat előre meghatározzák,
    - Valószínűségi (véletlen vagy statisztikus) tesztek: a tesztmintákat valószínűségi eloszlás alapján választják ki.

Szoftver tesztelés

- A tesztelés szintjei:
  - Modulteszt,
  - Több modul kapcsolatának tesztelése,
  - Integrációs teszt: teljes kapcsolatrendszer + kommunikáció,
  - Rendszerteszt: célhardveren való tesztelés,
  - Átvételi teszt: a megrendelő végzi,
  - Javítás utáni teszt.



Szoftver tesztelés

- Teszttterv:
  - A tesztstratégia leképzése az adott fejlesztésre:
    - tesztelési eljárások, validációs követelmények, tesztek lefutásának meghatározása, stb…

- A tesztkimenetek figyelésével dönthető el, hogy a teszt-feltételek teljesültek-e.
  - Összes tesztkimenet figyelése,
  - A tesztkimenetek kompakt reprezentációja.

- Referencia:
  - Kimeneti eredmények szimulálása,
  - Referenciarendszer („golden unit”),
  - Specifikáció,
  - Prototípus,
  - Másik implementáció.

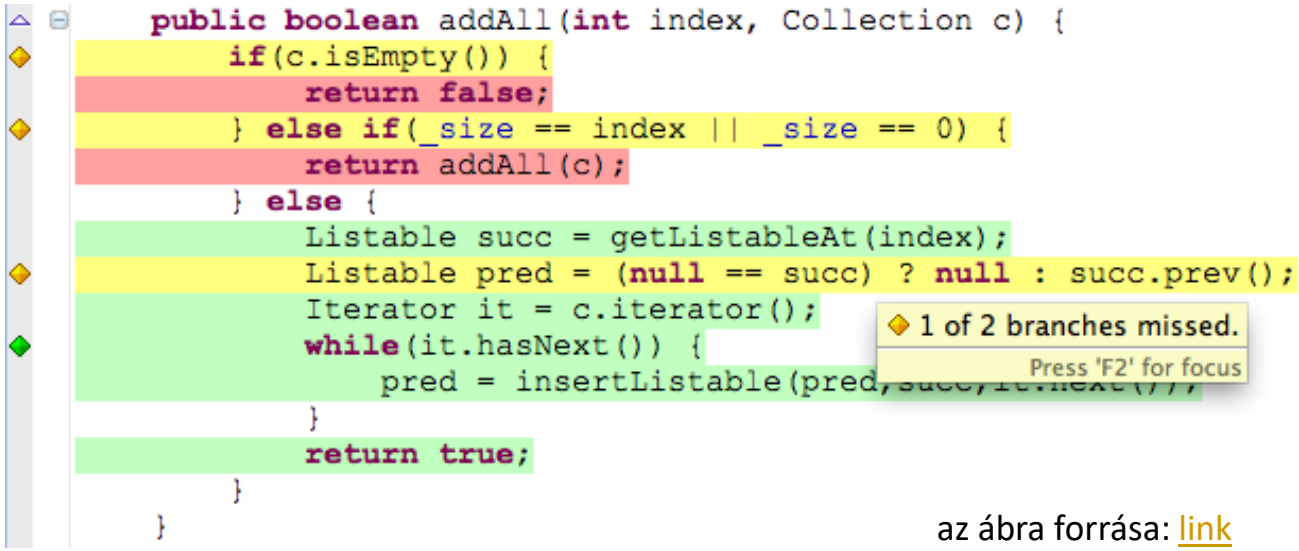
3 Tesztelési terv

3.1 Fejlesztői teszt
3.2 Prototípus (modul) teszt
3.3 Integrációs teszt
3.4 Elfogadási teszt
3.5 Terheléses teszt
3.6 Biztonsági teszt (audit)
3.7 Go live teszt
3.8 Tesztelési feladatok, teszt-esetek leírása
3.9 Tesztelési ütemterv, függőségek – tesztforgatókönyv

az ábra forrása: link

Szoftver tesztelés

- Teszt lefedettség mérés: menyire teljeskörű a teszt – test coverage, a teszt minőségére használt mérőszám, cél a 100 %, pl:
  - utasítás lefedettség: a program összes utasítása/a tesztelés során legalább egyszer végrehajtott utasítások száma,
  - elágazás lefedettség: az elágazások – döntések - (pl. if, switch, stb..): ágainak száma/bejárt ágak száma,
  - feltétel lefedettség: a tesztelt feltétel-kombinációk/összes lehetséges feltétel-kombinációk,
  - út lefedettség: tesztelt utak száma/összes út száma.



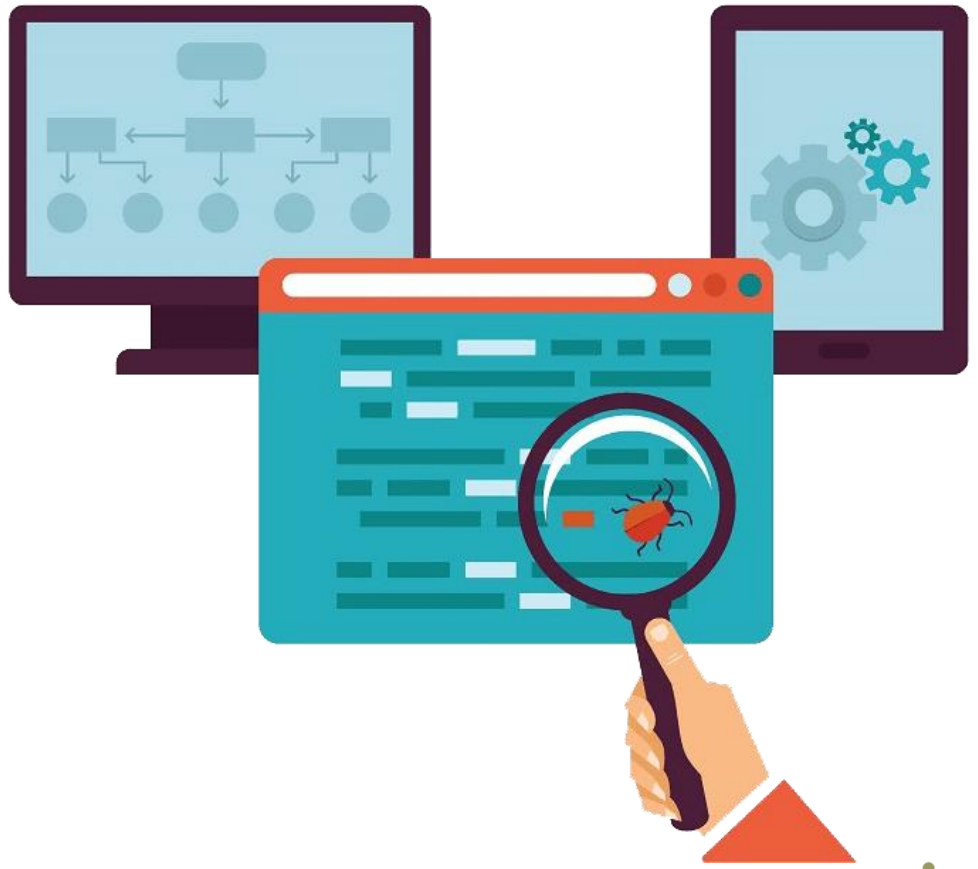
az ábra forrása: link

SW validáció

- objektív bizonyíték arról, hogy a SW megfelel a vevői követelményeknek:
  - bemenő adatok:
    - SW követelmény specifikáció,
    - teljes SW teszt specifikáció,
    - SW verifikációs terv,
    - SW validációs terv,
    - HW és SW dokumentáció a verifikációs eredményekkel,
    - biztonsági követelmény specifikáció,
  - kimenet:
    - teljes SW tesztjelentés,
    - SW validációs jelentés,
    - kiadási megjegyzések.

SW validáció

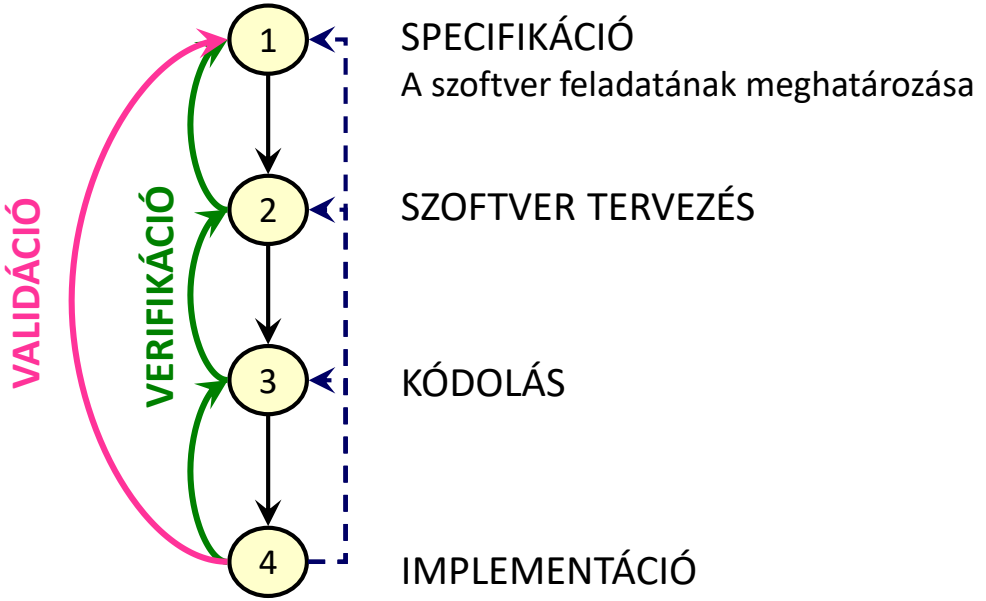
- SW validációs jelentés:
  - azonosított SW konfiguráció,
  - a szimulációs modellek azonosítása,
  - a teljes SW tesztjelentés megfelelőségének „kikiáltása”,
  - a talált eltérések listája,
  - az eltérések elemzése (kockázata),
  - korlátozások, melyeket az eltérések okoznak,
  - következtetések arról, hogy a SW megfelel az alkalmazásnak figyelembe véve az alkalmazási feltételeket és az esetleges korlátozásokat.



az ábra forrása: link

Összefoglalás

- Cél: a szisztematikus hibák keletkezésének elkerülése, a hibák mielőbbi felfedése és javítása:
  - életciklus modellek használata,
    - eljárások, módszerek a teljes életciklus fázisban a specifikációtól kezdve a „karbantartásig”,
    - módszerek a tervezésre, implementációra, iparági megközelítések használatával,
    - a tesztelés fontossága, módszerei.
  - számítok arra, hogy a kódom nem lesz hibamentes, megvalósításban nagy szerepe lesz a HW elemeknek és a rendszer architektúrájának is (lásd következő előadások).



Köszönöm a figyelmet!

Biztonsági folyamatirányító rendszerek szoftvere

Lövétei István Ferenc
(lovetei.istvan@mail.bme.hu)
Dr. Sághi Balázs