SOFTVERSKA KONSTRUKCIJA- KOMENTARISANJE PROGRAMA

SOFTWER CONSTRUCTION- COMMENTING PROGRAM

Mr Dragan Damjanović [1)]

Rezime: *na razumljivost napisanog koda programa znatno utiču dopunske informacije koje obezbeđuju komentari. Kao i u svemu i u pisanju komentara mora postojati mera, jer previše komentara je isto tako loše kao i premalo. Otuda komentarisanje mora biti efikasno.*
Ključne reči: *komentarisanje, program, kod, efikasnost, promenljive, konstante, funkcije*

Abstract: *at the intelligibility of the written code of significantly affecting the additional information to provide comments. As with all comments in writing must be a measure, because too many comments is just as bad as too little. Hence the comment must be efficient.*
Keywords*: comment, software, code, efficiency, variables, constants, functions*

1. UVOD

Veoma često, programe razvijaju timovi inženjera i programera i izvorni kod se često prerađuje i menja od strane različitih ljudi. Takođe, dešava se da jednom napisan kod mora nakon veoma dugog vremena od njegovog nastanka da se izmeni. Kvalitetno napisan kod olakšava snalaženje u ovakvim situacijama i obezbđuje značajnu uštedu vremena. Jedan od ključnih faktora za razumevanje koda je upotreba smislenih naziva promeljivih, konstanti, procedura i sl. Upotrebom razumljivih imena nastaje samo-dokumentujući kod, tj. kod koji je razumljiv sam po sebi bez potrebe za pisanjem dodatnih komentara.
Primer:
Por1: Real; // porez na dohodak (primer loše dodeljenog imena)
Por2: Real; // porez na promet (primer loše dodljeneog imena)
PorezNaDohodak: Real; //komentari nisu potrebni zahvaljujuci
PorezNaPromet: Real; //razumljivosti naziva promenljivih
Za dodeljivanje imena promenljivima, procedurama i drugim elementima programa treba koristiti pune nazive. Treba izbegavati skraćenice. Npr. treba koristiti razumljiva imena poput «NazivFirme», «AdresaFirme» ili «SrednjeSlovo» umesto «Nfirme», «aFirme» ili«Ss». Izbegavajte upotrebu donje crte «_» za razdvajanje reči unutar naziva kao npr. «Naziv _Firme» jer ovaj karakter nepotrebno povećava dužinu imena, a ne doprinosi boljoj čitljivosti.
Treba izbegavati imena duža od 15 karaktera jer ona smanjuju čitljivost. Npr. Umesto «PostaviDuzinuTesktaLabele» bolje je koristiti kraći naziv «PostaviDuzinu» koji je dovoljno jasan.
Treba izbegavati imena koja su veoma slična ili se razlikuju samo u jednom karakteru. Npr. treba izbegavati nazive kao što su «Proizvod» (promenljiva), «Proizvodi» (promenljiva) i «Proizvedi» (procedura, naredba) u okviru jednog istog programa kako bi
se sprečila mogućnost njihovog brkanja.
Ako naziv predstavlja složenicu sastavljenu od više reči, tada se svaka reč u složenici piše velikim slovom radi poboljšanja čitljivosti. Npr. «SetBrushColor» je mnogo čitljivije od «setbrushcolor». U zavisnosti od konkretnog jezika u kome se programira, postoje različite konvencije o tome da li početno slovo naziva treba biti veliko ili malo. Prema konvenciji usvojenoj u Delphi-u sva početna slova naziva promeljivih, tipova, klasa, objekata, propertija, procedura, funkcija, metoda i događaja pišu se velikim slovom (npr. «MojBroj», «Real», «Form1», «TPanel», «Font.Name», «SetLength», «OnMouseClick» itd.), dok se konstatne i sve službene reči pišu malim početnim slovom (npr. «mtError», «clRed», «begin», «end», «for» «string»1 i sl.).

1.1. Dodeljivanje imena promenljivama

Svrha dodeljivanja smislenih imena promenljivama jeste omogućavanje njihovog razlikovanja od drugih tipova podataka. Osnovno pravilo je da se izbegavaju generička imena poput «Broj» ili «Slovo» jer je njihova namena nejasna.
Za razliku od većine drugih struktuiranih programskih jezika, Delphi preporučuje da se

1) Mr Dragan Damjanović, e-mail: damjanovic1971@gmail.com

nazivi promenljivih pišu velikim početnim slovom. Java, C, C++, PHP i neki drugi jezici koriste mala pocetna slova za nazive svih promenljivih i svih funkcija. Za nizove je poželjno koristiti imena u množini. Npr. «RezultatiMerenja» umesto «RezultatMerenja», odnosno «Rezultati[0]» umesto «Rezultat[0]» Izuzeci od gornjih pravila su nazivi promeljivih koje se koriste u petljama. Ovi nazivi treba da budu što kraći jer se koriste često, a smisao im je apstraktan, pa ne utiču na razumljivost koda. Primeri su «I», «J», i «K» koji se često koriste kao celobrojni brojači u «for» petljama. Svaku promenljivu treba deklarisati u zasebnoj liniji koda, umesto korišćenja grupne deklaracije promenljivih odvojenih zarezima. Grupne deklaracije otežavaju pronalaženje deklaracionog bloka i komplikuju prepravke programa. Koliko je npr. Truda potrebno uložiti da bi se u sledećem primeru iskomentarisala deklaracija promeljive «Z» u poređenju sa situacijom da je u istom kodu potrebno iskomentarisati deklaraciju promenljive «K»?

```
var X, Y, Z, W, Q: Real;
var I: Integer;
var J: Integer;
var K: Integer;
var L: Integer;
```

Akonaziv i svrha promenljive nisu jasni, dodajte linijski komentar (//) na kraju reda iza deklaracije promenljive koji će objasniti za šta se promenljiva koristi i zbog čega. Kad kog to ima smisla i kad god je moguće postavite početne vrednosti promenljivih u istom redu gde su i deklarisane.

```
var I: Integer = 0;
```

1.2. Dodeljivanje imena konstantama

Za razliku od drugih programskih jezika koji preporučuju da se imena konstanti pišu isključivo velikim slovima uz upotrebu donje crte (ALL_UPPER_CASE), Delphi preporučuje da se imena konstanti pišu kombinacijom malih i velikih slova bez upotrebe donje crte, s tim da je početno slovo konstante uvek malo.

Svrha konstanti je izbeganje upotrebe magičnih borjeva. Magični brojevi su bilo koji konkretni brojevi osim 0 i 1, kao npr. 27 ili $F5A (heksadecimalno) koji imaju neko «podrazumevano» značenje. Umesto toga, razmotrite definisanje konstante koja će broju dati neki smisao.

Primer:

```
Brush.Color := $00FFFF00; //Oh, ne! Koriste se magični brojevi!?
const clYellow = $00FFFF00;
Brush.Color := clYellow; //Da! Ovo je jasno samo po sebi
```

1.3. Imena funkcija (procedura, metoda i događaja)

Uvek se potrudite da smislite razumljiv i smislen naziv koji sažeto opisuje svrhu funkcije. Za imena funkcija, procedura, metoda i događaja koristite mešana mala i velika slova počevši svaku novu reč u složenicama sa velikim slovom, kao i kod promenljivih. Prva reč u nazivu funkcije treba da bude glagol (npr. «Saberi»). Ako se naziv funkcije ne može osmisliti tako da u sebi sadrži glagol, tada uvođenje funkcije verovatno nije ni bilo potrebno. Ukoliko sam glagol nije dovoljan (npr. «Postavi» !?) glagolu treba dodati objekat (imenicu) kao npr. «PostaviPocetnuVrednost» koja će bolje objasniti smisao funkcije. Ne treba izbegavati ni ubacivanje predloga kada je to neophodno radi bolje razumljivosti kao npr. «PostaviNaNulu» i «PostaviNulu» semantički nemaju isto značenje.

2. KOMENTARI

Komentari obezbeđuju dopunske informacije koje znatno utiču na razumljivost napisanog koda. Komentare treba koristiti kako bi se obezbedio sažet opis nekog dela koda i kako bi se onome ko pokušava da razume kod obezbedila dodatna objašnjenja onih stvari koje nisu trivijalno dostupne iz samog koda. Posebno je poželjno komentarisati originalne I neuobičajene kombinacije naredbi. Izbegavajte komentare koji su nepotrebno ponavljanje očiglednih informacija.

2.1. Tipovi komentara

Postoje tri neizostavna tipa komentara koji se upotrebljavaju u profesionalno napisanom softveru:

•Komentar u zaglavlju programa koji identifikuje autora i namenu programa, daje informacije o autorkim pravima, verziji programa, poslednjim izmenama i sl.

```
//*****************************************
**********************
// Naziv: Program za nakupljanje poena iz MSuI
//
// Opis: Ovaj program ne radi ama bas nista ali mi omogucava da
// nakupim neke poene i tako (mozda) polozim ispit iz MSuI
//
// Autor: Pera Petrovic
// Indeks: E10000
// Datum kreiranja: 20. februar 2009.
//
// Datum poslednje izmene: 31. maj 2009.
// Napomene: Kunem se da sam ovaj kod napisao bas ja i da ga
```

```
// nisam ukrao sa Interneta niti prepisao od kolega.
// Ovom prilikom zelim da se zahvalim mami i tati koji su
// mi omogucili skolovanje i odrekli se svog dugo
// planiranog letovanja na Maldivima kako bih ja mogao
// dobiti diplomu i raditi (daj, Boze) dobro placen posao.
//**************************************************
**********************
```

• Samostalan linijski komentar koji objašnjava šta deo koda koji sledi treba da obavi. Odvajanje blokova koda praznim linijama i dodavanje komentara na početak svakog bloka omogućava brzu identifikaciju I razumevanje delova koda u kojima se odvijaju specifične akcije.

```
// Racunanje srednje vrednosti 100 rezultata merenja
Suma := 0;
for I := 0 to 100 do
Suma := IzmereneVrednosti[I];
SrednjaVrednost := Suma / 100
```

• Linijski komentar u produžetku linije sa kodom koji objašnjava šta ta linija koda radi. Ovo su komentari koje treba izbegavati, kao što se jasno vidi iz donjeg primera. Izuzetak od ovog pravila su programi koji predstavlja edukativne primere ili neke ekstremne situacije koje primenjuju globalne i sistemske prmenljive i funkcije

```
SS = S1 + S2 + S3; //sabiranje tri broja (!?)
A = SS / 3; //racunanje srednje vrednosti (!?)
Suma := S1 + S2 + S3;
SrednjaVrednost := Suma / 3;
if Komanda = CmdPUT then PostaviTekst; //ovde bi bas dobro
//dosao komentar (!!)
```

Uvek dodajte komentare u toku pisanja odgovarajućih koda (ili čak neposredno pre toga) umesto da čekate dok ne završite ceo programa. Tako će komentari biti aktuelni, tačni i sažeti, te će doprineti razumevanju koda, umesto da samo služe za popunjavanje praznina. Kao što u svemu treba imati meru, tako treba izbegavati i preterano komentarisanje koda. Učestali komentari ukazuju na loše napisan, nespretan i amaterski kod. Kad kod dođete u iskušenje da dodate neki komentar, dobro razmislite o tome da li možete prepraviti kod tako da on sam po sebi postane razumljiviji.

Vitičaste zagrade «{» i «}» mogu se koristiti za komentarisanje većeg dela koda prilikom razvoja i testiranja aplikacije kako bi se proverilo kako program funkcioniše ako se taj deo koda izostavi. Slično se mogu koristiti linijski komentari «//»za izostavljanje jedne linije koda i njeno lako naknadno vraćanje. Finalne verzije programa ne bi trebalo da sadrže nijednu komentarisu liniju koda. Kada se testiranje završi sve komentare čija je namena bila debagovanje treba ukoniti iz koda.

Veoma važan factor u razvoju kvalitetnog softvera je komentarisanje programa. Najčešće se dešava da kod ima netačan komentar, takođe se dešava da komentar prepisuje kod I nije preterano od koristi. S druge strane postoje i primeri dobrih komentara. Tako da je bolje ne pisati komentare nego pisati loše i siromašne komentare, tačnije loše napisani komentari su gori od nepostojećih komentara.

2.2. Klasifikacija komentara

Komentari se klasifikuju u šest kategorija:

• Ponavljanje koda- ova vrsta komentara ne pruža čitaocu dodatne informacije, jer samo drugim rečima precizira šta kod predstavlja.

• Objašnjenje koda- komplikovane, zamršene ili osetljive linije koda se objašnjavaju ovom vrstom komentara I zbog toga su veoma korisni, pre svega jer je kod često zbunjujuć ili konfuzan. U ovom slučaju, kada je kod toliko komplikovan da je potrebno objašnjenje, uvek je bolje poboljšati kod nego dodavati komentar. Kod treba učiniti jasnijim, a kasnije koristiti komentare za pregled (o kojima će biti reči kasnije).

• Markeri u kodu- ovi komentari ukazuju programeru da rad još nije završen I stavljaju se pod posebne karaktere da bi se lakše našli prilikom pretrage.

• Pregled koda- za one koji nisu autori koda ova vrsta komentara u jednoj ili dve rečenice opisuju nekoliko linija koda.

• Opis namene koda- svrhu koju ima sekcija koda objašnjava se ovim komentarima i navode se u formi rešenja a ne u formi problema, na primer: dohvatanje podataka o zaposlenom.

• Komentari nevezani za sam kod- iaoko nisu vezani za sam kod pojedine vrste komentara se isto navode u izvornom kodu (broj verzije, online reference, copyright informacije i sl.).

Za kompletiran kod su prihvatljive tri vrste komentara a to su Pregled koda, opis namene i komentari nevezani za sam kod.

2.3. Efikasno komentarisanje

Previše komentara je isto tako loše kao i premalo. Zato komentarisanje mora biti efikasno a samim tim ono ne oduzima toliko vremena. Preporuke za efikasno komentarisanje su:

• Korišćenje stilova koji se lako modifikuju

```
/**************************************************
*****************************
 * class:  GigaTron (GIGATRON.CPP) *
 *                                 *
 * author: Dwight K. Coder*
 * date:   July 4, 2014 *
 *                                 *
```

```
 * Routines to control the twenty-first century's code evaluation     *
 * tool. The entry point to these routines is the EvaluateCode()     *
 * routine at the bottom of this file.                               *
 ***********************************************************************/
```

Ovo je primer komentarisanja u C++ koji je težak za održavanje i to zbog zvezdica sa leve i desne strane koje treba ubacivati i pomerati pri bilo kakvoj modifikaciji iako ovako napisan blok komentar izgleda lepo i čitav blok je vidljivo zaokružen sa jasno istaknutim početkom i krajem ono što nije na prvi pogled vidljivo je težina održavanja.

Sledeći primer je lak za održavanje jer se lako vrši modifikacija.

```
/***********************************************************************
  class:  GigaTron (GIGATRON.CPP)
  author: Dwight K. Coder
  date: July 4, 2014
  Routines to control the twenty-first century's code evaluation
  tool. The entry point to these routines is the EvaluateCode()
  routine at the bottom of this file.
***********************************************************************/
```

- Korišćenje pseudokoda radi uštede vremena

Velike uštede se ostvaruju ukoliko se istaknu komentari pre pisanja koda. Kad se završi kodiranje komentari su već tu.

- Uključivanje komentarisanja u stil rada programera

Suprotno malopređašnjem kada se komentari istaknu pre pisanja koda ovde se ostavlja komentarisanje za sam kraj projekta što inače ima više nedostataka, jer se veći deo posla ostavlja za kasnije, umesto da se radi po malo s vremena na vreme, a pored toga potrebno je pamtiti i setiti se svega kodiranog umesto da se zapiše komentar kada se radi taj deo.

2.4. Tehnike komentarisanja

U zavisnosti od nivoa na koji se odnosi komentar postoji nekoliko različitih tehnika za komentarisanje. Znači, sve zavisi da li je u pitanju program, fajl, rutina ili linija koda. U dobrom kodu potreba za komentarisanjem pojedinih linija koda je retka. Linija koda ima komentar onda kada je suviše komplikovana i zahteva objašnjenje ili je nekada imala grešku i potrebno je zapisati informacije o tome. U nastavku linije sa kodom se unose komentari na kraju reda i ponekad mogu biti korisni, ali ih treba izbegavati jer unose nepreglednost u sam kod a i teški su za modifikovanje i formatiranje. Komentarisanje na kraju pasusa je najčešća vrsta komentara koja se koristi u dobro dokumentovanom programu. Ova vrsta komentara opisuje namenu koda i lako se održava.

```
// swap the roots
oldRoot = root[0];
root[0] = root[1];
root[1] = oldRoot;
```

3. ZAKLJUČAK

Samodokumentujući kod je onaj kod koji je nastao upotrebom razumljivih imena, drugim rečima to je kod koji je razumljiv sam po sebi i u ovom slučaju nema potrebe za pisanjem dodatnih komentara. U uvodu ovog rada opisano je kako na što efikasniji način napisati kod koji će biti svima razumljiv uz veoma malo ili nimalo komentarisanja. Pa tako promenljivim dodeljujemo imena da bi ih mogli razlikovati od drugih tipova podataka. Ako sve ovo nije moguće i ako su neophodne dopunske informacije koje znatno utiču na razumljivost napisanog koda upotrebljavaju se komentari. I u pisanju komentara postoje određena pravila kojih se programeri moraju pridržavati.

LITERATURA

[1] S. McConnell, Code Complete: A Practical Handbook of Software Construction, Microsoft Press, second ed., 2004.

[2] Card, David N. , Victor E. Church , and William W. Agresti . "An Empirical Study of Software Design Practices." IEEE Transactions on Software Engineering SE-12, 1986.