Predrag Oreški

DP "FOING" d.o.o. Varaždin

UDK: 681.32.06
Stručni rad

MODULARNOST I KVALITETA SOFTVERA

Razvoj sofvera je skup proces. Cijena održavanja softvera ovisna je o produktivnosti rada na održavanju, a na produktivnost održavanja utječe njegova kvaliteta. Na kvalitetu softvera bitno utječe složenost (kompleksnost, koja ovisi o njegovoj veličini), razumljivost i jednostavnost izvornog koda softvera. Da bi se smanjila kompleksnost i povećali razumljivost i jednostavnost, koristi se tzv. modularno programiranje. To je izgradnja arhitekture aplikacije od modula - dijelova programa koji čine logičke i zaokružene cjeline te mogu biti samostalno prevođeni od strane programa prevodioca. U ovom radu autor želi prikazati kakav je utjecaj modularnosti softvera na njegovu kvalitetu.

Modularnost; kvaliteta softvera; dizajn; arhitektura programa; metrika softvera.

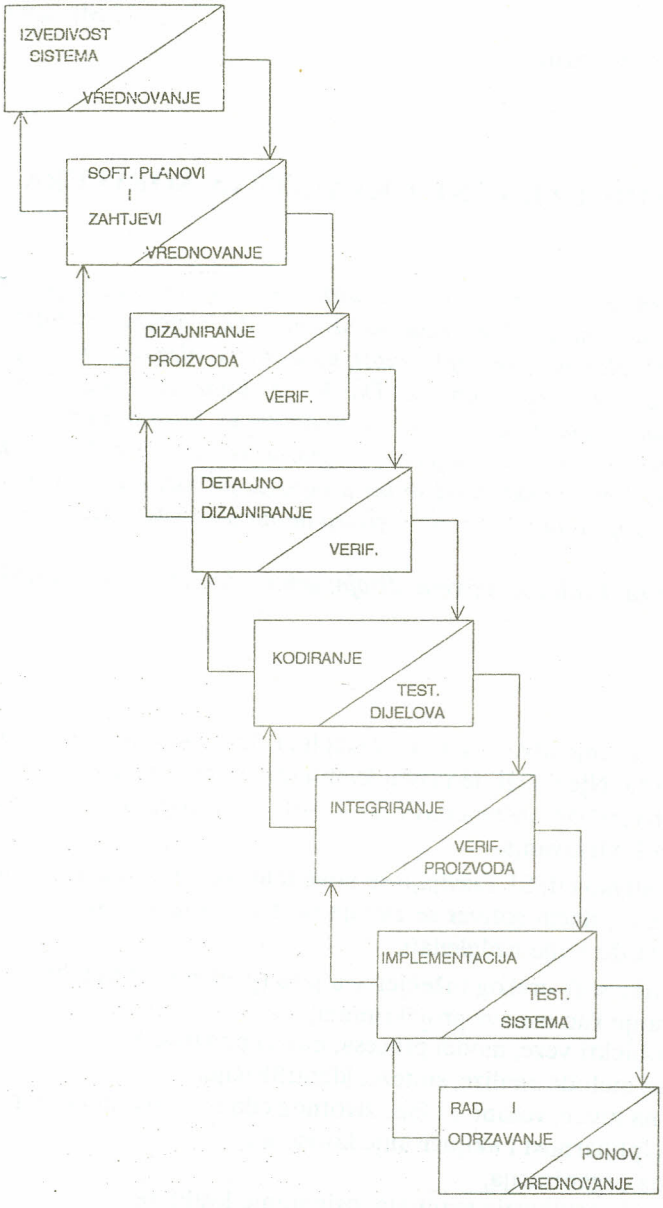
1 UVOD

Softversko inženjerstvo (SI) je disciplina koja se bavi tehnikama razvoja i održavanja softvera. Njezin cilj je razvoj kvalitetnijeg softvera uz niže troškove. Objekti promatranja softverskog inženjerstva jesu softver (programski proizvod) i proces njegovog razvoja i održavanja.

Da bi se postigao cilj SI, razvijaju se nove tehnike u razvoju softvera koje su slične inženjerskim. Na taj način softver se razvija po fazama te se tako njegov razvoj kao i utrošak resursa može bolje nadgledati.

Karakteristike softverskog inženjerstva jesu (prema STRAH 89):

" - modeliranje kao osnova projektiranja,
(model objekti-veze, model procesa, model podataka)
- koriste se metode analize, sinteze, identifikacije,
- podjela na nivoe, vezane uz faze životnog ciklusa softvera i faze projekta,
- interdisciplinarnost i uključivanje korisnika,
- projektna organizacija,
- validacija i verifikacija rezultata, osiguranje kvalitete,
prikaz rezultata i dokumentiranje."



Slika 1. Vodopadni (Waterfall) model životnog ciklusa softvera (Izvor: BOEHM 81, str. 36)

"Životni ciklus programskog proizvoda je vremenski period između iniciranja njegovog razvoja i trenutka kad on više nije na raspolaganju (ANSI/IEEE 729/1983)" (STRAH 89).

Između ovih 8 faza razvoja prikazanih na slici 1., dvije se tiču dizajna softvera: faza 3. Dizajn programskog proizvoda i faza 4. Detaljni dizajn.

U fazi Dizajn programskog proizvoda specificira se cjelokupna konfiguracija sistema, programski jezik, glavni moduli i njihove veze, strukture podataka i plan testiranja, a u fazi Detaljni dizajn detaljnije se specificiraju moduli, njihova neophodna međusobna komunikacija, algoritmi, struktura podataka i interne kontrolne strukture.

Kvaliteta softvera ovisi i o tome kako je on dizajniran. U vezi s dizajnom postoje određeni zahtjevi. Kako su oni ispunjeni, utvrđuje se mjerenjem.

Postoji mnogo karakteristika softvera koje možemo mjeriti. Postoji cijeli niz metrika[1] koje se bave mjerenjem softvera:

A Metrike veličine,
- A.1 Linije koda
- A.2 Broj 'tokena'
- A.3 Broj funkcija
- A.4 Mjere ekvivalentne veličine

B Metrike strukture podataka,
- B.1 Količina podataka
- B.2 Korištenje podataka unutar modula
- B.3 Zajedničko korištenje podataka između modula

C Metrike logičke strukture,
- C.1 Broj grananja
- C.2 Metrika minimalnog broja puteva i načina dostupa do određenog dijela programa
- C.3 Nivoi gniježdenja
- C.4 Način kontrole toga programa (korištenje GOTO naredbe)

D Kompozitne (složene) metrike,

E Kompozitne (složene) metrike znanosti o softveru, (Software Science Composite Metrics)

F Metrike napora i cijene koštanja,

G Metrike grešaka i pouzdanosti,

H Metrike dizajna.

---

1 Formalna definicija metrika kaže da su to "skale ili jedinice pomoću kojih se može mjeriti kvantitativni atribut" (KITCH 90).

Između više tipova metrika softvera postoji i metrika dizajna. Ona se bavi dizajnom strukture (strukturu čine veze između pojedinih komponenata programa) i dizajnom modula (dizajn modula čini njegova unutarnja konstrukcija od komponenata, a to su najčešće funkcije[2]).

2 DIZAJN - FAZA ŽIVOTNOG CIKLUSA SOFTVERA

Dizajnom se definira arhitektura kompjutorskih programa. Po metodologiji strukturnog dizajna arhitektura programa gradi se od dijelova programa - modula. Programski modul je dio programskog koda koji se može samostalno prevoditi uz pomoć programa prevodioca.

Arhitekturu kompjutorskog programa čine programske komponente - moduli i skup veza između njih, kao i veze prema radnom okruženju. Modularnost se definira kao "programska tehnika konstruiranja softvera od više nezavisnih dijelova" (CONTE 86). Zašto se koristi ova tehnika, odnosno koje su prednosti te tehnike?

Složenost programa povećava se s veličinom programa. Velike programe je teško pisati, oni vjerojatno sadrže više grešaka, a njih je teško ispravljati. Izgradnja programa od nezavisnih modula ima nekoliko prednosti:

- veća razumljivost,
- veća upravljivost,
- veća djelotvornost,
- manji broj grešaka,
- potreban je manji napor/vrijeme kod održavanja.

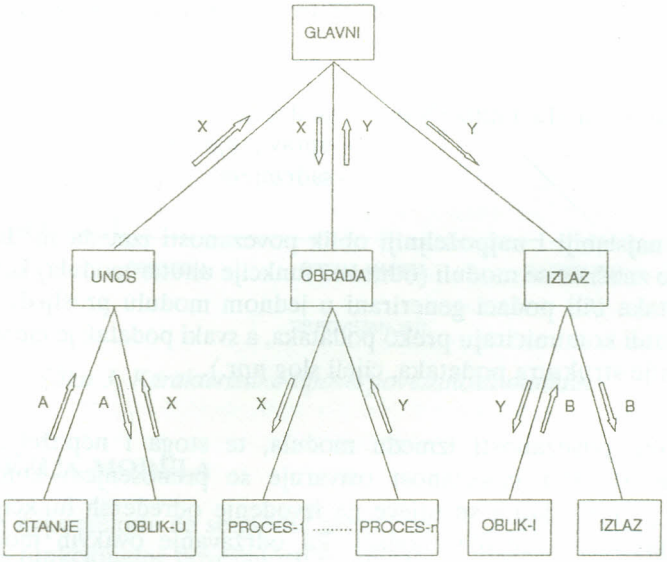
Metodologija strukturnog dizajna polazi od 5 principa dizajna:

1 - veze između modula,
2 - kohezija,
3 - složenost,
4 - modularnost,
5 - veličina.

---

2 Funkcija u kompjutorskom programu je skup izvršnih naredbi (koje obavljaju određeni zadatak), deklaracija formalnih parametara i lokalnih varijabli kojima upravljaju prije spomenute naredbe (CONTE 86).

Kod veza **između modula** mjeri se njihov broj. Smatra se da će dizajn s više veza između modula sadržavati više grešaka. **Kohezija** je mjera za povezanost funkcija unutar modula. Što je takva povezanost manja, to je veća mogućnost za greške i teže je održavanje. U vezi sa **složenošću** treba reći da dizajn treba biti uvijek što je moguće jednostavniji. Stupanj **modularnosti** utječe na kvalitetu dizajna. Prevelika modularnost (granularnost) je isto tako nepoželjna kao i premala. Naime, kod prevelike modularnosti naglo se povećava broj veza između modula pa postoji veća mogućnost za greške te se gubi kontrola



Slika 2. Primjer izgradnje aplikacije od programskih modula. Svaki modul ima točno određenu funkciju(e). (Izvor: RADOVAN 89, str. 49)

nad tokom procesa. Za **veličinu**, kao princip stukturnog dizajna, vrijedi već prije spomenuto: veći programi (moduli) sadrže više grešaka nego mali.

3 POVEZANOST I KOHEZIJA MODULA

Između više kriterija za dizajn u ovom radu autor spominje povezanost modula, koheziju modula, fan-out i fan-in.

3.1 Povezanost modula

Kod povezanosti modula nastoji se da ona bude što manja. Na taj način:

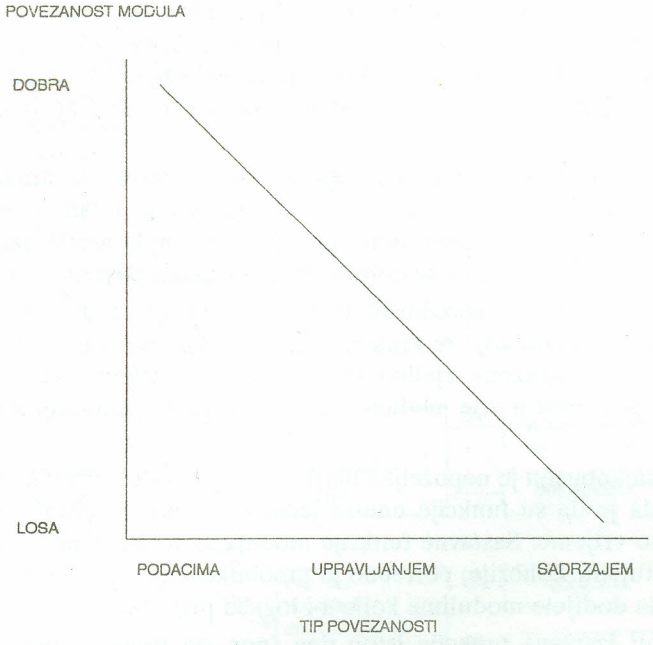
- manja je mogućnost za utjecaj greške u jednom modulu na drugi,
- manja je vjerojatnost da promjena u jednom modulu zahtijeva promjenu i u drugom,
- manja je potreba za poznavanjem drugih modula programa kod održavanja,
- smanjuje se složenost programa, složenost održavanja, a povećavaju se jednostavnost i razumljivost.

Povezanost modula može biti:
- podacima,
- upravljanjem i
- sadržajem.

Nužni, najslabiji i najpoželjniji oblik povezanosti između modula je **povezanost podacima**. To znači da se moduli (odnosno funkcije unutar modula) koriste zajedničkom bazom podataka i/ili podaci generirani u jednom modulu prosljeđuju se/koriste se u drugom. Moduli komuniciraju preko podataka, a svaki podatak je elementaran (to znači da podatak nije struktura podataka, cijeli slog npr.).

Jači oblik povezanosti između modula, te stoga i nepoželjan, je **povezanost upravljanjem**. Ovakva povezanost ostvaruje se prenošenjem kontrolnih podataka (indikatora) pomoću kojih se utječe na izvođenje određenih funkcija unutar modula kojemu se prenose kontrolni podaci. Za održavanje ovakvih modula potrebno je poznavanje internih logičkih struktura i pozivajućeg i pozivanog modula. Ovakva zavisnost drugog modula o prvom može se izbjeći podjelom drugog modula na više nezavisnijih modula.

Moduli su **povezani sadržajem** ako jedan modul postavlja podatke ili kontrolne indikatore drugom modulu. Treba napomenuti da kod ovakvog postavljanja taj drugi modul nije odmah i pozvan na izvođenje. Dakle, podaci se ne prenose, već se samo postavljaju skretnice drugom modulu za neko kasnije izvođenje. Ovakav način povezivanja modula je najnepovoljniji te ga svakako treba izbjeći.



Slika 3. Karakteristike tipova povezanosti modula

3.2 KOHEZIJA MODULA

Već je prije napomenuto da se modul sastoji od jedne ili više funkcija. "Kohezija modula je stupanj funkcionalne povezanosti elemenata unutar jednog modula" (STRAH 89).

S obzirom na funkcionalnu povezanost elemenata[3] (funkcija) modula, postoje različite kohezije modula.

Funkcionalna kohezija modula je najpoželjniji oblik kohezije modula. Ova kohezija vrijedi ako su svi elementi unutar modula podređeni jednoj i samo jednoj problemskoj funkciji. Obično se javlja na najnižem nivou dekompozicije i ujedno znači i najniži nivo povezanosti modula.

3 Element modula može biti instrukcija ili grupa instrukcija koja izvršava neki posao.

Ako se funkcije unutar modula izvršavaju jedna iza druge u sekvencijalnom redoslijedu i ako se izlaz iz prethodno izvršene funkcije koristi kao ulaz u slijedeću, tada možemo reći da modul karakterizira **sekvencijalna kohezija**. Ovo je još uvijek povoljan stupanj kohezije, iako se funkcije unutar modula mogu, ali i ne moraju, izdvojiti u posebne module.

Modul sadrži **komunikacijsku koheziju** ukoliko izvršava više funkcija koje koriste zajedničke podatke, a redoslijed izvršavanja funkcija nije bitan. I ovaj tip kohezije modula je prihvatljiv iako se, opet, može povećati stupanj kohezije ukoliko se funkcije izdvoje u posebne module. Time se dobiva i manji stupanj povezanosti modula.

Kohezija modula je **proceduralna** ako se funkcije unutar modula izvršavaju sekvencijalno, a operacije koje te funkcije obavljaju sasvim su različite. Ovakav modul ne čini logičku i zaokruženu cjelinu te je stoga nepoželjan. To znači da ga treba preoblikovati i podijeliti u više modula koje karakterizira povoljniji stupanj kohezije i povezanosti.

Vremenska kohezija je nepoželjan oblik kohezije modula. Karakteristika ovog tipa kohezije modula je da su funkcije unutar jednog modula povezane samo time što se izvršavaju u isto vrijeme. Sastavne funkcije modula su međusobno nezavisne. Da bi se izbjegao ovaj stupanj kohezije, potrebno je preoblikovanje modula te da se funkcije iz ovakvog modula dodijele modulima kojima i logički pripadaju.

Modul koji izvršava funkcije istog tipa (npr. svi ispisi na ekran, sva otvaranja datoteka, itd.), a ne sadrži ni jedan drugi oblik kohezije, označavamo **logičkim tipom kohezije**. Funkcija koja se unutar modula treba izvršiti bira se izvan modula. Ovakav oblik kohezije je slab te ga treba izbjegavati. Posljednji i najslabiji oblik kohezije modula je **slučajna kohezija**. U ovom slučaju funkcije unutar modula nisu povezane ni jednim od prethodnih oblika kohezije, a nemaju ni bilo kakve veze među sobom.

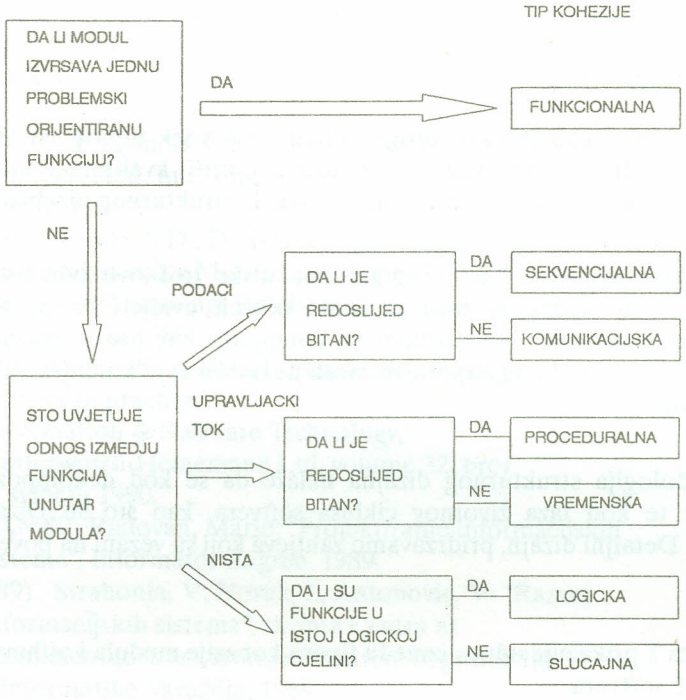
Kako identificirati tip kohezije modula ilustrirano je na slici 4.

3.3 FAN-IN i FAN-OUT

Još su dvije karakteristike modula koje bi autor u ovom radu htio spomenuti.

FAN-IN modula je broj njegovih neposredno nadređenih modula. U pravilu trebao bi biti što veći jer to tada znači da se modul u cijelom sistemu često koristi te da je na neki način univerzalan. Kada nadređeni moduli pozivaju podređene, treba imati u vidu da je potrebno da svaka takva veza mora imati isti broj i tip parametara koji se prenose.

FAN-OUT modula je broj njegovih neposredno podređenih modula. Preporučuje se da ovaj broj ne treba biti veći od 7 jer se tako ograničava složenost povezanosti modula s podređenim modulima.



Slika 4. Identificiranje tipa kohezije modula.
(Izvor: STRAH 89)

Koriste se tzv. metrike toka informacija koje se definiraju pomoću FAN-IN, FAN-OUT i LOC[4] (prema (CONTE 86)):

fan-in * fan-out

$(\text{fan-in} * \text{fan-out})^2$

$S * (\text{fan-in} * \text{fan-out})^2$

gdje je S veličina modula u LOC.

4 LOC je kratica za Line Of Code (engl.: linija izvornog koda programa).

5 ZAKLJUČAK

Modularnost kao tehnika programiranja ima velik utjecaj na kvalitetu softvera. Općenito vrijedi pretpostavka da se može razviti kvalitetan softver ukoliko se pridržavamo i zahtjeva koje nameće metodologija strukturnog dizajna.

Kvalitetan kompjutorski program ima niske troškove održavanja. Ukoliko se složimo s ovom tvrdnjom, tada možemo odmah uvidjeti da postoji veza između složenosti, razumljivosti i jednostavnosti kompjutorskog programa, na jednoj strani, i produktivnosti i potrebnog napora/vremena na održavanju kompjutorskih programa, na drugoj strani.

Metodologija strukturnog dizajna nalaže da se kod dekompozicije sistema na podsisteme te kod faza životnog ciklusa softvera, kao što su Dizajn programskog proizvoda i Detaljni dizajn, pridržavamo zahtjeva koji su vezani na povezanost i koheziju modula.

Tablica 1 prikazuje odnos između tipova kohezije modula i njihov utjecaj na razvoj i održavanje softvera.

TIP KOHEZIJE	POVEZA-NOST	IMPLEMEN-TACIJA	IZMJENJI-VOST	RAZUMLJI-VOST	EFEKT NA CIJELI SISTEM
FUNKC.	dobra	dobra	dobra	dobra	dobar
SEKVEN.	dobra	dobra	dobra	dobra	vrlo dobar
KOMUNIK.	srednja	srednja	srednja	srednja	srednji
PROCED.	srednja	srednja	srednja	srednja	loš
VREMEN.	slaba	srednja	srednja	srednja	loš
LOGIČKA	loša	loša	loša	slaba	loš
SLUČAJ.	loša	loša	loša	loša	loš

TABELA 1. USPOREDBA TIPOVA KOHEZIJE MODULA
(Izvor: (STRAH 89))

LITERATURA:

(BOEHM 81) Boehm, B.W.: "Software Engineering Economics", Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1981.

(CONTE 86) Conte, S.D., Dunsmore, H.E., Shen, V.Y.: "Software Engineering Metrics and Models", The Benjamin/Cummings Publishing Company, Inc., Menlo Park, 1986

(KITCH 90) Kitchenham, B.A., Linkman, S.J.: "Design metrics in practice", Information & Software Technology, Butterworth-Heinemann Ltd, volume 32, broj 4, svibanj 1990.

(RADOVAN 89) Radovan, Mario: "Projektiranje informacijskih sistema", Informator Zagreb, 1989.

(STRAH 89) Strahonja, V., Novak, I., Antonović, V.: "Razvoj informacijskih sistema", skripta Centra za permanentno obrazovanje Fakulteta organizacije i informatike Varaždin, 1989.

Primljeno: 1991-09-03

Oreški P. Modularity and Software Quality

SUMMARY

In this paper author discusses about modularity and its influence on the software quality. Among many software features there are complexity, clearness and simplicity. In order to reduce the complexity of the software source code and to increase the clearness and simplicity, programmers use programming technique called modular programming. It is the software design and development techique by which software application is build of discrete parts. In such a way software is easier to maintain. It means that software has higher quality.