Mr. sc. Mato Tudor
Mr. sc. Dragan Martinović
Pomorski fakultet u Rijeci
Rijeka, Studentska 2

Pregledni članak
UDK: 004.415.53
Primljeno: 10. lipnja 2005.
Prihvaćeno: 28. lipnja 2005.

TESTIRANJE PROGRAMSKE PODRŠKE

U radu su prikazane tehnike testiranja programske podrške računala. Za razliku od sklopovske podrške, gdje je testiranje relativno jednostavno i kod kojeg se na temelju ulaznih vrijednosti vrednuju njima pripadajuće izlazne, testiranje programske podrške je složenije. Za razliku od grešaka nastalih pri pisanju (kodiranju) programa koje je jednostavno otkriti i ukloniti, pogreške nastale u fazi dizajna je najčešće nemoguće u potpunosti otkriti. Može se kazati da će pogreške u programskim rješenjima uvijek egzistirati. Testiranje predstavlja više od običnog traženja grešaka u programu. Može se provoditi s ciljem procjene kvalitete, povećanja sigurnosti, provjere i potvrde ispravnosti ili procjene pouzdanosti gotovih programskih rješenja. Pri tome se koriste razne tehnike kao što je npr.: pristup crnom kutijom ili pristup bijelom kutijom kod testiranja ispravnosti, testiranje izdržljivosti ili stresa kod testiranja pouzdanosti ili uporaba gotovih programa za testiranje performansi.

Ključne riječi: testiranje programske podrške, crna kutija, bijela kutija

1. UVOD

Razvojem mikroelektronike te informatičke i komunikacijske tehnologije, rješenja temeljena na računalima primjenjuju se u svim područjima ljudskog djelovanja. Izvršavanje i najbizarnijih poslova postalo je danas ovisno o ispravnosti rada računala. Računalo se sastoji iz sklopovske opreme, njegovog fizičkog dijela i programske podrške, koja predstavlja upute fizičkom dijelu kako da izvrši postavljeni zadatak. Za ispravan rad neophodno je da su ispravna oba, odnosno sklopovski i programski dio. Kako bi se postigla željena kvaliteta prije nego se programska rješenja puste u eksploataciju, vrše se zahtjevna testiranja s ciljem uklanjanja što je moguće više grešaka.

Testiranje programske podrške je proces traženja grešaka. U njemu su obuhvaćene sve aktivnosti potrebne za vrednovanje sposobnosti programa da izvrši pred njim postavljeni zadatak, kao i za tumačenje dobivenih rezultata. Programska podrška nije poput drugih fizičkih sustava koji za date ulazne veličine generiraju odgovarajuće izlazne veličine. Razlika je u načinu u kojem dolazi do pojave kvara. Kod većine fizičkih sustava do kvara dolazi zbog određenog i najčešće nerazboritog skupa stanja koja u sustavu mogu nastupiti uslijed vanjskog utjecaja. Do pojave grešaka u programskoj podršci dolazi zbog mnogo bizarnijih razloga. Detektirati sve različite modove kvara kod programske podrške je općenito neizvodljivo.

Za razliku od većine materijalnih (fizičkih) sustava kod kojih do kvara dolazi zbog grešaka nastalih u fazi proizvodnje ili fazi eksploatacije, većina neispravnosti koje se pojavljuju

tijekom izvođenja programa su posljedica grešaka nastalih u fazi dizajna. Programska podrška nije podložna koroziji ili trošenju uslijed korištenja. Njene karakteristike ostaju iste tijekom cijelog vijeka eksploatacije. Općenito ona se ne mijenja sve dok se ne ukaže potreba za njenom nadogradnjom (upgrade) zbog zastarjelosti koja se izvodi kako bi se programska podrška promijenila, poboljšala ili kako bi se nadopunile njezine mogućnosti. Ova nadgradnja prati promjene u poslovima koji se izvršavaju primjenom ogovarajućeg programskog rješenja.

Jednom kad je program u eksploataciji, greške nastale u dizajnu ostat će prikrivene sve dok se u nekom trenutku tijekom korištenja programa ne pojave.

Programske pogreške gotovo će uvijek egzistirati u bilo kojem programskom modulu umjerene veličine, ne zbog nepažljivosti ili neodgovornosti programera, nego zato što je programska podrška općenito veoma kompleksna. Za kompleksne sustave može se sa sigurnošću reći da greške nastale prilikom dizajna, najvjerojatnije neće nikada biti sve u potpunosti uklonjene.

Otkrivanje grešaka dizajna u programskoj podršci je iz istog razloga, njezine kompleksnosti, jednako teško. Kako programi ili bilo koji drugi digitalni sustavi nisu kontinuirani, testiranje graničnih vrijednosti nije dostatno da garantira ispravnost. Sve moguće vrijednosti moraju biti testirane i verificirane, ali cjelovito testiranje najčešće nije moguće izvršiti. Iscrpljujuće testiranje jednostavnog programa zbrajanja samo dva cijela 32-bitna broja, u slučaju testiranja svih mogućnosti koje mogu nastupiti (koje uključuje 264 različitih slučajeva testiranja), trajalo bi stotine godina i to u slučaju da se izvodi tisuću testiranja u sekundi. Očito, za stvarne programske module, kompleksnost je neusporedivo veća od prije navedenog primjera. Ako se još promatraju i ulazni podaci (iz stvarnog svijeta), problem će biti neusporedivo teži zato što nepredviđeni utjecaji iz okoline, kao i ljudska interakcija, moraju biti uključeni u testiranje.

Daljnje komplikacije unosi dinamička priroda programa. Ako se greška dogodi za vrijeme početnog testiranja i programski kod se mijenja, program možda neće raditi za slučajeve koji su ranije bili testirani. Da bi se izbjegle ove opasnosti testiranje se mora ponovno započeti ispočetka, što povećava troškove testiranja.

2. RAZLOZI TESTIRANJA

Bez obzira na ograničenja, testiranje je integralni dio razvoja programske podrške. Ono se primjenjuje u svim fazama razvoja programske podrške. Obično se više od 50% vremena potroši na testiranje. Testiranje programske podrške [1] najčešće se izvodi zbog:

- poboljšanja kvalitete,
- provjeravanja i potvrde ispravnosti,
- procjene pouzdanosti.

POBOLJŠANJE KVALITETE

Kako se danas računala implementiraju u rješavanje najsloženijih problema, pojava programske greške (engl. bug) može uzrokovati velike gubitke. (npr. uzrok havarija brodova, uzrok zrakoplovnih nesreća, blokiranje rada burzi,...). Greška u radu računala može biti uzrok velikih katastrofa. Poznata je panika nastala zbog prijelaza u novo tisućljeće, kada se zbog greške 2000 (Y2K) kod pisanja datuma i zapisivanja samo godina, ali ne i tisućljeća i

stoljeća, smatralo da će na cijelom svijetu računala pogrešno raditi. U današnjem svijetu, gdje su računala uključena u sve pore ljudskog djelovanja, može se kazati da je pitanje kvalitete sigurnosti programskih rješenja pitanje života i smrti.

Kvaliteta znači prilagođenost specifičnim zahtjevima dizajna. Minimalni zahtjevi za kvalitetom znače izvršavanje zahtijevane radnje pod specificiranim okolnostima. Uklanjanje grešaka tzv. debugiranjem, kao uobičajenim načinom testiranja programa, teško će pronaći sve neispravnosti nastale zbog pogrešnog dizajna programera. Nesavršenost ljudi upravo je razlog različitih neispravnosti koje se događaju kod prvog pisanja imalo složenijih programskih rješenja. Debugiranje koje se provodi korištenjem razvojnih alata na računalu u programskoj fazi razvoja programa otkrit će i ukloniti jedan dio tzv. logičkih grešaka, odnosno utvrdit će odgovara li programsko rješenje zahtjevima dizajna. Međutim, greške nastale pri dizajnu programskog rješenja, ovom tehnikom neće biti otkrivene.

PROVJERA I POTVRDA ISPRAVNOSTI

Provjeravanje i potvrđivanje ispravnosti je sljedeća značajna svrha testiranja. Testiranje se može promatrati kao matrica. Definira se tumačenja rezultata testiranja tako da svaki testirani dio rješenja treba raditi, ili ne raditi, pod zadanim poznatim okolnostima. Također, se može uspoređivati kvaliteta između različitih programskih proizvoda pod istim okolnostima baziranih na rezultatima istoga testa.

Testiranje kvalitete ne može se direktno provesti, ali se može testirati relevantne čimbenike koji govore o kvaliteti. Kvaliteta se može sagledavati kroz skupine čimbenika iz funkcionalnosti, tehnike pisanja i prilagodljivosti programske podrške. Ove tri skupine čimbenika mogu se promatrati kao jedna dimenzija prostora kvalitete programskih rješenja. Svaka dimenzija može se podijeliti u manje dijelove. Tako se uzastopce može vršiti podjela na sve niže razine. Tablica 1. pokazuje neke od najčešćih citiranih čimbenika kvalitete [2].

Dobro provedeno testiranje mora dati procjenu vrijednosti svih relevantnih čimbenika. Važnost svakog pojedinog čimbenika ovisi od aplikacije do aplikacije.

Programsku podršku treba dizajnirati tako da se jednostavno može testirati, a kasnije i održavati. Kako dobro testiranje zahtijeva veliki utrošak ljudskog rada i vremena, što rezultira velikom cijenom koštanja, dizajn prilagođen testiranju je veoma važno pravilo dizajniranja koje se postavlja kao cilj pri razvoju programske podrške.

Tablica 1. Osnovni čimbenici kvalitete programske podrške

Funkcionalnost (vanjska kvaliteta)	**Tehnika pisanja (unutarnja kvaliteta)**	**Prilagodljivost (buduća kvaliteta)**
Ispravnost	Efikasnost	Fleksibilnost
Pouzdanost	Mogućnost testiranja	Ponovna uporabnost
Uporabnost	Dokumentacija	Održavanje
Cjelovitost	Struktura	

Testovi koji se provode sa svrhom potvrde ispravnosti zovu se pozitivni (čisti) testovi. Nedostatak im je što se pomoću njih može potvrditi da program ispravno radi samo za specificirane slučajeve. Konačan broj testova ne može potvrditi da program radi u svim

slučajevima. U suprotnosti s tim samo jedan dobiveni kriv rezultat znači da program nije ispravan i da ne radi ispravno. Negativni (prljavi) testovi odnose se na ciljano testiranje dijela programa, kako bi se pokazalo da on ne radi ispravno.

PROCJENA POUZDANOSTI

Pouzdanost programske podrške ovisi o mnogim aspektima programa. Isti problem će od različitih ljudi biti riješen s programskom podrškom koja je različita po dizajnu, veličini, kompleksnosti itd. Sve to za posljedicu ima i različiti broj testova koji se moraju izvršiti. Na osnovu eksploatacijskog profila (procjene učestalosti korištenja različitih ulaznih podataka u programu), rezultati testiranja mogu poslužiti kao uzorci različitim statističkim metodama za procjenu pouzdanosti.

3. METODE TESTIRANJA

Mnogo je različitih metoda i tehnika testiranja koje se provode u testiranju kod svih životnih faza programske podrške. Mogu se podijeliti na različite načine. Klasificirano po namjeni (cilju) testiranje programske podrške može se podijeliti na:

- testiranje ispravnosti,
- testiranje performansi,
- testiranje pouzdanosti,
- testiranje sigurnosti

Klasificirano po životnim ciklusima programske podrške, testiranje se može podijeliti na metode i tehnike koje se primjenjuju u različitim fazama razvoja programske podrške, a to su:

- testiranje u fazi postavljanja zahtjeva koji se žele riješiti (engl. requirements phase),
- testiranje u fazi dizajna (engl. design phase),
- testiranje u fazi izrade programa (engl. program phase),
- testiranje koje se provodi s ciljem provjere ispravnosti dobivenih rezultata (engl. evaluation test results),
- testiranje u fazi instalacije (engl. installation phase),
- testiranje u fazi održavanja (engl. maintenance phase).

Prema području na koje se odnosi, testiranje se može podijeliti na:

- testiranje jednog osnovnog dijela (jedinice),
- testiranje neke komponente koja se sastoji iz manjih dijelova,
- testiranje sastavljenih komponenti u cjelini (integracije),
- testiranje cijelog sustava.

TESTIRANJE ISPRAVNOSTI

Ispravnost je osnovni i najmanji zahtjev što se postavlja pred programsku podršku te predstavlja bitnu svrhu testiranja. Testiranje u sebi mora sadržavati određenu količinu predviđanja, koje će odvojiti ispravno od pogrešnog ponašanja programske podrške. Kako izvođač testiranja može, ali i ne mora, poznavati unutarnje detalje programske podrške koja se testira (npr. tok naredbi. tok podatka, itd.), za testiranje se mogu uporabiti različiti pristupi. Najčešće se koristi pristup bijele kutije (engl. white-box point) ili pristup crne kutije (engl. black-box point). Treba naglasiti da uporaba ovakvih pristupa nije ograničena samo na testiranje ispravnosti.

TESTIRANJA PRISTUPOM CRNE KUTIJE

Pristup crne kutije [3] je metoda testiranja u kojoj se dolazi do testnih podataka na temelju specificiranih funkcijskih zahtjeva, ne uzimajući u obzir konačnu strukturu programa. Ovo testiranje naziva se još i testiranje upravljano podacima (engl. data driven), ulazno/izlazno testiranje (engl. input/output driven) ili testiranje zasnovano na zahtjevima (engl. requirements based). Kako se kod testiranja uzima u razmatranje samo funkcionalnost programskih modula, ovo testiranje se uglavnom odnosi na metode testiranja kod kojih je naglašeno izvršavanje funkcija, kao i objašnjavanje njihovih ulaznih i izlaznih podataka. Kod testiranja, program se tretira kao crna kutija nepoznata sadržaja kod koje su vidljivi samo ulazi i izlazi, a funkcionalnost je određena promatranjem dobivenih izlaznih podataka na temelju odgovarajućih poznatih ulaza. Prilikom testiranja, na osnovu različitih ulaznih podataka dobiveni izlazni podaci se uspoređuju s unaprijed očekivanima te se na taj način vrši vrednovanje ispravnosti programa. Svi testovi proizlaze iz specifikacije programa pri čemu se ne vrši nikakvo razmatranje programskog koda. Očigledno je da će se, čim se primjeni više mogućih ulaznih podataka, pronaći i više neispravnosti te će se njihovim otklanjanjem povećati i pouzdanost glede kvalitete programa. Detaljno testiranje kombinacija različitih ulaznih podataka za većinu programa je neizvodivo, čak i ako se promatra ispravnost samo ulaznih podataka kroz najosnovnije čimbenike, kao što su ispravnost unesenih vrijednosti, vrijeme unosa, redoslijed unosa, itd. Mnogo različitih kombinacija koje mogu nastupiti kod ulaznih podataka glavna je prepreka u funkcijskom testiranju. Dodatnu poteškoću unosi nekompletnost ili neispravnost datih specifikacija programa. Do pojave različitih dvosmislenosti dolazi zbog ograničenja jezika (najčešće prirodni jezik) koji se koristi za opis specifikacija. Iako se koristi određeni tip formalnog ili restriktivnog jezika, još se može pogriješiti i pri opisivanju svih mogućih slučajeva. Ponekad, uzrok teško rješivog problema je i sama specifikacija, jer korištenjem formalnog ili restriktivnog jezika s ograničenim skupom riječi najčešće nije moguće specificirati svaku situaciju koja se može dogoditi. Također, i ljudi pri opisivanju rijetko znaju jasno specificirati što točno žele, nego toga postanu svjesni kod već završenog specificiranja. Problemi zbog neodgovarajućeg specificiranja obuhvaćaju približno 30% svih bug-ova u programima [3].

Istraživanja koja se provode kod ove metode uglavnom su fokusirana na to kako postići maksimalnu efikasnost testiranja uz minimalne troškove, odnosno kako odrediti potreban broj testova. Nije moguće smanjiti broj ulaznih podataka, ali je moguće smanjiti broj testova koji se odnose na određeni broj različitih kombinacija ulaznih podataka koje mogu nastupiti.

Jedna od tehnika koja se koristi je podjela ulaznih podataka na skupine (domene). Ako se prostor ulaznih podataka podijeli po skupinama i uz pretpostavku da su sve ulazne vrijednosti u jednoj skupini ekvivalentne, dovoljno je izvršiti testiranje samo za jednu reprezentativnu vrijednost iz svake skupine da bi se zadovoljavajuće pokrilo čitav ulazni prostor. Posebno su zanimljive granične vrijednosti. Iskustva pokazuju da testovi koji otkrivaju granične uvjete imaju višu cijenu od onih koji to ne rade. Analiza graničnih vrijednosti [4] zahtijeva da se jedna ili više graničnih vrijednosti selektira kao slučaj reprezentativnog testa. Teškoće koje mogu nastupiti kod primjene ove tehnike su takve da neispravno definirane skupine u specifikaciji nije naknadno moguće efikasno otkriti. Da bi se primijenila ova metoda i izvršila dobra podjela na skupine, potrebno je dobro poznavanje strukture programske podrške. Iz tog razloga dobar plan testiranja neće sadržavati samo ovu , nego i metodu bijele kutije kao i kombinaciju obiju metoda.

TESTIRANJE PRISTUPOM BIJELE KUTIJE

Suprotno pristupu crne kutije, ovdje se programska podrška pri testiranju promatra kao bijela kutija, što znači da su prilikom testiranja poznati struktura i tijek izvođenja pojedinih programskih modula. Plan testiranja se radi u skladu s detaljima implementacije programskog rješenja kao što su programski jezik, logika programa, stil, itd. Pojedini testovi proizlaze iz same programske strukture. Ovaj pristup testiranju naziva se još pristupom staklene kutije (engl. glass-box), testiranje vođeno logikom (engl. logic driven testing) ili testiranje zasnovano na dizajnu (engl. design-based testing).

Razvijene su različite tehnike koje se koriste pri ovom testiranju jer se kompleksnost problema smanjuje s poznavanjem i pozornim promatranjem strukture programa prilikom testiranja.

Kod ove metode prisutna je namjera detaljnog promatranja nekih aspekata programske podrške kao što je: izvršavanja svake linije programskog koda najmanje jednom (engl. statement coverage), prolaženje svakog skoka (engl. branch coverage), ili razotkrivanje svih mogućih kombinacija ispravnih i neispravnih uvjeta (engl. multiple condition coverage) [5].

Testiranje kontrole toka naredbi, testiranje ponavljanja i testiranje toka podataka je testiranje kod kojeg se tok programske strukture promatra kao graf. Pojedini testovi se pažljivo selektiraju u ovisnosti od kriterija, da se svi čvorovi i veze između njih pokriju ili obuhvate najmanje jednom. Na taj se način može otkriti nepoželjni «mrtvi» programski kod, kod koji nije u uporabi ili se nikad neće izvršavati, ali koji se ne može otkriti funkcijskim testiranjem.

TESTIRANJE PERFORMANSI

Iako često kod specifikacije programske podrške nisu eksplicitno naznačene njezine zahtijevane performanse, one se mogu implicitno iščitati jer u eksploataciji programska podrška ima na raspolaganju određeno vrijeme za izvršavanje, odnosno definirane resurse koje može koristiti (procesorsko vrijeme, veličinu memorije, itd.). Iz tog razloga greška u performansama, uzrokovana najčešće lošim dizajnom, može prouzročiti neuporabljivost gotovog programskog rješenja. Vrednovanju performansi [6] mora se posvetiti posebna pažnja kod razvoja programske podrške. Vrednovanje performansi obično uključuje: uporabu resursa, vrijeme odziva, dužina redova koja ovisi o maksimalnom broju zadaća koje čekaju da budu izvršene,

itd. Tipični resursi koji se moraju razmatrati uključuju: zahtijevanu propusnost prijenosa podataka (npr. kod računalne mreži), brzinu rada procesora, veličinu prostora na jedinicama vanjske memorije (tvrdog diska), vrijeme pristupa podacima na tvrdom magnetskom disku, veličinu RAM memorije, itd. Cilj testiranja performansi može biti identifikacija uskog grla performansi, usporedba s performansama drugog rješenja ili samo određivanje performansi i njihovo vrednovanje (zadovoljavaju li, hoće li će implementirano rješenje raditi u željenim granicama, itd.). Za testiranje performansi koriste se tzv. benchmark programi.

TESTIRANJE POUZDANOSTI

Pouzdanost programske podrške odnosi se na vjerojatnost da neće doći do greške prilikom njezinog izvođenja. Pouzdanost se može odnositi na mnoge aspekte, uključujući i pouzdanost procesa testiranja. Direktno ocjenjivanje pouzdanosti programske podrške i njezino kvantificiranje je veoma teško [7]. Testiranje je provjerena metoda za procjenu pouzdanosti. Testiranjem se (obično pristupom crne kutije) mogu dobiti ulazni podaci koji vode k pojavi greške te se daljnjim analiziranjem tih podataka može procijeniti sadašnja pouzdanost, kao i predvidjeti buduća. Na temelju takve procjene, proizvođači programske podrške mogu odlučiti hoće li će proizvod isporučiti ili će ga još poboljšavati. Krajnji korisnik, na temelju informacija o pouzdanost programske podrške,može procijeniti rizik njene uporabe. Može se kazati da primarni cilj testiranja mora biti mjera za ovisnost testirane programske podrške. Značenje ovisnosti programske podrške može se definirati kao da se u slučaju greške ona neće ponašati s nekim neočekivanim ili katastrofalnim posljedicama. S gledišta zadovoljavanja ovog kriterija mogu se provoditi različite varijacije testiranja pouzdanosti, kao što je testiranje izdržljivosti (engl. robustness testing) ili testiranje stresa (engl. stress testing). Izdržljivost komponente programske podrške je stupanj do kojeg ona funkcionira ispravno u prisutnosti neočekivanih ulaza ili u nepogodnim uvjetima okoline. Testiranje izdržljivosti razlikuje se od testiranja ispravnosti u dijelu da se kod ovog testiranja ne promatra funkcijska ispravnost programske podrške. Samo se promatraju problemi njene izdržljivosti kod problema kao što su rušenje sustava, nenormalni završeci kraja rada, nemogućnosti završetka pojedinih podprocesa itd. Testiranje stresa se često koristi za testiranje cijelog sustava, a ne samo njegove programske podrške. U takvom testiranje se sustav promatra unutar, odnosno ispod specificiranih granica. Tipični stresovi predstavljaju prekoračenje resursa, prekomjerne aktivnosti itd.

TESTIRANJE SIGURNOSTI

Kvaliteta, pouzdanost i sigurnost programske podrške je usko povezana. Pukotine u programskim rješenjima mogu se iskoristiti da se otvore sigurnosne rupe. S razvojem Interneta, svjetske računalne mreže, problem sigurnosti programske podrške postaje sve izraženiji. Mnoge kritične programske aplikacije imaju integrirane sigurnosne mjere protiv tzv. malicioznih napada. Svrha testiranja sigurnosti je identifikacija i uklanjanje programskih pukotina koje potencijalno mogu voditi k povredi sigurnosti, kao i vrednovanje efektivnosti sigurnosnih mjera. Simulirani sigurnosni napad može se izvršiti da bi se pronašle slabosti u programskoj podršci.

4. KADA PRESTATI TESTIRATI

Testiranje se praktično može izvoditi beskrajno. Nikad se ne može sa sigurnošću kazati da su otkriveni i uklonjeni svi mogući defekti. Ali u jednom trenutku treba prestati testirati. Pitanje je kada to učiniti. Realno promatrano testiranje je trgovina između cijene koštanja, vremena i kvalitete. Stoga se, na nesreću, najčešće rabi pristup da se testiranje prekida kad god je jedan od resursa (vrijeme, novac ili broj testova) prekoračen. Bolji je pristup da se definira prekid testiranja, kada je pouzdanost u zahtijevanim okvirima ili kad su dobici od daljnjeg testiranja manji od cijene testiranja. Ovo će obično zahtijevati uporabu pouzdanih modela da bi se vrednovala predviđena pouzdanost testirane programske podrške. Svaki proces vrednovanja zahtijeva ponavljanje izvođenja sljedećih ciklusa: prikupljanja podataka koji dovode do greške, modeliranja i predviđanje. Ova metoda neće biti najbolja ukoliko je riječ o jako ovisnim sustavima i to zato, što je kod njih potrebno mnogo vremena i truda da bi se akumulirali stvarni podaci koji dovode do pojave greške.

5. ZAKLJUČAK

Testiranje programske podrške je svaka aktivnost usmjerena na procjenu osobina ili sposobnosti programa te određivanje onoga što vodi zahtijevanim rezultatima. Teškoće u razvoju programske podrške proizlaze iz složenosti programskih rješenja. Testiranje je više od običnog debugiranja. Cilj testiranja nije samo u provjeri, radi li program ispravno, nego i u procjeni kvalitete sigurnosti, procjeni pouzdanosti ili procjeni performansi gotovih programskih rješenja. Testiranje se izvodi u svim fazama životnog vijeka programa. Mogu se testirati gotova programska rješenja ili samo pojedini dijelovi.

Tehnike koje se najčešće koriste kod testiranje su testiranje crnom kutijom, testiranje bijelom kutijom, testiranje stresa, testiranje izdržljivosti, kao i razni programi za testiranje performansi. Kako se zbog složenosti gotovih programskih rješenja nikad sa sigurnošću ne može kazati da su otkrivene i otklonjene sve greške unutar programske podrške, a samo testiranje predstavlja trošak, pitanje je kad prestati s testiranjem. Zato se može zaključiti da je testiranje programske podrške trgovina između raspoloživih novčanih sredstava, vremena i željene kvalitete.

LITERATURA

[1] Beizer, B., *Software Testing Techniques.* Second edition. 1990.
[2] Hetzel, W. C., *The Complete Guide to Software Testing,* 2nd ed. Publication info: Wellesley, Mass.: QED Information Sciences, 1988.
[3] Beizer, B., *Black-box Testing: techniques for functional testing of software and systems.* Publication info: New York : Wiley, 1995.
[4] Myers, G. J., *The art of software testing,* Publication info: New York: Wiley, 1979.

[5] Parrington, N.; Roper, M., *Understanding Software Testing*, Published by John Willey & Sons, 1989.
[6] Vokolos, F. I.; Weyuker, E. J., *Performance testing of software systems*, Proceedings of the first international workshop on Software and performance, 1998, pp. 80 – 87.
[7] Yang, M.C.K.; Chao, A., *Reliability-estimation and stopping-rules for software testing, based on repeated appearances of bugs*, IEEE Transactions on Reliability, vol.44, no.2, pp. 315-21, 1995.

Summary

PROGRAM SUPPORT TESTING

The paper describes testing techniques of computer program supports. Differently to the circuit support, where testing is relatively simple, as input values are employed to assess their appurtenant output value, program support testing is more complex. In contrast to errors originated when encoding a program, whose errors can be easily detected and removed, errors occurring at the designing stage are most often impossible to discover. Moreover, it can be asserted with a degree of certainty that error-free program solutions have not yet been achieved.

Testing means something more than plain detection of program errors. It may be put into effect with the aim of assessing quality, increasing fail-safe state, checking and confirming error-free condition, or assessing reliability of ready program solutions. Various techniques may be employed for the purpose such as, for example, the black box access or the white box access when testing error-free condition, endurance and stress testing in reliability testing or the usage of ready programs for performance testing.

Key words: program support testing, black box, white box

Faculty of Maritime Studies Rijeka
Studentska 2, 51000 Rijeka
Croatia