Szoftver tesztelés

Bevezetés

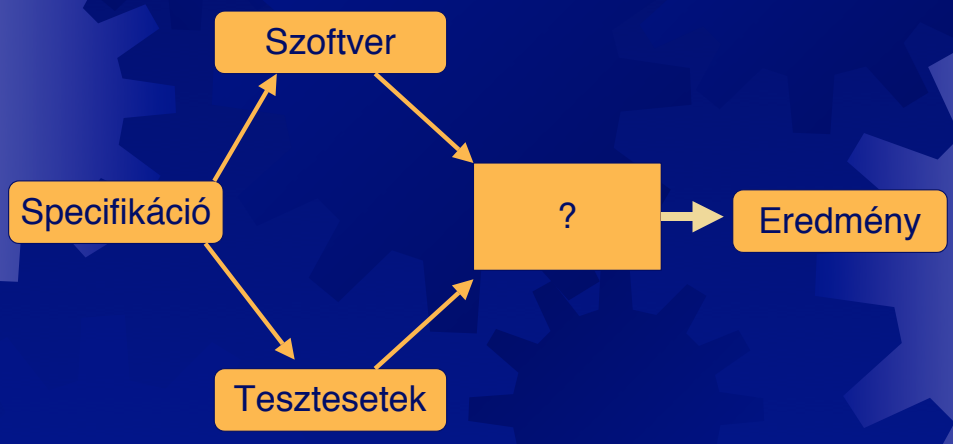
A szoftver tesztelés célja

- Szoftver-minőség biztosítása, javítása
- Szoftver hibák lokalizálása
- V & V támogatása
  - verifikáció
    - fázis transzformációs lépései helyesek
    - követelményeknek való megfelelés
  - validálás
    - termék megfelel a felhasználó elvárásainak
- Életciklus (SW folyamat) sajátosságok
  - folyamatosan az életciklusban
  - managementbeli függetlenség

A tesztelés szintjei

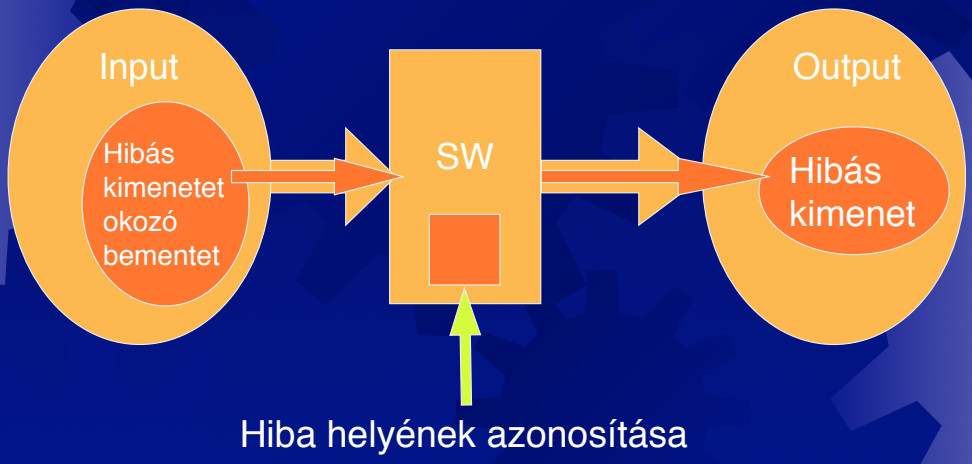
- alacsony - modul szintű
- magas - rendszer szintű
  - szolgáltatás
  - mennyiségi
  - löket terhelés
  - használhatóság
  - teljesítmény
  - konfiguráció
  - dokumentáció

A tesztelési probléma



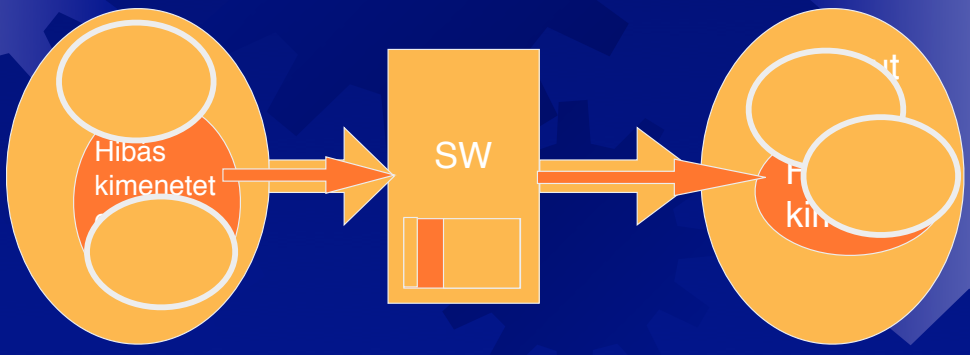
Szoftverhibák megjelenése

- SW mint input-output leképezés



Megismerési problémák

- sokszor nem lehetséges a kimerítő tesztelés
- megerősítési tévedés (pozitív tesztesetek)
- Megfelelő szemléletű megközelítés
  - A tesztelés a hibák kimutatására való és nem a jelenlétük hiányának demonstrálására



A tesztelés költsége

Tesztelés típusai

- Tesztelés végrehajtással
  - működésközben manifesztálódó hibák kiszűrése
- Tesztelés végrehajtás nélkül
  - kód, konfiguráció, dokumentáció stb. átvizsgálása, ellenőrzése

Átvizsgálás

- kevésbé formális
- 4-6 fős csoportok
  - specikáló
  - elemző (tervező)
  - megbízó
  - következő fázis képviselője
  - minőség biztosítási csoport
- lista szerinti átvizsgálás
  - kb. 2 óra
  - detektálás (nem javítás)
  - teljesítmény becslések

Részletes vizsgálatok, szemlék

- 5 formális lépés
  - áttekintés
  - előkészítés
    - hiba statisztikák
    - potenciális hibahelyek
  - szemle, vizsgálat
    - hibák megtalálása, dokumentálása
  - megoldások kidolgozása
  - ellenőrzés
    - újra szemlézés

Hiba statisztikák

- Hiba típus és súlyosság szerint rögzített előfordulási gyakoriságok
- Korábbi fejlesztések statisztikái
- Nem tipikus hibaszám eloszlás kezelése
- Veszélyes területek azonosítása
- Adott fázis hibastatisztikáinak figyelembevétele a következő fázisban

Átvizsgálások eredményei

- Átvizsgálások hibadetektáló képessége
  - összes detektált hiba 82% (IBM, 1976)
  - összes detektált hiba 70% (IBM, 1978)
  - összes detektált hiba 93% (IBM, 1986)
- Hibadetektálás költségének 90%-os csökkentése (Kapcsolórendszer, 1986)
- Hibaszám exponeciálisan csökken átvizsgálási fázisról-fázisra (JPL, 1992)

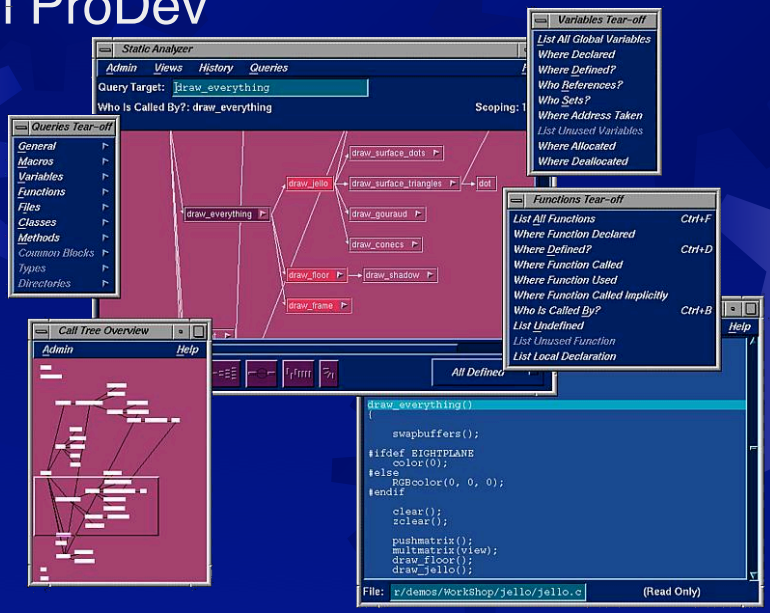
Átvizsgálási metrikák

- Hibasűrűség (hiba /KLOC)
- Hibadetektálási ráta (megtalált hiba / óra)

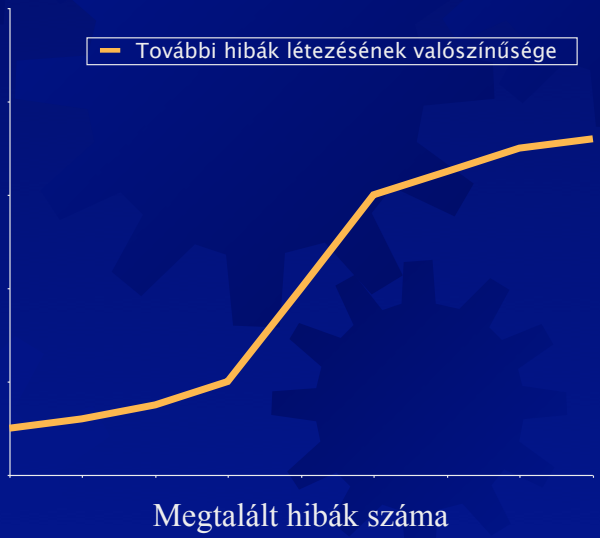
- Metrikák hibasúlyosság szerint
- Milyen következtetést lehet levonni a megtalált hibák számából?

Átvizsgálás statikus kódelemzéssel

- sgi ProDev



Tesztelés után maradó hibák



Tesztelés végrehajtással

- Viselkedés vizsgálata megválasztott bemenetekkel, rögzített környezetben
- Hibamodellek
  - hibák előfordulása és jellege?
  - fejlesztési- és tesztelési technológia függő

Tesztelési megközelítések

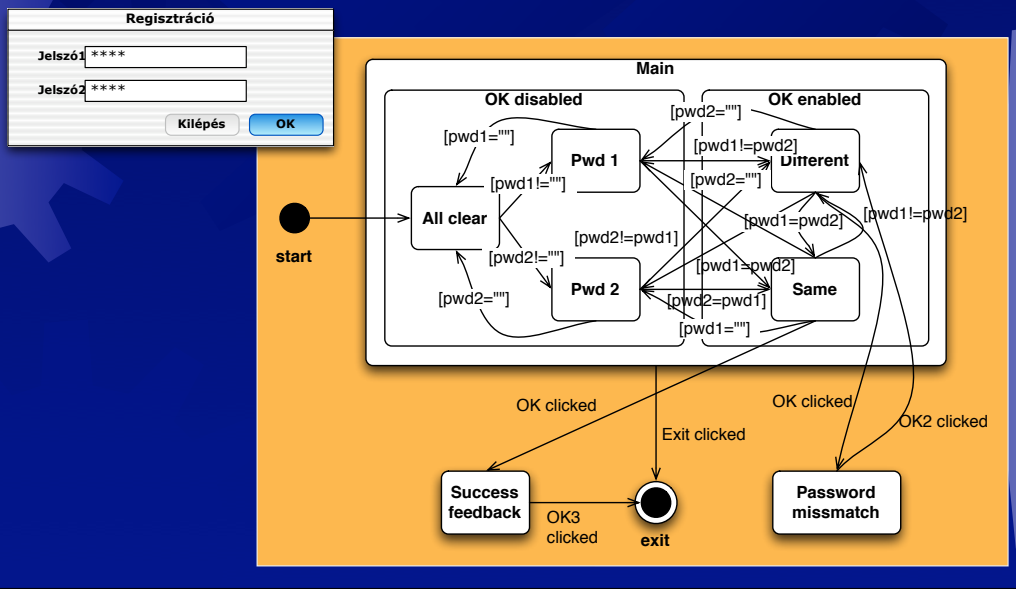
- Funkcionális tesztelés (black box)
  - specifikáció alapján
- Struktúrális tesztelés (white box)
  - működési logika alapján

Végrehajtásos tesztelés problémái

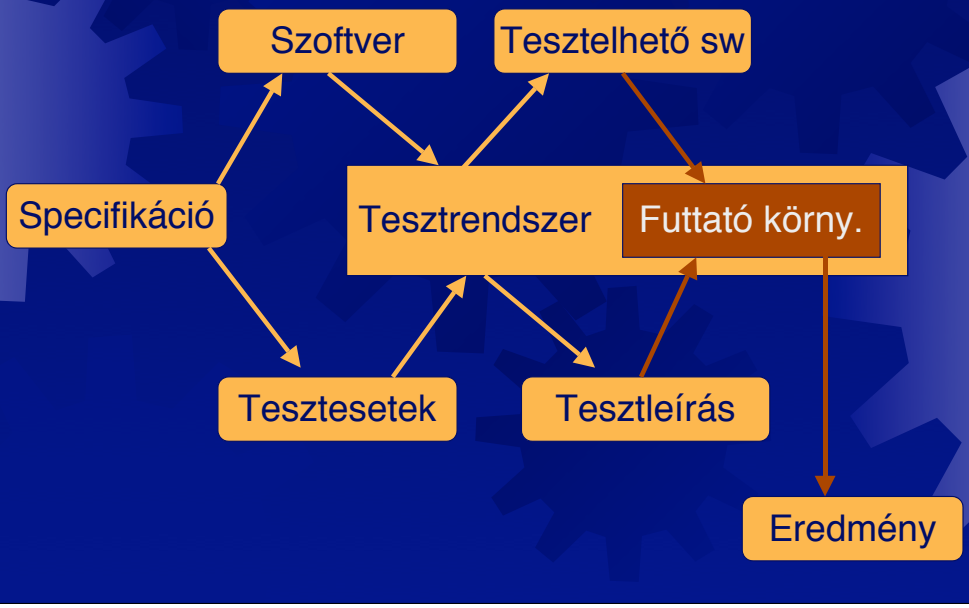
- Nem teljes mértékben kézbentartható környezet
- Tesztesetek meghatározása
  - Szoftver hiba modellek hiánya
- Tesztesetek végrehajtása
  - real-time környezet
  - kliens-szerver rendszerek
  - még nem létező rendszerelemek
  - kimenet, bemenet viselkedés definiálása
    - GUI tesztelés
    - több-szálú programok

GUI tesztelési problémák

- komplex állapotterek

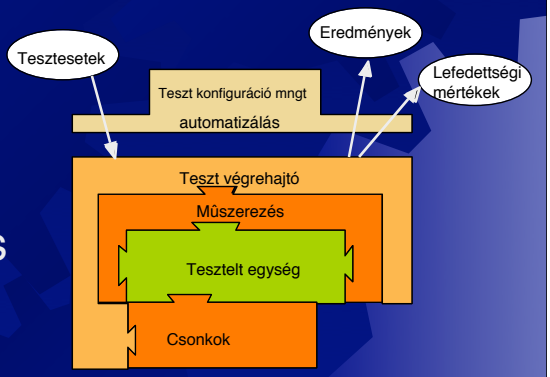


Végrehajtásos tesztelés kivitelezése



Tesztkörnyezetek

- dinamikus tesztek végrehajtása
- teszt végrehajtó (test driver)
- teszt automatizáló és konfiguráló komponens
- beműszerező komponens (instrumentation)
- szimuláló csonkok (stub)

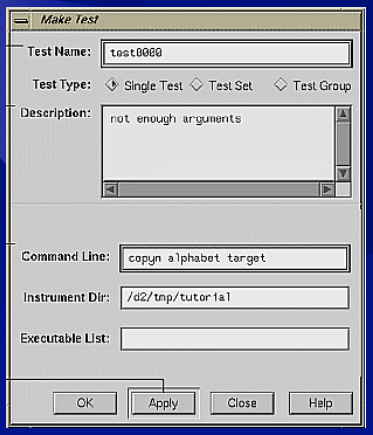


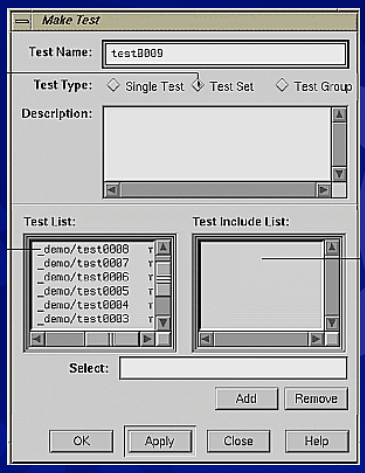
Tesztkörnyezetek szolgáltatásai

- teszt konfiguráció management
- automatizált tesztelés
- debugger
- teljesítmény tesztek (profiler)
- statikus analízis
- teszt eredmény dokumentáció, megjelenítés

Tesztkörnyezetek (pl.)

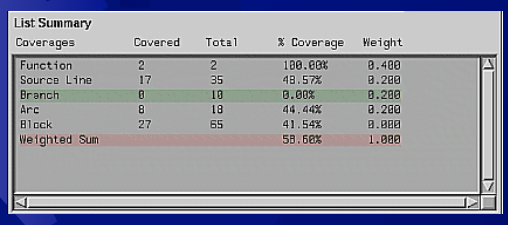
- sgi Tester
- teszt paraméterek beállítása

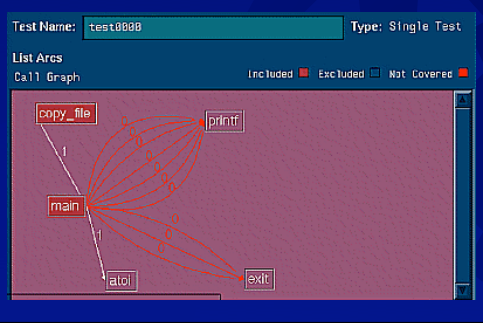


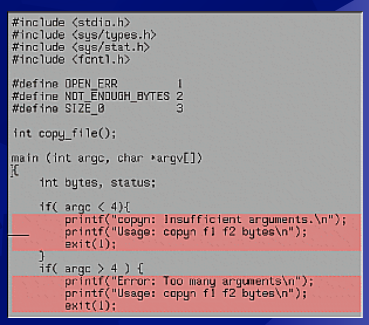


Tesztkörnyezetek (pl.)

- lefedettségi adatok



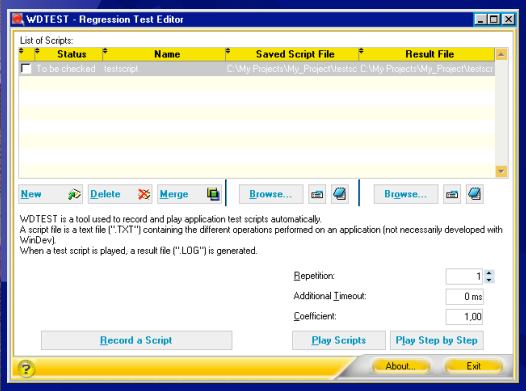




```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

#define OPEN_ERR        1
#define NOT_ENOUGH_BYTES 2
#define SIZE_0          3

int copy_file();

main (int argc, char *argv[])
{
    int bytes, status;

    if( argc < 4){
        printf("copyn: Insufficient arguments.\n");
        printf("Usage: copyn f1 f2 bytes\n");
        exit(1);
    }
    if( argc > 4 ) {
        printf("Error: Too many arguments\n");
        printf("Usage: copyn f1 f2 bytes\n");
        exit(1);
```

GUI tesztelő robot (event recorder)

- WDTest




```
T    512  860  753  47843    "Saj·tgÈp-CabinetWClass-717-653-0-0-"
T    512  861  753  47869    "Saj·tgÈp-CabinetWClass-718-653-0-0-"
T    512  863  753  47964    "Saj·tgÈp-CabinetWClass-720-653-0-0-"
T    512  864  753  47990    "Saj·tgÈp-CabinetWClass-721-653-0-0-"
T    512  864  752  48016    "Saj·tgÈp-CabinetWClass-721-652-0-0-"
T    512  866  752  48065    "Saj·tgÈp-CabinetWClass-723-652-0-0-"
T    512  867  752  48114    "Saj·tgÈp-CabinetWClass-724-652-0-0-"
T    516  867  752  48277    "-Shell_TrayWnd-865-8-0-0--TrayNotifyWnd-8-7-0-0-"
T    517  867  752  48371    "-Shell_TrayWnd-865-8-0-0--TrayNotifyWnd-8-7-0-0-"
T    512  867  752  48471    "WinVNC Tray Icon-WinVNC Tray Icon-819-685-0-0-"
T    512  867  752  48573    "WinVNC Tray Icon-WinVNC Tray Icon-819-685-0-0-"
T    512  869  750  48878    "WinVNC Tray Icon-WinVNC Tray Icon-821-683-0-0-"
T    512  885  742  48904    "WinVNC Tray Icon-WinVNC Tray Icon-837-675-0-0-"
T    512  897  740  48936    "WinVNC Tray Icon-WinVNC Tray Icon-849-673-0-0-"
T    512  903  739  48976    "WinVNC Tray Icon-WinVNC Tray Icon-855-672-0-0-"
```

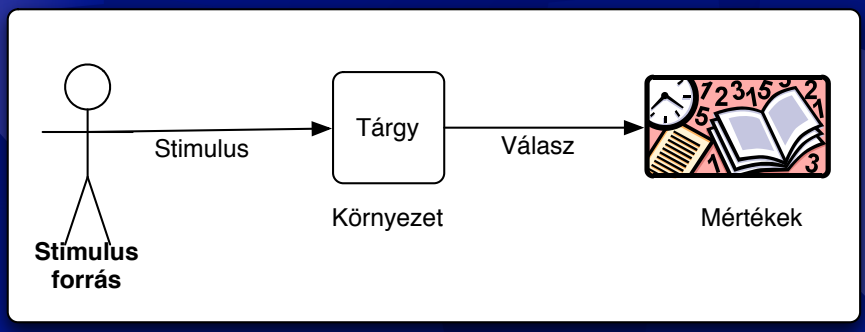
Teszt script (macro) részlet

Magasszintű tesztelési célok

- Hasznosság tesztelése
  - használhatóság
  - teljesítmény
  - költséghatékonyság
- Megbízhatóság
- Robusztusság
- Teljesítmény

Minőségi jellemzők tesztelése

- Minőség tesztelési scenáriók
- Metrikák



Tesztelési alapelvek

- Teszteredményeknek összevethetőnek kell lennie a követelményekkel
- A tesztelésnek tervezetten kell történnie
- A szoftver tesztelésre igaz a 80/20 elv: a tesztelés során feltárt hibák 80%-a a programkomponensek 20%-ára vezethető vissza
- A teszteléssel a “kicsitől” kell a “nagy” felétartani

Tesztelhetőség

- Tesztelhetőség mint tervezési szempont
  - biztonságkritikus rendszerek
- Működtethetőség
- Megfigyelhetőség
- Szabályozhatóság
- Szétbonthatóság
- Egyszerűség
- Érthetőség

Programhelyesség

- Programhelyesség
  - A program megfelel a kimenet-specifikációjának a megengedett feltételek és kívánt erőforrások mellett
  - specifikáció függő!

Program helyesség bizonyítás

- Matematikai technika
- A technika elemei
  - input specifikáció
  - output specifikáció
  - állítások (kijelentések)
  - vezérlési folyam invariáns
  - bizonyítási technikák
    - teljes indukció

A specifikáció helyessége

Input specification:	p : array of n integers, $n > 0$.
Output specfication:	q : array of n integers such that $q[0] \leq q[1] \leq \ldots \leq q[n-1]$

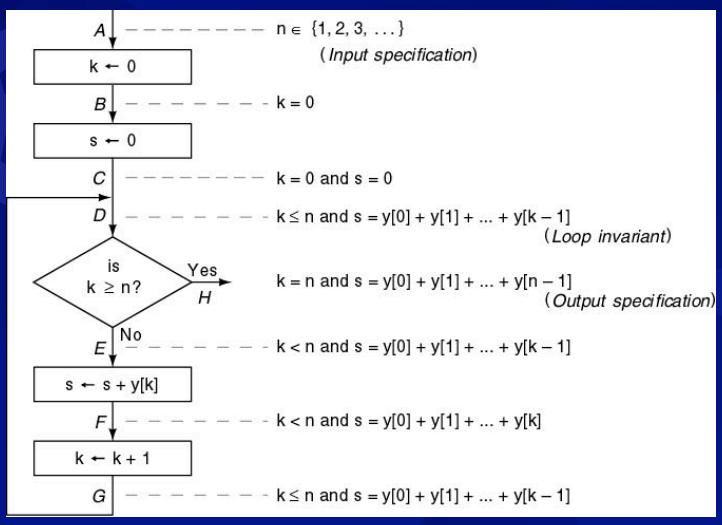
```
void trickSort (int p[], int q[])
{
   int i;
   for (i = 0; i < n; i++)
      q[i] = 0;
}
```

Input specification:	p : array of n integers, $n > 0$.
Output specfication:	q : array of n integers such that $q[0] \leq q[1] \leq \ldots \leq q[n-1]$
	The elements of array q are a permutation of the elements of array p, which are unchanged.

Programhelyesség bizonyítás (pl.)

```
int k, s;
int y[n];
k = 0;
s = 0;
while (k < n)
{
   s = s + y[k];
   k = k + 1;
}
```


Programhelyesség bizonyítás (pl.)



Programhelyesség bizonyítás (pl.)

- Java assertion

```
void foo() {
        for (...) {
            if (...)
                return;
        }
        assert false;
      // Execution should
      //never reach this point!
    }
```

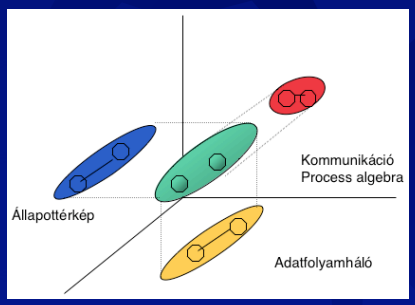
```
public class Foo
{
  public void m1( int value )
  {
    assert 0<=value;
    System.out.println( "OK" );
  }
  public static void main( String[] args )
  {
    Foo foo = new Foo();
    System.out.print( "foo.m1(  1 ): " );
    foo.m1( 1 );
    System.out.print( "foo.m1( -1 ): " );
    foo.m1( -1 );
  }
}
```


Formális módszerek

- szoftverrendszerek növekvő mérete és funkcionalitása -> növekvő komplexitás
- komoly következményekkel járó hibák nagyobb valószínűsége
- formális módszerek
  - szigorú specifikációs és verifikációs megoldások
    - matematikai (formális) nyelvek
    - technikák
    - eszközök
  - inkonzisztencia , nem teljesség, többértelműség felfedezése
  - megértés növelése

Formális specifikáció

- eldönthető szintaxis
- formális szemantika
- automatikus analízis
- gyenge és erős formális módszerek
- különböző nézetek integrálása



Formális verifikáció

- modell ellenőrzés
  - állapottér mérete (state space explosion)
    - 100-200 állapot változó, 10100 állapot
  - valószínűségi becslés
  - hardver és protokoll ellenőrzés
  - temporal model checking, automata
- tétel bizonyítás
  - végtelen állapotterek
  - axiómák, levezetési szabályok
  - heurisztikák használata
  - algoritmusok ellenőrzése

Formális módszerek sikeres alkalmazása

- Modell ellenőrzés
  - IEEE Futurebus cache koherencia protokoll
  - Bell Labs HDL Controller transmitter
- Tételbizonyítás
  - IBM PowerPC ekvivalencia ellenőrzés különböző terverzési szinteken
  - Motorola 68020 compiler generálta bináris kód ellenőrzése
  - AMD5K86 mikrokód

Formális módszerek alkalmazási problémái

- méretezési problémák
  - számítási komplexitás (NP-teljes problémák)
- tesztelési elemek a specifikáción kívül (bizt. kritikus rendszerek)
- felhasználói képzettség