



Univerzitet u Sarajevu  
Elektrotehnički fakultet u Sarajevu  
Odsjek za računarstvo i informatiku



**Verifikacija i validacija softvera  
Projektni zadatak - II dio**

Ime i prezime: Mina Duranović  
Broj indeksa: 19290  
Grupa: 5  
Datum: 14. 12. 2024.

Odabrana metoda:

```
4 references
public double ukupniMjesečniTroškovi(int korisnikId)
{
    Korisnik korisnik = null;
    for (int i = 0; i < _db.Users.Count; i++)
    {
        if (_db.Users[i].Id == korisnikId)
        {
            korisnik = _db.Users[i];
            break;
        }
    }

    if (korisnik == null)
    {
        throw new KorisnikNePostojiException("Korisnik ne postoji");
    }

    double iznos = 0;
    DateTime danasnjiDatum = DateTime.Now;
    DateTime pocetakMeseca = new DateTime(danasjniDatum.Year, danasnjiDatum.Month, 1);

    for (int i = 0; i < korisnik.troskovi.Count; i++)
    {
        Trosak trosak = korisnik.troskovi[i];
        if (trosak.Datum >= pocetakMeseca && trosak.Datum <= danasnjiDatum)
        {
            iznos += trosak.Iznos;
        }
    }

    return iznos;
}
```

## 1. stavka - McCabe metrika (ciklomatska kompleksnost)

McCabe metrika (graf kontrolnog toka je dat na dnu prve stavke):

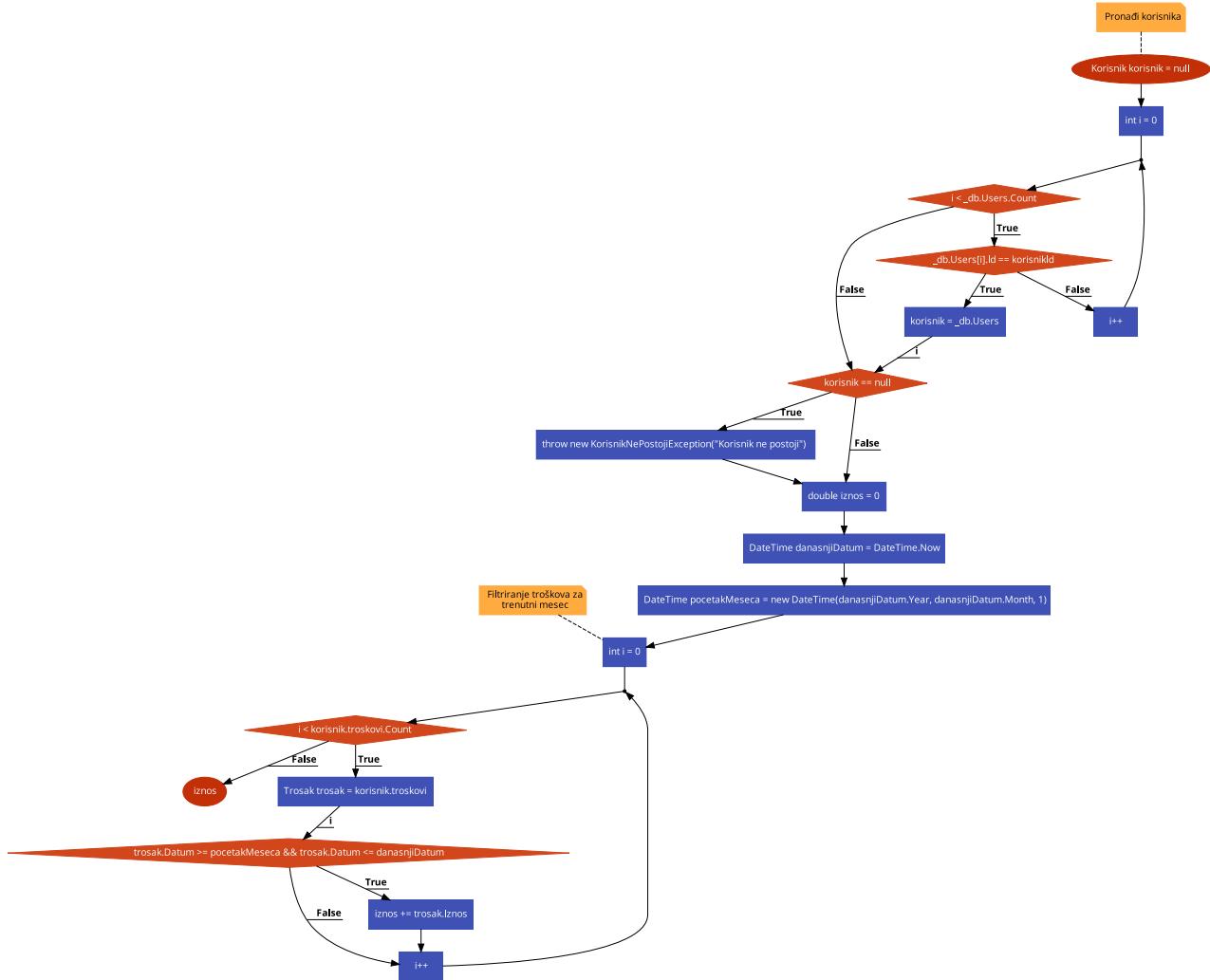
broj grana e = 24

broj čvorova n = 18

broj formiranih neovisnih dijelova grafa p = 1

$$\begin{aligned} M = V(G) &= e - n + 2p \\ &= 22 - 18 + 2 \\ &= 6 \end{aligned}$$

Kontrolni graf je dat u nastavku (generisani):

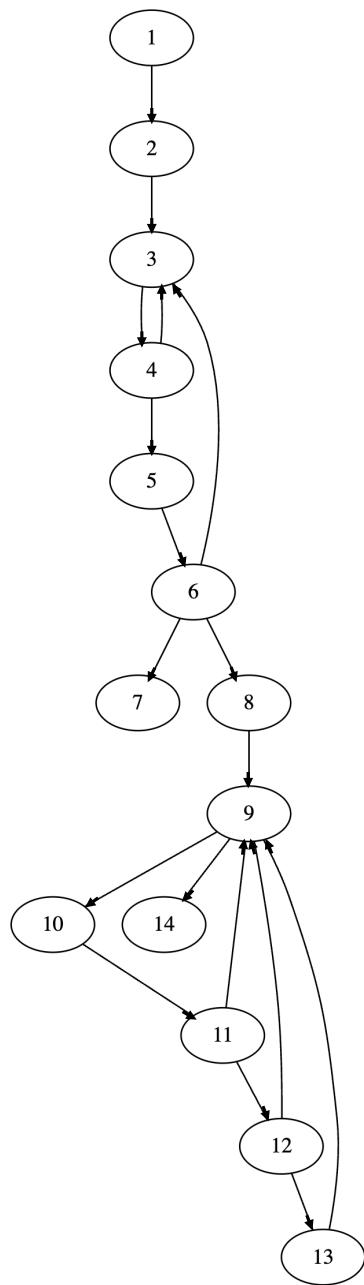


	Maintainability Index	Cyclomatic Complexity	Depth of Inheritance	Class Coupling	Lines of Source code	Lines of Executable code
Aplikacija za praćenje troškova (Debug)						
↳ Izvještaj	81	188	2	38	1,081	380
↳ Konverzija	66	12	1	14	69	25
↳ PreporukaServis	69	12	1	1	44	4
↳ Program	61	11	1	14	94	41
↳ Statistika	61	38	1	24	311	132
↳ db : DbClass	58	20	1	18	178	82
↳ Statistika(DbClass)	100	0		1	1	0
↳ ukupniMjesečniTroskovi(int) : double	96	1		1	4	1
↳ NajvećiTrosak(List<Trosak>) : double	54	7		7	32	15
↳ predikcijaOsnovnihTroskova(int, int) : double	75	3		2	9	4
↳ medijanTroskovaPoKategorijama(Korisnik) : Dictionary<Kategorija, StandardnaDjeljicijaTroskovaPoKategorijama(Korisnik) : Dictionary<Kategorija, ProsječniRastTroskova(int, int) : double	46	2		14	49	29
↳ StandardnaDjeljicijaTroskovaPoKategorijama(Korisnik) : Dictionary<Kategorija, ProsječniRastTroskova(int, int) : double	58	3		10	29	12
↳ Aplikacija_za_praćenje_troskova.Data	52	3		12	40	20
↳ Aplikacija_za_praćenje_troskova.Exceptions	100	1		0	4	1
↳ Aplikacija_za_praćenje_troskova.Models	95	4	1	2	17	2
↳ Aplikacija_za_praćenje_troskova.Services	96	3	2	1	12	0
TestProject1 (Debug)	76	110	1	21	207	59
				62	149	35
					1,277	365

## Rezultati iz CodeMetrics:

Rezultat se razlikuje za jedinicu u odnosu na naš proračun. Ciklomatska ili McCabeova kompleksnost od 7 ukazuje da metoda ima 7 nezavisnih putanja, tj. broj različitih načina na koje može da se izvrši metoda. Također možemo vidjeti da je indeks održivosti jednak 54 što ukazuje da je kod visoko održiv.

Graf kontrolnog toka:



## 2. stavka - White Box

Tabela puteva na osnovu grafa:

Redni broj puta	Čvorovi koje put obuhvata
1	1-2-3-4-5-6-7
2	1-2-3-4-5-6-8-9-10-11-12-13-9-14
3	1-2-3-4-5-6-8-9-10-11-12-9-14
4	1-2-3-4-5-6-8-9-10-11-9-14
5	1-2-3-4-5-6-8-9-14
6	1-2-3-4-5-6-3-4-5-6-8-9-10-11-12-13-9-14
7	1-2-3-4-5-6-3-4-5-6-8-9-10-11-12-9-14
8	1-2-3-4-5-6-3-4-5-6-8-9-10-11-9-14
9	1-2-3-4-5-6-3-4-5-6-7
10	1-2-3-4-3-4-5-6-7
11	1-2-3-4-3-4-5-6-8-9-10-11-12-13-9-14
12	1-2-3-4-3-4-5-6-8-9-10-11-12-9-14
13	1-2-3-4-3-4-5-6-8-9-10-11-9-14
14	1-2-3-4-3-4-5-6-8-9-14
15	1-2-3-4-3-4-5-6-3-4-5-6-8-9-10-11-12-13-9-14
16	1-2-3-4-3-4-5-6-3-4-5-6-8-9-10-11-12-9-14
17	1-2-3-4-3-4-5-6-3-4-5-6-8-9-10-11-9-14
18	1-2-3-4-3-4-5-6-3-4-5-6-7

itd.

- **Obuhvat iskaza/linija (Line coverage)**

Za obuhvat svih linija potrebna su nam 2 puta:

- Put 1 (1-2-3-4-5-6-7) - nije pronadjen korisnik, baca se izuzetak

```
55
56     [TestMethod]
57     [ExpectedException(typeof(KorisnikNePostojiException))]
58     0 references
59     public void KorisnikNePostoji_ThrowsException()
60     {
61         _statistika.ukupniMjesecniTroškovi(999); // Nevalidan ID korisnika
62     }
63 }
```

- Put 2 (1-2-3-4-5-6-8-9-10-11-12-13-9-14) pronadjen korisnik sa unijetim troškovima, vraća se validna suma

```

70  [TestMethod]
71  0 references
72  public void KorisnikPostoji_HasMonthlyExpenses_ReturnsSum()
73  {
74      double result = _statistika.ukupniMjesecniTroškovi(1); // Korisnik sa troškovima u tekućem mesecu
75      Assert.AreEqual(150, result);
76 }

```

### • Obuhvat uslova (Conditional coverage)

Obuhvat uslova (condition coverage) testira svaki uslov (condition) u true ili false stanju.Za obuhvat svih uslova potrebna su nam 4 puta:

Put 1 (1-2-3-4-5-6-7) korisnik nije pronađen, baca se izuzetak

Put 2 (1-2-3-4-5-6-8-9-10-11-12-13-9-14) pronađen korisnik sa unijetim troškovima, vraća sumu

Put 3 (1-2-3-4-5-6-8-9-10-11-9-14) pronađen korisnik, ali nije zadovoljen neki od uslova za datum troška

Pošto su 2 puta identični kao u prošlom pristupu, dodat ćemo samo još jedan test za provjeravanje kada je drugi uslov false.

```

77  [TestMethod]
78  0 references
79  public void KorisnikPostoji_HasNoValidMonthlyExpenses_ReturnsZero()
80  {
81      _db.Users[0].troškovi[0].Datum = DateTime.Now.AddMonths(-2);
82      double result = _statistika.ukupniMjesecniTroškovi(1); // Korisnik ima troškove, ali nijedan u datom mjesecu
83      Assert.AreEqual(0, result);
84 }
85

```

Modifikovani uslov/odluka obuhvat

Modified condition/decision coverage MCDC

Napraviti ćemo truth tabele za svaki od uslova:

```
if (_db.Users[i].Id == korisnikId)
```

Testni slučaj	<code>_db.Users[i].Id == korisnikId</code>	Ishod
1	FALSE	FALSE
2	TRUE	TRUE

Za prvi testni slučaj imamo već test u prethodnom pristupu gdje se baca exception za zadovoljeni uslov.

```
if (trosak.Datum >= pocetakMeseca && trosak.Datum <=
danasnjiDatum)
```

Testni slučaj	<code>trosak.Datum &gt;= pocetakMeseca</code>	<code>trosak.Datum &lt;= danasnjiDatum</code>	Ishod
1	FALSE	TRUE	FALSE
2	FALSE	FALSE	FALSE
3	TRUE	FALSE	FALSE
4	TRUE	TRUE	TRUE

U skladu sa tabelom dodajemo sljedeće testove:

```
0 references
98  public void DatumTroskaPrijePocetkaMjeseca_VratiZero()
99  {
100     _db.Users[0].troskovi[0].Datum = DateTime.Now.AddMonths(-1);
101
102     double rezultat = _statistika.ukupniMjesecniTroskovi(1);
103     Assert.AreEqual(0, rezultat);
104 }
105
```

```
105
106 [TestMethod]
107 0 references
108 public void DatumTroskaNakonDanasnjegDana_VratiZero()
109 {
110     _db.Users[0].troskovi[0].Datum = DateTime.Now.AddDays(1); // Datum u budućnosti
111
112     double rezultat = _statistika.ukupniMjesecniTroskovi(1);
113     Assert.AreEqual(0, rezultat);
114 }
```

- **Obuhvat petlji (Loop coverage)**

U funkciji postoje dvije petlje.

Za prvu for petlju formirat ćemo testne slučajeve.

- Preskočiti unutrašnjost petlje (`_db.Users.Count = 0`)
- Jedan prolazak kroz petlju (`_db.Users.Count = 1`)
- Dva prolaska kroz petlju (`_db.Users.Count = 2`)
- m prolazaka kroz petlju (`_db.Users.Count = 5`)
- n - 1 prolazaka kroz petlju (`drinks.Count = 7`)
- n prolazaka kroz petlju (`drinks.Count = 8`)
- n + 1 prolazaka kroz petlju nećemo i ne možemo testirati

U skladu sa tim, imamo sljedeće testove:

```
108  
109     [TestMethod]  
110     [ExpectedException(typeof(KorisnikNePostojiException))]  
111     0 references  
112     public void Test_UkupniMjesecniTroskovi_PreskokUnutrasnjosti()  
113     {  
114         _db.Users.Clear(); // Prazan skup korisnika  
115         var statistika = new Statistika(_db);  
116         statistika.ukupniMjesecniTroskovi(1); // Pokušaj pristupa korisniku koji ne postoji  
117     }  
118 }
```

```
109  
110     [TestMethod]  
111     0 references  
112     public void Test_Petlja_JedanProlaz()  
113     {  
114         _db.Users = new List<Korisnik>  
115         {  
116             new Korisnik(1, "Korisnik1", new List<Trosak>  
117             {  
118                 new Trosak(1, 50, DateTime.Now, Kategorija.Putovanje)  
119             }, new List<Prihod>(), new List<Budzet>(), new List<CiljStednje>(), 100)  
120         };  
121         var statistika = new Statistika(_db);  
122  
123         double ukupniTroskovi = statistika.ukupniMjesecniTroskovi(1);  
124  
125         Assert.AreEqual(50, ukupniTroskovi); // Trošak korisnika 1  
126     }  
127 }
```

```
137     [TestMethod]  
138     0 references  
139     public void Test_Petlja_DvaProlaza()  
140     {  
141         _db.Users = new List<Korisnik>  
142         {  
143             new Korisnik(1, "Korisnik1", new List<Trosak>  
144             {  
145                 new Trosak(1, 30, DateTime.Now, Kategorija.Putovanje)  
146             }, new List<Prihod>(), new List<Budzet>(), new List<CiljStednje>(), 100),  
147             new Korisnik(2, "Korisnik2", new List<Trosak>  
148             {  
149                 new Trosak(1, 40, DateTime.Now, Kategorija.Putovanje)  
150             }, new List<Prihod>(), new List<Budzet>(), new List<CiljStednje>(), 200)  
151         };  
152         var statistika = new Statistika(_db);  
153  
154         double ukupniTroskovi = statistika.ukupniMjesecniTroskovi(2); // Računamo trošak korisnika 2  
155  
156         Assert.AreEqual(40, ukupniTroskovi); // Trošak korisnika 2  
157     }  
158 }
```

```

161
162 [TestMethod]
163 0 references
164 public void Test_Petlja_MProlazaka()
165 {
166     _db.Users = new List<Korisnik>();
167     for (int i = 1; i <= 5; i++)
168     {
169         _db.Users.Add(new Korisnik(i, $"Korisnik{i}", new List<Trosak>
170         {
171             new Trosak(1, i * 10, DateTime.Now, Kategorija.Putovanje)
172         }, new List<Prihod>(), new List<Budzet>(), new List<CiljStednje>(), 100 * i));
173     }
174
175     var statistika = new Statistika(_db);
176
177     double ukupniTroskovi = statistika.ukupniMjesecniTroskovi(3); // Računamo trošak za korisnika 3
178
179     Assert.AreEqual(30, ukupniTroskovi); // Trošak korisnika 3
180 }

```

Isto ćemo uraditi i za drugu petlju:

```

[TestMethod]
0 references
public void Test_DrugaPetlja_PreskokUnutrasnjosti()
{
    _db.Users = new List<Korisnik>
    {
        new Korisnik(1, "Korisnik1", new List<Trosak>(), new List<Prihod>(), new List<Budzet>(), new List<CiljStednje>(), 100)
    };

    var statistika = new Statistika(_db);

    double ukupniTroskovi = statistika.ukupniMjesecniTroskovi(1);

    Assert.AreEqual(0, ukupniTroskovi); // Nema troškova
}

```

```

[TestMethod]
0 references
public void Test_DrugaPetlja_JedanProlaz()
{
    _db.Users = new List<Korisnik>
    {
        new Korisnik(1, "Korisnik1", new List<Trosak>
        {
            new Trosak(1, 50, DateTime.Now, Kategorija.Racuni) // Jedan trošak ovog mjeseca
        }, new List<Prihod>(), new List<Budzet>(), new List<CiljStednje>(), 100)
    };

    var statistika = new Statistika(_db);

    double ukupniTroskovi = statistika.ukupniMjesecniTroskovi(1);

    Assert.AreEqual(50, ukupniTroskovi); // Trošak iznos 50
}

```

```

[TestMethod]
0 references
public void Test_DrugaPetlja_DvaProlaza()
{
    _db.Users = new List<Korisnik>
    {
        new Korisnik(1, "Korisnik1", new List<Trosak>
        {
            new Trosak(1, 30, DateTime.Now, Kategorija.Racuni),
            new Trosak(2, 20, DateTime.Now, Kategorija.Zabava)
        }, new List<Prihod>(), new List<Budzet>(), new List<CiljStednje>(), 100)
    };

    var statistika = new Statistika(_db);

    double ukupniTroskovi = statistika.ukupniMjesecniTroskovi(1);

    Assert.AreEqual(50, ukupniTroskovi);
}

```

```

[TestMethod]
0 references
public void Test_DrugaPetlja_ViseProlaza()
{
    _db.Users = new List<Korisnik>
    {
        new Korisnik(1, "Korisnik1", new List<Trosak>
        {
            new Trosak(1, 10, DateTime.Now, Kategorija.Putovanje),
            new Trosak(2, 20, DateTime.Now, Kategorija.Zabava),
            new Trosak(3, 30, DateTime.Now, Kategorija.Stanarina),
            new Trosak(4, 40, DateTime.Now, Kategorija.Zabava),
            new Trosak(5, 50, DateTime.Now, Kategorija.Racuni)
        }, new List<Prihod>(), new List<Budzet>(), new List<CiljStednje>(), 100)
    };

    var statistika = new Statistika(_db);

    double ukupniTroskovi = statistika.ukupniMjesecniTroskovi(1);

    Assert.AreEqual(150, ukupniTroskovi);
}

```

```

[TestMethod]
0 references
public void Test_DrugaPetlja_NProlaza()
{
    _db.Users = new List<Korisnik>
    {
        new Korisnik(1, "Korisnik1", Enumerable.Range(1, 8).Select(i =>
            new Trosak(i, i * 10, DateTime.Now, Kategorija.Putovanje)).ToList(), new List<Prihod>(), new List<Budzet>(), new List<CiljStednje>(), 100)
    };

    var statistika = new Statistika(_db);

    double ukupniTroskovi = statistika.ukupniMjesecniTroskovi(1);

    Assert.AreEqual(360, ukupniTroskovi);
}

```

### 3. stavka - Code Tuning

- Tuning logičkih izraza

Možemo promijeniti for petlje sa while petljama čime se postiže veća fleksibilnost i čitljivost koda.

```
19 references
public double ukupniMjesečniTroskovi(int korisnikId)
{
    Korisnik korisnik = null;
    int j = 0;
    while (j < _db.Users.Count)
    {
        if (_db.Users[j].Id == korisnikId)
        {
            korisnik = _db.Users[j];
            break;
        }
        j++;
    }

    if (korisnik == null)
    {
        throw new KorisnikNePostojiException("Korisnik ne postoji");
    }

    double iznos = 0;
    DateTime danasnjiDatum = DateTime.Now;
    DateTime pocetakMeseca = new DateTime(danasjniDatum.Year, danasnjiDatum.Month, 1);

    int i = 0;
    while (i < korisnik.troskovi.Count)
    {
        Trosak trosak = korisnik.troskovi[i];
        if (trosak.Datum >= pocetakMeseca && trosak.Datum <= danasnjiDatum)
        {
            iznos += trosak.Iznos;
        }
        i++;
    }

    return iznos;
}
```

Napravili smo test koji će testirati performanse metode prije i poslije i rezultati su:

```
Vrijeme izvršavanja: 130 ms
Alocirana memorija: 24672 bajta
```

```
Vrijeme izvršavanja: 127 ms
Alocirana memorija: 24672 bajta
```

CodeMetrics nakon izgleda ovako:

Hierarchy	Maintainability Index	Cyclomatic Complexity	Depth of Inheritance	Class Coupling	Lines of Source code	Lines of Executable code	
Aplikacija za praćenje troškova (Debug)	82	188	1	29	1,087	384	
Izvjestaj	68	12	1	9	69	25	
Konverzija	69	12	1	2	44	4	
PreporukaServis	63	11	1	8	94	41	
Program	63	38	1	19	311	132	
Statistika	59	20	1	12	184	86	
db : DbClass	100	0	1	1	1	0	
Statistika(DbClass)	96	1	1	1	4	1	
ukupniMjesecniTroškovi(int) : double	53	7	1	8	36	19	
NaivesTrošak(List<Trošak>) : double	76	3	1	4	9	4	
predikcijaOsnovnihTroškova(int) : double	48	2	1	7	49	29	
medijanTroškovaPoKategorijama(Korisnik) : Dictionary<Kategorija, double>	60	3	1	6	29	12	
StandardnaDevijacijaTroškovaPoKategorijama(Korisnik) : Dictionary<Kategorija, double>	54	3	1	7	40	20	
ProsjecniRastTroškova(int, int) : double	100	1	1	0	4	1	
Aplikacija_za_praćenje_troškova.Data	98	4	1	3	17	2	
Aplikacija_za_praćenje_troškova.Exceptions	98	3	1	2	12	0	
Aplikacija_za_praćenje_troškova.Models	92	64	1	9	207	59	
Aplikacija_za_praćenje_troškova.Services	81	24	1	18	149	35	
TestProject1 (Debug)	77	126	1	37	1,550	419	

Vidimo da se indeks održavanja smanjio za jedan dok je ciklomatska kompleksnost ostala isti tako da se i nije neka velika promjena desila ovom tehnikom.

### • Tuning petlji - Unrolling petlje

Uraditi ćemo ovu tehniku nad našom metodom i zabilježiti rezultate, prije i poslije:

```
public double ukupniMjesecniTroškovi(int korisnikId)
{
    Korisnik korisnik = null;
    int i = 0;

    while (i < _db.Users.Count)
    {
        if (_db.Users[i].Id == korisnikId)
        {
            korisnik = _db.Users[i];
            break;
        }
        i++;
    }
    if (korisnik == null)
    {
        throw new Exception("Korisnik ne postoji");
    }
    double iznos = 0;
    DateTime danasjniDatum = DateTime.Now;
    DateTime pocetakMeseca = new DateTime(danasjniDatum.Year, danasjniDatum.Month, 1);

    int troškoviCount = korisnik.troskovi.Count;
    i = 0;
    int unrollStep = 4;
    while (i <= troškoviCount - unrollStep)
    {
        if (korisnik.troskovi[i].Datum >= pocetakMeseca && korisnik.troskovi[i].Datum <= danasjniDatum)
            iznos += korisnik.troskovi[i].Iznos;
        if (korisnik.troskovi[i + 1].Datum >= pocetakMeseca && korisnik.troskovi[i + 1].Datum <= danasjniDatum)
            iznos += korisnik.troskovi[i + 1].Iznos;
        if (korisnik.troskovi[i + 2].Datum >= pocetakMeseca & korisnik.troskovi[i + 2].Datum <= danasjniDatum)
            iznos += korisnik.troskovi[i + 2].Iznos;
        if (korisnik.troskovi[i + 3].Datum >= pocetakMeseca & korisnik.troskovi[i + 3].Datum <= danasjniDatum)
            iznos += korisnik.troskovi[i + 3].Iznos;

        i += unrollStep;
    }
    while (i < troškoviCount)
    {
        if (korisnik.troskovi[i].Datum >= pocetakMeseca & korisnik.troskovi[i].Datum <= danasjniDatum)
            iznos += korisnik.troskovi[i].Iznos;
        i++;
    }
    return iznos;
}
```

Vrijeme izvršavanja: 130 ms  
Alocirana memorija: 24672 bajta

Vrijeme izvršavanja: 128 ms  
Alocirana memorija: 24672 bajta

CodeMetrics nakon izgleda ovako:

Code Metrics Results						
Hierarchy	Maintainability Index	Cyclomatic Complexity	Depth of Inheritance	Class Coupling	Lines of Source code	Lines of Executable code
Aplikacija za praćenje troškova (Debug)	82	197	1	29	1,096	390
Izjestaj	68	12	1	9	69	25
Konverzija	69	12	1	2	44	4
PreporukaServis	63	11	1	8	94	41
Program	63	38	1	19	311	132
Statistika	57	29	1	13	193	92
_db : DbClass	100	0	1	1	1	0
Statistika (DbClass)	96	1	1	4	4	1
ukupniMjesecniTroškovi(int) : double	47	16	1	8	46	25
NajveciTrošak(List<Trošak>) : double	76	3	1	4	9	4
predikcijaSrednjihTroškova(int, int) : double	48	2	1	7	49	29
medijanTroškovaPoKategorijama(Korisnik) : Dictionary<Kategorija, double>	60	3	1	6	29	12
StandarnadnaDevijacijaTroškovaPoKategorijama(Korisnik) : Dictionary<Kategorija, double>	54	3	1	7	40	20
ProjektniRastTroškova(int, int) : double	100	1	1	0	4	1
Aplikacija za praćenje troškova.Data	98	4	1	3	17	2
Aplikacija_za_praćenje_troškova.Exceptions	98	3	1	2	12	0
Aplikacija_za_praćenje_troškova.Models	92	64	1	9	207	59
Aplikacija_za_praćenje_troškova.Services	81	24	1	18	149	35
TestProject1 (Debug)	77	126	1	37	1,559	419

Možemo primjetiti da se indeks održavanja smanjio sa 54 na 47 a ciklomatska kompleksnost povećala sa 7 na 16. Rezultati su se pogoršali tako da ne bi trebalo raditi CodeTuning ovom tehnikom.

## • LINQ optimizacija

LINQ (Language Integrated Query) je tehnologija u C#-u i .NET-u koja omogućava pisanje

```
0 references
public double UkupniMjesecniTroškovi(int korisnikId)
{
    var korisnik = _db.Users.FirstOrDefault(u => u.Id == korisnikId);

    if (korisnik == null)
    {
        throw new KorisnikNePostojiException("Korisnik ne postoji");
    }

    DateTime danasjniDatum = DateTime.Now;
    DateTime pocetakMeseca = new DateTime(danasjniDatum.Year, danasjniDatum.Month, 1);

    double iznos = korisnik.troskovi
        .Where(t => t.Datum >= pocetakMeseca && t.Datum <= danasjniDatum)
        .Sum(t => t.Iznos);

    return iznos;
}
```

upita za kolekcije podataka direktno u kodu na deklarativen i intuitivan način.

Iako smo očekivali da će se performanse koda poboljšati, desilo se suprotno:

Vrijeme izvršavanja: 130 ms  
Alocirana memorija: 24672 bajta

Vrijeme izvršavanja: 319 ms  
Alocirana memorija: 4045448 bajta

CodeMetrics nakon, izgleda ovako:

Hierarchy	Maintainability Index	Cyclomatic Complexity	Depth of Inheritance	Class Coupling	Lines of Source code	Lines of Executable code
Aplikacija za praćenje troškova (Debug)	82	185	1	30	1,083	381
Izjestaj	68	12	1	2	69	25
Konverzija	69	12	1	2	44	4
PreporukaServis	63	11	1	8	94	41
Program	63	38	1	19	311	132
Statistika	63	17	1	13	180	83
_DbClass	100	0	1	1	1	0
Statistika(DbClass)	96	1	1	1	4	1
@ukupniMjesecniTroškovi(int) : double	62	3	7	18	10	
@_PRVI_DAN_MJESECA : int	93	0	0	0	1	1
@ukupniMjesecniTroškovi(int) : double	73	1	3	10	5	
@NajveciTrošak(List<Trošak>) : double	76	3	4	9	4	
@predicijaOsnovnihTroškova(int, int) : double	48	2	7	49	29	
@medianTroškovapKategorijama(Korisnik) : Dictionary<Kategorija, double>	60	3	6	29	12	
@StandarnaDevacijaTroškovaPoKategorijama(Korisnik) : Dictionary<Kategorija, double>	54	3	7	40	20	
@ProjektniRastTroškova(int, int) : double	100	1	0	4	1	
@Aplikacija_za_praćenje_troškova.Data	98	4	1	3	17	2
@Aplikacija_za_praćenje_troškova.Exceptions	98	3	1	2	12	0
@Aplikacija_za_praćenje_troškova.Models	92	64	1	9	207	59
@Aplikacija_za_praćenje_troškova.Services	81	24	1	18	149	35
TestProject1 (Debug)	77	126	1	37	1,550	419

Možemo primjetiti da se indeks održavnja povećao sa 54 na 62, ciklomatska kompleksnost smanjila sa 7 na 3. Ovo su najbolji rezultati do sad tako da je ova tehnika CodeTuning najprimijerenija za optimizaciju metode.

## 4. stavka - Refaktoring

U refaktorisanju metode koristila sam sljedeće stavke iz **checkliste** sa predavanja:

- Rename a variable with a clearer or more informative name**

Varijabla korisnik preimenovana je u targetKorisnik da bi bolje opisala svoju svrhu (ciljni korisnik).

- Replace a magic number with a named constant**

Dan za prvi dan mjeseca zamijenjen je imenovanom konstantom PRVI\_DAN\_MJESECA.

- Replace an expression with a routine**

Logika za **pronalazak korisnika** izdvojena je u zasebnu metodu PronadjiKorisnika.

- Introduce an intermediate variable**

Dodana je varijabla trenutniMjesec za bolje razumijevanje.

- Extract a routine**

Logika za **računanje ukupnih troškova i provjeru datuma troškova** izdvojena je u metode IzracunajUkupneTroškoveZaMjesec i JeTrosakZaTrenutniMjesec.

Nakon refactoringa, metoda izgleda ovako:

```
1 reference
private const int PRVI_DAN_MJESECA = 1;

20 references
public double ukupniMjesecniTroškovi(int korisnikId)
{
    Korisnik targetKorisnik = PronadjiKorisnika(korisnikId);

    DateTime danasjniDatum = DateTime.Now;
    DateTime pocetakMjeseca = new DateTime(danasjniDatum.Year, danasjniDatum.Month, PRVI_DAN_MJESECA);
    string trenutniMjesec = $"{danasjniDatum.Month}/{danasjniDatum.Year}";

    return IzracunajUkupneTroškoveZaMjesec(targetKorisnik, pocetakMjeseca, danasjniDatum);
}
```

CodeMetrics nakon refactoringa izgleda ovako:

Hierarchy	Maintainability Index	Cyclomatic Complexity	Depth of Inheritance	Class Coupling	Lines of Source code	Lines of Executable code
Aplikacija za praćenje troškova (Debug)	82	190	1	29	1,099	381
Izvještaj	68	12	1	9	69	25
Konverzija	69	12	1	2	44	4
PreporukaServis	63	11	1	8	94	41
Program	63	38	1	19	311	132
Statistika	65	22	1	12	196	83
_db : DbContext	100	0		1	1	0
Statistika(DbClass)	96	1		1	4	1
PRVI_DAN_MJESECA : int	93	0		0	1	1
ukupniMjesecniTroškovi(int) : double	72	1		3	10	5
PronadjiKorisnika(int) : Korisnik	76	3		7	11	4
IzracunajUkupneTroškoveZaMjesec(Korisnik, DateTime, DateTime) : double	73	3		6	14	5
JeTrošakZaTrenutniMjesec(Trosak, DateTime, DateTime) : bool	89	2		2	4	1
NajveciTrosak(List<Trosak>) : double	76	3		4	9	4
predikcijaOsnovnihTroškova(int, int) : double	48	2		7	49	29
predikcijaOsnovnihTroškovaPoKategorijama(Korisnik) : Dictionary<Kategorija, double>	60	3		6	29	20
medijanTroškovaPoKategorijama(Korisnik) : Dictionary<Kategorija, double>	54	3		7	40	1
StandardnaDevijacijaTroškovaPoKategorijama(Korisnik) : Dictionary<Kategorija, double>	100	1		0	4	2
ProsječniRastTroškova(int, int) : double	98	4	1	3	17	0
Aplikacija_za_praćenje_troškova.Data	98	3	1	2	12	59
Aplikacija_za_praćenje_troškova.Exceptions	98	64	1	9	207	59
Aplikacija_za_praćenje_troškova.Models	92					

Možemo vidjeti, da se pored CodeTuninga, nakon refactoringa, indeks održavanja povećao na 72, a ciklomatska kompleksnost dodatno smanjila na 1. Tako zaključujemo da smo dobili najbolje moguće rezultate metrika, što je i bio krajnji cilj postići.