

Verificare și validare software

Testare white-box. Acoperire cu teste

<https://tinyurl.com/cursVVS>

Testarea white-box

Generează teste bazat pe *structura internă* a codului

Alte denumiri: glass box, clear box, open box

Terminologie alternativă

testare *comportamentală* (black-box) / *structurală* (white-box)

Comparație:

- black-box: la orice nivel / white-box: mai ales la nivel de modul
- white-box: modificarea codului $\Rightarrow$ modificarea testelor
- white-box: ușurează detectarea *erorilor de codare*,
dar nu poate detecta *erori de omisiune* (în cod sau specificații)

Structura unui program: terminologie

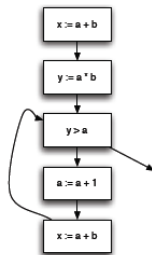
Graful de flux de control (control flow graph)

Reprezentare sub formă de graf
a structurii programului și implicit
a căilor sale de execuție

Noduri = instrucțiuni

Muchii = secvențierea (uneori condiționată) între instrucțiuni

```
x := a + b;  
y := a * b;  
while (y > a) {  
    a := a + 1;  
    x := a + b  
}
```



Uzual, instrucțiunile “în linie dreaptă”: grupate împreună $\Rightarrow$

basic block =

secvență de instrucțiuni cu un singur punct de intrare și de ieșire
(fără salturi în mijlocul codului, sau din mijloc afară)

Acoperirea prin teste: (code) coverage

= criteriu pentru a măsura *dacă un set de teste e adecvat*

La ce bun astfel de criterii ? [Burnstein]

Ce proprietăți ale programului să examinăm ?

Ce date de test selectăm pentru aceste proprietăți ?

Ce obiective *cantitative* stabilim pentru testare ?

S-a testat suficient sau nu ?

Idealul (... de neatins)

= testarea tuturor execuțiilor programului

⇒ adică a *tuturor căilor* prin graful de flux de control (CFG)

Dar: numărul căilor prin program e în general infinit (cicluri, etc.)

⇒ trebuie să ne multumim cu criterii structurale mai modeste

⇒ dar nu la întâmplare – trebuie alese *judicios*

Axiome ale testării (Weyuker)

Antiextensionalitate:

Există programe echivalente P și Q așa încât o anumită suită de teste T e adecvată pentru P dar nu pentru Q .

adică: programe echivalente *semantic* pot necesita teste diferite

Axiome ale testării (Weyuker)

Antiextensionalitate:

Există programe echivalente P și Q așa încât o anumită suită de teste T e adecvată pentru P dar nu pentru Q .

adică: programe echivalente *semantic* pot necesita teste diferite

General Multiple Change:

Există programe P și Q care au aceeași formă (structură) și o suită de teste T care e adecvată pentru P dar nu pentru Q .

adică: programe apropiate *sintactic* pot necesita teste diferite

Acoperirea la nivel de instrucțiune

statement coverage (line coverage, basic block coverage)

Teste suficiente pentru a executa *fiecare* instrucțiune de program

Criteriu evident *necesar* (cod neexecutat = cod netestat)

De regulă evident și *insuficient*

Acoperirea la nivel de instrucțiune

statement coverage (line coverage, basic block coverage)

Teste suficiente pentru a executa *fiecare* instrucțiune de program

Criteriu evident *necesar* (cod neexecutat = cod netestat)

De regulă evident și *insuficient*

```
char a[5], *s = NULL;
```

```
if (len < 5)
```

```
    s = a;
```

```
*s = 't';
```

Un test cu `len = 4` acoperă toate instrucțiunile; nu găsește eroarea

Acoperirea la nivel de ramificație

branch coverage (decision coverage)

Testează *fiecare* valoare posibilă a unei decizii (true/false)

Def. mai precisă: testează *și* fiecare intrare/ieșire din program
deseori, implică acoperirea instrucțiunilor
(fiecare instrucțiune e pe o ramură; v. *excepție* mai jos)

Acoperirea la nivel de ramificație

branch coverage (decision coverage)

Testează *fiecare* valoare posibilă a unei decizii (true/false)

Def. mai precisă: testează *și* fiecare intrare/ieșire din program deseori, implică acoperirea instrucțiunilor (fiecare instrucțiune e pe o ramură; v. *excepție* mai jos)

În categoria deciziilor / ramificațiilor se includ:
instrucțiuni switch/case (ramificații multiple)
generarea de *excepții* (greu de testat, des neglijată)
(fiecare excepție *posibilă* e un punct de ramificare)

Atenție: funcții sau efecte laterale în expresii de decizie

if (a && (b || f(x, y)))

nu execută f dacă a și b sunt adevărate

⇒ în general nu implică acoperirea instrucțiunilor

Acoperirea la nivel de condiție

condition coverage

O *condiție* e o *expresie booleană elementară* dintr-o *decizie*

Necesită teste pentru *fiecare* valoare posibilă a *fiecărei* condiții

Aparent mai complexă decât decision coverage, dar NU o include!

Acoperirea la nivel de condiție

condition coverage

O *condiție* e o *expresie booleană elementară* dintr-o *decizie*

Necesită teste pentru *fiecare* valoare posibilă a *fiecărei* condiții

Aparent mai complexă decât decision coverage, dar NU o include!

Exemplu:

```
if (x > 5 && y == 3) /*ceva cod */
```

Două teste: x = 6, y = 2 și x = 4, y = 3

generează toate valorile posibile (T și F, respectiv F și T)

Dar în ambele cazuri, decizia urmează ramura “fals”

Acoperirea mixtă decizie/condiție

condition/decision coverage

Necesită acoperirea simultană a *ambelor* criterii

Fața de cele două criterii individuale poate necesita teste suplimentare, sau doar recombinația lor

Acoperirea mixtă decizie/condiție

condition/decision coverage

Necesită acoperirea simultană a *ambelor* criterii

Fața de cele două criterii individuale poate necesita teste suplimentare, sau doar recombinația lor

de ex. pentru

```
if (x > 5 && y == 3) /*ceva cod */
```

se poate atinge tot cu două teste: $x = 6, y = 3$, și $x = 4, y = 2$

Poate fi insuficientă: efectul unor condiții le poate *masca* pe altele

Multiple condition coverage

Testează toate combinațiile posibile pentru *subexpresiile* deciziei

Multiple condition coverage

Testează toate combinațiile posibile pentru *subexpresiile* deciziei

Exponentială în numărul de condiții (2^n teste pentru n condiții)

⇒ adesea nefezabil de implementat

Practic, toate cele 2^n combinații

- pot fi irealizabile (când condițiile nu sunt independente)
- pot fi irelevante (limbaje cu evaluare în scurt-circuit)

⇒ în general, cerința nu se justifică

Modified Condition/Decision Coverage

Unul din cele mai puternice criterii; dezvoltat la Boeing, e o cerință pentru software în domeniul aeronautic (standard DO-178B)

Cerințele complete pentru o suită de teste MC/DC:

Toate punctele de intrare și ieșire din program atinse

Fiecare decizie exercitată pe **ambele** ramuri

Fiecare condiție ia **ambele** valori

***Fiecare** condiție e dovedită să **afecteze** decizia din care face parte (se păstrează fixe celălalte condiții, variind condiția de interes)*

Testele sunt aceleași, fie că limbajul are evaluare scurt-circuit sau nu.

Construirea unei suite de teste MC/DC

Pornim de la cazurile de bază, $\&\&$ și $||$ cu două condiții.

Operatorul $\&\&$ are un singur caz (tt) cu rezultat t .

Schimbând oricare din condiții în f , rezultatul devine f .

La fel pentru $||$ (operatorul dual), dar schimbând între ele t și f .

a	b	a && b			a	b	a b	
f	t	f	(1)	a: (1, 3) b: (2, 3)	t	f	t	(1)
t	f	f	(2)		f	f	t	(2)
t	t	t	(3)		f	f	f	(3)

Indicăm perechea de teste din suită relevantă pentru fiecare condiție:

(1, 3) arată că a poate influența decizia; la fel, (2, 3) pentru b.

Pentru n condiții: un test cu toate la fel, n teste cu câte una schimbată:

a	b	c	a && b && c		
f	t	t	f	(1)	a: (1, 4) b: (2, 4) c: (3, 4)
t	f	t	f	(2)	
t	t	f	f	(3)	
t	t	t	t	(4)	

Exemplu de construire a unei acoperiri MC/DC

Fie formula $a \ \&\& \ b \ \&\& \ (c \ || \ d \ \&\& \ e)$

Pornim de la expresia interioară, $d \ \&\& \ e$ (atenție la precedență!)

d	e	$d \ \&\& \ e$	
f	t	f	(1)
t	f	f	(2)
t	t	t	(3)

d: (1, 3)
e: (2, 3)

Adăugăm apoi $c \ ||$.

Cum $||$ cu f nu schimbă valoarea, adăugăm $c=f$ la toate testele (1-3).

Pentru testul (4), alegem un test cu f (2) și adăugăm $c=t$.

c	d	e	$c \ \ d \ \&\& \ e$	
f	f	t	f	(1)
f	t	f	f	(2)
f	t	t	t	(3)
t	t	f	t	(4)

Astfel arătăm și efectul lui c :
c: (2, 4)
d: (1, 3)
e: (2, 3)

Exemplu MC/DC (cont.)

Adăugăm acum $a \ \&\& \ b \ \&\&$. La testele anterioare, adăugăm $a=t, b=t$.
Apoi alegem un test cu t (4), și schimbăm pe rând a și b în f ,
arătând că atât a cât și b pot influența decizia:

a	b	c	d	e	$a \ \&\& \ b \ \&\& \ (c \ \ d \ \&\& \ e)$	
t	t	f	f	t	f	(1)
t	t	f	t	f	f	(2)
t	t	f	t	t	t	(3)
t	t	t	t	f	t	(4)
f	t	t	t	f	f	(5)
t	f	t	t	f	f	(6)

a: (4, 5)
b: (4, 6)
c: (2, 4)
d: (1, 3)
e: (2, 3)

Fiecare din perechile de teste au aceleași valori pentru toate celelalte condiții; condiția indicată e cea care singură influențează decizia.

Din construcție reiese direct că n variabile necesită $n + 1$ teste.

Acoperire MC/DC: exemplu 2

Fie formula $a \ \&\& \ b \ || \ c \ \&\& \ d$.

Scriem teste pentru cele două subexpresii (date de precedență):

a	b	$a \ \&\& \ b$		a: (1', 3')		c	d	$c \ \&\& \ d$	
f	t	f	(1')	b: (2', 3')		f	t	f	(1'')
t	f	f	(2')	c: (1'', 3'')		t	f	f	(2'')
t	t	t	(3')	d: (2'', 3'')		t	t	t	(3'')

Combinăm cu $||$. Cum $||$ cu f nu are efect, alegem câte un test cu f din fiecare grup (1' și 1'') și combinăm cu toate testele din celălalt grup.

a	b	c	d	$a \ \&\& \ b \ \ c \ \&\& \ d$	
f	t	f	t	f	(1=1'+1'') a: (1, 5)
f	t	t	f	f	(2=1'+2'') b: (4, 5)
f	t	t	t	t	(3=1'+3'') c: (1, 3)
t	f	f	t	f	(4=2'+1'') d: (2, 3)
t	t	f	t	t	(5=3'+1'')

Am păstrat astfel influența individuală a fiecărei condiții.

MC/DC în cazuri reale

Analiza anterioară e valabilă pentru *condiții independente*
e întotdeauna posibil să generăm testele proiectate

În realitate, condițiile pot fi *cuplate* (corelate)

Exemplu: $(z - x \geq 3 \ \&\& \ z - y \geq 1 \ || \ y < 5) \ \&\& \ x \leq 3$

MC/DC în cazuri reale

Analiza anterioară e valabilă pentru *condiții independente*
e întotdeauna posibil să generăm testele proiectate

În realitate, condițiile pot fi *cuplate* (corelate)

Exemplu: $(z - x \geq 3 \ \&\& \ z - y \geq 1 \ || \ y < 5) \ \&\& \ x \leq 3$

Pentru ca $z - x \geq 3$ să influențeze condiția ar trebui să avem:

$x \leq 3$, și $y \geq 5$, și $z - y \geq 1$

Dar de aici deducem $z - x \geq 3$, deci condiția nu poate fi falsă,
și deci nu poate influența decizia!

⇒ încercând o acoperire MC/DC putem detecta dacă o condiție e scrisă
prea complicat, sau are părți irelevante (o posibilă eroare de logică).

MC/DC în cazuri reale

Analiza anterioară e valabilă pentru *condiții independente*
e întotdeauna posibil să generăm testele proiectate

În realitate, condițiile pot fi *cuplate* (corelate)

Exemplu: $(z - x \geq 3 \ \&\& \ z - y \geq 1 \ || \ y < 5) \ \&\& \ x \leq 3$

Pentru ca $z - x \geq 3$ să influențeze condiția ar trebui să avem:

$x \leq 3$, și $y \geq 5$, și $z - y \geq 1$

Dar de aici deducem $z - x \geq 3$, deci condiția nu poate fi falsă,
și deci nu poate influența decizia!

⇒ încercând o acoperire MC/DC putem detecta dacă o condiție e scrisă
prea complicat, sau are părți irelevante (o posibilă eroare de logică).

În acest caz, cum $z - x < 3$ nu poate avea efect (chiar dacă apare),
condiția se poate rescrie înlocuind $z - x \geq 3$ cu *true*:

$(z - y \geq 1 \ || \ y < 5) \ \&\& \ x \leq 3$

Unique-Cause MC/DC vs. Masking MC/DC

Unique-Cause MC/DC

varianta prezentată inițial, influența unei condiții trebuie demonstrată păstrând toate celelalte la fel
uneori imposibil pentru condiții cuplate

Masking MC/DC

variantă relaxată:
în perechea de teste, nu toate celelalte condiții trebuie să fie la fel,
dar ambele combinații trebuie să evidențieze efectul condiției analizate

Combinație

cauză unică pentru toate condițiile independente
masking pentru condițiile cuplate

Acoperirea bazată pe predicate

predicate(-complete) coverage, [T. Ball, 2004]

Criteriile anterioare: *NU* corelează între ele deciziile.

⇒ ex. combinații a două decizii succesive în program

⇒ necesar un criteriu mai apropiat de *path coverage*
(care ar acoperi toate căile de execuție)

Acoperirea bazată pe predicate

predicate(-complete) coverage, [T. Ball, 2004]

Criteriile anterioare: *NU* corelează între ele deciziile.

⇒ ex. combinații a două decizii succesive în program

⇒ necesar un criteriu mai apropiat de *path coverage*
(care ar acoperi toate căile de execuție)

Se identifică n *predicate* (condiții) relevante în program

Se încearcă generarea tuturor combinațiilor posibile din cele $S \cdot 2^n$
(S stări = locații în program, n predicate)

⇒ corelează între ele *toate* condițiile și locațiile din program

Predicate coverage: exemplu [T. Ball]

```
void partition(int a[], int n) { // assume(n>2);
    int pivot = a[0];
    int lo = 1, hi = n-1;
    while (lo <= hi) {
        while (a[lo] <= pivot)
            lo++;
        while (a[hi] > pivot)
            hi--;
        if (lo < hi)
            swap(a,lo,hi);
    }
}
```

E corect ? Detectați o eroare ?

Predicate relevante: condițiile din program:

$lo \leq hi$, $lo < hi$, $a[lo] \leq pivot$, $a[hi] > pivot$

Criterii de acoperire pentru cicluri

[Beizer, *Software Testing Techniques*]

Pentru cicluri simple:

- zero iterații (nu se intră în ciclu)
suplimentar: contor negativ: se comportă corect ?
- o iterație
- două iterații (poate prinde *erori de inițializare*)
- o valoare tipică intermediară
- N-1 iterații
- N iterații
- se încearcă N+1 iterații (mai mult decât nr. maxim presupus)

Pentru minim nonzero: se încearcă min-1, min, min+1 ...

Acoperire pentru cicluri multiple [Beizer]

1. număr minim de iterații exterioare
testați complet ciclul interior (ca și ciclu independent)
2. continuă mergând în exterior pentru cicluri:
 - cu ciclurile interioare la o valoare tipică
 - și variind ciclul curent
3. în final, variază simultan toate ciclurile de la min la max

Alte criterii legate de căi (path testing)

Boundary interior path testing

- toate căile care parcurg un ciclu o dată, fără repetiție (boundary test)
- toate căile care repetă un test, cel mult de două ori (interior test)

Linear Code Sequence and Jump (LCSAJ)

o secvență LCSAJ: cod liniar urmat de un salt

criteriu LCSAJ de lungime N : N astfel de secvențe consecutive

$N = 1$ asigură acoperirea instrucțiunilor; $N = 2$ cea a ramificațiilor (chiar mai mult)

Acoperirea prin mutații

Se încearcă modificarea deciziilor / instrucțiunilor după anumite tipare, pentru a se detecta dacă programul *funcționează diferit*

Acoperirea prin mutații

Se încearcă modificarea deciziilor / instrucțiunilor după anumite tipare, pentru a se detecta dacă programul *functionează diferit*

Exemple:

- $<$ modificat în $\leq$, etc.
- $+1$ modificat în -1 sau ignorat
- limite modificate cu ± 1
- $a \ || \ b$ modificat în a , resp. b
(e relevant testul eliminat?); la fel pentru $a \ \&\& \ b$

Acoperirea prin mutații

Se încearcă modificarea deciziilor / instrucțiunilor după anumite tipare, pentru a se detecta dacă programul *funcționează diferit*

Exemple:

- $<$ modificat în $\leq$, etc.
- $+1$ modificat în -1 sau ignorat
- limite modificate cu ± 1
- $a \ || \ b$ modificat în a , resp. b
(e relevant testul eliminat?); la fel pentru $a \ \&\& \ b$

Dacă vreo mutație nu e prinsă măcar de un test:

⇒ fie testele sunt insuficiente

⇒ fie programul e potențial gresit (sau are cod irelevant)

Criterii de acoperire pentru date

Criteriile de până acum: legate de *fluxul de control* al programului
O alternativă: criterii legate de *fluxul de date* (*dataflow coverage*)

Criterii de acoperire pentru date

Criteriile de până acum: legate de *fluxul de control* al programului
O alternativă: criterii legate de *fluxul de date* (*dataflow coverage*)

Noțiuni cheie:

- *definirea* unei variabile (**def**): un loc unde e atribuită
- *utilizarea* unei variabile (**use**): un loc unde e citită
(folosită într-o expresie, testată într-o condiție)
- diverse criterii de acoperire, ex.: all-defs, all-uses

def-use coverage: acoperire cu un caz de test pentru fiecare pereche
(fezabilă) de definire-utilizare

Cât de relevantă e acoperirea ?

- complexitatea ciclomatică a CFG-ului.

Def. într-un graf, complexitatea ciclomatică e $E - V + 2$
(E = nr. de muchii, V = nr. de noduri)

- măsură bună pt. complexitatea unui program
- dorim acoperire mai mare pt. fragmentele mai complexe