

Szoftverellenőrzési technikák

Struktúra alapú tesztervezési módszerek

Majzik István, Micskei Zoltán

<http://www.inf.mit.bme.hu/>

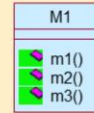
1

Utolsó módosítás: 2014.10.26.

Teszttervezés módszerei

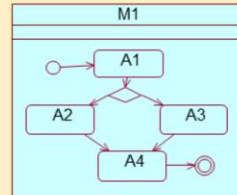
I. Specifikáció alapú

- A rendszer mint „fekete doboz” adott
- Csak a külső viselkedés (funkció) ismert, a belső felépítés (pl. forráskód) nem
- Tesztelés alapja: specifikált funkciók léte; extra funkciók hiánya



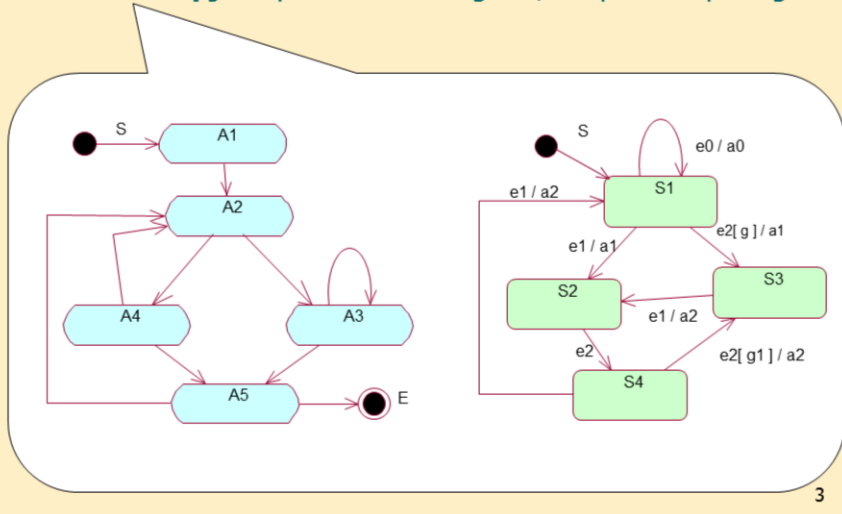
II. Struktúra alapú

- A rendszer mint „üvegdoboz” adott
- A belső struktúra is ismert
- Tesztelés alapja a **belső működés**: programgráf bejárása



A belső struktúra

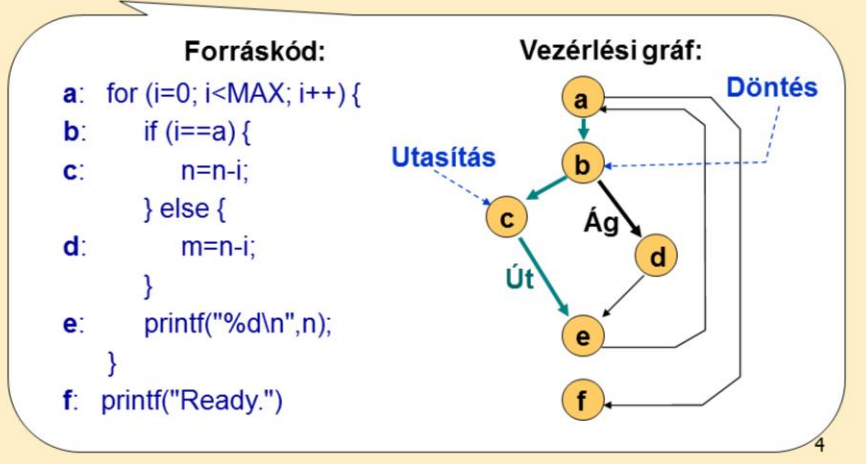
- Jól kezelhető struktúra:
 - **Modell alapján:** pl. aktivitás diagram, állapottérkép diagram



Ez vonatkozhat egy komponens belső működésére is, komponensek közötti együttműködés leírására, de akár pl. egy egész üzleti folyamat leírására.

A belső struktúra

- Jól kezelhető struktúra:
 - Modell alapján: pl. aktivitás diagram, állapotterkép diagram
 - **Forráskód alapján:** vezérlési gráf (programgráf)



Tesztminőségi mértékszámok

A tesztelés hatékonyságának számszerű jellemzése:

A tesztelhető elemek mekkora részét teszteltük, pl.

- | | |
|----------------------|------------------------|
| 1. Utasítások | → Utasítás lefedettség |
| 2. Döntések | → Döntési lefedettség |
| 3. Feltételek | → Feltétel lefedettség |
| 4. Végrehajtási utak | → Út lefedettség |

Ez **nem** a hibafedés!

Szabványok előírása lehet (DO-178B, MSZ EN 50128,...)

- 100% utasítás fedettség általában alapkövetelmény

Tartalom

- Vezérlési folyamat alapú kritériumok
 - Utasítás lefedettség
 - Döntési lefedettség
 - Feltétel lefedettségek
 - Útvonal lefedettség
- Adatfolyam alapú kritériumok
 - Definiálás – használat fedettségek
 - Definíciómentes útvonalak fedettsége
- Összefoglalás
 - Módszerek kombinációja
 - Teszt fedettség mérése

Alapfogalmak

- Utasítás (statement)
- Blokk (block)
 - Utasítások egybefüggő sorozata, amik között nincs elágazás vagy függvényhívás
- Feltétel (condition)
 - Egyszerű vizsgálat, amiben nincs logikai (Boole) operátor
- Döntés (decision)
 - Nulla vagy több logikai operátorral összekötött feltételből álló kifejezés
- Döntési ág (branch)
 - Egy döntés lehetséges kimenetele
- Út (path)
 - Utasítások sorozata, tipikusan a modul be és kilépési pontja között

7

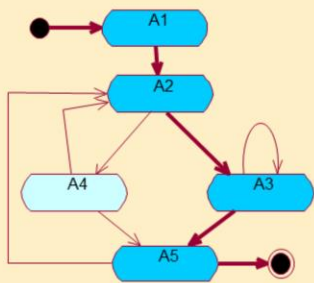
Sajnos ezeknek az alapvető fogalmaknak a definíciójában nincs teljes egyetértés (pl. ISTQB vs. DO-178B vs. IEEE)

1. Utasítás lefedettség (Statement coverage)

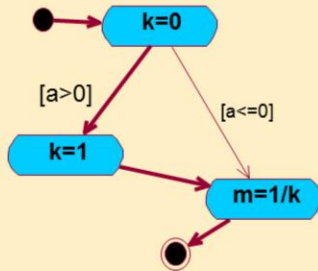
Definíció:

$$\frac{\text{Tesztelés során végrehajtott utasítások száma}}{\text{Összes utasítás száma}}$$

Utasítások kihagyási feltételeit nem veszi figyelembe!



Utasítás fedettség: 80%



Utasítás fedettség: 100%

- Előkerül még a line coverage elnevezés is, ami nem szerencsés, mert egy sorban lehet több utasítás is.

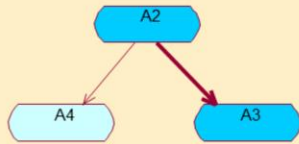
2. Döntés lefedettség (Decision coverage)

Definíció:

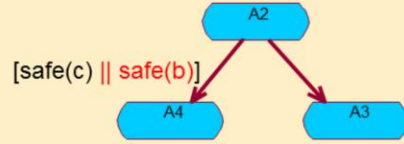
Tesztelés során végrehajtott döntési ágak száma

Összes lehetséges döntési ág száma

Nem vesz figyelembe minden feltétel-kombinációt!



Döntési ág fedettség: 50%



Döntési ág fedettség: 100%

Példa 100%-os döntés lefedettséghez:
1. safe(c) = true, safe(b) = false
2. safe(c) = false, safe(b) = false

9

-Használják még a branch coverage elnevezést is, azonban az például repülő környezetben nem ugyanaz mint a decision coverage, lásd:

FAA, Certification Authorities Software Team, „What is a "Decision" in Application of Modified Condition/Decision Coverage (MC/DC) and Decision Coverage (DC)?”, 2002.
URL:

http://www.faa.gov/aircraft/air_cert/design_approvals/air_software/cast/cast_papers/media/cast-10.pdf

3. Feltétel lefedettség (Condition coverage)

Generikus fedettségi mérték feltétel fedettségekhez:

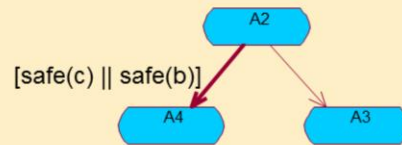
$$\frac{\text{Feltételek tesztelt kombinációinak száma}}{\text{Feltételek megcélzott kombinációinak száma}}$$

Definíció (milyen kombinációkat célzunk meg):

- Minden feltétel legyen igaznak és hamisnak is beállítva a tesztelés során
 - Nem feltétlenül eredményez 100% döntés lefedettséget!

Példa 100%-os feltétel lefedettséghez:

1. $\text{safe}(c) = \text{true}$, $\text{safe}(b) = \text{false}$
2. $\text{safe}(c) = \text{false}$, $\text{safe}(b) = \text{true}$



Más definíció:

- Minden feltételt igaznak és hamisnak is kiértékelünk
 - Ez nem ugyanaz mint a fenti, a lusta kiértékelés miatt

10

Példa arra, hogy 100%-os feltétel lefedettség nem eredményez 100%-os döntés lefedettséget:

- Döntés: $((a \text{ AND } b) \text{ OR } (c \text{ AND } d))$

- T1: $a=1, b=0, c=1, d=0$

- T2: $a=0, b=1, c=0, d=1$

- Minden feltétel (a,b,c,d) szerepel igaz és hamis értékkel is, de a döntés mindkét esetben hamisra értékelődik ki, így a döntés lefedettség csak 50%-os

Forrás: Y. T. Yu and M. F. Lau, "A comparison of MC/DC, MUMCUT and several other coverage criteria for logical decisions," *Journal of Systems and Software*, vol. 79, no. 5, pp. 577-590, May. 2006.

4. Feltétel/döntés lefedettség

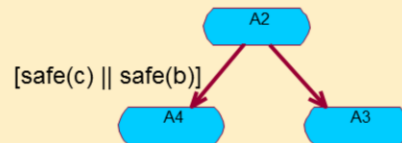
Condition/Decision Coverage (C/DC)

Definíció:

- Minden feltétel felveszi az összes lehetséges kimenetét legalább egyszer, és
- minden döntés felveszi az összes lehetséges kimenetét egyszer

100%-os C/DC lefedettséghez:

1. `safe(c) = true, safe(b) = true`
2. `safe(c) = false, safe(b) = false`



Nem vizsgálja meg, hogy a feltételnek tényleg van-e hatása a döntés eredményére!

11

Az egyik paraméter true értéke önmagában meghatározza a döntést.

De a false érték esetén meg kellene nézni, hogy a másik váltása hogyan befolyásol (befolyásol-e).

5. Módosított feltétel/döntés lefedettség

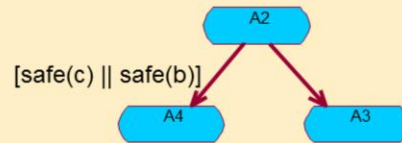
Modified Condition/Decision Coverage (MC/DC)

Definíció:

- Minden feltétel felveszi az összes lehetséges kimenetét legalább egyszer, és
- minden döntés felveszi az összes lehetséges kimenetét egyszer, és
- minden feltétel a többitől függetlenül befolyásolja a hozzá tartozó döntés kimenetelét

100%-os MC/DC lefedettséghez:

1. `safe(c) = true, safe(b) = false`
2. `safe(c) = false, safe(b) = true`
3. `safe(c) = false, safe(b) = false`



12

OR (||) művelet van: Az egyik paraméter false értéke mellett befolyásolhatja a másik paraméter a döntés eredményét, ezzel kell tehát összepárosítani a másik paraméter true és false értékét is!

6. Minden feltétel kombináció lefedettsége

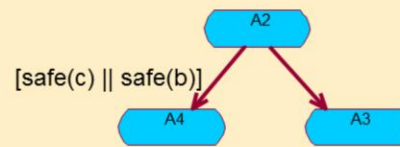
Multiple Condition Coverage

Definíció:

- A feltételek kimeneteinek minden lehetséges kombinációja bekövetkezett a tesztelés során
 - Általában a feltételek számával exponenciálisan növekedő számú teszt szükséges
 - Kevesebb, ha a lusta kiértékelést is figyelembe vesszük

100%-os feltétel kombináció lefedettséghez:

1. `safe(c) = true, safe(b) = false`
2. `safe(c) = false, safe(b) = true`
3. `safe(c) = false, safe(b) = false`
4. `safe(c) = true, safe(b) = true`



13

Akkor nem praktikus, ha sok feltételből álló döntések vannak. Pl. repülőipari szoftverekben nem ritka a 30 feltételből álló döntés.

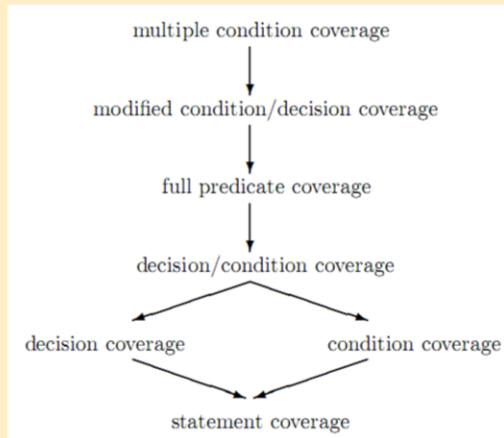
Feltétel és döntési lefedettségek összefoglalása

Table 1. Types of Structural Coverage

Coverage Criteria	Statement Coverage	Decision Coverage	Condition Coverage	Condition/Decision Coverage	MC/DC	Multiple Condition Coverage
Every point of entry and exit in the program has been invoked at least once		•	•	•	•	•
Every statement in the program has been invoked at least once	•					
Every decision in the program has taken all possible outcomes at least once		•		•	•	•
Every condition in a decision in the program has taken all possible outcomes at least once			•	•	•	•
Every condition in a decision has been shown to independently affect that decision's outcome					•	• ⁸
Every combination of condition outcomes within a decision has been invoked at least once						•

Forrás: Kelly J. Hayhurst et al. „A Practical Tutorial on Modified Condition/Decision Coverage”, NASA/TM-2001-210876, 2001

Feltétel és döntési lefedettségek összefoglalása



Vezérlési folyamat
alapú kritériumok
viszonya

Forrás: S. A. Vilkomir and J. P. Bowen, "From MC/DC to RC/DC: formalization and analysis of control-flow testing criteria," *Formal Aspects of Computing*, vol. 18, no. 1, pp. 42-62, 2006.

Példa: Vezérlési folyamat alapú kritériumok

```
Product getProduct(String name, Category cat) {  
    if (name == null || ! cat.isValid)  
        throw new IllegalArgumentException();  
  
    Product p = ProductCache.getItem(name);  
  
    if (p == null) {  
        p = DAL.getProduct(name, cat);  
    }  
  
    return p;  
}
```

Cél: Utasítás lefedettség, döntés lefedettség, C/DC fedettség

16

- Statement coverage:
 - name: null, cat: valid
 - name: valid, cat valid, product not in cache
 - Problem: getting from cache not tested
- Decision:
 - name: null, cat: valid
 - name: valid, cat valid, product not in cache
 - name: valid, cat valid, product in cache
 - Problem: cat.isValid call is not tested
- C/DC coverage:
 - name: null, cat: valid
 - name: valid, cat: invalid
 - name: valid. cat valid, product not in cache
 - name: valid. cat valid, product in cache

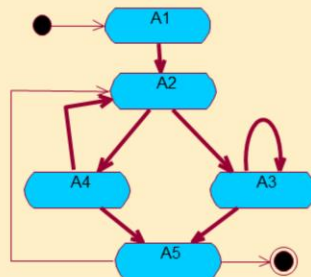
7. Alap út lefedettség (Basis path coverage)

Definíció:

$$\frac{\text{Tesztelés során bejárt független utak száma}}{\text{Összes független út száma}}$$

100% út lefedettség együtt jár:

- 100% utasítás lefedettség, 100% döntés lefedettség
- feltétel lefedettség nem garantált



Út fedettség: 80%

Utasítás fedettség: 100%

17

A teljes út lefedettség (full path coverage) a ciklusok miatt általában nem praktikus.

Tesztelés az út fedettség alapján 1/2

- **Cél: Független utak bejárása**
 - Független utak a tesztelés szempontjából:
Van olyan utasítás vagy elágazás,
ami a másikban nincs meg
- **A független utak maximális száma:**
 - CK , ciklomatikus komplexitás
 - Szabályos vezérlési gráf alapján meghatározható:
 $CK(G)=E-N+2$, ahol
 - E : élek száma
 - N : csomópontok száma a G vezérlési gráfban
(vezérlési gráf összekötött, 1 kimenet és 1 bemenet)
- **A független utak halmaza nem egyedi**

Tesztelés az út fedettség alapján 1/2

- **Cél: Független**

- Független utak
Van olyan utasítás ami a másikban

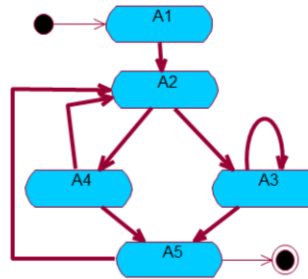
- **A független utak**

- CK, ciklomatikus
- Szabályos vezérlés
 $CK(G) = E - N + 2$, ahol

E: élek száma

N: csomópontok száma a G vezérlési gráfban
(vezérlési gráf összekötött, 1 kimenet és 1 bemenet)

- **A független utak halmaza nem egyedi**



N=5,
E=8,
CK=5
max. 5
független út!

Tesztelés az út fedettség alapján 2/2

- **Algoritmus:**
 - Max. CK számú független út kiválasztása
 - Bemenetek generálása egy-egy út bejárásához
- **Problémák:**
 - Nem minden út bejárható (ld. feltételek).
 - Generálható-e az úthoz bemeneti szekvencia?
 - Lehetséges-e a belső változók beállítása?
 - Ciklusok: Korlátozni (minimalizálni) kell a bejárást!
- **Teljesen automatikus módszerek nem léteznek**

Kiegészítő fedettségi mértékek

- Loop
 - Ciklusok 0 (ahol értelmezhető), 1, illetve többszöri végrehajtása
- Race
 - Több szál futása egyszerre egy-egy kódrészleten
- Relational operator
 - Határértékek használata összehasonlító operátorok esetén
- Weak mutation
 - Operátor vagy operandus hibákra tesztek készítése
- Table
 - Ugrási táblák (állapotgép megvalósítások) teljes tesztelése
- Linear code sequence and jump
 - Lineáris szekvenciák fedése a forráskódban (lehetnek benne vezérlési utasítások, de lineárisan bejárva)
- Object code branch
 - Gépi kódú feltételes ugrások fedése (fordító függő)

Tartalom

- Vezérlési folyamat alapú kritériumok
 - Utasítás lefedettség
 - Döntési ág lefedettség
 - Feltétel lefedettségek
 - Útvonal lefedettség
- Adatfolyam alapú kritériumok
 - Definiálás – használat fedettségek
 - Definíciómentes útvonalak fedettsége
- Összefoglalás
 - Módszerek kombinációja
 - Teszt fedettség mérése

Alapfogalmak

- Alapötlet: Változók értékadásának és felhasználásának viszonyát vizsgáljuk
 - Hibás értékadás, hibás felhasználás detektálható
- A programgráf címkézése
 - $\text{def}(v)$: adott helyen a v változónak értéket adunk
 - $\text{use}(v)$: adott helyen a v változót felhasználjuk
 - $\text{p-use}(v)$: v értékét feltételben használjuk
 - $\text{c-use}(v)$: v értékét számolásban használjuk
- A programgráf útvai
 - Definíciómentes út (definition clear path):
 v -nek nem adunk értéket az út csomópontjaiban

23

p-use: predicate use

c-use: computation use

Példa a címkézésre

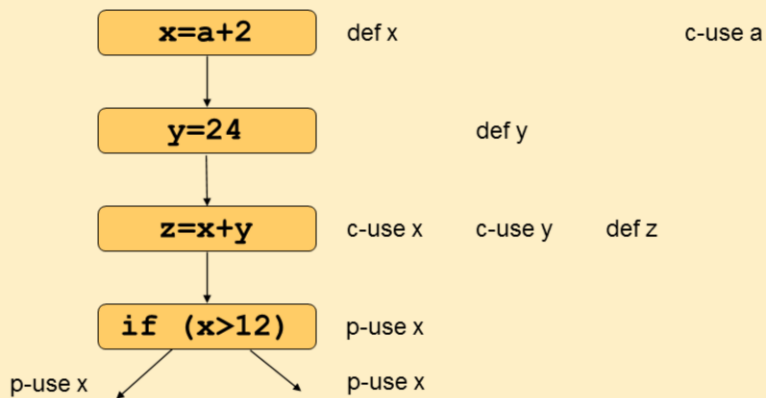
Változók:

x

y

z

a



Adatfolyam alapú tesztelési kritériumok

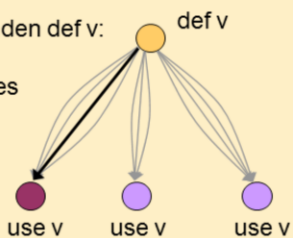
- All-defs:

- def v
- use v

minden v, minden def v:

egy def-mentes
út tesztelve:

egy use-v-ig:



Adatfolyam alapú tesztelési kritériumok

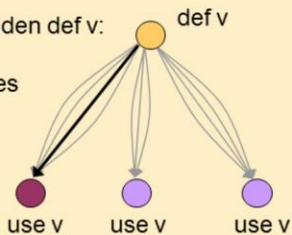
- All-defs:

- def v
- use v

minden v, minden def v:

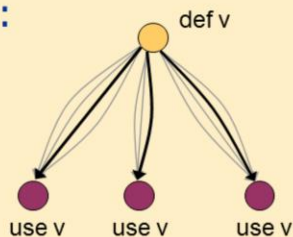
egy def-mentes
út tesztelve:

egy use-v-ig:

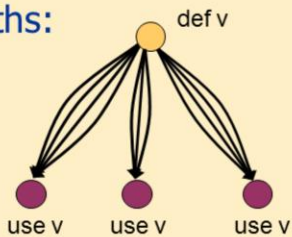


- All-uses:

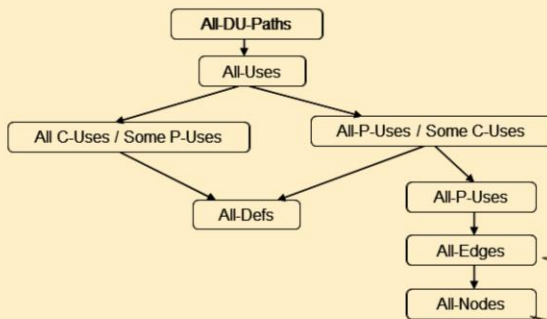
(p-uses,
c-uses)



- All-paths:



Strukturális fedettségi kritériumok viszonya



Bonyolultabb fedési kritériumok a biztonságkritikus szabványokban

Ezt már egy igényesebb tesztelési cél

A projektek nagy része ezt sem éri el, 90%-os lefedettség már egy jó cél

Tartalom

- Vezérlési folyamat alapú kritériumok
 - Utasítás lefedettség
 - Döntési ág lefedettség
 - Feltétel lefedettségek
 - Útvonal lefedettség
- Adatfolyam alapú kritériumok
 - Definiálás – használat fedettségek
 - Definíciómentes útvonalak fedettsége
- **Összefoglalás**
 - Módszerek kombinációja
 - Teszt fedettség mérése

Teszttervezési módszerek összefoglalása

- Specifikáció és struktúra alapú módszerek
 - Sokféle módszer illetve technika
 - Mindegyik alkalmazásához gyakorlat kell
- A gyakorlatban általában csak a legegyszerűbb módszereket használják
 - Biztonságkritikus rendszerek: Vannak előírt módszerek (pl. DO178B szabvány: MC/DC szerinti tesztelés)
- Technikák kombinációja hatásos általában:
 - Példa (MS tanulmány):
 - specifikáció alapú: 83%-os kódfedés
 - + exploratory: 86%-os kódfedés
 - + strukturális: 91%-os kódfedés

Mire jók a teszt fedettségi mértékek?

- Mire jók?
 - Megtalálhatók azok a programrészek, ahol hiányos a tesztelés
 - Ez alapján bővíthető a teszt készlet
 - Azonosíthatók a redundáns tesztek (azonos részeket fednek le)
 - Adatfüggésre is figyelni kell (más adattal más hibát tesztel)
 - **Indirekt** mértéke a kód minőségének a sikeres tesztekhez tartozó fedettség
 - Inkább mértéke a tesztkészlet teljességének
 - A tesztelés befejezése a mértékekhez köthető
- Mire nem jók?
 - Az implementációból kihagyott (nem megvalósított) követelmények tesztelése
 - Kódrészletek kiragadásával történő tesztelés eredményessége kérdéses (a kódrészlet elveszíti környezetét)

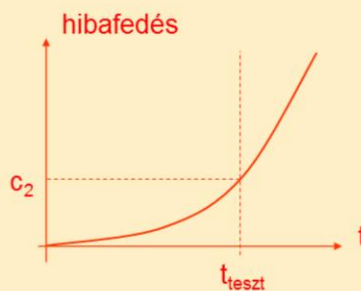
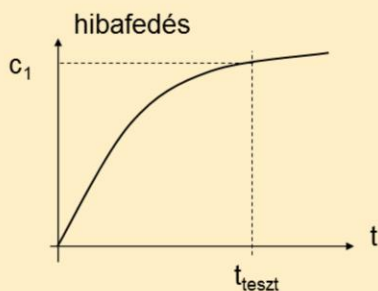
Tesztek végrehajtása

Milyen **sorrendben** hajtsuk végre a tesztek:

Ha várhatóan kevés a hiba:

Először a *hatékony* (nagyobb hibafedésű) tesztek!

- Hosszabb utak,
- összetett feltételű elágazások tesztje



Példa: Teszt fedettség mérése: gcov, ggcov

- gcov: gcc mellett használható
 - Korlátozott tudású (felülvizsgálat szükséges)
- Program felműszerezése: Speciális fordítás

```
gcc -ftest-coverage -fprofile-arcs src.c
```

 - Naplózza az egyes programágak végrehajtását
 - `-fprofile-arcs` esetén döntési ágakat is
 - Elkészülő log fájlok: `.bb`, `.bbg`, `.da`
- Off-line elemzés: gcov
 - `gcov src.c`

```
88.89% of 9 source lines executed in file cov.c
```
 - Annotált forráskód
- Grafikus jelentés: ggcov

Példa: gcov annotált forráskód

```
#include <stdio.h>
int main (void)
{
1   int i;

10  for (i = 1; i < 10; i++)
    {
9   if (i % 3 == 0)
3   printf ("%d is divisible by 3\n",i);
9   if (i % 11 == 0)
##### printf ("%d is divisible by 11\n",i);
9   }

1   return 0;
1  }
```

Példa: gccov elemzés

GGCov: Summary Overall

File Settings Windows Help

Overall

Filename: foo.c View...

Function: function_one View...

Range: foo.c View...

1 to 1

Lines 21/26 80.8%

Functions 3+1/5 60.0+20.0%

Calls 5/6 83.3%

Branches 5/5 100.0%

Blocks 16/19 84.2%

GGCov: File List

File View Settings Windows Help

File	Blocks	Lines	Functions	Calls	Branches
/home/gnb/proj/gccml2/	24.01	28.20	35.92	27.65	27.15
gtk	31.12	34.20	31.62	33.08	31.77
page_gui.c	64.83	64.95	64.29	65.69	73.91
tristate.c	51.19	53.23	71.43	48.84	53.57
node_gui.c	44.11	50.53	75.00	52.50	41.96
main.c	43.83	42.18	25.45	43.15	54.67
brokenwin.c	14.71	20.83	25.00	16.00	0.00
logwin.c	11.54	14.29	16.67	20.83	0.00
debug.c	7.14	7.46	10.00	7.55	9.52
choice.c	0.00	0.00	0.00	0.00	0.00
help.c	0.00	0.00	0.00	0.00	0.00
string.c	0.00	0.00	0.00	0.00	0.00
banner.c	0.00	0.00	0.00	0.00	0.00
unknown.c	0.00	0.00	0.00	0.00	0.00
limited_integer.c	0.00	0.00	0.00	0.00	0.00
integer.c	0.00	0.00	0.00	0.00	0.00
menu.c	0.00	0.00	0.00	0.00	0.00
libcml	21.12	25.95	38.40	23.34	26.33
debug.c	77.78	70.37	100.00	83.33	80.00
postparse.c	44.94	61.79	100.00	34.62	56.99

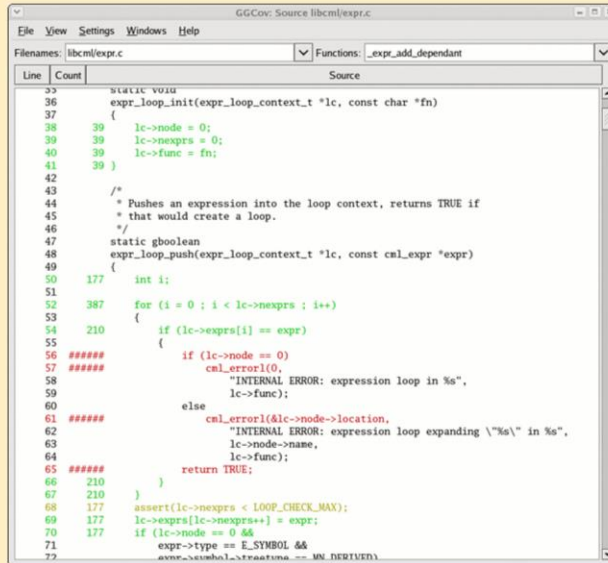
GGCov: Function List

File View Settings Windows Help

Blocks	Lines	Calls	Branches	Function
100.00	100.00	100.00		function_two
100.00	100.00			function_three
100.00	100.00	100.00	100.00	function_one
0.00	0.00			unused_function
77.78	90.00	66.67	100.00	main

Példa: gccov elemzés

- green - executed at least once
- yellow - part of the line executed
- red - never executed
- blue - suppressed
- black - no executable code



The screenshot shows a window titled "GGCov: Source libcm/expr.c". The window displays source code with a table of line numbers and counts. The code is color-coded according to the legend: green for lines executed at least once, yellow for lines partially executed, red for lines never executed, blue for suppressed lines, and black for non-executable code.

Line	Count
33	
36	
37	
38	39
39	39
40	39
41	39
42	
43	
44	
45	
46	
47	
48	
49	
50	177
51	
52	387
53	
54	210
55	
56	#####
57	#####
58	
59	
60	
61	#####
62	
63	
64	
65	#####
66	210
67	210
68	177
69	177
70	177
71	
72	

```
static void
expr_loop_init(expr_loop_context_t *lc, const char *fn)
{
    38     lc->nnode = 0;
    39     lc->nexprs = 0;
    40     lc->func = fn;
    41 }
/*
 * Pushes an expression into the loop context, returns TRUE if
 * that would create a loop.
 */
static gboolean
expr_loop_push(expr_loop_context_t *lc, const cml_expr *expr)
{
    50     int i;
    51
    52     387 for (i = 0 ; i < lc->nexprs ; i++)
    53     {
    54         210 if (lc->exprs[i] == expr)
    55         {
    56             ##### if (lc->nnode == 0)
    57                 ##### cml_error(0,
    58                     "INTERNAL ERROR: expression loop in %s",
    59                     lc->func);
    60             else
    61                 ##### cml_error(&lc->nnode->location,
    62                     "INTERNAL ERROR: expression loop expanding \"%s\" in %s",
    63                     lc->nnode->name,
    64                     lc->func);
    65             ##### return TRUE;
    66         }
    67     }
    68     177 assert(lc->nexprs < LOOP_CHECK_MAX);
    69     177 lc->exprs[lc->nexprs++] = expr;
    70     177 if (lc->nnode == 0 &&
    71         expr->type == E_SYMBOL &&
    72         expr->symbol->trantime == MN_DERIVED)
```

IPL Cantata++ architektúra

