ELEKTROTEHNIČKI FAKULTET UNIVERZITETA U BEOGRADU

ZNAČAJ I ULOGA QA TESTIRANJA U RAZVOJU VEB APLIKACIJA

– Master rad –

Kandidat:

Marko Đerić 2012/3390

Mentor:

doc. dr ZoranČiča

Beograd, Septembar 2015.

SADRŽAJ

1. UVOD

Testiranje je veoma važna aktivnost u razvoju softvera koja je posebno dobila na značaju kada je vrednost softvera počela da raste. Greške u softveru koji je vredan nekoliko miliona dolara mogu prouzrokovati ogromne novčane štete, tako da se moraju otkloniti što je moguće ranije. Zbog toga se u poslednje vreme ne štedi na aktivnostima testiranja. Postepeno testiranje postaje aktivnost koja je važna kao i sam razvoj softvera tako da je potrebno ozbiljno prići ovoj temi.

Neki softverski timovi ne shvataju proces testiranja i složenost uloga koje postoje u procesu testiranja. Nekada se pogrešno smatra da su testeri manje-više sporedno osoblje koje čeka da se softverski sistem napravi, dobije sistem i onda rade sa njim šta god im je volja kako bi pronašli sve probleme. Međutim, proces testiranja je znatno složeniji, sa ozbiljnim planovima, strukturom test tima i metodologijom rada.

Cilj ovog master rada je da se studentima približi testiranje softvera kao jedno novo i sve traženije zanimanje u polju razvoja softvera.

U okviru ovog rada biće objašnjeno kada se javila uopšte potreba za testiranjem, kao i kako se vrši testiranje softvera u različitim modelima proizvodnje softvera. Kroz rad biće obrađene različite vrste i metode testiranja.

U okviru ovog rada biće dat prikaz testiranja jedne veb aplikacije koja će služiti kao jedan dobar primer testiranja softvera.

Ostatak rada organizovan je na sledeći način. U drugom poglavlju opisan je pojam testiranja kao i istorijat softverskih grešaka koje su i uzrok pojave testiranja softvera. U trećem poglavlju opisani su modeli testiranja softvera i uloga testiranja unutar njih. Četvrto poglavlje sastoji se od metoda i vrsta testiranja. Peto poglavlje je gore pomenuti primer, i na kraju u poglavlju šest nalazi se zaključak.

2. ISTORIJAT TESTIRANJA

Mnogo je primera softverskih neuspeha kroz istoriju. U daljem tekstu neki od njih će i biti navedeni. Upravo ti softverski neuspesi kao i veliki finansijski gubici nekih kompanija doveli su do potrebe za testiranjem softvera. Uvođenjem testiranja u izradu softvera povećali su se troškovi proizvodnje, ali su se kompanije koje proizvode softver na neki način osigurale od velikih grešaka koje bi ih kasnije mnogo koštale.

U tabeli 1 prikazana je cena greške u zavisnosti u kojoj fazi izrade softvera je nađena.

Faza u kojoj je greska pronadjena	***Cena greške***
Zahtevi	$1
Dizajn	$10
Kodiranje	$100
Programsko testiranje	$1000
Sistemsko testiranje	$10000
Testiranje sa aspekta korisnika	$100000
Gotov program	$1000000

Tabela 1. – Cena greške

Upravo ovi finansijski gubici su i najveći razlog uvođenja testiranja u proces proizvodnje softvera.

Neki primeri softverskih grešaka kroz istoriju[1] :

- Prilikom lansiranja ARIANE 5 rakete od strane Evropske svemirske agencije juna 1996 došlo je do otkaza posle samo 37,5 sekundi. Greška u softveru je prouzrokovala da raketa ne poleti vertikalno i došlo je do nepredvidivog kretanja rakete.
- Prilikom lansiranja Online softvera za povrat poreza u Velikoj Britaniji javila se greška da su korisnici ponekad mogli da vide količinu novca koja je bila vraćena prethodnom korisniku bez obzira da li su korisnici radili povrat poreza na istoj lokaciji.

- U novembru 2005, objavljene su informacije o 10 najtraženijih kriminalaca u Velikoj Britaniji na veb stranici. Vesti o ovoj veb strani su objavljene u medijima što je imalo za posledicuda je veliki broj ljudi pokušao da je poseti i došlo je do preopterećenja. Pokazalo se da performanse veb strane nisu zadovoljile standarde koji su se očekivali od nje i sajt je morao da bude ugašen.
- Na jednoj poznatoj veb stranici za onlajn kupovinu knjiga prilikom naručivanja negativnog broja knjiga koje poručilac želi da kupi poručiocu bi se uplaćivala ta vrednost na njegov račun kao povraćaj novca.
- Blokiranje aerodroma u Los Anđelesu. Usled greške u softveru, pogrešne informacije su bile poslate u mrežu carine Sjedinjenih Američkih Država, što je uslovilo da 17.000 aviona budu osam sati zarobljeni na aerodromu 2007 godine.
- 2002. studija Nacionalnog instituta za standarde i tehnologiju (NIST) je pokazala da softverske greške koštaju američku ekonomiju 59,5 milijardi dolara godišnje. Studija govori da bi se 22,2 milijarde dolara mogle uštedeti boljim testiranjem.

Ovo su samo neki od primera zaobilaženja testiranja softvera pre njegovog puštanja u rad.

2.1. Terminologija

U ovomradu će senaići na dosta termina specifičnih za oblast testiranja. Neki termini koji su bitni za dalje praćenje rada su:

Proizvod rada - bilo koji dokument, programski kod, plan, izveštaj ili test koji naprave članovi projektnog tima tokom rada na projektu. Proizvodi rada su predmet testiranja.

Testiranje - proces evaluacije softvera i svih pratećih proizvoda rada koji se kreiraju tokom razvoja projekta kako bi se utvrdilo da li su u skladu sa očekivanim zahtevima, da li odgovaraju svojoj svrsi, kao i da li postoje neki defekti u njima.

Test slučaj - stanje opisano akcijom koju treba izvršiti, podacima koje treba poslati sistemu, kao i očekivanim odgovorom od strane softvera. Umesto ovog termina koriste se i termini test primer ili slučaj testiranja.

Test scenario - sekvenca akcija koje treba izvršiti tokom testiranja sistema kako bi se proverili jedan ili više test slučaja. Često se umesto termina test scenario koristi termin test procedura ili test skript.

Defekt - greška u softveru koja uslovljava da se softver ne ponaša kao što je očekivano. Defekti u programskom kodu često se nazivaju bagovi .

Pronalaženje defekata - proces analize softvera kako bi se pronašlo šta uzrokuje pronađenu programsku grešku. Često se naziva debagovanje**.**Pošto se programska greška pronađe i ispravi, sistem postaje stabilniji.

Programski kod - tekst napisan na nekom jeziku koji se može ili izvršiti direktno od strane računara ili se može prevesti u oblik razumljiv računaru.

Dizajn testa - proces analize i specifikacije test slučajeva i scenarija koji će biti korišćeni tokom testiranja. U dizajn testa pored funkcionalnih test slučajeva mogu da se uključe dizajn testa sigurnosti, opterećenja, platforme, lakoće korišćenja i slično.

2.2. Pojam testiranja

U ovoj sekciji će biti objašnjeno šta je u stvari testiranje i zašto je ono uopšte važno u softveru. Greške koje se prave tokom rada su normalna i svakodnevna stvar i nisu direktno vezane za izradu softvera. Greške se dešavaju u svim oblastima rada i uzrokovane su kako ljudskim, tako i mašinskim faktorima. Imajući u vidu da su greške realna pojava, u svim oblastima rada se uvodi sistem kontrole kvaliteta kojim se greške identifikuju i otklanjaju pre nego što izazovu veće probleme u radu.

Konkretno, u izradi softvera testiranje predstavlja pokušaj da se pronađu greške u softveru koji je napravljen. Softver je implementiran prema korisničkim zahtevima kojima se rešava neki realni problem ili se kreira neka korisna funkcionalnost koja predstavlja nešto što je potrebno krajnjim korisnicima. Kada se implementira, softver može u većoj ili manjoj meri da odgovara originalnim zahtevima prema kojima je i napravljen. Svako ponašanje softvera koje se ne slaže sa originalnim zahtevima predstavlja grešku koju je potrebno identifikovati i otkloniti. U užem smislu, testiranje predstavlja upravo proveru da li je određeni softver u potpunosti implementiran prema originalnim korisničkim zahtevima.

U širem smislu testiranje predstavlja sistem kontrole kvaliteta (QA - Quality Assurance) kojim se ne proverava samo softver već i sve njegove prateće komponente i karakteristike.

3. MODELI TESTIRANJA SOFTVERA I TESTIRANJE UNUTAR NJIH

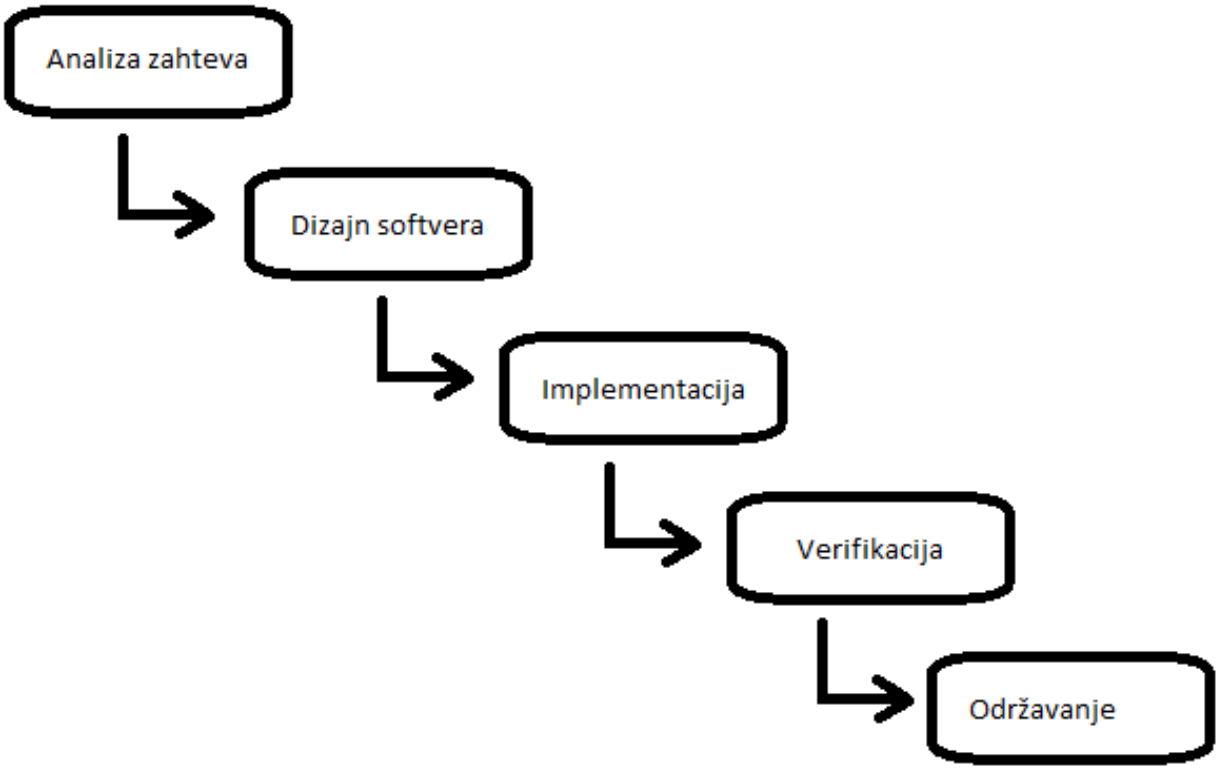
Razvoj softvera uključuje veliki broj raznorodnih aktivnosti kao što su analiza zahteva koji bi trebalo da budu implementirani, dizajn rešenja, programiranje, testiranje, upravljanje projektom i tako dalje. Ovaj skup aktivnosti mora se uklopiti u jedinstven proces razvoja kojim se projekat uspešno vodi od trenutka kada su dobijeni zahtevi do trenutka kada se projekat predaje krajnjim korisnicima. Način na koji su organizovane ove aktivnosti naziva se model životnog ciklusa softvera ili model razvoja softvera.

Postoji veliki broj različitih modela razvoja softvera kojima se uklapaju pomenute aktivnosti. Modeli razvoja softvera mogu se podeliti na sledeće grupe:

1. Fazni modeli - modeli u kojima se aktivnosti razvoja dele po grupama koje se rade sekvencijalno po fazama.
2. Iterativni modeli - u kojima se projekat deli na manje periode (iteracije) u kojima se vrše sve aktivnosti u razvoju.
3. Agilne metode - slične iterativnim ali se na manje formalan način kombinuju aktivnosti razvoja softvera.

Vodopad - Model vodopada je jedan od prvih modela razvoja softvera koji je zasnovan na sekvencijalnom pristupu razvoja. Osnovna ideja sekvencijalnog pristupa je podela procesa razvoja na pojedine faze koje predstavljaju skupove srodnih aktivnosti razvoja softvera. Faza traje sve dok se ne završe sve aktivnosti u okviru nje i ne može se preći u sledeću fazu dok prethodna nije završena kao što je prikazano na slici 1.

Ideja modela vodopada je sledeća - da bismo razvili neki softver, prvo moramo da razumemo zahteve. Rezultat ove faze je specifikacija zahteva koju korisnici sistema mogu da pročitaju i potvrde da je to ono što im treba. Kada su zahtevi shvaćeni i potvrđeni, projektni tim mora da definiše kako će ih implementirati (faza dizajna). Pošto je definisano kako će se sistem implementirati prelazi se na programiranje. Kada je sistem završen, potrebno ga je testirati kako bi se otklonile greške i onda je spreman za prodaju korisnicima.



Slika 1. – Model vodopada

- Analiza zahteva - faza u kojoj se sakupljaju zahtevi od krajnjih korisnika, analizira šta je potrebno raditi, kreiraju ugovori kojima se definiše šta će biti urađeno i potvrđuju zahtevi koji će biti implementirani

- Dizajn softvera - faza gde se detaljnije analiziraju zahtevi prikupljeni tokom analize, specificira kako će se implementirati sofitver, kreira tehnička specifikacija i dizajn softvera. Dizajn softvera predstavlja konkretan plan kako će biti implementiran sistem od generalne arhitekture softvera do detaljnog opisa implementacije pojedinih komponenti i algoritama. Na kraju ove faze je poznato kako će se sistem implementirati.

- Implementacija - faza u kojoj se projektovani softver implementira u određenom programskom jeziku i platformi. Na kraju ove faze softver je završen i spreman za upotrebu.

- Verifikacija - faza u kojoj se planira testiranje, testira se sistem koji je implementiran u prethodnoj fazi, prijavljuju i otklanjaju problemi nađeni u softveru. Na kraju ove faze softver je testiran i spreman na predaju krajnjim korisnicima.

- Održavanje - tokom faze održavanja softver je predat krajnjim korisnicima i vrše se eventualne dorade u skladu sa izmenama traženim od korisnika. Ova faza traje sve dok korisnici upotrebljavaju softver.

Fazna priroda modela vodopada zasniva se na „zdravorazumskoj“ podeli posla. Koji god posao je potrebno izvršti logično je da se najpre analizira šta će biti urađeno, potom da se odredi kako će to biti urađeno, a nakon izvršenja posla potrebno je proveriti da li je posao stvarno urađen kako treba. Model vodopada je samo primena ovog načina rada u softverskim projektima.

Na žalost, ovakav idealan slučaj sekvencijalnog razvoja u praksi nije moguć zato što se često neplanski projektni tim vraća u prethodne faze. Često se dešava da se tokom kasnijih faza projekta pronalaze propusti i nepredviđene stvari nasleđene iz prethodnih faza, koje uzrokuju povratke u prethodne faze. Ono što bi trebalo da bude najveća prednost modela vodopada (činjenica da se u svakoj fazi zna koji članovi tima su potrebni) u povratnim tokovima predstavlja najveću slabost modela. Ako se pronađe problem u nekoj od kasnijih faza projekta, on često zahteva nekog člana tima koji više nije na projektu, pošto je on otišao sa projekta čim više nije bio potreban.

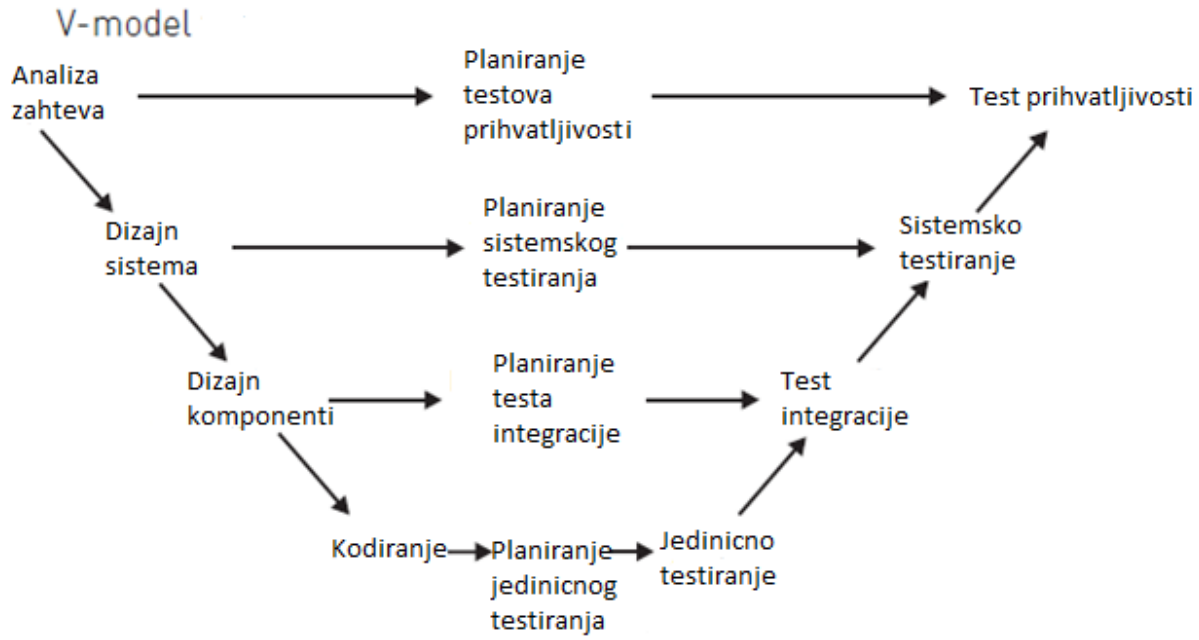
PRIMER - *Kreirana je aplikacija u kojoj korisnici mogu da obrađuju podatke o firmama i proizvodima koje firme nude. Jedna od funkcionalnosti je dodavanje novog proizvoda. Zahtev je da svaki proizvod mora da ima dodeljenu kompaniju koja ga proizvodi - ovo je korisnički zahtev definisan tokom faze analize zahteva. Tokom faze dizajna softvera specificirano je da će korisnik kreirati novi proizvod u dijalogu u kome će se nalaziti naziv proizvoda i lista kompanija iz koje će biti odabrana ona kompanija koja proizvodi proizvod koji se trenutno kreira. Tokom faze implementacije je ova funkcionalnost implementirana i tokom verifikacije je potvrđeno da ona radi po specifikaciji i zahtevima. Međutim, kada je aplikacija predata krajnjim korisnicima, primećen je problem. Sistem radi sa ogromnim brojem kompanija i kada su korisnici počeli da rade sa stvarnim podacima, prilikom dodavanja novih proizvoda korisnik je morao da pronađe kompaniju u listi od više hiljada stavki što je bilo nemoguće. Čak ni brze izmene tipa dodavanja filtera u listu nisu pomagale, korisnici su jedva radili, a performanse su bile očajne usled učitavanja ogromnog broja podataka u listu.*

Kao rezultat, ova funkcionalnost je vraćena u fazu dizajna gde je napravljen novi izgled forme u kojoj korisnik prvo pretražuje kompanije koje se dodaju u tabelu, sa straničenjem (paginacijom), sortiranjem i biranjem jedne od kompanija. U ovakvom povratku nije samo problem u kašnjenju sa isporukom funkcionalnosti koja je naknadno implementirana, nego i u probijanju budžeta projekta. Za inicijalni dizajn sa jednostavnom listom bilo je predviđeno pola dana za implementaciju funkcionalnosti, međutim novi dizajn koji uključuje mnogo složeniji proces zahteva dva dana za implementaciju. Ovo dodatno vreme uopšte nije predviđeno projektom i predstavlja gubitak na projektu. Na ovaj način, neočekivani povratni tokovi uništavaju projekte u modelu vodopada.

Testiranje u modelu vodopada - Testiranje u projektima koji se rade modelom vodopada se vrši isključivo tokom faze verifikacije. Kada se kompletira programski kod, softver se predaje test timu koji ga testira i verifikuje da programski kod ispravno radi.Testiranje u ovoj fazi obuhvata analizu funkcionalnih zahteva, definisanje test planova, testiranje softvera i prijavljivanje problema. U ovoj fazi angažovan je kompletan test tim, ali i deo razvojnog tima pošto neko mora da bude angažovan radi ispravljanja problema nađenih tokom testiranja.

Kada počne faza verifikacije, test menadžer napravi plan rada. Test analitičar, inženjer i tester analiziraju zahteve i upoznaju se sa aplikacijom. Potom, test analitičar završi dizajn testova po kojima tester i test inženjer testiraju sistem (paralelno sa ovom aktivnošću potreban je i jedan ili više programera koji će rešavati probleme). Na kraju fazeverifikacije, test menadžer podnosi izveštaj i potvrđuje da u sistemu nema više problema.

V-model - je fazni model koji predstavlja modifikaciju standardnog modela vodopada u kome se više pažnje posvećuje testiranju. U V-modelu, faza testiranja se ne posmatra kao jedna celina nego se deli na podfaze u kojima svaka podfaza testiranja odgovara jednoj fazi razvoja. Na ovaj način faze razvoja sistema se mogu prikazati u obliku slova V, kao što je prikazano na slici2.



Slika 2. – V-model

V-model kao i vodopad ima sve sekvencijalne faze analize, dizajna, programiranja i testiranja gde se u sledeću fazu prelazi samo ako je završena prethodna, ali uz dve značajne izmene. Svaka faza razvoja ima odgovarajuću fazu testiranja kojom se ta faza validira. Posle svake faze razvoja, pre prelaska u sledeću fazu, planira se kako će se izvršiti verifikacija trenutne faze (iako se odgovarajuća faza verifikacije neće izvršiti do kraja projekta). Preduslov za prelazak u sledeću fazu razvoja je da je definisanona koji način ćebiti testirani rezultati prethodne faze razvoja.

Osnovna ideja u V-modelu je da iako nije moguće stvarno verifikovati određenu fazu, može se bar isplanirati kako će se ta verifikacija izvršiti. Planiranjem verifikacije mogu se otkriti neki problemi u samoj fazi razvoja.Na primer, ako je definisana lista zahteva i ako se počne sa planiranjem kako će ti zahtevi biti testirani, za neke od zahteva se može zaključiti da su nejasni ili dvosmisleni tako da ih je nemoguće testirati. U tom slučaju, projektni tim se odmah vraća u fazu analize zahteva, kako bi odmah razjasnio taj zahtev. Na ovaj način se sprečava problem iz modela vodopada gde neko tek po implementaciji samog sistema zaključuje da je originalni zahtev nejasan pa se mora vratiti na početak. U V-modelu je veća verovatnoća da će rezultati pojedinih faza biti validni i spremniji za sledeću fazu. U V-modelu postoje pravila kojima se definišu preduslovi za prelazak u sledeću fazu:

- Iz faze analize zahteva može se preći u fazu dizajna samo ako su analizirani zahtevi i ako je definisano kako će se ti zahtevi testirati tokom testa prihvatljivosti.

- U fazu dizajna arhitekture može se preći ako je završena faza sistemskog dizajna i definisano kako će se testirati kompletan sistem
- U fazu dizajna pojedinih modula može se preći ako je dizajnirana arhitektura i definisano kako će se testirati komponente tokom integracije
- U fazu kodiranja može se preći ako su dizajnirani moduli koji će se kodirati i ako je definisano kako će se ti moduli testirati.

Druga značajna izmena je definisanje više nivoa testiranja kojima se proveravaju različiti delovi sistema. Nivoi testiranja po V-modelu su:

- Jedinično testiranje kojim se testiraju pojedini delovi sistema (moduli, komponente, forme).
- Integraciono testiranje kojim se testiraju komunikacija, povezivanje i tokovi među modulima.
- Sistemsko testiranje kojim se testira system u celini.
- Test prihvatljivosti kojim krajnji korisnici potvrđuju da aplikacija radi upravo ono što im treba.

U skladu sa ovim aktivnostima modifikuje se i proces testiranja koji se uklapa u proces razvoja softvera.

Proces testiranja u V-modelu - Kao što je već pokazano, V-model izvlači aktivnosti planiranja, specifikacije i dizajna testova u što je moguće ranije trenutke projekta. Ove aktivnosti nisu više deo poslednje faze nego se ubacuju kaosastavni elementi prethodnih faza. Na ovaj način značajno je promenjen proces testiranja - test tim više ne čeka kraj projekta kako bi dobio i testirao gotovu aplikaciju nego biva uključen od početka projekta u svim fazama razvoja.

Tokom faze dizajna sistema, dok analitičari i dizajneri kreiraju dizajn sistema i pišu specifikacije, test analitičari i inženjeri prave dizajn sistemskih, integracionih i jediničnih testova. Kao i u fazi analize, dizajn ovih testova omogućava analitičarima i dizajnerima sistema da provere svoju specifikaciju kako bi se smanjio broj grešaka koji se prenosi u fazu implementacije. Tokom faze implementacije vrši se jedinično testiranje komponenti čim se one implementiraju na osnovu dizajna jediničnih testova.

U fazi verifikacije, kada se završi implementacija sistema, preostaje da se ispita da li je integracija komponenti moguća i da se testira ceo sistem. Svi problemi koji se primete ispravljaju se tokom ove faze. U slučaju da su aktivnosti testiranja, koje su prebačene u ranije faze, dale rezultate time što su poboljšale zahteve i specifikaciju, ne bi trebalo da dođe do prevelikih skokova u prethodne faze i sve ispravke bi trebalo da se završe u fazi verifikacije.

Sastanovišta testiranja veliki napredak je u činjenici da su test aktivnosti uključene od početka rada na projektu i da je testiranje uslov za nastavak projekta. Testiranje u ranim fazama omogućava da se projekat uspešnije privede kraju pošto se testira što je pre moguce.

PRIMER - *U primeru u prethodnoj sekciji datje primer funkcionalnosti kreiranja proizvoda gde je pokazano kako je povratni tok stvorio problem u projektu. U tom primeru je po isporuci klijentima primećeno da u sistemu ima nekoliko hiljada kompanija i da se prilikom kreiranja novog proizvoda kompanija ne može izabrati iz liste. U V-modelu bi se, kao i u modelu vodopada, tokom faze analize zahteva definisalo da je tokom kreiranja novog proizvoda potrebno odabrati kompaniju koja ga proizvodi iz liste. Međutim, umesto da se direktno pređe u sledeću fazu projekta, počele bi*

pripreme i definisanje testa prihvatljivosti. Tokom priprema za test prihvatIjivosti, testeri bi ispitali koliko može da bude kompanija u sistemu ili bi posmatrali kakotrenutno korisnici rade sa aplikacijom u slučaju da se postojeći sistem zamenjuje novim, i videli bi da će biti nekoliko hiljada kompanija. Na ovaj način bi zaustavili dalji nastavak projekta i primorali projektni tim da promeni specifikaciju u skladu sa novim pogledom na zahteve. U V-modelu se ovakvim pristupom onemogućava da zahtevi ili dizajn koji nisu validni nastave ka fazi implementacije.

Agilne metode- ovi modeli razvoja softvera predstavljaju novi pristup razvoju softvera, koji se bori protiv formalizacije i birokratizacije procesa razvoja softvera krutim modelima razvoja, u kojima se precizno moraju definisati sve potrebne faze i aktivnosti kako bi se kompletirao projekat [4]. Agilni model je pokušaj da se ukloni formalizacija procesa razvoja softvera, minimizuju prateće aktivnosti koje nisu direktno vezane za razvoj softvera i tako dobije maksimalna efikasnost.

Osnovna karakteristika agilnog razvoja softvera je iterativno/spiralni razvoj u kome se projekat, umesto na dugačke faze ili verzije kao u prethodnim modelima, deli na kratke iteracije (trajanja od dve do tri nedelje), u okviru kojih se vrše sve potrebne aktivnosti analize, dizajna, kodiranja i testiranja kao i u ostalim modelima.

U okviru svake iteracije, koja može trajati dan, nedelju ili par nedelja, ponavljaju se aktivnosti analize, dizajna, implementacije i testiranja gde je fokus samo na funkcionalnostima koje se implementiraju u trenutnoj iteraciji. Po završetku jedne iteracije kreće se u novu po istom šablonu.Ideja agilnih procesa je napuštanje formalizacije nastale modelom vodopada i fokusiranje na ljude, efikasnost i komunikaciju, a ne na procese, dokumentaciju i planove kao u formalnim metodama.

Skram je agilna tehnika, koja je postala popularan model razvoja softvera poslednjih godina, zato što kombinuje efikasnost agilnih metoda, ali donekle zadržava kontrolisanost formalnih metoda. U Skram modelu postoji nekoliko uloga u timu:

- Skram master - osoba koja poznaje Skram proces i stara se da se poštuju procesi i pravila skram procesa.
- Vlasnik proizvoda - osoba ili grupa osoba koje odlučuju o zahtevima - uobičajeno je da to budu korisnici sistema ili sponzori projekta.
- Projektni tim - programeri, testeri, analitičari i ostali članovi tima koji konkretno rade na projektu.

Ideja Skram procesa je da vlasnik proizvoda definiše listu zahteva, projektni tim ih implementira, dok Skram master kontroliše da se sve obavlja po pravilima rada. Skram model razvoja softvera podeljen je na tri osnovne faze:

- Priprema - faza u kojoj se priprema projekat za implementaciju, analiziraju zahtevi i kreira inicijalni skup zahteva koji će biti implementirani (engl. Product backlog).
- Igra - ovo je ključna faza u kojoj se implementira sistem. Ovo je klasična iterativna faza koja se sastoji od onoliko iteracija koliko je potrebno da se implementiraju sve funkcionalnosti. U svakoj iteraciji se vrše aktivnosti koje se odnose samo na funkcionalnosti koje se trenutno implementiraju.
- Zatvaranje - faza u kojoj se projekat privodi kraju, završava dokumentacija i predaje projekat krajnjim korisnicima.

Osnova Skram modela je faza igre u kojoj se projekat razvija kroz skup kratkih iteracija (ili „sprintova“ u Skram terminologiji). Preduslov za početak igre je da je spremna lista inicijalnih zahteva (takozvani *produkt beklog* u Skram terminologiji). Kada *produkt beklog* ima dovoljno zahteva za početak razvoja počinje se sa prvom iteracijom u fazi.

U svakom sprintu se bira skup zahteva iz produkt bekloga koji će biti implementiran. Odabir zahteva se vrši na osnovu prioriteta zahteva koje određuje vlasnik proizvoda, ali tim određuje koliko od tih zahteva će se implementirati u novom sprintu. Ovaj podskup zahteva naziva se sprint beklog i predstavlja specifikaciju poslova koji će se raditi u trenutnom sprintu. Dok se rade zahtevi za trenutni sprint, paralelno se modifikuju ili dodaju novi zahtevi u produkt beklog za buduće iteracije/sprintove.

Po završetku sprinta svi zahtevi koji su bili planirani za implementaciju moraju biti završeni i spremni za predaju korisnicima. Sprintovi u Skramu traju od jedne nedelje do mesec dana i uvek imaju fiksno trajanje. Ideja je da svi članovi tima imaju fokus na datumu kada treba da završe poslove u sprintu. Poslovi su završeni ako su analizirani, implementirani, testirani i spremni za predaju korisnicima. Na kraju svakog sprinta kao rezultat se dobija skup funkcionalnosti koje se mogu predati korisnicima.Tokom sprinta svakodnevno se održavaju sastanci timova na kojima se raspravlja o tome šta je urađeno tokom jučerasnjeg dana i šta će se raditi sledećeg dana. Na ovaj način se svakodnevno utvrđuje da li je bilo problema i planira se šta će se dalje raditi.

Proces testiranja u agilnim modelima - Sa gledišta testiranja, test aktivnosti se vrše kontinualno, čime se dobija stabilniji softver pošto nema nepotrebnog odlaganja testiranja. Testeri su tesno integrisani u tim i zajedno sa ostatkom tima planiraju i implementiraju testove u iteracijama. Ne postoji trenutak kada se počinje sa testiranjem pošto se analiza zahteva za testiranje i testiranje aplikacije vrše paralelno sa razvojem i primenjuje na svaki zahtev ili slučaj korišćenja. Na ovaj način testiranje predstavlja integrisani deo razvoja softvera, a ne samo krajnju aktivnost.

U agilnim metodama testiranje je istovremeno i preventivno i evaluaciono. Na početku svake iteracije testeri evaluiraju zahteve pre nego što se počne sa implementacijom kako bi se obezbedilo da se samo validan zahtev preda timu na implementaciju. Dok se zahtevi implementiraju i pretvaraju u softver, testeri u paraleli kreiraju testove kojima će biti provereno ono što je napravljeno na osnovu zahteva. Na ovaj način se vrši i preventivno testiranje pošto se testovi kreiraju i pre samog programskog koda.

U iterativnom procesu razvoja potrebno je napraviti strategiju rešavanja problema. Tokom testiranja pronalaze se programske greške, koje je potrebno rešiti, što zahteva određeno programersko vreme koje se troši na uštrb vremena planiranog za razvoj funkcionalnosti. U slučaju da je programski kod napravljen tokom iteracije suviše nestabilan, neće biti dovoljno vremena da se završe sve planirane funkcionalnosti do roka. U ostalim modelima to se rešava tako što se probija rok i dolazi do kašnjenja ali u iterativnim modelima nema probijanja roka - iteracija se završava posle određenog vremena bez obzira na to šta je uradeno . Pošto je kraj iteracije fiksan bez obzira na implementirane funkcionalnosti i probleme koji se dese potrebno je odlučiti šta treba raditi sa greškama koje se nađu. Neke od mogućih strategija su:

- Ispravljanje grešaka u okviru trenutnog sprinta - ukoliko su nađene greške koje se mogu rešiti u razumnom roku, vreme za rešavanje grešaka uklapa se u vreme koje će se trošiti u okviru iteracije/sprinta, one će se rešiti u okviru istog sprinta. Po rešavanju grešaka funkcionalnosti se ponovo testiraju i isporučuju na kraju sprinta.
- Kreiranje scenarija za sledeću iteraciju - greške koje se pronađu dokumentuju se kao scenariji koji izazivaju probleme i unose u glavnu listu zahteva. U trenutnoj iteraciji

isporučuje se funkcionalnost koja radi samo sa ispravnim scenarijima dok se scenariji koji prave probleme gledaju kao novi zadaci koji će biti implementirani u sledećem sprintu.

- Obaranje sprinta - u slučaju da nije moguce rešiti sve greške u okviru sprinta i kompletirati planirane funkcionalnosti, sprint se proglašava za neuspešan i svi zahtevi se vraćaju u glavni log. U novom sprintu se iz početka planira šta će biti implementirano.

Kao što se vidi od strategije rešavanja zavisi kako će teći projekat. Ono što je bitno kod agilnih metoda je da ne postoje formalni planovi kao u ostalim metodama. U agilnim metodama se prihvata činjenica da nije moguće precizno predvideti kako će se projekat realizovati i da su moguće razne promene u toku rada. Jedna od nepredviđenih promena je upravo preveliki broj pronađenih grešaka koji bi oborio ceo projekat koji je napravljen sa fiksnim rokovima. U agilnim metodama nema fiksnih krajnjih rokova pošto se ne zna koliko iteracija će biti do kraja niti koliko iteracija će propasti a koliko će biti uspešno urađeno. Ceo agilni proces se modifikuje i prilagođava stvarima koje će se desiti na projektu.

4. VRSTE I METODE TESTIRANJA

U ovom poglavlju biće opisane vrste i metode testiranja. Vrsta testiranja testeru govori šta će se testirati, dok mu metoda govori kako će to uraditi.

4.1. Vrste testiranja

Zavisno od toga na koje se rizike i karakteristike proizvoda test tim mora fokusirati, može se definisati više vrsta testiranja.

Funkcionalno testiranje – gde se proverava da li su funkcionalni zahtevi ispravno implementirani u sistemu.Osnovno testiranje softvera predstavlja funkcionalno testiranje. Softver se implementira prema zahtevima korisnikakojima se definišu funkcionalnosti koje suim potrebne. Ako te funkcionalnosti ne rade, korisnici nemaju neku korist od softvera.

Funkcionalno testiranje se vrši tako što se pregledom zahteva prolazi kroz razne scenarije korišćenja sistema i utvrđuje da li ima nekih propusta. Dva osnovna cilja ovog testiranja su:

- Provera da li sistem radi sve što je predviđeno zahtevima.
- Provera da sistem ne radi ništa što nije predviđeno zahtevima.

Testiranje opterećenja – predstavlja proveru kako sistem radi u slučaju kada se koristi u realnim situacijama. Postoje dve vrste testiranja opterećenja sistema:

- Test performansi - Kojim se ispituje rad sistema pod normalnim opterećenjem. Služi da se ispita vremenska dimenzija sistema, to jest da li je vreme odziva sistema, koje mu je potrebno da vrati odgovor na neku korisničku akciju dovoljno dobro.
- Test naprezanja - Kojim se ispituje ponašanje sistema u ekstremnim uslovima. To su uglavnom testovi koji se rade sa ogromnim brojem korisnika ili sa ogromnim brojem podataka, gde se gleda ponašanje sistema u takvim uslovima.

Testiranje sigurnosti -Pošto se danas softver pravi za sisteme u kojima se radi sa važnim privatnim i finansijskim podacima, postoji sve veća potreba da se proveri da li je sistem dovoljno siguran i otporan na napade.

Testiranje konfiguracije - Predstavlja skup svih softverskih i hardverskih komponenti, parametara sistema koji predstavljaju okruženje u kome softverski sistem radi, kao i dokumentacije koja ih opisuje.

Testiranje konfiguracije u užem smislu predstavlja proveru da li komponente mogu da rade zajedno i da li korektno rade u okruženju u kom se nalaze.

U širem smislu, u konfiguraciju se uključuju i verzije dokumentacije koje opisuju aplikaciju, zahtevi koji su implementirani o pojedinim verzijama softvera, greške koje su pronađene ili rešene.

Testiranje korisničkog interfejsa - Korisnički interfejs je izuzetno važna komponenta svakog softverskog sistema tako da mu se mora posvetiti posebna pažnja. Veoma često se naprave izuzetno dobre aplikacije sa složenom logikom koju implementirajuza kratko vreme ili rešavaju bitne probleme, ali ih malo ljudi koristi upravo zato što su jako komplikovane za korišćenje. Čest slučaj je da od dve konkurentske aplikacije korisnik bira onu koja ima lepši interfejs i lakše se koristi, a ne onu koja brže radi i ima više funkcionalnosti. Upravo su ovo razlozi zbog kojih se velika pažnja usmerava upravo ka testiranju korisničkog interfejsa.

Jedinično testiranje -Predstavlja testiranje izolovanih celina u sistemu [2]. Uslov je da se komponenta koja se testira može posmatrati kao nezavisna celina koja se može izvući iz konteksta sistema i testirati izolovano od ostalih komponenti. Prednost ovakvog načina testiranja je što se greške mogu lokalizovati i ispraviti u elementarnim komponentama kako bi se obezbedilo da ispravna komponenta uđe u sistem. Kada se vrši jedinično testiranje, sistem se sastoji iz lanca povezanih komponenti i testrira se svaka komponenta posebno.

Integraciono testiranje – se vrši kao nastavak jediničnog testiranja. Kada su napravljene komponente i testirane svaka ponaosob potrebo ih je i testirati u radu sa ostalim komponentama i proveriti da li rade bez grešaka.

Sistemsko testiranje – Predstavlja testiranje sistema koji je potpuno integrisan [3]. Iako sistemsko testiranje može da liči na posednji korak integracionog testiranja, ipak postoji značajna razlika. U integracionom testiranju se uvek testiraju direktno povezane komponente i traži se greška u komunikaciji i interfejsima, dok se kod sistemskog testiranja vrši testiranje zajedničkog rada komponenti koje nemaju direktne veze jedna sa drugom i ne komuniciraju direktno, ali imaju neku zavisnost po podacima koje koriste.

4.2. Metode testiranja

Metode testiranja predstavljaju načine kojima će tim testirati sistem. Na primer, potrebno je definisati da li će se testirati ručno ili će se pisati skripta koja testira sistem, da li će se testirati samo nove funkcionalnosti, ili će se ponavljati testovi na svim prethodno urađenim funkcionalnostima. Metoda testiranja predstavlja strategiju testa koja govori test timu na koji način će vršiti testiranje.

Ručno testiranje – je često osnovna metoda testiranja i predstavlja metodu kojom testeri bez nekog alata prolaze kroz sistem i testiraju funkcionalnosti. Ručno testiranje se može vršiti ili po slobodnim scenarijima ili po unapred pripremljenim test skriptama i test procedurama.

Testiranje po test skriptama - predstavlja metodu testiranja u kojoj se prvo definišu test scenariji i test slučajevi koje je potrebno proveriti, a onda testeri i test inženjeri rade testove po ovim skriptama. Testeri prate scenarije kao uputstvo o tome kako je potrebno testirati sistem dok test inženjeri koriste test scenarije kao specifikaciju za pravljenje automatskih testova kojima će se testirati sistem.

Istraživačko testiranje –je metoda testiranja u kojoj postoje test procedure ali se ne prate striktno.Tokom istraživačkog testiranja testeri otkrivaju alternativne pravce korisćenja sistema.Ideja ovog testiranja je da testeri tokom testiranja sami prave alternativna scenarija koja nisu unapred planirana. Ovaj tip testiranja ima smisla pošto u većini slučajeva tester piše test skripte a da uopšte nije ni video aplikaciju. Uspešnost ovog tipa testiranja zavisi od samih testera i njihovih sposobnosti.

Automatsko testiranje –Predstavlja metodu kojom se test procedure automatizuju pomoću nekih skripti za izvršavanje testova. Na ovaj način, test koji se jednom isplanira i za koji se kreira skripta može se uvek izvršavati bez trošenja vremena na ponavljanje koraka testa. Dobro testirani sistemi imaju stotine automatskih testova kojima se lako mogu testirati sve funkcionalnosti u projektu.

Regresivno testiranje – predstavlja metodu testiranja koja se često koristi u iterativnim metodama razvoja softvera. Tokom iterativnog razvoja se tokom svake iteracije implementira deo funkcionalnosti sistema i fokus testiranja je na tim funkcionalnostima. Kod ovakve metode testiranja rizik je u tome što funkcionalnosti koje su ranije testirane i potvrđene mogu prestati da rade zato što je tokom trenutne iteracije promenjen neki deo aplikacije koji utiče i na ranije implementirane funkcionalnosti. Na ovaj način, u testovima prolaze sigurno samo funkcionalnosti implementirane u trenutnoj iteraciji, dok sve ranije implementirane funkcije mogu pogrešno da rade zato što ih niko ne proverava.

Regresivno testiranje je metoda kojom se ponavljaju testovi koji su izvršeni u ranijim iteracijama kako bi se ispitalo da li ranije implementirane funkcije i dalje rade ispravno.

5.PRIMER

Aplikacija koju ćemo testirati kroz ovaj rad je posebno napisana za prikaz kako treba testirati i koje su to sve greške koje mogu da se jave u jednom softveru. Primer će se sastojati iz veb aplikacije koja izračunava površinu trougla, zahteva korisnika koji nam opisuju šta i kako bi ta aplikacija trebalo da radi i na kraju test slučajeva kao i defekata koji su se javili kroz testiranje.

```
<!DOCTYPE html>
<html>

<head>

<style>

input[type=radio] {
display: none;
}
input[type=radio] + label::before {
content: '';
display: inline-block;
border: 1px solid #000;
border-radius: 50%;
margin: 0 0.5em;
}
input[type=radio]:checked + label::before {
background-color:black;
}

.radio1 + label::before {
width: 0.2em;
height: 0.2em;
}

.radio2 + label::before {
width: 0.3em;
height: 0.3em;
}

.radio3 + label::before {
width: 0.45em;
height: 0.45em;
}

.radio4 + label::before {
width: 0.67em;
height: 0.67em;
```

```
}

*:focus {
outline: 0;
}
</style>


<script>
document.onmousedown=disableclick;

function disableclick(event)
{
if(event.button==2)
{
returnfalse;
}
}
</script>


</head>

<bodyoncontextmenu=" return false">

<script>

var a,b,c;

function izracunaj(){

var mere1 = document.getElementsByName("group1");
var mere2 = document.getElementsByName("group2");
for(var i =0; i < mere1.length; i++){
if(mere1[i].checked ==true){
for(var j =0; j < mere2.length; j++){
if(mere2[j].checked ==true){
var scale = i-j;
break;
}
}
break;
}
}
/*
switch(i) {
case 0:
var u = "mm"
break;
case 1:
var u = "cm"
break;
case 2:
var u = "dm"
break;
case 3:
var u = "m"
break;
    }
```

```
    */

switch(j){
case0:
var m ="mm"
break;
case1:
var m ="cm"
break;
case2:
var m ="dm"
break;
case3:
var m ="m"
break;
}

    a
=Number((document.getElementById("x").value).replace(",",".))*Math.pow(10,scale
);
    b
=Number((document.getElementById("y").value).replace(",",".))*Math.pow(10,scale
);
    c
=Number(parseFloat((document.getElementById("z").value).replace(",",".")))*Math.
pow(10,scale);
/*
document.getElementById("x").value=document.getElementById("x").value.toString()
+u;
document.getElementById("y").value=document.getElementById("y").value.toString()
+u;
document.getElementById("z").value=document.getElementById("z").value.toString()
+u;
    */
validiraj();

var p =(a+b+c);

var pov = parseFloat(Math.sqrt(p*(p-a)*(p-b)*(p-c)).toFixed(8));

var u1 =(Math.acos(-(c*c-a*a-b*b)/(2*a*b))*180/Math.PI).toFixed(2);
var u2 =(Math.acos(-(b*b-a*a-c*c)/(2*a*c))*180/Math.PI).toFixed(2);
var u3 =(Math.acos(-(a*a-c*c-b*b)/(2*c*b))*180/Math.PI).toFixed(2);


if(b>c){
    u2=(u2*1.03).toFixed(2);
}

document.getElementById("pov").value ="Povrsina: "+pov+m;
document.getElementById("u1").value ="Ugao 1: "+u1+"";
document.getElementById("u2").value ="Ugao 2: "+u2+"";
document.getElementById("u3").value ="Ugao 3: "+u3+"";




}
```

```
function validiraj(){

document.getElementById("red").style.backgroundColor ="transparent";
document.getElementById("green").style.backgroundColor ="transparent";
document.getElementById("red").style.color ="black";
document.getElementById("green").style.color ="black";
if(a+b>c&&a+c>b&&b+c>a&&a>0&&b>0&&c>0){
var flag=true;
boji(flag);
}else{
var flag=false;
boji(flag);
}
}
function boji(flag){
if(flag){
document.getElementById("green").style.backgroundColor ="green";
document.getElementById("green").style.color ="white";
document.getElementById("green").style.borderColor ="Black";

}else{
document.getElementById("red").style.backgroundColor ="red";
document.getElementById("red").style.color ="white";
document.getElementById("red").style.borderColor ="Black";
}
}


</script>
<fieldsetstyle="width:500px;height:330px;">
<pstyle = "color:blue;margin-left:120px;">PROGRAM ZA TROUGLOVE</p>
<div>
<divstyle = "width:70px;height:100px;float:left;line-height:250%;display:inline;
padding-right: 20px;">
        Stranica 1<br/>
        Stranica 2<br/>
<spanstyle = "font-size:94%;">Stranica 3</span><br/>
</div>
<divstyle = "width:120px;height:100px;float:left;line-
height:40px;display:inline;padding-right:30px;">
<inputtype="text"id="x"size="7"value=""/>
<inputtype="text"id="y"size="7"value=""/>
<inputtype="text"id="z"size="7"value=""/>
<inputtype="button"id="button"onclick = "izracunaj()"value="IZRACUNAJ"/>
</div>
<divstyle = "border-style:solid;border-
width:1px;float:left;width:220px;height:160px;display:block;">
<inputtype="text"id="pov"size="27"value="Povrsina: "style =
"border:none;display:block;"readonly/>
<inputtype="text"id="u1"size="27"value="Ugao 1: "style =
"border:none;display:block;"readonly/>
<inputtype="text"id="u2"size="27"value="Ugao 2: "style =
"border:none;display:block;"/>
<inputtype="text"id="u3"size="27"value="Ugao 3: "style =
"border:none;display:block;"readonly/>
</div>
```

```
</div>

<divstyle = "display:block;float:left;padding-top:15px;padding-left:12px;">
    Jedinica mere za stranice
<br/>
<inputid="l1"type="radio"name="group1"class="radio1"/>
<labelfor="l1">mm</label>
<inputid="l2"type="radio"name="group1"class="radio2"checked/>
<labelfor="l2">cm</label>
<inputid="l3"type="radio"name="group1"class="radio3"/>
<labelfor="l3">dm</label>
<inputid="l4"type="radio"name="group1"class="radio4"/>
<labelfor="l4">m</label>
</div>

<divstyle = "display:block;float:left;padding-top:15px;padding-
left:50px;padding-right:40px;">
      Jedinica mere za rezultat
<br/>
<inputid="r1"type="radio"name="group2"class="radio1"/>
<labelfor="r1">mm </label>
<inputid="r2"type="radio"name="group2"class="radio2"checked/>
<labelfor="r2">cm</label>
<inputid="r3"type="radio"name="group2"class="radio3"/>
<labelfor="r3">dm</label>
<inputid="r4"type="radio"name="group2"class="radio4"/>
<labelfor="r4">m</label>
</div>
<divid = "green"style = "display:block;border-
style:solid;float:right;width:20px;height:20px;margin:7px 10px 0 450px;border-
radius:50%;border-width:1px;padding:2px 0 0 2px;">
 Z
</div>
<divid = "red"style = "display:block;border-
style:solid;float:right;width:20px;height:20px;margin:5px 10px 0 0;border-
radius:50%;border-width:1px;padding:2px 0 0 2px;">
 R
</div>

</fieldset>

</body>
</html>
```

Početak izrade svakog softvera ili veb aplikacije podrazumeva da su specificirani korisnički zahtevi. U tabeli 2nalazi se primer korisničkih zahteva.

Zahtevi za PROGRAM ZA TROUGLOVE
1. Svrha ove aplikacije je izračunavanje površine trougla, kao ulazni parametri koriste se stranice trougla.
2. Aplikacija izračunava sva tri ugla trougla na osnovu dužina stranica.
3. Polja "Stranica 1", "Stranica 2" i "Stranica 3" za unošenje vrednosti strana imaju istu veličinu i nalaze se na jednakim razmacima.
4. Dugme "IZRACUNAJ" je 50% šire i duže od polja "Stranica X".
5. Radio dugmići za mere su u sledećem rasporedu: mm, cm, dm, m. Svako dugme je za 50% veće od prethodnog.
6. U polje sa rezultatima se ne mogu upisivati vrednosti.
7. Naslov aplikacije je "PROGRAM ZA TROUGLOVE" i napisan je plavim slovima.
8. Na donjem desnom delu aplikacije nalaze se dva polja: 1)Polje sa slovom "Z" se nalazi iznad i zasvetli zelenom bojom, kada se unesu vrednosti koje formiraju trougao u polje sa stranicama i klikne se na IZRACUNAJ 2)Polje sa slovom "R" se nalazi ispod i zasvetli crvenom bojom kada se unesu vrednosti koje ne formiraju trougao i klikne se na IZRACUNAJ 3)Kada se klikne na ova polja ne dešava se ništa.
9. Vrednosti koje se unose mogu imati do tri decimale.
10. Svi rezultati su zaokruženi na dve decimale.
11. Dozvoljeni nivo greške je 1%.
12. Polja za unos stranica prihvataju samo brojeve i "." kao oznaku za decimalu.

Tabela 2. Zahtevi za program za trouglove

U agilnom načinu testiranja, testeri i inženjeri dobijaju korisničke zahteve u isto vreme kada i programeri. Prvi korak testera je da analiziraju zahteve koje su dobili.

Posle čitanja zahteva primećuje se da korisnički zahtev broj osam nije logičan. Prvi korak testera u ovom slučaju je kontaktiranje klijenta i konsultacija po pitanju Z i R. Dakle u realnosti bi ovaj zahtev verovatno bio prepravljen, pa bi Z (zeleno) bilo promenjeno na G (green) ili bi R (red) bilo prepravljeno na C (crveno).

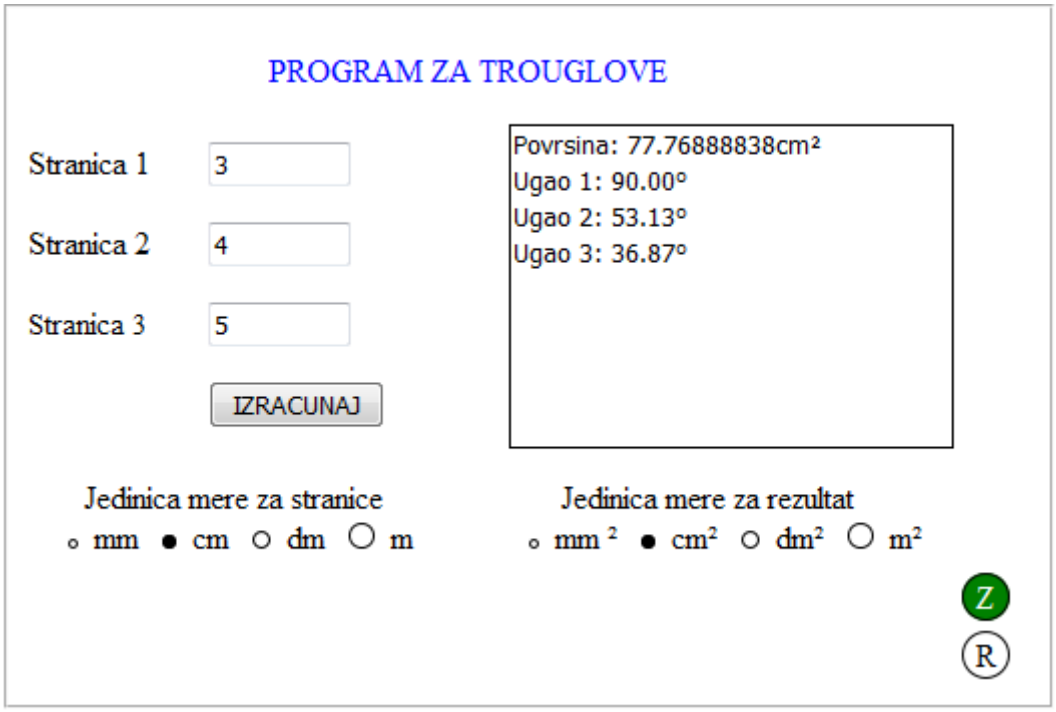
Nakon utvrđivanja da su zahtevi ispravni kreće se sa pisanjem test slučajeva.

Svaki zahtev mora biti pokriven test slučajevima. Svaki test slučaj povezan je sa jednim ili više korisničkih zahteva. Tester prvo analizira korisničke zahteve i osmišljava slučajeve po kojima će testirati veb aplikaciiju. U nastavku rada će se videti test slučajevi koji testiraju korisničke zahteve iz tabele 1.

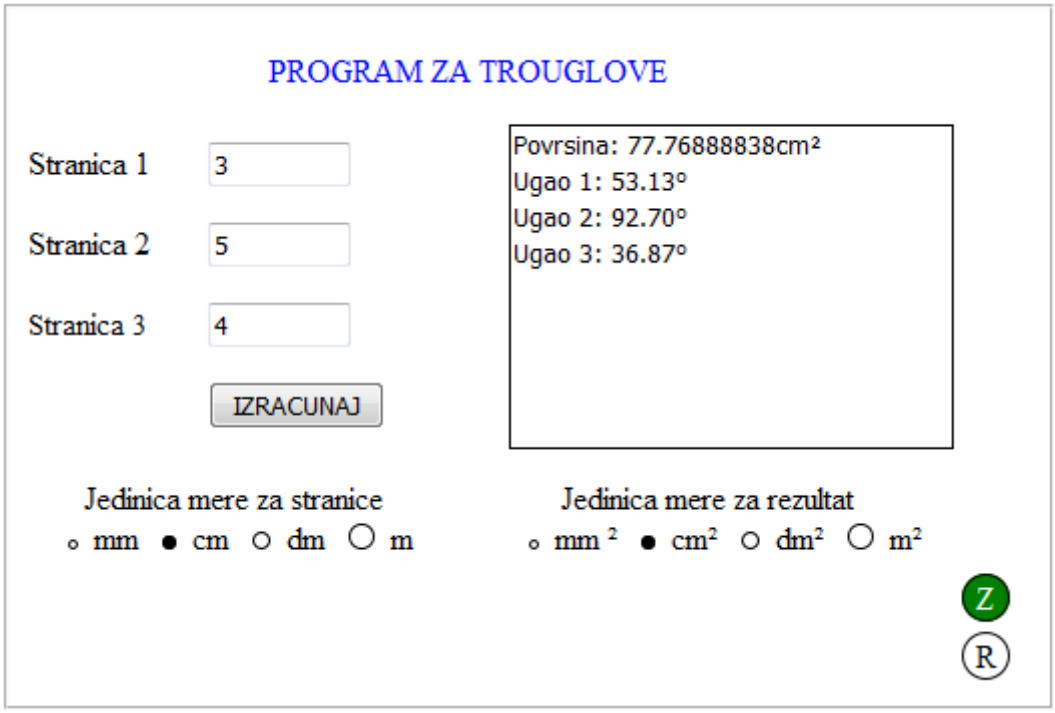
5.1. Test slučajevi

Test slučaj 1	
Koraci	Očekivani rezultat
Otvori aplikaciju	Aplikacija uspešno otvorena
Popuni polja sa sledećim vrednostima: -Stranica 1=3 -Stranica 2=4 -Stranica 3=5 i klikni na dugme IZRACUNAJ NAPOMENA: Ovaj test slučaj treba proveriti sa svim mogućim kombinacijama stranica: -stranica 1 najduža -stranica 2 najduža -stranica3 najduža	Površina i svi uglovi tačno su izračunati

Tabela 3. Test slučaj 1



Slika 3. Test slučaj 1

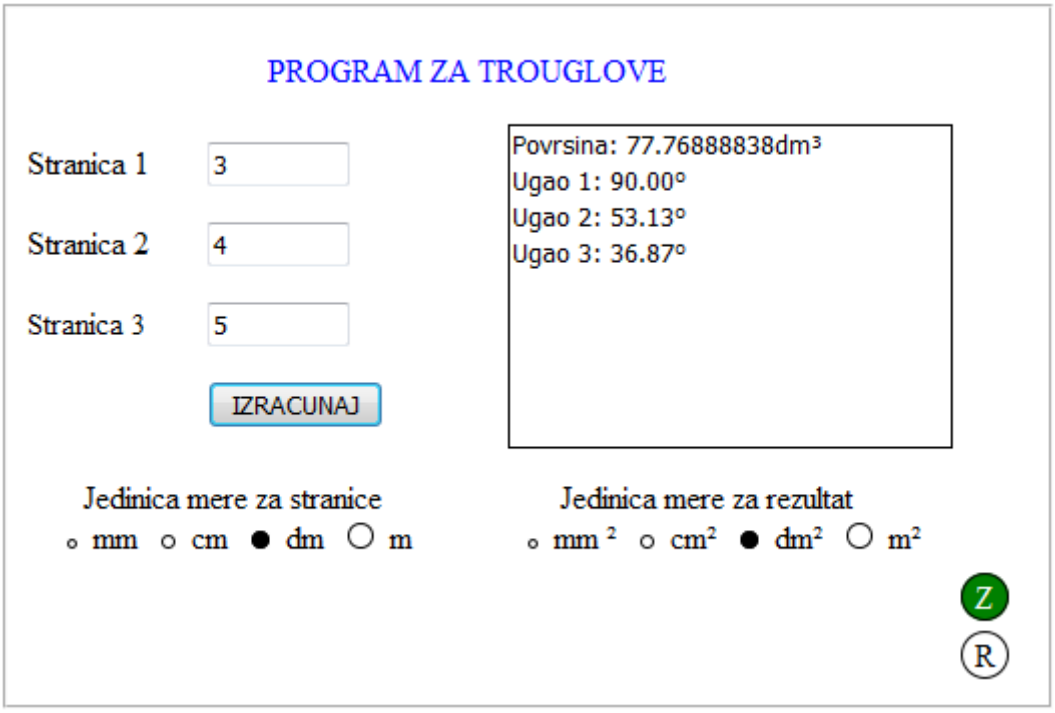


Slika 4. Test slučaj 1a

Ovaj test slučaj je detektovao defekt broj 1. U slučaju kada je stranica 2, veća od stranice 3, ugao 2 nije dobro izračunat i zbir uglova nije 180 stepeni.

Test slučaj 2	
Koraci	Očekivani rezultat
Otvori aplikaciju	Aplikacija uspešno otvorena
Popuni polja sa sledećim vrednostima: -Stranica 1=3 -Stranica 2=4 -Stranica 3=5 i klikni na dugme IZRACUNAJ NAPOMENA: Ovaj test slučaj treba istestirati sa svim mogućim kombinacijama radio dugmića.	Površina i svi uglovi tačno su izračunati

Tabela 4. Test slučaj 2

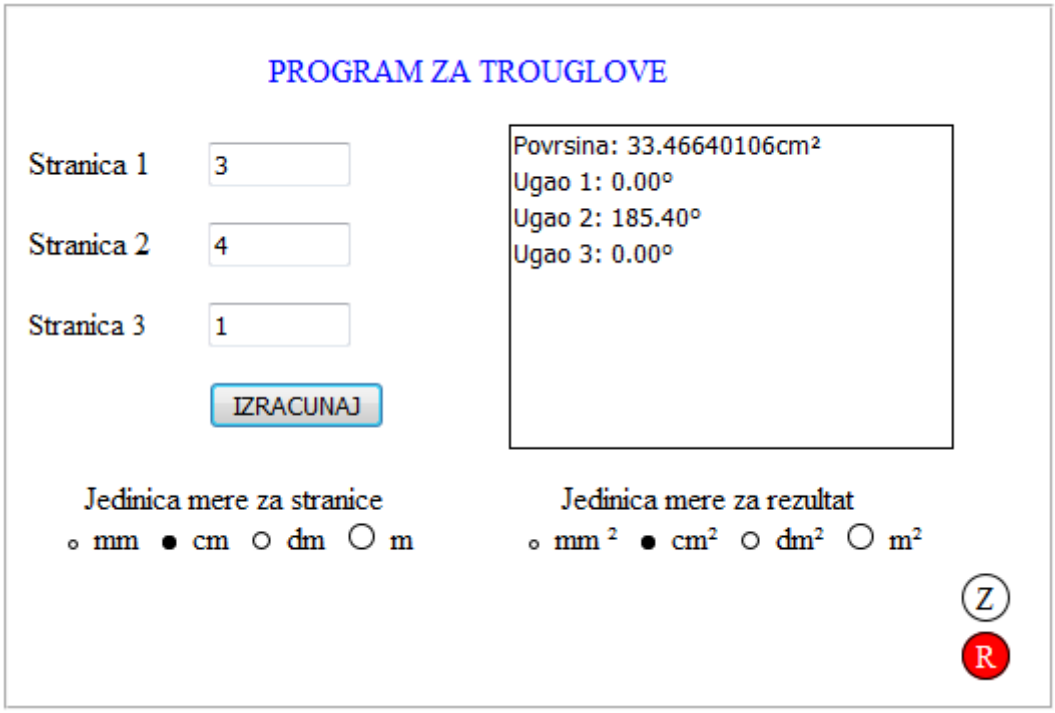


Slika 5. Test slučaj 2

Ovaj test slučaj je detektovao defekt broj 2, na slici 5 jasno se vidi da je površina izražena u dm^3.

Test slučaj 3	
Koraci	Očekivani rezultat
Otvori aplikaciju	Aplikacija uspešno otvorena
Popuni polja sa sledećim vrednostima: -Stranica 1=3 -Stranica 2=4 -Stranica 3=1 i klikni na dugme IZRACUNAJ	Crveno dugme R signalizira da zadate stranice ne formiraju trougao.

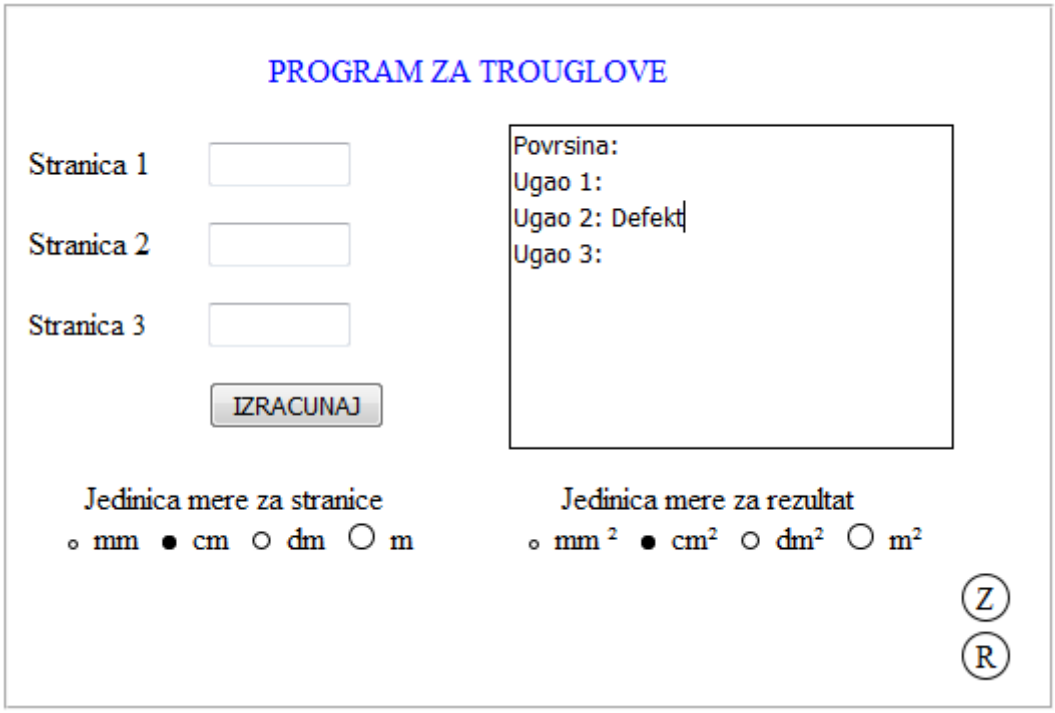
Tabela 5. Test slučaj 3



Slika 6. Test slučaj 3

Test slučaj 5	
Koraci	Očekivani rezultat
Otvori aplikaciju	Aplikacija uspešno otvorena
Proveriti da je dugme "IZRACUNAJ" 50% šire i duže od polja "Stranica X".	Dugme "IZRACUNAJ" je 50% šire i duže od polja "Stranica X".
Test slučaj 6	
Koraci	Očekivani rezultat
Otvori aplikaciju	Aplikacija uspešno otvorena
Proveriti da su radio dugmići za mere u sledećem rasporedu: mm, cm, dm, m. Svako dugme je za 50% veće od prethodnog.	Radio dugmići za mere su u sledećem rasporedu: mm, cm, dm, m. Svako dugme je za 50% veće od prethodnog.
Test slučaj 7	
Koraci	Očekivani rezultat
Otvori aplikaciju	Aplikacija uspešno otvorena
Proveriti da u polje sa rezultatima ne mogu da se upisuju vrednosti	U polje sa rezultatima se ne može upisati ništa

Tabela 7. Test slučajevi 5, 6 i 7

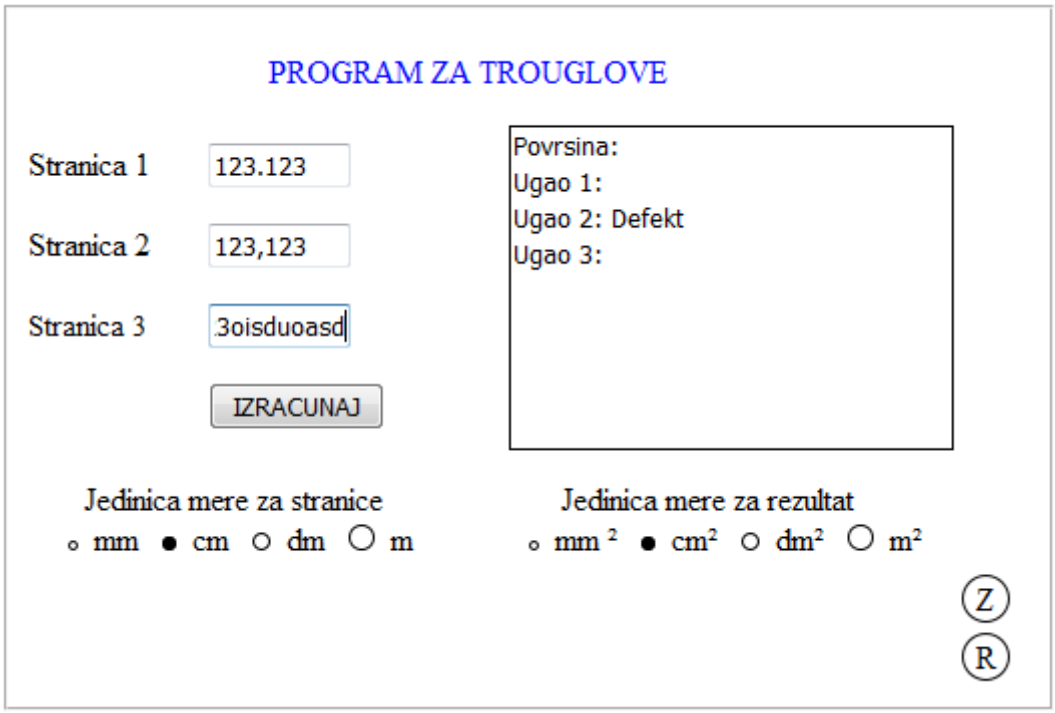


Slika 7. Test slučaj 7

Ovaj test slučaj je detektovao defekt broj 3. Polje za ugao 2 dozvoljava korisniku da upisuje vrednosti.

Test slučaj 8	
Koraci	Očekivani rezultat
Otvori aplikaciju	Aplikacija uspešno otvorena
Proveriti da je naslov aplikacije "PROGRAM ZA TROUGLOVE" i napisan je plavim slovima.	Naslov aplikacije je "PROGRAM ZA TROUGLOVE" i napisan je plavim slovima.
Test slučaj 9	
Koraci	Očekivani rezultat
Otvori aplikaciju	Aplikacija uspešno otvorena
Proveriti da Dugmići Z i R rade pravilno	Dugmići Z i R rade pravilno
Test slučaj 10	
Koraci	Očekivani rezultat
Otvori aplikaciju	Aplikacija uspešno otvorena
Proveriti da polja Stranica 1, Stranica 2 i Stranica 3 prihvataju samo brojeve i tačku	Polja Stranica 1, Stranica 2 i Stranica 3 prihvataju samo brojeve i tačku

Tabela 8. Test slučajevi 8, 9 i 10



Slika 8. Test slučaj 10

Ovaj test slučaj je detektovao defekt broj 4. Aplikacija osim brojeva i tačke prihvata i zarez, a polje za stranicu 3 prihvata i slova.

Nakon izvršavanja svih testova u aplikaciji, nađen je veliki broj defekata.

Spisak svih defekata:

1. Ako je Stranica 2> Stranica 3 -> Ugao 2 netačan -> zbir nije 180
2. Kada je izabrana površina u kvadratnim decimetrima, prikazuje se rezultat u kubnim decimetrima.
3. Polje za ugao 2 dozvoljava korisniku da unosi vrednosti.
4. Aplikacija osim brojeva i tačke prihvata i zarez, a polje za stranicu 3 prihvata i slova.

Sledeći korak test tima je da listu ovih defekata prosledi programerima. Programeri su ti koji sada treba da preprave greške u kodu i reše sve defekte koje su dobili od test tima. Kada test tim dobije aplikaciju sa ispravljenim defektima, ponovo se prolazi kroz sve testove kako bi se utvrdilo da aplikacija radi po korisničkim zahtevima.

6. ZAKLJUČAK

U softverskoj industriji se formalno svi zalažu za testiranje i podizanje kvaliteta, međutim, čini se da testiranje nije dobilo ravnopravan položaj sa ostalim granama razvoja softvera. Položaj i uloga test timova se znatno razlikuje od kompanije do kompanije. U nekim kompanijama testeri su samo pomoćno osoblje koje samo čeka da glavni tim razvije aplikaciju pa da onda oni pogledaju. Takve firme u test timove ne stavljaju dovoljno kvalitetne ljude nego ili početnike ili programere koje je vreme pregazilo samo da ih nekako iskoriste. Uslov je uglavnom da nisu spremni da uđu u pravi programerski tim ili da ne koštaju mnogo pošto se doživljavaju samo kao neizbežan trošak. U drugim kompanijama testeri su profesionalci specijalizovani za kontrolu kvaliteta. Takve kompanije se lako prepoznaju po tome što imaju stabilne softverske proizvode koje koristi veliki broj zadovoljnih korisnika. Ove kompanije znaju mali trik u proizvodnji softvera. Naime, uvek možete da nađete izvanredne stručnjake u oblastima programiranja koji mogu da naprave sve što vam treba, ali da li će rezultat njihovog rada biti kvalitetan softver ili će biti odbačen zavisi od kvaliteta proizvoda koji obezbeđuje test tim. Takve kompanije se razlikuju od ostalih po tome što u svojoj korporativnoj kulturi iskreno pridaju pravi značaj kontroli kvaliteta.

Često nisu toliko potrebni timovi izuzetnih programera da bi se napravio softver. Kompleksni algoritmi se često nalaze u bibliotekama funkcija koje samo čekaju da budu pozvane, sporije aplikacije ionako rade brzo kada se stave u moderne infrastrukture kao što su oblaci, kompleksan kod se lako generiše brojnim alatima i generatorima koda. Najveće softverske kompanije i grupe svakodnevno objavljuju sve bolje i bolje arhitekture i komponente koje samo čekaju da ih neko prouči i iskoristi i napravi odličan softver. Danas većina firmi može da napravi aplikacije odličnih karakteristika čak i sa osrednjim programerskim timovima. Pošto je mnogim firmama omogućeno da na jednostavan način dostignu zadovoljavajući nivo kvaliteta potrebno je nešto drugo što će ih dići iznad proseka kvaliteta koji se očekuje. Jedna od najčešćih stvari koja razlikuje dobru kompaniju sa prosečnim softverom od odlične kompanije sa odličnim softverom je upravo proces kontrole kvaliteta.

Veliki softverski sistemi odličnih performansi i dobrih funkcionalnosti često su propadali samo zato što nisu odgovarali korisnicima kada su bili isporučeni. Ogromno vreme, potrošeno na fantastične projekte, bačeno je u vodu samo zato što rezultat rada, koliko god bio kvalitetan, nije odgovarao korisnicima. Dok je projektni tim ispravio greške, brzo se pojavio neko drugi ko je svojim rešenjem osvojio korisnike. Razlog propadanja je jednostavan - niko nije u pravom trenutku proverio da li projektni timovi rade pravu stvar. To je upravo nedostatak kontrole kvaliteta na projektu.

Mnoge uspešne softverske firme nisu na svojim pozicijama samo zbog softvera koji korisnicima nudi prave funkcionalnosti, nego zbog sistematičnog pristupa verifikaciji i validaciji svih aktivnosti razvoja softvera. Drugim rečima, te firme znaju koliko je proces testiranja bitan i znaju da cene napor kontrole kvaliteta, pošto će na kraju to blti jedina razlika između njih i konkurencije, ali i najbolji način da se dobar proizvod isporuči korisnicima. Kvalitet je danas

zadajna stvar i samo kompanije koje mogu da ga obezbede mogu da opstanu na tržištu. Naravno pored formalnog zalaganja za kvalitet, kompanije moraju i stvarno da promene fokus tako što neće stavljati u test tim najjeftinije nego najkvalitetnije ljude koji se mogu potpuno posvetiti kontroli kvaliteta.

Ovaj rad bi trebalo da približi studentima značaj testiranja softvera, kao i same metode i načine testiranja. Rad može da posluži studentima kao polazna tačka za istraživanje ove oblasti.

LITERATURA

[1] Peter Morgan, Angelina Samaroo, Brian Hambling - Software Testing - An ISTQB-ISEB Foundation Guide, 2nd Edition

[2] Quality code, Stephen Vance

[3] Foundations of software testing, third edition

[4] The art of software testing, third edition