Strukturno testiranje: Tehnike zasnovane na kontroli toka

Šta znači „strukturno“ testiranje?

CRNA KUTIJA

Polazište: specifikacija

Koje ulaze i očekivane ishode treba proveriti?

BELA KUTIJA

Polazište: struktura koda

Koje naredbe i izlazi kontrolnih odluka su izvršeni?

Ove tehnike se dopunjuju: pokrivenost nije zamena za specifikaciju ni za oracle.

Podela tehnika strukturnog testiranja

Tehnike zasnovane na kontroli toka

Pokrivanje iskaza
Pokrivanje odluka
Pokrivanje uslova
Pokrivanje putanja

Tehnike zasnovane na toku podataka

Prate kretanje vrednosti od dodele promenljivoj do njenog korišćenja

Primer: cena sa popustom

Pretpostavimo da je ulazna cena validna.

```
int priceAfterDiscount(int price, boolean student) {
    int result = price;
    if (student) {
        result = price - 100;
    }
    return result;
}
```

Test T1

price = 1200, student = true

Očekivani rezultat: 1100

Test T2

price = 1200, student = false

Očekivani rezultat: 1200

Za sada pratimo samo putanju izvršavanja — logiku specifikacije ćemo pretpostaviti poznatom.

Test T1: šta se izvršava?

1
result = price
IZVRŠENO

2
if (student)
true

3
result = price - 100
IZVRŠENO

4
return result
IZVRŠENO

Sve naredbe na izabranoj putanji izvršene su bar jednom.

Ishod student = false nije isproban.

Važno: telo if-a se izvršilo, ali smo videli samo jedan ishod odluke.

Kriterijum 1 — pokrivanje iskaza

Svaki izvršivi naredbeni iskaz treba da bude izvršen bar jednom.

Za naš metod i test T1 (računamo i if-iskaz):

1	2	3	4
inicijalizacija	if-uslov	dodela popusta	return

Pokrivenost iskaza

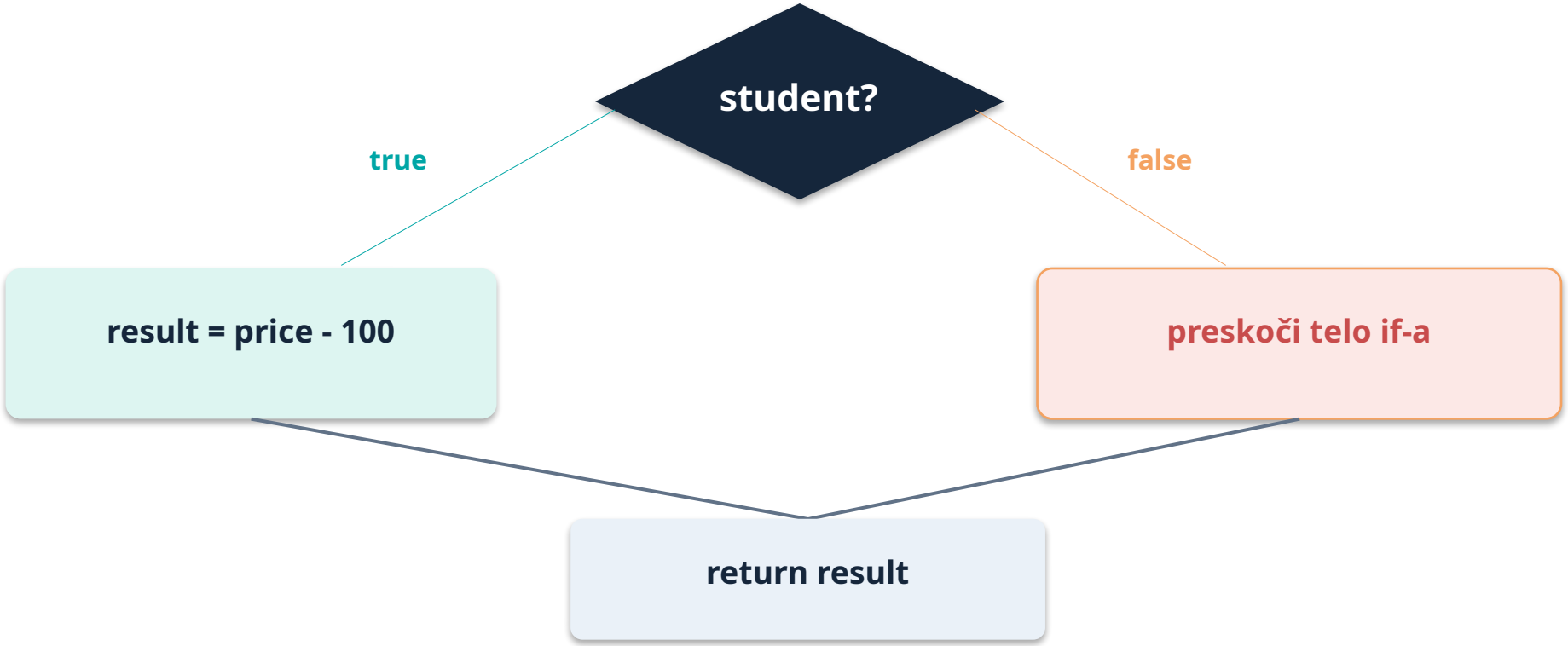
4 / 4 = 100%

T1 je dovoljan za ovaj kriterijum.

Ovaj rezultat govori da su naredbe dosegnute. Ne govori da su svi izlazi kontrolne odluke testirani.

Šta je T1 propustio?

Odluka if (student) ima dva moguća ishoda, ali T1 aktivira samo jedan.



T1 ne pokriva granu false.

Kriterijum 2 — pokrivanje odluka / grana

Svaki mogući ishod svake odluke mora se izvršiti bar jednom.

Za odluku if (student):

true — T1

false — T2

Oba ishoda zahtevaju najmanje dva testa.

Pokrivenost odluka

2 / 2 = 100%

Testovi: T1 (student=true) + T2 (student=false)

Pokrivanje odluka proverava ponašanje na izlazima uslovnih blokova, ne samo izvršavanje tela naredbe.

Poređenje kriterijuma

Kriterijum	Pitanje na koje odgovara	Primer sa T1
Iskazi	Da li je svaki izvršivi iskaz dosegnut?	100% — svi iskazi su izvršeni.
Odluke / grane	Da li je svaki ishod svake odluke izvršen?	50% — false ishod nije testiran.
Efektivnost	Da li test proverava očekivani rezultat?	Ne sledi iz procenta pokrivenosti.

100% pokrivenost odluka podrazumeva 100% pokrivenost iskaza. Obrnuto ne važi

Alat Java Code Coverage (JaCoCo): šta se neposredno meri

- Instruction coverage (C0): izvršene Java bajtkod instrukcije.
- Branch coverage (C1): ishodi `if` i `switch` odluka.
- Line coverage: izvorne linije povezane sa bajtkodom.
- Cyclomatic complexity: procena broja nezavisnih putanja iz CFG-a.

Važno ograničenje

JaCoCo ne prijavljuje MC/DC niti formalnu pokrivenost DU-lanaca.

Praktičan tok rada sa JaCoCo

Pogledati primer kao dodatak predavanju za maven konfiguraciju

```java
public class CoverageExample {

    public static int priceAfterDiscount(
                int price, boolean student) {
        int result = price;
        if (student) {
            result = price - 100;
        }
        return result;
    }
}
```

```java
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertEquals;

public class CoverageExampleTest {

    @Test
    public void testCena() {
        // Preslikane provere iz vaše prvobitne main metode
        CoverageExample ce = new CoverageExample();
        assertEquals(1100, ce.priceAfterDiscount(1200,true), "true branch");
        assertEquals(1200, ce.priceAfterDiscount(1200,false), "false branch");
        assertEquals(0, ce.priceAfterDiscount(100,true), "boundary");
    }
}
```

Praktičan tok rada sa JaCoCo

mvn clean test

CoverageExample

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
priceAfterDiscount(int, boolean)		100%		100%	0	2	0	4	0	1
CoverageExample()		100%		n/a	0	1	0	1	0	1
Total	0 of 13	100%	0 of 2	100%	0	3	0	5	0	2

CoverageExample.java

```
1.  public class CoverageExample {
2.
3.      public static int priceAfterDiscount(int price, boolean student) {
4.          int result = price;
5.          if (student) {
6.              result = price - 100;
7.          }
8.          return result;
9.      }
10. }
```

100% pokrivenost nije isto što i dobar test

SLAB TEST

```
@Test void discountTest() {
    priceAfterDiscount(1200, true);
    // rezultat nije proveren
}
```

Metod se izvrši, ali test može proći i kada je rezultat pogrešan.

BOLJI TEST

```
@Test void discountTest() {
    assertEquals(1100,
        priceAfterDiscount(1200, true));
}
```

Test proverava očekivani ishod — postoji oracle / asercija.

Pokrivenost je dokaz o izvršavanju. Asercije, izbor ulaza i specifikacija određuju šta će test otkriti (obezbeđuju efektivnost testiranja).

Kriterijumi testiranja u složenim uslovnim blokovima

- **Odluke i elementarni uslovi**

  Kako uslovi utiču na odluke

- **Dodatni kriterijumi**

  Pokrivanje uslova, kombinacija uslova i Modifed Condition Decision Coverage MC/DC.

- **Poređenje kriterijuma**

Elementarni uslovi

ODLUKA

Ishod celog uslovnog izraza

ELEMENTARNI USLOVI

Prosta logička promenljiva ili relacioni izraz koji se ne može dalje rastaviti na manje logičke celine

```
D = (godine > 18 AND poseduje_licnu_kartu == true)
```

D je odluka; Elementarni uslovi su: (godine > 18), (poseduje_licnu_kartu == true)

Zajednički primer: pravilo ulaska

Ulaz je dozvoljen ako je korisnik aktivan član ili ima propusnicu za gosta.

```
boolean canEnter(boolean member,
                 boolean active,
                 boolean guestPass) {
    return (member && active) || guestPass;
}
```

Mapiranje

A = member
B = active
C = guestPass
D = rezultat odluke



Kriterijum 1 — pokrivanje elementarnih uslova



Svaki elementarni uslov mora bar jednom biti true i bar jednom false.

Primer skupa testova (vrednosti ulaza):

T1

A=T, B=T, C=F

T2

A=F, B=F, C=T

Vrednosti po testovima

A: true i false
B: true i false
C: true i false

Uslovi pokriveni

U ovom kriterijumu merimo pojedinačne komponente izraza — ne zahtevamo da se konačna odluka D oba puta promeni.

Važna zamka: uslovi jesu pokriveni, odluka nije

Za izraz D = (A && B) || C razmotrimo dva testa iz prethodnog slajda.

Test	A	B	C	A && B	D
T1	T	T	F	T	T
T2	F	F	T	F	T

Rezultat

D = true
u oba testa

A, B i C su svaki jednom true i jednom false, ali testovi ne dostižu D = false.

Zaključak: pokrivanje uslova samo po sebi ne garantuje pokrivanje odluka.

Ni pokrivanje odluka ne podrazumeva pokrivanje uslova

Druga dva testa daju različit rezultat odluke, ali C nikada nije true.

Test	A	B	C	A && B	D
T1	F	F	F	F	F
T2	T	T	F	T	T

Rezultat

D = false
i true

Odluka D ima oba ishoda, ali C ima samo vrednost false.

Pokrivanje odluka i pokrivanje uslova mere različite osobine.

Kriterijum 2 — pokrivanje višestrukih uslova

Svaka moguća kombinacija vrednosti elementarnih uslova mora se pojaviti bar jednom.

Za n nezavisnih Booleovih uslova broj kombinacija je 2^n.

n = 3

$2^3 = 8$

test-kombinacija

Pokrivanje svih kombinacija uključuje i kombinacije koje možda ne mogu nastati u domenu aplikacije.

Prednost: iscrpno pokriva logičke kombinacije. Cena: broj testova raste eksponencijalno.

Primer: sve kombinacije za tri uslova

Za D = (A && B) || C, pun skup kombinacija ima osam redova.

A	B	C	A && B	D = (A && B) || C
F	F	F	F	**F**
F	F	T	F	**T**
F	T	F	F	**F**
F	T	T	F	**T**
T	F	F	F	**F**
T	F	T	F	**T**
T	T	F	T	**T**
T	T	T	T	**T**

Napomena: u ovoj tabeli A, B i C su nezavisni logički ulazi.

Zašto „sve kombinacije“ ponekad nije praktično?

```
if (ch == 'x' || ch == 'y') {
    ch = 'a';
}
```

Uslovi su međusobno zavisni: isti znak ne može istovremeno biti i x i y.

Moguća je kombinacija:

(ch == 'x') = T, (ch == 'y') = F

(ch == 'x') = F, (ch == 'y') = T

Ako se kriterijum traži doslovno, neizvodljive kombinacije moraju se identifikovati i obrazložiti.

Modified Condition/Decision Coverage MC/DC: svaki uslov mora nezavisno da utiče na odluku

Za svaki elementarni uslov potražimo par testova u kome se menja samo taj uslov, a rezultat odluke se promeni.

Test	A	B	C	D = (A && B) || C
T1	F	T	F	F
T2	T	T	F	T
T3	T	F	F	F
T4	T	F	T	T

Parovi koji pokazuju uticaj

A: T1 T2 (B,C ostaju isti)
B: T3 T2 (A,C ostaju isti)
C: T3 T4 (A,B ostaju isti)

Na ovaj način umesto 2^N testova, optimalno se dobija N+1

Poredjenje kriterijuma

Kriterijum	Šta zahteva	Šta NE garantuje sam po sebi
Pokrivanje uslova	Svaki uslov true i false	Oba ishoda odluke
Pokrivanje odluka	D svaki put true i false	Svaki uslov true i false
Sve kombinacije	Svaka kombinacija vrednosti uslova	Praktičnost; možda neizvodljive kombinacije
MC/DC	Svaki uslov nezavisno menja D	Ispravnost specifikacije i izlaznih provera

Izbor kriterijuma uslovljen podrškom alata za praćenje kriterijuma, standardima u određenoj industriji

Pregled alata

Alat	Koje kriterijume podržava	Ograničenja
JaCoCo	Java instrukcije, grane if/switch, linije i ciklomatska kompleksnost	Ne izveštava o MC/DC pokrivenosti niti o formalnim DU lancima
SonarQube	Metrike pokrivenosti linija, klasa, metoda, ciklomatska kompleksnost, duplirane linije, kohezija klasa	Neke metrike može uvesti iz spoljašnjih alata, na primer pokrivenost grana
PIT	Mutaciono testiranje Java bajtkoda, uz ishode kao što su killed, survived i no coverage	Ne zamenjuje analizu pokrivenosti i ne dokazuje da je svaki mutant relevantan
LLVM/Clang i GCC/gcov	Podrška za MC/DC u odgovarajućim C/C++ tokovima rada	Nisu Java alati
Testwell CTC++	Iskazi, odluke, uslove, višestruke kombinacije i MC/DC	Dostupnost pojedinačnih mera zavisi od konkretne konfiguracije i dodatnih komponenti
LDRA Testbed/TBvision	LCSAJ, MC/DC i dinamička analiza toka podataka, uz ostale strukturne mere	Za konkretan projekat treba proveriti podržani jezik, kompajler i konfiguraciju