

TESTIRANJE SOFTVERA KAO MERA ZAŠTITE KORISNIKA SOFTVERA

Ljubomir Lazić, Državni Univerzitet u Novom Pazaru,
llazic@np.ac.rs, www.np.ac.rs

Sažetak: Testiranje softvera je već više godina, pa i decenija, ustanovljeno kao naučna, odnosno inženjerska disciplina. Testiranje softvera se sastoji od aktivnosti dinamičke verifikacije ponašanja programa na bazi konačnog skupa testova, odabranih na pogodan način iz beskonačnog skupa mogućih načina izvršavanja programa, a prema specificiranom očekivanom ponašanju softvera u razvijanoj aplikaciji tj. zahtevanog kvaliteta softvera. Zbog toga u ovom radu se daje opis rezultata višegodišnjeg istraživanja integralnog i optimiziranog procesa testiranja (TS) softvera, ukazano je na prednosti i nedostatke primene raznih tehnika i strategija TS, predloženi su postupci za donošenje odluke o odabiru tehnika testiranja u opštijem slučaju da se spreče defekti u softveru u toku eksploatacije od strane korisnika, čime se izbegavaju sudski procesi usled nedovoljnog kvaliteta softvera.

Ključne reči: *Testiranje softvera, prevencija grešaka, optimizacija, rizici, zaštita korisnika*

1. PROCES TESTIRANJA

Sa rastom kompleksnosti realizovanih funkcija i primena, posebno je narastao zahtev za kvalitetom softvera u pogledu: pouzdanosti (kod softvera sa kritičnom misijom), pogodnosti za testiranje i održavanje, ponovne upotrebljivosti, otpornosti na greške i drugih faktora kvaliteta softvera. U razvojnom ciklusu softvera sve je značajniji zadatak Proces Testiranja Softvera (PTS) ili Verifikacije i Validacije (V&V) koji treba da obezbedi zahtevani nivo poverenja u ispravnost (korektnost) softvera kao i obezbeđenja ostalih zahtevanih karakteristika softvera. Testiranje softvera, kao proces, ima tehničke, ali i ekonomske aspekte. Ekonomski aspekti su povezani sa činjenicom da su resursi i vreme na raspolaganju tima za testiranje ograničeni. Naime, iscrpno (potpuno) testiranje u mnogim slučajevima nije praktično izvodljivo zbog ekonomskih ograničenja. Organizacija koja se bavi razvojem softvera mora završiti projekat na vreme, u okvirima budžeta, i zadovoljiti zahteve klijenata [1,2,6]¹.

Tržište zahteva od kompanija koje razvijaju softver sve kraće vreme razvoja softvera koje zbog toga vode bespoštednu borbu u izvršavanju poznate mantre: brže, bolje, jeftinije. Sa rastom kompleksnosti realizovanih funkcija i primena posebno je narastao zahtev za kvalitetom softvera u pogledu pouzdanosti (kod softvera sa kritičnom misijom), pogodnosti za testiranje i održavanje, ponovne upotrebljivosti, otpornosti na greške i

¹ Ovaj rad delimično je finansiralo Ministarstvo za Nauku i Tehnološki razvoj Republike Srbije u okviru projekta tehnološkog razvoja: "Integralni i optimizirani proces testiranja i održavanja softvera", TR 13018.

drugih faktora kvaliteta softvera. Nedostatak iskusnog i utreniranog razvojnog tima, ograničeni resursi i sredstva za automatizovan razvoj softvera doveo je do niza sličnih problema kako u velikim tako i u malim kompanijama koje razvijaju softverski proizvod. Ovi problemi uključuju problem nekompletnog razvoja, neefikasnog testiranja softvera, nizak nivo kvaliteta softvera, visoki troškovi razvoja i održavanja softvera, nezadovoljan kupac, ali se time lista problema ne završava.

U naporima da spreče defekte u softveru koji se predaje korisniku na upotrebu, tj. da 'uteknu' do korisnika, kompanije sve više sredstava investiraju u testiranje softvera. U razvojnem ciklusu softvera sve je značajniji zadatak TS ili V&V koji treba da obezbedi zahtevani nivo poverenja u ispravnost (korektnost) softvera kao i obezbeđenja ostalih zahtevanih karakteristika softvera. Testiranje Softvera je skup proces, jer u proseku oko 50% ukupnog budžeta za razvoj softverskog proizvoda se troši na testiranje softvera dok je u nekim oblastima primene čak i preko 80% .

Veliki su gubici firmi usled grešaka u softveru **koji su usloveli razvoj oblasti krivične odgovornosti proizvođača softvera i zaštite kupaca zbog neadekvatnog kvaliteta softverskog proizvoda** [1].

Do danas nije razvijena metodologija testiranja softvera, kao najvažnija aktivnost u procesu obezbeđenja kvaliteta softvera, koja garantuje da će program biti bez grešaka, ali sistematiziran i planirani postupak testiranja softvera, kroz izvršavanje programa sa pažljivo odabranim skupom ulaznih podataka, značajno povećava poverenje u korektnost programa. Testiranje softvera odavno nije samo faza u procesu razvoja softvera već paralelni pod-proces. Pošto je stalan pritisak, u raznim poslovnim primenama, za upotrebom novih, produktivnijih programskih jezika i razvojnih alata, sve obimniji programski kod se proizvodi u vrlo kratkom vremenskom periodu. Kompanije koje razvijaju softverski proizvod su pred izazovom razvoja sve složenijeg softvera, uz skraćanje vremena razvoja i sve većih očekivanja korisnika u pogledu kvaliteta softvera (pouzdanost, pogodnost za održavanje, pogodnost za testiranje, efikasnost, korektnost, robustnost i dr.) što potvrđuje da je testiranje softvera postalo izuzetno važna aktivnost u softverskom inženjerstvu. Aktivnosti u procesu razvoja softvera, u svakoj njegovoj fazi je podložna greškama, tako da otkazi softvera (defekti) igraju veoma važnu ulogu u procesu razvoja softvera. Problem testiranja softvera je takođe kompleksan zbog astronomsko velikog broja scenarija upotrebe softverskog proizvoda i stanja u kojima se program nalazi tokom upotrebe. Tipičan proces testiranja softvera je mentalno intenzivan ljudski rad i kao takav je neproduktivan, podložan greškama tj. urađen na neadekvatan način. Problem testiranja softvera je prošao kroz faze "ad hoc" tj. neorganizovanog, nesistematskog, nekontrolisanog i nedovoljno jasno definisanog mesta i uloge u procesu razvoja softvera, preko faze kreativnog ili umetničkog pristupa, zatim zanatskog pristupa nakon većeg broja publikovanih tehnika testiranja softvera po metodama "crne i bele kutije" (engl. *white and black box software methods*), do inženjerski postavljenog procesa testiranja softvera u zadnjih deset godina.

U poslednje vreme pojedini istraživači, među koje spada i autor ovog rada [2-5], svojim rezultatima istraživanja daju doprinos uspostavljanju naučne oblasti o testiranju softvera. Rezultati ovih istraživanja treba u značajnoj meri da spreče defekte u softveru koji se predaje korisniku na upotrebu, tj. da 'uteknu' do korisnika, čime se štite korisnici softvera, od manjih do enormnih gubitaka u poslovanju pa čak i gubitke ljudskih života (veliki je broj udesa u avio saobraćaju usled grešaka u softveru) čime se izbegavaju skupi sudski procesi kojih sve više u svetu ima danas.

2. OSNOVNI IZAZOVI U TESTIRANJU SOFTVERA

2.1. Stanje u oblasti testiranja softvera

Inženjering kvaliteta softvera je komponovan od dve osnovne aktivnosti koje se bave – nivoom kvaliteta procesa razvoja softvera, a koji se obično naziva obezbeđenjem kvaliteta i kvalitetom samog proizvoda. Nivo kvaliteta procesa razvoja softvera uspostavlja tehnike, procedure i alate koji pomažu promovisanju, ohrabrivanju, olakšavanju i kreiranju softverskog razvojnog okruženja u kome se izrađuje efikasan, optimiziran, prihvatljiv softverski proizvod sa minimalnim brojem grešaka. Kvalitet softverskog proizvoda se usmerava ka obezbeđenju što je moguće manje grešaka u softveru, njegovoj funkcionalnosti koja zadovoljava ili prevazilazi očekivanje korisnika softvera. Aktivnost testiranja softvera se normalno izvodi kao sredstvo pronalaženja grešaka sa ciljem njihovog odstranjivanja iz softverskog proizvoda. Ovo konačno dovodi do ključnog pitanja – pa šta se stvarno podrazumeva pod testiranjem softvera? Sve definicije testiranja softvera [6] čine skup – mitova o testiranju softvera, kao:

“Testiranje softvera je proces kojim se demonstrira da defekti ne postoje u softveru razvijenom za datu aplikaciju.”

“Testiranje softvera je aktivnost ili proces kojim se pokazuje ili demonstrira da program ili sistem izvršava sve predviđene funkcije korektno.”

“Testiranje softvera je aktivnost kojom se obezbeđuje potreban nivo poverenja u to da program ili sistem izvršava ono što je trebalo da uradi, na osnovu skupa zahteva koje je korisnik specificirao.”

Svi gore navedeni mitovi su opšti za sve definicije testiranja softvera. Međutim, u njima ima osnovnih pogrešnih shvatanja koje spadaju u mitove o testiranju softvera. Problem je u tome što – svi mitovi imaju pozitivistički pristup u shvatanju definicije testiranja softvera. Drugim rečima, svaki od navedenih mitova testiranja softvera, u nabrojanim definicijama je da je testiranje softvera aktivnost kojom se potvrđuje da program ili sistem nešto radi dobro. Međutim, vrlo je lako dokazati da softver/sistem nešto radi dobro, ali je teško dokazati da nešto ne radi dobro. Prvenstveno zbog toga, što ako neki konkretan test nije otkrio grešku (defekt), to ne znači da defekti u softveru ne postoje. Postavlja se pitanje, šta znači to što taj test nije otkrio defekt? Ovi mitovi ukopavaju nas u uverenju da svi testiranje softvera vide kao izbegavanje suštine grešaka u softveru [1-5]. Pa, šta je onda testiranje softvera? „Testiranje softvera je proces izvršavanja programa/funkcije sistema sa ciljem da se otkriju greške.“

Naglašena je posvećenost aktivnosti otkrivanja grešaka. Ovo je bitna razlika u odnosu na shvatanje da je važno potvrditi da program ili sistem radi. Ovakva definicija je data iz razloga što je softver jedan od najkompleksnijih proizvoda ljudskog umnog rada. Pa, zašto testiranje softvera u prvom planu? Zato što je jasno da je nemoguće otkriti sve greške u softveru. Zato što je jasno da je nemoguće dokazati da je softver bez greške. Takođe je jasno da je nemoguće pobediti prosto ubeđenje o imperativu da se otkriju sve greške. Pa, zašto se onda tolika pažnja u poslednje vreme posvećuje testiranju softvera kada ima toliko ograničavajućih faktora? Zato što su veliki gubici kompanija koje razvijaju softver upravo zbog velikog broja defekata u isporučenom softveru kupcima. Zato je prvenstveni zadatak test inženjera otkrivanje problema u softveru sa ciljem da se oni otklone pre predaje softverskog proizvoda kupcu. Zato što je nepouzdan proces razvoja softvera zahtevanog nivoa kvaliteta, a proizvođači vešto izbegavaju da u licencama za isporučen softver garantuju kvalitet. Ove i ostale činjenice, koje su navedene u ovom radu, su uslovile razvoj oblasti krivične odgovornosti proizvođača softvera i zaštite kupaca zbog neadekvatnog kvaliteta softverskog proizvoda [1].

Test inženjer mora i želi da otkrije što je moguće više problema i to što više onih, vrlo ozbiljnih čije posledice mogu biti katastrofalne sa materijalnog i bezbednosnog aspekta, prvenstveno. Tako postaje kritičan zahtev da se proces testiranja softvera učini što efikasnijim i uz što manje troškove ukoliko je to moguće.

Doprinos aktivnosti testiranja softvera je programski kôd visoke pouzdanosti, velike otpornosti (robustan), vrlo stabilan i da potvrdi da softver zadovoljava skoro sve zahteve krajnjeg korisnika ili one koje je obavezno zahtevao. Zato se aktivnost testiranja softvera smatra destruktivnom prema programskom kodu, mada se na kraju pokazuje da je ta aktivnost vrlo konstruktivna. Testiranje softvera je aktivnost za koju se vezuje negativan prizvuk iako je njen krajnji cilj, stvaranje boljeg softverskog proizvoda jer je operativno usmerena na 'slabe karike' procesa razvoja kao i ka kvalitetu samog softverskog proizvoda [2,3,6]. Stoga, ukoliko se uspostavi strožiji proces obezbeđenja kvaliteta softvera u pogledu efikasne prevencije i detekcije grešaka, utoliko menjamo kolektivnu svest o tome kako se obezbeđuje visok kvalitet softverskog proizvoda.

Drugi problem je, pak, taj da skoro nikad nema dovoljno vremena za testiranje softvera (TS). Zato treba da menjamo odnos prema aktivnosti testiranja softvera i vremena koje je planirano za testiranje, tako što ćemo ga efikasnije utrošiti u ranim fazama razvoja softvera [3]. Potrebno je misliti o testiranju softvera već prvog dana nakon početka projekta, a ne kao što je uobičajeno da se aktivnost testiranja softvera planira na kraju razvoja softvera .

Šta je to tako posebno u testiranju softvera?

- Čitav niz pitanja: tehničke, psihološke prirode, pitanja upravljanja projektom, marketingom kao i sama oblast primene.
- Na pitanju testiranja softvera se ukrštaju sva pitanja softverskog inženjerstva.
- Kako projekat odmiče, za sve propuste ostaje malo resursa da se oni otklone. Posledice odluke se odmah vide. Sa složenim odlukama se treba odmah suočiti i blagovremeno ih donositi.
- Testiranje igra sudbonosnu ulogu u uspehu – ili – neuspehu projekta jer:
 - Efikasan test menadžer i iskusan stručnjak (tester) mogu osigurati isporuku visoko-kvalitetnog softverskog proizvoda korisniku.
 - Manje iskusan tim za testiranje dovešće do raskoraka između ostvarenog doprinosa i onoga što aktivnost testiranja može objektivno da doprinese.

Pet fundamentalnih izazova za ostvarenje kompetentnog procesa testiranja softvera:

- Kompletan testiranje softvera je nemoguće zbog ograničenja u resursima, budžetu, vremenu i zahtevima kvaliteta softvera
- Test stručnjaci pogrešno koriste sredstva koja su im na raspolaganju zato što veruju u postojeće kompanijske mitove o procesu razvoja softvera
- Test grupe dobijaju različite zadatke i uloge koje su često u sukobu, retko usklađene
- Test grupe obično nemaju u svom sastavu iskusne programere niti viziju uspeha projekta koju tester i sa velikim programerskim iskustvom imaju
- Testiranje softvera kao deo procesa razvoja softvera je veoma intenzivan umni rad koji je nepouzdan, podložan greškama tj. sa visokim rizikom uspešnosti završetka projekata u planiranim okvirima.
- Zbog gore navedenog, potrebno je bilo redefinisati aktivnost testiranja softvera koja je poslužila kao osnova za istraživanje generičkog rešenja **Integralnog i Optimiziranog Proces Testiranja Softvera – IOPTS**, čijom *primenom se mogu*

izbeći sudski procesi usled nedovoljnog kvaliteta softvera [2]. Definicija je trebalo da obezbedi odgovore na otvorena pitanja iz oblasti testiranja softvera, a u okviru jedinstvene vizije i da usmeri istraživanje u pravom smeru.

Definicija 1. *Testiranje softvera se sastoji od aktivnosti dinamičke verifikacije ponašanja programa na bazi konačnog skupa testova, odabranih na pogodan način iz beskonačnog skupa mogućih načina izvršavanja programa, a prema specificiranom očekivanom ponašanju softvera u razvijanoj aplikaciji.*

2.2. Kompletno testiranje softvera je nemoguće

U bilo kom upotrebljivom softveru u praksi, postoji beskonačan broj mogućih testova koje je praktično nemoguće sve izvesti iz razloga koji su navedeni u literaturi [1,2,6]. Da bi se sve testiralo mora se, ali nije konačno:

- Testirati svaka moguća ulazna vrednost promenljivih sistema (fizička vrednost, svaki izbor operatera od ponuđenih opcija).
- Testirati svaku moguću kombinaciju ulaznih vrednosti promenljivih sistema zbog nepoznate zavisnosti odziva sistema na njihove interakcije.
- Testirati svaku moguću putanju kroz programski kod.
- Testirati svaku moguću hardversko-softversku konfiguraciju, uključujući i konfiguracije onih elemenata sistema koji nisu pod kontrolom testera.
- Testirati svaku moguću upotrebu softvera/sistema od strane korisnika (operatera) tzv. scenario misije sistema.

U literaturi se kao očigledan primer za potvrdu nemogućnosti kompletnog testiranja softvera navodi jednostavan misaoni eksperiment koji je zamislio Beizer [6], a koji se sastoji u tome da ako imamo program koji učitava niz (engl String) od 10 znakova i izvršava neku od potrebnih operacija nad njim. Da bi se ispitala ova funkcija programa kompletno, iscrpno t.j. bez ostatka potrebno je proveriti za sve moguće ulazne vrednosti niza tj. $2^{8*10}=2^{80}$, odnosno preko 120 890 000 000 000 000 000 000 test slučajeva i utvrditi da li je odziv programa korektan. Ako bi za jedan test trebalo mikro sekunda tada se dobija da je vreme potrebno za testiranje 38,334,790,264 godina. Pored toga potrebno je uključiti i nedozvoljene vrednosti niza npr. koji ima više od 10 znakova, pa i one koji imaju manje od 10 znakova kako bi se proverila otpornost programa t.j. robusnost na greške u ulaznim podacima što znatno povećava ionako već ogroman broj testova za dozvoljene ulazne vrednosti.

Jedan pristup (pokušaj) rešavanju ovog problema je njegovo uprošćavanje kroz tvrdnju da je izvršeno 'kompletno testiranje' ako izvršite 'kompletno prekrivanje'.

Šta je 'prekrivenost'?

- Mera za nivo istestiranosti određenog programskog svojstva, kao što je prekrivenost ispitanih instrukcija (komandi) programa ili grananja kroz program ili uslova u programu (IF, CASE, WHILE i ostalih struktura).
- Mera za nivo istestiranosti softvera u odnosu na populaciju mogućih testova, zahteva, specifikacija, karakteristika okruženja izvršavanja softvera i dr.

Ipak, pristup je prilično uprošćen. On je ispustio iz vida na primer:

- Ponašanje programa u uslovima rada pod prekidima ili paralelnim konkurentnim procesima.
- Specifične vrednosti ulaznih podataka programa i njihovim kombinacijama.
- Ispušten, nezavršen deo programa.

- U praksi, broj promenljivih koje treba meriti je astronomsko veliki, a još veći problem je što neke ne možemo ni kontrolisati, kao što su karakteristike okoline tj. okruženje softvera/sistema.

Mera prekrivenosti je interesantan način da se izrazi činjenica da je testiranje izvršeno kompletno, ali pristup testiranju softvera sa ciljem da se postigne ‘visok’ nivo prekrivenosti najčešće dovodi do preteranog broja testova niske efikasnosti – dobijaju se manje korisne informacije o programu nakon testa. Ljudi, pa i projektanti, obično optimiziraju ono što obično mere, čemu poklanjamo pažnju tj. zanemaruju ono što ne merimo. Više autora, kao Brian Marick [7], je isticalo pogrešan pristup samoj definiciji, svrsi testiranja softvera kao što je i pristup prekrivenosti u člancima koji se mogu naći na njegovom sajtu www.testing.com (npr. *How to Misuse Code Coverage*). Neki kao rešenje nude mere kompletnosti ili nivo ispitivosti softvera prikazujući grafički stanje grešaka u softveru kao:

- Broj novih otkrivenih grešaka u nedelji
- Nepopravljene greške (svake nedelje)
- Odnos otkrivenih prema otklonjenim (popravljenim) greškama
- Slaganje realne sa teorijskom krivom, obično krive pouzdanosti i odstupanje (varijansa) od teorijske.

U određenom trenutku jasno se ‘vidi’ sa grafika šta je urađeno. Opšti model raspodele grešaka tzv. Vejbulova raspodela i njene pretpostavke da :

1. Testiranje pokazuje kako će se program ponašati u eksploataciji.
2. Sve greške imaju istu verovatnoću pojavljivanja.
3. Greške su nezavisne jedna od druge.
4. U početku testiranja postoji fiksni broj grešaka u softveru tj. statična pretpostavka generisanja grešaka.
5. Vremenska raspodela otkrivanja grešaka podleže Vejbulovoj raspodeli.
6. Broj otkrivenih grešaka u nekom vremenskom intervalu je nezavisan od broja otkrivenih grešaka u drugom vremenskom intervalu.

U praksi se pokazalo kao apsurdno, za neki konkretan projekat, pretpostaviti Vejbulovu funkciju raspodele grešaka u vremenu, jer je neka od pretpostavki bila apsurdna itd.

Postavljaju se onda sledeća pitanja. Šta nam sve to govori? Kako ih onda objasniti?

U početnom periodu istorije testiranja softvera: Pritisak je bio na što većem broju otkrivenih grešaka kroz

- Izvršavanje onih testova koji će otkriti krah onih funkcija softvera koje nisu do kraja realizovane ili čije izvršavanje će verovatno dovesti do otkaza sistema.
- Izvršavati međusobno povezane testove koji dovode do niza grešaka koje su višestruka manifestacija istog uzroka.
- Fokusiranje na lakše greške kako bi broj otkrivenih grešaka bio veći, a ne na one čije su posledice katastrofalne.
- Manje angažovanje na infrastrukturi procesa testiranja, automatizaciji, alatima već posvećenost na što veći broj otkrivenih grešaka (kratkoročan uspeh ali dugoročna neefikasnost).

U kasnijem periodu istorije testiranja softvera: Pritisak je bio na smanjenju broja novootkrivenih grešaka kroz:

- Izvršavanje što više onih testova koji su već otkrili greške tj. regresiono testiranje.
- Smanjenje angažovanja na otkrivanju novih grešaka u softveru.
- Prenosivanje pažnje na procenjivanje, izveštaje o stanju procesa testiranja softvera.
- Klasifikaciju dupliranih grešaka.

- Klasifikaciju povezanih grešaka kao duplirane (i kao rešene, otklonjene) skrivajući ključne podatke o simptomima/uzrocima greške u softveru.
- Odlaganje izveštavanja o otkrivenim greškama do kontrolnih tačaka razvojnog procesa (faza) čime je dolazilo do gubitaka informacija o greškama u softveru.
- Izveštavanje o greškama neformalno, nije uključeno u jedinstven sistem praćenja projekta.
- Stručnjaci za testiranje se premeštaju na druge aktivnosti pre kontrolnih tačaka razvoja softvera.
- Ignorisanje grešaka koje programeri otkriju dok ih testeri ne otkriju.
- Odnos prema greškama je prepušten ličnom stavu članova tima.
- Veliki broj detektovanih grešaka je odbačen, ignorisan.

U kasnijem periodu istorije testiranja softvera problem se sastojao u uprošćenim pogledima i rešenjima na složenost procesa testiranja softvera, u šta se svako mogao uveriti na osnovu izveštaja u medijima o katastrofalnim posledicama nekih grešaka u softveru. To je dovelo do otkrića da je rezervisano tj. planirano vreme za aktivnosti testiranja softvera mnogostruko manje od potrebnog. Na primer: procenjeno vreme potrošeno na analizu, otklanjanje grešaka i opisivanje uzroka grešaka je mnogostruko manje od stvarno utrošenog. Rezultat je bio, da nije ostalo dovoljno vremena za:

- - Projektovanje testova,
- - Dokumentovanje testova,
- - Izvršavanje testova,
- - Automatizaciju testova,
- - Preglede i provere testova,
- - Uvođenje novih alata i opreme u proces testiranja softvera,
- - Obuku novih kadrova.

Da bi ispunili današnju biznis mantru: *bolje, jeftinije i brže* potrebno je realizovati više kompromisa kao:

- Iz velikog broja mogućih testova iz čitave populacije moguće je izvršiti samo mali, ograničen uzorak testova. To je dovelo do pitanja: koje testove od mnogih treba odabrati tj. kako optimalno izvršiti uzorkovanje?
- Odrediti konkurentne (koje treba uzeti u obzir i vrednovati) karakteristike dobrih testova. Jedan test je bolji od drugog ako:
 - Je snažniji u pogledu mogućnosti pribavljanja informacija o softveru nakon testa
 - Koji sa većom verovatnoćom doprinosi značajne rezultate (veća motivisanost uspeha projekta, ubedljivost)
 - Ima veće uverenje i verodostojnost u rezultat testa (kredibilnost)
 - Bolje reprezentuje korisničke potrebe i primere primene softvera u praksi
 - Lakše je izvršiti vrednovanje doprinosa izvršavanjem testa
 - Korisniji za otklanjanje, ispravljanje greške u softveru
 - Ako daje više informacija o softveru
 - Manje komplikovan za realizaciju
 - Sa većom verovatnoćom može pomoći testeru ili programeru da spozna neku karakteristiku softvera, potrebu kupca ili uslova, okruženja eksploatacije softvera.
- Iz velikog broja mogućih testova iz čitave populacije moguće je izvršiti samo mali, ograničen uzorak testova. To je dovelo do pitanja: koje testove od mnogih treba odabrati tj. kako optimalno izvršiti uzorkovanje?

Nijedan test ne zadovoljava sve ove karakteristike dobrog testa. Kako izvršiti balansiranje, kompromis? Ovo je još uvek otvoreno pitanje u oblasti testiranja softvera.

2.3. Mitovi o procesu razvoja softvera

Svaki učesnik u procesu razvoja softvera (PRS) ističe da veruje u:

- mi se pridržavamo procedura u PRS po modelu 'Vodopada' (engl Waterfall lifecycle),
- mi prikupljamo sve zahteve korisnika, kupaca za softverski proizvod na početku projekta, tako da aktivnosti baziramo na dokumentu o zahtevima za performanse softvera,
- mi pravimo detaljnu, korektnu specifikaciju karakteristika softvera i održavamo ih ažurirane tokom PRS,
- mi verujemo da će kupac prihvatiti softver čije je ponašanje striktno prema dokumentovanoj specifikaciji,
- mi ispravljamo (otklanamo) sve greške prioriteta (ozbiljnosti) x i nikad ne snižavamo stepen ozbiljnosti (prioriteta) sa ciljem da se izbegne njihovo ispravljanje.

Začuđujuće je da mnogi koji se bave testiranjem softvera veruju u gore spomenute tvrdnje, iz projekta u projekat i ponašaju se u skladu sa njima. Efekti verovanja i ponašanja prema opisanim mitovima su:

- Tester i projektuju testove na osnovu specifikacije/zahteva mnogo ranije pre gotovog programskog koda. Konačno, mi znamo kako će program izgledati.
- Tester i ocenjuju programske mogućnosti u poređenju sa pisanim zahtevima za softver, bez sopstvenog rasuđivanja i stava o softverskom proizvodu. Konačno, mi znamo šta je korisnik želeo.
- Tester i ocenjuju korektnost programa jedino u odnosu na njegovu specifikaciju, bez sopstvenog rasuđivanja i stava o softverskom proizvodu. Konačno, mi znamo šta je korisnik želeo.
- Tester i prave obimne, krhke, skupove testova za korisnički grafički interfejs (engl. GUI) koji se često ponavljaju. Konačno, korisnički interfejs je detaljno specificiran. Mi znamo da se on neće menjati.
- Mi znamo očekivani rezultat svakog testa (engl Oracle). Orakl problem se sastoji u pronalaženju nekog postupka, načina da se oceni da li je test uspešan ili ne, da li je softver/sistem ispunio, odnosno zadovoljio neki zahtev ili ne. U praksi se, ipak često dešava da ne možemo sigurno suditi o rezultatu testa, često je potrebno više načina, metoda da se utvrdi tačan rezultat testa tj. više Oracle mehanizama u jednom projektu, a često se donesu i pogrešne ocene o ishodu testa i tumačenje rezultata softverskog testa sa aspekta misije sistema koji se ispituje.

2.3.1. Višestruke uloge

Ukoliko je nivo zrelosti kompanije koja razvija softver nizak tzv. SEI CMM [34,177] nivo i nivo zrelosti procesa testiranja tj. TMM [27,179] nivo nizak, tada se članovima test tima dodeljuju različita nejasna zaduženja, često i pogrešna u vezi:

- Detekcije grešaka u softveru
- Isporuke nezrelog softverskog proizvoda
- Pomoći menadžerima u donošenju odluke o momentu isporuke softvera
- Minimizacije troškova tehničke podrške
- Saglasnosti sa zakonskom regulativom

- Minimiziranja rizika vezanih za bezbednost aplikacije
- Određivanja bezbednih scenarija upotrebe softvera
- Ocene kvaliteta softvera
- Ocene saglasnosti performansi softvera sa specifikacijom
- Verifikacije korektnosti softvera

2.3.2. Slab izbor članova test tima

Nužan je sistematičan pristup razvoju softverskog proizvoda (akviziciji) koja povećava stepen zadovoljenja korisnikovih potreba kroz usklađene aktivnosti različitih naučnih i inženjerskih disciplina tokom razvojnog ciklusa kao u strategiji Integracije Razvojnog Procesu i Proizvoda (IRPP) [6]. Posebna pažnja mora da se posveti Timskom pod-procesu, njegovom kreiranju i implementaciji tzv. Integrisanog Proizvodnog Timu kao delu organizacione strukture kompanije i konkretnog projekta, koji verovatno predstavlja kritičan momenat za uspeh celog IRPP. Neke pogrešne predstave o izboru članova tima za posledicu imaju nekoliko dole nabrojanih zabluda. Optimalni sastav tima za testiranje softvera treba da uključi članove koji poseduju različito iskustvo i nivo znanja za datu aplikaciju. Sledeći zaključci se pogrešno izvode, kao:

- Malo iskustvo u programiranju znači slaba sposobnost testiranja softvera i obrnuto.
- Dobro iskustvo u programiranju osigurava uspeh u testiranju.
- Veliki broj strategija testiranja softvera ne zahteva veliko znanje i iskustvo u programiranju.
- Zaposleni koji žele da budu u procesu proizvodnje, a koji ne znaju da programiraju, nemaju čime da se bave već testiranjem softvera.
- Zaposleni koji imaju veliko iskustvo u programiranju plaše se da ne završe kao članovi tima za testiranje.

Sposobnost članova tima za testiranje u značajnoj meri utiče na uspeh ili neuspeh aktivnosti uložених u testiranje pa čak i na ceo projekat. Efikasan tim mora biti kombinovan od stručnjaka sa iskustvom iz tehničke oblasti primenjene u misiji softvera/sistema kao i specifičnih iskustava iz oblasti softverskog inženjerstva. To znači da nije samo važno iskustvo, znanje tehnika i alata za testiranje softvera. Ova znanja i iskustva omogućavaju da se realizuje efikasan test proces. Dalje, tim stručnjaka koji su uključeni u proces testiranja moraju imati preciznu i definisanu organizacionu strukturu gde su zadaci i uloga svakog člana tima maksimalno precizirani kako bi se izbeglo dupliranje poslova i odgovornosti.

2.4. Nepouzdanost uspeha procesa testiranje softvera

Postoje bar četiri oblasti u softverskom inženjerstvu gde je nepouzdanost tj. rizik uspeha misije evidentan: nepouzdana analiza korisničkih potreba tj. softverskih zahteva, nepouzdan proces realizacije korisničkih zahteva u softverski dizajn i programski kod, nepouzdan proces redizajna i iskorišćavanja gotovih softvera [2]. Testiranje softvera je visoko intelektualna aktivnost, kao i ostale aktivnosti u razvoju softvera, pa stoga uključuju najvećim delom ljudski rad koji je podložan greškama tj. vrlo je nepouzdan i zadovoljava maksimu nepouzdanosti u softverskom inženjerstvu [1,2]. Pomenuta nepouzdanost u testiranju softvera ima za posledicu realan rizik koji može značajno uticati na uspeh celog projekta razvoja softverskog proizvoda pa ga stoga treba uračunavati pri izradi plana testiranja softvera. Identifikovana su tri moguća rizika pri izradi plana testiranja softvera:

nepoznavanje softvera/sistema koji se razvija i koga treba testirati, neizvesnost u planiranju, proceni napora (sredstva, vreme, troškovi) za izvršavanje test aktivnosti kao i pouzdanost realizacije planiranih test aktivnosti. Takođe postoji čitav niz nepouzdatih aktivnosti u procesu testiranja softvera, konkretno, provera rezultata testova je česta i rutinska jer se testovi ponavljaju, pa ako se radi ručno i uz angažovanje čoveka onda je vrlo verovatno da će biti sa puno grešaka tj. uneće dodatni rizik u oceni kvaliteta softvera. U izradi plana testiranja još uvek čovek igra glavnu ulogu, naročito u početnim fazama razvoja softvera, unoseći rizik u obavljanje tog posla koji se dalje prostire u naredne faze razvoja softvera. Posebna pažnja mora biti posvećena izradi rezultata testova (test Oracle) kako bi se pouzdano ocenjivali rezultati testova od čega zavise dalje aktivnosti u razvoju softvera. Pogrešna ocena rezultata testa (ponašanje softvera) bilo da se tvrdi da je test uspešan, a on stvarno nije bio, ili da je rezultat testa neuspešan pa je otkrivna greška koja stvarno to nije, nakon čega se vrši dodatno projektovanje, pa kodiranje pa ponovno testiranje radi provere da je ta greška otklonjena itd., može doprineti produženju rokova, troškova i isporuke nekvalitetnog softverskog proizvoda. U praksi je, često potrebno napraviti više tipova rezultata testova tj. test Oracle mehanizama za jedan test kako bi se pouzdano ocenio rezultat testa. Opet, dešava se da jedan test Oracle može biti iskorišćen za više testova. U analizi rezultata testa, ponašanja softvera u datom testu, može se koristiti neki od test Oracle tipova kao: a) test Oracle koji daje tačan odziv softvera za svaki ulazni podatak, ulaz softvera, b) test Oracle koji daje opseg ponašanja, odziva softvera i c) test Oracle koji ne može da da izlaz, odziv softvera kako bi ocenio rezultat testa u nekim slučajevima.

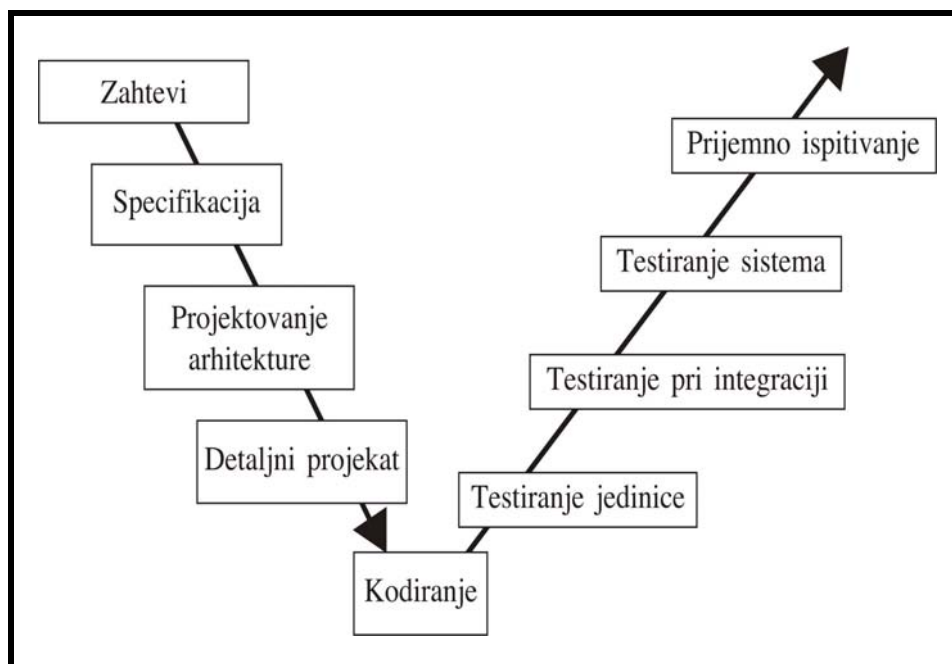
3. MODEL PROCESA RAZVOJA –TESTIRANJA SOFTVERA

Sve kompanije koje se bave razvojem softvera slede neki model procesa razvoja softverskog proizvoda. U nedovoljno zrelih organizacijama, proces se često odvija na iskustvu nekoliko stručnjaka i nije dokumentovan ili jeste dokumentovan ali se ne poštuju propisani postupci. U kompanijama sa zrelijom organizacijom, Proces Razvoja Softvera-Testiranja (PRS-PTS) je detaljno dokumentovan, striktno se sprovodi i aktivno se njime upravlja, a kao rezultat se potvrdilo da je isporučen softver sa manje grešaka . Aktivno upravljanje PRS-PTS podrazumeva da je proces dinamičan a ne statičan, da se njegova efikasnost meri na unapred definisan i regulisan način tako da se u realnom vremenu, a na bazi merenja efikasnosti vrši njegovo poboljšanje. Pošto izbor modela PRS-PTS ima veliki uticaj na kvalitet razvijenog softverskog proizvoda, vreme završetka razvoja i pojave na tržištu, troškove razvoja i troškove podrške u eksploataciji softverskog proizvoda u njegovom životnom veku, upravo zbog ovih činjenica u ovom radu se glavna pažnja poklanja *modelu procesa testiranja softvera* pored samih tehnika testiranja softvera. Razvoj složenog softverskog proizvoda se meri stotinama inženjer-godina angažovanih na njegovom razvoju i aktivno opstaje na tržištu i u upotrebi više decenija. Kao rezultat, kod ovakvih softverskih proizvoda, ukupni troškovi njihovih kompanija vezani za tehničku podršku i održavanje tog softverskog proizvoda pokazalo se da direktno zavise od robusnosti rešenja i kompletnosti dokumentacije softvera. Pošto je proces testiranja softvera složen dinamički proces, sa velikim brojem promenljivih, posebnu pažnju treba posvetiti upravljanju ovim procesom u cilju ostvarenja njegove prediktivnosti i stabilnosti.

3.1. Kontrolabilnost i obzervabilnost Procesa Testiranja Softvera

Stalni tehnološki napredak nameće sve veće zahteve u razvoju softverskog proizvoda tako da je on sve složeniji i glomazniji, da je robustan, pouzdan uz stalni rast broja primena. Takođe su sve veći zahtevi u pogledu upravljanja sve složenijim softverskim projektima. Projektanti softvera i serviseri tj. menadžeri i ostalo osoblje su stalno suočeni sa novim tehnološkim izazovima, sve oštrijom tržišnom utakmicom, sve većom konkurencijom iskusnih projekatana i ostalog osoblja na koje moraju brzo da odgovore. U isto vreme su suočeni sa nekompletiranim zahtevima u pogledu performansi i kvaliteta, slabom kontrolom promena sofvera, nedovoljne ispitanosti softvera, neadekvatne obuke kadrova, proizvoljnim rokovima završetka razvojnih zadataka, nedovoljnim budžetom kao i otvorenim pitanjima vezanim za standardizaciju, pouzdanost i stabilnost softverskog proizvoda. Definicija procesa razvoja softverskog proizvoda, prema SEI CMM (Institutu za Softverski Inženjering) [6], uključuje ljude, materijale, energiju, opremu, softverske inženjerske tehnike skladno ukomponovane u proces koji se može predstaviti ‘nizom koraka’ kojima osoblje uz pomoć alata i odgovarajuće opreme materijal pretvaraju u proizvod. Za potrebe ovog istraživanja, mi ćemo koristiti najrasprostranjeniji V-model PRS-PTS u industriji softvera koji je prikazan na slici 1 [2], na kome ćemo demonstrirati svu složenost PRS-PTS, otvorena pitanja i probleme koji traže odgovor i rešenja.

U V-modelu, aktivnosti testiranja softvera su predstavljene kroz testiranje softverskih jedinica čiji zadatak je da se proverí da li je programski kod u saglasnosti sa detaljnim projektom. Aktivnost testiranja u fazi (koraku) integracije softverskih jedinica treba da pokaže da prethodno testirane softverske jedinice u međusobnoj interakciji izvršavaju korektno zahtevane funkcije softvera. U fazi ispitivanja sistema treba da se proverí da li integrisan sistem/softver zadovoljava specifikaciju sistema/softvera. Na kraju, prijemno ispitivanje treba da proverí da li su ispunjeni i zahtevi korisnika u pogledu performansi i kvaliteta sistema/softvera. Treba naglasiti da korisnici V-modela po pravilu razdvajaju proces projektovanja od aktivnosti testiranja. Projektovanje aktivnosti testiranja se vrši onda kada je odgovarajući dokument ili element softvera spreman tj. razvijen. Testovi su pravljani tako da se otkriju greške u određenoj izolovanoj softverskoj jedinici, okruženoj sa pripremljenim pokretačima i odzivnicima (engl. *drivers and stubs*). Takođe, mogu biti testirani delovi softvera kao podsistemi- zajedno sa testovima pravljenim da se otkriju greške pri integraciji softvera, ili pak, da se simuliraju veze između podsistema, pošto se pri testiranju podsistema takođe zahtevaju pokretači (engl. *Drivers*) i odzivnici (engl. *Stubs*) što ponekad dovodi do boljeg rešenja da se testiranje nekih softverskih jedinica ili čak testiranje integracije podsistema odloži do faze kada je ceo sistem ili odgovarajuća njegova celina završena. U tom koraku onda, osoblje za testiranje izvršava testiranje softverske jedinice, testiranje integrisanog sistema, celog sistema/softvera u interakciji sa spoljašnjom okolinom jer se tada postiže optimizacija procesa testiranja softvera kroz minimiziranje troškova procesa testiranja softvera, isporuku softvera zahtevanog nivoa kvaliteta, a uz ograničenje u vremenu i resursima za izvođenje pojedinih aktivnosti testiranja softvera. To nas navodi na zaključak da V-model treba korigovati, jer ako se želi postići optimizirani PTS tada oštra razlika u statičnom V-modelu između aktivnosti testiranja “softverske jedinice”, “integracije” i “sistema” mora da se ukine. Kao rezultat imamo da treba početi tragati za modelom optimiziranog PTS preko modifikacije V-modela prikazanog na slici 2 [2,3]..



Slika 1. V-model procesa razvoja i testiranja softvera- PRS-PTS

3.2. Uočeni nerešeni problemi i osnovne hipoteze

Testiranje softvera je bitan preduslov za uspešan razvoj i implementaciju Informacionog Sistema (IS). Ali, često se testiranje softvera smatra đavolskim poslom: zato što je komplikovan i nekontrolisan proces koji traje predugo, preskup je, a na kraju, najčešće ne dovodi do IS bez grešaka tj. ozbiljnih problema. Nažalost, u najvećem broju slučajeva ova tvrdnja je potvrđena. Softverski projekti su veoma rizični po pitanju uspeha. Velike, ali i male kompanije, još uvek imaju problema sa tačnom predikcijom troškova i vremena završavanja pojedinih aktivnosti u procesu projektovanja softvera, u postizanju visokog kvaliteta softverskog proizvoda i konačno u zadovoljavanju sve većih očekivanja potencijalnih kupaca softverskog proizvoda. U svom, već klasičnom radu iz 1994 u časopisu *Scientific American*, W. Wayt Gibbs je citirao podatke iz nekoliko projekata koji ukazuju na to da je PRS trajao 50% duže od planiranog, da su 2 od 6 velikih projekata prekinuti kao neuspešni. John Thorp je publikovao još poraznije statističke podatke u *The Information Paradox* citirajući rezultate istraživanja Gartner Group u kojima je navedeno da je u oblasti IT, od ukupnih investicija u periodu od 1985 do 1995, bednih 1% ostvarilo očekivani profit i da je glavni razlog da je dobijen softverski proizvod sa ozbiljnim greškama ili vrlo komplikovane upotrebe. U časopisu *Business Week*, od 6 decembra 1999, predsednik kompanije Standish Group, koja se bavi analizom tržišta IT je konstatovao da je loš softver koštao industriju SAD \$85 milijardi. Poslednji izveštaj njegove kompanije iz 1994, koji je kombinovao podatke nekoliko hiljada projekata rađenih za potrebe državnih organa i industrije konstatovao je da je samo 16% softverskih projekata završeno na vreme i u okviru planiranog budžeta pri čemu je prosečno 61% od zahtevanih funkcija realizovano, a nedopustivih 31% projekata nije nikada završeno! Takođe, oko 53% projekata koštalo je 190% od procenjenih troškova na početku realizacije projekta. U međuvremenu, određeni pomak je učinjen, ali još uvek nedovoljan jer poznati izveštaj iste *Standish Group Chaos Report* kompanije iz 2001 na bazi 8,000 pregledanih velikih projekata konstatovao je da 25% velikih projekata nikada nije završen usled znatnog prekoračenja u troškovima, vremenu razvoja, lošem kvalitetu ili njihovom kombinacijom. Takođe je prosečno 90% više potrošeno od planiranog, preko 120% je duže trajao razvoj

od planiranog kod završenih projekata. Mada na testiranje softvera u proseku otpada 25 do 70 procenata ukupnog budžeta projekta, u samo nekoliko organizacija menadžment je planirao potrebne resurse za proces testiranja softvera i na pravi način. Još, ako se tome dodaju katastrofalne posledice grešaka u softveru iz Top 10 liste najvećih softverskih grešaka, koje su izazvale veliku nesreću ili kvar sistema kao što je eksplozija evropske svemirske letelice Ariane 5, 1995 godine vredne \$800 miliona dolara, THERAC-25 greška u linearnom akceleratoru, uređaju u medicini koji je usmrtio 3 pacijenta..., postaje očigledna potreba da se istraživanje posveti razvoju efikasnog i efektivnog procesa testiranja softvera koji će u okviru ograničenja u pogledu vremena razvoja, budžeta i ostalih resursa kompanije optimalno realizovati zahtevani nivo kvaliteta i performanse softvera.

3.3. Ciljevi istraživanja

Dosadašnji pristup problemu testiranja softvera je bio takav da se kroz čitav repertoar međusobno nezavisnih tehnika i strategija testiranja softvera oceni nivo kvaliteta softverskog proizvoda tj. pristup je bio pasivan. Ovakav pristup je kao rezultat davao samo konstataciju da softverski proizvod nije korektan jer su detektovane (otkrivene) greške i to najčešće u kasnijim fazama razvoja softvera tj. u fazi testiranja dok su one pravljene mnogo ranije te je život softverskih grešaka bio dug od momenta generisanja do momenta otkrivanja. Kako je dobro poznato, cena popravke softverskih grešaka je za red veličine veća ukoliko se one ne otkriju u fazi razvoja u kojoj su napravljene tako da je otklanjanje grešaka napravljenih u ranim fazama npr. u fazi izrade projektnih zahteva karakteristika softverskog proizvoda a otkrivenih tek u završnoj fazi pri testiranju sistema i hiljadu puta veća nego da su odmah i otkrivene kad su nastale, što je prikazano na slici 3. Pošto je nemoguće u razumnom vremenskom periodu, sa ograničenim resursima (ljudi, oprema i alati) izvršiti iscrpno (potpuna provera korektnosti softvera za sve moguće slučajeve i stanja programa) kao ni matematički dokazati da je softver bez greške, cilj istraživanja je bio da se predloži ***Integralni i Optimizirani Proces Testiranja Softvera (IOPTS)*** (prikazan šematski na Slici 4) uz navedena ograničenja u pogledu vremena, predviđenog budžeta, a uz postizanje što boljeg kvaliteta projektovanog softverskog proizvoda. Generičko rešenje IOPTS [2,3] trebalo je da:

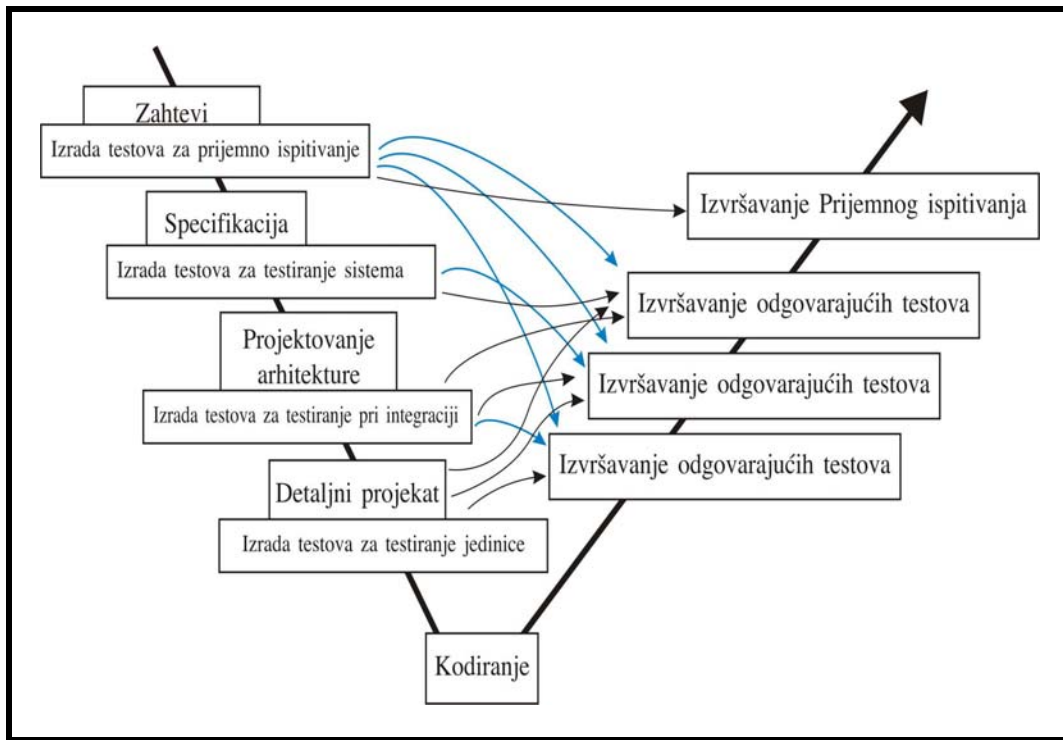
- **Smanji vreme razvoja softverskog proizvoda**

U odnosu na ranije pristupe modelovanju PTS, koji se zasnivao na sekvencijalnim aktivnostima testiranja softvera, nov pristup se mora bazirati na paralelizovanju aktivnosti na bazi integralnog pristupa PTS. Sve odluke u upravljanju projektom se moraju donositi na bazi parametara celokupnog ciklusa razvoja softvera i svih aktivnosti koje se odvijaju na planiran, sistematičan način kroz minimiziranje broja i kompleksnosti izmena u softverskom proizvodu u toku razvojnog ciklusa i eksploatacije softverskog proizvoda. Smanjenje broja izmena kao i onih složenih izmena u softveru smanjiće troškove doradivanja softverskog proizvoda u pojedinim fazama razvoja softvera i imaće pozitivan uticaj na smanjenje trajanja pojedinih faza razvoja tokom celog ciklusa razvoja i na smanjenje ukupnih troškova razvoja softverskog proizvoda.

- **Smanji troškove razvoja softvera-sistema**

Odgovarajući integralni pristup u modelovanju PTS od samog početka PRS omogućio optimiziranje procesa, a i samog softverskog proizvoda. Raniji pristupi modelovanja PTS na bazi istorijskih podataka ranijih projekata više nisu relevantni. Dodatna sredstva i naponi su potrebni u ranijim fazama PRS-PTS tako da će jedinična cena proizvoda kao i ukupni troškovi razvoja biti manji usled smanjenog broja izmena softvera, povećanja opštih

moćnosti za postizanje postavljenih ciljeva u pogledu trajanja pojedinih aktivnosti u PRS-PTS, intenzivne primene analize usaglašenih parametara za pronalaženje efikasnih i jeftinih rešenja.



Slika 2. Modifikovani V-model PRS-PTS

- **Smanji rizik**

Dobro isplaniran i blagovremen trening budućeg tima u ranim fazama PRS-PTS obezbediće bolje razumevanje softverskog proizvoda koji treba da se razvije, ocenu raspoloživih tehnologija i procesa koje treba primeniti u konkretnom projektu. Ovo će na kraju doprineti boljem razumevanju svih rizika koji će uticati na troškove, planirano vreme i performanse softverskog proizvoda. Efikasna procena rizika dovešće do izbora adekvatnih metoda i procesa koji će smanjiti ili ublažiti potencijalne rizike i omogućiti postizanje realnijih ciljeva u pogledu performansi, troškova i vremena realizacije pojedinih aktivnosti u PRS-PTS. Ključnu ulogu u ostvarenju ovog cilja igra PTS koji menadžere na projektu snabdeva važnim i kredibilnim informacijama o performansama PRS-PTS i kvalitetu razvijanog softvera.

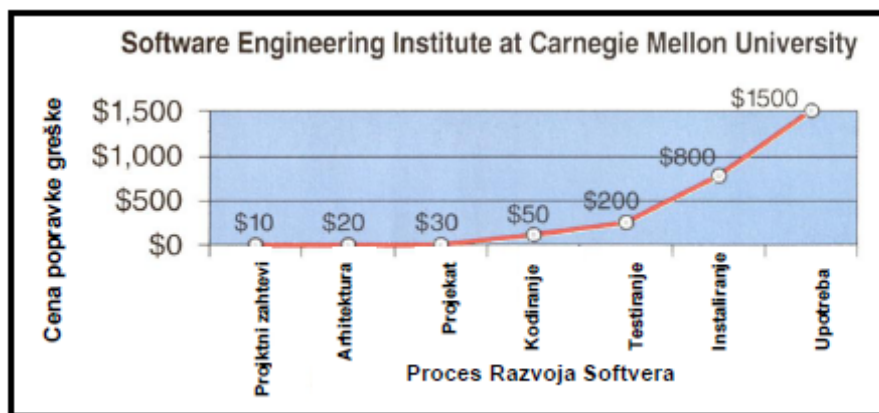
- **Poboljša kvalitet softvera i PRS-PTS**

Glavni cilj PRS-PTS je da se postigne visoki kvalitet softvera tj. softverski proizvod koji je zadovoljio postavljene zahteve kvaliteta softvera i koji će u potpunosti zadovoljiti potrebe korisnika. Da bi se blagovremeno otkrile greške u dizajnu i kodiranju, da bi se ocenila pouzdanost softvera i da bi se kupci ubedili u prihvatljive performanse softvera, on se mora adekvatno testirati. Timski rad projektanata i test osoblja mora se zasnivati na dobroj komunikaciji i uzajamnoj pomoći u primeni svih saznanja u cilju stalnog poboljšanja kvaliteta softverskog proizvoda kao i samog PRS-PTS.

3.4. Značaj istraživanja

a) Predloženo rešenje IOPTS, koje je šematski prikazano na slici 4, uključilo je najbolja rešenja (engl „best practices“) u oblasti projektovanja i testiranja softvera kojima se na

dokumentovan, sistematizovan i kontrolisan način obezbeđuje i ugrađuje zahtevani kvalitet softvera, u okviru predviđenog vremena i raspoloživih sredstava preduzeća (budžet, ljudi, oprema itd.)



Slika 3. Eksponencijalni rast troškova detekcije i otklanjanje grešaka u TS

b) Implementiranje agilnog IOPTS kroz prevenciju generisanja grešaka primenom: modelovanja, simulacije, planiranog eksperimenta, six-sigma u projektovanju i razvoju softvera, sprečava propagaciju grešaka u kasnije faze, a posebno smanjuje troškove održavanja softvera kao i katastrofalne posledice nekih grešaka u softveru tokom njegove eksploatacije [3,5].

c) Očekivani pojedinačni efekti, značajnog unapređenja efikasnosti testiranja softvera primenom IOPTS u vojnim i ostalim primenama sa kritičnom misijom u periodu od 3 godine, doneli bi povećanje ukupne produktivnosti više od 100 puta za svaku uloženu novčanu jedinicu, tj. ROI od 100:1 [2], što je i potvrđeno u objavljenim rezultatima u radovima [3,5].

3.5. Postavka problema

Koliko je autoru ovog rada poznato, nema do sada publikovanog rada koji rešava integralni i optimizirani proces testiranja softvera koji zadovoljava 3,4 i 5 nivo zrelosti procesa testiranja softvera prema gradaciji tj. modelima zrelosti procesa testiranja softvera (TMM) kako je opisano u Tabeli 1, a koji je objavio Illinois Institute of Technology iz Čikaga 1996 godine [6], analogno CMM modelu zrelosti procesa razvoja softvera (CMM) koji je razvio Software Engineering Institute (SEI) of Carnegie Mallon University.

Pošto je u praksi potvrđeno da postoji visok rizik tj. nizak nivo poverenja u uspešnost razvoja kvalitetnog softverskog proizvoda u planiranom vremenskom periodu, sa planiranim budžetom uz ograničenja u resursima svake kompanije (ljudi, oprema i razvojni alati) postavljene su i potvrđene sledeće hipoteze:

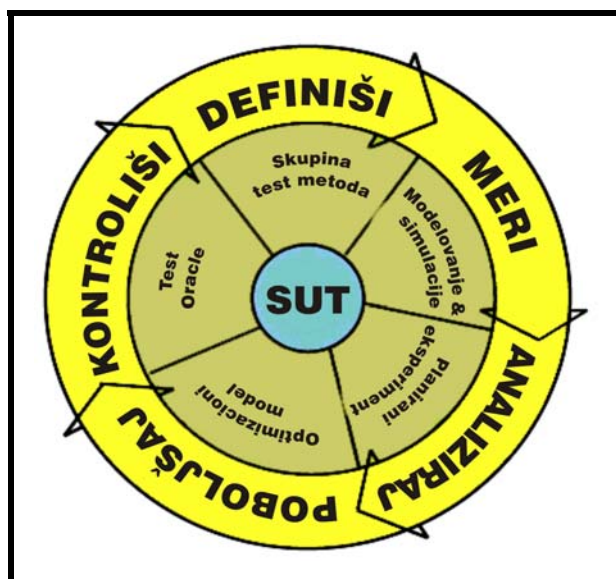
1. Moguće je razviti Integralni i Optimizirani model testiranja softvera (IOPTS) kroz simulaciju scenarija testiranja softvera na osnovu raspoloživih (merenih) iskustvenih podataka date, konkretne kompanije u pogledu razvojnih resursa (znanje i iskustvo ljudi, razvojnih alata i opreme) i/ili raspoloživih (merenih) iskustvenih podataka drugih kompanija sličnog nivoa zrelosti (TMM i CMM) [2,3,5].

2. Generičko rešenje IOPTS koje uključuje Modelovanje i Simulaciju (M&S) na kompjuteru je aktivno rešenje (umesto dosadašnjih pasivnih rešenja) koje je izvor informacija za stalno poboljšanje procesa testiranja softvera tj. njegovu optimizaciju u svim fazama razvoja softvera, jer pravljenje modela softverskog proizvoda je ujedno i testiranje i obrnuto! Pored toga M&S je vrlo efikasno rešenje referentnog softvera (korektnog) tzv. (engl. Test Oracle) problema [2,5,6].
3. Generičko rešenje IOPTS koje uključuje merenje efikasnosti i efektivnosti procesa testiranja softvera na osnovu potrebnog i dovoljnog broja metrika (osnovnih i izvedenih) procesa razvoja i testiranja softverskog proizvoda je aktivno rešenje koje je izvor informacija za stalno poboljšanje procesa testiranja softvera tj. njegovu optimizaciju [2,3,6].

1. Tabela 1. Nivoi zrelosti TS tzv. TMM levels

TMM Nivo 1 Inicijalna faza: Testiranje softvera (TS) je haotičan proces; loše je definisan i nije jasno razgraničen sa fazom otklanjanja grešaka (debugging). Testiranju se pristupa neplanirano i na kraju faze kodiranja programa. Cilj testiranja softvera je da se pokaže da program radi. Softver se iznosi na tržište bez primene sistema obezbeđenja kvaliteta. Nedostaju resursi, alati i adekvatno obučeni kadrovi. Ovaj tip organizacije odgovara SEI CMM Level 1, zrelosti softverske kompanije.	TMM Nivo 4 Faza Merenja i Upravljanja: Proces TS se meri i kvalitet (cena, efikasnost, efektivnost) ocenjuje. Inspekcije i revizije se primenjuju planski u svim fazama SDLC kao obavezna aktivnost u TS i kontroli kvaliteta. Softverski proizvod se testira radi ocene faktora kvaliteta kao što su pouzdanost, upotrebljivost i pogodnost za održavanje. Ažurira se baza podataka o test-primerima sa svih projekata radi ponovne upotrebe pri regresionom (ponovljenom) testiranju. Otkazi, greške, se evidentiraju u bazi podataka o otkazima, greškama i dodeljuje im se značaj (kritičnost). Nedostatak TS na ovom TMM nivou je i dalje nedovoljno primenjena preventivna aktivnost generisanja softverskih grešaka, slabo razvijena metrika kvaliteta TS kao i sredstva automatizacije TS.
TMM Nivo 2 Faza Definisanja: Testiranje softvera je odvojena od faze otklanjanja grešaka (debugging) i definisano je kao odvojena faza nakon kodiranja. Mada je planirana kao aktivnost, Testiranje softvera na Nivou 2, je definisano nakon faze kodiranja zbog nezrelosti samog procesa Testiranja softvera. Glavni cilj Testiranja softvera, na ovom nivou zrelosti (TMM), je da se pokaže da je softver zadovoljio specifikaciju. Primenjuju se osnovne tehnike i postupci. Mnogi problemi vezani za kvalitet softvera na ovom TMM nivou posledica su planiranja TS kasno u ciklusu razvoja softvera (SDLC). Dalje, greške (otkazi) softvera u ranim fazama prostiru se do poslednjih faza SDLC tj. ne otkrivaju se blagovremeno, odnosno onda kada se i generišu.	TMM Nivo 5 Faza Optimizacije, Prevencije grešaka i Kontrola kvaliteta: Nakon uspešne izgradnje infrastrukture kroz sazrevanje TMM od Nivoa 1 do 4, za koji se može reći da je TS definisan i kontrolisan, preko metrika kao što su troškovi, efikasnost, efektivnost sada se na TMM Nivo 5 pristupa finom podešavanju i stalnom unapređenju kvaliteta TS. Proces TS je kontrolisan statističkim postupcima uzorkovanja i merenja nivoa poverenja metrika kvaliteta TS kao što su troškovi, efikasnost, efektivnost. Uspostavljena je procedura za izbor i ocenu sredstava i alata za TS. Automatska sredstva TS se koriste u svim fazama testiranja softvera dizajnu test-primeri, izvršavanju testova, ponovnom izvršavanju, ažuriranju baze podataka o otkazima, greškama, alati za metriku, praćenje generisanja i analizu uzroka istih kao i sredstva održavanja tzv. "Testware".
TMM Nivo 3 Faza Integrisanja: TS nije više faza koja sledi fazu kodiranja, naprotiv, TS je integrisani deo SDLC. Organizacije koje su ovladale TMM Nivo 2, za razliku od TMM Nivoa 2, na Nivou 3 aktivnost TS se odvija i planira od početka SDLC tj. projektnih zahteva za softver pa do kraja najčešće V modela SDLC. Ciljevi i zadaci TS su utvrđeni na bazi zahteva klijenata i mogućih kupaca softvera i koriste se u fazi dizajna test primera i kriterijuma uspešnog odziva testa. Organizaciono je uspostavljena grupa za TS, a TS je profesionalni posao članova tima. Obuka kadra je fokusirana na oblast TS. Osnovna sredstva, alati za TS su u upotrebi. Iako organizacije na ovom TMM nivou znaju za značaj kontrole i obezbeđenja kvaliteta, ova funkcija nije formalno primenjena u SDLC. Program	

4. Pošto je razvoj softvera bez grešaka nemoguć, potrebno je za definisani nivo poverenja tj. prihvatljiv nivo grešaka odgovarajućeg nivoa kritičnosti za datu primenu softverskog proizvoda, tako da generičko rešenje IOPTS koje uključuje tehnike statističkog Planiranog Eksperimenta (engl. Design of Experiments) značajno smanjuje troškove, vreme, resurse u procesu testiranja softvera uz održavanje postavljenog nivoa poverenja u ostvareni nivo kvaliteta softverskog proizvoda tj. dovodi do optimizacije procesa testiranja softvera [2-5].
5. Primena Šest Sigma (engl Six Sigma) metodologije u generičkom rešenju IOPTS tj. modelu procesa testiranja softvera obezbeđuje da proces testiranja softvera bude stalno unapređivan na kontrolabilan i prediktabilan, a to znači stabilan način [2-5].



Slika 4 Generičko rešenje IOPTS [2]

4. ISTRAŽIVAČKE METODOLOGIJE I TEHNIKE PRIMENJENE U IOPTS

Testiranje je proces, i kao takvim, njime se mora upravljati. To, u najmanju ruku, znači, da se na nivou organizacije mora osmisliti i dokumentovati strategija testiranja. Procedure i koraci procesa testiranja se moraju definisati. Proces mora da bude isplaniran, tester obučeni, i proces mora imati merljive ciljeve koji se mogu pratiti [3,6]. Proces mora biti sposoban da evoluirati i da uvek bude u stanju usavršavanja.

Za potvrdu navedenih hipoteza u prethodnom odeljku analizirane i primenjene su sledeće metode istraživanja (prikazan šematski na Slici 4):

- Identifikacija, metodom analize publikovanih procesa testiranja softvera, adekvatnost i usaglašenost sa savremenim metodologijama razvoja softvera (SDLC Software Development Cycles, CMM, SPICE, VOCAL Test Methodology, IEEE standardi iz oblasti softverskog inženjerstva, vojni standardi npr. DoD-2167A, NASA D-4000 i sl.)
- Analizirane su specifičnosti razvoja softvera za razne oblasti primene (softverski proizvodi sa kritičnom misijom, ugrađeni softverski proizvodi u sistemima upravljanja u realnom vremenu, softverski proizvodi široke potrošnje, distribuirani softverski sistemi i sl.)

- Izvršena je analiza izvora grešaka po fazama razvoja softvera, klasifikaciju tipova grešaka, mere i postupke prevencije pravljenja softverskih grešaka
- Identifikovana je metrika za ocenu prednosti i mane t.j ‘valjanosti’ test metodologija kao što su [2-6]:
 - o White-Box (Structural) Testing Methodolgy (kriterijumi adekvatnosti testiranja na osnovu pokrivenosti izvršenih instrukcija programskog koda, putanja izvršavanja programa, grananja, logičkih uslova i sl., veličina test uzorka (test suit), verovatnoća otkrivanja grešaka, efikasnost testiranja i dr.
 - o Black-Box (Specification Based, Functional) Testing Methodology (Category Partition-ekvivalentne klase, Boundary Value Analysis- test primeri na bazi graničnih i specifičnih vrednosti, Cause-Effect Graph – uzročno posledični grafovi)
 - o Razvijena je Kombinovana Test Metodologija (Gray-Box) kao optimalna metodologiju testiranja.
- Primenjene su i ocenjene mogućnosti i efikasnost tehnika kao što su: kompjutersko modelovanje i simulacija, inspekcija, statička analiza, demonstracija i njihov značaj u fazama integracije, ispitivanja sistema i regresionog (ponovljenog) testiranja softvera.
- Analizirana je eksperimentalna metodologija u oblasti softverskog inženjerstva, planirani i izvedeni eksperimenti za ocenu ‘valjanosti’ predložene optimizirane metodologije testiranja softvera (combined-test methodology).
- Analizirana je Šest Sigma (Six Sigma) metodologija sa aspekta primene u softverskom inženjerstvu u cilju obezbeđenja kvaliteta softvera t.j. kontrolabilnosti, obzervabilnosti i predvidljivosti odnosno, stabilnosti koja insistira na konstantnom poboljšanju procesa testiranja softvera u cilju njegovog optimiziranja.

5. ZAKLJUČAK

Proces testiranja softvera može da bude veoma problematičan zbog nametnutih ograničenja u resursima i vremenu. Promišljeno planiranje strategije testiranja je ključno za kvalitetno upravljanje procesom razvoja i testiranja softvera. Moraju se uzeti u obzir kako ekonomski, tako i tehnički aspekti, posebno, rizici neotkrivanja grešaka koji za posledicu imaju sudske sporove.

Testiranje Softvera je skup proces, jer u proseku oko 50% ukupnog budžeta za razvoj softverskog proizvoda se troši na testiranje softvera dok je u nekim oblastima primene čak i preko 80% .

Veliki su gubici firmi usled grešaka u softveru **koji su uslovili razvoj oblasti krivične odgovornosti proizvođača softvera i zaštite kupaca zbog neadekvatnog kvaliteta softverskog proizvoda.** U ovom radu su identifikovani problemi i nedostaci postojećih procesa i tehnika testiranja softvera kroz postavljanje nekoliko hipoteza poboljšanja efikasnosti i efektivnosti procesa testiranja softvera u vidu generičkog rešenja Integralnog i Optimiziranog Procesa Testiranja Softvera. Generičko rešenje IOPTS, predstavlja pro-aktivni model sistematskog, planiranog, kontrolabilnog i prediktabilnog t.j. stabilnog procesa testiranja softvera koji će obezbediti efikasno, efektno i rano otkrivanje grešaka u procesu razvoja softvera kroz estimaciju optimalnog scenarija testiranja softvera u konkretnim uslovima razvoja softverskog proizvoda. Takvo rešenje smanjiće propagaciju softverskih grešaka kroz faze razvoja softvera t.j. životni vek softverskih grešaka, a što će dovesti do značajnog smanjenja troškova i vremena testiranja, odnosno razvoja softvera, uz

zadati nivo poverenja t.j. minimalni rizik neuspeha projekta razvoja softverskog proizvoda zahtevanog nivoa kvaliteta.

6. LITERATURA

- [1] Kaner C, "Liability for defective content" ACM SIGDOC'04, Memphis, TN, October 10–13, 2004.
- [2] Lj. Lazićj. *The Integrated and Optimized Software Testing Process*. PhD Thesis, School of Electrical Engineering, Belgrade, Serbia, 2007.
- [3] Lazić Lj., Mastorakis N. "OptimalSQM: Integrated and Optimized Software Quality Management", WSEAS TRANSACTIONS on INFORMATION SCIENCE and APPLICATIONS, Issue 10, Volume 6, pp 1636-1664, ISSN: 1790-0832, October 2009.
- [4] Lazić Lj., Popovic S., Mastorakis N. "A Simultaneous Application of Combinatorial Testing and Virtualization as a Method for Software Testing", WSEAS TRANSACTIONS on INFORMATION SCIENCE and APPLICATIONS, Issue 11, Volume 6, p p 1802-1813, ISSN: 1790-0832, November 2009.
- [5] Lazić Lj, Velašević D. "Applying simulation and design of experiments to the embedded software testing process". STVR, Volume 14, Issue 4, p257-282, John Wiley & Sons, Ltd., 2004.
- [6] Burnstein I. "*Practical Software Testing*", Springer-Verlag New York, Inc., 2003.
- [7] Brian Marick, How to Misuse Code Coverage, www.testing.com, personal web site.

SOFTWARE TESTING AS END USER SECURITY MEASURE

Abstract: Software Testing is established as a scientific discipline, as well as one of engineering, for many years, and even decades. To control efficiency, but also keep in mind the spent effort and resources for achieving this efficiency, it is necessary to apply quality techniques in choosing the elements of the product to be tested. Advantages and disadvantages of all test strategies and techniques will be displayed, and furthermore, a Integrated and Optimized Software Testing Process, in a broader sense, will be suggested as a user incident protection solution, from released software defects.

Key words: *software testing, errors prevention, optimization, risks, end user protection*