

# Szoftvertechnológia

2022 Március 30

## 6. előadás

Verifikáció és validáció

A szoftver költsége



**KADA ZSOLT**

STRATÉGIAI ÉS FEJLESZTÉSI  
IGAZGATÓ

GIRO ZRT.



PÁZMÁNY PÉTER KATOLIKUS EGYETEM - KIEMELT FELSŐOKTATÁSI INTÉZMÉNY

INFORMÁCIÓS TECHNOLÓGIAI ÉS BIONIKAI KAR - KUTATÓ KAR

# Tartalom

1

A SZOFTVER VERIFIKÁCIÓJA ÉS VALIDÁCIÓJA

2

A VERIFIKÁCIÓ ÉS VALIDÁCIÓ TERVEZÉSE

3

A SZOFTVER KÖLTSÉGEI



PÁZMÁNY PÉTER KATOLIKUS EGYETEM - KIEMELT FELSŐOKTATÁSI INTÉZMÉNY

INFORMÁCIÓS TECHNOLÓGIAI ÉS BIONIKAI KAR - KUTATÓ KAR

# A szoftver verifikációja és validációja

## Verifikáció:

- Annak ellenőrzése, hogy valóban a megfelelő terméket készítjük el, vagyis, hogy a szoftver megfelel a specifikációnak.  
(„*The product was built right*”)

## Validáció:

- Annak bizonyítása, hogy a terméket jól készítjük el, vagyis hogy a szoftver valóban a megrendelő elvárásainak megfelelően működik (*esetleg a specifikációval ellentétesen*).  
(„*The right product was built*”)
- A szoftvernek azt kell megvalósítania, amit a felhasználó valóban elvár tőle.



# A verifikáció és validáció folyamata (V&V)



PÁZMÁNY PÉTER KATOLIKUS EGYETEM - KIEMELT FELSŐOKTATÁSI INTÉZMÉNY  
INFORMÁCIÓS TECHNOLÓGIAI ÉS BIONIKAI KAR - KUTATÓ KAR



# A verifikáció és validáció folyamata (V&V)

A verifikáció és validáció (V&V) folyamata a szoftver teljes életciklusára kiterjed, a szoftverfolyamat minden fázisában szerepet kap.

## Alapvető céljai:

- ✓ Felfedni a rendszerben (esetleg már a tervek szintjén) rejlő hibákat,
- ✓ Meggyőződni arról, hogy a rendszer egy-egy konkrét működési szituációban használhatóan működik.
- ✓ Meggyőződni arról, hogy a rendszer a megrendelő igényeinek megfelelően készül/készült el.



# A verifikáció és validáció folyamata (V&V)

## Statikus és dinamikus verifikáció

A V&V folyamatban kétféle technika alkalmazható:

- **Szoftverátvizsgálás (*inspekció*)**

A rendszer reprezentációjának elemzése (*Követelményspecifikáció, tervek, grafikus ábrázolások, forráskód*). A forráskód elemzése automatizálható.

(*Statikus verifikáció*)

- **Szoftvertesztelés**

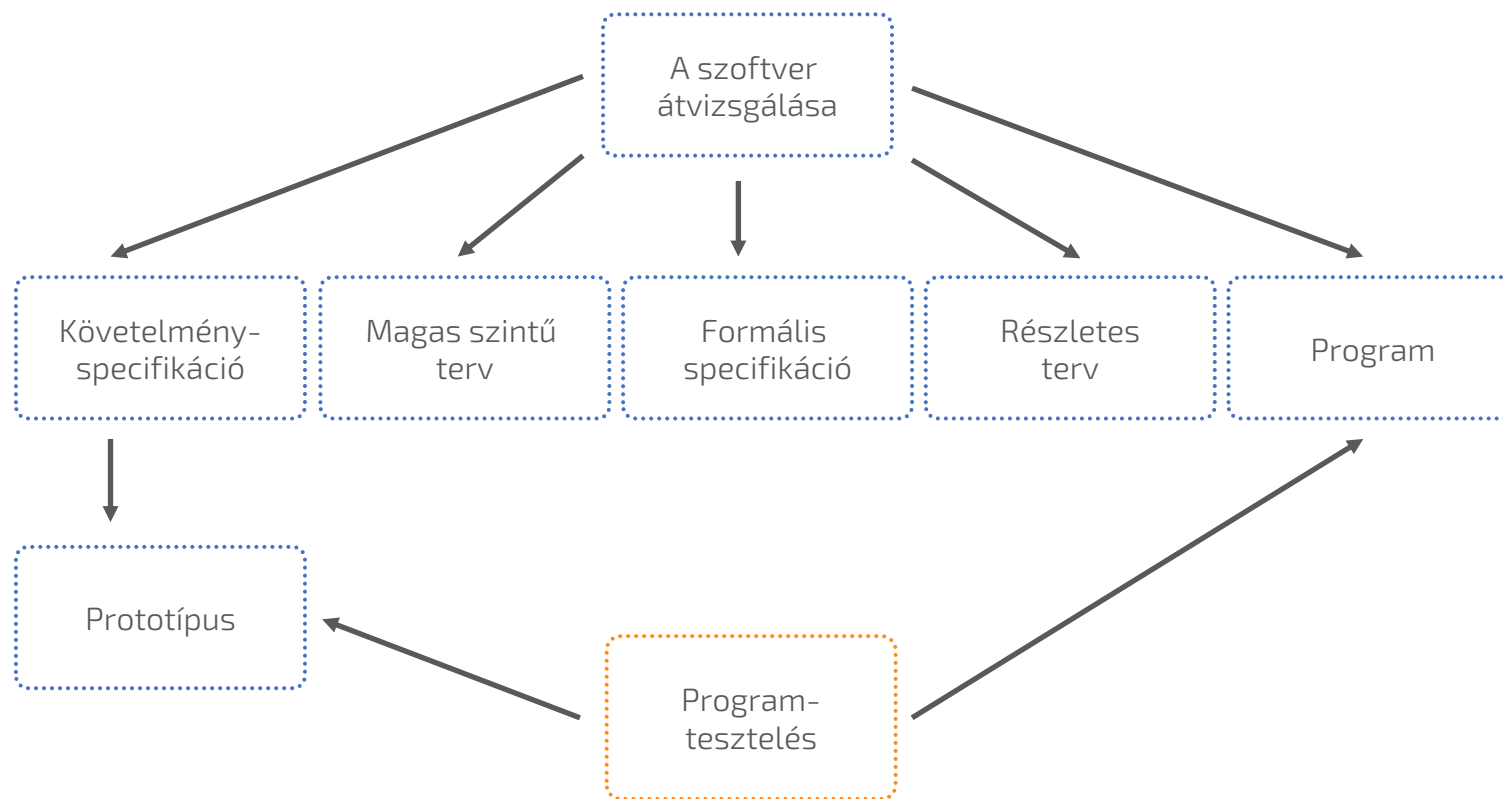
A szoftver implementációjának tesztadatokkal való futtatása és a viselkedés megfigyelése

(*Dinamikus verifikáció*)



# A verifikáció és validáció folyamata (V&V)

## Statikus és dinamikus V&V



# Programtesztelés



PÁZMÁNY PÉTER KATOLIKUS EGYETEM - KIEMELT FELSŐOKTATÁSI INTÉZMÉNY

INFORMÁCIÓS TECHNOLÓGIAI ÉS BIONIKAI KAR - KUTATÓ KAR

# Programtesztelés

- Még ma is a legelterjedtebb V&V technika (*bár sokak szemében a szoftverfolyamat végén helyezkedik el*).
- A hiba meglétét kell felfedeznie, nem a hiba hiányát.
- Az a sikeres teszt, amely legalább egy hibát felfedez.
- Az egyetlen módszer a nem-funkcionális követelmények validálására.
- A statikus verifikációval (*szemlék*) együtt célszerű alkalmazni.



# Programtesztelés

## A tesztek típusai

### Hiányosságok tesztelése

- Feladata a rendszer hibáinak és hiányosságainak felfedése.
- Fajtái:
  - **Komponensteszték:** Fekete doboz, ekvivalenciaosztályok, struktúrateszt, útvonalteszt
  - **Integrációs tesztek:** „fentről lefelé/lentről felfelé”, interfészteszt, stresszteszt
  - **Objektumorientált tesztelés**
- Például egy interaktív rendszer esetén tesztelni kell:
  - A menükön elérhető összes funkciót,
  - Egyazon menüponton elérhető valamennyi rendszerfunkciót,
  - A felhasználói inputok által használt összes függvényt helyes és helytelen bemeneti adatokkal egyaránt.

### Statisztikai tesztelés

- A rendszer teljesítményének és megbízhatóságának tesztelése valós helyzetekben (*valós felhasználói inputtal és gyakorisággal*).





# Programtesztelés

## A verifikáció és validáció céljai

- A V&V célja: megbizonyosodni arról, hogy a szoftverrendszer **megfelel a céljának**.  
(Vagyis nem arról, hogy hibamentes!)
- Az elfogadás szintje különböző célú rendszereknél különböző. Ezt befolyásolja:
  - A szoftver funkciója  
(biztonsági rendszer  $\longleftrightarrow$  prototípus)
  - A felhasználó elvárásai  
(olcsó szoftver – több hiba)
  - Piaci környezet  
(árak, versenytársak)
  - Kritikus rendszerek  
(ne okozzon tragikus eseményeket)



# Programtesztelés

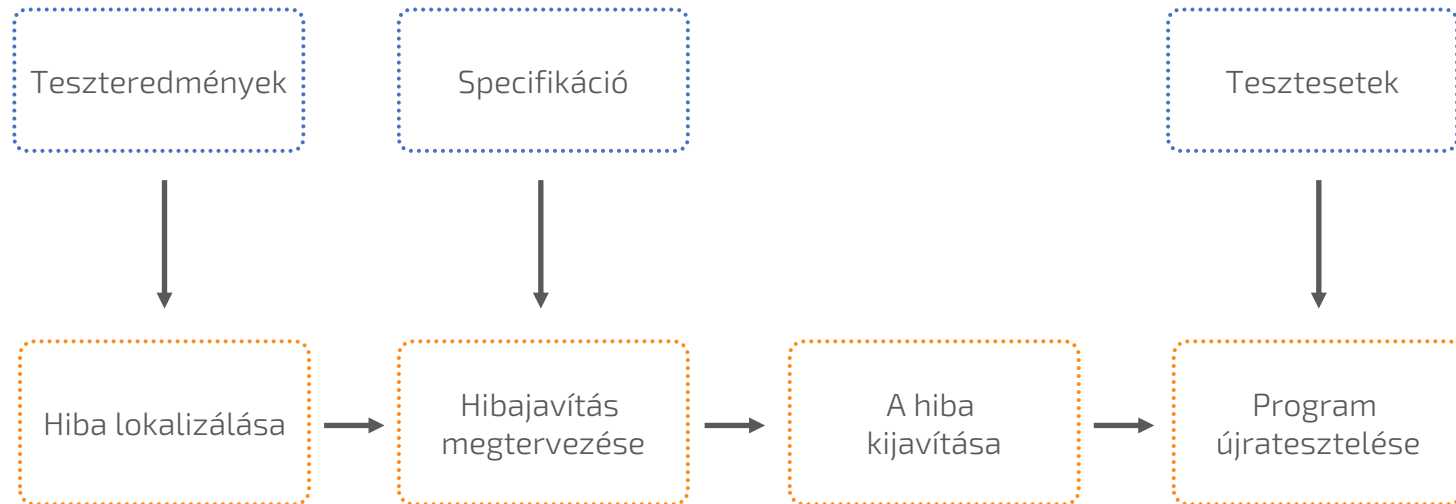
## Tesztelés és belövés

- A hiányosságok tesztelése és a belövés különböző folyamatok:
  - A verifikáció és validáció feladata a hibák, hiányosságok létezésének felfedezése.
  - A belövés ezen hibák helyének lokalizálása és kijavítása.
- A **belövés** a program viselkedésére vonatkozó feltételezések felállításával kezdődik, majd ezen feltételezések vizsgálatával próbálja megtalálni a hibákat.
- A belövés során felfedezett hibák javítása után újra kell tesztelni a programot.



# Programtesztelés

## A belövés folyamata



# Tartalom

1

A SZOFTVER VERIFIKÁCIÓJA ÉS VALIDÁCIÓJA

2

A VERIFIKÁCIÓ ÉS VALIDÁCIÓ TERVEZÉSE

3

A SZOFTVER KÖLTSÉGEI



PÁZMÁNY PÉTER KATOLIKUS EGYETEM - KIEMELT FELSŐOKTATÁSI INTÉZMÉNY

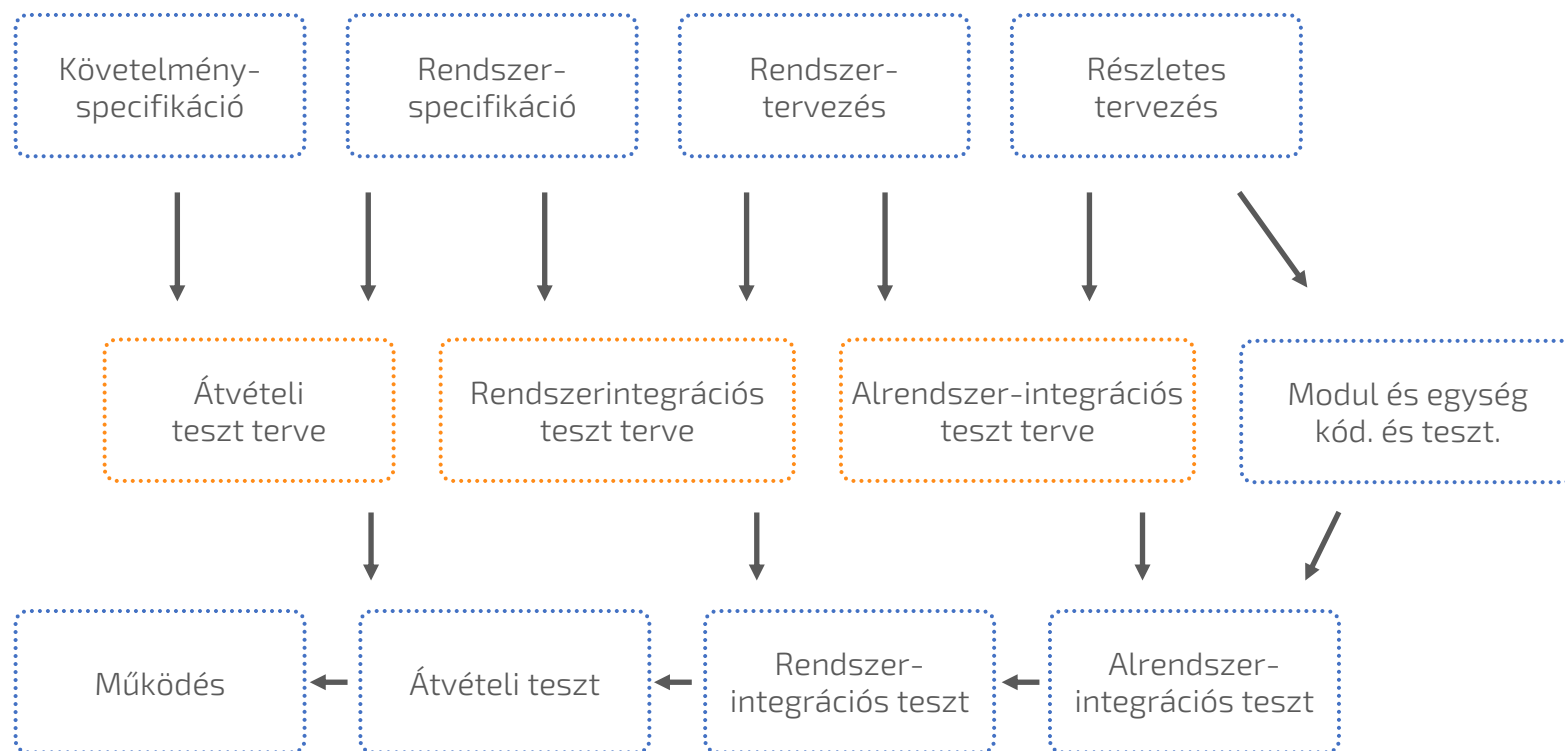
INFORMÁCIÓS TECHNOLÓGIAI ÉS BIONIKAI KAR - KUTATÓ KAR

# A verifikáció és validáció tervezése

- Alapos tervezésre van szükség, hogy a legtöbb eredményt kapjuk az egyébként igen költséges tesztelésből és felülvizsgálatból.
- A V&V tervezését a fejlesztési folyamat elején meg kell kezdeni.
- A tervnek meg kell határoznia az arányokat a statikus verifikáció és a tesztelés között.
- A teszttervezésre a nagyobb cégeknél általános szabványokat, szabályokat dolgoznak ki. Ennek alapján kell megtervezni és végrehajtani a termék konkrét tesztelését, és a tesztek dokumentálását.



## A verifikáció és validáció tervezése





# A verifikáció és validáció tervezése

## A szoftvertesztterv struktúrája

### A tesztelési folyamat

- A fő tesztfázisok leírása.

### A követelmények nyomonkövethetősége

- Minden követelményt külön kell tesztelni.

### A tesztelt elemek

- A tesztelendő szoftvertermékek listája.

### A tesztelés ütemezése

- A szoftverfejlesztési projekt részeként.

### A tesztek dokumentálása

- A tesztelés utólagos ellenőrzésére (*minőségbiztosítás*).

### A tesztek hardver- és szoftverkövetelményei

- A teszteléshez szükséges erőforrások.

### Megszorítások

- A tesztelést gátló tényezők.



# A verifikáció és validáció tervezése

## A szoftver átvizsgálása

- A szoftverátvizsgálás célja a hiányosságok felderítése, a költséges tesztelés helyett a hibák kb. 60%-a felfedhető az átvizsgálás során.
- A fejlesztési folyamat kezdetétől alkalmazható a dokumentumok *(követelmények, tervek)* átvizsgálásával.
- Egy átvizsgálás során több hiányosság felfedezhető, amíg egy teszt többnyire egy hibát fed fel. *(Legalábbis ha egy hibát detektál, a tesztelést abba kell hagyni, és a hiba kijavítását követően újból előlről kell kezdeni.)*
- A tapasztalt vizsgálók *(inspektorok)* már ismerik és könnyen megtalálják a típushibákat.



# A verifikáció és validáció tervezése

## Átvizsgálás és tesztelés

- Az inspekció és a tesztelés nem helyettesítik egymást, de a korai fázistól rendszeresen végzett átvizsgálás sok költséges tesztelést előzhet meg.
- Mindkettőt alkalmazni kell a V&V folyamatban.
- Az inspekció alkalmas eszköz arra, hogy ellenőrizze, megfelel-e a program a specifikációnak.
- A nem-funkcionális rendszerkövetelmények vizsgálatára azonban a felülvizsgálat nem használható.



# Programátvizsgálás



PÁZMÁNY PÉTER KATOLIKUS EGYETEM - KIEMELT FELSŐOKTATÁSI INTÉZMÉNY

INFORMÁCIÓS TECHNOLÓGIAI ÉS BIONIKAI KAR - KUTATÓ KAR

# Programátvizsgálás

- A dokumentumok átvizsgálásának formalizált eszköze:  
Tapasztalt szakemberek nézik át a dokumentumokat és a kódot ellenőrző lista alapján.
- Célja a hiányosságok felderítése a tervekben, dokumentumokban és a forráskódban  
(*logikai hibák, kezdőérték nélküli változók, szabványoknak való meg nem felelés, stb.*)
- Az átvizsgálást követően a programozó módosítja a programot, az új verziót nem kell feltétlenül újravizsgálni (*de tesztelni igen!*).



# Automatizált statikus elemzés



PÁZMÁNY PÉTER KATOLIKUS EGYETEM - KIEMELT FELSŐOKTATÁSI INTÉZMÉNY  
INFORMÁCIÓS TECHNOLÓGIAI ÉS BIONIKAI KAR - KUTATÓ KAR



# Automatizált statikus elemzés

- A statikus elemzők a forráskódot vizsgáló szoftvereszközök.
- Nem futtatják a programot, hanem elemzik a program szövegét.
- Fázisai:
  - A vezérlés folyamatának elemzése
  - Az adatok használatának elemzése
  - Interfész-elemzés
  - Az információáramlás elemzése
  - A végrehajtási útvonalak elemzése



# Automatizált statikus elemzés

## A statikus elemzők használata

- A statikus elemzés különösen az olyan nyelveknél hasznos, mint a C, amelyek nem tartalmaznak szigorú szabályokat a típusokra, így a fordító sok hibát nem vesz észre.
- A Unix és a Linux tartalmazza a LINT statikus elemzőt, amely kimutatja az iniciálatlan változókat, elérhetetlen kódrészleteket, stb. Az Eclipse FindBugs elemzője sok hibát (pl. *null-pointer*, *bázis nélküli rekurzív függvény*, stb.) tud felfedezni.
- A Java nyelvből sok olyan tulajdonság hiányzik, amely hibaforrás lehet (*nincs goto*, *iniciálni kell a változókat*, *a tárkezelés automatikus*, stb.).



# Cleanroom szoftverfejlesztés



PÁZMÁNY PÉTER KATOLIKUS EGYETEM - KIEMELT FELSŐOKTATÁSI INTÉZMÉNY  
INFORMÁCIÓS TECHNOLÓGIAI ÉS BIONIKAI KAR - KUTATÓ KAR

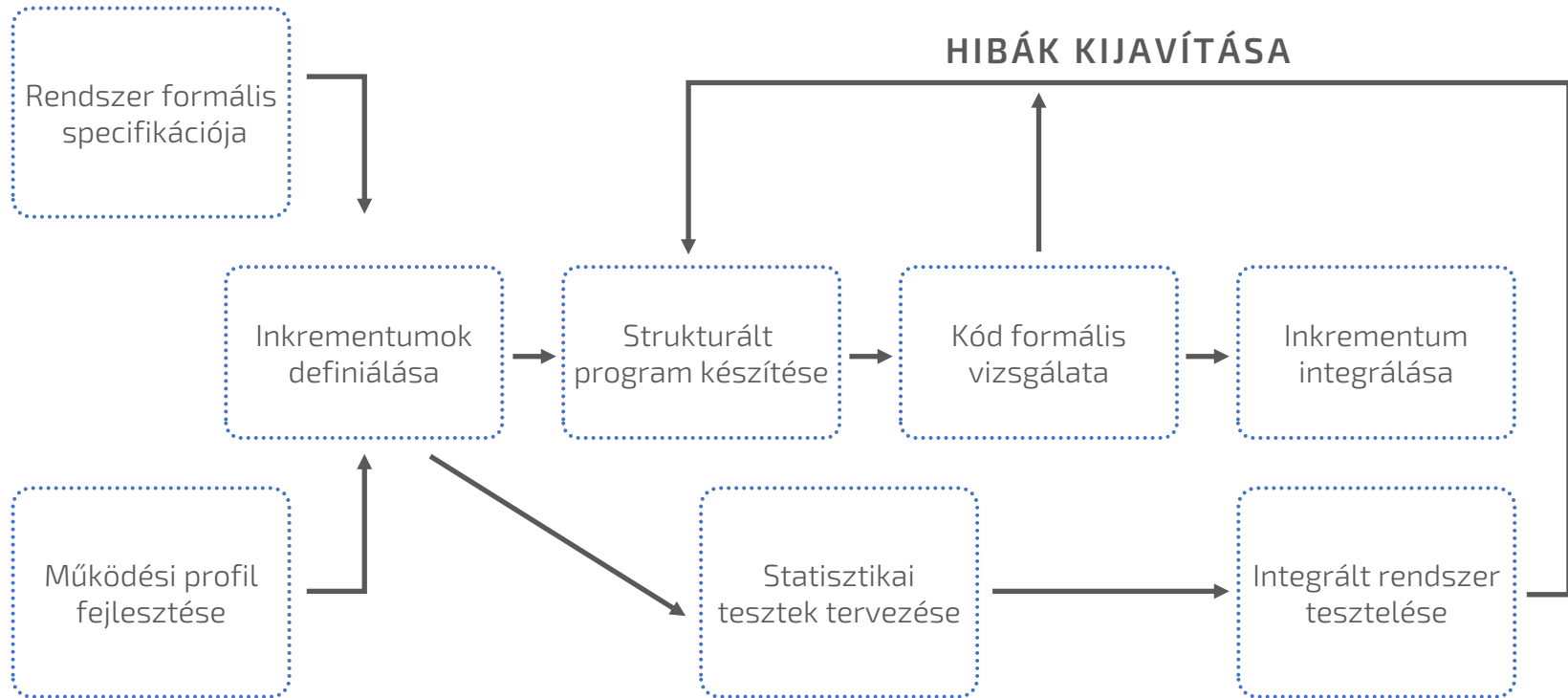
# Cleanroom szoftverfejlesztés

- A szoftverhibák elkerülését, nem pedig megtalálását és kijavítását célzó szigorú átvizsgálási folyamat. *(A név a félvezetőgyártásból származik)*
- A rendszer komponenseinek tesztelését helyettesíti átvizsgálásokkal, megfelelnek-e a specifikációnak.
- Inkrementális fejlesztési módszer, először a kritikus inkrementumokat szállítja le.
- A Cleanroom jellemzői:
  - Formális specifikáció *(állapotátmenet-modell, strukturált programozás, csak néhány vezérlési és adatabsztrakciós konstrukció használható)*
  - Inkrementális fejlesztés
  - Statikus verifikáció *(szigorú átvizsgálások)*
  - A rendszer statisztikai tesztelése



# Cleanroom szoftverfejlesztés

## A Cleanroom folyamat



# Cleanroom szoftverfejlesztés

## A Cleanroom folyamat szervezete

- **Specifikációs csapat:**

A rendszerspecifikáció kidolgozását és karbantartását végzi.

- **Fejlesztőcsapat:**

A fejlesztést és verifikálást végzi. A szoftvert nem futtatja.

- **Hitelesítő csapat:**

A formális specifikáción alapuló statisztikai tesztekkel dolgozza ki *(a fejlesztéssel párhuzamosan)* és futtatja le.



# Programátvizsgálás

## Szoftver tanúsítás

### Tanúsítás:

Egy független szervezet vagy hatóság igazolja, hogy a szoftver megfelel egy bizonyos célnak, vagy szabványnak.

- Célja: leginkább a bizalom erősítése, de gyakran a bizonytalan, hibás szoftverek kitiltása egy alkalmazási körből  
(pl. NAV (APEH, VPOP), stb. tanúsítványok)
- Egy komponens tanúsításának célja lehet, hogy a komponens felhasználóit meggyőzze egy szabványnak való megfeleléséről,

### DE!

a tanúsítvánnyal rendelkező komponens nem jelent garanciát a rendszer megfelelésére!

- A minőségi tanúsítás legbiztosabb módja, ha a szoftverfejlesztési folyamatot tanúsítják - CMM



# Összefoglalás

- A verifikáció és a validáció két különböző tevékenység:
  - A verifikáció a specifikációnak való megfelelést vizsgálja,
  - A validáció bizonyítja, hogy a rendszer megfelel a felhasználó igényeinek.
- A tesztelési folyamatot tervezni kell.
- A statikus verifikációs technikák a szoftver vizsgálatát és analizálását is tartalmazzák.
- A felülvizsgálat hatékony eszköz a hibák felderítésére.
- A statikus elemzőeszközök a forráskódban olyan rejtett hibákat keresnek, mint a nem inicialt változók, vagy nem használt kódrészletek.





# Tartalom

1

A SZOFTVER VERIFIKÁCIÓJA ÉS VALIDÁCIÓJA

2

A VERIFIKÁCIÓ ÉS VALIDÁCIÓ TERVEZÉSE

3

A SZOFTVER KÖLTSÉGEI



PÁZMÁNY PÉTER KATOLIKUS EGYETEM - KIEMELT FELSŐOKTATÁSI INTÉZMÉNY

INFORMÁCIÓS TECHNOLÓGIAI ÉS BIONIKAI KAR - KUTATÓ KAR

# A szoftver költségei

## A szoftver költségbecslésének alapvető kérdései



**Mekkora munkát igényel egy feladat elvégzése?**

**Mennyi időbe kerül a feladat végrehajtása?**

**Mennyi a tevékenység összes költsége?**

- A költségbecslés és projektütemezés folyamatos projektvezetési tevékenység.
- A szoftverfejlesztési projekt költségelei:
  - A hardver és szoftver költségei a karbantartással együtt,
  - Utazási és képzési költségek,
  - Munkaköltségek (*bér, közteher, helység, kisegítő munkák, kommunikáció, rekreáció...*)



# A szoftver költségei

## A szoftver költsége és ára

- A költség és az ár között nincs egyszerű arányosság. Befolyásolják:
  - Piaci lehetőségek,
  - A költségbecslés bizonytalanságai,
  - A szerződéses feltételek (*tulajdonjog*),
  - A követelmények változékonysága,
  - A fejlesztő gazdasági helyzete.



# A szoftver költségei

## A szoftverfejlesztés termelékenységé

- Mélni kell a szoftver valamilyen jellemzőjét és osztani a fejlesztési idővel.
  - **Mennyiségi mérések:**  
(Forráskódsorok száma, utasítások száma, dokumentáció oldalszáma, stb.)
  - **Funkcionális mérések:**  
(Az időegység alatt előállított hasznos funkciók száma: Funkciópontok, objektumpontok.)
- A termelékenység összehasonlítása:
  - Alacsonyabb szintű nyelven több kódsort lehet írni, de azonos funkciót több kóddal kell megvalósítani,
  - A jó programozó ugyanazt a funkciót kevesebb kóddal készíti el, mint a „bőbeszédű” programozó,
  - Hogyan vegyük figyelembe a kommenteket?



# A szoftver költségei

## Funkciópontok

- A program jellemzőinek kombinációján alapuló, nyelvfüggetlen módszer.
- Méri az alábbi jellemzőket:
  - Külső bemenetek és kimenetek
  - Felhasználói interakciók
  - Külső interfészek
  - A rendszer által használt fájlok
- Mindegyikhez súlyozási tényezőt rendel:
  - Egyszerű külső bemenet: 3
  - Bonyolult belső állományok: 15
- A súlyozási tényezőt egy szervezeten belül, hasonló jellegű szoftverek készítése során gyűjtött statisztikák alapján finomítja.



# A szoftver költségei

## A funkciópontok számítása

- A funkciópontok (FP) alapján a kódsorok számára (LOC – Lines Of Code) lehet következtetni:
  - $LOC = AVC * FP$  ahol:
    - AVC nyelvfüggő szorzófaktor  
(200-300 az assembly és 2-40 a 4GL nyelvekre)
- A funkciópont-számítás nagyon sok szubjektív elemet tartalmaz.
- Automatikus számítása nem lehetséges, mert a specifikáció alapján kell a funkciópontokat megbecsülni.



# A szoftver költségei

## Objektumpontok

- 4GL vagy más magas szintű nyelvek esetén a funkciópontok alternatívája. Magas szintű specifikáció alapján könnyebben becsülhető.
- Az objektumpont (NTC) nem azonos az objektumok számával, hanem az alábbiakból számítható:
  - A külön megjelenítendő képernyők száma, az egyszerűtől (1), a nagyon bonyolultig (3),
  - A készítendő jelentések száma (2 – 5 – 8)
  - A 4GL kiegészítése miatt szükséges 3GL modulok száma (10)



# A szoftver költségei

## A termelékenység becslése

- Valósídejű, beültetett rendszerek:  
40 – 160 LOC / hó
- Rendszerprogramok:  
150 – 400 LOC / hó
- Kereskedelmi alkalmazások:  
200 – 800 LOC / hó
- Objektumpontban számolva a termelékenység 4 és 50 pont / hónap közötti, az eszköztámogatottságtól és a fejlesztők képességeitől függően.





# A szoftver költségei

## A termelékenység befolyásoló tényezők

### Az alkalmazási terület ismerete:

- A hatékony szoftverfejlesztéshez szükséges a szakterület ismerete.

### A folyamat minősége:

- A fejlesztési folyamat minőségének jelentős hatása van a termelékenységre.

### A projekt mérete:

- A nagyobb projekt több csoportkommunikációs és adminisztrációs tevékenységet igényel.

### Technológiai támogatás:

- Támogató technológiával (pl. CASE) a termelékenység növelhető.

### Munkakörnyezet:

- A jó munkakörülmények és légkör javítják a termelékenységet.



# A szoftver költségei

## Algoritmikus költségmodellezés (COCOMO)

- Empirikus modell, a projektek gyakorlatából gyűjtött adatokon alapul.
- Jól dokumentált, hosszú tapasztalat áll mögötte (első verzió: 1981)
- A COCOMO 2 (1995) három szintű modellt alkalmaz:
  - **Korai prototípuskészítés szintje**  
(becslés objektumpontok alapján)
  - **Korai tervezés szintje**  
(funkciópontok alapján a forráskódok számát becsli)
  - **Posztarchitektúrális szint**  
(az architektúraterv elkészülte után becsli a szoftver méretét)



# A szoftver költségei

## Korai prototípuskészítés szintje

- Prototípuskészítést és újrafelhasználást is figyelembe vesz.
- A fejlesztői produktivitást objektumpontokkal számolja és a CASE használatot is bekalkulálja.
- A formula:  $PM = (NOP * (1 - \%reuse)) / PROD$
- Ahol: PM – a munka emberhónapban, NOP – az objektumpontok száma, PROD – produktivitás

A fejlesztő gyakorlata és képessége	Nagyon kevés	Kevés	Átlagos	Sok	Nagyon magas
A CASE érettsége és lehetőségei	Nagyon kevés	Kevés	Átlagos	Sok	Nagyon magas
PROD (NOP/hónap)	4	7	13	25	50



# A szoftver költségei

## Korai tervezési szint

- A követelmények tisztázása után végezhető a becslés.
- Az alábbi képlettel számol:

$$PM = A * Méret^B * M + PM_m$$

ahol...

$M = PERS * RCPX * RUSE * PDIF * PREX * FCIL * SCED$

$PM_m = (ASLOC * (AT/100)) / ATPROD$

$A = 2,5$  a kezdeti számításban

$B = 1,1 - 1,24$  a projekt mérete, újdonsága függvényében.

$M$  = projekttenyezők:

PERS – személyi képességek,

RCPX – termékmegbízhatóság,

RUSE – szükséges újrafelhasználás,

PDIF – platform nehézségei

PREX – személyek gyakorlata,

FCIL – támogató eszközök,

SCED – ütemezés

ASLOC = automatikusan generált kódsorok,

AT = aut. rendszerkód,

ATPRO = termelékenység,



# A szoftver költségei

## Posztarchitekturális szint

- Ugyanazt a formulát alkalmazza, mint a korai tervezési becslés, de két tényezőt figyelembe vesz:
  - A követelmények változékonysága,
  - A lehetséges újrafelhasználás mértéke.
- A szükséges új kódsorok számának becslésekor statisztikai és egyéb értékeket is figyelembe vesz, mint:
  - A korábbi hasonló projektek hiánya,
  - A fejlesztés rugalmassága,
  - A csapat összetartása,
  - A folyamat fejlettsége.



*Köszönöm a figyelmet!*