Testiranje programskih rješenja u mrežnom okruženju

Eskić, Endrina

Sveučilište J.J. Strossmayera u Osijeku

Filozofski fakultet

Dvopredmetni diplomski studij informatologije i informacijskih tehnologija

Endrina Eskić

Testiranje programskih rješenja u mrežnom okruženju

Diplomski rad

Mentor: izv. prof. dr. sc. Boris Bosančić

Sumentor: Maja Klajić, mag. inform.

Osijek, 2019.

Sveučilište J.J. Strossmayera u Osijeku
Filozofski fakultet Osijek
Odsjek za informacijske znanosti
Dvopredmetni diplomski studij informatologije i informacijskih tehnologija

---

Endrina Eskić

Testiranje programskih rješenja u mrežnom okruženju

Diplomski rad

Društvene znanosti, Informacijske i komunikacijske znanosti, Informacijsko i programsko inženjerstvo

Mentor: izv. prof. dr. sc. Boris Bosančić

Sumentor: Maja Klajić, mag. inform.

Osijek, 2019.

Prilog: Izjava o akademskoj čestitosti i o suglasnosti za javno objavljivanje

Obveza je studenta da donju Izjavu vlastoručno potpiše i umetne kao treću stranicu završnog odnosno diplomskog rada.

IZJAVA

Izjavljujem s punom materijalnom i moralnom odgovornošću da sam ovaj rad samostalno napravio te da u njemu nema kopiranih ili prepisanih dijelova teksta tuđih radova, a da nisu označeni kao citati s napisanim izvorom odakle su preneseni.
Svojim vlastoručnim potpisom potvrđujem da sam suglasan da Filozofski fakultet Osijek trajno pohrani i javno objavi ovaj moj rad u internetskoj bazi završnih i diplomskih radova knjižnice Filozofskog fakulteta Osijek, knjižnice Sveučilišta Josipa Jurja Strossmayera u Osijeku i Nacionalne i sveučilišne knjižnice u Zagrebu.

U Osijeku, datum

Ružica Ešić 01222 18387
ime i prezime studenta, JMBAG

Sažetak

Svrha ovog rada je ukazati na važnost testiranja programskih rješenja u općenitom smislu, s naglaskom na testiranje mrežnih stranica. Osim toga, rad nastoji predstaviti automatsko testiranje programskih rješenja kao opciju koja je dugoročno isplativija te manje vremenski zahtjevna od ručnog testiranja. U teorijskom dijelu rada definira se testiranje kao pojam te navode razine i metode testiranja. Pritom, razlikuju se četiri razine te tri metode testiranja. Nadalje, navode se i klasificiraju vrste grešaka koje se javljaju pri testiranju. U radu je opisan i proces testiranja koji se sastoji od pet dijelova, a to su planiranje i kontrola, analiza i projektiranje, provedba i izvršenje, ocjenjivanje izlaznih kriterija i izvješćivanja te aktivnosti zatvaranja testova. U praktičnom dijelu rada, prije izrade testova za automatsko testiranje mrežnih stranica Filozofskog fakulteta u Osijeku, navedene su i opisane sve tehnologije korištene u izradi testova. Zatim se opisuje proces izrade automatskih testova koji uključuje podjelu kategorija za testiranje, definiranje varijabli te pojašnjavanje metoda koje su korištene u pisanju testa. Izradom automatskih testova ukazalo se na neke nedostatke mrežnog mjesta, ali je glavna svrha testova ispunjena, a to je olakšano testiranje validnosti mrežnog mjesta. Automatski testovi kreirani u ovom radu namijenjeni su tome da postanu sastavni dio uobičajene procedure testiranja mrežnog mjesta Filozofskog fakulteta u Osijeku.

Ključne riječi: osigurane kvalitete, automatsko testiranje, node.js, JavaScript, WebDriverIO

SADRŽAJ

Sažetak 4

SADRŽAJ 5

1. UVOD 7

2. POJAM TESTIRANJA: RAZINE, METODE, VRSTE I PROCES TESTIRANJA 9

2.1. Uvodna razmatranja 9

2.2. Razine testiranja 13

2.3. Metode testiranja 16

2.3.1. Metoda bijele kutije 16

2.3.2. Metoda crne kutije 16

2.3.3. Metoda sive kutije 17

2.3.4. Vrste grešaka i njihova klasifikacija 19

2.4. Vrste testiranja 22

2.4.1. Ručno testiranje 22

2.4.2. Automatsko testiranje 23

2.4.3. Usporedba ručnog i automatskog testiranja 24

2.5. Kvaliteta programskog rješenja koje se testira 25

2.6. Proces testiranja 28

2.6.1. Uvodna razmatranja 28

2.6.2. Planiranje i kontrola 28

2.6.3. Analiza i projektiranje 30

2.6.4. Provedba i izvršenje 31

2.6.5. Ocjenjivanje izlaznih kriterija i izvješćivanje 32

2.6.6. Aktivnosti zatvaranja testova 33

3. AUTOMATSKO TESTIRANJE MREŽNIH STRANICA FILOZOFSKOG FAKULTETA U OSIJEKU 34

3.1. Cilj i svrha automatskog testiranja mrežnih stranica FFOS-a 34

3.2. Tehničke informacije 35
3.2.1. Uvodna razmatranja 35
3.2.2. Selenium 35
3.2.3. WebDriverIO 36
3.3. Instalacija programa za testiranje 37
3.4. Podjela testova 38
3.5. Varijable 43
3.6. Testovi 44
3.7. Prikaz testova 46
3.8. Problemi u testiranju 48
4. ZAKLJUČAK 50
Literatura 51

1. UVOD

U današnje vrijeme programska rješenja smatraju se sastavnim dijelovima života. Od obrazovanja, posla, zdravstva i zabave, društvo se koristi različitim programskim rješenjima. Dijelom jer je primorano na to, a dijelom jer to želi. Istina je i da je susretanje s programskim rješenjima danas neizbježno. Informatika je postala sastavni dio života do te mjere da se informatička pismenost smatra osnovom svake pismenosti. S obzirom da društvo konzumira velik broj programskih rješenja, logično je kako bi bilo poželjno da su ti proizvodi kvalitetni. Izrada programskog rješenja nije jednostavan posao. Dijeli se u nekoliko koraka, od njegova osmišljavanja kao proizvoda preko same izrade do testiranja i isporuke korisnicima. Često se dogodi da se korak testiranja zanemari, odnosno da mu se ne posveti potrebna pažnja. Testiranje je proces koji se odvija prije isporuke proizvoda i osigurava njegovu kvalitetu.[1] Tester je osoba koja se bavi procesom testiranja, bilo da se radi o ručnom ili automatiziranom postupku.[2] Programska rješenja koja imaju brojne greške su ona čija se izrada ne isplati. Takva se programska rješenja kratko koriste te zahtijevaju stalno ulaganje novca. Testiranjem se izbjegavaju greške u programskim rješenjima koje uzrokuju gubitak korisnika. Očekivanje da će programsko rješenje koje je prošlo kroz proces testiranja biti bez grešaka pomalo je utopijsko, no ono će zasigurno imati barem trivijalne greške koje korisnici neće primijetiti.

Može se reći da je danas sve *online*, popularizacija informacijskih tehnologija i njihovo korištenje dale su novu dimenziju kvaliteti programskog rješenja. Mrežna programska rješenja u formi dinamičkih mrežnih mjesta (engl. *web site*) mogu se promatrati kroz razne aspekte. Kroz te iste aspekte ona se mogu i testirati. Zato je testiranje proces koji obično dugo traje. Rješenje za taj problem predstavlja automatizacija testova, odnosno pisanje kôda za automatske testove koji samostalno testiraju programsko rješenje.[3] No tester u procesu testiranja može saznati kako korisnik percipira programsko rješenje, i kako ga je najlakše prilagoditi korisnicima. Rješenje mora biti intuitivno i lako za korištenje. Kada govorimo o programskim rješenjima u mrežnom okruženju, to znači da korisniku ne bi trebalo biti teško pronaći informacije koje su mu potrebe na određenoj mrežnoj stranici.

U ovom diplomskom radu pojašnjen je pojam testiranja validnosti programskih rješenja te razine i metode procesa testiranja. Opisane su i vrste grešaka koje se javljaju u testiranju. Nadalje

[1] Usp. Linz, Tino; Spillner,Andreas; Schafer,Hans. Software testing foundations. Santa Barbara: Rocky Nook Inc., 2014. str. 9.
[2] Usp.Isto, str. 173.
[3] Usp.Isto, str.213.

definira se proces testiranja te se pojašnjavaju njegovi koraci. Nakon toga pojašnjavaju se dvije osnovne vrste testiranja - ručno i automatsko - te se daje njihova usporedba. U praktičnom dijelu rada, opisuju se koraci testiranja mrežnih stranica. Generalno, testirale su se interaktivnost i učitavanje elemenata. Važno je naglasiti da automatski testovi izrađeni u sklopu ovog diplomskog rada ne obuhvaćaju, odnosno ne testiraju mrežne stranice Filozofskog fakulteta u Osijeku u potpunosti već su otvoreni za daljnje proširenje, no dobra su polazišna točka za izradu plana testiranja spomenutih mrežnih stranica u budućnosti.

2. POJAM TESTIRANJA: RAZINE, METODE, VRSTE I PROCES TESTIRANJA

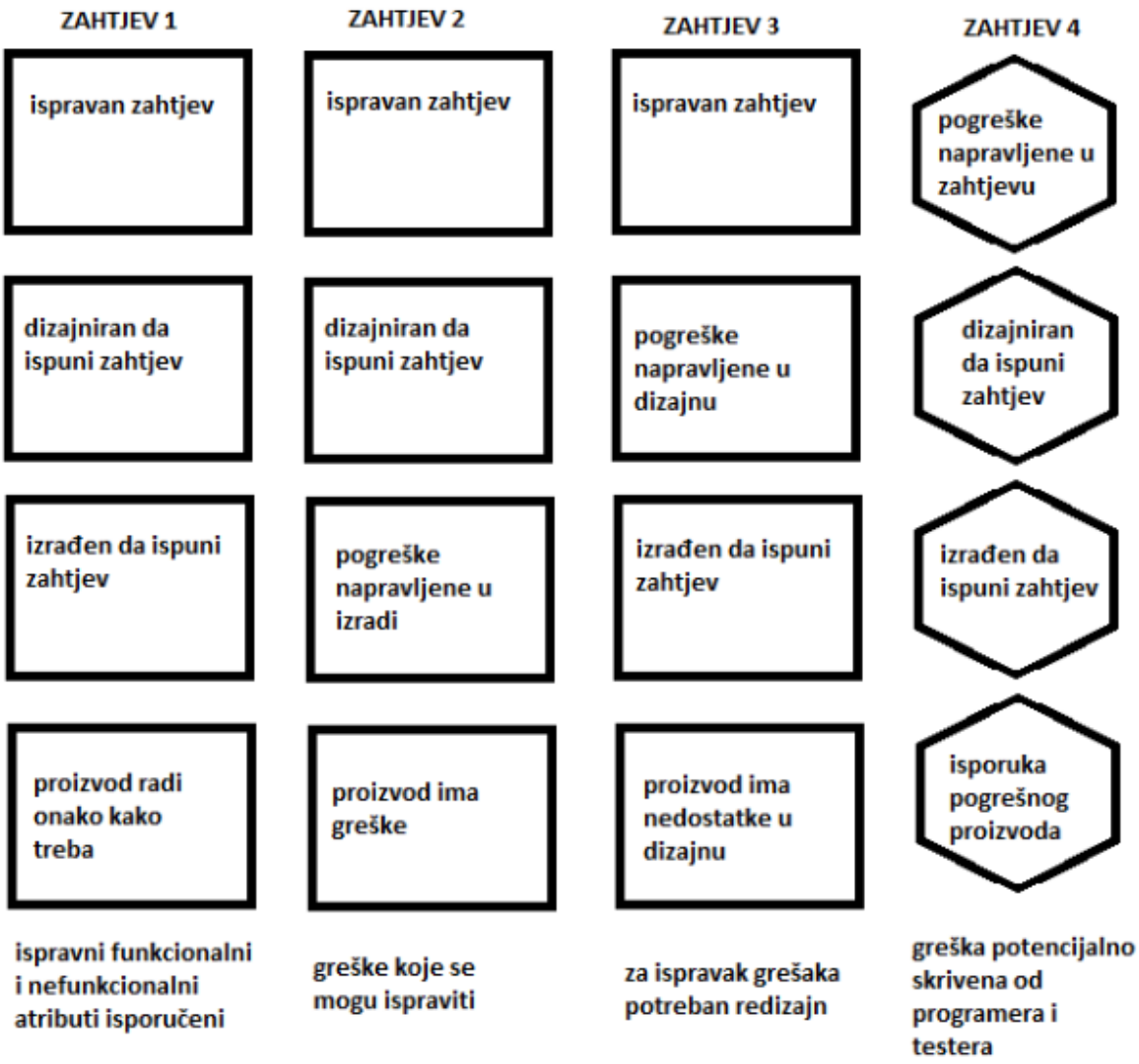
2.1. Uvodna razmatranja

Kada se u svakodnevnom životu zapitamo što je to testiranje, najvjerojatnije je da ćemo ga opisati kao radnju kojom osiguravamo da je ono što testiramo ispravno. Možemo se zapitati zašto nam je testiranje uopće potrebno? Ono je nužno ako želimo programsko rješenje sa što manjim brojem grešaka, jer svi rade pogreške. Neke greške mogu biti zanemarive, dok neke mogu predstavljati veliku opasnost prema raznim čimbenicima. Kako bi rizik od velikih opasnosti smanjili na što je moguće manju mjeru, koristimo procese testiranja. Važno je uvijek biti svjestan mogućnosti pogreške upravo kako bi se ona mogla eliminirati. Dok su neke pogreške jasno vidljive i provjerom ih možemo lako uočiti, pogreške nastale na temelju loših pretpostavki teško ćemo uočiti. U idealnom slučaju, proces testiranja vrši neka treća strana koja u proces testiranja ulazi objektivnog, nepristranog stava. No, je li bitno ima li pogrešaka u onome što radimo i je li toliko važno ako neke od tih grešaka ne pronađemo? Znamo da su u životu neke greške potpuno zanemarive, dok su neke vrlo važne. Isto vrijedi i za programska rješenja. Moramo poznavati greške sustava i koje one probleme donose. Kako bismo time uspješno ovladali, moramo u obzir uzeti kontekst unutar kojeg programsko rješenje nastaje i u kojem se okruženju koristi. U današnje vrijeme programska rješenja koristimo u skoro svakoj sferi svakodnevnice, od poslovnog okruženja do provođenja slobodnog vremena. Samim time, velika je mogućnost za susret s programskim rješenjem koje jednostavno ne funkcionira, bilo da se mrežne stranice ne učitavaju u preglednik ili da se obrada kreditne kartice u bankomatu ne može provesti. Sva programska rješenja nisu ista, pa tako ni greške koje se u njima javljaju nisu od iste važnosti. Kako bismo proveli testiranje, kreiramo niz testnih slučajeva na osnovi unaprijed definiranih uvjeta putem kojih ispitivač određuje da li sustav koji se testira zadovoljava zahtjeve ili radi ispravno.[4] Rizik je nešto što se nije dogodilo, ali ima potencijal da se dogodi.[5] Rizici nas zabrinjavaju, jer svojom mogućnošću na nas ostavljaju negativan dojam. Primjerice, kada putujemo avionom postoji određeni rizik od pada aviona koji ovisi o brojnim čimbenicima, npr. o vremenskoj prognozi, stanju aviona, iskustvu pilota i sl. Kod programskih rješenja, ti rizici obuhvaćaju, primjerice, kvalitetu hardvera na kojem se ono izvršava, verziju programskog rješenja, znanje korisnika, intuitivnost korisničkog sučelja i sl. Problemi s kojima se

[4] Usp. Software Testing Fundamentals. URL:http://softwaretestingfundamentals.com/test-case/ (16.06.2019.)
[5] Usp. Graham, Dorothy [et.al.]. Foundations of Software Testing ISTQB Certification. UK: Gaynor Redvers-Mutton, 2008. str.5.

susrećemo kada programsko rješenje ima nedostatak mogu biti trivijalni kao primjerice tipografske greške, no mogu biti i potencijalno skupi i štetni, kao što bi bio gubitak novca kod greške u obradi kreditne kartice. Važno je napomenuti da i tipografske greške koje su u većini slučajeva trivijalne, ovisno o namjeni programskog rješenja i konteksta, mogu biti itekako značajne. Kao društvo, stvorili smo dojam da su programska rješenja savršena tj. da svoje zadatke uvijek izvršavaju onako kako je to zamišljeno. Zašto, onda, ponekad ne rade ispravno? Dvije su mogućnosti. Prva jest da se programsko rješenje ne koristi ispravno što izravno dovodi do problema da se ne "ponaša" onako kako to od njega očekujemo. Druga mogućnost jest greška u samom procesu dizajniranja i izrade programskog rješenja. Takva se greška popularno naziva *bug*.[6] Moguće je da unutar programskog rješenja postoje brojne greške koje nismo otkrili te samim time ne uzrokuju nikakve probleme. No to je prevelik rizik kada je u programsko rješenje uloženo puno vremena i novca. Upravo je zato testiranje veoma važno. Testiranje pomaže u mjerenju kvalitete programskog rješenja u smislu broja pronađenih grešaka, slučajeva testiranja i sustava pokrivenih testovima. Testiranje je proces koji se sastoji od aktivnosti, kako statičkih tako i dinamičkih, koje se bave planiranjem, pripremom i vrednovanjem programskih rješenja i srodnih proizvoda za rad kako bi se utvrdilo jesu li zadovoljili određene zahtjeve, kako bi pokazali da su prikladni za određenu svrhu te kako bi se otkrili njihovi nedostaci.[7] Na slici 1. prikazana su 4 načina zbog kojih dolazi do grešaka u programskim rješenjima. [8]

[6] Usp. Isto, str.6.
[7] Usp. ISTQB Glossary. URL: https://glossary.istqb.org/en/search/testing (18.06.2019.)
[8] Usp. Graham, Dorothy [et.al.]. Foundations of Software Testing ISTQB Certification. UK: Gaynor Redvers-Mutton, 2008. str.8.

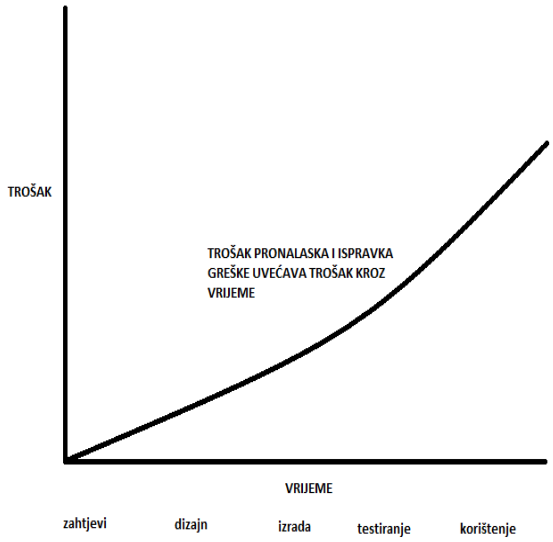


Slika 1. Načini pojave grešaka[9]

Možemo vidjeti da je zahtjev 1 implementiran ispravno. Pojam proizvod u ovom se kontekstu odnosi na programska rješenja. Razumjeli smo zahtjev klijenta, ispravno dizajnirali proizvod kako bi zadovolji taj zahtjev, izgradili ga ispravno u skladu s dizajnerskim načelima, i isporučili taj proizvod s ispravnim atributima. Proizvod je funkcionalan i radi ono što to bi i trebao. Po pitanju ostalih zahtjeva, pogreške su napravljene u različitim fazama. Zahtjev 2 je u redu sve dok se programsko rješenje ne počne izrađivati, kada dolazi do pogrešaka i pojavljivanja nedostataka. Te je greške lako uočiti i ispraviti tijekom testiranja, jer vidimo da proizvod ne zadovoljava dogovorene specifikacije. Nedostaci koji se pojavljuju u zahtjevu 3 se teže rješavaju. Kreirano je ono što je dogovoreno, ali nažalost dizajner je napravio neke pogrešne korake zbog kojih postoje

[9] Usp. Isto.

mane u dizajnu. Ako se proizvod ne provjeri prema zahtjevima, ne možemo uočiti te nedostatke tijekom testiranja. Kada ih primijetimo, bit će ih teško popraviti jer će biti potrebne promjene u dizajnu. Nedostaci u zahtjevu 4 uvedeni su tijekom definiranja zahtjeva; proizvod je dizajniran i izgrađen u skladu s pogrešnim zahtjevima. Ako testiramo proizvod, on će zadovoljiti zahtjeve i dizajn, i proći svoje testove, ali može biti odbijen od strane korisnika ili kupca. Nedostaci koje je korisnik prijavio prilikom korištenja mogu biti vrlo skupi. Nažalost, nedostaci u dizajnu i zahtjevima nisu rijetki. Procjenjuje se da nedostaci uvedeni tijekom definiranja zahtjeva i dizajna čine gotovo polovicu ukupnog broja nedostataka.[10] Uz utjecaj grešaka koji proizlaze iz nedostataka koje nismo pronašli, moramo uzeti u obziri i utjecaj otkrivanja tih nedostataka. Trošak pronalaženja i popravljanja grešaka značajno raste tijekom životnog ciklusa proizvoda tj. programskog rješenja. Nedostatak unutar programskog rješenja jednostavnije je i jeftinije ukloniti što ga se ranije u njegovom životnom ciklusu pronađe. Nadovezujući se na Sliku 1, na Slici 2 vidimo da trošak ispravljanja pogreške raste u navedenim fazama.[11]



Slika 2. Rast troška ispravljanja grešaka[12]

[10] Usp. Isto, str.8.
[11] Usp. Isto. str.9.
[12] Usp. Isto.

2.2. Razine testiranja

Razina testiranja programskog rješenja odnosi se na proces u kojem se testira svaka jedinica ili komponenta programskog rješenja. Glavni cilj testiranja sustava je procjena usklađenosti sustava s navedenim potrebama. Postoji mnogo različitih razina testiranja koje pomažu u provjeri ponašanja i performansi testiranja programskog rješenja. No uglavnom svode se na četiri osnovne razine testiranja:

- Testiranje jedinica
- Integracijsko testiranje
- Testiranje sustava
- Testiranje prihvatljivosti

Testiranje jedinica (engl. *unit testing*) omogućuje testiranje svakog modula programskog rješenja zasebno. Jedinica je najmanji testirani dio sustava ili aplikacije koji se može kompajlirati, učitavati i izvršavati. Cilj je testirati svaki dio programskog rješenja tako da ga rastavimo na sastavne dijelove. Ono provjerava ispunjava li pojedina jedinica programskog rješenja svoju funkcionalnost ili ne. Ovu vrstu testiranja provode programeri. Integracijsko testiranje - integracija znači kombiniranje. Na primjer, u ovoj fazi testiranja, različiti programski moduli su kombinirani i testirani kao grupa kako bi se osiguralo da je integrirani sustav spreman za testiranje sustava. Integracijsko testiranje provjerava protok podataka iz jednog modula u druge. Ovu vrstu ispitivanja vrše stručni testeri. Testiranje sustava najčešće je završno ispitivanje kako bi se provjerilo zadovoljava li sustav specifikaciju. Ono procjenjuje funkcionalnu i nefunkcionalnu potrebu za ispitivanje. Testiranje sustava provodi se na cjelovitom, integriranom sustavu. To omogućuje provjeru sukladnosti sustava prema zahtjevima. Testiranje sustava testira ukupnu interakciju komponenti. To uključuje ispitivanje opterećenja, performansi, pouzdanosti i sigurnosti sustava u cjelini. Testiranje prihvatljivosti je test koji se provodi kako bi se utvrdilo jesu li ispunjeni zahtjevi specifikacije ili ugovora u isporuci programskog rješenja. Testiranje prihvatljivosti u osnovi obavlja korisnik. Međutim, u ovaj proces mogu biti uključeni i drugi sudionici.[13]

[13] Usp. Guru 99. URL: https://www.guru99.com/levels-of-testing.html (17.06.2019.)

Slika 3. Usporedba razina testiranja[14]

Osim navedenih razina postoje još i alfa, beta, tzv. *buddy* te regresijsko testiranje. Alfa testiranje je vrsta testiranja prihvatljivosti. Ono se koristi kako bi se identificirali svi mogući problemi prije objavljivanja proizvoda javnosti. Fokus ovog testiranja je simulirati stvarne korisnike pomoću metoda crne i bijele kutije (engl. *black and white box method*). Cilj je izvršiti zadatke koje tipični korisnik može izvršiti. Alfa testiranje se provodi u promatranom okruženju i obično su testeri interni zaposlenici organizacije. Ova vrsta testiranja naziva se alfa jer se provodi relativno rano, pri kraju razvoja programskog rješenja i prije beta testiranja. Beta testiranje obavljaju stvarni korisnici programskog rješenja u "stvarnom okruženju" i može se smatrati oblikom eksternog korisničkog testiranja prihvatljivosti. Beta verzija programskog rješenja objavljuje se ograničenom broju krajnjih korisnika proizvoda kako bi se dobila povratna informacija o kvaliteti proizvoda. Beta testiranje smanjuje rizike kvara proizvoda i osigurava povećanu kvalitetu proizvoda kroz provjeru valjanosti korisnika. To je konačni test prije slanja proizvoda korisnicima. Izravna povratna informacija od korisnika glavna je prednost beta testiranja.[15]

[14] Usp. Wacker, Mike: Just Say No to More End-to-End Tests, 22.04.2015. URL:https://testing.googleblog.com/2015/04/just-say-no-to-more-end-to-end-tests.html (17.06.2019.)
[15] Usp. Guru 99. URL: https://www.guru99.com/alpha-beta-testing-demystified.html (17.06.2019.)

Tablica 1. Usporedba alfa i beta testiranja.[16]

Alfa testiranje	Beta testiranje
Alfa testiranje obavljaju testeri koji su obično interni zaposlenici organizacije.	Beta testiranje izvode klijenti ili krajnji korisnici koji nisu zaposlenici organizacije.
Alfa testiranje izvodi se na mrežnoj lokaciji za razvojne programere	Beta testiranje provodi se na lokaciji klijenta ili krajnjeg korisnika proizvoda
Ispitivanje sigurnosti se ne provodi pri dubinskom Alfa testiranju.	Pouzdanost, sigurnost, robusnost provjeravaju se tijekom Beta testiranja.
Alfa testiranje uključuje tehnike bijele kutije (engl. ***white box***) i crne kutije (engl. ***black box***)	Beta testiranje obično koristi tehniku crne kutije
Alfa testiranje zahtijeva kontrolirano okruženje laboratorija ili okruženje za testiranje.	Beta testiranje ne zahtijeva kontrolirano okruženje ili testiranje u laboratoriju. Programsko rješenje je dostupno javnosti u stvarnom vremenu.
Za Alfa testiranje možda će biti potreban dugi ciklus izvođenja.	Za provedbu Beta testiranja potrebno je samo nekoliko tjedana.
U Alfa testiranju kritični problemi ili ispravci mogu se odmah dojaviti programerima.	Većina problema ili povratnih informacija prikupljeno Beta testiranjem implementirat će se tek u budućim verzijama proizvoda.
Alfa testiranje osigurava kvalitetu proizvoda prije prelaska na Beta testiranje.	Beta testiranje također je koncentrirano na kvalitetu proizvoda, ali okuplja korisničke doprinose o proizvodu i osigurava da je proizvod prilagođen stvarnim korisnicima.

Tzv. *buddy* testiranje je tehnika testiranja koju provode dva člana tima koji rade na istom programskom rješenju na istom uređaju. Jedan od članova tima će raditi sa sustavima (s tipkovnicom i mišem), a drugi će raditi bilješke i scenarije. Jedan član tima trebao bi biti tester, a drugi član razvojnog tima ili poslovni analitičar.[17] Regresijsko testiranje definira se kao vrsta testiranja programskog rješenja kako bi se potvrdilo da nedavna promjena programskog rješenja ili kôda nije negativno utjecala na postojeće značajke. Regresijsko testiranje nije ništa drugo nego

[16] Usp. Isto.

[17] Usp. Intense testing. URL: https://intensetesting.wordpress.com/2014/03/28/pair-testing-buddy-testing/ (18.06.2019.)

potpuni ili djelomični odabir već provedenih testnih slučajeva koji se ponovno izvršavaju kako bi se osiguralo da postojeće funkcionalnosti funkcioniraju na zadovoljavajući način. Ovo se testiranje provodi kako bi se osiguralo da nove promjene kôda ne donose nuspojave na postojeće funkcionalnosti. Ono osigurava da stari kôd i dalje radi nakon što se izvrši nova promjena kôda.[18]

2.3. Metode testiranja

Metode testiranja programskih rješenja dijele se na metode bijele, crne i sive kutije. Njima se opisuje aspekt s kojeg tester testira programsko rješenje, odnosno kreira testne slučajeve (engl. *test cases*).

2.3.1. Metoda bijele kutije

Bijela kutija je metoda testiranja programskog rješenja u kojoj je testeru poznata unutarnja struktura predmeta koji se testira. Tester bira načine testiranja kôda te određuje scenarij testnog procesa. Informatička znanja i znanja o implementaciji su od ključne važnosti. Metoda bijele kutije odnosi se na testiranje izvan korisničkog sučelja i na najvažnijim aspektima sustava. Ova metoda je tako nazvana jer je programsko rješenje, u očima testera, poput bijele kutije unutar koje se jasno vidi. Jedna od prednosti metode bijele kutije je da testiranje može započeti u ranijoj fazi. Nije potrebno čekati da GUI bude dostupan. Testiranje je temeljitije, s mogućnošću pokrivanja većine funkcionalnosti. Što se nedostataka tiče, budući da testovi mogu biti vrlo složeni potrebni su visokostručni resursi, uz temeljito poznavanje programiranja i provedbe testiranja. Održavanje test-skripti može biti opterećenje, ako se implementacija često mijenja. Budući da je ova metoda testiranja usko povezana s aplikacijom koja se testira, alati možda neće biti lako dostupni.[19]

2.3.2. Metoda crne kutije

Metoda crne kutije, također poznata kao bihevioralno testiranje je metoda testiranja programskog rješenja u kojoj tester ne poznaje unutarnju strukturu programskog rješenja koje se testira. Ovi testovi mogu biti funkcionalni ili nefunkcionalni, iako su obično funkcionalni. Ova metoda je tako nazvana jer je programsko rješenje u očima testera poput crne kutije unutar koje se ne može vidjeti. Ova metoda pokušava pronaći pogreške u sljedećim kategorijama:

- Neispravne ili nedostajuće funkcije
- Greške sučelja

---

[18] Usp. Guru 99. URL:https://www.guru99.com/regression-testing.html (18.06.2019.)

[19] Usp. Software Testing Fundamentals. URL:http://softwaretestingfundamentals.com/white-box-testing/ (18.06.2019.)

- Greške u strukturi podataka ili pristup vanjskoj bazi podataka
- Greške u „ponašanju“ ili izvedbi
- Greške inicijalizacije i završavanja

Prednosti ove metode jesu da se testovi provode s gledišta korisnika i pomažu u otkrivanju odstupanja u specifikacijama. Tester ne mora znati programske jezike ili na koji je način programsko rješenje implementirano. Testove može provoditi osoba neovisna od programera, omogućujući objektivnu perspektivu uz izbjegavanje pristranosti programera. Test slučajevi mogu biti osmišljeni čim se specifikacija proizvoda dovrši. Što se nedostataka tiče, može se testirati samo mali broj mogućih opcija, a mnoge opcije neće biti testirane. Bez jasnih specifikacija test slučajeve će biti teško dizajnirati. Testovi mogu biti suvišni ako je programer već pokrenuo testni slučaj.[20]

2.3.3. Metoda sive kutije

Metoda sive kutije je metoda testiranja programskih rješenja koja predstavlja kombinaciju metoda bijele i crne kutije. U metodi crne kutije, unutarnja struktura stavke koja se testira je nepoznata ispitivaču, dok je u metodi bijele kutije ona poznata. U testiranju u sivoj kutiji, unutarnja struktura je djelomično poznata. To uključuje pristup internim strukturama podataka i algoritmima u svrhu dizajniranja testnih slučajeva, dok testiranje ostaje na razini korisnika i crne kutije. Metoda sive kutije je tako nazvana, jer je programsko rješenje u očima testera nalik sivoj boji unutar koje se može tek djelomično vidjeti.[21]

Tablica 2. Usporedba metoda[22]

Metoda crne kutije	Metoda bijele kutije	Metoda sive kutije
Poznavanje interne radne strukture (kôda) nije potrebno za ovu vrstu ispitivanja. Za testne slučajeve potreban je samo GUI (grafičko korisničko sučelje).	Za ovu vrstu ispitivanja nužno je poznavanje interne radne strukture (kôda).	Djelomično je potrebno poznavanje interne radne strukture.

[20] Usp. Software Testing Fundamentals. URL:http://softwaretestingfundamentals.com/black-box-testing/ (18.06.2019.)
[21] Usp. Software Testing Fundamentals. URL:http://softwaretestingfundamentals.com/gray-box-testing/ (18.06.2019.)
[22] Usp. Java Point. URL:https://www.javatpoint.com/black-box-testing-vs-white-box-testing-vs-grey-box-testing (10.08.2019.)

Metoda crne kutije poznata je i kao funkcionalno testiranje, testiranje na temelju podataka i testiranje zatvorene kutije.	Metoda bijele kutije poznata je i kao strukturno testiranje, ispitivanje jasnih okvira, testiranje na bazi kôda i transparentnog testiranja.	Metoda sive kutije poznana je i kao prozirno ispitivanje, jer ispitivač ima ograničeno znanje kodiranja.
Pristup testiranju uključuje probne tehnike i metodu nagađanja o greškama, jer ispitivač nema znanje o internom kodiranju programskog rješenja.	Metoda bijele kutije nastavlja se provjerom granica sustava i domena podataka svojstvenih programskom rješenju, jer ne postoji nedostatak internog znanja o kodiranju.	Ako tester ima znanje kodiranja, tada se programsko rješenje provjerava validacijom podatkovnih domena i njegovih unutarnjih granica.
Prostor za testiranje tablica za ulaze (unosi koji će se koristiti za izradu testnih slučajeva) je enorman i najveći među svim "ispitnim prostorima".	Prostor za testiranje tablica za ulaze (unosi koji će se koristiti za izradu testnih slučajeva) manji je u usporedbi s metodom crne kutije.	Prostor za testiranje tablica za ulaze (unosi koji će se koristiti za izradu testnih slučajeva) manji je i od metoda bijele i crne kutije.
Vrlo je teško otkriti skrivene pogreške programskog rješenja, jer pogreške mogu nastati zbog internog rada koji je nepoznat u metodi crne kutije.	Lako je otkriti skrivene pogreške, jer se mogu dogoditi zbog internog rada koji je duboko istražen u metodi bijele kutije.	Teško je otkriti skrivenu pogrešku. Može se naći u testiranju na razini korisnika.
Ne uzima se u obzir za testiranje algoritama.	Prikladno i preporučuje se za testiranje algoritama.	Ne uzima se u obzir za testiranje algoritama.
Potrošnja vremena u metodi crne kutije testiranju ovisi o dostupnosti funkcionalnih specifikacija.	Metoda bijele kutije zahtijeva dugo vremena da se dizajniraju testni slučajevi zbog količine programskog kôda.	Dizajn testnih slučajeva može se obaviti u kratkom vremenskom razdoblju.
Tester, programer i krajnji korisnik mogu biti dio ispitivanja.	Samo tester i programer mogu biti dio ispitivanja; krajnji korisnik nije uključen.	Tester, programer i krajnji korisnik mogu biti dio ispitivanja.

To je najmanje dugotrajan proces od svih procesa testiranja.	Cjelokupni postupak testiranja zahtjeva najviše vremena od svih procesa testiranja.	Ova metoda testiranja zahtjeva manje vremena od testiranja u Bijeloj kutiji.
Otpornost i sigurnost protiv virusnih napada pokriveni su metodom crne kutije.	Otpornost i sigurnost protiv virusnih napada nisu obuhvaćeni metodom bijele kutije.	Otpornost i sigurnost protiv virusnih napada nisu obuhvaćeni metodom sive kutije.
Osnova ovog ispitivanja su vanjska očekivanja, interno ponašanje nije poznato.	Osnova ovog testiranja je kodiranje odgovorno za unutarnji rad proizvoda.	Ispitivanje na temelju dijagrama baza podataka na visokoj razini i dijagrama protoka podataka.
Manje je zahtjevna od metoda bijele i sive kutije.	Najzahtjevnija je metoda testiranja.	Djelomično zahtjevna; ovisi o vrsti test slučajeva koji se temelje na kodiranju ili GUI-u.

2.3.4. Vrste grešaka i njihova klasifikacija

Greška se može promatrati kao odstupanje u stvarnom radu programskog rješenja. "Neispravnost" u programskom rješenju odražava njegovu nesposobnost ili neučinkovitost u ispunjavanju specificiranih zahtjeva, odnosno da funkcionira na očekivani način. Budući da je primarna svrha testiranja ući u trag najvećem broju grešaka prisutnih u programskom rješenju, tester mora biti svjestan različitih oblika grešaka u programskom rješenju. Razumijevanje višestrukih varijanti grešaka pomaže testeru u prepoznavanju i pronalaženju velike većine grešaka - jednostavno i brzo.[23] Neke od najčešće poznatih grešaka u području testiranja su:

1. Greška korisničkog sučelja: To su greške koje se obično javljaju tijekom interakcije korisnika sa sustavom ili kada korisnik koristi programsko rješenje. Može se očitovati kao pogrešna funkcionalnost, nedostatak funkcionalnosti, pravopisna, činjenična i kontekstualna greška, neravnomjerni uzorak boje, netočna poruka, nedostupnost uputa na

[23] Usp. Professional QA.com. URL:http://www.professionalqa.com/types-of-defects-in-software-testing (20.06.2019.)

zaslonu, greška u izgledu izbornika, nepostojanje funkcije povratka, sporo reagiranje, nekompatibilnost s operativnim sustavom te česti prekidi rada.

2. Greške pri rukovanju greškama: Jedna od najčešćih vrsta grešaka jesu one koje se ne mogu unaprijed predvidjeti, pa se samim time od njih ne može niti zaštititi. Takve se greške očituju kao nedostatak zaštite od *bugova* u operativnom sustavu, nemogućnost prijave greške te nesposobnost rješavanja hardverskih kvarova.
3. Granične greške: to su greške koje se odnose na programsko rješenje s obzirom na njegovo funkcioniranje unutar i izvan izvjesnih granica. Očituje se u nevidljivim granicama, granicama u prostoru, vremenu i memoriji te nepravilnom upravljanju slučajevima izvan određenih granica kao što su numeričke ili vremenske granice.
4. Greške kontrole protoka: To su greške u pogledu prosljeđivanja kontrole programskog rješenja u pogrešnom smjeru tj. neočekivano ponašanje programskog rješenja u sljedećem koraku izvršenja. Očituju se kroz izvođenje beskonačne petlje, kojemu uzrok može biti pogrešna logika ili postavljeni uvjeti za izvođenje petlje, neograničeno čekanje za nevažeće stanje ili petlju, pogreške u tablicama s podacima, povratak na pogrešno stanje te izvještaj o sintaktičkoj pogrešci.
5. Greške hardvera: Očituju se kroz korištenje pogrešnog uređaja, nedostupnost uređaja, nespojivost s uređajem, pogrešan uređaj za pohranu, neispravna adresa uređaja, pogrešno dohvaćanje i tumačenje uputa za rukovanje uređajem te problemi s istekom vremena u obavljanju određene operacije (engl. *time-out*).
6. Kalkulacijske greške: To su greške proizašle iz računskih pogrešaka. Očituju se kao nepravilna implementacija računske logike, pogreške u korištenju operatora te greške koje se odnose na nedostatak preciznosti u izračunu.
7. Uvjeti otvaranja: To su greške koje se manifestiraju pokretanjem sljedećeg zadatka bez završetka prethodnog zadatka odnosno bez preduvjeta za izvođenje drugog zadatka.
8. Uvjet učitavanja: To su greške do kojih je došlo zbog nedostupnosti resursa, nedovoljne memorije za prilagodbu veličinama podataka zanemarujući zadatke niskog prioriteta za izvršenje.
9. Kontrola izvora, verzije i ID-a: Neočekivana pojava prethodnoga, starog *buga* zbog privitka s prethodnom verzijom. Očituje se tako što se programskom rješenju ne dodjeljuje naslov ili ime, pogrešna ili nepostojeća verzija u naslovu programskog rješenja, ne ažurira programski kôd na svim modulima gdje se koristi.
10. Greške testiranja: To su greške nastale tijekom provedbe i izvršenja procesa testiranja, vezane uz izradu dokumentacije, metode testiranja i izvješćivanje. Očituju se tako što nije

moguće izvršiti plan testiranja, pronaći *bugove*, prijaviti grešku ili kvar u sutavu. Isto tako, greške testiranja se odnose i na pogrešno interpretirane ili propuštene zahtjeve i specifikacije, nemogućnost reproduciranja identificiranih nedostataka, oštećene datoteke testnih podataka te neuspjeh u provjeri ispravaka.[24]

U području testiranja programskih rješenja, izrazi "ozbiljnost greške" i "prioritet u otklanjanju greške" su metrički alati povezani s greškama koji definiraju i opisuju greške s različitih gledišta. Ozbiljnost greške definira stupanj utjecaja greške na programsko rješenje. Što je "veća" ozbiljnost, veći je utjecaj na funkcionalnost programskog rješenja. Terminologija i značenje kod klasifikacije ozbiljnosti greške može varirati ovisno o ljudima, projektima, organizacijama ili alatima za praćenje grešaka, no najčešće se u literaturi javlja sljedeća klasifikacija:[25]

- **Kritično** (engl. *critical*) : Greška utječe na kritičnu funkcionalnost ili kritične podatke. Ne postoji zaobilazno rješenje. Primjer: Neuspješna instalacija predstavlja potpuni neuspjeh.
- **Značajan** (engl. *major*) : Greška utječe na glavne funkcionalnosti ili glavne podatke. Ima zaobilazno rješenje, koje nije očigledno. Primjer: Značajka nije funkcionalna iz jednog modula, ali zadatak je izvediv ako se u drugom modulu slijedi 10 kompliciranih indirektnih koraka.
- **Manji** (engl. *minor*) : Greška utječe na manje funkcionalnosti ili nekritične podatke. Jednostavno rješenje. Primjer: Manja značajka koja nije funkcionalna u jednom modulu, ali je lako izvediva iz drugog modula.
- **Trivijalno** (engl. *trivial*) : Greška ne utječe na funkcionalnost ili podatke. Ne treba čak ni zaobilazno rješenje. To ne utječe na produktivnost ili učinkovitost programskog rješenja. To je samo neugodnost. Primjer: sitne razlike u rasporedu, pravopisne / gramatičke pogreške i sl.[26]

Za razliku od ozbiljnosti greške, prioritet u otklanjanju greške definira grešku u smislu stupnja utjecaja na poslovne i organizacijske potrebe i zahtjeve, kao i hitnost popravljanja te greške. Što je veći prioritet, greška će se prije razriješiti.[27] Prioritet se može kategorizirati na sljedeće razine:

---

[24] Usp. Professional QA.com. URL: http://www.professionalqa.com/types-of-software-error (20.06.2019.)
[25] Usp. Professional QA.com. URL :http://www.professionalqa.com/severity-vs-priority (20.06.2019.)
[26] Usp. Software Testing Fundamentals. URL: http://softwaretestingfundamentals.com/defect-severity/ / (20.06.2019.)
[27] Usp. Professional QA.com. URL :http://www.professionalqa.com/severity-vs-priority (21.06.2019.)

- **Hitno**: Greška se mora odmah otkloniti
- **Važno**: Mora biti ispravljena u svakom od nadolazećih izdanja, ali bi trebala biti uključena u svaku buduću verziju.
- **Srednje**: Greška se može popraviti nakon izdavanja / u sljedećem izdanju.
- **Nisko**: Može, ali i ne mora biti ispravljena.[28]

Prioritizacijom grešaka treba pažljivo upravljati kako bi se izbjegla nestabilnost proizvoda, posebice kada postoji veliki broj nedostataka. Važno je uzeti u obzir da je prioritizacija subjektivna stvar ovisna o testeru ili proizvodu. Prioritizacija testnog slučaja najbolja je tehnika za osiguranje učinkovitosti i kvalitete proizvoda tijekom procesa razvoja i testiranja. To je metoda koja određuje prioritete i raspoređuje testne slučajeve prema njihovim najvišim i najnižim zahtjevima. Kroz određivanje prioriteta testnih slučajeva, testeri mogu najprije osigurati izvršavanje najvažnijih testnih slučajeva. Vjerojatnost greške, također poznata kao vidljivost, ukazuje na vjerojatnost da korisnik naiđe na grešku. Može biti:

- **Visoka**: Većina korisnika uočila je grešku.
- **Srednja**: 50% korisnika uočilo je grešku.
- **Niska**: Manji broj korisnika uočio je grešku.

Vjerojatnost greške također se može označiti u postocima (%). Mjera vjerojatnosti / vidljivosti odnosi se na korištenje funkcionalnosti, a ne cjelokupnog programskog rješenja. Dakle, greška u rijetko korištenoj funkcionalnosti može imati veliku vjerojatnost ako se greška lako susreće. Slično tome, greška u široko korištenoj značajki može imati malu vjerojatnost ako ju korisnici rijetko detektiraju.[29]

2.4. Vrste testiranja

2.4.1. Ručno testiranje

Ručno testiranje je vrsta testiranja programskih rješenja gdje testeri ručno izvršavaju testne slučajeve bez upotrebe bilo kojeg alata za automatizaciju.[30] Ručno testiranje je najprimitivnije od

[28] Usp. Software Testing Fundamentals. URL: http://softwaretestingfundamentals.com/defect-priority/ (21.06.2019.)
[29] Usp. Software Testing Fundamentals. URL: http://softwaretestingfundamentals.com/defect-probability/ (21.06.2019.)
[30] Usp. Guru 99. URL: https://www.guru99.com/manual-testing.html (21.06.2019.)

svih vrsta testiranja koji pomažu u pronalaženju grešaka u programskom rješenju. Sve nove aplikacije moraju se ručno testirati prije automatskog testiranja. Ručno testiranje zahtijeva više napora, ali je potrebno za provjeru izvedivosti automatizacije. Ono ne zahtijeva poznavanje alata za testiranje. Jedna od temeljnih značajki testiranja programskih rješenja jest da potpuno automatsko testiranje nije moguće što ručno testiranje dovodi u prvi plan. Ključni koncept ručnog testiranja je osigurati da je programsko rješenje bez grešaka i da radi u skladu s navedenim funkcionalnim zahtjevima. Test slučajevi su dizajnirani tijekom faze testiranja i trebali bi imati sto postotnu pokrivenost testom. Oni također osiguravaju da su prijavljeni nedostaci utvrđeni od strane programera, dok su ponovno testiranje obavili testeri na fiksnim greškama. Uglavnom, ovo testiranje provjerava kvalitetu sustava i isporučuje kupcu proizvod bez grešaka. Ručno testiranje izvršava se u nekoliko koraka:

1. Pročitati i razumjeti projektnu dokumentaciju te proučiti programsko rješenje.
2. Kreirati nacrt test slučajeva koji pokrivaju sve zahtjeve navedene u dokumentaciji.
3. Pregledati testne slučajeve s vodećim timom i klijentom (prema potrebi)
4. Izvršiti testne slučajeve
5. Prijaviti pogreške
6. Nakon ispravljanja grešaka, ponovno pokrenuti neuspješne testne slučajeve te provjeriti jesu li se sada uspješno izvršili.[31]

2.4.2. Automatsko testiranje

Automatsko testiranje uključuje korištenje programskog rješenja koje testira kontrolu izvršavanja testova i usporedbu stvarnih ishoda s predviđenim ishodima.[32] Ova vrsta testiranja smatra se veoma učinkovitom, jer jednom kreirane test slučajeve možemo opetovano koristiti. Automatizacija testiranja podrazumijeva korištenje alata za automatizaciju za izvršavanje test slučaja. Programsko rješenje za automatizaciju također može unijeti podatke za testiranje u sustav, usporediti očekivane i stvarne rezultate i generirati detaljna izvješća o ispitivanju. Test zahtijeva znatna ulaganja novca i resursa. Uzastopni razvojni ciklusi zahtijevat će ponavljanje istog skupa testova. Pomoću alata za automatizaciju moguće je snimiti paket za testiranje i ponovno ga reproducirati prema potrebi. Jednom kada je paket za testiranje automatiziran, nije potrebna nikakva ljudska intervencija. Cilj

[31] Usp. Guru 99. URL: https://www.guru99.com/manual-testing.html (21.06.2019.)

[32] Usp. Kolawa, Adam; Huizinga, Dorota (2007). Automated Defect Prevention: Best Practices in Software Management. Wiley-IEEE Computer Society Press. p. 74. ISBN 978-0-470-04212-0. URL: https://epdf.pub/automated-defect-prevention-best-practices-in-software-management44993.html

automatizacije je smanjiti broj testnih slučajeva koji se ručno pokreću, a ne i eliminirati ručno testiranje u potpunosti. Automatizirano testiranje programskog rješenja važno je zbog toga što je ručno testiranje svih polja i svih negativnih scenarija gubitak vremena i novca. Na primjer, teško je ručno testirati višejezične stranice, dok automatizacija u tom slučaju ne zahtijeva ljudsku intervenciju. Automatizirani test se također može pokrenuti bez nadzora te on povećava brzinu izvršenja testa i pomaže u povećanju pokrivenosti programskog rješenja testom. Osim toga, ručno testiranje može postati dosadno i stoga sklono greškama. Testni slučajevi koje treba automatizirati jesu visoko rizični testovi, testni slučajevi koji se ponavljaju, testni slučajevi koji su vrlo dosadni ili teško izvodljivi ručno te testni slučajevi koji zahtijevaju puno vremena. S druge strane testni slučajevi koji nisu prikladni za automatizaciju jesu slučajevi koji su novo dizajnirani i ne izvršavaju se ručno barem jednom, slučajevi za koje se zahtjevi često mijenjaju te testni slučajevi koji se izvode na *ad hoc* osnovi.[33]

2.4.3. Usporedba ručnog i automatskog testiranja

Ručno testiranje daje brze i točne vizualne povratne informacije, što je jeftinije, jer se ne mora trošiti vlastiti budžet na alate i procese automatizacije. Ljudska prosudba i intuicija uvijek dolaze do izražaja u ručnom testiranju. U slučaju testiranja neznatne značajke programskog rješenja, automatsko testiranje zbog kodiranja moglo bi biti dugotrajno dok se ručno takva značajka može testirati u hodu. S druge strane, ručno testiranje je manje pouzdana metoda testiranja jer ju provodi čovjek. Uvijek je sklona greškama. Postupak ručnog testiranja ne može se snimiti, tako da nije moguće ponovno koristiti ručno testiranje. Za vrijeme ručnog testiranja, određene zadatke je čak teško izvesti ručno, što može zahtijevati dodatno vrijeme testiranja programskog rješenja.[34]
Automatizirano testiranje pak u pravilu pronalazi više grešaka u usporedbi s ljudskim testerima. Budući da je veći dio postupka testiranja automatiziran, to je brz i učinkovit proces. Automatizacijski proces se može zabilježiti što omogućuje ponovno korištenje i izvršavanje istih vrsta operacija testiranja. On se provodi pomoću programskih alata, tako da se odvija bez pomoći čovjeka. Lako može povećati produktivnost jer omogućuje brz i točan rezultat testiranja. Automatizirano testiranje podržava različite aplikacije. Pokrivenost programskog rješenja automatskim testiranjem znatno se povećava. No, bez ljudskog elementa, teško je dobiti uvid u vizualne aspekte korisničkog sučelja kao što su boje, font i veličine slova, kontrast ili veličine gumba. Alati za automatsko testiranje mogu biti skupi, što može povećati troškove testiranja. Svaki alat za automatizaciju testiranja ima svoja ograničenja koja smanjuju opseg automatizacije.

[33] Usp. Guru 99. URL: https://www.guru99.com/automation-testing.html (21.06.2019.)
[34] Usp. Guru 99. URL: https://www.guru99.com/difference-automated-vs-manual-testing.html (21.06.2019.)

Ispravljanje pogrešaka testne skripte je još jedan veliki problem u automatskom testiranju, a i održavanje testova je skupo.[35]

Tablica 4. Usporedba ručnog i automatskog testiranja[36]

Ručno testiranje	Automatsko testiranje
Ručno testiranje nije uvijek točno zbog ljudske pogreške, stoga je manje pouzdano.	Automatizirano testiranje je pouzdanije, jer se izvodi pomoću alata i / ili skripti.
Ručno testiranje iziskuje puno vremena i korištenje ljudskih resursa.	Automatizirano testiranje provodi se programskim alatima, tako da je znatno brže od ručnog pristupa.
Potrebno je ulagati u ljudske resurse.	Potrebno je ulagati u alate za testiranje.
Ručno testiranje praktično je samo kad se testni slučajevi izvode jednom ili dva puta i nije potrebno učestalo ponavljanje.	Automatizirano testiranje praktična je opcija kada se testni slučajevi ponavljaju više puta u dužem vremenskom razdoblju.
Ručno testiranje omogućava čovjeku promatranje, što može biti korisnije ako je cilj prilagođenost korisnicima ili poboljšano korisničko iskustvo.	Automatizirano testiranje ne uključuje promatranje ljudi i ne može jamčiti pozitivno korisničko iskustvo.

2.5. Kvaliteta programskog rješenja koje se testira

Proces testiranja osigurava pouzdanost u valjanost programskog rješenja ako se utvrdi da ono ima manje ili nikakve nedostatke, pod uvjetom da smo zadovoljni samim testom pod kojim se programsko rješenje testiralo. Ako test nije dobro osmišljen potencijalno nedostaci mogu ostati unutar programskog rješenja, a osobe koje su sudjelovale u procesu kreiranja određenog programskog rješenja mogu zadobiti osjećaj lažne sigurnosti. Dobro osmišljen test otkrit će sve ili barem većinu grešaka i ako programsko rješenje prođe takav test, s pravom ćemo biti sigurniji u njega i moći tvrditi da je ukupna razina rizika korištenja sustava smanjena. Kada testiranje otkrije nedostatke i kada su ti nedostaci uklonjeni, kvaliteta programskog rješenja se povećava, pod uvjetom da je postupak uklanjanja izveden ispravno.[37] Projekti nastoje isporučiti programska

[35] Usp. Isto.
[36] Usp. Apica. URL: https://www.apicasystems.com/blog/automated-testing-vs-manual-testing/ (12.8.2012.)
[37] Usp. Graham, Dorothy [et.al.]. Foundations of Software Testing ISTQB Certification. UK: Gaynor Redvers-Mutton, 2008. str.10.

rješenja prema određenim specifikacijama. Kako bi projekt isporučio ono što klijent zahtjeva, potrebno je odrediti specifikaciju programskog rješenja koje isporučeno programsko rješenje mora zadovoljavati. Kako bi klijent bio zadovoljan, ono mora biti dostavljeno i unutar dogovorenog vremenskog roka a trošak izrade i dostave unutar dogovorenog budžeta. Važno je da projektni tim, klijenti i bilo koji drugi sudionici u projektu odrede i dogovore vlastita očekivanja. Važno je razumjeti što klijenti podrazumijevaju pod kvalitetom i kakva su njihova očekivanja. Programeri i testeri kao kvalitetu vide programsko rješenje koje zadovoljava definiranu specifikaciju, i koje ima tek nekoliko grešaka. Ono što oni vide kao kvalitetu možda neće pružiti kvalitetno rješenje za klijente. Nadalje, ako se klijenti dovedu u situaciju da moraju potrošiti više novca nego što su to planirali ili da im programsko rješenje ne pomaže u izvršavanju njihovih zadataka, neće biti impresionirani tehničkom izvrsnošću programskog rješenja. Tablica 1. sažima i objašnjava stavove i očekivanja u pogledu kvalitete programskog rješenja koristeći 'proizvodnju i kupnju rajčice' kao analogiju za 'proizvodnju i kupnju programskog rješenja'. Analizirajući tablicu jasno je da pristup testiranju ovisi o tome koje se gledište preferira.[38]

Tablica 3. Stajališta, očekivanja i kvaliteta programskog rješenja u usporedbi s kvalitetom rajčica.[39]

Stajalište	Softver	Rajčice
Kvaliteta se mjeri pogledom na atribut proizvoda.	Izmjerit će se atributi programskog rješenja, npr. njegovu pouzdanost u smislu srednjeg vremena između kvarova (MBTF) i izdanja kada dosegne određenu razinu, npr. MTBF od 12 sati.	Rajčice su prave veličine i oblika za pakiranje za supermarket. Rajčice imaju dobar ukus i boju.
Kvaliteta je pogodnost za upotrebu. Kvaliteta može imati i subjektivne aspekte, a ne samo kvantitativne.	Korisnici ocjenjuju mogu li izvršavati svoje zadatke; ako da, proizvod će se dati korisnicima na uporabu.	Rajčice odgovaraju našem receptu.
Kvaliteta se temelji na dobrim proizvodnim procesima i udovoljava definiranim zahtjevima. Mjeri se ispitivanjem, pregledom i analizom kvarova.	Koristit će se uobičajeni postupak razvoja programskog rješenja. Programsko rješenje će se dati korisnicima na korištenje samo ako postoji	Rajčice se organski uzgajaju. Rajčice nemaju mrlje i nemaju oštećenja od štetočina.

[38] Usp. Isto, str.11.
[39] Usp. Isto.

	manje od pet otvorenih nedostataka visokog prioriteta nakon što su planirani testovi provedeni.	
Očekivana vrijednost za uloženi novac. Pristupačnost i kompromis između vremena, napora i troškova temelji se na vrijednosti. Može se priuštiti kupnja programskog rješenja i očekuje se povrat ulaganja.	Vrijeme za testiranje je dva tjedna kako bi se ostalo u proračunu projekta.	Rajčice imaju dobar rok trajanja. Vrijednost rajčice odgovara uloženom novcu.
Subjektivni osjećaji - ovdje se radi o osjećajima pojedinca ili grupe pojedinaca spram proizvoda ili dobavljača.	Programsko rješenje je zadovoljavajuće! Zabavno je! Pa što ako ima nekoliko problema? Svejedno ga što prije treba prepustiti korisnicima na uporabu... Ugodan tim i radna atmosfera. Bilo je nekoliko problema - riješili su se stvarno brzo – postoji povjerenje u tim.	Rajčice se nabavljaju s malog lokalnog uzgajališta i odnos s uzgajivačima je dobar.

Jasno je da testiranje pomaže pronaći nedostatke i poboljšati kvalitetu programskog rješenja. No valja se zapitati koliko je testiranja dovoljno? Testirati se može sve, ništa ili samo određene dijelove programskih rješenja. Logičan odgovor je ako se želi valjano programsko rješenje, potrebno je testirati programsko rješenje u svakom mogućem aspektu. Ono što je bitno razmotriti odnosi se na dvojbu je li uopće moguće pojedino programsko rješenje testirati u potpunosti. Koliko bi testova trebalo napraviti da bi se u potpunosti testirali jednoznamenkasta numerička polja? Postoji 10 mogućih važećih numeričkih vrijednosti, ali postoje i vrijednosti koje trebamo proglasiti nevažećima. Postoji 26 velikih slova, 26 malih slova, najmanje 6 posebnih znakova i znakova interpunkcije, kao i prazna vrijednost. Tako bi barem 68 testova trebalo izraditi za testiranje jednoznamenkastog polja. Takvo testiranje zahtjeva veći budžet i više vremena, što klijentima većinom ne odgovara. Umjesto toga potreban nam je testni pristup koji osigurava prikladnu "količinu" testiranja u skladu s danim projektima. To se čini usklađivanjem procesa testiranja s procjenama rizika za klijente, projektni tim i programsko rješenje. Procjena i upravljanje rizikom jedna je od najvažnijih aktivnosti u bilo kojem projektu i predstavlja ključni razlog za provedbu

testiranja. U odlučivanje o tome u kojoj mjeri je potrebno provesti testiranje treba uzeti u obzir razinu rizika, uključujući tehničke i poslovne rizike vezane uz proizvod i ograničenja projekta, kao što su vrijeme i budžet. Nakon procjene rizika potrebno je uložiti napor u osiguranje kvalitete i aktivnosti testiranja s obzirom na rizike i troškove povezane s projektom. Zbog ograničenja u budžetu, vremenu i testiranju na temelju rizika moramo odlučiti i na što ćemo se usredotočiti u procesu testiranja.

2.6. Proces testiranja

2.6.1. Uvodna razmatranja

Iako je provedba testova važna, potreban je i plan djelovanja i izvješće o ishodu testiranja. Projektni i testni planovi trebaju uključivati vrijeme planiranja testova, dizajniranje testnih slučajeva, pripremu za izvršenje i ocjenu statusa. Pojam temeljnog testnog procesa razvijao se tijekom godina gdje su uočene iste vrste glavnih aktivnosti, iako mogu postojati i određene formalne različitosti. Aktivnosti unutar temeljnog procesa testiranja možemo podijeliti na sljedeće osnovne korake:

- planiranje i kontrola;
- analiza i projektiranje;
- provedba i izvršenje;
- ocjenjivanje izlaznih kriterija i izvješćivanja;
- aktivnosti zatvaranja testova.[40]

Te aktivnosti logično slijede jedna iza druge, ali se u određenom projektu mogu preklapati, odvijati istodobno ili se čak ponavljati.

2.6.2. Planiranje i kontrola

Tijekom koraka planiranja, osiguravamo razumijevanje ciljeva klijenata i projekta te rizika na koje se testiranje namjerava usredotočiti. To rezultira misijom testiranja ili zadatkom testa. Na temelju tog razumijevanja, postavljamo ciljeve samog testiranja te osmišljavamo pristup i plan testova, uključujući specifikaciju aktivnosti testiranja. Moguće je koristiti i unaprijed određenu strategiju testiranja ili politiku testiranja. Politika testiranja osigurava pravila, dok strategija testiranja osigurava sveobuhvatni pristup procesu testiranja na visokoj razini. Ako su politika i strategija već

[40] Usp. Graham, Dorothy [et.al.]. Foundations of Software Testing ISTQB Certification. UK: Gaynor Redvers-Mutton, 2008. str.23.

definirani, oni pokreću planiranje, ali ako nisu, važno je zahtijevati da budu navedeni i definirani. Korak planiranja procesa testiranja ima sljedeće glavne zadatke:

1. Odrediti opseg, rizike i ciljeve testiranja: definirati komponente za testiranje te poslovne, proizvodne, projektne i tehničke rizike kojima se treba pozabaviti; odrediti testira li se programsko rješenje kako bi se otkrile greške ili kako bi se izmjerila kvaliteta programskog rješenja.
2. Odrediti pristup testu (tehnike, testne stavke, pokrivenost, identificiranje i povezivanje s timovima koji sudjeluju u testiranju, alate koji će se koristiti); razmotriti kako će se provesti testiranje, koje će se tehnike koristiti, što je potrebno testirati i koliko opsežno. Definirati tko se treba uključiti u testiranje i kada (ovo može uključivati programere, korisnike, timove IT infrastrukture); odlučiti što će se proizvoditi kao dio testiranja (npr. testni programi kao što su testni postupci i testni podaci). Posljednje se odnosi i na zahtjeve strategije testiranja.
3. Provođenje testne politike i/ili testne strategije: tijekom planiranja važno je osigurati da je ono što je planirano ujedno u skladu s politikom i strategijom testiranja.
4. Odrediti potrebne resurse testiranja (npr. ljude, testno okruženje, računala): od planiranja koje je već obavljeno sada se može "ući" u detalje; odrediti članove tima koji provodi testiranje i uspostaviti potrebne hardverske i softverske uvjeti u testnom okruženju.
5. Odrediti raspored testnih analiza i dizajniranja zadataka, provedbu testa, izvršenje i evaluaciju: potrebno je odrediti i raspored svih zadataka i aktivnosti, kako bi se moglo pratiti i osigurati da se testiranje dovrši u predviđenom vremenu.
6. Odrediti kriterije izlaza: važno je uspostaviti kriterije kao što su kriteriji pokrivenosti koji će nam pomoći pratiti ispravnost izvršavanja testova. Kriteriji pokrivenosti ukazat će na zadatke koje moramo provesti za određenu razinu testiranja prije nego što možemo reći da je testiranje završeno.[41]

S druge strane, testna kontrola je stalna aktivnost. Potrebno je usporediti stvarni napredak s planiranim napretkom i izvijestiti voditelja projekta i klijenta o trenutnom stanju testiranja, uključujući sve promjene ili odstupanja od plana. Postoji realna mogućnost za poduzimanje mjera gdje je to potrebno kako bi se ciljevi projekta ispunili. Takve akcije mogu značiti i promjenu izvornog plana, a što se u praksi često događa. Testna kontrola ima sljedeće glavne zadatke:

[41] Usp. Isto, str.24.

1. Mjerenje i analiza rezultata testiranja: Važno je znati koliko je testova obavljeno, kao i vrstu i važnost pronađenih nedostataka.
2. Pratiti i dokumentirati napredak, pokrivenost testova i kriterije izlaza: Važno je informirati projektni tim o tome koliko je testova obavljeno, koji su njihovi rezultati a koji zaključci i procjena rizika. Rezultati testova moraju biti vidljivi cijelom timu.
3. Navesti informacije o testiranju: pisati redovita izvješća voditelju projekta, sponzoru projekta, klijentu i drugim ključnim sudionicima kako bi im se pomoglo u donošenju odluka o statusu projekta.
4. Korektivne radnje: postrožiti kriterije izlaza za popravljene nedostatke, i zatražiti ulaganje više napora kako bi se pogreške ispravile; ujedno, odrediti prioritetne greške.
5. Donositi odluke: na temelju informacija prikupljenih tijekom testiranja donosit će se odluke ili će se omogućiti drugima da ih donose.[42]

2.6.3. Analiza i projektiranje

Analiza i projektiranje procesa testiranja predstavlja aktivnost u kojoj se opći ciljevi testiranja preoblikuju u opipljive uvjete testiranja i projektiranja testova. Tijekom analize i projektiranja testova, koristimo opće ciljeve testiranja koji su identificirani tijekom planiranja postupaka testiranja. Analiza i dizajniranje testova ima sljedeće glavne zadatke:

1. Provjeriti osnovu za testiranje ispitujući specifikaciju programskog rješenja koje se testira. Može se početi s dizajniranjem određenih vrsta testova (koji se nazivaju testovi crne kutije) prije nego što se napiše odgovarajući kôd za potrebe testiranja.
2. Identificirati uvjete testiranja na temelju analize testnih stavki, njihovih specifikacija i onoga što se zna o njihovom ponašanju i strukturi. To će imati za posljedicu kvalitetan popis onoga što je važno u procesu testiranja.
3. Osmisliti testove koristeći tehnike koje pomažu u odabiru reprezentativnih testova koji se odnose na određene aspekte programskog rješenja te koji nose rizike ili koji su od posebnog interesa.
4. Procjena *testabilnosti*[43] zahtjeva i sustava. Zahtjevi moraju biti napisani na način koji testeru omogućuje dizajniranje testova; na primjer, ako je pojedina značajka programskog rješenja važna, test treba osmisliti na način koji to može provjeriti. Na primjer, ako zahtjev

[42] Usp. Isto, str.25.
[43] Isplativosti testiranja

glasi "programsko rješenje mora dovoljno brzo odgovoriti", to se ne može testirati, jer 'dovoljno brzo' može značiti različite stvari različitim ljudima. Testabilnost sustava ovisi o različitim aspektima poput onoga je li moguće postaviti sustav u okruženju koje odgovara operativnom okruženju i može li se razumjeti i testirati sve načine na koje se sustav može konfigurirati ili koristiti.

5. Dizajnirati testno okruženje i identificirati svu potrebnu infrastrukturu i alate. To uključuje alate za testiranje i alate za podršku kao što su proračunske tablice, alati za planiranje projekata i alati i oprema izvan IT-a.[44]

2.6.4. Provedba i izvršenje

Tijekom implementacije i izvođenja testa, preuzimaju se uvjeti testiranja i stavljaju se u testne slučajeve i testna pomagala koji se, onda, postavljaju u testno okruženje. Uvjete za testiranje pretvaramo u testne slučajeve i postupke ili druge testove kao što su skripte za automatizaciju. Također moramo uspostaviti okruženje u kojem ćemo izvoditi testove i izgrađivati naše testne podatke. Provedba i izvedba testa imaju sljedeće glavne zadatke:

Implementacija:

1. Razvoj i određivanje prioritetnih testnih slučajeva te kreiranje testnih podataka za testove. Također, važno je napisati upute za provođenje testova (tzv. test procedure).
2. Izrada testnih paketa iz testnih slučajeva za učinkovito izvršenje testa. Paket za testiranje je logična zbirka testnih slučajeva koji uobičajeno rade zajedno. Testni paketi često dijele podatke i zajednički skup ciljeva na visokoj razini. Također, važno je isplanirati raspored testnog izvođenja.
3. Implementirati i provjeravati okruženje.

Izvršenje:

1. Izvršiti testne pakete i pojedinačne testne slučajeve, prateći testne postupke. To se može učiniti ručno ili pomoću odgovarajućih alata.
2. Prijaviti ishod izvršenja testa i zabilježiti verzije programskog rješenja koje se testira te nazive testnih alata. Važno je znati koji su se testovi provodili na određenoj verziji programskog rješenja kako bi se nedostaci prijavili u odnosu na te točno određene verzije.

---

[44] Usp. Graham, Dorothy [et.al.]. Foundations of Software Testing ISTQB Certification. UK: Gaynor Redvers-Mutton, 2008. str.26.

3. Usporedba stvarnih rezultata s očekivanim rezultatima.
4. Ako postoje razlike između stvarnih i očekivanih rezultata, potrebno je prijaviti odstupanja kao incidente. Oni se nadalje analiziraju kako bi se prikupile dodatne pojedinosti o greški, identificirali uzroci greške te kako bi se razlikovali problemi u programskom rješenju i drugim proizvodima koji se testiraju od problema koji se odnose na bilo kakve nedostatke u testnim podacima, testnim dokumentima ili greškama u načinu na koji je test izveden.
5. Ponoviti aktivnosti testiranja kao rezultat radnji poduzetih za svako odstupanje. Ispravljeni testovi izvode se ako postoje nedostaci u prijašnjim testovima. Ponovno se testira ispravljeno programsko rješenje kako bi se osiguralo da je greška uistinu ispravljena i da programeri nisu unosili greške u nepromijenjenim područjima programskog rješenja, kao i da popravljanje kvara nije otkrilo druge nedostatke.[45]

2.6.5. Ocjenjivanje izlaznih kriterija i izvješćivanje

Ocjenjivanje izlaznih kriterija je aktivnost u kojoj se izvršenje testa procjenjuje u odnosu na definirane ciljeve. To bi trebalo učiniti za svaku razinu ispitivanja, jer je za svaku razinu važno znati je li testiranje provedeno u dovoljnoj mjeri. Na temelju procjene rizika, postavljaju se kriteriji prema kojima se određuje "dovoljna mjera" testiranja. Ti kriteriji razlikuju se za svaki projekt te su poznati kao izlazni kriteriji. Oni definiraju može li se određenu aktivnost testiranja deklarirati te treba li ih postaviti i procijeniti za svaku razinu ispitivanja. Ocjenjivanje izlaznih kriterija ima sljedeće glavne zadatke

1. Provjera log testova prema kriterijima navedenim u planiranju testa: traže se dokazi za izvršene i provjerene testove, te koje su se greške pojavile a koje su ispravljene.
2. Procjena je li potrebno više testova ili je potrebno izmijeniti zadane kriterije: možda se mora pokrenuti još testova ako se nisu pokrenuli svi testovi koji su kreirani ili ako se testiranjem nisu obuhvatili svi aspekti programskog rješenja.
3. Pisanje izvješća o ispitivanju: nije dovoljno da ispitivači znaju ishod testa. Svi sudionici testiranja trebaju znati koja su testiranja obavljena i koji je ishod testiranja kako bi mogli donositi potrebite odluke o programskom rješenju.[46]

---

[45] Usp. Isto, str.27.

[46] Usp. Isto, str.28.

2.6.6. Aktivnosti zatvaranja testova

Tijekom aktivnosti zatvaranja testova prikupljaju se podaci iz završenih aktivnosti testiranja kako bi se objedinili rezultati testiranja, uključujući provjeru i popunjavanje testova, te analiziranje činjenica i brojeva. Aktivnosti testa zatvaranja uključuju sljedeće glavne zadatke:

1. Provjera jesu li sva izvješća o incidentima riješena popravkom kvara ili odgodom. Za odgođene se nedostatke može tražiti promjena u budućoj verziji programskog rješenja. Dokumentirati prihvaćanje ili odbijanje programskog rješenja.
2. Finalizirati i arhivirati dokumentaciju, kao što su skripte, testno okruženje i bilo koju drugu testnu infrastrukturu za kasniju ponovnu upotrebu. Važno je ponovno upotrijebiti sve što je moguće, jer samo tako štedimo trud i vrijeme testiranja.
3. Predaja testnog programskog rješenja organizaciji za održavanje koja će održavati programsko rješenje i izvršavati buduće ispravke grešaka.
4. Sveobuhvatna procjena provedenog procesa testiranja i analiza situacija za buduće verzije programskog rješenja i projekte. To može uključivati poboljšanja u životnom ciklusu razvoja programskog rješenja, kao i poboljšanja procesa testiranja.[47]

[47] Usp. Isto, str.29.

3. AUTOMATSKO TESTIRANJE MREŽNIH STRANICA FILOZOFSKOG FAKULTETA U OSIJEKU

3.1. Cilj i svrha automatskog testiranja mrežnih stranica FFOS-a

Svrha ovog rada je izraditi testove za automatsko testiranje mrežne stranice Filozofskog fakulteta u Osijeku koji će se voditi načelima struke kada je u pitanju testiranje programskih rješenja, odnosno osiguranju kvalitete (engl. *quality assurance*) istog. Automatski testovi izrađeni u okviru diplomskog rada namijenjeni su tome da postanu sastavni dio uobičajene procedure testiranja mrežnog mjesta Filozofskog fakulteta u Osijeku koje provodi informatička služba fakulteta.[48]

Dakako, testove je moguće nadograditi i proširiti kako bi obuhvatili kompletno mrežno mjesto te vanjske poveznice vezane uz fakultet kao što su *Moodle* te *Squirel Mail*. S obzirom da ovi testovi služe automatskom testiranju, nadalje će se dati preporuke na koji način i kada koristiti automatsko testiranje. Kao što je navedeno, automatsko testiranje je savršena opcija kada je potrebno testirati puno značajki u kratko vrijeme. Osim toga, automatsko testiranje je i prvi izbor kada je programsko rješenje potrebno regresijski testirati, odnosno u trenucima dok se ono aktivno koristi. Ako je programsko rješenje potrebno testirati dok ga rabi tisuće korisnika, automatsko testiranje i dalje predstavlja najbolju opciju. Upravo česte promjene funkcionalnosti u programskom rješenju su poželjni uvjeti u kojima je efikasnije koristiti se automatskim testiranjem. No, danas se automatsko testiranje još uvijek ne koristi u tolikoj mjeri. Početno ulaganje u uvođenje automatskog testiranja je skupo, jer osim zaposlenja stručnog kadra, potrebna je i dodatna obuka te resursi za održavanje istih. S druge strane, bilo bi utopijski misliti kako je programsko rješenje moguće testirati isključivo i u cijelosti samo automatskim testiranjem. Naravno, postoje područja kao što su testiranje performansi, regresijsko testiranje, testiranje opterećenja, gdje se može dostići visok stupanj automatizacije. No, područja poput korisničkog sučelja, dokumentacije, instalacije i kompatibilnosti se i dalje moraju ručno testirati. Najbolja praksa je kombinacija ručnog i automatskog testiranja.[49]

[48] Kôd automatskih testova dostupan je na https://github.com/endriina/DiplomskiRad_ffos_wdio.

[49] Usp. Software testing help. URL: https://www.softwaretestinghelp.com/10-tips-you-should-read-before-automating-your-testing-work/ (10.08.2019.)

3.2. Tehničke informacije

3.2.1. Uvodna razmatranja

Testovi su pisani u JavaScript programskom jeziku. U izradi testova korištena je Node.js platforma. Svrha Node.js platforme jest osigurati jednostavan način izrade programskih rješenja u JavaScript programskom jeziku. JavaScript izvršava se u internet pregledniku, no pomoću Node.js platforme on se pokreće na poslužiteljskoj razini.[50] Korišten je WebDriverIO kao glavni alat za automatsko testiranje.[51] WebDriverIO je *custom framework* izgrađen na temelju Selenium API-ja. To je skup programskih alata čija je svrha podrška automatskom testiranju mrežnih aplikacija. Selenium je programsko rješenje otvorenog kôda, jednostavno se integrira s drugim platformama, podržava sve važne internet preglednike te programske jezike. Od Selenium komponenti, korišten je Selenium WebDriver.[52] Za provjeru testova koristio se Chrome internet preglednik, a za pisanje automatskih testova koristio se Visual Studio Code. To je uređivač izvornog kôda koji je razvio Microsoft za Windows, Linux i macOS. Uključuje podršku za uklanjanje pogrešaka, ugrađenu Git kontrolu i GitHub, isticanje sintakse, dovršavanje kôda i preuređivanje kôda. Vrlo je prilagodljiv, omogućuje korisnicima da promijene temu, koriste prečace na tipkovnici te postave i instaliraju proširenja koja dodaju funkcionalnosti. Besplatan je i otvorenog kôda.[53]

3.2.2. Selenium

Selenium je skup različitih programskih alata koji imaju različit pristup podršci automatskom testiranju. Cjelokupni paket alata rezultira bogatim setom funkcija testiranja posebno prilagođenih potrebama testiranja mrežnih aplikacija svih vrsta. Ove su operacije vrlo fleksibilne te omogućuju mnogo opcija za pronalaženje UI elemenata i usporedbu očekivanih rezultata testiranja sa stvarnim „ponašanjem“ aplikacije. Jedna od glavnih karakteristika Seleniuma je podrška za izvršavanje testova na više preglednika. Selenium je prvi put zaživio 2004. godine kada je Jason Huggins testirao internu aplikaciju ThoughtWorks. Razvio je JavaScript knjižnicu koja bi mogla pokretati interakcije sa stranicom, što omogućava automatsko pokretanje testova na više preglednika. Ta je

---

[50] Usp. Raguž, Robetra. Prednosti i nedostaci uporabe node.js. platforme, Osijek, 2017. Str:1 URL: https://repozitorij.etfos.hr/islandora/object/etfos%3A1267/datastream/PDF/view (10.08.2019.)

[51] Usp. WebDriverIO. URL:https://webdriver.io/docs/gettingstarted.html (10.08.2019.)

[52] Usp. Selenium HQ. URL:https://www.seleniumhq.org (10.08.2019.)

[53] Usp. Lardinois, Frederic. Microsoft Launches Visual Studio Code, A Free Cross-Platform Code Editor For OS X, Linux And Windows. URL: https://techcrunch.com/2015/04/29/microsoft-shocks-the-world-with-visual-studio-code-a-free-code-editor-for-os-x-linux-and-windows/?guccounter=1&guce_referrer_us=aHR0cHM6Ly9lbi53aWtpcGVkaWEub3JnLw&guce_referrer_cs=zHbu753eJL-A3ARIbhN5vA (29.08.2019.)

knjižnica s vremenom postala Selenium Core, koja je u osnovi svih funkcionalnosti Selenium Remote Control (RC) i Selenium IDE.[54]

3.2.3. WebDriverIO

WebDriverIO je alat koji se koristi za izradu automatiziranih testova u pregledniku, a testove pokreće uz pomoć Node.js-a. Svoje zahtjeve niže razine pretvara u korisne naredbe sa sažetom sintaksom. WebdriverIO je u potpunosti izgrađen pomoću JavaScripta. Samim time, pojedinci koji znaju JS, mogu odmah početi pisati automatizaciju koristeći WebDriverIO dokumentaciju. WebDriverIO sintaksa relativno je jednostavna. U osnovi, element se dohvati pomoću selektora i na njemu se poziva metoda. Selektori odabiru HTML elemente prema id-u, klasi, xPathu, itd.[55] U ovom radu koristili smo XPath koji do željenog HTML elementa dolazi kretnjom kroz hijerarhijsku strukturu XML dokumenta. Koristi sintaksu koja nije XML tako da se može koristiti u vrijednostima URI-ja i XML atributa.[56] Nadalje, *Page Object pattern*[57] je konvencija koja se lako postavlja s WebdriverIO-om. *Page Object pattern* koristi se u automatizaciji testova za poboljšanje održavanja testova i smanjenje dupliciranja kôda. Testovi koriste metode kad god je potrebno kako bi stupili u interakciju s korisničkim sučeljem mrežne stranice. *Page Object pattern* omogućuje organiziranje selektora i funkcija u zasebnu strukturu te pozivanje istih u više različitih testova. Na taj način održavamo selektore na jednom mjestu, a koristimo ih na više mjesta što čini kôd čitljivim i lako održivim.[58] WebDriverIO dolazi i s pokretačem testova, a programsko rješenje za konfiguriranje pomaže u stvaranju konfiguracijske datoteke. Sadrži mnoštvo komentara i omogućuje prilagođavanje projekata. WebDriverIO dokumentacija je jasna, a postavljanje je relativno jednostavno. Nudi mnoštvo opcija za prilagodbu i koristi Selenium.[59]

---

[54] Usp. SeleniumHQ. URL: https://docs.seleniumhq.org/docs/01_introducing_selenium.jsp (28.08.2019.)

[55] Usp. WebDriverIO. URL: https://webdriver.io/docs/selectors.html (29.08.2019.)

[56] Usp. MDN web docs. URL: https://developer.mozilla.org/en-US/docs/Web/XPath (29.08.2019.)

[57] "Page object pattern je uzorak koji je postao popularan u automatskom testiranju i koristi se za poboljšanje održavanja ispitivanja i smanjenje dupliciranja kôda. Page object je objektno orijentirana klasa koja služi kao sučelje stranici aplikacije koja se testira (AUT).

[58] Usp. Medium. URL: https://medium.com/tech-tajawal/page-object-model-pom-design-pattern-f9588630800b (29.08.2019.)

[59] Usp. Medium. URL: https://medium.com/@specktackle/selenium-and-webdriverio-a-historical-overview-6f8fbf94b418 (28.08.2019.)

3.3. Instalacija programa za testiranje

Najprije, bilo je potrebno instalirati sve potrebne programe na računalo: Chrome internet preglednik, Node.js platformu[60] te Visual Studio Code kao uređivač kôda.[61] Upute za instalaciju koje slijede kreirane su za Windows operacijski sustav.

Najprije je potrebno napraviti direktorij u koji se želi pohraniti svoj projekt. Na primjer:

```
c:> Desktop> webdriver.
```

Zatim se otvori terminal, pozicionira se u mapu koja je upravo stvorena (webdriver) i utipka:

```
npm init.
```

Ova naredba inicira novi projekt, odnosno kreira package.json file za dani projekt.
U terminalu se instalira WebdriverIO tako da se upiše:

```
npm install webdriverio --save-dev.
```

Nakon toga, potrebno je konfigurirati WebdriverIO pokretanjem naredbe:

```
./node_modules/.bin/wdio config
```

Ubrzo nakon pokretanja naredbe, u terminalu će biti postavljen niz pitanja. Možete ostaviti zadane vrijednosti, no preporuča se prilagođavanje konfiguracije specifičnim potrebama projekta. U ovom slučaju, izmijenili smo sljedeće vrijednosti:

Reporter: spec
Service: selenium-standalone
Logging verbosity: warn
Base Url: https://www.ffos.unios.hr

[60] Node.js može se preuzeti s adrese https://nodejs.org/en/download/.
[61] Visual Studio Code može se preuzeti s adrese https://code.visualstudio.com/

Nakon uspješne konfiguracije, generira se wdio.config.js datoteka koja sadrži informacije o lokaciji testova, paketima za izvješće, preglednicima te ostalim konfiguracijskim pojedinostima. Testovi se kreiraju unutar test datoteke. Iz webdriver direktorija potrebno je pokrenuti sljedeću naredbu:

```
mkdir -p ./test/specs
```

Potom se kroz terminal kreira i otvara prvi basic.spec.js test

```
touch ./test/specs/basic.spec.js
```

U sljedećem koraku kopira se i sprema kôd testa:

```
const assert = require('assert');
describe('webdriver.io page', () => {
    it('should have the right title', () => {
        browser.url('https://webdriver.io');
        const title = browser.getTitle();
        assert.equal(title, 'WebdriverIO · Next-gen WebDriver test
framework for Node.js');
    });
});
```

Na kraju, prvi test pokreće se naredbom:

```
npm run test.
```

3.4. Podjela testova

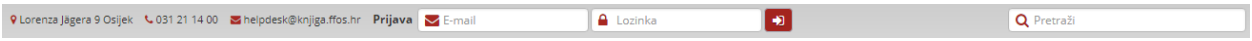
Mrežne stranice Filozofskog fakulteta u Osijeku za potrebe testiranja podijeljene su u devet kategorija koje se očituju u devet spec.js datoteka pri pisanju testova. Kategorije su formirane na temelju UI komponenata kojima pripadaju te su navedene u nastavku:

1. Zaglavlje (*Header*)
2. Gornji izbornik (*Top Menu*)

3. Glavni izbornik (*Main Menu*)
4. Klizač (*Slider*)
5. Izdvojene vijesti (*Featured News*)
6. Novosti i događanja (*News and Events*)
7. Poveznice (*Sidebar Links*)
8. Video (*Video*)
9. Podnožje (*Footer*)

Zaglavlje

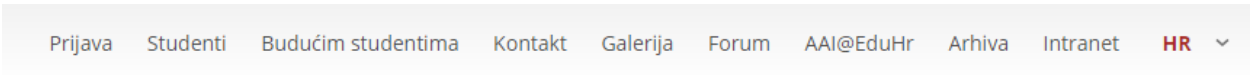
Unutar ove kategorije testirat će se učitavanje svih elemenata tablice (adresa, broj telefona, email, forma za prijavu, obrazac za pretraživanje). Za prva tri elementa te obrazac za pretraživanje testirat se samo učitavanje na glavnoj stranici www.ffos.unios.hr. Osim učitavanja, u formi za prijavu testirat će se oba unosa emaila i lozinke te gumb za prijavu.



Slika 4. Kategorija Zaglavlje

Gornji izbornik

Unutar ove kategorije također će se testirati učitavaju li se svi elementi (prijava, studenti, budućim studentima, kontakt, galerija, forum, AAI@EduHr, arhiva, intranet, izbor jezika). Za prijavu, arhivu i intranet će se testirati što se dogodi kada se klikne na taj element (prema specifičnim elementima na otvorenoj stranici te URL-u) te će se testirati forme koje se na tim stranicama pojavljuju. Kod elemenata studenti, budućim studentima, kontakt i galerija testirat će se pojavljuju li se padajući izbornici te što se događa kada se klikne na svaku stavku unutar padajućeg izbornika, prema URL-u.

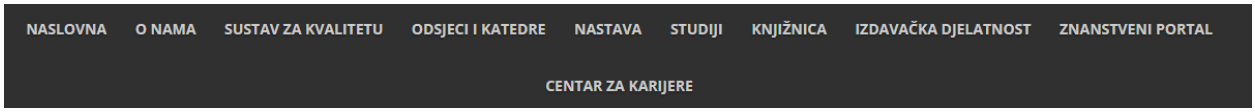


Slika 5. Kategorija Gornji izbornik

Glavni izbornik

Unutar ove kategorije testirat će se učitavanje elemenata (naslovna, o nama, sustav za kvalitetu, odsjeci i katedre, nastava, studiji, knjižnica, izdavačka djelatnost, znanstveni portal, centar za karijere). Kod elemenata o nama, odsjeci i katedre, nastava, studiji, izdavačka djelatnost te centar

za karijere testirat će se pojavljuju li se padajući izbornici te što se događa kada se klikne na svaku stavku unutar padajućeg izbornika. Kod ostalih elemenata testirat će se valjanost po URL-u.



Slika 6. Kategorija Glavni izbornik

Klizač

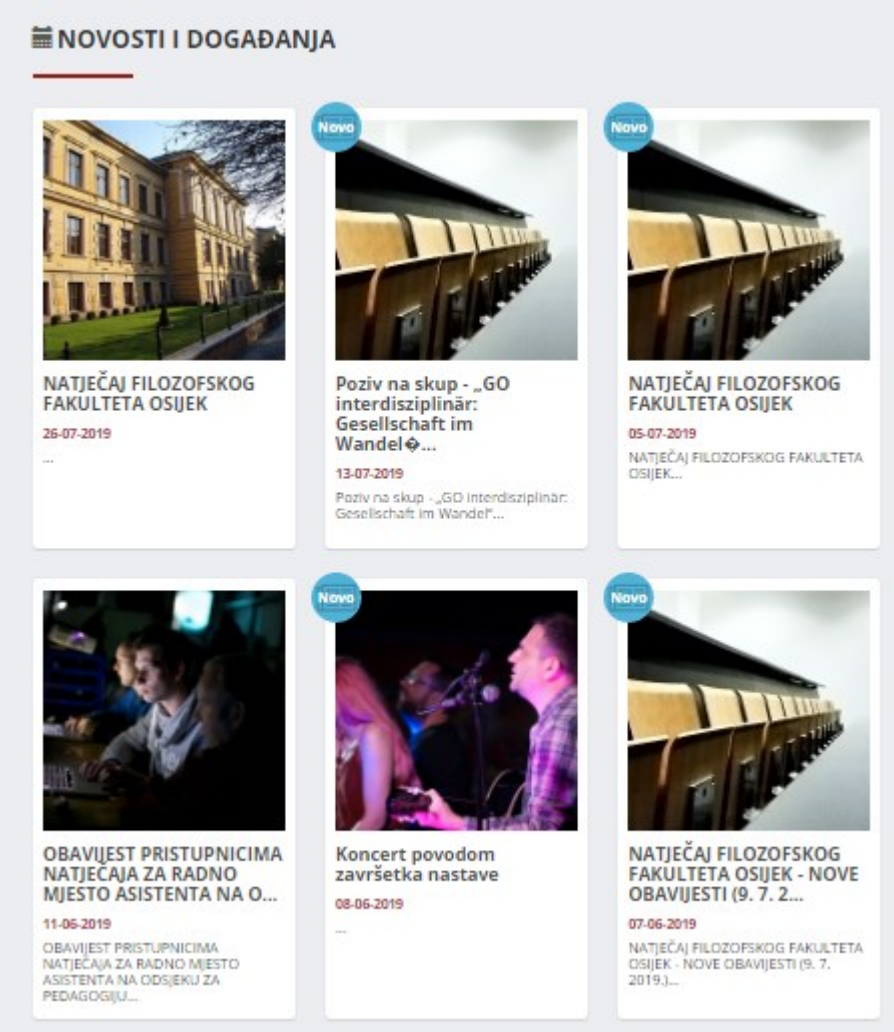
U ovoj kategoriji testirat će se učitavanje slike i naslova. Zatim će se testirati što se događa kada se na njih klikne, kao i na prethodnim elementima ostalih kategorija, prema specifičnim elementima te URL adresi.

Slika 7. Kategorija Slider

Izdvojene vijesti / Novosti i događanja

Kao i kod prethodnih kategorija, ovdje će se testirati učitavaju li se elementi koje čine slika, naslov i poveznica te kod *Novosti i događanja* opis. Također će se testirati što se dogodi kada se klikne na poveznicu, prema promjeni URL adrese.

Slika 8. Kategorija Izdvojene vijesti.



Slika 9. Kategorija *Novosti i događanja*.

Poveznice

U ovoj će se kategoriji testirati učitavanje elemenata. Testirat će se i što se događa klikom na element prema usporedbi URL adrese.

Slika 10. Kategorija Poveznice

Video

Ova kategorija sadrži jedan element, a to je videozapis. Ovdje će se testirati učitava li se taj element te pokreće li se videozapis kada na njega kliknemo.

Slika 11. Kategorija Video

Podnožje

S obzirom na sadržaj ovih kategorija, testirat će se samo učitavanje elemenata.

Slika 12. Kategorija Info / Podnožje

3.5. Varijable

Varijabla se, gledajući iz aspekta programiranja, definira kao prostor u memoriji kojemu je pridružena određena vrijednost.[62] Pri izradi automatskih testova, varijable predstavljaju selektore, odnosno UI elemente nad kojima želimo izvršiti određene akcije - klik, *submit* i sl. Varijable su se imenovale na hrvatskom jeziku, prema onome što predstavljaju kako bi osobi koja se kasnije susretne s kôdom bilo jasno na što se varijabla odnosi. Varijable su se deklarirale na sljedeće načine:

```
var nazivVarijable ='selektor ' ;
var nazivVarijable = "selektor";
var nazivVarijable = $(selektor);
```

Nadalje, do selektora dolazimo putem klase, ID-a te XPatha. Ako se sadržaj varijable deklarira prema klasi, onda se unutar navodnika piše točka te ime klase. Umjesto svakog znaka razmaka u nazivu klase stavlja se točka.

```
var poveznice = ".col-lg-3.col-md-3.col-sm-3.col-xs-12.hidden-xs.leftSep";
```

Ako se sadržaj varijable deklarira prema XPath-u, onda se unutar zagrade pišu jednostruki navodnici, a unutar jednostrukih navodnika XPath. Do njega se dolazi tako što se na mrežnom mjestu desnim klikom ode na ispitaj element (engl. *inspect element*). Zatim se klikne na  gumb

[62] Usp.WebDriver. URL:https://webdriver.io/docs/selectors.html#mobile-selectors (27.08.2019.)

te se mišem odabere element čiji se XPath traži. U pregledniku se označeni dio istakne plavom bojom. Desnim klikom na taj element dobiva se izbornik koji nudi opciju kopiraj, uz kopiranje XPatha. Klikom na kopiraj XPath, on se kopira u međuspremnik i zatim se stavlja unutar jednostrukih navodnika.

```
var naslovna =    '//*[@id="menu"]/li[1]/a'   ;
```

Treći način deklariranja varijabli je prema ID-u. Ako se varijabla deklarira na takav način, onda se unutar navodnika unosi naziv ID-a sa znakom # ispred naziva.

```
var topMenu = "#top-menu";
```

Selektori u WebdriverIO-u pozivaju se na sljedeći način: $(selektor). Stoga je varijablu moguće deklarirati i na sljedeći način:

```
var selektor = $('//*[@id="page"]/div/div/div[1]/h3');
```

3.6. Testovi

Kada se kreira novi test, prvo što se unutar datoteke upisuje odnosi se na naziv testa na sljedeći način:

```
describe("NazivTesta", function() {
});
```

Nakon toga piše se:

```
 beforeEach(function() {
      browser.url("./");
    });
```

kako bismo osigurali učitavanje stranice koja se odnosi na *baseURL* unutar datoteke wdio.conf. Ova funkcija nije nužna, no u sklopu ovog diplomskog rada je zastupljena, jer se želi osigurati učitavanje mrežne stranice fakulteta prije svakog testa.
Unutar *describe* funkcije pišu se testovi koji započinju s funkcijom *it*, dok se u navodnicima navodi ime testa:

```
describe("NazivTesta", function() {
it("ImeTesta", function() {
        $(selektor).click();
        let VarURL= "https://www.facebook.com/ffosijek/";
        let currentURL = browser.getUrl();
        assert.equal(VarURL, currentURL);
    });
});
```

Najčešće korištene funkcije u ovom programskom rješenju jesu *moveTo(), waitForExist(), click(), scrollIntoView(), addValue() te clearValue()*. Funkcija *moveTo()* pomiče miš na navedeni element.[63] Koristi se na idući način:

```
$(selektor).moveTo();
```

Funkcija *waitForExist()* čeka da u DOM-u[64] bude prisutan element za određeni broj milisekundi. Vraća *true* ako selektor odgovara barem jednom elementu koji postoji u DOM-u, u suprotnom vraća pogrešku.[65] Koristi se na idući način:

```
selektor.waitForExist(90000);
```

gdje 90000 označava broj milisekundi koliko se čeka za učitavanje elementa.
Funkcija *click()* klika mišem na element. Međutim, ako postoji element s fiksnim položajem (poput fiksnog zaglavlja ili podnožja) koji prekrivaju odabrani element nakon što se on pomakne unutar okvira za pregled, test će pokazati grešku.[66] Funkcija *click()* koristi se na sljedeći način:

```
selektor.click();
```

Funkcija *scrollIntoView()* pomičite element u okvir za prikaz. Koristi se kada se nam je određeni element potreban, no preglednik ga ne vidi niti ga automatski stavlja u okvir za prikaz kao kod *moveTo()* funkcije.[67] Koristi se na sljedeći način:

---

[63] Usp. WebDriverIO URL: https://webdriver.io/docs/api/element/moveTo.html (10.08.2019.)
[64] Document Object Model – odnosi se na API za XML i HTML dokumente
[65] Usp. WebDriverIO URL: https://webdriver.io/docs/api/element/waitForExist.html (10.08.2019.)
[66] Usp. WebDriverIO URL: https://webdriver.io/docs/api/element/click.html (10.08.2019.)
[67] Usp. WebDriverIO URL: https://webdriver.io/docs/api/element/scrollIntoView.html (10.08.2019.)

```
selektor.scrollIntoView();
```

Funkcija *addValue()* najčešće se koristi kod testiranja formi te služi za unos vrijednosti u određeni element, tj. polje za unos. Također se mogu koristiti UNICODE znakovi kao što su strelica s lijeve ili desne strane. WebdriverIO će se pobrinuti da ih prevede u znakove UNICODE-a. Da bi se to dogodilo, vrijednost mora odgovarati ključu iz tablice koja se može pronaći na WebDriverIO stranicama.[68] Koristi se na sljedeći način:

```
selektor.addValue("mail @mail.hr");
```

gdje se u zagradi unutar dvostrukih navodnika navodi vrijednost koju je potrebno unijeti.
Funkcija *clearValue()* čisti vrijednost elementa <textarea> ili teksta <input>. Prije korištenja ove naredbe treba provjeriti može li se komunicirati s elementom. Ulazni element s kojim se ne može komunicirati ili je samo za čitanje nije moguće očistiti.[69] Koristi se na sljedeći način:

```
selektor.clearValue();
```

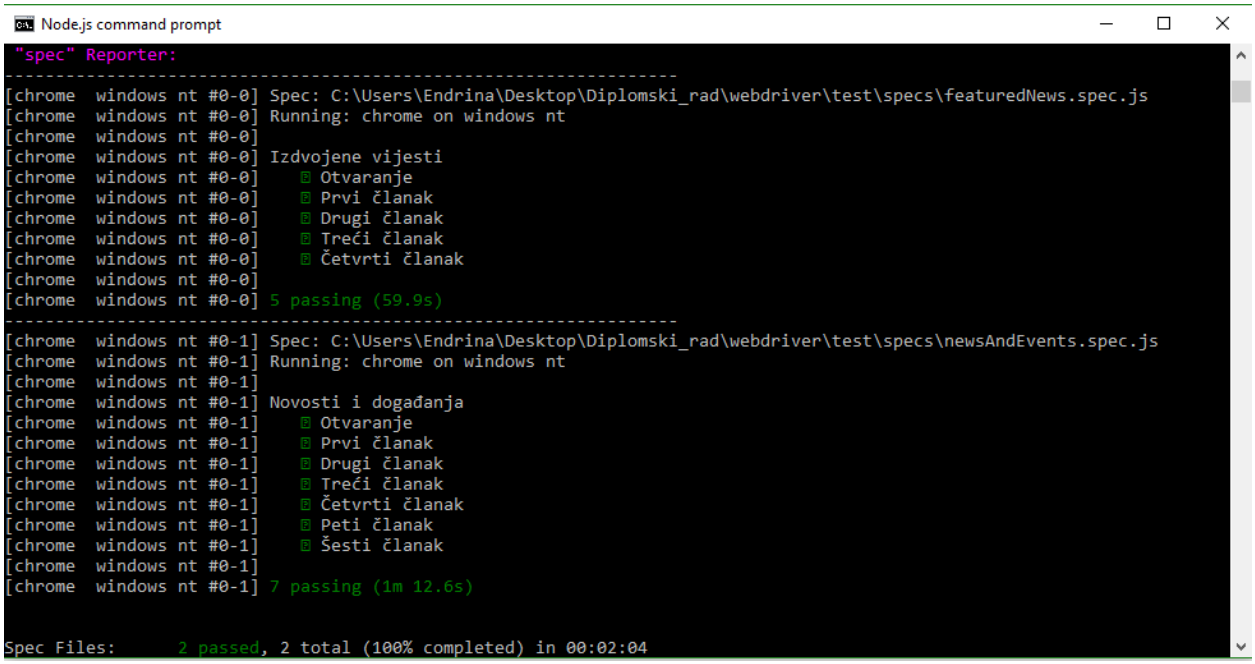
Kako bismo provjerili da je vrijednost elementa očišćena, prvo dohvaćamo vrijednost elementa i onda ju uspoređujemo na sljedeći način:

```
value = selektor.getValue();
assert(value === "");
```

3.7. Prikaz testova

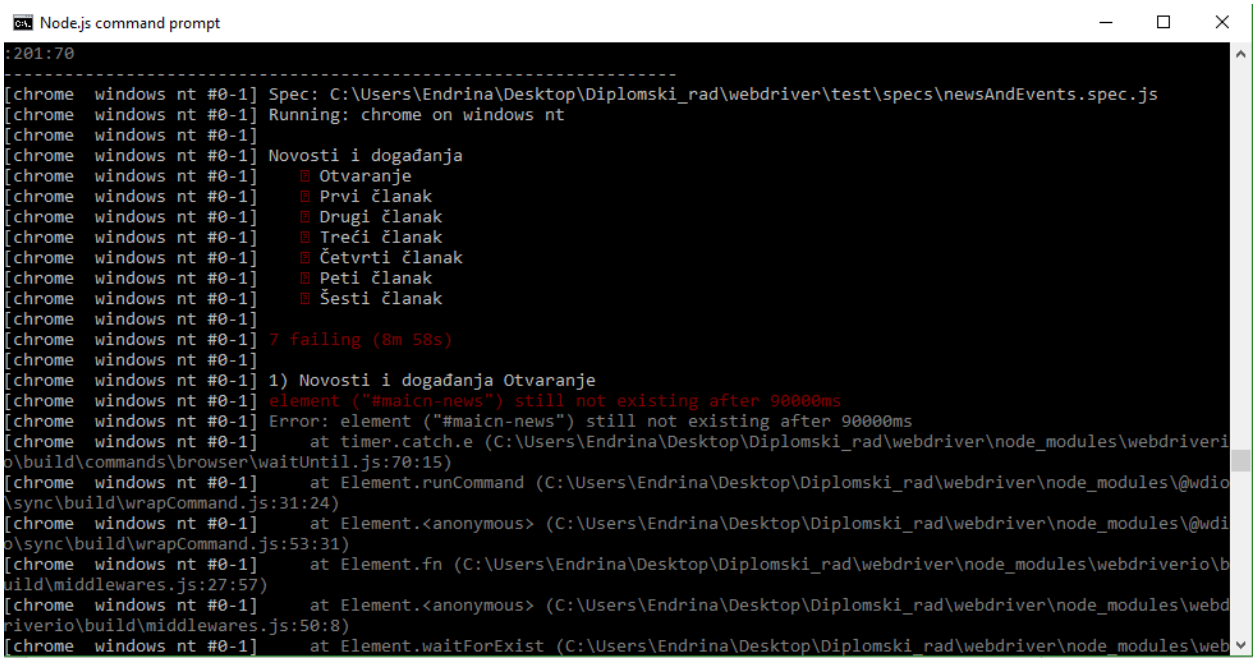
Na slici 13. prikazan je rezultat automatskih testova koji se dobije u terminalu kada programsko rješenje uspješno prolazi automatske testove.

[68] Usp. WebDriverIO URL: https://webdriver.io/docs/api/element/addValue.html (10.08.2019.)
[69] Usp. WebDriverIO URL: https://webdriver.io/docs/api/element/clearValue.html (10.08.2019.)

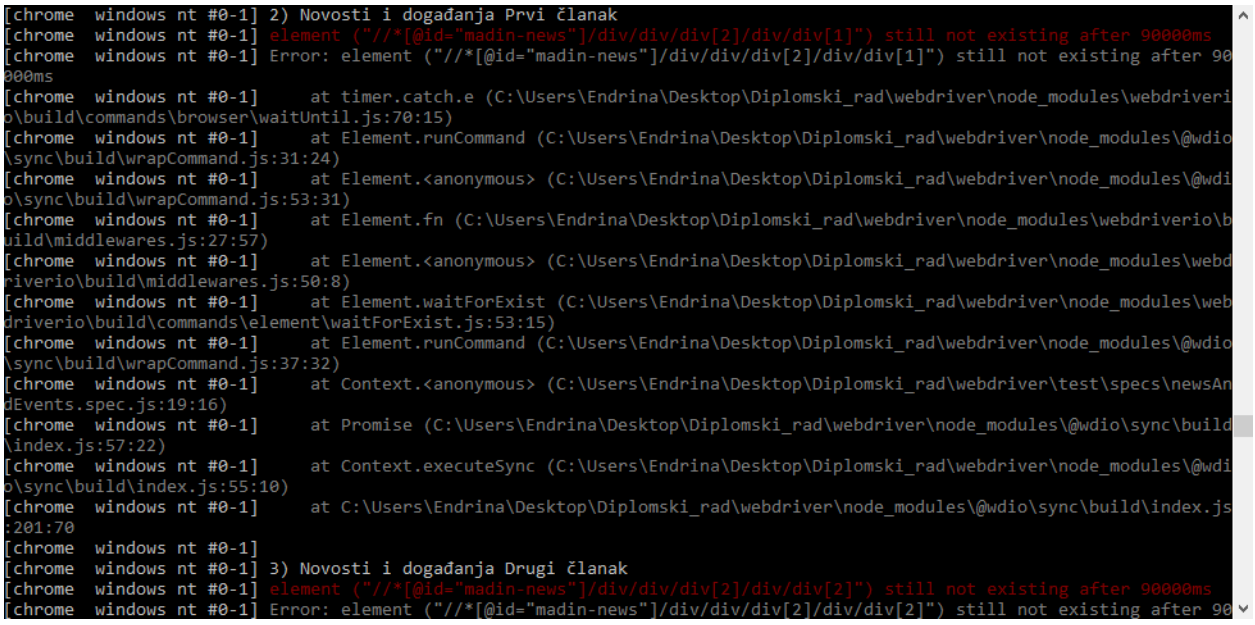


Slika 13. Prikaz terminala u slučaju prolaznosti testova.

No, može se dogoditi i neželjeni rezultat a to jest da programsko rješenje ne prolazi automatske testove.



Slika 14. Prikaz terminala u slučaju pada testova.



Slika 15. Prikaz terminala u slučaju pada testova – različite greške.

Na slici 14. i 15. prikazan je rezultat automatskih testova koji se dobije u terminalu kada programsko rješenje ne prolazi automatske testove. Mnogo je potencijalnih uzroka pada testova, no važno je napomenuti kako je u terminalu vidljivo gdje se točno javlja greška te je samim time ispravljanje greške u testu olakšano.

3.8. Problemi u testiranju

Za vrijeme provedbe testiranja mrežnih stranica Filozofskog fakulteta u Osijeku pojavili su se i određeni problemi. Prvi problem odnosio se na nemogućnost testiranja formi administratora za koje pristupni podaci nisu bili dostupni. Drugi problem ticao se dostupnosti isključivo internetskih (ne i intranetskih) mrežnih stranica fakulteta. Treći problem ogledao se u nepostojanju stranica na koje bi određeni linkovi trebali voditi kao što je *Forum*. Četvrti i posljednji problem na koji se naišlo u procesu testiranja ticao se korištenja dijakritika u nazivu mrežne stranice što nije uobičajena praksa u mrežnom okruženju. To je vidljivo na primjeru padajućeg izbornika odsjeci i katedre; nakon što se klikne na opciju izbornika Mađarski jezik i književnost. URL stranice koja se otvara glasi: https://www.ffos.unios.hr/mađarski.

U skladu sa svim navedenim, mrežne stranice Filozofskog fakulteta u Osijeku ne mogu se opisati kao „test-friendly“, a razlog tomu je slaba pokrivenost DOM objekata ID-jevima. Zbog toga, najčešće se koristio XPath- što je generalno loša praksa u svijetu automatizacije. Samim time što stranica ne pruža ID-jeve, koristile su se klase i XPath-ovi koji su skloni promjenama te izmjene

u kôdu mogu ugroziti trenutne testove. Osim toga, na stranici se javljaju neke netipične greške kao što je poveznica na stranice Pravnog fakulteta u Osijeku klikom na logo Filozofskog fakulteta u Osijeku na stranici Zašto studirati na FFOS-u (https://www.ffos.unios.hr/upisiffos/index.html). Stvari koje se mogu poboljšati u budućnosti ovih automatskih testova jesu razdvojiti selektore u zasebne datoteke, slijediti *Page Object pattern*[70] te izdvojiti skup koraka koji se učestalo koristi u zasebne metode koje će se pozivati u različitim testovima što automatski znači manje kôda, a lakšu održivost.

[70] Usp. WebDriverIO URL: https://webdriver.io/docs/pageobjects.html (10.08.2019.)

4. ZAKLJUČAK

U današnje je vrijeme doticaj s različitim programskim rješenjima neizbježan dio svakodnevnog života većine ljudi. Stoga, važno je da su ta programska rješenja kvalitetna. Popularizacija testiranja programskih rješenja pokazuje koliko je kvaliteta proizvoda bitna ne samo krajnjim korisnicima, nego i proizvođačima. Razlikujemo dvije vrste testiranja, ručno i automatsko. No ona se ne mogu odvojiti, već samo nadopunjavati.

U radu je prikazan proces automatskog testiranja mrežnih stranica Filozofskog fakulteta u Osijeku namijenjen informatičkoj službi fakulteta koja se bavi održavanjem navedenog mrežnog mjesta. Za pisanje kôda automatskih testova koristili su se JavaScript programski jezik, Node.js, Selenium, WebDriver, WebDriverIO, Visual Studio Code te Chrome internet preglednik. Cjelokupni test podijeljen je u devet kategorija, odnosno u devet različitih spec.js. datoteka koje se nalaze u Prilogu rada. Navedeni automatski testovi mrežnih stranica Filozofskog fakulteta u Osijeku služe kao temelj za automatsko testiranje spomenute mrežne stranice. Generalno, testirale su se značajke interaktivnosti i učitavanje elemenata. Testirani su temeljni dijelovi mrežnih stranica. Dakako, u samim testovima prostoji prostor za njihovo unaprjeđenje te proširenje obuhvata automatskog testiranja. Za unaprjeđenje testova potrebno je poznavati HTML sintaksu, barem jedno razvojno okruženje, osnove JavaScript programskog jezika te WebDriverIO kao framework za Node.js. Prije samog proširenja ovih testova, važno je pokrenuti prethodno postojeće testove kako bi bili sigurni da promjena klasa ili XPatha nije uzrokovala pojavu greške.

Iako se testiranje danas smatra sastavnim dijelom razvojnog procesa programskih rješenja, vidljivo je da u praksi nisu sva programska rješenja testirana, ili barem ne na način koji će osigurati najveću moguću kvalitetu u određenom trenutku. Ovi automatski testovi testirali su mrežno mjesto obrazovne ustanove, čija bi kvaliteta trebala biti pri samom vrhu, jer su obrazovne ustanove ujedno i važan segment društva.. Rezultati testova pokazuju da je stranica veoma kvalitetna, no ipak postoje neke greške koje se trebaju ispraviti. Ovim radom željelo se ukazati na važnost testiranja mrežnih stranica proširujući praksu testiranja i na mrežne stranice obrazovnih ustanova čime se umnogome podiže njihova kvaliteta.

Literatura

1. Apica. URL: https://www.apicasystems.com/blog/automated-testing-vs-manual-testing/
2. Graham, Dorothy [et.al.]. Foundations of Software Testing ISTQB Certification. UK: Gaynor Redvers-Mutton, 2008.
3. Guru 99. URL: https://www.guru99.com/levels-of-testing.html
4. Intense testing. URL: https://intensetesting.wordpress.com/2014/03/28/pair-testing-buddy-testing/
5. ISTQB Glossary. URL: https://glossary.istqb.org/en/search/testing
6. Java Point. URL:https://www.javatpoint.com/black-box-testing-vs-white-box-testing-vs-grey-box-testing
7. Kolawa, Adam; Huizinga, Dorota (2007). Automated Defect Prevention: Best Practices in Software Management. Wiley-IEEE Computer Society Press. p. 74. ISBN 978-0-470-04212-0. URL: https://epdf.pub/automated-defect-prevention-best-practices-in-software-management44993.html
8. Lardinois, Frederic. Microsoft Launches Visual Studio Code, A Free Cross-Platform Code Editor For OS X, Linux And Windows. URL: https://techcrunch.com/2015/04/29/microsoft-shocks-the-world-with-visual-studio-code-a-free-code-editor-for-os-x-linux-and-windows/?guccounter=1&guce_referrer_us=aHR0cHM6Ly9lbi53aWtpcGVkaWEub3JnLw&guce_referrer_cs=zHbu753eJL-A3ARIbhN5vA
9. Linz, Tino; Spillner,Andreas; Schafer,Hans. Software testing foundations. Santa Barbara: Rocky Nook Inc., 2014.
10. MDN web docs. URL: https://developer.mozilla.org/en-US/docs/Web/XPath
11. Medium. URL: https://medium.com/@specktackle/selenium-and-webdriverio-a-historical-overview-6f8fbf94b418
12. Professional QA.com. URL:http://www.professionalqa.com/types-of-defects-in-software-testing
13. Raguž, Robetra. Prednosti i nedostaci uporabe node.js. platforme, Osijek, 2017. URL: https://repozitorij.etfos.hr/islandora/object/etfos%3A1267/datastream/PDF/view
14. Selenium HQ. URL:https://www.seleniumhq.org
15. Software Testing Fundamentals. URL:http://softwaretestingfundamentals.com/test-case/

16. Software testing help. URL: https://www.softwaretestinghelp.com/10-tips-you-should-read-before-automating-your-testing-work/
17. Wacker, Mike: Just Say No to More End-to-End Tests, 22.04.2015. URL:https://testing.googleblog.com/2015/04/just-say-no-to-more-end-to-end-tests.html
18. WebDriver.IO. URL:https://webdriver.io/docs/gettingstarted.html