Testarea Sistemelor Software

Proiect de Laborator - Testarea Functiei de Calcul a Taxei de Livrare

Disciplina: Testarea Sistemelor Software (TSS)

Autor: Sabău Eduard

Data: Decembrie 2025

Cuprins

1. Introducere si Cerinte
2. Descrierea Metodei Testate
3. Equivalence Partitioning (EP)
4. Boundary Value Analysis (BVA)
5. Cause-Effect Graphing (CEG)
6. Comparatie Code Coverage
7. Graful de Control si MC/DC
8. Analiza Mutantilor
9. Concluzii

---

1. Introducere si Cerinte

1.1 Obiectivul Proiectului

Acest proiect demonstreaza aplicarea diferitelor tehnici de testare software pe functia `calculateDeliveryFee(double distanceKm, double weightKg)`.

1.2 Cerinte Implementate

Nr.	Cerinta	Status
1a	Equivalence Partitioning	Implementat
1b	Boundary Value Analysis	Implementat
1c	Cause-Effect Graphing	Implementat
2	Code Coverage (JaCoCo)	Implementat
3	MC/DC cu graf orientat	Implementat
4	Mutant echivalent de ordinul 1	Implementat
5	Mutanti ne-echivalenti (killed/survived)	Implementat

1.3 Tehnologii Utilizate

- **Java 17** - Limbaj de programare
- **Maven** - Build tool si dependency management
- **JUnit 5** - Framework de testare
- **JaCoCo** - Code coverage
- **PITest** - Mutation testing

---

2. Descrierea Metodei Testate

2.1 Semnatura Metodei

```
public double calculateDeliveryFee(double distanceKm, double weightKg)
```

Parametri: - `distanceKm` - Distanta de livrare in kilometri (trebuie > 0) - `weightKg` - Greutatea pachetului in kilograme (trebuie > 0)

Return: Taxa totala de livrare in RON

Exceptii: `IllegalArgumentException` pentru valori invalide

2.2 Formula de Calcul

Taxa Totala = Taxa Baza + Taxa Distanta + Taxa Greutate

Sau matematic:

T_totala = T_B + T_D + T_G

Unde: - T_B = 5.00 RON (taxa fixa de baza) - T_D = Taxa variabila in functie de distanta - T_G = Taxa fixa per interval de greutate

2.3 Reguli pentru Taxa pe Distanta (T_D)

Interval Distanta	Formula	Tarif
(0, 10] km	d x 0.50	0.50 RON/km
(10, 50] km	d x 0.40	0.40 RON/km
(50, infinit) km	d x 0.30	0.30 RON/km

2.4 Reguli pentru Taxa pe Greutate (T_G)

Interval Greutate	Taxa Fixa
(0, 2] kg	0.00 RON
(2, 5] kg	4.50 RON

Interval Greutate	Taxa Fixa
(5, 15] kg	8.00 RON
(15, infinit) kg	15.00 RON

2.5 Implementarea

```
public class DeliveryService {

    private static final double BASE_FEE = 5.00;
    private static final double RATE_SHORT_DISTANCE = 0.50;     // 0-10 km
    private static final double RATE_MEDIUM_DISTANCE = 0.40;    // 10-50 km
    private static final double RATE_LONG_DISTANCE = 0.30;      // >50 km

    private static final double DISTANCE_THRESHOLD_SHORT = 10.0;
    private static final double DISTANCE_THRESHOLD_MEDIUM = 50.0;

    private static final double WEIGHT_FEE_LIGHT = 0.00;        // 0-2 kg
    private static final double WEIGHT_FEE_MEDIUM = 4.50;       // 2-5 kg
    private static final double WEIGHT_FEE_HEAVY = 8.00;        // 5-15 kg
    private static final double WEIGHT_FEE_VERY_HEAVY = 15.00; // >15 kg

    private static final double WEIGHT_THRESHOLD_LIGHT = 2.0;
    private static final double WEIGHT_THRESHOLD_MEDIUM = 5.0;
    private static final double WEIGHT_THRESHOLD_HEAVY = 15.0;

    public double calculateDeliveryFee(double distanceKm, double weightKg) {
        // Validare inputuri
        if (distanceKm <= 0 || weightKg <= 0) {
            throw new IllegalArgumentException(
                "Distanta si greutatea trebuie sa fie pozitive.");
        }

        // Calcul Taxa Distanta
        double distanceFee;
        if (distanceKm <= DISTANCE_THRESHOLD_SHORT) {
            distanceFee = distanceKm * RATE_SHORT_DISTANCE;
        } else if (distanceKm <= DISTANCE_THRESHOLD_MEDIUM) {
            distanceFee = distanceKm * RATE_MEDIUM_DISTANCE;
        } else {
            distanceFee = distanceKm * RATE_LONG_DISTANCE;
        }

        // Calcul Taxa Greutate
        double weightFee;
        if (weightKg <= WEIGHT_THRESHOLD_LIGHT) {
```

```
            weightFee = WEIGHT_FEE_LIGHT;
        } else if (weightKg <= WEIGHT_THRESHOLD_MEDIUM) {
            weightFee = WEIGHT_FEE_MEDIUM;
        } else if (weightKg <= WEIGHT_THRESHOLD_HEAVY) {
            weightFee = WEIGHT_FEE_HEAVY;
        } else {
            weightFee = WEIGHT_FEE_VERY_HEAVY;
        }

        // Taxa Totala
        return BASE_FEE + distanceFee + weightFee;
    }
}
```

---

3. Equivalence Partitioning (EP)

3.1 Principiul EP

Equivalence Partitioning imparte domeniul de intrare in **clase de echivalenta** - grupuri de valori care ar trebui sa fie tratate identic de catre program.

Ideea: Daca o valoare din partitie produce un rezultat corect, toate valorile din aceeasi partitie ar trebui sa produca rezultate corecte.

3.2 Partitii pentru Distanta (distanceKm)

Partitie	Domeniu	Reprezentant	Comportament
EP_D1	d <= 0	-5, 0	IllegalArgumentException
EP_D2	(0, 10]	5 km	Tarif 0.50 RON/km
EP_D3	(10, 50]	25 km	Tarif 0.40 RON/km
EP_D4	(50, infinit)	75 km	Tarif 0.30 RON/km

3.3 Partitii pentru Greutate (weightKg)

Partitie	Domeniu	Reprezentant	Comportament
EP_W1	w <= 0	-3, 0	IllegalArgumentException
EP_W2	(0, 2]	1 kg	Taxa = 0.00 RON
EP_W3	(2, 5]	3 kg	Taxa = 4.50 RON
EP_W4	(5, 15]	10 kg	Taxa = 8.00 RON
EP_W5	(15, infinit)	20 kg	Taxa = 15.00 RON

3.4 Diagrama Partitiilor

```
Distanta (km):
        invalid           scurt           mediu           lung
    |-------------|-------------|---------------|--------------->
    -inf          0             10              50             +inf
         EP_D1          EP_D2           EP_D3          EP_D4

Greutate (kg):
     invalid      usor       mediu       greu        foarte greu
    |-------|----------|----------|-----------|---------------->
    -inf    0          2          5           15             +inf
       EP_W1     EP_W2      EP_W3       EP_W4          EP_W5
```

3.5 Cazuri de Test EP

Test ID	Distanta (km)	Greutate (kg)	Rezultat Asteptat
EP_T1	-5	1	IllegalArgumentException
EP_T2	0	1	IllegalArgumentException
EP_T3	5	-3	IllegalArgumentException
EP_T4	5	0	IllegalArgumentException
EP_T5	5	1	7.50 RON
EP_T6	25	1	15.00 RON
EP_T7	75	1	27.50 RON
EP_T8	5	3	12.00 RON
EP_T9	5	10	15.50 RON
EP_T10	5	20	22.50 RON

3.6 Exemplu de Calcul Manual

Test EP_T5: d = 5 km, w = 1 kg

1. Taxa Baza: T_B = 5.00 RON
2. Taxa Distanta: 5 km x 0.50 = 2.50 RON (deoarece 5 <= 10)
3. Taxa Greutate: 0.00 RON (deoarece 1 <= 2)
4. **Total: 5.00 + 2.50 + 0.00 = 7.50 RON**

3.7 Implementare Test EP

```java
@Test
@DisplayName("EP_T5: d=5km (scurt), w=1kg (usor) -> 7.50 RON")
void testEP_T5_ShortDistance_LightWeight() {
    double result = service.calculateDeliveryFee(5, 1);
    assertEquals(7.50, result, DELTA);
}
```

```java
@Test
@DisplayName("EP_T1: d=-5km (invalid) -> Exception")
void testEP_T1_InvalidDistance() {
    assertThrows(IllegalArgumentException.class,
            () -> service.calculateDeliveryFee(-5, 1));
}
```

---

4. Boundary Value Analysis (BVA)

4.1 Principiul BVA

Boundary Value Analysis testeaza valorile la **granitele** intervalelor, unde erorile sunt cele mai probabile ("off-by-one errors").

Regula: Pentru fiecare limita, testam: - Valoarea **sub** limita (limita - epsilon) - Valoarea **pe** limita (exact) - Valoarea **peste** limita (limita + epsilon)

4.2 Valori Limita pentru Distanta

Prag	Sub Limita	Pe Limita	Peste Limita
0 km	-0.01 (invalid)	0 (invalid)	0.01 (valid)
10 km	9.99 (0.50 RON/km)	10.0 (0.50 RON/km)	10.01 (0.40 RON/km)
50 km	49.99 (0.40 RON/km)	50.0 (0.40 RON/km)	50.01 (0.30 RON/km)

4.3 Valori Limita pentru Greutate

Prag	Sub Limita	Pe Limita	Peste Limita
0 kg	-0.01 (invalid)	0 (invalid)	0.01 (valid)
2 kg	1.99 (0.00 RON)	2.0 (0.00 RON)	2.01 (4.50 RON)
5 kg	4.99 (4.50 RON)	5.0 (4.50 RON)	5.01 (8.00 RON)
15 kg	14.99 (8.00 RON)	15.0 (8.00 RON)	15.01 (15.00 RON)

4.4 Diagrama Limitelor

```
Distanta (km):
                    PRAGURI
    -0.01  0  0.01          9.99 10 10.01        49.99 50 50.01
      |    |   |             |   |   |             |   |   |
```

```
  X    X   o------------>o   o   o---------->o   o   o------>
invalid                 scurt   mediu              lung

X = invalid (exception)
o = valid (calculeaza)
```

4.5 Cazuri de Test BVA (Selectie)

Test ID	d (km)	w (kg)	Rezultat
BVA_D1	-0.01	1.0	Exception
BVA_D2	0	1.0	Exception
BVA_D3	0.01	1.0	5.005 RON
BVA_D4	9.99	1.0	9.995 RON
BVA_D5	10.0	1.0	10.00 RON
BVA_D6	10.01	1.0	9.004 RON
BVA_D7	49.99	1.0	24.996 RON
BVA_D8	50.0	1.0	25.00 RON
BVA_D9	50.01	1.0	20.003 RON
BVA_W5	5.0	2.0	7.50 RON
BVA_W6	5.0	2.01	12.00 RON
BVA_W8	5.0	5.0	12.00 RON
BVA_W9	5.0	5.01	15.50 RON
BVA_W11	5.0	15.0	15.50 RON
BVA_W12	5.0	15.01	22.50 RON

4.6 Exemplu: Schimbare Tarif la d = 10 km

Test BVA_D5: d = 10.0 km, w = 1.0 kg - Conditie: 10.0 <= 10.0 este TRUE - Tarif: 0.50 RON/km - T_D = 10 x 0.50 = 5.00 RON - **Total: 5.00 + 5.00 + 0.00 = 10.00 RON**

Test BVA_D6: d = 10.01 km, w = 1.0 kg - Conditie: 10.01 <= 10.0 este FALSE - Tarif: 0.40 RON/km - T_D = 10.01 x 0.40 = 4.004 RON - **Total: 5.00 + 4.004 + 0.00 = 9.004 RON**

4.7 Implementare Test BVA

```java
@Test
@DisplayName("BVA_D5: d=10.0 km (pe limita) -> tarif scurt")
void testBVA_D5_OnBoundary() {
    // 10.0 <= 10 este TRUE -> tarif 0.50
    double result = service.calculateDeliveryFee(10.0, 1.0);
    assertEquals(10.00, result, DELTA);
}
```

```java
@Test
@DisplayName("BVA_D6: d=10.01 km (peste limita) -> tarif mediu")
void testBVA_D6_JustOverBoundary() {
    // 10.01 <= 10 este FALSE -> tarif 0.40
    double result = service.calculateDeliveryFee(10.01, 1.0);
    assertEquals(9.004, result, DELTA);
}
```

---

5. Cause-Effect Graphing (CEG)

5.1 Principiul CEG

Cause-Effect Graphing identifica relatiile dintre **cauze** (conditii de intrare) si **efecte** (rezultate), generand un **tabel de decizie** complet.

Avantaj: Asigura testarea TUTUROR combinatiilor valide de intrari.

5.2 Cauze Identificate

Cauza	Conditie	Descriere
C1	distanceKm <= 0	Distanta invalida
C2	weightKg <= 0	Greutate invalida
C3	0 < d <= 10	Distanta scurta
C4	10 < d <= 50	Distanta medie
C5	d > 50	Distanta lunga
C6	0 < w <= 2	Greutate usoara
C7	2 < w <= 5	Greutate medie
C8	5 < w <= 15	Greutate mare
C9	w > 15	Greutate foarte mare

5.3 Efecte Identificate

Efect	Descriere
E1	IllegalArgumentException
E2	Tarif distanta 0.50 RON/km
E3	Tarif distanta 0.40 RON/km
E4	Tarif distanta 0.30 RON/km
E5	Taxa greutate 0.00 RON
E6	Taxa greutate 4.50 RON
E7	Taxa greutate 8.00 RON
E8	Taxa greutate 15.00 RON
E9	Taxa totala calculata

5.4 Relatii Cauza-Efect

```
CAUZE                              EFECTE

C1 (d<=0) ----\
               >--OR--> E1 (Exception)
C2 (w<=0) ----/

C3 (0<d<=10) --------> E2 (0.50 RON/km)
C4 (10<d<=50) -------> E3 (0.40 RON/km)
C5 (d>50) -----------> E4 (0.30 RON/km)

C6 (0<w<=2) ---------> E5 (0.00 RON)
C7 (2<w<=5) ---------> E6 (4.50 RON)
C8 (5<w<=15) --------> E7 (8.00 RON)
C9 (w>15) -----------> E8 (15.00 RON)

(C3 OR C4 OR C5) AND (C6 OR C7 OR C8 OR C9) --> E9 (Total)
```

Constrangeri: - C3, C4, C5 sunt **mutual exclusive** (distanta e intr-o singura categorie) - C6, C7, C8, C9 sunt **mutual exclusive** (greutatea e intr-o singura categorie)

5.5 Tabel de Decizie Complet

Cazuri Invalide (Exception)

Test	C1	C2	Efecte	Exemplu (d, w)
T1	T	-	E1	(-5, 1)
T2	F	T	E1	(5, -1)

Cazuri Valide - Toate Combinatiile

Test	C3	C4	C5	C6	C7	C8	C9	Efecte	Exemplu
T3	T	-	-	T	-	-	-	E2, E5, E9	(5, 1)
T4	T	-	-	-	T	-	-	E2, E6, E9	(5, 3)
T5	T	-	-	-	-	T	-	E2, E7, E9	(5, 10)
T6	T	-	-	-	-	-	T	E2, E8, E9	(5, 20)
T7	-	T	-	T	-	-	-	E3, E5, E9	(25, 1)
T8	-	T	-	-	T	-	-	E3, E6, E9	(25, 3)
T9	-	T	-	-	-	T	-	E3, E7, E9	(25, 10)
T10	-	T	-	-	-	-	T	E3, E8, E9	(25, 20)
T11	-	-	T	T	-	-	-	E4, E5, E9	(75, 1)
T12	-	-	T	-	T	-	-	E4, E6, E9	(75, 3)

Test	C3	C4	C5	C6	C7	C8	C9	Efecte	Exemplu
T13	-	-	T	-	-	T	-	E4, E7, E9	(75, 10)
T14	-	-	T	-	-	-	T	E4, E8, E9	(75, 20)

Legenda: T = True, F = False, - = Don't care (mutual exclusiv)

Total teste CEG: 14 (2 invalide + 12 combinatii valide = 3 distante x 4 greutati)

5.6 Diferenta EP vs CEG

Aspect	EP	CEG
Abordare	Alege un reprezentant din fiecare partitie	Testeaza TOATE combinatiile
Nr. teste	4 + 5 = 9 (sau mai multe)	3 x 4 + 2 = 14
Combinatii acoperite	Partial	100%
Complexitate	Simpla	Mai complexa

---

6. Comparatie Code Coverage

6.1 Rezultate JaCoCo

Tehnica	Nr. Teste	Line Coverage	Branch Coverage
EP	15	100% (16/16)	100% (14/14)
BVA	38	100% (16/16)	100% (14/14)
CEG	27	100% (16/16)	100% (14/14)

6.2 Code Coverage vs Input Space Coverage

ATENTIE: Code Coverage 100% nu inseamna testare completa!

Metrică	Ce masoara
Code Coverage	% linii de cod executate
Input Space Coverage	% combinatii de intrari testate

6.3 Input Space Coverage

Tehnica	Combinatii Posibile	Combinatii Testate	Input Coverage
EP	20 (4 x 5)	15	75%
BVA	108 (9 x 12)	38	35%
CEG	14 (3 x 4 + 2)	14	**100%**

6.4 Analiza Comparativa

Criteriu	EP	BVA	CEG
Eficienta (teste necesare)	Excelenta	Scazuta	Buna
Detectare erori la limite	Slaba	**Excelenta**	Moderata
Detectare erori combinatii	Slaba	Slaba	**Excelenta**
Efort implementare	Scazut	Mediu	Ridicat

6.5 Concluzii

1. **EP** - Rapid dar incomplet
   - [+] Putine teste, coverage rapid
   - [-] Nu testeaza toate combinatiile
2. **BVA** - Excelent pentru limite
   - [+] Detecteaza erori “off-by-one”
   - [-] Multe teste, nu acopera combinatii
3. **CEG** - Cel mai complet
   - [+] 100% combinatii testate
   - [-] Mai complex de implementat

Recomandare: Combinatia **BVA + CEG** ofera acoperire completa.

---

7. Graful de Control si MC/DC

7.1 Ce este Graful de Control al Fluxului (CFG)?

Control Flow Graph este o reprezentare grafica a tuturor cailor de executie posibile intr-un program.

Componente: - **Noduri** = instructiuni sau blocuri de cod - **Muchii** = tranzitii intre instructiuni - **Decizii** = noduri cu ramificari (if/else)

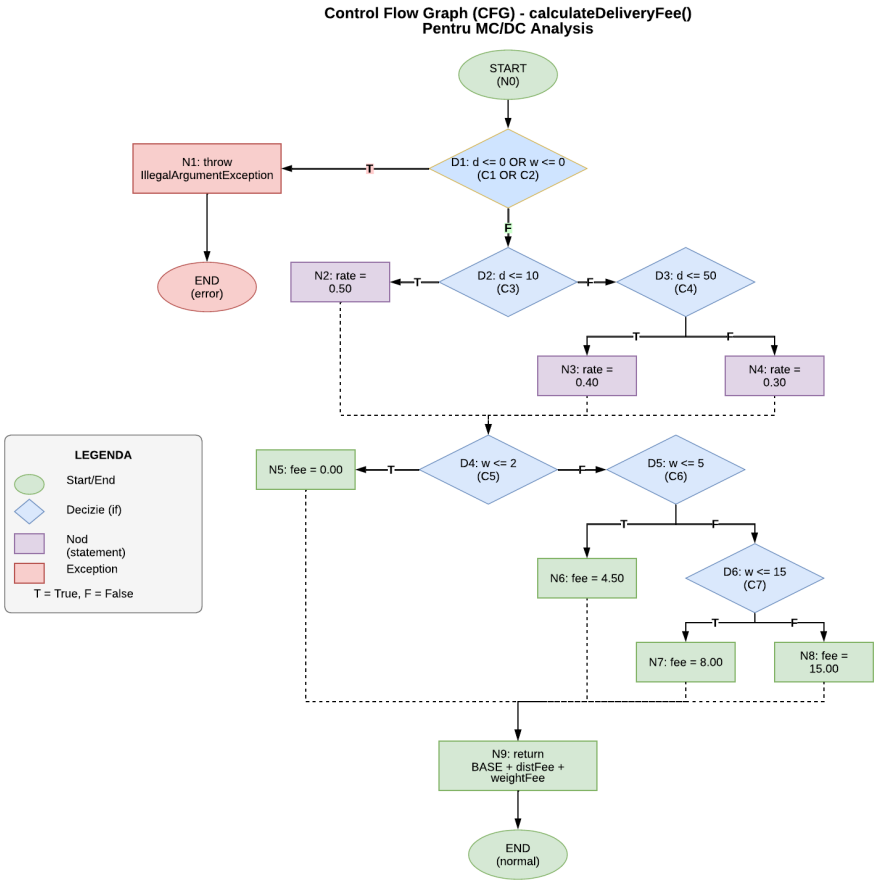

Figure 1: Control Flow Graph - CFG pentru calculateDeliveryFee()

7.2 Graful CFG pentru calculateDeliveryFee

Legenda: - **Elipse verzi** = START / END - **Romburi albastre** = Decizii (if statements) - **Dreptunghiuri mov** = Noduri de procesare - **Dreptunghiuri rosii** = Exception - **T / F** pe sageti = True / False branches

7.3 Identificarea Conditiilor

Decizie	Conditii	Tip
D1	C1: d <= 0 **OR** C2: w <= 0	Compusa
D2	C3: d <= 10	Simpla
D3	C4: d <= 50	Simpla
D4	C5: w <= 2	Simpla
D5	C6: w <= 5	Simpla
D6	C7: w <= 15	Simpla

7.4 Ce este MC/DC?

Modified Condition/Decision Coverage este un criteriu riguros de testare care cere:

1. Fiecare **punct de intrare si iesire** sa fie traversat
2. Fiecare **decizie** sa aiba rezultat TRUE si FALSE
3. Fiecare **conditie** din decizii compuse sa afecteze **independent** rezultatul

7.5 MC/DC pentru Decizia D1 (C1 OR C2)

Pentru operatorul **OR**, trebuie sa demonstram ca fiecare conditie poate schimba independent rezultatul:

Test	C1 (d<=0)	C2 (w<=0)	D1	Scop
M1	F	F	**F**	Baseline (ambele false)
M2	**T**	F	**T**	C1 schimba rezultatul
M3	F	**T**	**T**	C2 schimba rezultatul

Perechi pentru independenta: - **C1:** M1 vs M2 (C2=F constant, C1 schimba D1 de la F la T) - **C2:** M1 vs M3 (C1=F constant, C2 schimba D1 de la F la T)

Nota: Pentru OR, combinatia (T, T) nu este necesara pentru MC/DC minim.

7.6 Set Minim de Teste MC/DC

Test	d (km)	w (kg)	Conditii Acoperite	Rezultat
M1	5	1	C1=F, C2=F, C3=T, C5=T	7.50 RON
M2	-5	1	C1=T, C2=F	Exception
M3	5	-1	C1=F, C2=T	Exception
M5	25	1	C3=F, C4=T	15.00 RON
M6	75	1	C3=F, C4=F	27.50 RON
M8	5	3	C5=F, C6=T	12.00 RON
M9	5	10	C5=F, C6=F, C7=T	15.50 RON
M10	5	20	C5=F, C6=F, C7=F	22.50 RON

Total: 8 teste pentru MC/DC complet

7.7 Implementare Teste MC/DC

```
@Test
@DisplayName("MC/DC M2: C1=T demonstreaza independenta")
void testMCDC_M2_C1_True() {
    // C1=T (d<=0), C2=F (w>0)
    assertThrows(IllegalArgumentException.class,
            () -> service.calculateDeliveryFee(-5, 1));
}

@Test
@DisplayName("MC/DC M1: Baseline - ambele conditii false")
void testMCDC_M1_Baseline() {
    // C1=F (d>0), C2=F (w>0)
    double result = service.calculateDeliveryFee(5, 1);
    assertEquals(7.50, result, DELTA);
}
```

---

8. Analiza Mutantilor

8.1 Ce este Mutation Testing?

Mutation Testing evalueaza calitatea testelor prin:

1. Introducerea unor **modificari mici** (mutatii) in cod
2. Rularea testelor pe codul modificat (mutant)
3. Verificarea daca testele **detecteaza** mutatia

```
COD ORIGINAL                MUTANT
if (d <= 10)         ->    if (d < 10)    // Mutatie ROR
```

Rezultate posibile: - **KILLED** - Testele detecteaza mutatia (BUN!) - **SURVIVED** - Testele nu detecteaza mutatia (RAU - teste slabe) - **EQUIVALENT** - Mutatia nu schimba comportamentul (nu poate fi detectata)

8.2 Operatori de Mutatie Comuni

Operator	Nume	Exemplu
ROR	Relational Operator Replacement	`<=` -> `<`
AOR	Arithmetic Operator Replacement	`+` -> `-`
LCR	Logical Connector Replacement	`&&` -> `||`
SVR	Scalar Variable Replacement	`0.0` -> `0.0 * x`

8.3 Mutant 1: ECHIVALENT

Fisier: `DeliveryServiceMutantEquivalent.java`

Mutatia:

```
// Original:
weightFee = 0.0;

// Mutant:
weightFee = 0.0 * weightKg;  // 0 * orice = 0
```

De ce este echivalent: - Matematic: `0.0 * x = 0.0` pentru orice valoare x - Comportamentul este **identic** pentru toate inputurile - **Niciun test nu poate detecta diferenta**

Tip mutatie: Scalar Variable Replacement (SVR)

8.4 Mutant 2: NE-ECHIVALENT OMORAT (KILLED)

Fisier: `DeliveryServiceMutantKilled.java`

Mutatia:

```
// Original:
if (distanceKm <= 10.0)

// Mutant:
if (distanceKm < 10.0)  // <= schimbat in <
```

De ce NU este echivalent:

d (km)	Original (<=)	Mutant (<)
9.99	0.50 RON/km	0.50 RON/km
10.0	**0.50 RON/km**	**0.40 RON/km**

d (km)	Original (<=)	Mutant (<)
10.01	0.40 RON/km	0.40 RON/km

Test care omoara mutantul: BVA_D5 (d=10.0, w=1.0) - Original: 5.00 + 5.00 + 0.00 = **10.00 RON** - Mutant: 5.00 + 4.00 + 0.00 = **9.00 RON** - Assert esueaza -> **MUTANT KILLED**

8.5 Mutant 3: NE-ECHIVALENT SUPRAVIETUITOR (SURVIVED)

Fisier: `DeliveryServiceMutantSurvived.java`

Mutatia:

```
// Original:
if (distanceKm <= 0 || weightKg <= 0)

// Mutant:
if (distanceKm < 0 || weightKg <= 0)  // <= schimbat in <
```

De ce NU este echivalent:

d (km)	Original (<=)	Mutant (<)
-0.01	Exception	Exception
0	**Exception**	**Calculeaza!**
0.01	Calculeaza	Calculeaza

De ce supravietuieste:

Test EP (d=5, w=1): - Original: 7.50 RON - Mutant: 7.50 RON - Assert trece -> **MUTANT SURVIVED**

Cum l-am putea omori: Testul BVA_D2 (d=0, w=1) care testeaza explicit d=0.

8.6 Rezumat Mutanti

Mutant	Tip	Mutatia	Rezultat	Explicatie
Equivalent	Echivalent	`0.0 -> 0.0*w`	SURVIVED	OK - e identic matematic
Killed	Ne-echivalent	`d<=10 -> d<10`	KILLED	BVA_D5 detecteaza eroarea
Survived	Ne-echivalent	`d<=0 -> d<0`	SURVIVED	EP nu testeaza d=0 exact

8.7 Mutation Score

$$\text{Mutation Score} = \frac{\text{Mutanti omorati}}{\text{Total mutanti - Mutanti echivalenti}} \times 100\%$$

Interpretare: - 100% = Teste excelente - 80%+ = Teste bune - <60% = Teste slabe

---

9. Concluzii

9.1 Tehnici de Testare Utilizate

Tehnica	Tip	Scop Principal
EP	Black-box	Acoperire partitii
BVA	Black-box	Detectare erori la limite
CEG	Black-box	Acoperire combinatii
MC/DC	White-box	Acoperire conditii independente
Mutation	Evaluare	Calitatea testelor

9.2 Rezultate Obtinute

Metrică	Valoare
Code Coverage (JaCoCo)	100%
Branch Coverage	100%
Teste EP	15
Teste BVA	38
Teste CEG	27
Teste MC/DC	8
Mutanti creati	3
Mutanti omorati	1
Mutanti echivalenti	1

9.3 Ce am Invatat

1. **Code Coverage 100% nu garanteaza teste bune**
   - EP atinge 100% dar nu acopera toate combinatiile
2. **BVA este esential pentru functii cu praguri**
   - Detecteaza erori “off-by-one”
3. **CEG asigura acoperire completa a combinatiilor**
   - Singura tehnica care testeaza 100% combinatii
4. **MC/DC este riguroasa dar eficienta**

- Doar 8 teste pentru acoperire completa a conditiilor

5. **Mutation Testing evalueaza calitatea testelor**
   - Arata daca testele detecteaza erori mici

9.4 Recomandari Practice

Pentru testarea functiei `calculateDeliveryFee`:

1. **Prima etapa:** EP pentru verificare rapida
2. **A doua etapa:** BVA pentru testarea pragurilor
3. **A treia etapa:** CEG pentru combinatii complete
4. **Verificare:** Mutation testing pentru evaluare calitate

Combinatia ideala: BVA + CEG + MC/DC

Anexe

A. Structura Proiectului

```
tss-laborator/
+-- pom.xml
+-- README.md
+-- PREZENTARE.md
+-- diagrams/
|   +-- cfg-drawio.png
+-- src/main/java/ro/tss/delivery/
|   +-- DeliveryService.java
|   +-- DeliveryServiceMutantEquivalent.java
|   +-- DeliveryServiceMutantKilled.java
|   +-- DeliveryServiceMutantSurvived.java
+-- src/test/java/ro/tss/delivery/
    +-- EquivalencePartitioningTest.java
    +-- BoundaryValueAnalysisTest.java
    +-- CauseEffectGraphingTest.java
    +-- MCDCTest.java
    +-- MutantTest.java
```

B. Comenzi de Rulare

```
# Compilare
mvn compile

# Rulare toate testele
mvn test
```

```
# Rulare teste specifice
mvn test -Dtest=EquivalencePartitioningTest
mvn test -Dtest=BoundaryValueAnalysisTest
mvn test -Dtest=CauseEffectGraphingTest
mvn test -Dtest=MCDCTest
mvn test -Dtest=MutantTest

# Generare raport coverage
mvn clean test jacoco:report
# Raport: target/site/jacoco/index.html

# Mutation testing
mvn org.pitest:pitest-maven:mutationCoverage
# Raport: target/pit-reports/
```

C. Referinte

1. Myers, G. J., Sandler, C., & Badgett, T. (2011). *The Art of Software Testing*
2. Ammann, P., & Offutt, J. (2016). *Introduction to Software Testing*
3. JUnit 5 Documentation: https://junit.org/junit5/
4. JaCoCo Documentation: https://www.jacoco.org/jacoco/
5. PITest Documentation: https://pitest.org/

Sfarsit Document