

Organizovanje pravolinijskog koda

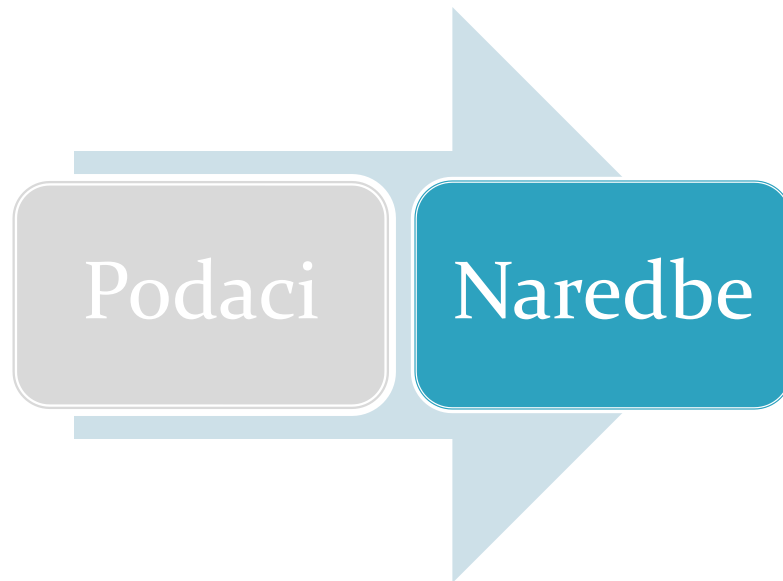
Lazar Mrkela

Organizing Straight-Line Code
Chapter 14, Code Complete by Steven C. McConnell



Tema poglavlja

- Fokus se pomera sa podataka na naredbe



Najjednostavniji vid
kontrole toka
izvršavanja

- Sekvencijalno organizovanje pojedinačnih naredbi i blokova naredbi

Pravolinijski kod

Jeste

```
1: int x;  
2: int y;  
3: double z;  
4: z=(x+y)/2.0;  
5: printf("%f",z);
```

Nije

```
1: int x;  
2: scanf("%d",&x);  
3: if (x % 2 == 0)  
4:     printf("Paran");  
5: else  
6:     printf("Neparan");
```

Uticaj organizovanja koda

Kvalitet

Tačnost

Čitljivost

Održavanje

Vrste naredbi

Poredak

- Bitan
- Nebitan

Bitan poredak - dobar kod

```
1: podaci=Ucitaj();  
2: rezultat=Izracunaj(podaci);  
3: Stampaj(rezultat);
```



- ▶ Naredbe moraju da se izvrše u prikazanom redosledu
- ▶ Postoje zavisnosti između operacija
- ▶ Zavisnosti se lako prepoznaju preko naziva funkcija

Bitan poredak - loš kod

```
1: prihod.IzracunajMesečne();  
2: prihod.IzracunajKvartalne();  
3: prihod.IzracunajGodisnje();
```



- ▶ Prikazane zavisnosti među funkcijama ne možemo lako da uočimo samim čitanjem koda

Bitan poredak - loš kod

```
1: IzracunajTroskoveMarketinga  
2: IzracunajTroskoveProdaje  
3: IzracunajTroskovePutovanja  
4: IzracunajTroskoveOsoblja  
5: PrikaziUkupanTrosak
```

- ▶ Zavisnosti su skrivene
- ▶ Dodatna informacija da metod *IzracunajTroskoveMarketinga* inicijalizuje objekat klase u koji sve ostale metode smeštaju svoje podatke nam govori da ona mora da ide prva
- ▶ Sam kod ne sugeriše da je ovo obavezan raspored naredbi

Čitljivost zavisnosti

- ▶ Kada postoje zavisnosti između naredbi koje zahtevaju određeni redosled izvršavanja, kod treba pisati tako da se te zavisnosti jasno vide

Koraci

- Organizovanje koda
- Imenovanje metoda
- Upotreba parametara metoda
- Komentarisanje
- Provera zavisnosti

Organizovanje koda

```
1: InicijalizujPodatke  
2: IzracunajTroskoveMarketinga  
3: IzracunajTroskoveProdaje  
4: IzracunajTroskovePutovanja  
5: IzracunajTroskoveOsoblja  
6: PrikaziUkupanTrosak
```

- ▶ Neprirodno je da metod *IzracunajTroskoveMarketinga* vrši i inicijalizaciju podataka
- ▶ Svi metodi samo računaju neku vrstu troškova, takvo ponašanje se očekuje i od metoda koji računa troškove marketinga
- ▶ Izdvaja se poseban metod koji inicijalizuje podatke i njegovo ime jasno govori da on prvi treba da se poziva

Imenovanje metoda

```
1: IzracunajTroskoveMarketinga
2: IzracunajTroskoveProdaje
3: IzracunajTroskovePutovanja
4: IzracunajTroskoveOsoblja
5: PrikaziUkupanTrosak
```

- ▶ Ime treba da opisuje šta metod radi
- ▶ Ime *IzracunajTroskoveMarketinga* nije dobro jer metod pored toga što izračunava troškove marketinga dodatno i inicijalizuje podatke
- ▶ Moguće rešenje je novo ime
InicijalizujPodatkeIzracunajTroskoveMarketinga
- ▶ Svakako bolje je prethodno predloženo izdvajanje posebnog metoda za inicijalizaciju

Upotreba parametara metoda

```
1: InicijalizujPodatke  
2: IzracunajTroskoveMarketinga  
3: IzracunajTroskoveProdaje  
4: IzracunajTroskovePutovanja  
5: IzracunajTroskoveOsoblja  
6: PrikaziUkupanTrosak
```

- ▶ Na osnovu ovog koda ne možemo da zaključimo da li neke metode koriste iste podatke
- ▶ Prikazati razmenu podataka između metoda



Upotreba parametara metoda

```
1: InicijalizujPodatke(troskovi)
2: IzracunajTroskoveMarketinga (troskovi)
3: IzracunajTroskoveProdaje (troskovi)
4: IzracunajTroskovePutovanja (troskovi)
5: IzracunajTroskoveOsoblja (troskovi)
6: PrikaziUkupanTrosak (troskovi)
```

- ▶ Vidi se da metode rade nad istim podacima
- ▶ Moguće da je redosled izvršavanja bitan

Upotreba parametara metoda

```
1: troskovi = InicijalizujPodatke(troskovi)
2: troskovi = IzracunajTroskoveMarketinga (troskovi)
3: troskovi = IzracunajTroskoveProdaje (troskovi)
4: troskovi = IzracunajTroskovePutovanja (troskovi)
5: troskovi = IzracunajTroskoveOsoblja (troskovi)
6: PrikaziUkupanTrosak (troskovi)
```

- ▶ Metode uzimaju zajednički podatak troskovi kao ulaz i vraćaju ažurirani podatak
- ▶ Jasnije se naznačava zavisnost među metodama
- ▶ Jasno se vidi da je redosled izvršavanja bitan

Upotreba parametara metoda

```
1: IzracunajTroskoveMarketinga (marketing)
2: IzracunajTroskoveProdaje (prodaja)
3: IzracunajTroskovePutovanja (putovanja)
4: IzracunajTroskoveOsoblja (osoblje)
5: PrikaziUkupanTrosak (marketing,prodaja,putovanja,osoblje)
```

- ▶ Podaci ne sugerišu neki određeni raspored
- ▶ Kod implicira da redosled izvršavanja prve četiri naredbe nije bitan
- ▶ Peta naredba koristi podatke prethodnih. Zaključujemo da se ona mora izvršiti na kraju.

Komentarisanje

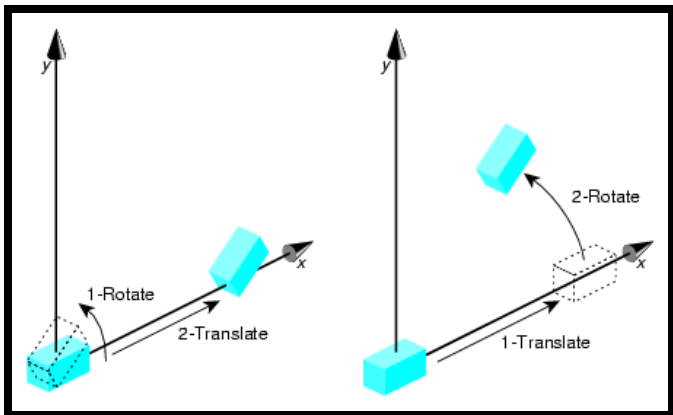
1. Pokušati sa pisanjem naredbi čiji redosled nije bitan
 2. Pisati kod tako da zavisnost među naredbama bude očigledna
 3. Eksplicitno dokumentovanjem naglasiti postojeće zavisnosti
-
- ▶ Kada ne možemo da popravimo kod
 - ▶ Ublažavanje slabosti koda

Primer - OpenGL

```
1: glTranslatef(10,0,0);  
2: glRotatef(45,0,0,1);
```

```
1: glRotatef(45,0,0,1);  
2: glTranslatef(10,0,0);
```

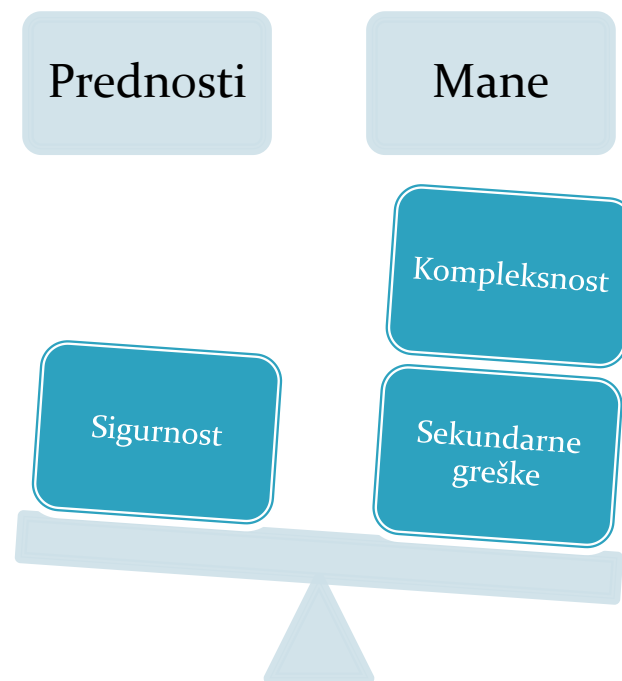
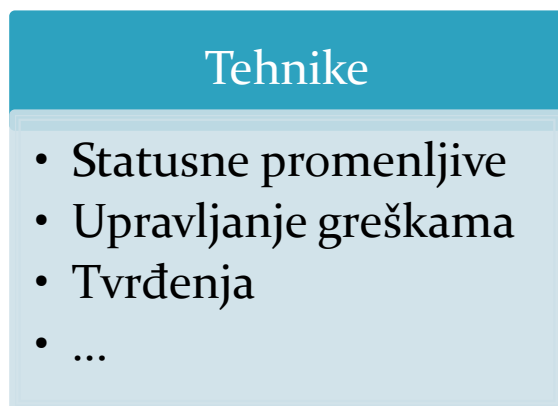
- ▶ Bitan je redosled izvršavanja
- ▶ Ne možemo da menjamo strukturu koda - OpenGL
- ▶ Komentarisanje



```
/*  
OpenGL izvrsava transformacije u  
obrnutom redosledu. Translacija  
mora da se navede posle rotacije.  
*/  
glRotatef(45,0,0,1);  
glTranslatef(10,0,0);  
drawObject();
```

Provera

- ▶ Kada je kod veoma kritičan
- ▶ Proveravamo kritične sekvencijalne zavisnosti
- ▶ Tehnike provere povećavaju kompleksnost i kreiraju mogućnosti za nove greške



Primer - Statusna promenljiva

```
1: ... podaciInicijalizovani = false; ...  
2: void InicijalizujPodatke() { ... podaciInicijalizovani = true; }  
3: void IzracunajTroskoveProdaje () { if(!podaciInicijalizovani)  
                                     return;  
                                     ...  
                                     }
```

- ▶ U konstruktoru statusnu promenljivu postavimo na netačno
- ▶ Funkcija za inicijalizovanje pre završavanja postavi promenljivu na tačno
- ▶ Funkcije koje koriste podatke prvo provere statusnu promenljivu

Nebitan redosled

- ▶ Pojedinačne naredbe ili blokove pišemo u proizvoljnom redosledu, ako posmatramo samo primarni kriterijum

Sekundarni kriterijumi

- Čitljivost
- Efikasnost
- Održavanje

Primarni kriterijum

- Tačnost

Kod koji se čita redom

Princip blizine:

Držati povezane naredbe zajedno

- ▶ Nije dovoljno samo da se kod izvršava pravim redosledom
- ▶ Kod se čita odozgo na dole bez potrebe za preskakanjem
- ▶ Povećava se čitljivost koda
- ▶ Nema potrebe za čitanjem celog programa kako bi se došlo do željenih informacija

Primeri - Loša organizacija

```
1  MARKETING_PODACI *marketingPodaci = new MARKETING_PODACI;
2  PRODAJA_PODACI *prodajaPodaci = new PRODAJA_PODACI;
3  PUTOVANJA_PODACI *putovanjaPodaci = new PUTOVANJA_PODACI;
4
5  putovanjaPodaci.IzracunajKvartalni();
6  prodajaPodaci.IzracunajKvartalni();
7  marketingPodaci.IzracunajKvartalni();
8
9  prodajaPodaci.IzracunajGodisnji();
10 marketingPodaci.IzracunajGodisnji();
11 putovanjaPodaci.IzracunajGodisnji();
12
13 prodajaPodaci.Stampaj();
14 putovanjaPodaci.Stampaj();
15 marketingPodaci.Stampaj();
16
17 delete prodajaPodaci;
18 delete marketingPodaci;
19 delete putovanjaPodaci;
```

- ▶ Slaba čitljivost
- ▶ Prolazi se kroz ceo kod

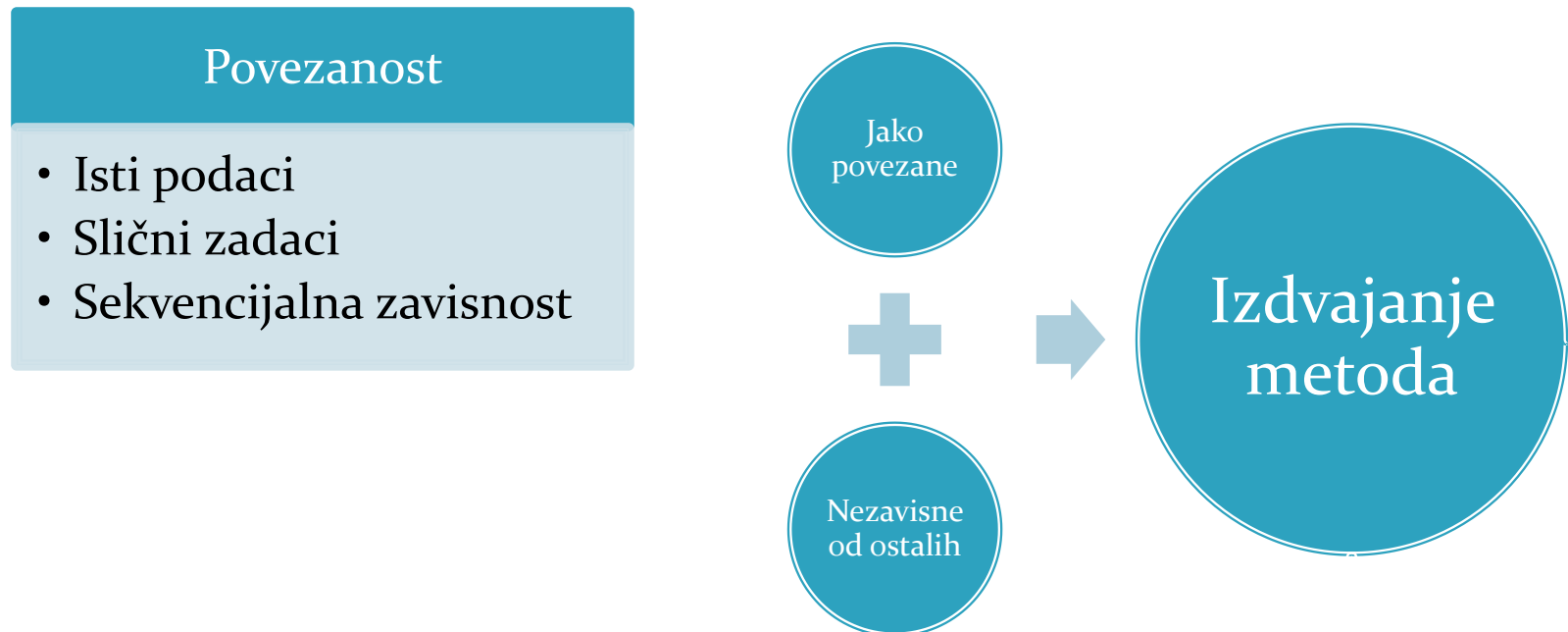
Primeri - Dobra organizacija

```
1  MARKETING_PODACI *marketingPodaci = new MARKETING_PODACI;  
2  marketingPodaci.IzracunajKvartalni();  
3  marketingPodaci.IzracunajGodisnji();  
4  marketingPodaci.Stampaj();  
5  delete marketingPodaci;  
6  
7  PRODAJA_PODACI *prodajaPodaci = new PRODAJA_PODACI;  
8  prodajaPodaci.IzracunajKvartalni();  
9  prodajaPodaci.IzracunajGodisnji();  
10 prodajaPodaci.Stampaj();  
11 delete prodajaPodaci;  
12  
13 PUTOVANJA_PODACI *putovanjaPodaci = new PUTOVANJA_PODACI;  
14 putovanjaPodaci.IzracunajKvartalni();  
15 putovanjaPodaci.IzracunajGodisnji();  
16 putovanjaPodaci.Stampaj();  
17 delete putovanjaPodaci;
```

- ▶ Lokalizovanost
- ▶ Izdvajanje metoda

Grupisanje naredbi

- ▶ Princip: Grupisati povezane naredbe

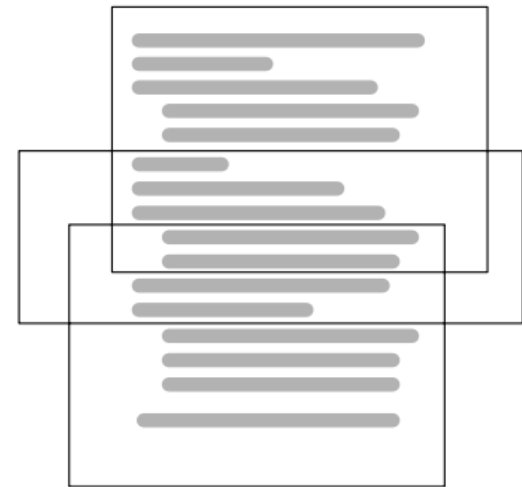


Provera kvaliteta organizacije koda

- ▶ Odštampamo listu naredbi
- ▶ Nacrtamo okvire oko povezanih naredbi
 - Ako je kod dobro organizovan, okviri se neće preklapati
 - Inače, okviri se preklapaju. Potrebno je reorganizovati kod

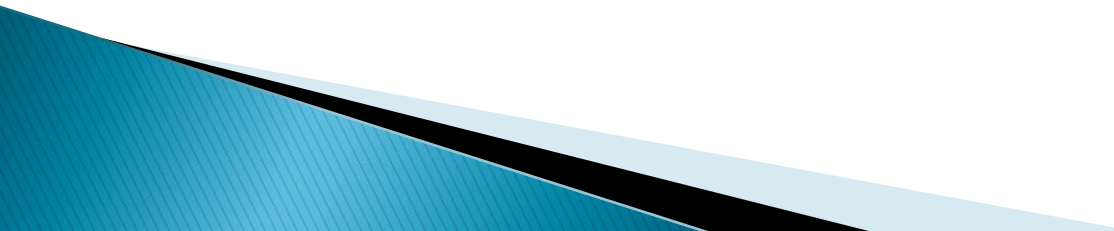


Dobro



Loše

Smernice

- ▶ Zavisnosti između naredbi lako uočljive samim čitanjem koda?
 - ▶ Imena metoda čine zavisnosti očiglednim?
 - ▶ Parametri metoda čine zavisnosti očiglednim?
 - ▶ Nejasne zavisnosti opisane komentarima?
 - ▶ Vršiti se proveru zavisnosti u kritičnim delovima?
 - ▶ Kod se čita redom?
 - ▶ Povezane naredbe su grupisane?
 - ▶ Nezavisne grupe naredbi su izdvojene u svoje metode?
- 

Podsetnik

- ▶ Najjači princip organizovanja pravolinijskog koda su sekvencijalne zavisnosti
- ▶ Zavisnosti treba učiniti očiglednim pomoću:
 - Dobrih naziva metoda
 - Parametara metoda
 - Komentara
 - Provere u slučaju kritičnih delova
- ▶ Ako ne postoje sekvencijalne zavisnosti, tj. redosled nije bitan, povezane naredbe treba držati što je moguće bliže

Hvala na pažnji!

Pitanja?

