Metodologija testiranja softverskog proizvoda

Slavica Boštjančič, Mirjana Stojanović, Ranko Nedeljković

***Sadržaj* — U radu su prvo opisane mere pouzdanosti koje predstavljaju važan aspekt kvaliteta softverskog proizvoda. Zatim je predložena fazna metodologija testiranja softvera koja omogućava sistematično pronalaženje i uklanjanje grešaka u razvoju softvera. Takođe je opisana primena predložene metodologije na testiranje objektno-orijentisane aplikacije za ugovaranje kvaliteta servisa u IP mreži.**

***Ključne reči* — kvalitet softvera, objektno-orijentisano projektovanje, pouzdanost softvera, sporazum o nivou servisa, testiranje softvera.**

I. Uvod

KVALITET softverskog proizvoda određen je brojnim faktorima, kao što su: saglasnost sa funkcionalnom i programskom specifikacijom, tačnost, potpunost, održavanje, skalabilnost, portabilnost, dokumentacija i dr.

Pouzdanost softvera je bitna karakteristika kvaliteta. U IEEE standardu 610.12 [1] pouzdanost softvera se definiše kao "sposobnost sistema ili komponente sistema da izvodi zahtevane funkcije pod određenim uslovima u određenom vremenskom periodu." Iako postoje različite metode obezbeđivanja i određivanja pouzdanosti softvera, nijedna od njih nije standardizovana i primenljiva za svaki softverski proizvod [2].

U ovom radu je razmatrana pouzdanost kao mera kvaliteta softvera. S obzirom da se procena pouzdanosti zasniva na analizi otkaza, predložena je fazna metodologija testiranja softvera koju čine: debagovanje, testiranje ispravnosti, testiranje performansi, testiranje pouzdanosti i testiranje bezbednosti.

Predložena metodologija primenjena je na testiranje aplikacije za ugovaranje kvaliteta servisa (*Quality of Service,* QoS) u telekomunikacionim mrežama zasnovanim na tehnologiji Internet protokola (IP).

II. Pouzdanost kao mera kvaliteta softvera

Procena pouzdanosti zasniva se na analizi otkaza softvera i njihovog uticaja na celokupan sistem. Otkaz softvera može biti posledica grešaka u kodu, pogrešnog korišćenja, pogrešne interpretacije specifikacije koju softver treba da zadovolji ili nestručnosti programera, primene neodgovarajućih testova ili drugih nepredviđenih problema.

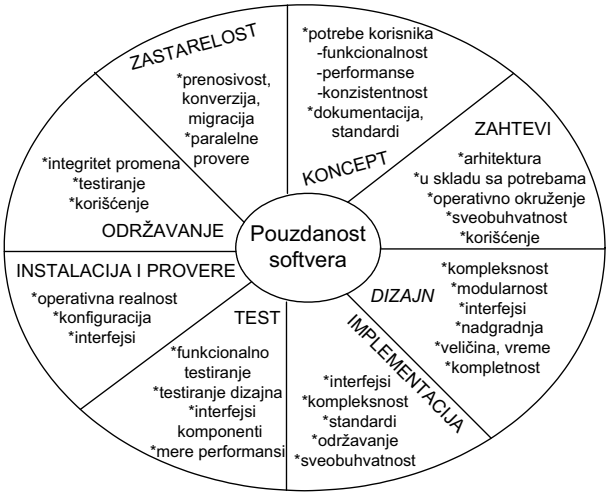
S. Boštjančič, Institut Mihajlo Pupin, Volgina 15, 11050 Beograd, Srbija (telefon: 381-11-775460; e-mail: slavica@kondor.imp.bg.ac.yu).

M. Stojanović, Institut Mihajlo Pupin, Volgina 15, 11050 Beograd, Srbija; (e-mail: stojmir@kondor.imp.bg.ac.yu).

R. Nedeljković, Saobraćajni fakultet u Beogradu, Vojvode Stepe 305, 11000 Beograd, Srbija; (e-mail: rankon@EUnet.yu).

Karakteristika otkaza softvera u fazi operativnog ciklusa beleži pikove koji ukazuju na povećanje broja otkaza usled nadogradnje softvera. U poslednjoj fazi ciklusa nema povećanja broja otkaza s obzirom da softver ulazi u fazu zastarelosti u kojoj nema više potrebe za nadogradnjom.

Obezbeđivanje visoke pouzdanosti softvera zavisi od parametara kvaliteta softvera u svakoj od faza razvojnog ciklusa. U svakoj fazi neophodni su parametri pomoću kojih se vrši merenje karakteristika kvaliteta. U IEEE standardu 982.2 [3] prikazan je uticaj svih faza razvojnog ciklusa na pouzdanost softvera (slika 1).



Sl. 1. Parametri kvaliteta koji utiču na pouzdanost [3]

Da bi se povećala pouzdanost softvera prevencijom softverskih grešaka, neophodno je identifikovati i izmeriti karakteristike kvaliteta softvera u svakoj od faza ciklusa, pri čemu se najviše pažnje posvećuje **zahtevima**, **dizajnu**, **implementaciji** i **testiranju**. Zahtevi moraju biti jasno i nedvosmisleno definisani, a plan testiranja mora da bude sveobuhvatan.

Procena pouzdanosti softvera vrši se pomoću različitih mera [4]:

(1) **Mere konačnog proizvoda** Veličina softvera govori o kompleksnosti softvera. Osnovni način određivanja veličine softvera jeste određivanje broja linija koda.

Za potrebe analize kvaliteta objektno orijentisanog softvera potrebne su dodatne mere. U [5] su predložene sledeće mere za analizu kvaliteta objektno orijentisanog softvera: težinske metode po klasi, odziv po klasi, sprega između objekata, veličina stabla i broj izvedenih klasa.

(2) **Mere upravljanja projektom.** Pokazalo se da postoji veza između procesa izrade i mogućnosti realizovanja projekta na vreme sa odgovarajućim kvalitetom. Veća pouzdanost se postiže korišćenjem boljeg procesa razvoja, boljeg procesa upravljanja rizicima, boljeg procesa upravljanja konfiguracijom itd.

(3) **Mere procesa izrade softvera.** Bazirajući se na pretpostavci da kvalitet proizvoda predstavlja direktnu funkciju procesa izrade, ove mere koriste se za estimaciju, nadzor i povećanje pouzdanosti i kvaliteta softvera.

(4) **Mere grešaka i otkaza**. Obično se ovi parametri zasnivaju na informacijama korisnika o greškama koje su oni otkrili. Na osnovu prikupljenih podataka o otkazima i greškama može da se izračuna srednje vreme do otkaza (MTBF, *Mean Time Before Failure*) i ostali parametri koji određuju ili predviđaju pouzdanost softvera.

(5) **Mere zahteva za pouzdanost.** U cilju razvoja pouzdanog softvera, specifikacija zahteva ne sme da sadrži dvosmislene izraze, opcione zahteve, zahteve sa višestrukim značenjem. Programeru ne sme da se ostavi izbor da li da neki zahtev bude implementiran ili ne.

III. METODOLOGIJA TESTIRANJA SOFTVERA

Testiranje softvera je proces verifikacije programa ili sistema sa ciljem otkrivanja i ispravljanja grešaka i predstavlja integralni deo razvoja softvera. Primenjuje se u svakoj fazi razvojnog ciklusa [2], a obuhvata više od 50% vremena potrebnog za razvoj softvera.

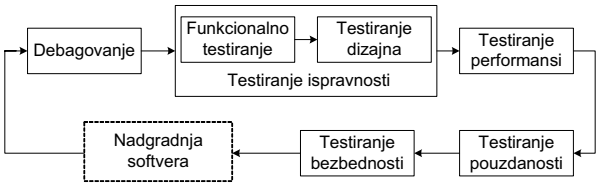
Ciljevi testiranja softvera su [6]:

(1) Verifikacija i validacija, kojima se proverava da li je softver saglasan sa specifikacijom zahteva [2], što ne znači uvek da je softver tehnički ispravan, pouzdan i bezbedan.

(2) Poboljšanje kvaliteta softvera.

(3) Procena pouzdanosti.

Metodologija testiranja softverskog proizvoda, koja se predlaže u ovom radu, podrazumeva ciklus koji se sastoji iz sledećih faza: debagovanje, testiranje ispravnosti, testiranje performansi, testiranje pouzdanosti i testiranje bezbednosti (slika 1).



Sl.1. Metodologija testiranja softverskog proizvoda

Debagovanje podrazumeva otklanjanje sintaksičkih i logičkih grešaka tokom razvoja aplikacije.

Testiranje ispravnosti obuhvata sledeće metode:

(a) **Funkcionalno testiranje (metoda crne kutije – *"Black-box"*).** Analizira se samo izvršavanje specificiranih funkcija i vrši se provera ulaznih i izlaznih podataka. Tačnost izlaznih podataka proverava se na osnovu specifikacije zahteva za softver. U ovim testovima se ne vrši analiza izvornog koda. Problem funkcionalnog testiranja može da se pojavi u slučaju dvosmislenih zahteva i nemogućnosti opisivanja svih načina korišćenja softvera. Skoro 30% svih grešaka u kodu posledica su problema nepotpunih ili neodređenih specifikacija [2].

(b) **Testiranje dizajna softvera (metoda bele kutije – *"White-box"*).** Ovo testiranje proverava i analizira izvorni kod i zahteva dobro poznavanje programiranja, odgovarajućeg programskog jezika, kao i dizajna konkretnog softverskog proizvoda. Plan testiranja se određuje na osnovu elemenata implementacije softvera, kao što su programski jezik, logika i stilovi. Testovi se izvode na osnovu strukture programa. Kod ove metode postoji mogućnost provere skoro celokupnog koda, na primer proverom da li se svaka linija koda izvršava barem jednom, proverom svih funkcija ili proverom svih mogućih kombinacija različitih programskih naredbi. Specifičnim testovima može se proveravati postojanje beskonačnih petlji ili koda koji se nikada ne izvršava.

Testiranje performansi obuhvata pronalaženje i otklanjanje problema koji degradiraju performanse softvera. Ispitivanje performansi najčešće obuhvata: iskorišćenje procesorskih resursa, protok podataka i vreme odziva. Tipični resursi koji se proveravaju su: propusni opseg, brzina procesora, iskorišćenje memorije, zauzetost prostora na disku.

Testiranje pouzdanosti određuje verovatnoću funkcionisanja softvera bez greške i bez otkaza. Testiranje robusnosti i testiranje opterećenjem su primeri testova pouzdanosti, koji pokušavaju da izazovu otkaz softvera u određenim situacijama.

Svrha **testiranja bezbednosti** je identifikacija i otklanjanje grešaka koje potencijalno ugrožavaju bezbednost, kao i validacija efikasnosti mera zaštita. Simulacijom napada mogu se pronaći osetljiva mesta softvera.

IV. SLAM: PROTOTIP APLIKACIJE ZA UGOVARANJE KVALITETA SERVISA U DIFFSERV IP MREŽI

SLAM (*SLA Manager*) je prototip aplikacije za ugovaranje QoS u IP mrežama sa diferenciranim servisima (*Differentiated Services*, DiffServ [7]). Razvijen je i testiran u Institutu Mihajlo Pupin u Beogradu. Ugovaranje QoS rezultuje sporazumom o nivou servisa između provajdera i korisnika (krajnjeg korisnika ili drugog administrativnog domena) – SLA (*Service Level Agreement*). SLA sadrži sve tehničke aspekte servisa (parametri QoS i profili saobraćaja), kao i ekonomske i pravne aspekte.

A Funkcionalna specifikacija

SLAM treba da omogući sledeće funkcije:

(1) Korisniku: kreiranje naloga odnosno korisničkog imena i lozinke; logovanje sa mogučnošću promene lozinke i ličnih podataka; ugovaranje novog servisa sa mogučnošću izbora između pet klasa servisa (premium, zlatna, srebrna, bronzana i klasa najboljeg pokušaja) i vrednosti tehničkih parametara koji definišu profil ulaznog saobraćaja (prosečni, vršni protok, prosečna, vršna eksplozivnost saobraćaja, diferencijacija putanji i raspored

servisa); reugovaranje odnosno modifikaciju postojećih ugovora; uvid u postojeće ugovore u vidu izveštaja; raskid ugovora.

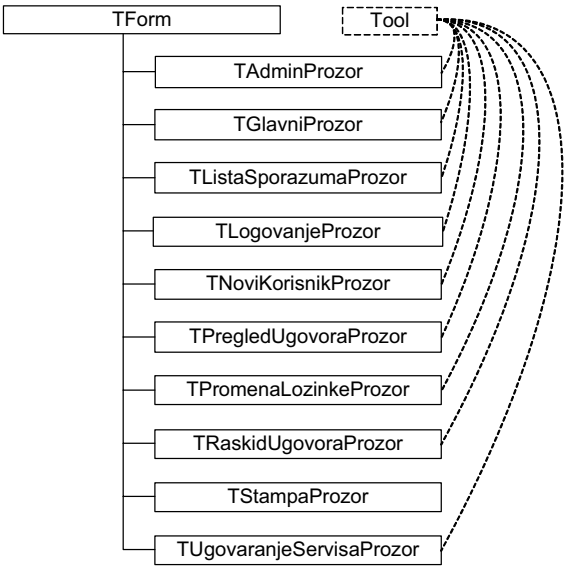
(2) **Administratoru:** pregled podataka vezanih za korisnike i njihove ugovore; pretraživanje ugovora i korisnika.

Na osnovu funkcionalne specifikacije definisani su struktura i format SLA. Tehnički deo SLA obuhvata skup deskriptora koji parametarski opisuju tokove i profile saobraćaja, mere performansi, diferencijaciju putanja, raspoloživost i pouzdanost servisa i dr. [8].

B Objektno orijentisani dizajn

Logička struktura aplikacije SLAM odgovara funkcionalnoj specifikaciji i obuhvata tri agenta: (1) prvi agent je korisnički orijentisan i služi za ugovaranje odgovarajućeg nivoa kvaliteta servisa; (2) drugi agent simulira funkcije entiteta za upravljanje resursima mreže; (3) treći agent je namenjen administratoru i omogućava pregled svih ugovorenih sporazuma.

SLAM je razvijena objektno orijentisanim metodama projektovanja softvera u programskom jeziku C++. Organizovana je u nekoliko prozora koji se naizmenično pojavljuju izborom određenih komandi. Korisnički interfejs aplikacije SLAM detaljno je prikazan u [9].



Sl. 1. SLAM: hijerarhija i interakcija klasa

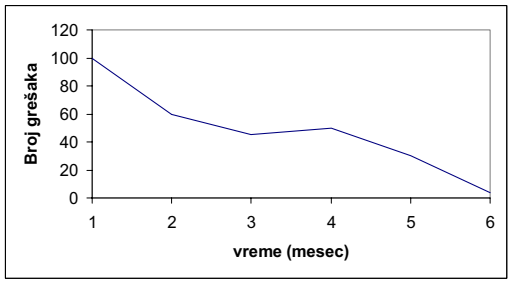
SLAM predstavlja tipičnu *Windows* aplikaciju odnosno skup prozora koji se naizmenično pojavljuju izborom određenih komanadi (izborom stavki iz menija, prečica sa tastature, izborom određenog "dugmeta" itd.), pa su tako formirane i klase [10]. Sve klase osim jedne (`Tool`) izvedene su iz klase `TForm`, koja ustvari definiše prozor odnosno formu. Klasa `Tool` je autentična klasa aplikacije (nije izvedena klasa) u kojoj su smeštene sve globalne funkcije i globalne promenljive koje koriste funkcije ostalih klasa. Na slici 1 prikazana je hijerarhija klasa i njihova međusobna interakcija.

Glavni prozor aplikacije SLAM predstavlja glavnu formu iz koje se granaju svi ostali prozori. `Klasa TAdminProzor` odgovara agentu koji je namenjen administratoru. Klasa `TNoviKorisnikProzor` omogućava novom korisniku kreiranje naloga. Klase `TLogovanjeProzor` i `TPromenaLozinkeProzor` obezbeđuju logovanje i promenu lozinke korisnika, respektivno. Klasa `TUgovaranjeProzor` omogućava autorizovanim korisnicima ugovaranje novog SLA ili reugovaranje postojećeg SLA. Klase `TListaSporazumaProzor`, `PregledUgovoraProzor` i `TRaskidUgovoraProzor` omogućavaju autorizovanom korisniku prikaz liste njegovih ugovora, pregled i raskid izabranog SLA, respektivno. Klasa `TStampaProzor` odgovorna je za štampanje izabranog izveštaja.

V TESTIRANJE APLIKACIJE SLAM

Metodologija testiranja softvera predložena u poglavlju III primenjena je na testiranje aplikacije SLAM.

Debagovanje koda aplikacije. Na slici 2 prikazana je faza otklanjanja grešaka u kodu tokom 6 meseci koliko je trajao razvoj i testiranje aplikacije. Broj grešaka je najveći na početku, što je posledica prisustva sintaksičkih grešaka. Nakon njihovog otklanjanja ostaju logičke greške. Otklanjanje logičkih grešaka može izazvati pojavu novih grešaka, što se uočava na slici 2 sa pikom oko četvrtog meseca. Nakon toga, nastavlja se opadanje broja grešaka.



Sl. 2. Faza otklanjanja grešaka u kodu aplikacije

Funkcionalno testiranje je izvršeno na osnovu funkcionalne specifikacije, čime je verifikovano da su svi specificirani zahtevi obezbeđeni aplikacijom [11].

Testiranje dizajna softvera obuhvatilo je utvrđivanje statističkih podataka, testiranje interakcija objekata aplikacije i određeni broj testova koji su se odnosili na otkrivanje viška koda, postojanje beskonačnih petlji i sl.

Aplikacija je dizajnirana modularno, uz obezbeđenje visokog nivoa portabilnosti i mogućnosti efikasne nadgradnje. Statistički podaci o softveru (tabele 1 i 2) zadovoljavaju preporuke iz [5].

TABELA 1: STATISTIČKI PODACI O MODULIMA

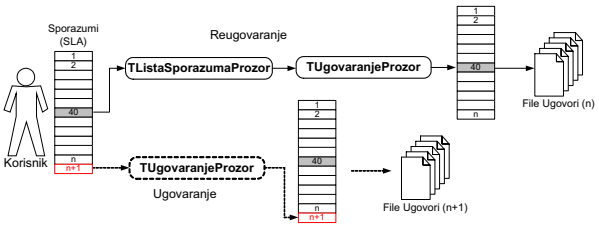
Broj modula (fajlova)	Ukupan broj linija koda	Ukupan broj funkcija	Izvedene klase
*.cpp - 13	1605	117	Sve, osim klase `Tool`
*.h -13			

Na slici 3 je prikazan primer **testiranja interakcije** objekata u aplikaciji SLAM, odnosno dela koda koji izvršava ugovaranje i reugovaranje SLA.

TABELA 2: STATISTIČKI PODACI O KLASAMA

Klasa aplikacije SLAM	Broj metoda po klasi	Broj metoda preporučen u [4]
TNoviKorisnikProzor	11	20 – 40
TLogovanjeProzor	10	
TGlavniProzor	25	
TUgovoranjeProzor	12	
TListaSporazuma	8	
TPregledUgovoraProzor	5	
TPromenaLozinkeProzor	6	
TRaskidUgovoraProzor	7	
TAdminProzor	21	
Tool	12	

Dodatno testiranje dizajna (višak koda, beskonačne petlje, opseg podataka i dr.) izvršeno je polazeći od specificiranih funkcija, analizom izvršavanja programa pomoću *debuggera* pridruženog razvojnom alatu. Specificirano je 20 testova kojima se vrši provera načina izvršavanja programa u neregularnim situacijama, koje u slučaju aplikacije SLAM prvenstveno potiču od korisnika (nepravilna manipulacija, unos podataka van dozvoljenog opsega, zlonamernost i dr.). Iako je broj testova ograničen (zbog troškova projekta) vreme potrebno za njihovo izvršavanje iznosi približno 25% od ukupnog vremena potrebnog za razvoj i testiranje aplikacije.



Sl. 3. Testiranje interakcije objekata

Testiranje performansi. Aplikacija SLAM instalira se na standardnom PC računaru sa operativnim sistemom Windows 98/2000/XP. Minimalni potrebni računarski resursi su 128MB RAM memorije, brzina procesora 500MHz i 12MB za instalaciju na hard disku, pri čemu treba predvideti dodatni prostor za čuvanje ugovora. Podaci o jednom SLA prosečno zauzimaju 10kB u odgovarajućoj bazi, što ukazuje da resursi hard diska nisu kritični, čak i za veliki broj ugovora. Procesorski resursi takođe nisu kritični, jer je ugovaranje servisa interaktivno (aplikacija se ne izvršava u realnom vremenu).

Testiranje pouzdanosti obuhvatilo je testove robusnosti aplikacije. Aplikacija je data na korišćenje korisnicima različitih profila, u trajanju od 7 dana. Četiri korisnika otkrila su 12 grešaka kao što su na primer: komanda ne radi na očekivan način, prikazuju se pogrešni podaci u izveštajima, ažuriranje pojedinih podataka nije trenutno i sl. Nijedna od uočenih grešaka nije izazvala blokiranje programa.

Testiranjem bezbednosti utvrđeno je da postoji zaštita vertikalnog i horizontalnog pristupa [11]. Zaštita vertikalnog pristupa podrazumeva postojanje dve vrste korisnika aplikacije sa različitim ovlašćenjima: (1) krajnji korisnik ili drugi administrativni domen i (2) mrežni administrator. Zaštita horizontalnog pristupa podrazumeva pristup ovlašćenog korisnika odgovarajućim funkcijama.

VI ZAKLJUČNA RAZMATRANJA

U radu je predložena fazna metodologija testiranja softverskog proizvoda, koju čine: debagovanje, testiranje ispravnosti, performansi, pouzdanosti i bezbednosti. Debagovanjem se uklanjaju sintaksičke i logičke greške u programu. Testiranje ispravnosti obuhvata funkcionalnu proveru, kao i testiranje dizajna, odnosno analizu strukture i načina izvršavanja izvornog koda. Testiranjem performansi otkrivaju se problemi koji mogu da utiču na performanse softverskog proizvoda. Testiranjem pouzdanosti se utvrđuje verovatnoća funkcionisanja softvera bez otkaza, a testiranjem bezbednosti verifikuje se nivo zaštite pristupa softveru.

Primena opisane metodologije na testiranje aplikacije SLAM omogućila je sistematično otkrivanje i uklanjanje sintaksičkih, logičkih, funkcionalnih grešaka, kao i grešaka u dizajnu. Metode i rezultati testiranja su posebno značajni, s obzirom da se radi o prototipu softverskog proizvoda, koji će potencijalno biti prilagođavan potrebama različitih korisnika.

LITERATURA

[1] IEEE Std 610.12-1990: "IEEE Standard Glossary of Software Engineering Terminology", 1990.

[2] W. E. Perry, *Quality Assurance for Information Systems: Methods, Tools, and Techniques*, QED Information Sciences, Inc, Wellesley, MA 1991.

[3] IEEE Std 982.2-1988, "IEEE Guide for the Use of Standard Dictionary of Measures to Produce Reliable Software", 1988.

[4] P. Jiantao, "Software Reliability", Carnegie Mellon University, Spring 1999, [Online]. Available: http://www.ece.cmu.edu

[5] L. Rosenberg et al., "Software Metrics and Reliability", NASA Software Assurance Technology Center, [Online]. Available: http://satc.gsfc.nasa.gov

[6] P. Jiantao, "Software Testing", Carnegie Mellon University, Spring 1999, [Online]. Available: http://www.ece.cmu.edu

[7] S. Blake et al., "An Architecture for Differentiated Services", RFC 2475 (Informational), IETF, 1998.

[8] M. Stojanović, V. Aćimović-Raspopović, *Inženjering telekomunikacionog saobraćaja u multiservisnim IP mrežama*, naučna monografija, Saobraćajni fakultet Univerziteta u Beogradu, oktobar 2006.

[9] S. Boštjančič, M. Stojanović, V. Aćimović-Raspopović, "SLAM: an object-oriented application for IP Quality of Service negotiation", accepted for TELSIKS 2007, Niš, September 2007

[10] S. Boštjančič, "Ugovaranje kvaliteta servisa u IP mrežama sa diferenciranim servisima" , magistarska teza, prijavljena na Saobraćajnom fakultetu Univerziteta u Beogradu, 2007.

[11] ITU-T Recommendation M.3400, "TMN Management Functions", 2000.

ABSTRACT

In this paper, we first describe reliability metrics, which represent an important aspect of the overall software quality. Further, a five-phase methodology of software product testing is proposed. It enables systematic discovery and correction of software product failures. The methodology has been used for testing of prototype application for negotiating quality of service in IP-based networks.

METHODOLOGY OF SOFTWARE PRODUCT TESTING

Boštjančič, S., Stojanović, M., Nedeljković, R.