Szoftver-minőségbiztosítás

Struktúra alapú (white-box) technikák

A struktúrális tesztelés

- Implementációs részletek figyelembevétele
- Tesztelési célok -> **lefedettség**
- Implicit hibamodell
  - A hibák a vezérlési szerkezeteket érintik
- Vezérlési folyam követése
- Kimerítő út tesztelés nem végezhető
- Metrika alapú teszttervezés

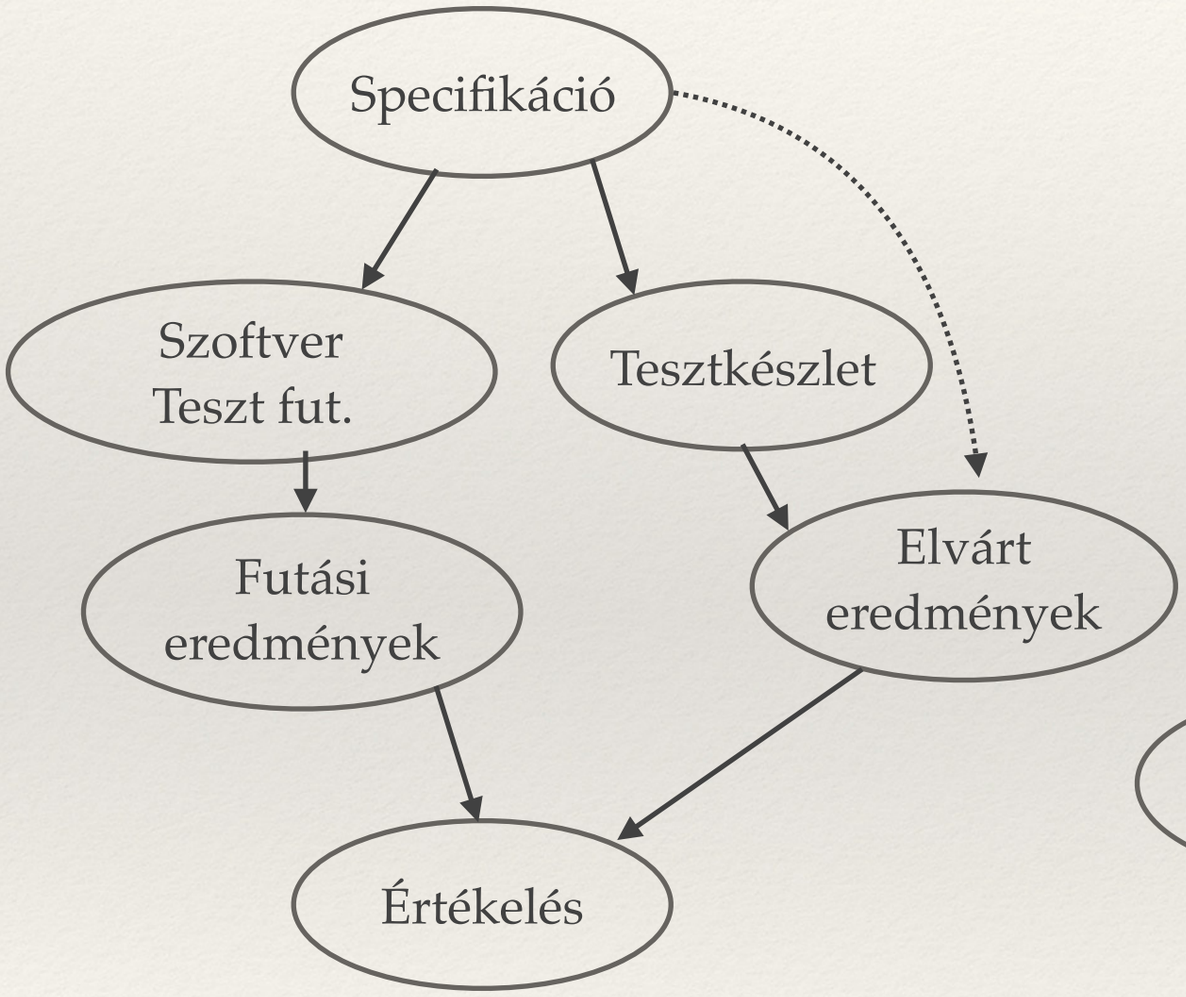
Struktúrális tesztelés alkalmazása

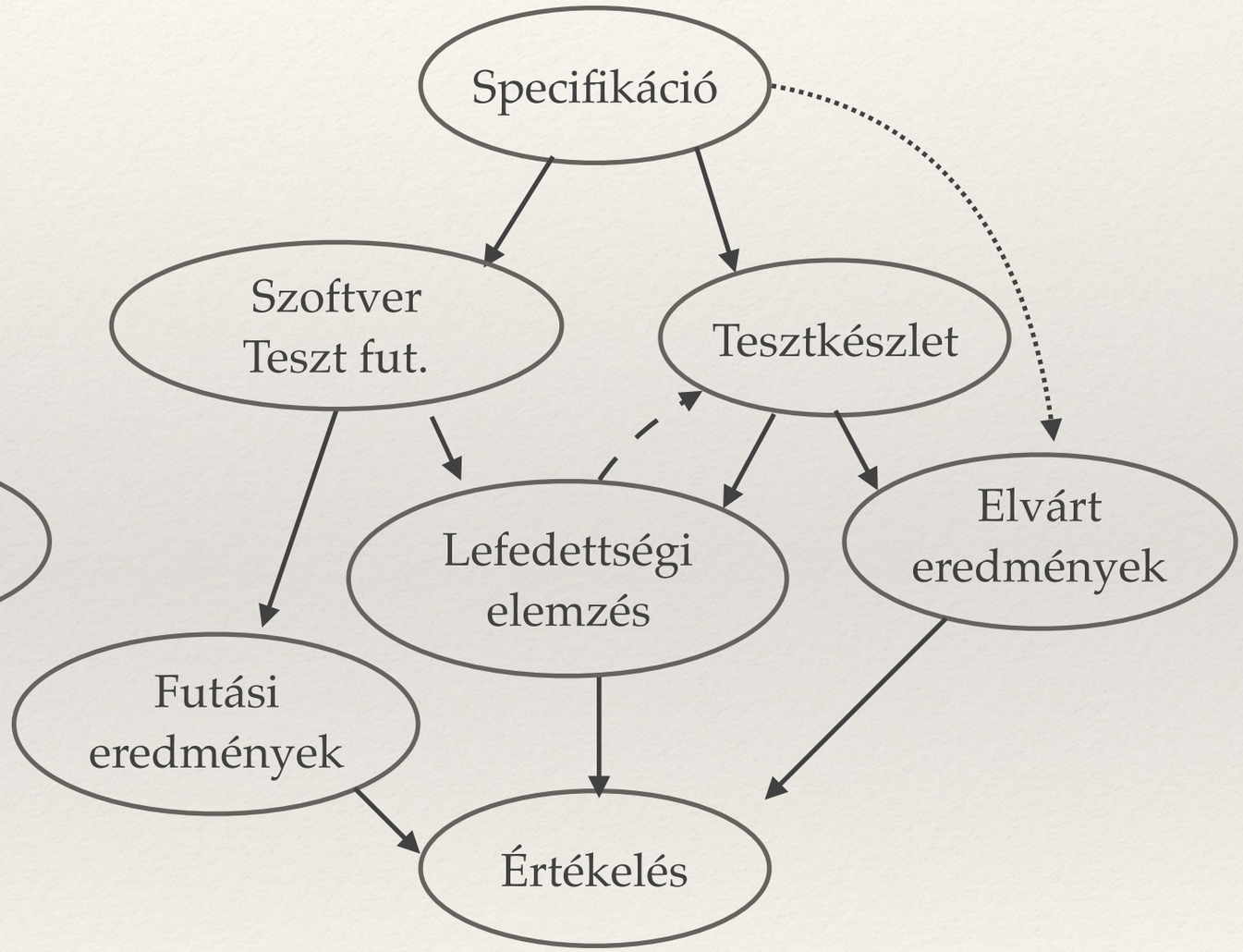
- Vezérlés-intenzív alkalmazások
- Tervezési hibák felderítése
- Szabványok szerinti tesztelés

Funkcionális és struktúrális tesztelés

- Funkcionális tesztelés alaposságának ellenőrzése
  - Specifikáció alapú (funkcionális) tesztekkel elért lefedettség mérése
    - kiegészítés további tesztesetekkel
    - valamilyen értelemben teljes lefedettség elérése
- Tisztán struktúrális alapon létrehozott tesztesetek

Funkcionális és struktúrális tesztelés (folyt.)





Funkcionális és struktúrális tesztelés (folyt.)

- önmagában egyik módszer sem kielégítő
- nincs szignifikáns különbség a hatékonyságban
- kiegészítik egymást
- végrehajtási sorrend:
  - funkcionális tesztek
  - struktúrális tesztek

Tesztlefedettség

- Annak a mértéke, hogy egy tesztkészlet lefuttatásával milyen mértékben hajtódott végre a program kód (Milyen alapos volt a tesztelés?)
- **L=(végrehajtott lefedettségi elemek száma)/(az összes a kódban lévő lefedettségi elemek száma)**
- A kódnak rendelkezésre kell állnia
- A lefedettségi elemeket meg kell tudni számlálni
- A tesztvégrehajtás közben "mérni" kell az elért lefedettséget

A tesztlefedettség jellemzői

- Ugyan azon tesztkészlet eltérő lefedettséget eredményez(het) eltérő lefedettségi elemekre
- Több eltérő tesztkészlet eredményezhet ugyan olyan lefedettséget (pontosan ugyanazokat a lefedettségi elemeket)
  - de **eltérő lehet a hiba-felderítő képességük**
- A lefedettségi mutató segítségével nem lehet azonosítani a specifikáció nem implementált részeit

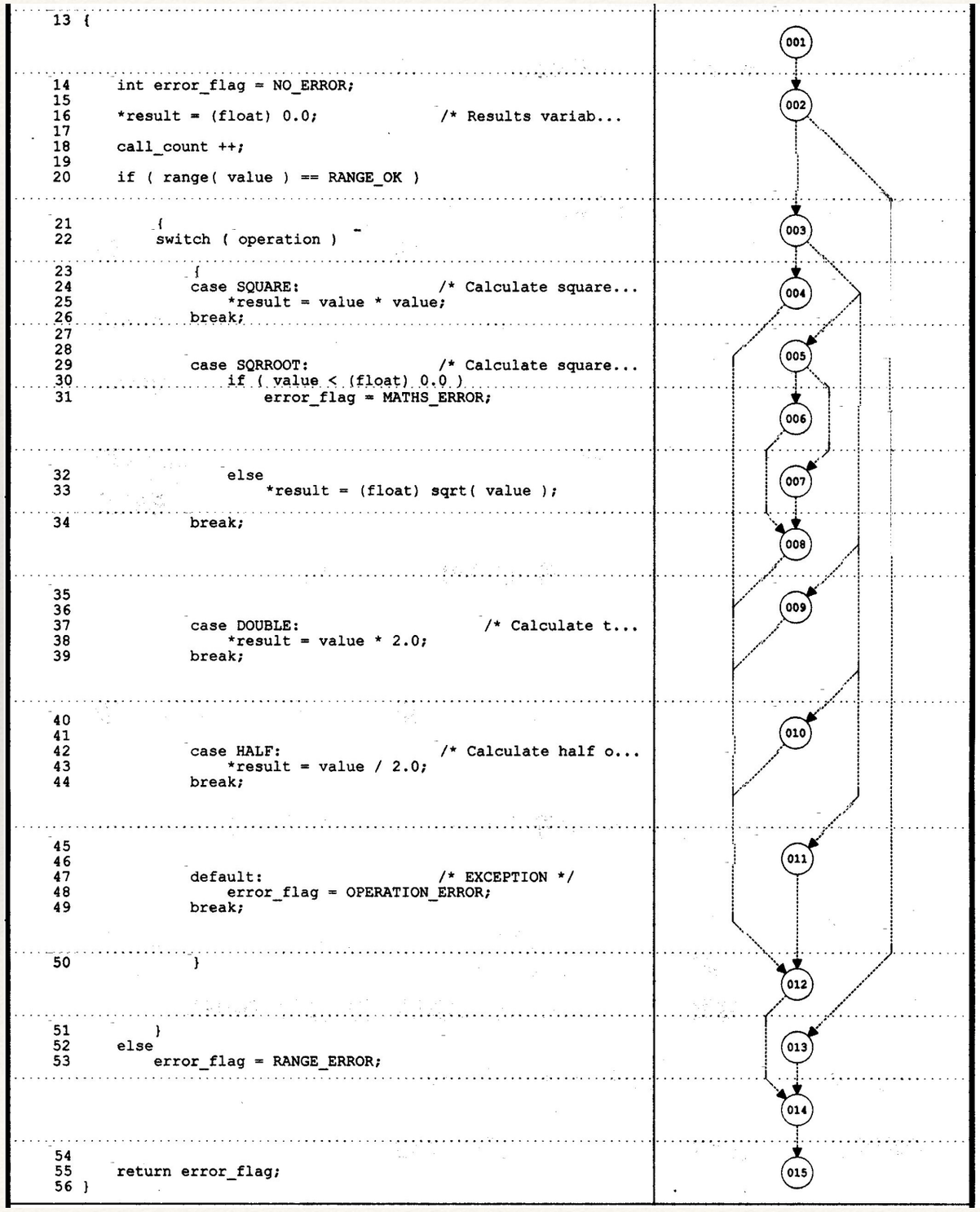
Tesztlefedettség elemzők

- Speciális teszteszközök
- a lefedettség mérésével kapcsolatos tevékenységek automatizálására
  - a program kód **instrumentálása** (beműszerezés)
  - tesztek végrehajtása (az instrumentált kód lefedettségi adatainak gyűjtése)
  - tesztjelentés készítése (melyből meghatározhatók a lefedett elemek ill. előállnak a lefedettségi statisztikák)

Vezérlésfolyam gráf

- A vezérlésfolyam (control flow) elemzés a program futás közbeni viselkedésének a program struktúrára vetítése
- Vezérlésfolyam gráf (CFG)
  - A struktúrális tesztek (lefedettség elemek) meghatározásának egyik legfontosabb kiindulási modellje
  - A program lebontása alap blokkokra (szekvenciálisan egymást követő utasítások)
  - A gráf elemei:
    - N - az alapblokkok halmaza, gráf csomópontok
    - E - a blokkok egymásutánisága, gráf élek
    - n0 - a kiinduló blokk

Vezérlésfolyam gráf (pl.)



Lefedettség típusok

- Magas szintű tesztelési lefedettség elemek
  - követelmény-lista elem, menü opció, tranzakció típus, adatbázis struktúra elem, interfész elem, állapot átmenet, stb.
- **Kódlefedettség típusok**
  - vezérlésfolyam gráf elemek:
    - utasítások
    - alap blokkok (döntés-döntés utak)
    - döntés kimenetelek (ágak)
    - összetett logikai kifejezések részkifejezései
    - független utak

Kód lefedettség mértékek

Metrika	Leírás
C0	Utasítás lefedés (minden utasítás)
C1	Döntés-döntés út lefedés (minden alap blokk)
C1p	Döntés kimenet lefedés (minden döntés minden kimenete, ág lefedés)
C2	C1 + ciklus lefedés
Cd	C1 + függő út-párok figyelembevétele
CMCC	Összetett feltétel lefedés (boole lefedés)
C∞	Út lefedés (minden lehetséges út)

Lefedettség mérése

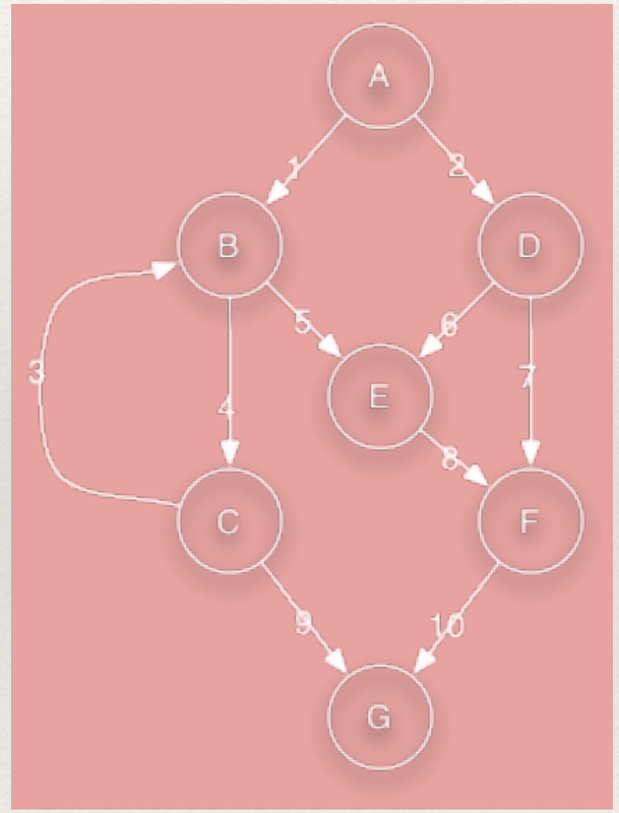
- Az instrumentálás intruzív tevékenység!
- A különböző lefedettség elemeket általában a magaszintű programozási nyelv elemeire értjük (utasítás, ág, logikai kifejezés, stb.)
  - object code visszavezetése magasszintű forrásra (optimalizáló compiler 💀)
- végrehajtáskor az instrumentálás okozhat overhead-et
  - real time követelmények, spec. időzítések?!

Ciklomatikus komplexitás

- McCabe definiálta program-komplexitás mérésére
- Gráf ciklomatikus száma
  - erősen összefüggő síkbeli irányított gráf ennyi régióra osztja a síkot
  - független körök száma a gráfban
  - V(G)=e-n+p
- Vezérlésfolyam gráfra:
  - V(G)=e-n+2p (p=1) => **V(G)=e-n+2**
  - lineárisan független program-utak száma (lehetséges program-utak "bázis rendszerének" számossága)

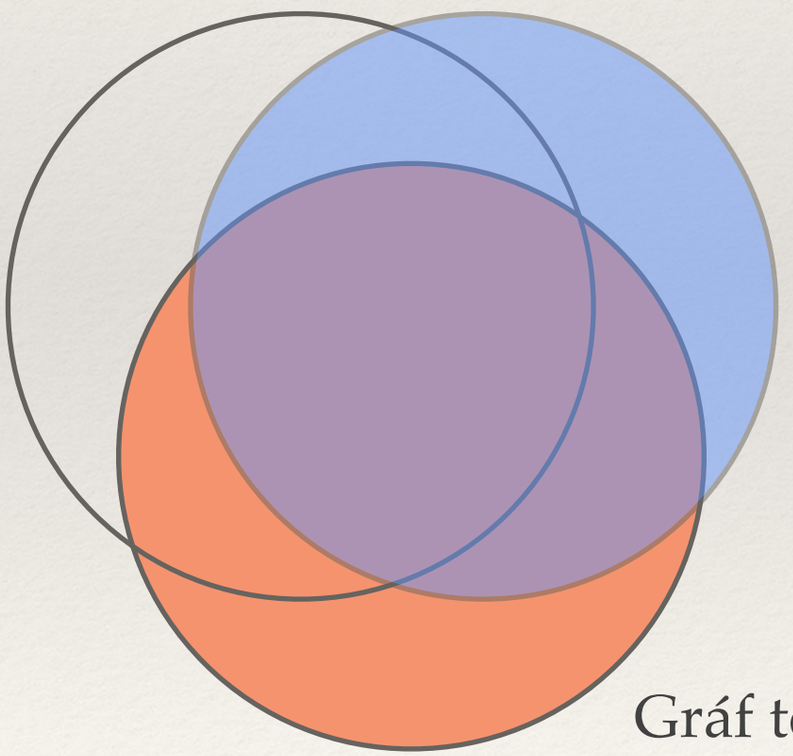
Lineárisan független utak (pl.)

Utak\Élek	1	2	3	4	5	6	7	8	9	10
A,B,C,G	1	0	0	1	0	0	0	0	1	0
A,B,C,B,C,G	1	0	1	2	0	0	0	0	1	0
A,B,E,F,G	1	0	0	0	1	0	0	1	0	1
A,D,E,F,G	0	1	0	0	0	1	0	1	0	1
A,D,F,G	0	1	0	0	0	0	1	0	0	1
A,B,C,B,E,F,G	1	0	1	1	1	0	0	1	0	1
A,B,C,B,C,B,C,G	1	0	2	3	0	0	0	0	1	0



Program és gráf-topológia

- A gráfutak halmaza eltérhet a program (implementált) valódi bejárható útjaitól!



Ciklomatikus komplexitás és tesztelés

- **minimális cél a független utak egy maximális halmazának lefedése tesztekkel**
- a maximális (számosságú) úthalmaz **nem egyedi**
- a ciklomatikus komplexitás használható a független utak számának meghatározására
- McCabe a ciklomatikus komplexitásról és az útalapú tesztelésről:
  - csak tesztelési minőségi kritérium, **nem módszer a tesztesetek azonosítására**

Ciklomatikus komplexitás és tesztelhetőség

- A ciklomatikus komplexitás jó metrika a **program komplexitására** (fejlesztési, megértési erőfeszítés)
  - hibavalószínűség => tesztelési figyelem
  - tesztelési nehézség

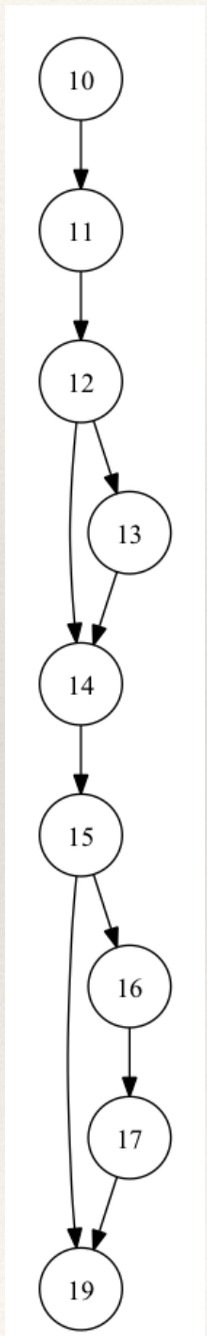
Ciklomatikus komplexitás	Kockázat
1-10	Egyszerű program
11-20	Komplex program, mérsékelt kockázat
21-50	Komplex, magas kockázat
50-	Tesztelhetetlen program

A lefedettség erőssége

- Lefedettség típusok erőssége (alaposssága):
  - C∞ > C2 > C1 > C0
- Az erősebb lefedettség magában hordozza a gyengébbet
- A szabványok általában C1 lefedettség elérését kívánják meg

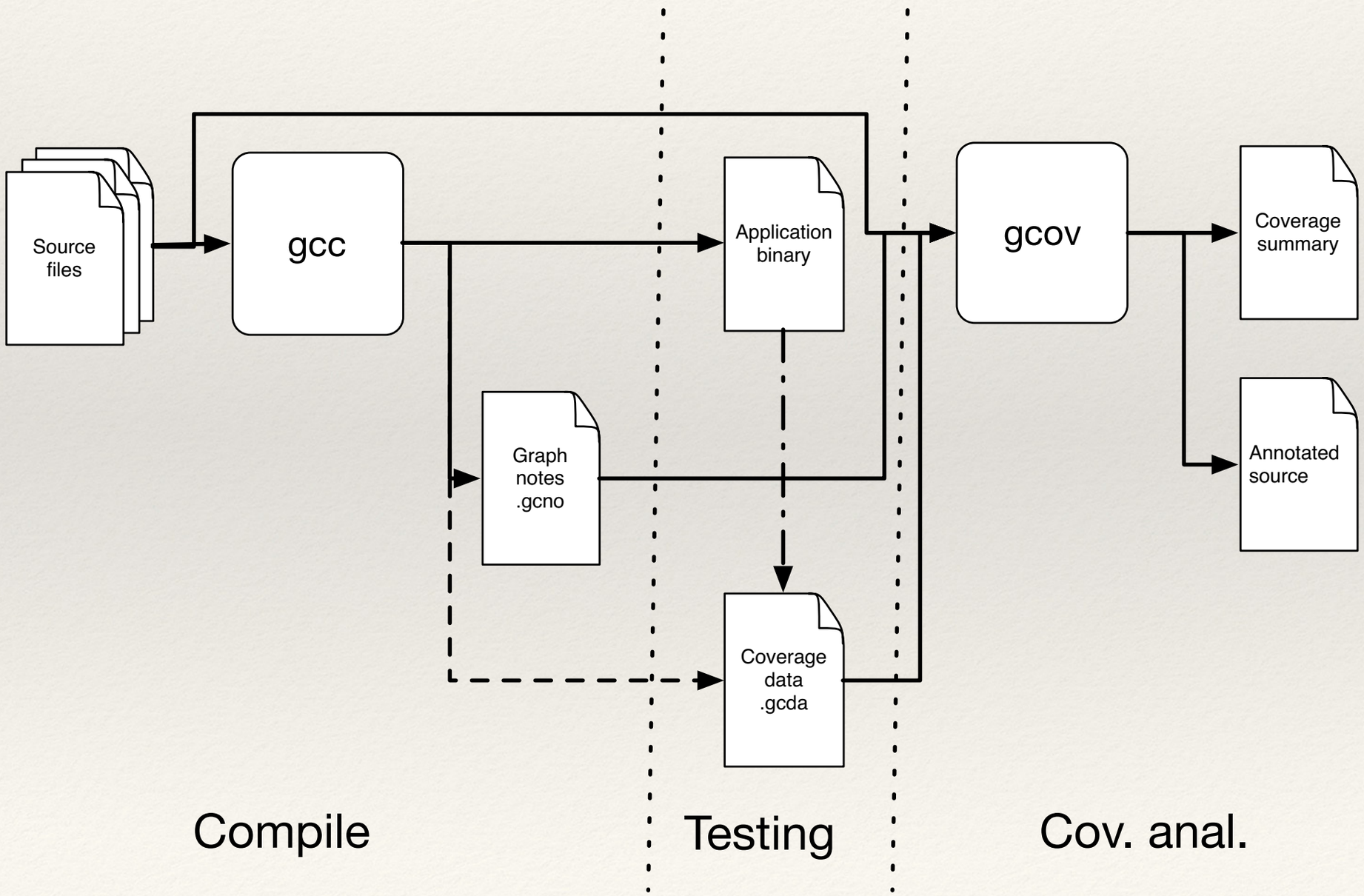
Szemléltetés

```
1     #define PI    3.14159265358979323846
2
3     double dmod(double x, double y){
4       double q;
5
6       q=x/y;
7       return x-y*(long long)q;
8     }
9
10     double sine(double x){
11       double a=dmod(x,2.0*PI);
12       if(a<0.0)
13         a+=2*PI;
14       int s=1;
15       if(a>PI){
16         a-=PI;
17         s=-1;
18       };
19       return s*(a-a*a*a/6.0+a*a*a*a*a/120.0-a*a*a*a*a*a*a/5040.0);
20     }
```



Szemléltetés (folyt.)

- gcov lefedettség elemzés



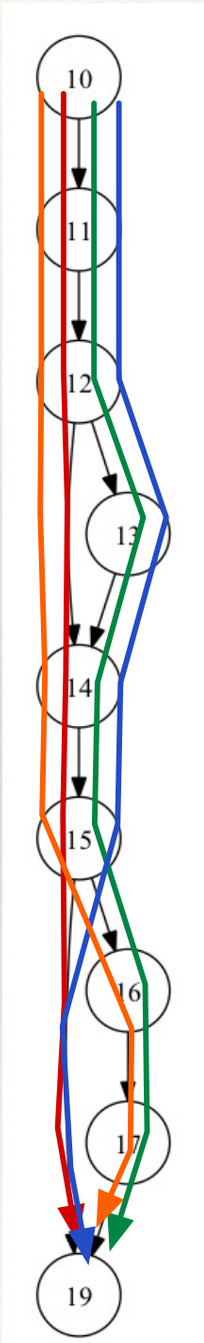
Szemléltetés (folyt.)

```
Tesztesetek:
 IN: -2.000000 OUT: -0.909288 RES: OK => 100% statement cov., 50% branch cov.
+IN: 1.000000 OUT: 0.841468 RES: OK => 100% branch cov.
+IN: -4.000000 OUT: 0.752369 RES: OK => 100% indep. path cov.
+IN: 4.000000 OUT: -0.756802 RES: OK => 100% path cov.
```

```
File 'sine.c'
Lines executed:100.00% of 12
Branches executed:100.00% of 4
Taken at least once:100.00% of 4
Calls executed:100.00% of 1
Creating 'sine.c.gcov'
```

gcov output és annotált forrás fájl

```
        -:    0:Source:sine.c
        -:    0:Graph:sine.gcno
        -:    0:Data:sine.gcda
        -:    0:Runs:4
        -:    0:Programs:1
        -:    1:#define PI             3.14159265358979323846
        -:    2:
function dmod called 4 returned 100% blocks executed 100%
        4:    3:double dmod(double x, double y){
        -:    4:  double q;
        -:    5:
        4:    6:  q=x/y;
        4:    7:  return x-y*(long long)q;
        -:    8:}
        -:    9:
function sine called 4 returned 100% blocks executed 100%
        4:   10:double sine(double x){
        4:   11:  double a=dmod(x,2.0*PI);
call    0 returned 4
        4:   12:  if(a<0.0)
branch  0 taken 2 (fallthrough)
branch  1 taken 2
        2:   13:    a+=2*PI;
        4:   14:  int s=1;
        4:   15:  if(a>PI){
branch  0 taken 2 (fallthrough)
branch  1 taken 2
        2:   16:    a-=PI;
        2:   17:    s=-1;
        -:   18:  };
        4:   19:  return s*(a-a*a*a/6.0+a*a*a*a*a/120.0-a*a*a*a*a*a*a/5040.0);
        -:   20:}
```



Összetett logikai kifejezések feltételekben

- Az összetett logikai kifejezés összes lehetséges kombinációja tesztelendő
- utasítás komplexitás <-> út komplexitás
- különböző nyelvek eltérő logikai operátor szemantikája
- "mellékhatások" az összetett logikai kifejezésben

Összetett logikai kifejezések feltételekben (pl.)

Amikor a C_{MCC} elérése fontos lehet:

```
if ( feltétel1 && ( feltétel2 || függvény1() ) )
   utasítás1;
 else
    utasítás2;
```

Tesztesetek:
1. feltétel1 hamis
2. feltétel1 igaz, feltétel2 igaz

Teljes C_1 lefedés, de függvény1() nem került meghívásra.

Szemléltetés (Összetett logikai kif.)

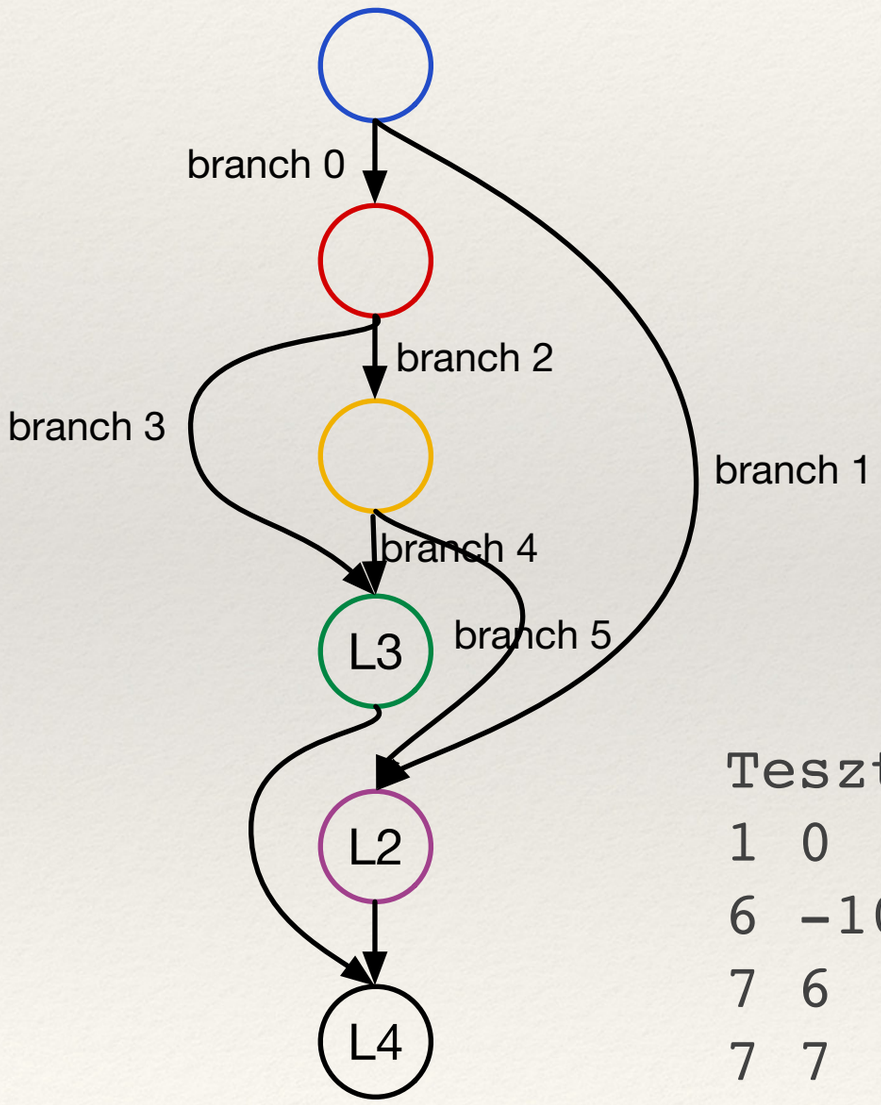
- gcc, gcov és összetett logikai kifejezések

```
#include <stdio.h>

int main(int argc, char *argv[]){
  int a,b;

  a=atoi(argv[1]);
  b=atoi(argv[2]);
  if(a > 5 && (b<0 || 360%b==0))
    printf("Yes\n");
  else
    printf("No\n");
  return 0;
}
```

```
.LC0:
        .ascii  "Yes\000"
        .align  2
.LC1:
        .ascii  "No\000"
        .text
        .align  2
...
        ldr   r3, [r7, #12]   @ tmp119, a
        cmp   r3, #5      @ tmp119,
        ble   .L2   @,
        .loc 1 8 0 is_stmt 0 discriminator 1
        ldr   r3, [r7, #8]    @ tmp120, b
        cmp   r3, #0      @ tmp120,
        blt   .L3   @,
        mov   r3, #360    @ tmp121,
        mov   r0, r3      @, tmp121
        ldr   r1, [r7, #8]    @, b
        bl    __aeabi_idivmod @
        mov   r3, r1      @ tmp126,
        cmp   r3, #0      @ D.5431,
        bne   .L2   @,
.L3:
        .loc 1 9 0 is_stmt 1
        movw  r0, #:lower16:.LC0    @,
        movt  r0, #:upper16:.LC0    @,
        bl    puts  @
        b     .L4   @
.L2:
        .loc 1 11 0
        movw  r0, #:lower16:.LC1    @,
        movt  r0, #:upper16:.LC1    @,
        bl    puts  @
.L4:
```

Szemléltetés (Összetett logikai kif. folyt.)

- gcc, gcov és összetett logikai kifejezések



assembly CFG

```
Tesztek:
1 0
6 -10
7 6
7 7
```

```
Lines executed:100.00% of 7
Branches executed:100.00% of 6
Taken at least once:100.00% of 6
Calls executed:100.00% of 4

Source:cmcc.c
        -:    0:Graph:cmcc.gcno
        -:    0:Data:cmcc.gcda
        -:    0:Runs:4
        -:    0:Programs:1
        -:    1:#include <stdio.h>
        -:    2:
function main called 4 returned 100% blocks executed 100%
        4:    3:int main(int argc, char *argv[]){
        -:    4:  int a,b;
        -:    5:
        4:    6:  a=atoi(argv[1]);
call    0 returned 4
        4:    7:  b=atoi(argv[2]);
call    0 returned 4
        4:    8:  if(a > 5 && (b<0 || 360%b==0))
branch  0 taken 3 (fallthrough)
branch  1 taken 1
branch  2 taken 2 (fallthrough)
branch  3 taken 1
branch  4 taken 1 (fallthrough)
branch  5 taken 1
        2:    9:    printf("Yes\n");
call    0 returned 2
        -:   10:  else
        2:   11:    printf("No\n");
call    0 returned 2
        4:   12:  return 0;
        -:   13:}
```

Összetett logikai kif. nem független részkifejezésekkel

Nem érhető el teljes boole lefedés

704	E		763
705		+ +	763
706		+ +	212
707			
708		+ +	631
709		+ -	80
710			
711		+ +	711
712		+ +	1526
713			763
714			763
715			
716			167
717			
718			167
719			
720	E		763
721	E		763
722			

```
vp_display := p_sum_wig_tcr (vp_occupation_status, p1.wig_entry);
IF ((p1.common_data.f_number     = 0 ANDIF
     p1.common_data_add.f_number = 0)
     ORIF
    ((p1.common_data.f_number     /= 0 ORIF
     p1.common_data_add.f_number /= 0)
     AND
     p0.common_range_displ_deact))          AND
   NOT p1.virtual                           AND
   v_sb_elem_updated (p1.identifier.f_name) AND
   vp_display /= vp_last_occup_display
THEN
  p_elem_displ (p1.identifier, 0,
                p_displ_from_state (vp_display));
  vp_last_occup_display := vp_display;
FI;
CHANGE (p1 periph_status      := vp_occupation_status,
           last_occup_display := vp_last_occup_display);
```

shortcircuite AND

shortcircuite OR

Az adatfolyam tesztelése

- Az adatfolyam alapú tesztelés vezérlés folyam információ (gráf) segítségével elemzi az adatfolyamot
- Adatfolyam
  - értékadásoktól (DEF) az érték felhasználásáig (USE)
  - két féle felhasználás: számításban (C-USE), elágazási feltételben (P-USE)
- Input (változók)tól az output (változók)ig
- Cél: a tesztekkel végrehajtani az összes útvonalat a definícióktól a felhasználásokig

Az adatfolyam tesztelése (folyt.)

- változók értékadásai és felhasználásuk
- definiálás/hivatkozás hibák
  - never used
  - never defined
  - define more times before used
- **DF-utak** (definíció/felhasználás)
  - DEF(v,m)
  - USE(v,n), p-use, c-use
  - definition-clear path

Az adatfolyam tesztelése (folyt.)

- Rapps-Weyuker adatfolyam lefedési metrikák
  - All-DU paths
  - All-Uses
  - All C-uses / some P-uses
  - All P-uses / some C-uses
  - All-Defs

Adatfolyam utak (pl.)

```
1  int testfloat(char s[]){
2        int i=0;
3        int f=0;
4        while(isspace(s[i]))
5                i++;
6        if(s[i]=='+'||s[i]=='-')
7                i++;
8        f=i;
9        while(isdigit(s[i]))
10               i++;
11       if(s[i]=='.'){
12               i++;
13               f=i;
14               while(isdigit(s[i]))
15                       i++;
16               if(f==i)
17                       return 0;
18       }
19       if(f==i)
20                return 0;
21       if(s[i]=='e'||s[i]=='E'){
22               i++;
23               if(s[i]=='+'||s[i]=='-')
24                       i++;
25               f=i;
26               while(isdigit(s[i]))
27                       i++;
28               if(f==i)
29                       return 0;
30       }
31       if(s[i])
32                return 0;
33       else return 1;
34 }
```

du-path1: DEF(i,2)..C-USE(i,8) nem definíció mentes pl. DEF(i,5)
du-path2: DEF(f,3)..P-USE(f,16) nem definíció mentes pl. DEF(f,8)
du-path3: DEF(f,25)..P-USE(f,28) definíció mentes