PREDNOSTI IN SLABOSTI BETA TESTIRANJA

Tomaž Dogša
cV&Vs Center za verifikacijo in validacijo sistemov, Fakulteta za elektrotehniko, računalništvo in informatiko
Univerza v Mariboru, Smetanova 17, 2000 Maribor
tdogsa@uni-mb.si

Povzetek

V prispevku je opisana problematika testiranja programske opreme, ki je namenjena večjemu krogu neznanih uporabnikov. Najprej je napravljena analiza testiranja korektnosti, uporabljivosti in beta testiranja. Iz analize sledi, da beta testiranje rešuje mnoge slabosti, s katerimi se srečamo pri preverjanju korektnosti oziroma uporabljivosti. Napravljena je tudi primerjava med klasičnim in beta testiranjem, ki pomaga pri objektivni odločitvi o vključitvi beta testiranja v načrt preverjanja.

Abstract

In this paper we discuss testing problems of software intended for many unknown users. Firstly we analyse correctness, usability testing and beta testing. The results show that beta testing is a very promising method. Finally, a comparison between beta and traditional testing method is made. The results of comparison support the objectiveness of decision about including beta testing in the V&V (verification and validation) plan.

1. Uvod

Kompleksnost današnjih programskih produktov skokovito narašča. Microsoftov Winword 2.0 je imel približno 200 funkcij - naslednja verzija (Winword 6.0) pa že približno 350 funkcij. Hkrati se širi tudi kompleksnost grafičnega uporabniškega vmesnika. V začetnem obdobju računalnikov je bila programska oprema pisana za majhen krog strokovnjakov - torej za znanega naročnika. Danes je na tržišču veliko število produktov namenjenih neznanim uporabnikom, ki v splošnem niso strokovnjaki na področju informatike oziroma računalništva. V tem prispevku bi radi osvetlili enega izmed novejših pristopov k preverjanju tovrstne programske opreme - beta testiranje.

Najprej bomo na kratko opisali značilnosti preverjanja korektnosti in uporabljivosti (več o tem je v [NIELSEN,1992], [DOGŠA,1995]) in nato opisali beta testiranje. Ta metoda je v bistvu neke vrste hibridna metoda, ki združuje nekatere posebnosti preverjanja uporabljivosti in korektnosti. Na koncu bomo s primerjavo med metodami za preverjanje korektnosti in beta testiranjem skušali ugotoviti prednosti in slabosti obeh pristopov. Rezultati primerjave bodo pomagali pri naslednjih odločitvah o vključitvi beta testiranja v preverjanje.

2. Preverjanje korektnosti

Vsako izmed karakteristik, s katerimi opisujemo kakovost programske opreme, preverjamo z ustrezno tehnologijo preverjanja. Kljub raznim posebnostim, imajo vse vrste preverjanja določene podobnosti, ki jih lahko modeliramo s poenostavljenim splošnim preverjevalnim modelom (slika 1). Ta model vsebuje le najbistvenejše elemente tega procesa. Podrobnejši model je opisan v [DOGŠA,1994].

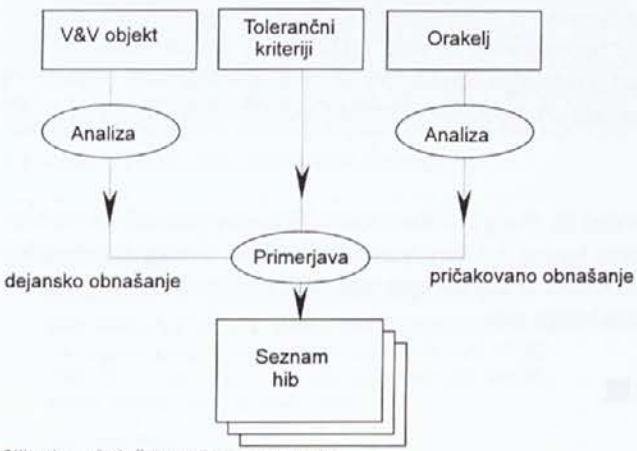
Vsako preverjanje je sestavljeno iz naslednjih aktivnosti:

- analize predmeta, ki ga preverjamo
- analize oraklja in
- primerjave pričakovanih lastnosti z dejanskimi.

Orakelj je neka referenca, v katero imamo zaupanje, oziroma za katero velja majhna verjetnost, da ne bi bila pravilna. Najbolj pogosti oraklji so specifikacije ali pa uporabnikov priročnik. Najbolj razširjena metoda za analizo predmeta, ki ga preverjamo, je testiranje. Z analizo oraklja določimo pričakovano obnašanje, z analizo predmeta pa dejansko obnašanje. Glede na tolerančne kriterije se s primerjavo nato odločimo o nastopu hibe.

Ker bomo kasneje napravili primerjavo med preverjanjem korektnosti in uporabljivosti, moramo na tem mestu podrobneje opredeliti pojem korektnosti. Večina zahtev oziroma sistemskih specifikacij se nanaša na funkcijo produkta. V formalnem smislu bi rekli, da gre za definiranje preslikave vhodnih podatkov v izhodne. Korektnost je lastnost, ki pove v kolikšni meri je ta

preslikava pravilno izvedena [IEEE,1990b]. Npr. uporabnikova zahteva je bila, da naj program sestavi račun za električno energijo. To je ena izmed funkcij, ki jo mora program pravilno izvesti, v kolikor ne vsebuje nobenih napak. Pri izpisu računa, ki je neke vrste izhodni podatek, se pri preverjanju korektnosti vprašamo: ali so vse postavke pravilne? Ne sprašujemo pa po drugih neustreznostih, kot npr. ali je uporabnik programa razumel, kaj piše na računu, ali je znal program brez težav pognati ipd.



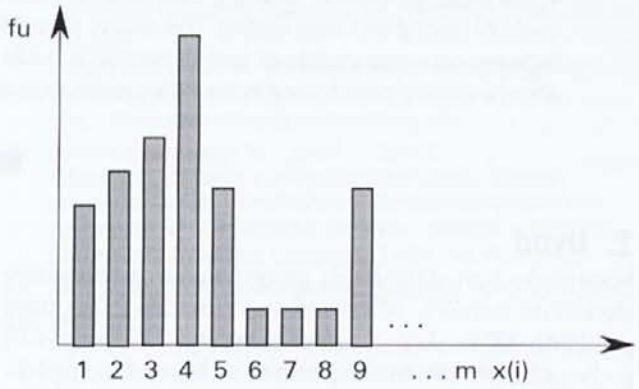
Slika 1: **Splošni model preverjanja**

Najstarejša in tudi najbolj razširjena metoda za preverjanje korektnosti je funkcijsko testiranje. V nadaljevanju se bomo pri obravnavi omejili zgolj na to metodo. Pri testiranju je večinoma orakelj dobro definiran in odločitev o nastopu hibe oziroma odpovedi ni težka, saj je možno v numerično orientiranih programih to primerjavo celo avtomatizirati. Problemi, s katerimi se soočajo preverjevalci korektnosti, so naslednji:

1. Kompleksnost funkcij, ki jih opravlja softver, skokovito narašča.
2. Vseh napak ni možno odpraviti .
3. Stroški preverjanja znašajo od 20 % do 50 % vseh stroškov.
4. Skupina, ki opravlja preverjanje, je v primerjavi z razvojno izrazito majhna.
5. Avtomatska testirna orodja so na zelo nizkem nivoju, ali pa jih sploh ni.
6. Profil uporabe programa je zelo redko poznan.

Prvi problem je posledica hitrega razvoja strojne opreme in zahtev uporabnikov. Da je nemogoče odpraviti vse napake, so ugotovili že zelo zgodaj (glej npr. [MYERS,1979]). Razen za trivialne programe, je nemogoče totalno preveriti produkt, saj je vhodna domena prevelika. Ker je tudi cena produkta navzgor omejena, je to drugi vzrok, zaradi česa je nemogoče odkriti in odpraviti prav vseh napak. Ker je skupina preverjevalcev relativno majhna in ker preverjanja z orodji ne moremo popolnoma avtomatizirati, moramo ustrezno prilagoditi terminalno kriterijsko funkcijo[1] ter testirno strategijo - če že ne moremo odkriti vseh hib, potem odkrijmo vsaj tiste, ki se bodo najpogosteje pojavljale. Pri odpravljanju hib oziroma napak velja podobno. Odpravimo samo najresnejše oziroma tiste, ki bodo pri uporabniku povzročile največ neprijetnosti. Torej ključno merilo za vodenje testiranja in odpravljanja napak je pogostost in resnost hib.

Pri tem pristopu se preskuševalci srečajo s problemom določevanja resnosti hib in operativnega profila uporabe (glej zgled na sliki 2). Kljub temu, da je ta podatek ključnega pomena za prej omenjeni pristop[2], ga imamo zelo redko na razpolago, saj so stroški, ki so potrebni za določitev operativnega profila, zelo visoki.



Slika 2: **Operativni profil uporabe nekega programa. *fu* je frekvenca uporabljanja, x(i) je indeks funkcije.**

Zgornji problem je možno relativno enostavno rešiti s tem, da v testirni proces vključimo tudi resnične uporabnike in ne samo specialiste. Ta pristop se uporablja pri testiranju uporabljivosti, kar bomo v nadaljevanju na kratko opisali.

3. Preverjanje uporabljivosti

Ker se krog uporabnikov programske opreme skokovito širi, postaja uporabljivost (usability) ena izmed čedalje pomembnejših karakteristik programske opreme. Vedno več je konkurenčnih produktov, ki izvajajo iste funkcije, vendar na različen način. Poleg cene se kupec odloča tudi glede enostavnosti uporabljanja, prijaznosti ipd. V bistvu gre za enega izmed atributov

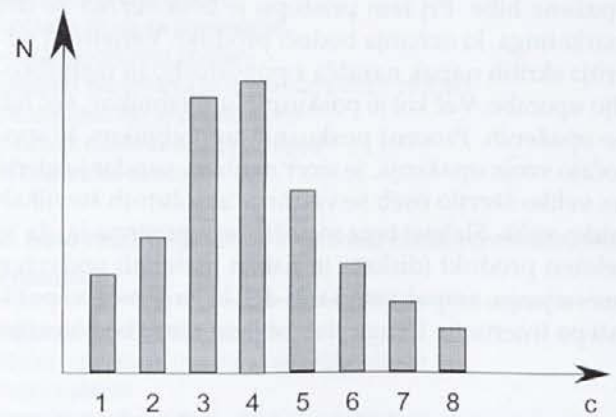
1 *S to funkcijo določimo, kdaj bomo končali s testiranjem.*
2 *To še posebej velja za določevanje zanesljivosti programske opreme.*

kakovosti, ki so ga poimenovali uporabljivost. V IEEE standardu [IEEE,1990b] je uporabljivost definirana kot enostavnost učenja, uporabe, priprave vhodnih podatkov in interpretiranja rezultatov. Večina zahtev oziroma sistemskih specifikacij se v splošnem nanaša na korektnost, manjši del pa na uporabljivost. Podrobnejši opisi preverjanja uporabljivosti se nahajajo v [PREECE, 1994], [ERLICH,1994], [NIELSEN,1992], tukaj povzemamo le osnovne lastnosti.

Preverjanje uporabljivosti poteka v dveh fazah:
- predhodno vrednotenje (formative evaluation) - znani so osnutki grafičnega uporabniškega vmesnika,
- končno preverjanje - programski produkt je skoraj popolnoma ali pa v celoti končan.

Če želimo preverjati uporabljivost, morajo biti izpolnjeni naslednji pogoji:
- znan mora biti profil uporabnikov,
- specifikacije, ki se nanašajo na uporabljivost, morajo biti podane na kvantitativen način,
- na razpolago moramo imeti uporabnike.

Največji problem, ki ga srečamo pri preverjanju uporabljivosti, je iskanje dejanskih uporabnikov. Zelo težko dobimo osebe, ki bi bile pripravljene sodelovati pri preverjanju. Pogosto se napačno predpostavlja, da so razvijalci dovolj dober približek za tipičnega uporabnika. Le s skrbno študijo ali pa z izkušnjami lahko določimo profil populacije uporabnikov. Mnogi proizvajalci programske opreme si pomagajo tako, da pri registraciji sprašujejo po določenih podatkih, ki jim pomagajo pri določevanju profila uporabnikov. Pogosto uporabnike razvrščajo glede na izkušnje na področju informatike oziroma računalništva (glej sliko 3). Ker v večini primerov v proces preverjanja ni možno vključiti celotne populacije, je potrebno izbrati reprezentativen vzorec. Kot so potrdile tudi lastne raziskave [DOGŠA,1995],



Slika 3: **Hipotetičen profil uporabnikov glede na izkušnje. N je število uporabnikov; c je število let.**

lahko izpeljemo preverjanje uporabljivosti z relativno majhnim številom tipičnih uporabnikov, če sodelujejo v kontroliranem okolju preverjanja. Testiranje z dejanskimi uporabniki je v ZDA zelo pogosto, saj Nielsen navaja [NIELSEN,1992], da to počenja kar 69% softverskih podjetij.

Glede na definicijo uporabljivosti, je preverjanje seveda osredotočeno predvsem na uporabniški vidik - tipična vprašanja, na katera želimo dobiti odgovor, so:
- ali uporabniki v resnici uporabljajo produkt, tako kot smo si zamislili,
- kaj ga moti pri uporabi,
- ali so dosežene kvantitativno določene zahteve (npr. tipični uporabnik naj v 5 min. inštalira produkt)?

Tem odgovorom ustreza tudi strategija preverjanja. Namen preverjanja je odkrivanje neustreznosti predvsem v uporabnikovem vmesniku ter v drugih delih programa, ki so povezani z uporabljivostjo - torej ni poudarka na korektnosti. Zato tudi preverjanje uporabljivosti ne sme ostati edina aktivnost procesa.

Pri preverjanju uporabljivosti izstopata dva problema:
- zahteve glede uporabljivosti večinoma sploh niso definirane,
- zelo težko je pridobiti tipične uporabnike.

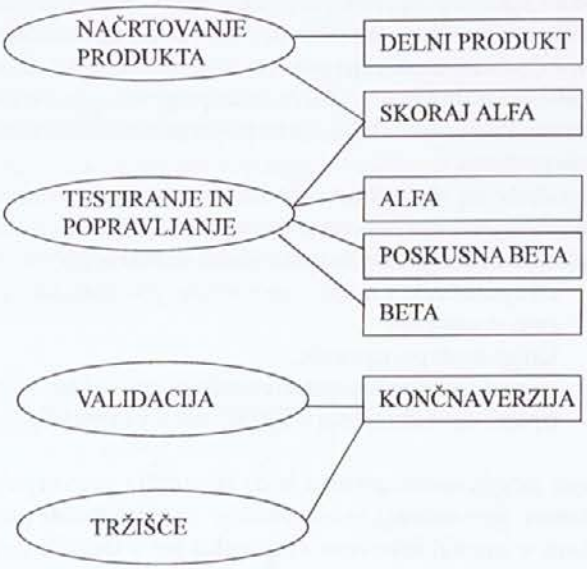
Zelo redko obstajajo **konkretne zahteve** glede uporabljivosti. Če pa že obstajajo, so večinoma opisne: npr. grafični urejevalnik naj bo razumljiv. Ker niso podane na kvantitativen način, jih ni možno uporabiti za orakelj. Pogosto se za orakelj uporablja kar konkurenčni produkt. V tem primeru bi se npr. prej omenjena zahteva glasila: novinec mora narisati določen objekt vsaj tako hitro kot s konkurenčnim produktom. Pri tem pristopu se vloženi napor v preverjanje skoraj podvoji, saj je potrebno najprej analizirati konkurenčni produkt[3], če hočemo dobiti potrebne referenčne vrednosti.

4. Beta testiranje

Beta testiranje rešuje mnogo problemov, ki smo jih srečali pri predhodni obravnavi. Dokaj nenavadno ime je dobilo po beta verziji programske opreme, ki je v bistvu skoraj izgotovljen končni produkt (glej sliko 4).

Osnovna ideja beta testiranja je, da so v testiranje vključeni dejanski uporabniki, ki program preskušajo v svojem okolju (več o tem glej v [DOGŠA,1996]). Te uporabnike imenujemo beta preskuševalci oziroma, če gre za organizacijo, beta testirno mesto (beta site). Po določenem času pošljejo beta preskuševalci k razvojni

[3] *V bistvu gre za analizo oraklja (glej splošni model preverjanja na sliki 1.)*



Slika 4: **Del poenostavljenega življenskega ciklusa programske opreme [KANER,1993]**

firmi poročila, v katerih je seznam opaženih hib, mnenje in morebitne sugestije. Razvijalci poročila skrbno analizirajo in glede na statistično verjetnost pojavljanja hib in glede na resnost se odločijo o odpravljanju napak. Nato sledi zaključno preverjanje v smislu validacije oziroma sprejemnega testiranja.

Beta testiranje, ki je zelo dober nadomestek za testiranje uporabljivosti, oziroma za testiranje grafičnega uporabniškega vmesnika, je torej v bistvu hibrid med testiranjem korektnosti in uporabljivosti.

Slika 5: **Beta produkti so navzven razpoznavni.**

4.1. Nevarnosti beta testiranja

Beta preskuševalci so opozorjeni na nekatere znane hibe[4], ki pa niso kritične, razen v določenih razmerah. Firma, ki preda program beta preskuševalcem, želi, da ga le-ti uporabljajo v polnem obsegu čim dlje. Testiranje lahko traja nekaj ur ali pa cel mesec. Čas je odvisen od kompleksnosti programa. Poseben problem je varnost (safety) beta programske opreme. Ker beta verzija skoraj zagotovo vsebuje še napake, se še ne ve natančno, kakšne hibe bodo morebiti nastopile. Bo potrebno ponovno inštalirati operacijski sistem ali pa formatirati disk? Ker vsak beta preskuševalec ne razpolaga s sistemi za arhiviranje, pomeni, da so beta preskuševalci izpostavljeni določenemu tveganju.

4.2. Iskanje beta preskuševalcev

Pri iskanju ustreznih preskuševalcev uporabljamo dva pristopa: testiranje z znanimi in testiranje z neznanimi preskuševalci. Pri prvem imamo majhno število znanih preskuševalcev, pri drugem pa masovno število neznanih. Izbor pristopa je odvisen od zahtevnosti programske opreme. Programsko opremo, ki zahteva posebno znanje (npr. CAD/CAE softver), lahko uporabljajo le dovolj strokovno usposobljeni preskuševalci. Ker je tudi relativno draga (od nekaj tisoč dolarjev navzgor), uporabimo raje znane preskuševalce. Običajno jih nagradimo s končno verzijo po zelo znižani ceni, kar je tudi eden izmed njihovih motivov za sodelovanje.

Uspešnost beta testiranja v začetku strmo narašča s številom beta preskuševalcev, nato pa se počasi ustali. Ker je beta testiranje povezano z veliko dela in pisanjem končnega poročila, morajo preskuševalci imeti ustrezno močan motiv.

Kadar potrebujemo zelo veliko število preskuševalcev, lahko uporabimo beta testiranje z *neznanimi preskuševalci*. Najprej začnemo namerno razširjati svojo beta verzijo, kar je s pomočjo Interneta relativno enostavno. Pošljemo jo na razne računalniške arhive, od koder je s ftp protokolom vsakemu dostopna. Zraven priložimo še prošnjo, v kateri prosimo za mnenje, sugestije in opažene hibe. Pri tem pristopu je beta verzija že del marketinga, ki oznanja bodoči produkt. Verjetnost odkritja skritih napak narašča s pogostostjo in raznolikostjo uporabe. Več kot je poskusnih uporabnikov, več hib bo opaženih. Procent poskusnih uporabnikov, ki sporočajo svoja opažanja, je sicer majhen, vendar je glede na veliko število oseb še vedno v absolutnih številkah lahko velik. Slabost tega množičnega pristopa je, da ni celoten produkt (diskete in tiskan material) podvržen preverjanju, ampak samo tisti del, ki ga je možno pošiljati po Internetu. Druga slabost je ta, da so beta verzije,

[4] *Ker osnovna terminologija na področju testiranja še ni popolnoma izkristalizirana, se v prispevku naslanjamo na [DOGŠA,1993].*

če so dobre in če uporabnik ni prezahteven, uporabne. Z drugimi besedami to pomeni, da na ta način zgubljamo potencialne kupce. Vendar ne smemo pozabiti, da je mnogokrat ravno ta uporabnost osnovni motiv beta preskuševalcev. Število izgubljenih kupcev delno kompenzira tudi prej omenjen propagandni učinek. Omeniti moramo še znižanje stroškov testiranja, saj so ga prostovoljci opravili brezplačno.

Ker je običajno testiranje, ki je v splošnem mešanica testiranja korektnosti in uporabljivosti, relativno drago, se k neznanim beta preskuševalcem zatekajo nekatere zelo majhne firme in tiste, katerih produkt se bo masovno prodajal po relativno nizki ceni (npr. Microsoftov Windows 95). Masovno tržišče tudi pomeni, da mora program zanesljivo delovati na raznih računalniških konfiguracijah.

Ne glede na pristop veljajo neke splošne zahteve glede preskuševalcev. Npr. preskuševalci morajo biti vredni zaupanja, da ne bodo programa razširjali. Kljub pisnim obljubam s strani preskuševalcev, firme pogosto raje pošiljajo personalizirane verzije, ki imajo vpisane identifikacijske podatke preskuševalca, kar je vidno ob vsakem zagonu. Ker je možno ta problem relativno enostavno rešiti, zakodirajo identifikacijske podatke tudi v samo izvršilno kodo po posebnem ključu. V primeru, da se firma kasneje dokoplje do piratske beta verzije, lahko s posebnim programom preskuševalčeve podatke izpiše in najde krivca.

5. Primerjava beta testiranja s testiranjem korektnosti

Namen vsakega testiranja je iskanje prisotnosti napak, ki se kaže v eni ali več hibah. Temu skupnemu namenu ustrezata oba pristopa. Testiranje korektnosti se izvaja znotraj firme ali pa ga prevzame posebna neodvisna testirna organizacija, ki zagotavlja večjo objektivnost. Testiranje korektnosti poteka po določenem planu na sistematičen način. Ker so testni vzorci shranjeni, nimamo problemov s ponovljivostjo hib. Če ponovljivost ni zagotovljena, je zelo težko poiskati vzroke za nastanek določene hibe. Pri beta testiranju ni tako natančnega nadzora nad procesom testiranja, saj beta preskuševalec ne beleži vseh svojih akcij in vhodnih podatkov. To se pozna pri večjih naporih, ki jih moramo vložiti v iskanje napak.

Preskuševalci znotraj firme imajo na razpolago posebna testirna orodja, ki lahko pospešijo testiranje. Ker so že več čas seznanjeni s produktom, oni niso več tipični uporabniki. Oddaljenost skupine, ki je zadolžena za testiranje, veča objektivnost. Najbolje pa ta problem reši neodvisna testirna organizacija.

Ker nimamo nadzora nad beta preskuševalci, lahko zato podvomimo o doslednosti njihovega preverjanja. Nikakor ne vemo, če so v resnici uporabljali program tako dolgo in intenzivno, kot je bilo dogovorjeno. Beta preskuševalec sicer lahko dobi testirni načrt, za katerega pa ne vemo, če se ga bo držal dosledno. Zanesljivost njihovih poročil je zaradi tega manjša kot pri običajnem testiranju.

Pri testiranju ima zelo pomembno mesto sposobnost prognoziranja stroškov, ozirom resursov ter rokov. Ker še ni nobenih praktično uporabnih stroškovnih modelov, ki bi bili sposobni ocenjevanja na podlagi kompleksnosti testirnega objekta, se največkrat uporabljajo izkustvene metode. Te pa temeljijo ne preteklih izkušnjah oziroma

1. preverjanje korektnosti z notranjimi preskuševalci

Prednosti:	**Slabosti**:
hitra komunikacija	majhna oddaljenost od razvoja
izkušnje s preverjanjem	niso tipični uporabniki
uporaba orodij za preverjanje	prevelik obseg dela
velika zanesljivost opažanj	veliki stroški
predmet preverjanja je lahko kompleten produkt	
ponovljivost hib je zagotovljena	
možnost sistematičnega pristopa	
ni nevarnosti piratstva	

2. beta testiranje (znani in neznani zunanji preskuševalci):

Prednosti:	**Slabosti**:
velika oddaljenost od razvoja	nezanesljiva poročila
tipičen uporabnik (možnost selekcioniranja)	preskuševalci nimajo orodij za preverjanje
majhni stroški	počasna komunikacija
propagiranje produkta	ponovljivost hib ni zagotovljena
ni potrebno poznati profil uporabe	ni sistematičnega pristopa
	nevarnost piratstva

podatkih. Pri beta testiranju v splošnem izgubimo podatke o vloženem trudu.

Razen kadar gre za znane preskuševalce, smo z beta testiranjem omejeni samo na testirne objekte, ki jih je možno razširjati po Internetu.

6. Sklep

Če razvijamo programsko opremo, ki je namenjena večjemu številu neznanih uporabnikov, in imamo dostop do Interneta, je beta testiranje zelo primerna metoda. Kljub svoji atraktivnosti ne sme ostati **edina** aktivnost v procesu, ampak naj bo le ena izmed dodatnih, ki naj zagotovijo večjo kakovost produktov. Posebno pozornost moramo posvetiti kvaliteti in kvantiteti beta preskuševalcev. Rezultati primerjave med običajnim in beta testiranjem naj pomagajo pri odločitvi, ali bomo uporabili beta testiranje, ali ne. O uspešnosti žal v dostopnih virih ne poročajo, saj je izvedba takšne komparativne študije dokaj zahtevna.

7. Literatura

[BEIZER,1990] B.Beizer:
"Software Testing Techniques",*Van Nostrand Reinhold, New York,1990, 2. izdaja.*

[DOGŠA,1994] T.Dogša:
"Verifikacija in validacija programske opreme",*Tehniška fakulteta, Maribor, 1993.*

[DOGŠA,1995] T. Dogša,
"Usability verification methods", *ICSQ95, Fakulteta za elektrotehniko, računalništvo in informatiko, Maribor, 1995, str. 65-69.*

[DOGŠA,1996] T. Dogša,
"Beta testing", *zbornik: Simpozij ERK 96, Portorož sept. 1996, Fakulteta za elektrotehniko, Fakulteta za računalništvo in informatiko, Ljubljana.*

[ERLICH,1994] K.Erlich, M.B. Butler, K.Pernice:
"Getting The Whole Team Into Usability Testing", *IEEE Software, januar 1994, str.89-91.*

[IEEE,1990b]
"IEEE Standard Glossary of Software Engineering Terminology", *IEEE Std. 610.12-1990, Revision and redesignation of IEEE Std. 792-1983, The Institute of Electrical and Electronics Engineers, USA, 1990.*

[KANER,1993] Cem Kaner, Jack Falk, Hung Quoc Nguyen:
"Testing Computer Software", *Van Nostrand Reinhold, 1993.*

[MYERS ,1979] J. G. Myers:
"The Art of Software Testing", *John Wiley and Sons, Inc., New York, 1979.*

[NIELSEN,1992] J. Nielsen:
"The Usability Engineering Life Cycle", *COMPUTER, marec 1992, str. 12-22.*

[NIELSEN,1996] J. Nielsen:
"Putting the User in User-Interface Testing", *IEEE Software, maj 1996, str. 89- 90.*

[PREECE,1994] J.Preece et al.:
"Human-Computer Interaction",*Addison-Wesley Publishing Co., 1994.*

◆

Avtor je docent na Fakulteti za elektrotehniko, računalništvo in informatiko, kjer predava na dodiplomski in podiplomski stopnji (Verifikacija in validacija programske opreme). Vodi tudi cV&Vs (Center za verifikacijo in validacijo sistemov). Na raziskovalnem področju se ukvarja predvsem s tehnologijo preverjanja oziroma testirnimi orodji.

◆