Invarijanta klase u objektno orijentisanom programiranju i njena primena

Aleksandar D. Kupusinac, Dušan T. Malbaški, *Member, IEEE*

***Sadržaj* — Jedan od primarnih kriterijuma kvaliteta softverskog sistema jeste korektnost. Fokusirajući razmatranje samo na objektno orijentisane programske jezike, tačnije na problem korektnosti klase dolazi se do pojma invarijanta klase. U radu je data definicija i osnovne osobine invarijante klase, koja zauzima centralno mesto kad se govori o korektnosti objektno orijentisanog programa. Ukazano je na pojedine probleme i na samu primenu pojma invarijante klase i time dat doprinos istraživanju opštih principa proizvodnje kvalitetnog softverskog sistema.**

***Ključne reči* — C++, invarijanta klase, korektnost klase, objektno orijentisano programiranje.**

I. Uvod

SOFTVERSKI inženjering podrazumeva proizvodnju kvalitetnog softverskog sistema, pa je od velikog značaja razvoj teorijskih i praktičnih metoda koje mogu da poboljšaju kvalitet softverskog sistema. Kriterijumi kvaliteta softverskog sistema su mnogobrojni i mogu se podeliti na dve grupe: unutrašnje i spoljašnje kriterijume [1]. Spoljašnji kriterijumi (brzina, jednostavnost korišćenja softvera i sl.) obično su predmet interesovanja krajnjih korisnika, koji nisu profesionalni programeri. Međutim, profesionalnog programera više će interesovati unutrašnji kriterijumi kvaliteta (višekratna upotreba, proširivost, robustnost, korektnost i sl.), jer ključ za postizanje kvaliteta po spoljašnjim kriterijumima, leži u postizanju kvaliteta po unutrašnjim kriterijumima. Drugim rečima, da bi krajnji korisnici mogli da uživaju u vidljivim kvalitetima jednog softverskog sistema, potrebno je da projektanti i programeri prethodno obezbede skrivene kvalitete tog softverskog sistema, što podrazumeva postizanje kvaliteta po unutrašnjim kriterijumima.

Kriterijumi kvaliteta su raznovrsni i nalaze se u stalnom sukobu, stoga je neophodno da se profesionalni programer odluči za optimalnu varijantu i napravi kompromis između sukobljenih kriterijuma kvaliteta u zavisnosti od namene softverskog sistema koji projektuje. Međutim, ma koliko bilo neophodno da se takvi kompromisi prave, postoji jedan kriterijum kvaliteta koji predstavlja izuzetak, a to je korektnost. Ne postoji opravdanje za profesionalnog programera koji zbog postizanja boljeg kvaliteta na osnovu drugih kriterijuma dovede u pitanje korektnost softverskog sistema, pa se zbog toga može zaključiti da korektnost predstavlja jedan od primarnih kriterijuma kvaliteta softverskog sistema. Fokusirajući razmatranje specijalno na korektnost objektno orijentisanog programa, odnosno na razmatranje korektnosti klase dolazi se do pojma invarijante klase. Shodno tome, u ovom radu će biti razmotren pojam korektnosti softverskog sistema i pojam invarijante klase, a zatim razmotrene i osnovne osobine, problemi i primena invarijante klase, s ciljem da se ukaže na značaj istraživanja opštih principa proizvodnje kvalitetnog softverskog sistema.

A. D. Kupusinac, Fakultet tehničkih nauka, Trg Dositeja Obradovića 6, 21000 Novi Sad, Srbija (telefon: 381-21-4852417; faks: 381-21-6350727; e-mail: sasak@uns.ns.ac.yu).

D. T. Malbaški, Fakultet tehničkih nauka, Trg Dositeja Obradovića 6, 21000 Novi Sad, Srbija; (e-mail: malbaski@uns.ns.ac.yu).

II. Korektnost

Korektnost neke programske jedinice (program, klasa, potprogram, metoda) se definiše kao sposobnost date programske jedinice da funkcioniše prema svojoj specifikaciji, pri čemu pod specifikacijom se podrazumeva precizan opis šta taj program treba da radi. Neka je data programska jedinica P i neka je X skup ulaza, a Y skup izlaza u najširem smislu. Specifikacija S date programske jedinice povezuje ulaze sa izlazima i zapravo je binarna relacija između skupova ulaza X i izlaza Y, odnosno $S \subseteq X \times Y$. Programska jedinica P u opštem slučaju takođe je relacija između skupova X i Y koja najčešće ima prirodu funkcije. Domen programa obuhvata samo one ulaze za koje program terminira. Korektnost programske jedinice P se može definisati na sledeći način: za programsku jedinicu P se kaže da je korektna u odnosu na specifikaciju S ako i samo ako važi $dom(P \cap S)=dom(S)$, što znači da za svaki ulaz program terminira i generiše izlaz predviđen specifikacijom [2].

Korektnost je relativan pojam, jer jedna programska jedinica sama za sebe nije ni korektna ni nekorektna. Programska jedinica je korektna, odnosno nekorektna samo u odnosu na neku specifikaciju. Postoje razni načini za zadavanje specifikacije, a jedan od najpoznatijih je pomoću formula totalne i parcijalne korektnosti. Neka se posmatra bilo koja sintaksna jedinica S (deo naredbe, naredba, blok naredbi, potprogram, metoda, program), preduslov P i postuslov Q, koji su po prirodi predikati i koji u različitim stanjima date sintaksne jedinice imaju vrednosti $\top$ i $\bot$. Za datu sintaksnu jedinicu S preduslov određuje stanje u kojem otpočinje izvršavanje S, a postuslov stanje u kojem se S završava. Stanje sintaksne jedinice određuju trenutne vrednosti njenih promenljivih. Formula totalne korektnosti glasi: ako sintaksna jedinica S otpočinje u stanju koje zadovoljava preduslov P, tada S

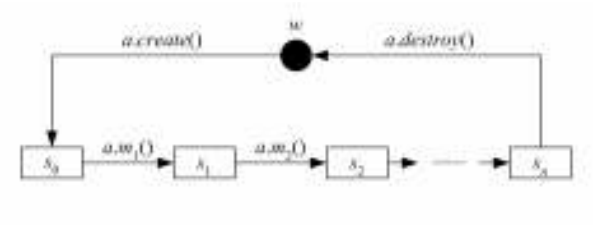
terminira i stanje po završetku S zadovoljava predikat Q, što se simbolički prikazuje $\{P\}S\{Q\}$. Formula parcijalne korektnosti glasi: ako sintaksna jedinica S otpočinje u stanju koje zadovoljava preduslov P i ako S sigurno terminira, tada stanje po završetku S zadovoljava predikat Q, što se simbolički prikazuje $P\{S\}Q$. Obe formule su zapravo predikati i poznatije pod nazivom Hoare-ovi tripleti, u čast njihovog tvorca C.A.R. Hoare-a [3, 4], a u literaturi je zastupljenija formula totalne korektnosti, pa će o njoj biti više reči.

Sada se razmatranje može fokusirati na korektnost potprograma. Ako je specifikacija nekog potprograma f uređen par (P,Q), gde je P preduslov, a Q postuslov, tada je potprogram korektan ako i samo ako je tačan predikat $\{P\}f\{Q\}$. Primera radi, neka je dat potprogram *sort*(a,n), koji služi za sortiranje po rastućem redosledu niza realnih brojeva a dužine n, tada je specifikacija potprograma uređen par (P,Q), gde je predikat $P\equiv$(a je niz realnih brojeva dužine n)$\wedge$($n\geq 2$) preduslov, a predikat $Q\equiv a_i\leq a_{i+1}$, i=1,2,…,n-1 postuslov. Potprogram je korektan u odnosu na specifikaciju (P,Q) ako i samo ako je predikat $\{P\}sort(a,n)\{Q\}$ tačan.

III. KOREKTNOST KLASE

Korektnost realizacije klase razmatra se sa nivoa pojedinačnih metoda i sa nivoa cele klase. Razmatranje korektnosti metoda skoro da je identično sa razmatranjem korektnosti potprograma, s tim da određena razlika proističe iz nesamostalnosti metoda, što implicira da preduslovi, odnosno postuslovi svake metode moraju da izraze relacije između tih podataka članova. Zbog toga postoji potreba za izražavanjem globalnih svojstava objekata, onih koje sve metode moraju da sačuvaju, a takva svojstva čine invarijantu klase. Naime, svi objekti iste klase mogu imati različite podatke članove, ali relacije između tih podataka članova moraju ostati iste. Invarijante mogu takođe da izražavaju semantičke relacije između metoda, ali i između metoda i podataka članova.

Neka je Σ skup svih stanja objekta ili prostor stanja, odnosno $\Sigma=\{\sigma_i \mid i=1,2,...,N\}$. Stanje u kojem se objekat nalazi određuju trenutne vrednosti njegovih podataka članova u "stabilnim" trenucima, a to znači neposredno po stvaranju objekta (početno stanje) i neposredno posle izvršenja svake metode. Sl. 1. ilustruje kretanje objekta a u prostoru stanja. Stanje nepostojanja u kojem objekat a ne postoji označeno je sa w. Objekat a nastaje metodom *a.create*(), posle čega se nalazi u početnom stanju $s_0\in\Sigma$. Izvršavanjem metoda $a.m_1()$, $a.m_2()$ itd. objekat prelazi u različita stanja $s_1,s_2,...,s_n\in\Sigma$. Objekat a se uništava metodom *a.destroy*(), posle čega se ponovo nalazi u stanju nepostojanja w. Stanja $s_0,s_1,s_2,...,s_n$ su proizvoljni elementi skupa svih stanja Σ i određuju putanju kretanja objekta a kroz prostor stanja. Na osnovu specifikacije klase skup svih stanja Σ se može podeliti na skup dozvoljenih stanja Σ_{DS} i skup nedozvoljenih stanja Σ_{NS}. Invarijanta klase I je svaki predikat koji ima vrednost T u svakom dozvoljenom stanju svakog objekta date klase, odnosno $(\forall\sigma_i\in\Sigma_{DS})$ $I(\sigma_i)=\mathsf{T}$. Stroga invarijanta klase I_K je predikat koji ima vrednost T u svakom dozvoljenom stanju svakog objekta date klase, odnosno $(\forall\sigma_i\in\Sigma_{DS})$ $I(\sigma_i)=\mathsf{T}$, a vrednost $\perp$ u svakom nedozvoljenom stanju svakog objekta date klase, odnosno $(\forall\sigma_i\in\Sigma_{NS})$ $I(\sigma_i)=\perp$ [2]. Svaka stroga invarijanta je i invarijanta klase, a obrnuto ne važi. Stroga invarijanta klase jednoznačno opisuje skup dozvoljenih stanja, tj. $\Sigma_{DS}=\{\sigma_i \mid I_K(\sigma_i)=\mathsf{T}\}$, gde je i=1,2,...,N, ali pošto se jedan isti skup može opisati pomoću ekvivalentnih predikata, može se zaključiti da stroga invarijanta klase je jedinstvena do nivoa ekvivalencije, odnosno sve stroge invarijante su ekvivalentne. Na primer, skup $A=\{1,2,3,4,5\}$ može se opisati kao $A=\{x|(x\in N)\wedge(x\leq 5)\}$, gde je N skup prirodnih brojeva, ali i kao $A=\{x|(x\in Z)\wedge(1\leq x\leq 5)\}$, gde je Z skup celih brojeva [5].



Sl. 1. Kretanje objekta a u prostoru stanja.

Prilikom određenja invarijante klase treba voditi računa i o tome da invarijanta klase može u sebi imati redundantni deo, koji sa jedne strane ne menja njene osobine i ne donosi ništa novo, a sa druge strane nepotrebno je usložnjava. Na primer, skup $A=\{1,2,3,4,5\}$ se može opisati kao $A=\{x|(x\in N)\wedge(1\leq x\leq 5)\}$, ali kada je već rečeno da je $x\in N$, onda je nepotrebno reći da je $1\leq x\leq 5$, jer svaki prirodan broj je veći od nule, stoga je dovoljno reći $x\leq 5$.

Ako se klasa posmatra kao model klase entiteta, onda je očigledno da pravi smisao invarijante klase jeste da dosledno očuva untrašnje relacije koje važe između atributa svakog konkretnog entiteta iz date klase entiteta [2]. Na primer, svaki konkretan entitet «trougao» ima tri stranice koje mogu imati različite dužine stranica, ali su sve dužine pozitivni realni brojevi (jer nepostoji trougao koji ima stranicu negativne dužine) i za njegove stranice važi da je zbir dužina svake dve stranice veći od dužine treće stranice. Ako se projektant odluči da modeluje klasu entiteta «Trougao», tako što će za relevantne osobine uzeti dužine stranica, pa će klasa (objekata) *Trougao* imati tri podatka člana a, b i c, tada predikati kao što su $I_1\equiv(a>0)$, $I_2\equiv(b>0)$, $I_3\equiv(c>0)$, $I_4\equiv(a+b>c)$, $I_5\equiv(b+c>a)$ $I_6\equiv(c+a>b)$ itd. su invarijante klase. Stroga invarijanta klase Trougao biće predikat $I_K\equiv I_1\wedge I_2\wedge I_3\wedge I_4\wedge I_5\wedge I_6$. Sad se može zaključiti da se nijedan objekat klase ne sme nalaziti u nedozvoljenom stanju, tj. u stanju u kojem je vrednost stroge invarijante klase $\perp$, jer bi to značilo da takav objekat predstavlja model entiteta koji ne postoji. Dakle, stroga invarijanta klase obezbeđuje da svaki objekat bude u dozvoljenom stanju, a to znači da bude verodostojan model entiteta (koji može postojati u domenu problema).

Već je rečeno da razmatranje korektnosti metoda skoro da je identično sa razmatranjem korektnosti potprograma, s tim da određena razlika proističe iz nesamostalnosti

metoda, što implicira da preduslovi, odnosno postuslovi svake metode moraju se modifikovati. Metode za kreiranje (konstruktor) i uništavanje objekata (destruktor) specifične su po tome što konstruktor stvara objekat i iz stanja nepostojanja dovodi ga u početno stanje, a destruktor ga uništava i iz nekog stanja dovodi ga u stanje nepostojanja. Može se uvesti predikat $W \equiv$"Objekat se nalazi u stanju nepostojanja". Ako je c konstruktor date klase sa preduslovom P_c i postuslovom Q_c, a d destruktor sa preduslovom P_d i postuslovom Q_d i ako je I_K stroga invarijanta klase, tada je konstruktor korektan ako i samo ako je predikat $\{W \wedge P_c\} c \{Q_c \wedge I_K\}$ tačan, a destruktor ako i samo ako je predikat $\{P_d \wedge I_K\} d \{Q_d \wedge W\}$ tačan. Promene stanja i kretanje objekta kroz prostor stanja izvodi se posredstvom ostalih metoda, stoga za sve ostale metode važi da se njihovim preduslovima i postuslovima mora očuvati stroga invarijanta klase, jer stroga invarijanta klase mora biti zadovoljena u svakom stanju u kojem se objekat nalazi. Ako je m metoda date klase sa preduslovom P_m i postuslovom Q_m i ako je I_K stroga invarijanta klase, tada je metoda korektna ako i samo ako je predikat $\{P_m \wedge I_K\} m \{Q_m \wedge I_K\}$ tačan. Sada se može doneti zaključak o korektnosti cele klase u celini. Neka je I_K stroga invarijanta klase, tada je data klasa korektna ako i samo ako su tačni predikati:

- $\{W \wedge P_c\} c \{Q_c \wedge I_K\}$, za svaki konstruktor c,
- $\{P_d \wedge I_K\} d \{Q_d \wedge W\}$, za destruktor d,
- $\{P_m \wedge I_K\} m \{Q_m \wedge I_K\}$, za svaku metodu m.

Invarijanta klase-naslednice I_N može se prestaviti kao konjukcija invarijanti klasa-predaka I_{P1}, I_{P2},..., I_{Pn} i dela koji je specifičan za samu klasu-naslednicu N, odnosno $I_N \equiv I_{P1} \wedge I_{P2} \wedge ... \wedge I_{Pn} \wedge N$.

IV. PRIMER I PRIMENA

Primena i značaj invarijante klase će biti objašnjeni na primerima klase *Trougao* i klase *PravougliTrougao*, pri čemu treba imati na umu da naredni primeri imaju isključivo teorijski značaj i služe kao ilustracija onoga što je u ovom radu objašnjeno. Stroga invarijanta klase *Trougao*, koja ima tri podatka člana a, b i c je predikat $I_T \equiv (a>0) \wedge (b>0) \wedge (c>0) \wedge (a+b>c) \wedge (b+c>a) \wedge (c+a>b)$. Svaki objekat klase *Trougao* koji se nalazi u dozvoljenom stanju, ima vrednosti podataka članova, takve da predikat I_T je tačan. Sl. 2. prikazuje primer realizacije metode *invariant*() u okviru klase *Trougao*, na programskom jeziku C++.

```
void Trougao::invariant() {
    if((a>0)&&(b>0)&&(c>0)&&
       ((a+b)>c)&&
       ((b+c)>a)&&
       ((c+a)>b))
      return;
    else
      throw IRREGULAR_STATE;
}
```

Sl. 2. Primer realizacije metode *invariant*() u okviru klase *Trougao*

Projektant klase je taj koji na osnovu specifikacije klase realizuje metodu *invariant*(), a obaveza svakog klijenta koji koristi datu klasu je da se pridržava napisane invarijante klase. Invarijanta klase može se opisati kao ugovor koji sadrži opšte klauzule ili pravila koja su primenjiva na sve pojedinačne ugovore između metoda klase i klijenta [6]. Projektant klase je odgovoran da klasa bude korektna, tj. napisana u skladu sa specifikacijom, a odgovornost klijenta je da poštuje napisanu invarijantu klase. Sl. 3. prikazuje jednostavan primer na programskom jeziku C++ pomoću *try-catch* bloka naredbi za klijenta koji koristi klasu *Trougao*. Posle poziva konstruktora i stvaranja objekta t klase *Trougao*, na osnovu zadatih parametara koje je zadao klijent, klijent je obavezan da pozove metodu *t.invariant*() da bi utvrdio da li je početno stanje objekta s_0 dozvoljeno stanje, odnosno da li važi $s_0 \in \Sigma_D$. Zatim, klijent poziva ostale metode. Posle metode koja ne menja stanje objekta, kao što je na primer, metoda *t.getArea*() klijent nije obavezan da pozove metodu *t.invariant*(), ali posle metode koja menja stanje objekta, kao što je na primer metoda *t.setA*() klijent je obavezan da pozove metodu *t.invariant*() i utvrdi da li je objekat u dozvoljenom stanju.

```
try {
      Trougao t(3,4,5);
      t.invariant();
      cout<<t.getArea();
      t.setA(2);
      t.invariant();
      cout<<t.getArea();
      t.setA(1);
      t.invariant();
}
catch(Exceptions e) {
      if(e==IRREGULAR_STATE) {
        ...
      }
}
```

Sl. 3. Primer *try-catch* bloka naredbi za klijenta koji koristi klasu *Trougao*

Početno stanje objekta t je (3,4,5). Pošto je 3>0, 4>0, 5>0, 3+4>5, 4+5>3 i 5+3>4, očigledno je da predikat I_T u ovom stanju ima vrednost $\top$ i da se objekat t nalazi u dozvoljenom stanju. Pozivom metode *t.setA*(2) objekat prelazi u stanje (2,4,5). Pošto je 2>0, 4>0, 5>0, 2+4>5, 4+5>2 i 5+2>4, očigledno je da predikat I_T i u ovom stanju ima vrednost $\top$ i da se objekat t nalazi u dozvoljenom stanju. Metoda *getArea*() izračunava površinu trougla na osnovu trenutnih vrednosti podataka članova i kao takva ne menja stanje objekta i ne može narušiti strogu invarijantu klase, odnosno ako je stroga invarijanta klase bila zadovoljena pre poziva ove metode, onda će sigurno biti zadovoljena i posle. Pozivom metode *t.setA*(1) objekat prelazi u stanje (1,4,5). Pošto nije 1+4>5 predikat I_T u ovom stanju ima vrednost $\bot$ i objekat t se nalazi u nedozvoljenom stanju. Iz ovog primera se jasno vidi da stroga invarijanta klase obezbeđuje da svaki objekat bude verodostojan model entiteta (koji može postojati u domenu problema), jer entitet «trougao» sa stranicama 3,4 i 5 može postojati, dok entitet «trougao» sa stranicama 1,4 i 5 ne može postojati.

Klasa *PravougliTrougao* se može izvesti iz klase *Trougao*. Već je rečeno da invarijanta klase-naslednice može se prestaviti kao konjukcija invarijanti klasa-predaka i dela koji je specifičan za samu klasu-naslednicu. To što je specifično za samu klasu *PravougliTrougao* je da je kvadrat nad hipotenuzom jednak zbiru kvadrata nad katetama, pa neka je I_T stroga invarijanta klase *Trougao*, tada je $I_{PT} \equiv I_T \wedge ((a^2=b^2+c^2) \vee (b^2=c^2+a^2) \vee (c^2=a^2+b^2))$ stroga invarijanta klase *PravougliTrougao*. Sl. 4. prikazuje primer realizacije metode *invariant*() u okviru klase *PravougliTrougao*, na programskom jeziku C++.

```
void PravougliTrougao::invariant() {
    if(Trougao::invariant()&&
      ((a*a==b*b+c*c)||
       (b*b==c*c+a*a)||
       (c*c==a*a+b*b)))
      return;
    else
      throw IRREGULAR_STATE;
}
```

Sl. 4. Primer realizacije metode *invariant()* u okviru klase *PravougliTrougao*

Već je rečeno da projektant klase jeste odgovoran da klasa bude korektna, tj. napisana u skladu sa specifikacijom, a odgovornost klijenta je da ispoštuje napisanu invarijantu klase. Sl. 5. prikazuje primer na programskom jeziku C++ pomoću *try-catch* bloka naredbi za klijenta koji koristi klasu *PravougliTrougao*, koja je izvedena iz klase *Trougao*. Posle poziva konstruktora i stvaranja objekta *t* klase *PravougliTrougao*, na osnovu zadatih parametara koje je zadao klijent, klijent je obavezan da pozove metodu *t.invariant*() da bi utvrdio da li je početno stanje objekta s_0 dozvoljeno stanje, odnosno da li važi $s_0 \in \Sigma_D$.

```
try {
      PravougliTrougao t(3,4,5);
      t.invariant();
      cout<<t.getArea();
      t.setA(2);
      t.invariant();
}
catch(Exceptions e) {
      if(e==IRREGULAR_STATE) {
        ...
      }
}
```

Sl. 5. Primer *try-catch* bloka naredbi za klijenta koji koristi klasu *PravougliTrougao*

Početno stanje objekta t je (3,4,5). Pošto je već pokazano da invarijanta klase-pretka u stanju (3,4,5) je tačna, ostaje još da se ispita tačnost onog dela koji je specifičan za samu klasu-naslednicu. Pošto očigledno važi da je $5^2=3^2+4^2$ invarijanta klase-naslednice u ovom stanju ima vrednost ⊤ i objekat *t* se nalazi u dozvoljenom stanju. Pozivom metode *t.setA*(2) objekat prelazi u stanje (2,4,5). Pošto $2^2 \neq 4^2+5^2$, $4^2 \neq 5^2+2^2$ i $5^2 \neq 2^2+4^2$ očigledno je da invarijanta klase *PravougliTrougao* u ovom stanju ima vrednost ⊥ i da se objekat *t* nalazi u nedozvoljenom stanju.

V. ZAKLJUČAK

Invarijanta klase je svaki predikat koji je tačan u svakom dozvoljenom stanju svakog objekta date klase, pa zbog toga zauzima centralno mesto kada se razmatra korektnost objektno orijentisanog programa. Ako se klasa posmatra kao model klase entiteta, onda je očigledno da pravi smisao invarijante klase jeste da dosledno očuva unutrašnje relacije koje važe između atributa svakog konkretnog entiteta iz date klase entiteta i obezbedi da svaki objekat bude u dozvoljenom stanju, a to znači da bude verodostojan model entiteta, koji može postojati u domenu problema. Odrediti invarijantu klase, a pogotovo strogu invarijantu klase po običaju nije nimalo trivijalno, ali bez obzira na teškoće ovakvo istraživanje predstavlja temelj proizvodnje kvalitetnog softvera i predstavlja veliki izazov za dalji rad i istraživanje metoda i algoritama za određivanje invarijante klase, kao i metoda za njenu primenu u razvoju softverskih sistema.

LITERATURA

[1] B. Meyer, *Object-Oriented Software Construction.* Prentice Hall, 1988.

[2] D. T. Malbaški, *Objekti i objektno programiranje kroz programske jezike C++ i Pascal*. Novi Sad: Fakultet tehničkih nauka, 2006.

[3] M. J. C Gordon, *Programming Language Theory and its implementation*. Prentice Hall, 1988.

[4] E. W. Dijkstra, *A Discipline of Programming*. Prentice Hall, 1976.

[5] D. Cvetković, S. Simić, *Diskretna matematika – matematika za kompjuterske nauke*. Beograd: Naučna knjiga, 1990.

[6] M. Barnett, R. DeLine, M. Fähndrich, K. R. M. Leino, W. Schulte: "Verification of object-oriented programs with invariants," in *Journal of Object Technology*, vol. 3, no. 6, June 2004, Special issue: ECOOP 2003 workshop on FTfJP, pp. 27-56, http://www.jot.fm/issues/issue_2004_06/article2

ABSTRACT

One of the basic criteria for determining software system quality is correctness. Focusing on object oriented programming languages only, primarily on class correctness problem, we arrive at the concept of class invariant. Class invariant is the central point of correctness consideration of object oriented programme and its definition and characteristics are given in this paper. Some problems have been highlighted and application of class invariant concept have been analysed and in that way a contribution has been given to the research of common principles of quality software system design.

CLASS INVARIANT IN OBJECT ORIENTED PROGRAMMING AND ITS APPLICATION

Aleksandar D. Kupusinac, Dušan T. Malbaški