Automatsko testiranje programa

Jukić, Ivana

Master's thesis / Diplomski rad

2018

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Organization and Informatics / Sveučilište u Zagrebu, Fakultet organizacije i informatike**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:211:043593

Rights / Prava: Attribution-NonCommercial-NoDerivs 3.0 Unported / Imenovanje-Nekomercijalno-Bez prerada 3.0

Download date / Datum preuzimanja: **2024-04-23**

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

Repository / Repozitorij:

Faculty of Organization and Informatics - Digital Repository

zir.nsk.hr

DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Ivana Jukić

AUTOMATSKO TESTIRANJE PROGRAMA

DIPLOMSKI RAD

Varaždin, 2018.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Ivana Jukić

Matični broj: 45360/16-R

Studij: Organizacija poslovnih sustava

AUTOMATSKO TESITRANJE PROGRAMA

DIPLOMSKI RAD

Mentor/Mentorica:

Prof. dr. sc. Strahonja Vjeran

Varaždin, rujan 2018.

Ivana Jukić

Izjava o izvornosti

Izjavljujem da je moj završni/diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

U ovome radu opisano je testiranje programskih proizvoda, tj. definicija testiranja, procesi, vrste te metode testiranja. Osim toga opisano je automatsko testiranje, navedene su značajke automatizacije te alati za automatiziranje. U praktičnom dijelu rada prikazan je primjer automatiziranja testnih scenarija.

Ovaj rad uvodi čitatelja u teoriju o testiranju sustava te prikazuje značaj testiranja za razvoj softvera. Navedene su prednosti i nedostaci načina testiranja te postojeći standardi za provođenje testiranja. Nakon upoznavanja s značenjem testiranja i načinima provođenja, u radu je opisana automatizacija testnih procesa. Navedene su značajke automatizacije te problemi koji se pojavljuju.

Rad je temeljen na velikom broju izvora literature te na mom osobnom stajalištu i iskustvu. Za većinu poglavlja, osim teorijskog dijela, navedeno je moje iskustvo iz prakse i moj doživljaj o temi poglavlja.

Ključne riječi: definicija testiranja, proces testiranja, vrste testiranja, metode testiranja, standardi testiranja, automatsko testiranje

Sadržaj

1. Uvod 1
2. Pojam testiranja 2
2.1. Povijest testiranja 2
2.2. Pojmovi u testiranju 2
2.3. Definicija testiranja 4
2.4. Cilj testiranja 4
2.5. Svrha testiranja 6
3. Proces testiranja 8
3.1. Proces testiranja kroz vodopadni model 8
3.2. Proces testiranja prema ISTQB 9
3.2.1. Planiranje i kontrola 10
3.2.2. Analiza i dizajn 10
3.2.3. Implementacija i izvršenje 10
3.2.4. Evaluacija i izvješćivanje 11
3.2.5. Aktivnosti zatvaranja testiranja 11
3.3. Proces testiranja prema ISO standardu 12
4. Vrste testiranja 14
4.1. Generičke vrste testiranja 14
4.1.1. Funkcionalno testiranje 14
4.1.2. Nefunkcionalno testiranje 15
4.1.3. Testiranje softverske strukture 16
4.1.4. Testiranje povezano s promjenama i regresijsko testiranje 16
4.2. Vrste testiranja kroz životni ciklus softvera 17
4.2.1. Jedinično testiranje 17
4.2.2. Integracijsko testiranje 18
4.2.3. Testiranje sustava 19
4.2.4. Testiranje prihvatljivosti 19

5. Metode testiranja 21
5.1. Testiranje crne kutije 21
5.2. Testiranje bijele kutije 22
5.3. Testiranje sive kutije 24
5.4. Agilno testiranje 25
5.5. Ad-hoc testiranje 26
6. Standardi testiranja 27
6.1. ISO/IEC/IEEE 29119 27
6.1.1. ISO/IEC/IEEE 29119-1: Koncepti i definicije 27
6.1.2. ISO/IEC/IEEE 29119-2: Procesi testiranja 28
6.1.3. ISO/IEC/IEEE 29119-3: Testna dokumentacija 28
6.1.4. ISO/IEC/IEEE 29119-4: Tehnike testiranja 28
6.1.5. ISO/IEC/IEEE 29119-5: Testiranje na temelju ključnih riječi 28
6.2. Ostali standardi 29
7. Automatsko testiranje 30
7.1. Pojam automatskog testiranja 30
7.2. Svrha automatskog testiranja 31
7.3. Problemi i ograničenost automatskog testiranja 32
7.4. Alati za automatsko testiranje 34
7.4.1. Selenium – alat za automatsko testiranje 35
7.4.2. IntelliJ IDEA 36
7.4.3. Maven 36
7.4.4. Cucumber 37
7.4.5. Gherkin 37
8. Kreiranje novog projekta 38
8.1. Google.feature 39
8.2. TestSteps.java 41
8.3. SharedSteps.java 41
8.4. GooglePO.java 42

8.5. runTest.java 42
9. Praktični primjer testiranja FOI web mjesta 45
9.1. ELF sustav 45
9.2. E-mail sustav 47
9.3. E-portfolio sustav 48
9.4. Sustav knjižnice 48
9.4.1. Servis kontakti 49
9.5. Servis planer 50
9.6. Rezultati testova 50
9.7. Opažanja 51
10. Zaključak 53
11. Literatura 55
12. Popis slika 59
13. Prilog 1 – programski kod 60
13.1. Elf.feature 60
13.2. Email.feature 61
13.3. EPortfolio.feature 62
13.4. Knjiznica.feature 63
13.5. Kontakti.feature 64
13.6. Planer.feature 65
13.7. EflPrijavaSteps.java 66
13.8. EmailSteps.java 67
13.9. EPortfolioSteps.java 68
13.10. KontaktiSteps.java 70
13.11. LoginSteps.java 71
13.12. NaslovnaSteps.java 72
13.13. PlanerSteps.java 73
13.14. PretragaKnjizniceSteps.java 75
13.15. runTest.java 76

13.16. SharedObjects.java 77
13.17. ElfPO.java.....79
13.18. EmailPO.java.....80
13.19. EPortfolioPO.java 81
13.20. KnjiznicaPO.java.....83
13.21. KontaktiPO.java.....84
13.22. LoginPO.java 85
13.23. NaslovnaPO.java.....87
13.24. PlanerPO.java 89

1. Uvod

Proces razvoja programskih proizvoda nije nimalo jednostavan. Veliki utrošak vremena je na analizi potreba korisnika. Tek nakon što su utvrđene potrebe i kreirana funkcionalna specifikacija može se početi s implementacijom. Nakon uspješne implementacije programski proizvod se isporučuje korisniku.

Testiranje programskih proizvoda je proces koji se obavlja prije isporuke proizvoda. Budući da se tokom implementacije pojavi velik broj grešaka u sustavu, potrebno je provjeriti ispravnost sustava prije isporuke kako bi osigurali zadovoljstvo korisnika.

Kako bi isporučili što kvalitetniji proizvod potrebno je provesti testiranje sustava na nekoliko razina. Vrste testiranja ovise o stanju sustava te mjestu u razvojnom ciklusu. Ovisno o tome da li je implementacija tek započeta ili je pri kraju primjenjivati će se drugačiji načini testiranja.

Osim što testiranje omogućava pronalaženje grešaka u sustavu, testiranjem se provjerava doživljaj sustava iz perspektive korisnika, tj. tester ima zadatak promotriti sustav kao krajnji korisnik.

Budući da je većina sustava vrlo opsežna te sadrži veliki broj funkcionalnosti, potrebno je uložiti puno vremena u proces testiranja. Ukoliko se testiranje ne obavi na ispravan i detaljan način, velika je vjerojatnost da će korisniku biti isporučen proizvod s nepravilnostima. Kako bi smanjili potrebu za ručnim radom testera moguće je automatizirati testne scenarije.

Automatizacija testiranja je proces pisanja programskog koda koji će samostalno pokretati zadane testne scenarije te provjeravati ispravnost sustava. Automatizacijom se oslobađa vrijeme testera te je on u mogućnosti obavljati druge potrebe zadatke.

U ovom diplomskom radu opisan je pojam testiranja te načini na koje je testiranje moguće provoditi. Osim toga, u radu je prikazano automatsko testiranja te primjer automatizacije testnih scenarija. Smatram da je ovo vrlo važna tema te je potrebno educirati širu populaciju o značajkama testiranja.

2. Pojam testiranja

U ovom poglavlju navedena je kratka povijest testiranja, opisani su osnovni pojmovi korišteni u testiranju, navedena je definicija testiranja te cilj i svrha.

2.1. Povijest testiranja

Prije Industrijske revolucije i doba modernog kapitalizma, osiguranje kvalitete proizvoda nije postojalo. Zbog malog tržišta kupci su bili prisiljeni kupiti proizvode čija kvaliteta nije odgovarala traženoj cijeni. U 19. stoljeću, kada je došlo do povećanja tržišta{} cilj prodavača bio je osigurati proizvode bez grešaka kao bi privukli nove i zadržali postojeće kupce. [19]

Sredinom dvadesetog stoljeća dolazi do razvoja prvih računala te time i prvih programskih grešaka. Jedno od prvih računala, Mark II, izrađen je na Harvard sveučilištu. Tokom rada na računalu tehničari su primijetili da je ono prestalo raditi. Dok su pokušavali pronaći uzrok prestanka rada računala pronašli su kukca, moljca, unutar računala. Iz ovoga je nastao danas korišteni naziv za greške u sustavu – eng. *bug*. [34, str. 32]

Ubrzo je prepoznata potreba za testiranjem programa te dolazi do formiranja prvih testnih timova te se pojavljuje pojam osiguranja kvalitete softvera. U 1979. godini nastala je prva knjiga isključivo za testiranje softvera. [25]

S vremenom testiranje stoftvera dobiva sve veći značaj, razvijaju se razni modeli i načini testiranja, kao i automatiziranje procesa testiranja.

U početku testiranje programa obavljao je sam programer, dok danas tester surađuje sa programerom usmjeravajući ga na kvalitetniji rad. Ovim načinom tester je slobodan pronaći najbolji način za osiguranje kvalitete proizvoda. [29, str. 37]

Danas testiranje programa ima veliku važnost pri izgradnji programskog proizvoda te je pretpostavka da će u bližoj budućnosti održati, ili povećati, ulogu u razvoju. Kako bi testiranje zadržalo svoju poziciju potrebno je prilagoditi se promjenama koje se svakodnevno događaju. Pojavom novih tehnologija kao što su umjetna inteligencija ili roboti, posao testera se značajno mijenja te područje testiranja mora biti spremno na ovakve promjene.

2.2. Pojmovi u testiranju

Kako bi razumjeli testiranje programskih proizvoda, tj. softvera, potrebno je znati neke osnovne pojmove.

Jedan od osnovnih pojmova i razlog testiranja je osiguranje kvalitete. Kvalitetu proizvoda teško je jednoznačno definirati iz razloga što je kvaliteta subjektivna. [33, str. 23] Prema ISO/IEC standardu kvaliteta softvera ovisi o sljedećim faktorima: [23]

- funkcionalnosti proizvoda – označava osnovnu svrhu proizvoda
- pouzdanosti proizvoda – sposobnost sustava da održava pružanje usluga u definiranim uvjetima za određeno razdoblje
- upotrebljivosti proizvoda – odnosi se na jednostavnog korištenja određene funkcije sustava
- učinkovitosti proizvoda – odnosi se na resurse sustava koji se koriste pri pružanju funkcionalnosti (npr. memorija, mreža, prostor na disku itd.)
- održivosti proizvoda – označava sposobnost prepoznavanja i uklanjanja kvara unutar softvera, tj. odnosit se na podršku sustava
- prenosivosti proizvoda – označava koliko dobro softver može usvojiti promjene u okruženju ili u zahtjevima, tj. koliko je sustav prilagodljiv

Osim navedenoga, osiguravanje kvalitete proizvoda znači i osigurati da korisnik dobije proizvod koji je tražio.

Tester pridonosi kvaliteti proizvoda pronalaženjem programskih grešaka i usmjeravajući na njihov ispravak.

Prilikom testiranja čest je slučaj da dođe do neočekivanog ponašanja sustava. Mogući su razni slučajevi neočekivanog ponašanja te zbog toga je potrebno koristiti pravilnu terminologiju pri objašnjavanju zatečenog problema.

U literaturi postoji velik broj različitih izraza za probleme u softveru. Na primjer, ukoliko dođe do problema pri izvršavanju određenih uvjeta kaže se da je došlo do neuspjeha (eng. *failure*). Neuspjeh je nastao zbog kvara ili greške u sustavu (eng. *fault*, *defect*). Često se za greške sustava koristi termin „bug“ prema uzroku prve računalne pogreške. Greške u sustavu nastale se zbog pogreške ili propusta programera (eng. *mistake*, *error*). [33] U praksi su najčešći izrazi problem, greška i „bug“.

Možemo reći da postoji greška u sustavu ako:

1. softver ne može obaviti akcije koje su navedene u specifikaciji sustava
2. softver obavlja akcije koje ne bi trebao smjeti prema specifikaciji sustava
3. softver radi akcije koje nisu spomenute u specifikaciji sustava

4. softver ne može obaviti neke akcije koje bi mosi spomenute u specifikaciji ali bi ih trebao moći obaviti
5. softver je teško razumljiv, teško ga je koristiti, usporen je i nije intuitivan

[34]

2.3. Definicija testiranja

Prema International Software Testing Qualifications Bord (ISTQB) „testiranje je proces koji se sastoji od svih aktivnosti životnog ciklusa, statičnih i dinamičnih, koje se tiču planiranja, pripreme i evaluacije softverskih proizvoda i povezanih proizvoda kako bi se utvrdilo da zadovoljavaju određene zahtjeve, da su prikladni za svoju svrhu te da se otkriju nedostaci.“ [24, str. 27]

Kraće rečeno, testiranje je proces utvrđivanja zadovoljstva zadanih zahtjeva i otkrivanja nedostataka softvera. Testiranjem se provjerava da li je softver dovoljno dobar za isporuku, da li postoje nedostaci ili greške u radu sa softverom te da li su zadovoljeni svi korisnički kriteriji.

Također, testiranjem se povećava kvaliteta proizvoda. Pronalaskom grešaka u sustavu prije nego što je proizvod isporučen korisniku osigurava zadovoljstvo korisnika jer on dobije softver bez grešaka.

2.4. Cilj testiranja

Rečeno na jednostavan način, „cilj testiranja je pronaći greške u sustavu“. Moguć je slučaj kada je cilj testiranja samo potvrditi da izrađene funkcionalnosti rade, ali u tom slučaju propuste se greške u sustavu. Ukoliko takav sustav dođe do krajnjeg korisnika posljedice su značajnije, vremenski i financijski. Što se prije pronađe greška to ju je jeftinije i jednostavnije za popraviti. Prema tome nastao je novi cilj testiranja: „pronaći greške u sustavu što je prije moguće“. Gledajući iz perspektive korisnika, ni ovo nije dovoljno. Iako u procesu testiranja budu otkrivene greške u sustavu moguće je da one neće biti riješene do isporuke korisniku. Prema tome, „cilj testiranja je pronaći greške u sustavu što je prije moguće i pobrinuti se da budu riješene“. Ovaj cilj upotpunjuje zadatak testiranja i osigurava da korisniku stigne zadovoljavajući proizvod koji će raditi bez grešaka. [34]

Kada pokušamo stručno prikazati cilj testiranja softvera uvode se dva termina: validacija i verifikacija softvera.

Validacija, kao koncept testiranja programskog proizvoda, se odnosi na provjeru valjanosti proizvoda, tj. validacijom potvrđujemo da proizvod ispunjava svoju namjeravanu uporabu.

Validacijom provjeravamo da li softver zadovoljava očekivanja kupaca te možemo reći da se validacija odnosi na konačni proizvod koji se testira iz perspektive kupca. Validaciju je potrebno provjeravati u što ranijoj fazi razvoja softvera kako bi se izbjegao rizik povećanih troškova do kojih može doći ukoliko se validacija provodi na završetku proizvodnog procesa. [35, str. 40]

Za razliku od validacije softvera, verifikacija, tj. provjera, služi za procjenu softvera na kraju razvojne faze tako da se provjerava da li su zadovoljeni zahtjevi postavljeni na početku razvojne faze. Drugim riječima, verifikacijom provjeravamo da li softver na kraju razvojne faze odgovara specifikaciji sustava. [35, str. 40]

U praksi se uglavnom pod testiranjem softvera misli na verifikaciju programskih proizvoda. Većina proizvođača programskih proizvoda radi u fazama razvoja te je zbog toga verifikacija, tj. provjera nakon završetka faze, jednostavniji i brži način testiranja programskih proizvoda. Validacija softvera izvršava se nakon završetka izgradnje nekoliko funkcionalnosti softvera.

Boris Beizer, američki softver inženjer, ciljeve testiranja promatrao je kao „razine zrelosti procesa testiranja“ pri čemu su razine karakterizirane ciljevima testera. Definirao je pet razina:

0. razina – ne postoji razlika između testiranja i ispravljanja pogrešaka (eng. *debugging*)
1. razina – cilj testiranja je dokazati da softver radi ispravno
2. razina – cilj testiranja je dokazati da softver ne radi
3. razina – cilj testiranja nije dokazivati specifičnosti nego smanjiti rizik korištenja softvera
4. razina – testiranje je mentalna disciplina koja pomaže povećanju kvalitete softvera

[30, str. 35]

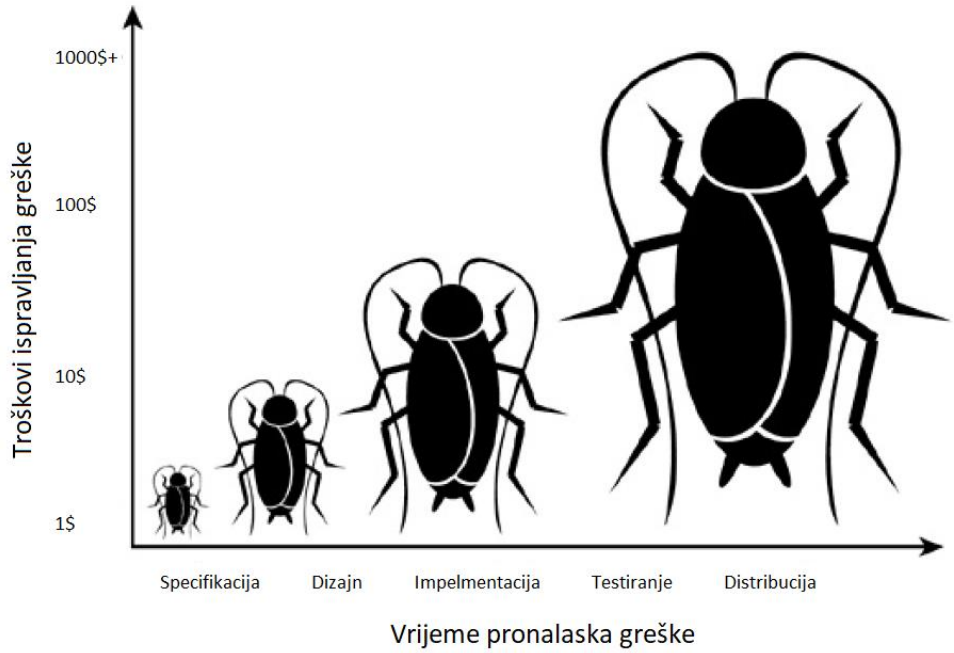
Razmatrajući Beizerov način definiranja ciljeva testiranja smatram da bi svakom testeru cilj trebao biti upravo najviša razina, tj. povećanje kvalitete softvera. Prema mome mišljenju, prva, tj. nulta razina, prema meni se ne može nazvati testiranjem proizvoda. Smatram da je ispravljanje pogreški od strane programera osnovna stvar pri izgradnji sustava te da ne spada u domenu testiranja. Iduća razina potvrđuje ispravnost sustava, ali ne uzimaju se u obzir rubni slučajevi i različiti scenariji koje korisnik može koristiti prilikom rada sa sustavom te zbog toga ni ova razina ne odgovara pravilnom testiranju sustava. Sljedeća razina, dokazivanje da softver ne radi, pokriva rubne slučajeve i osigurava pravilan rad sustava, ali na ovaj način tester ne pridodaje nikakvu dodanu vrijednost sustavu (kao npr. prijedlog za intuitivniji dizajn sustava). Na sljedećoj razini cilj testiranja je smanjiti rizik pri uporabi sustava. Smatram da se ova razina odnosi na iznošenje mišljenja o potencijalnim rizicima te djelovanje kako bi se ti rizici smanjili. Na zadnjoj razini, cilj testiranja je pripomoći pri proizvodnji proizvoda visoke kvalitete. Smatram da je to moguće postići pridodavanjem novih vrijednosti sustavu, suradnjom sa programerima pri samoj izradi sustava te detaljnim razmatranjima korisničkih potreba. Ukoliko tester dobro poznaje korisničke potrebe može bolje provjeriti da li sustav radi

na način zadovoljavajući za korisnika te da li je moguće napraviti sustav na bolji način nego što je to korisnik zamislio.

2.5. Svrha testiranja

Tokom istraživanja i rada u praksi, naišla sam na mišljenja da je testiranje gubitak vremena i novaca za poduzeće te da su sami programeri dovoljni testeri proizvoda. Oslonac ovog stava je činjenica da nikada nije moguće testirati softver u potpunosti te da će često i sustav s greškama završiti kod krajnjeg korisnika. Kao potvrdu potrebe za testiranjem uzet je primjer s nogometnim sucem. Iako sudac u nogometu ne sudjeluje u utakmici veliki dio vremena, kada bi ga u potpunosti uklonili znatno bi se povećao broj prekršaja i neispravnog ponašanja igrača. Ista analogija može se primijeniti i na životni ciklus softvera. Kada bi se testiranje uklonilo iz procesa životnog ciklusa, krajnji proizvod sadržavao bi znatno više grešaka i nepravilnosti što bi rezultiralo nezadovoljstvom korisnika. Upravo zbog ovoga je testiranje iznimno važan korak u razvoju softvera.

Značaj testiranja možemo vidjeti na sljedećoj slici.



Slika 1. Odnos vremena pronalaska greške i troškova ispravka [34, str. 45]

Na Slici 1. vidimo kako vrijeme pronalaska greške utječe na troškove ispravljanja greške. Ukoliko je greška uočena u fazi specifikacije ili dizajna troškovi ispravljanja su mali te je takve greške najjednostavnije ispraviti. Ukoliko je greška uočena nakon faze implementacije ispravak greške je zahtjevniji ali je trošak ispravljanja i dalje prihvatljiv. Ukoliko je greška

uočena tek u fazi distribucije, kada je softver isporučen korisniku, troškovi ispravka mogu biti i nekoliko milijuna.

Prema istraživanjima, 10 % grešaka nastane tijekom faze specifikacije, 40 % tijekom faze dizajna te 50 % grešaka nastane u fazi implementacije. [30] S obzirom na to da najviše grešaka nastane u fazi implementacije nužno je obavljati testiranje sustava tijekom i nakon implementacije. Na ovaj način se smanjuje mogućnost isporučivanja sustava sa greškama te time i visoki troškovi ispravljanja grešaka u fazi distribucije. Osim testiranja samog sustava, potrebna je provjera i dizajna i specifikacije sustava kako bi što ranije uočili greške nastale u ovim fazama te ih uklonili prije same faze implementacije.

Ukratko, možemo reći da je svrha testiranja osiguravanje isporuke ispravnog sustave te smanjivanje troškova ispravljanja grešaka u sustavu.

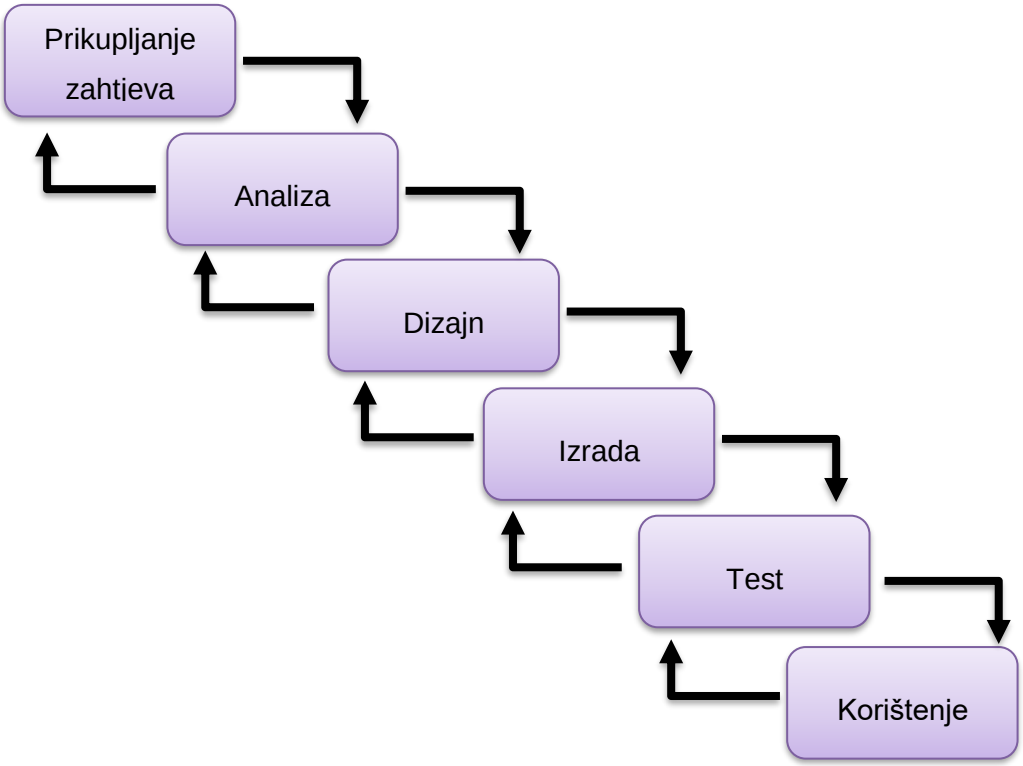
3. Proces testiranja

Pri razvoju softvera postoji nekoliko metodologija – agilni razvoj, vodopadni razvoj, V-model itd. Nastajanjem različitih metodologija za razvoj softvera pojavile su se i metodologije za proces testiranja.

Različiti autori različito percipiraju proces testiranja. Percepcija testiranja ovisi o metodologiji rada te su u nastavku prikazane najčešći pristupi procesu testiranja.

3.1. Proces testiranja kroz vodopadni model

Vodopadni model razvoja softvera bio je prvi nastali model. Ovaj model je vrlo jednostavan i poznat. Sastoji se od nekoliko povezanih faza od kojih iduća može započeti tek kada je prethodna završena. Ukoliko je potrebno, moguće je vratiti se na prijašnju fazu. Osnovni vodopadni model prikazan je na sljedećoj slici. [33, str. 33]

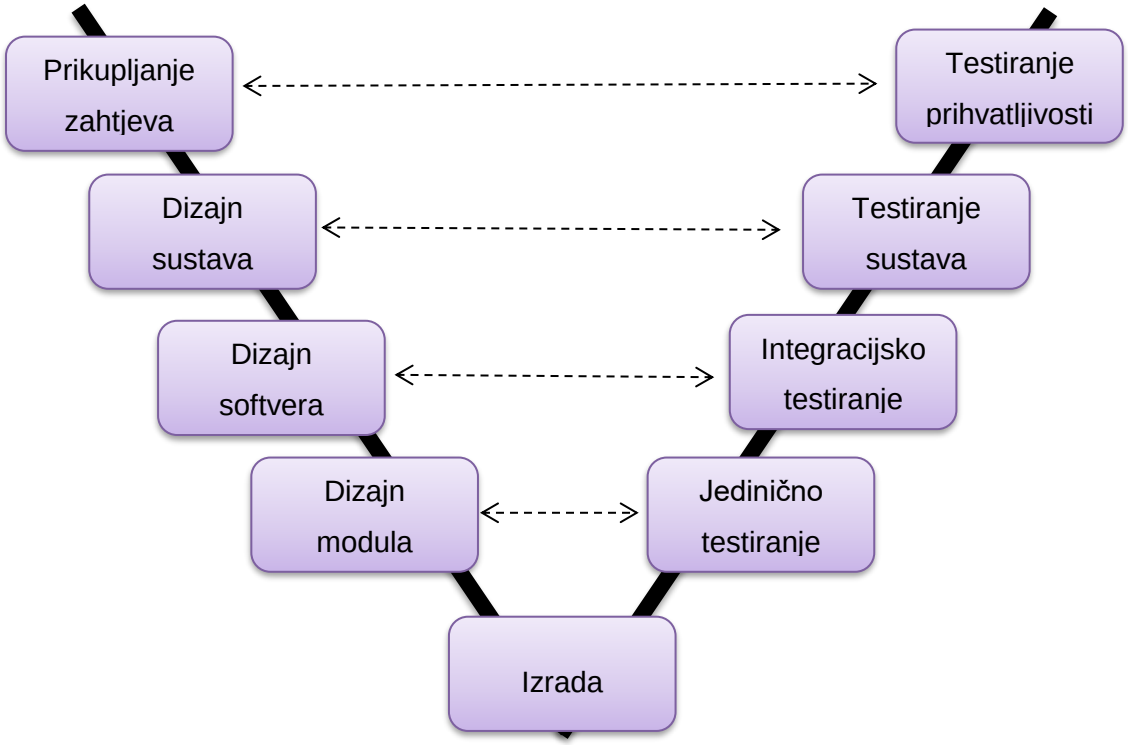


Slika 2. Vodopadni model [33, str. 33]

Vidimo da je testiranje jedna od faza vodopadnog modela razvoja softvera. Osnovna mana ovog modela je što se testiranje izvršava samo jednom u cijelom procesu. Prema ovom modelu testiranje je samo validacija ispravnosti sustava prije isporuke korisniku. Ranije je navedeno

kako ovaj način testiranja može dovesti do velikih problema te se zbog toga ovakav proces rijetko primjenjuje u praksi.

Nakon prepoznavanja potrebe za temeljnijim testiranjem nastao je poboljšani vodopadni model u kojeg je uključeno testiranje nakon svake faze.



Slika 3. Poboljšani vodopadni model [45]

Prema ovom modelu svaka faza razvoja softvera ima odgovarajuću fazu testiranja. Na ovaj način testiranje se obavlja završetkom određene faze te ukoliko su rezultati testiranja zadovoljavajući može se preći na sljedeću razvojnu fazu. Način testiranja ovisi o fazi koja se testira te će ovi načini biti objašnjeni u nastavku.

3.2. Proces testiranja prema ISTQB

Pojavom agilnih metoda razvoja bilo je potrebno osmisliti univerzalni proces testiranja. International Software Testing Qualifications Bord (ISTQB) podijelio je proces testiranja u 5 glavnih faza:

1. Planiranje i kontrola testiranja
2. Analiza i dizajn testiranja
3. Implementacija i izvršenje testiranja
4. Evaluacija i izvješćivanje o testiranju
5. Aktivnosti zatvaranja testiranja

3.2.1. Planiranje i kontrola

Prva faza testiranja softvera, planiranje i kontrola, započinje na početku procesa razvoja softvera. U fazi planiranja mora biti definirana misija i ciljevi testiranja te resursi potrebni za izvršavanje testiranje na temelju čega nastaje dokumentirani plan testiranja. Osim toga, u fazi planiranja potrebno je odrediti strategiju testiranja. Budući da nije moguće u potpunosti testirati sustav, potrebno je odrediti prioritetne funkcionalnosti. Kontrola testiranja je proces uspoređivanja stvarnog napretka testiranja sa planom. Također, kontrola uključuje izvještavanje o statusu i odstupanjima od plana. [33, str. 34]

Iz vlastitog iskustva zaključujem da u praksi nije pridodana velika važnost fazi planiranja testiranja. Većina testnih aktivnosti je improvizirana, ovisno o kapacitetu ljudi ili o roku isporuke te za sada se nisam susrela sa procesom planiranja testiranja.

3.2.2. Analiza i dizajn

Iduća faza, analiza i dizajn testiranja odnosi se na pregledavanje testnog plana. Potrebno je provjeriti zahtjeve, specifikaciju dizajna, analizu rizika i arhitekturu sustava. Na temelju analize kreiraju se testni slučajevi. Analizom određujemo preduvjete i zahtjeve testnih slučajeva, očekivano ponašanje sustava te strukturu testnog objekta. Osim pripremanja specifikacije testova, potrebno je i pripremiti testnu okolinu u kojoj će se provoditi testovi. [33, str. 37]

Smatram da se ovoj fazi daje više značaja u praksi nego fazi planiranja. U praksi testeri raspisuju testne scenarije prema specifikaciji sustava i prema vlastitom iskustvu sa sličnim problemima kako bi sebi i drugim kolegama olakšali testiranje softvera.

3.2.3. Implementacija i izvršenje

Treća faza, faza implementacije i izvršenja testiranja, uključuje izvršavanje kreiranih testova. Testovi se izvršavaju redoslijedom određenim u fazi planiranja, ovisno o prioritetu. Preporuča se započeti testiranje sa provjerom osnovne funkcionalnosti. Ukoliko je pronađena greška u osnovnoj funkcionalnosti nije potrebno testirati ostatak funkcionalnosti nego je prvo potrebno ispraviti pronađenu grešku. Ukoliko pronađena greška ne utječe na rad sustava (npr. pogrešno napisana riječ) testiranje se može nastaviti te se na kraju testiranja prijavi pronađeni problem. Izvršenje testova mora biti ispravno i potpuno dokumentirano, tj. potrebno je dokumentirati koji su testovi izvršeni te koji su rezultati. Osim toga potrebno je dokumentirati svaku pronađenu grešku u sustavu kako bi ona mogla biti ispravljena. Nakon ispravke pronađenih grešaka u sustavu potrebno je ponoviti testove koji su bili neuspješni kako bi se potvrdilo uspješno ispravljanje greške. [33, str. 40]

U praksi testiranje proizvoda označava treću fazu, tj. izvođenje samih testnih slučajeva. Većinu ranog vremena tester se bavi provođenjem testova nad implementiranim funkcionalnostima. Svaku novu funkcionalnost sustava nužno je ručno testirati, dok se postojeće funkcionalnosti mogu testirati i automatski. Nakon što je funkcionalnost bar jednom ručno testirana i potvrđeno je ispravno funkcioniranje sustava pokreću se automatski testovi za provjeru ostalih funkcionalnosti te se kreira automatski test za novu funkcionalnost.

3.2.4. Evaluacija i izvješćivanje

Iduća faza, evaluacija i izvješćivanje o testiranju je proces definiranja kada testiranje treba prestati. Nakon izvršenja testiranja dobiveni rezultati se uspoređuju sa kriterijima zadanima na početku testiranja. Ukoliko su svi kriteriji zadovoljeni može se završiti proces testiranja ili se mogu dodati novi kriteriji koje također treba provjeriti. Do dodavanja novih kriterija u testiranje dolazi ako se odluči da originalni kriteriji nisu dovoljni ili ukoliko su neostvarivi. Ukoliko prema rezultatima testiranja bar jedan kriterij nije zadovoljen potrebno je odraditi dodatne testove. Svaki problem pronađen tokom testiranja potrebno je popraviti te ponoviti proces testiranja. Nakon zadovoljavanja svih kriterija potrebno je sastaviti izvješće o provedenom testiranju. [33, str. 43]

Prema vlastitom iskustvu zaključujem da faza evaluacije nije ništa više nego završetak testiranja u kojemu je potrebno prijaviti pronađene greške ili potvrditi ispravnost softvera, ukoliko greške nisu pronađene. S obzirom na to da se testiranje softvera obavlja cijeli dan, nisam se susrela sa izvješćima o provedenom testiranju.

3.2.5. Aktivnosti zatvaranja testiranja

Posljednja faza, faza zatvaranja testiranja, događa se kada je softver spreman za isporuku. U ovoj fazi potrebno je provesti analize o cjelokupnom procesu testiranja kako bi se znala razlika između planiranih vrijednosti i stvarnih rezultata. Ova analiza pomaže pri planiranju za novi proces testiranja. [33, str. 45]

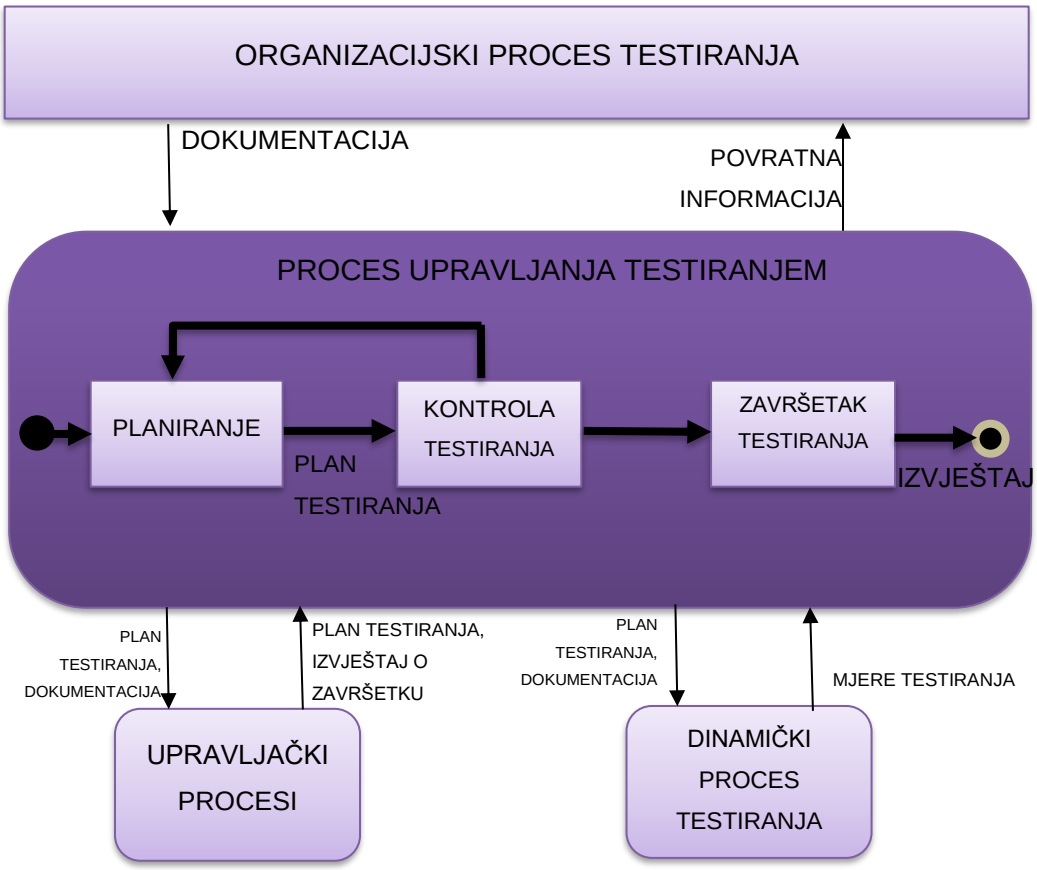
S obzirom na to da se u praksi nisam susrela sa dokumentiranim planom testiranja, lako je za pretpostaviti da se nisam susrela niti sa analizom procesa testiranja. Iako smatram da bi bilo iznimno korisno dokumentirati cjelokupni proces testiranja te na kraju usporediti procjene sa stvarnim rezultatima, koliko sam vidjela iz iskustva, to je vrlo teško za provesti u praksi. Projektni timovi previše su zauzeti postojećim poslom te nisam prepoznala mogućnost odvajanja vremena i za dokumentiranje testiranja.

3.3. Proces testiranja prema ISO standardu

ISO/IEC/IEEE 29119 – 2 standard odnosi se na procese testiranja. Cilj ovo standarda je definiranje generičkog procesa testiranja koji može biti korišten u bilo kojem životnom ciklusu. Proces testiranja temelji se na troslojnom procesu koji pokriva organizacijske specifikacije testova, upravljanje testiranjem te dinamičko testiranje.[8]

Prema standardu koristi se pristup testiranju temeljen na riziku. Ovaj način je korišten je se smatra najboljom praksom za postavljanje i održavanje testova te omogućava određivanje prioriteta testova i fokusiranje na najbitnija svojstva sustava. [8]

Proces testiranja prikazan je kroz sljedeći dijagram.



Slika 4. Proces testiranja prema ISO standardu [8]

Na dijagramu sa slike 4 jasno je vidljiva podjela procesa testiranja. Vidimo da se glavni proces testiranja odvija u procesu upravljanja testiranjem. Na vrhu se nalaze organizacijski procesi koji mogu uključivati određivanje testnih standarda ili pravila organizacije, kao i ostale dokumente koji postoje u organizaciji a vezani su uz proces testiranja. Organizacija prosljeđuje dokumente u proces upravljanja testiranjem. Ovi dokumenti se obrađuju te se nove verzije prosljeđuju organizaciji. Sami proces testiranja počinje sa planiranjem. Tokom faze planiranja

kreira se plan testiranja koji se prosljeđuje na provođenje i kontrolu testiranja te u ostale procese upravljanja testiranjem. Po potrebni plan testiranja je ažuriran. U fazi provođenja i kontrole provodi se testiranje. Dinamičko testiranje provodi se za vrijeme trajanja testiranja te testiranje završava u posljednjoj fazi. U fazi završetka testiranja kreira se izvještaj o završetku te je proces gotov.

4. Vrste testiranja

U literaturi postoji nekoliko shvaćanja vrsta testiranja. Podjela testiranja prema vrstama ovisi o načinu gledanja na proces testiranja. Najčešća podjela tipova testiranja je prema njihovom mjestu u životnom ciklusu razvoja softvera. Osim ove podjele, postoji i generička podjela vrsta testiranja, tj. osnovna podjela vrsta testiranja. U literaturi sam se susrela sa još nekoliko podjela, ali sam odlučila pažnju usmjeriti na navedene dvije podjele te ih opisati u ovom poglavlju.

4.1. Generičke vrste testiranja

Zbog postojanja velikog broja vrsta testiranja, tj. načina na koje je moguće testirati proizvod, osmišljene su generičke vrste testiranja kao viša razina podjele. Svaka od generičkih vrsta podrazumijeva testiranja na jedan ili više načina.

Generičke vrste testiranja su: funkcionalno testiranje, nefunkcionalno testiranje, testiranje softverske strukture i testiranje povezano sa promjenama i regresijsko testiranje. [33, str. 85] Iako neki autori pod generičkim vrstama smatraju samo funkcionalno i nefunkcionalno testiranje, prema ISTQB postoje navedene vrste te su opisane u nastavku.

4.1.1. Funkcionalno testiranje

Jednostavno rečeno, funkcionalno testiranje je proces testiranja sustava prema zahtjevima. Njime provjeravamo da li sustav za određene ulazne parametre vraća željene izlazne parametre. Za funkcionalno testiranje potrebna je specifikacija funkcionalnih zahtjeva sustava. U specifikaciji je opisano na koji način bi sustav trebao funkcionirati te ona nastaje na početku životnog ciklusa softvera. [33, str. 85]

Za provođenje funkcionalnog testiranja koriste se testni slučajevi kojima se provjeravaju funkcionalnosti sustava. Ukoliko na kraju testiranja svi testni slučajevi odgovaraju željenom ponašanju sustava, funkcionalno testiranje je završeno. [33, str. 86]

Funkcionalno testiranje može biti pozitivno testiranje ili negativno. Pozitivno funkcionalno testiranje koristi se kada želimo potvrditi da softver funkcionira kao u specifikaciji. Negativno funkcionalno testiranje koristi se kada želimo pronaći grešku u funkcioniranju sustava. [20]

Funkcionalno testiranje podrazumijeva više načina testiranja sustava. Pod funkcionalno testiranje spada [32]:

- jedinično testiranje (eng. *unit testing*) – testiranje jedinica softvera kako bi se provjerilo da li zasebno ispravno funkcioniraju

- integracijsko testiranje (eng. *integration testing*) – testiranje nekoliko jedinica softvera kao grupe
- sustavno testiranje (eng. *system testing*) – testiranje cijelog sustava kako bi se potvrdilo zadovoljstvo zahtjeva
- eng. smoke *testing* – testiranje velikih dijelova sustava kako bi se provjerilo da li sustav ispravno radi
- eng. *sanity testing* – površno testiranje većih dijelova sustava nakon promjene koda kako bi se utvrdilo da sustav ispravno radi
- regresijsko testiranje – detaljno provjeravanje da promjene u kodu nisu kreirale nove probleme u sustavu
- testiranje prihvatljivosti (eng. *user acceptance testing*) – testiranje da li softver zadovoljava sve potrebe korisnika, često ih provode ključni korisnici sustava
- itd.

4.1.2. Nefunkcionalno testiranje

Za razliku od funkcionalnog testiranja, nefunkcionalno testiranje provjerava karakteristike sustava kao što je brzina rada sustava. [20] Ono ne opisuje funkcije sustava nego atribute funkcionalnog ponašanja sustava. [33, str. 87]

Prema ISO 9126, karakteristike koje se provjeravaju nefunkcionalnim testiranjem su pouzdanost, upotrebljivosti, učinkovitost, kompatibilnost i sigurnost. Provjera ovih karakteristika uvelike povećava zadovoljstvo korisnika jer se osigurava brz i jednostavan sustav za korištenje. Osim toga, što je sustav prilagodljiviji lakše je promijeniti određene funkcionalnosti ako korisnik to zatraži. [33, str. 87]

Pod nefunkcionalno testiranje spadaju: [41]

- testiranje performansi (eng. *preformance testing*) – koristi se za provjeravanja vremena reakcije sustava, tj. provjerava se vrijeme potrebno za dobivanje odgovora od sustava
- testiranje opterećenja (eng. *load testing*) – koristi se za provjeru da li sustav može podnijeti planirano opterećenje, npr. veliki broj korisnika u isto vrijeme
- stres testiranje (eng. *stress testing*) – služi za provjeravanje reakcije sustava pri opterećenju većem od očekivanog korištenja
- testiranje sigurnosti (eng. *security testing*) – služi provjeravanju očuvanja podataka u sustavu ukoliko dođe do zlonamjernih napada; testira se povjerljivost, integritet, dostupnost, autentično i autorizacija
- testiranje kompatibilnosti (eng. *compatibility testing*) – provjerava se radu sustava na različitom hardveru i softveru

- testiranje upotrebljivosti (eng. *usability testing*) . testira se jednostavnost korištenja sučelja aplikacije

itd.

4.1.3. Testiranje softverske strukture

Testiranje softverske strukture se odnosi na provjeravanje strukture sustava ili njegove komponente. Na ovaj način provjeravamo što se, i kako, događa unutar same aplikacije.

Prilikom testiranja softverske strukture, testerima je potrebno znanje o internim implementacijama sustava. Cilj tester je provjeriti kako sustav obavlja svoje zadatke. Na primjer, tijekom testiranja softverske strukture tester će provjeriti kako radi pojedina petlja u programskog kodu te što se njome izvršava. [42]

Testiranje softverske strukture osigurava da će programeri pisati ispravan kod te dobro promisliti o načinu implementacije. Osim toga, služi za otkrivanje grešaka unutar programskog koda i za uočavanje nepotrebnog koda.

Glavni nedostatak testiranja softverske strukture je potreba da testeri imaju dobro znanje o programskom kodu i načinu implementacije. [48]

U praksi nisam uočila slučajeve testiranja softverske strukture od strane testera. Razlog ovome je što ovakvo testiranje zahtijeva puno vremena i dobro poznavanje programskog jezika u kojemu je sustav implementiran. Elemente strukturalnog testiranja mogu prepoznati pri testiranju od strane programera. Svaki novi programski kod provjerava netko od članova tima kako bi u sustavu završio kod pisan prema najboljoj praksi i bez grešaka.

4.1.4. Testiranje povezano s promjenama i regresijsko testiranje

Prilikom implementacije novih funkcionalnosti sustava, nužne su promjene u postojećem programskom kodu. Prilikom uvođenja promjena moguć je nastanak grešaka u sustavu. Kako bi se provjerilo da funkcionalnosti koje su radile prije promjena funkcioniraju i nakon dodavanja novog programskog koda potrebno je ponoviti testove koji su već prije bili izvršeni. Ovakav način testiranja naziva se regresijsko testiranje. [33, str. 90]

Testiranje povezano sa promjenama provodi se nakon ispravljanja greške u sustavu. Ovaj način testiranja služi za potvrdu ispravnog funkcioniranja sustava te se njime testira samo pronađeni problem. Za razliku od ovoga, regresijskim testiranjem se pokrivaju sve funkcionalnosti sustava, uključujući i one koje su prije izmjena ispravno funkcionirale. Regresijsko testiranje se provodi ukoliko je potvrđeno da je navedeni problem ispravljen te je potrebno utvrditi da nisu nastale nove greške u sustavu.

U praksi je ovo jedan od najčešćih načina testiranja. Prilikom svake promjene provodi se testiranje te tester veliki dio radnog vremena utroši na osiguravanje ispravnog nakon uvođenja promjena. Ovisno o složenosti promjene ovaj proces za jednu funkcionalnosti može trajati i više od jednog radnog dana.

4.2. Vrste testiranja kroz životni ciklus softvera

„Životni ciklus softvera je detaljno definiran i strukturiran slijed faza u softverskom inženjerstvu za razvoj softverskog proizvoda.“ [10]

Postoji nekoliko modela životnog ciklusa. Već je govoreno o vodopadnom modelu, najpoznatijem načinu provođenja životnog ciklusa. Prema vodopadnom modelu postoje sljedeće vrste testiranja: jedinično testiranje, integracijsko testiranje, testiranje sustava i testiranje prihvatljivosti.

4.2.1. Jedinično testiranje

Sagar Naik navodi kako se jedinično testiranje (eng. *unit testing*) odnosi na testiranje programskih jedinica u izolaciji. [35, str. 83]

Iako se jediničnim testiranjem provjerava ispravnost jedinice sustava, ne postoji jedinstvena definicija jedinice. Ovisno o načinu implementiranja programskog koda jedinica može predstavljati funkciju, postupak, metodu, klasu, itd. Jednostavno rečeno, programska jedinica je komad koda koji se poziva izvan jedinice i koji može pozvati druge programske jedinice. [35, str. 83]

Jediničnim testiranjem provjeravamo ispravnost pojedini jedinice bez doticaja sa drugim jedinicama. Ovim načinom testiranja možemo biti sigurni da sve se sve pronađene greške odnose samo na testiranu jedinicu te ih je zbog toga jednostavnije ispraviti. Osim toga, ukoliko provjerimo da se svaki dio programskog koda ponaša na željeni način manja je vjerojatnost da će se pojaviti greške u završnom proizvodu. [35, str. 83]

S obzirom na to da se testiranje provodi nad jedinicama programskog koda jasno je da jedinično testiranje predstavlja prvu razinu testiranja sustava. Jedinično testiranje provodi programer koji je napisao programski kod jedinice koja se testira. Ovaj način je prikladan jer je on već upoznat s funkcionalnošću jedinice te s programskim kodom. [35, str. 83]

Prednost jedinično testiranja je pronalazak grešaka u ranoj fazi implementacije zbog čega je jednostavnije i brže ispraviti pronađene nepravilnosti. Osim toga, jedinično testiranje pomaže pri održavanju programskog koda jer se osigurava da nastale promjene ne utječu na ostatak programskog koda. [5]

Prema mom mišljenju, jedinično testiranje je iznimno važno za održavanje kvalitete i ispravnosti programskog koda. Iako se osobno nisam susrela sa pisanje jediničnih testova, poznato mi je da kvalitetni jedinični testovi mogu uvelike olakšati posao programera, kao i testera. Vjerujem da iscrpno testiranje jedinice nakon implementacije skraćuje potrebno vrijeme testiranja od strane testera te uklanja mogućnost pojave grešaka kod osnovnih funkcionalnosti jedinice. Više puta sam doživjela da osnovna funkcionalnost ne djeluje ispravno te vjerujem da se to može izbjeći kvalitetnim jediničnim testiranjem.

4.2.2. Integracijsko testiranje

Integracijsko testiranje slijedi nakon jediničnog testiranja. Povezivanjem programskih jedinica formira se veća jedinica ili podsustav softvera te ovaj proces nazivamo integracijom. [33, str. 65]

Iako je svaka programska jedinica testiranja, nakon povezivanja nekoliko jedinica potrebno je provjeriti da li surađuju ispravno. Integracijskim testiranjem provjeravamo zajednički rad nekoliko programskih jedinica i njihovu međusobnu interakciju. [33, str. 65]
Jedan od najvećih problema pri integracijskom testiranju je određivanje najboljeg načina povezivanja programskih jedinica. Cilj je povezati jedinice u smislene cjeline kako bi se sustav lakše testirao. [35, str. 196]

Za grupiranje programskih jedinica može se koristiti neki od sljedećih pristupa: [35, str. 196]

- inkrementalna integracija - integracija ostvaruje se postupnim dodavanjem programskih jedinica
- integracija odozgo prema dole – postoji hijerarhijska struktura sustava, integracija se odvija od najvišeg modula prema nižim modulima
- integracija odozdo prema gore - postoji hijerarhijska struktura sustava, integracija se odvija od najnižih modula prema višim modulima
- sendvič integracija – kombinacija pristupa odozgo prema dole i odozdo prema gore, integracija prema dole se koristi za više module, a integracija prema dole za niže module
- veliki prasak – provodi se testiranje svih pojedinačnih modula te se nakon testiranja u jednom koraku integriraju u cjelinu

4.2.3. Testiranje sustava

Nakon završetka integracijskog testiranja može započeti testiranje sustava (eng. *system testing*). Testiranjem sustava provjerava se da li integrirani sustav zadovoljava specificirane zahtjeve.

Testiranje sustava provode testeri, testirajući sustav iz perspektive korisnika. Zadatak testera je provjeriti da li sustav odgovara svim zahtjevima te da li se ponaša na željeni način. Osim toga, potrebno je testirati cijeli sustav jer neke funkcionalnosti sustava mogu ispravno raditi jedino ako su u potpunosti integrirane sa ostatkom sustava. [33, str. 73]

Tokom testiranja sustava možemo prepoznati neke načine testiranja: [35, str. 225]

- osnovni testovi
- funkcionalni testovi
- testovi interoperabilnosti
- testovi performansi
- stres testovi
- regresijski testovi
- itd.

Glavni problem kod testiranja sustava je nedostatak potrebne dokumentacije. Ukoliko zahtjevi korisnika nisu detaljno dokumentirani i ne postoji specifikacija sustava, tester nema jasnu sliku kako sustav treba funkcionirati te je veća vjerojatnost da će korisnici biti nezadovoljni sustavom. [33, str. 75]

U praksi većinom primjenjujem testiranje sustava. Susrela sam se s navedenim problemom nedostatka dokumentacije, kao i s radom na detaljno dokumentiranom projektu. Na temelju ovih saznanja mogu potvrditi značaj funkcionalne specifikacije kod testiranja softverskog proizvoda. Sustav nije moguće ispravno testirati ukoliko svi zahtjevi nisu dovoljno jasno opisani.

4.2.4. Testiranje prihvatljivosti

Testiranje prihvatljivosti provodi se prije isporuke sustava te se odnosi na korisničku provjeru sustava gdje je zadatak testera proučiti sustav iz korisničke perspektive. U nekim slučajevima su i korisnici uključeni u testiranje prihvatljivosti proizvoda. [33, str. 76]

Prihvatljivost sustava testira se prema korisničkim kriterijima za prihvaćanje sustava. Ovi kriteriji moraju biti definirani na početku implementacije sustava te se koriste pri testiranju prihvatljivosti. Sustav mora odgovarati svim korisničkim kriterijima. [35, str. 483]

Ovisno o načinu testiranja postoje dvije kategorije testiranja prihvatljivosti:

- korisničko testiranje prihvatljivosti – provodi ga korisnik kako bi potvrdio da sustav zadovoljava sve potrebne uvjete
- poslovno testiranje prihvatljivosti – provodi se unutar organizacije isporučitelja sustava kako bi se uvjerili da će sustav proći korisničko testiranje [35, str. 482]

U praksi sam doživjela integriranje poslovnog testiranja prihvatljivosti sa testiranjem sustava. Drugim riječima, testiranje prihvatljivosti se provodi tokom testiranja sustava kako bi se skratilo vrijeme potrebno za provođenje testiranja. Testiranje prihvatljivosti s korisničke strane se obavlja prije svake nove isporuke te se na ovaj način osigurava upoznatost ključnih korisnika sa novim funkcionalnostima sustava i zadovoljstvo prije krajnje isporuke.

5. Metode testiranja

„Metoda je sistematičan postupak, tehnika ili način ispitivanja korišten ili u skladu s određenom disciplinom.“ [12]

Prilikom testiranja programskih proizvoda koristi se nekoliko vrsta testiranja, ovisno o fazi životnog ciklusa u kojoj se softver nalazi i definiranim zahtjevima. Kako bi se sistematizirao postupak testiranja osmišljene su metode testiranja softvera.

U literaturi postoji velik broj shvaćanja metoda testiranja. Neki autori metodama smatraju prethodno navedene vrste testiranja, neki ih opisuju kroz životni ciklus proizvoda dok neki kao metode navode sve načine testiranja.

Unatoč mnogim različitim mišljenjima, najčešća podjela metoda je na testiranje crne kutije, testiranje bijele kutije, testiranje sive kutije, agilno testiranje i ad-hoc testiranje. U nastavku su opisane ove metode testiranja.

5.1. Testiranje crne kutije

Testiranje crne kutije (eng. *black box testing*) je metoda testiranja softvera u kojoj unutarnja struktura sustava nije poznata testeru.[4] U procesu testiranja tester se fokusira na ulazne i izlazne parametre bez ikakvih saznanja o internim procesima sustava.[1] Zbog ovakvog načina korištenja testiranje crne kutije se također naziva i funkcionalnom metodom testiranja. [27]

Slika 5. Model testiranja crne kutije [4]

Na slici 5 prikazan je model testiranja crne kutije. Crna kutija sa slike predstavlja sustav koji se testira. Vidimo kako su testeru poznati samo ulazni i izlazni parametri te da nema nikakvih informacija o unutarnjoj strukturi sustava.

Svrha ove metode testiranja je provjeriti ispravnost funkcionalnosti.[1] Preciznije rečeno, metodom crne kutije možemo uočiti neispravne ili nedostajuće funkcije sustava, pogreške sučelja, pogreške u strukturama podataka ili prilikom vanjskog pristupa bazi podataka, pogreške u ponašanju sustava, itd. [4]

Metoda testiranja crne kutije primjenjiva je kod integracijskog testiranja, testiranja sustava i testiranja prihvatljivosti.[1] Ovaj način nije primjenjiv kod jediničnog testiranja jer jedinično testiranje zahtjeva poznavanje internog rada sustava.

Prednosti primjene metode testiranja crne kutije:

- nije potrebno poznavanje programskog jezika i načina implementacije [1]
- testni scenariji mogu se napisati prije početna implementacije prema funkcionalnoj specifikaciji [1]
- metoda je dobro prilagođena i efikasna za testiranje velikih segmenata programskog koda [47]
- razdvaja korisničku perspektivu od perspektive razvojnog programera [47]

Nedostaci primjene metode testiranja crne kutije:

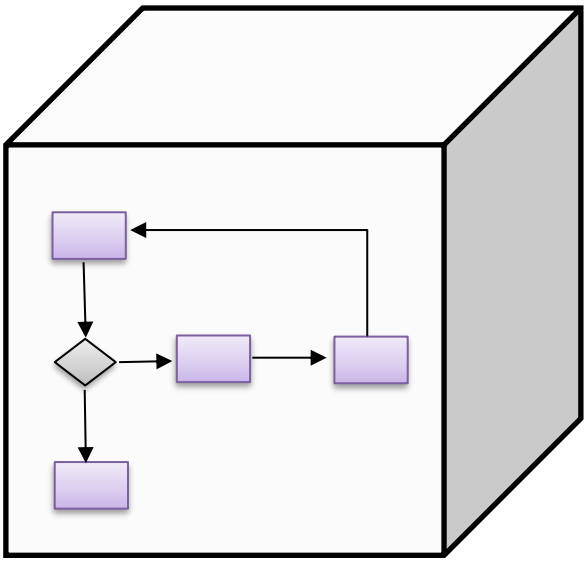
- ograničena je pokrivenost testnim scenarijima [47]
- teško je dizajnirati testne slučajeve ukoliko specifikacija sustava nije dovoljno jasna [4]
- provedeno je neučinkovito testiranje budući da tester ima ograničeno znanje o sustavu [47]
- velika je vjerojatnost da će se ponavljati testovi koje je izvršio programer [1]

Iz vlastitog iskustva mogu prepoznati da je primjena metode testiranja crne kutije jako raširena u praksi. Većina testera sa kojima sam imala dodira testira sustave upravo na ovaj način. Smatram da ovakva praksa nije idealna i efikasna, ali zahtjeva manje ulaganja u znanja i vještine testera te vjerujem da je to razlog širokoj primjeni. Vjerujem da bi sustavi na kojima se radi trebali biti testirani i na neke druge načine kako bi krajnji proizvod bio kvalitetniji.

5.2. Testiranje bijele kutije

Testiranje bijele kutije (eng. *white box testing*) je metoda testiranja softvera u kojoj je unutarnja struktura sustava poznata testeru. [15]

Kod metode testiranja bijele kutije provodi se detaljna istraga interne logike sustava i strukture programskog koda. [47] Ukoliko se uoči greška prilikom testiranja, tester analizira programski kod kako bi otkrio uzrok greške. [27]



Slika 6. Model metode bijele kutije [1]

Na slici 6 vidimo ilustraciju sustava te bijelu kutiju koju on predstavlja. Na slici su ilustrirani procesi i interno funkcioniranje sustava koje mora biti poznato testeru kako bi se provela ova metoda testiranja.

Svrha testiranja bijele kutije je povećanje sigurnosti, povećanje protoka parametara kroz sustav te poboljšanje dizajna i upotrebljivosti sustava. [1]

Metoda testiranja bijele kutije može se primijeniti prilikom jediničnog testiranja, integracijskog testiranja te testiranja sustava. [15] Budući da za ovaj način primjene testiranja je potrebno znanje o programskom kodu te dobro poznavanje internih procesa, metodu testiranja bijele kutije nije moguće primijeniti na testiranje prihvatljivosti. Razlog tome je velik broj funkcionalnosti na razini testiranja prihvatljivosti te otežano shvaćanje sustava iz korisničke perspektive.

Prednosti metode testiranja bijele kutije:

- testeru koji ima znanje o programskom kodu lako je otkriti koja vrsta podataka može pomoći učinkovitosti testiranja sustava [47]
- pomaže pri optimizaciji programskog koda [47]
- testiranje je temeljitije, tj. pokriven je velik broj različitih slučajeva korištenja [1]
- testiranje može započeti prije izrade korisničkog sučelja [1]

Nedostaci metode testiranja bijele kutije:

- povećani su troškovi radi potrebe za stručnim testerom [47]
- teško je za održavanje ukoliko dolazi do čestih promjena [15]

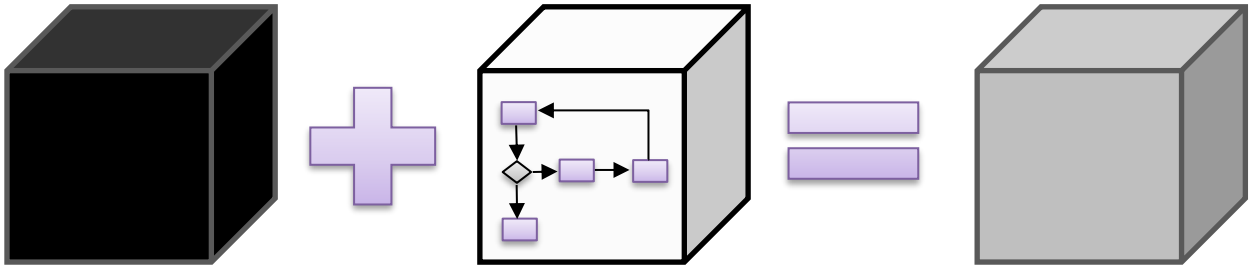
- alati za provjeru svake vrste implementacije na različitim platformama često nisu lako dostupni [15]
- zahtjeva detaljno znanje o programskom jeziku i implementaciji sustava [1]

Iako se osobno nisam susrela sa metodom testiranja bijele kutije znam da se djelomično primjenjuje od strane programera. Elementi koje mogu prepoznati su samostalno testiranje sustava pri i nakon implementacije te ispravljanje pronađenih grešaka. Na žalost, nisam u praksi upoznata sa potpunom primjenom metode testiranja bijele kutije ali vjerujem da primjena ove metode u ranijim fazama životnog ciklusa može uvelike smanjiti broj grešaka u kasnijim fazama.

5.3. Testiranje sive kutije

Testiranje sive kutije (eng. *gray box testing*) je tehnika testiranja aplikacije sa ograničenim znanjem o internim procesom aplikacije. [1]

Metoda testiranja sive kutije je kombinacija metoda testiranja crne kutije i testiranja bijele kutije, tj. tester je djelomično upoznat s internom strukturom. Ovo podrazumijeva pristup internim strukturama podataka i algoritmima za svrhu dizajniranja testnih scenarija dok se samo testiranje vrši iz perspektive korisnika.[6]



Slika 7. Model metode sive kutije [1]

Na slici 7. prikazano je pojašnjenje metode testiranja sive kutije. Vidi se kako je ova metoda nastala kombinacijom metode testiranja crne kutije i metode testiranja bijele kutije te se da zaključiti da će imati elemente obje metode.

Metoda testiranja sive kutije najviše se primjenjuje na integracijskoj razini testiranja iako je primjenjiva i na ostalim razinama. [15]

Prednosti metode testiranja sive kutije:

- nudi kombinaciju prednosti testiranja crne kutije i testiranja bijele kutije [47]
- testeri se ne oslanjaju na programski kod nego na definiciju sučelja i funkcionalnu specifikaciju [47]

- tester je u mogućnosti dizajnirati odlične testne scenarije jer je upoznat s podacima u sustavu [47]
- testiranje se provodi iz perspektive korisnika [47]

Nedostaci metode testiranja sive kutije:

- ograničena je mogućnost proučavanja programskog koda [47]
- postoji mogućnost ponavljanja testnih scenarija [47]

Što se tiče primjene metode testiranja sive kutije u praksi, mogu reći da nisam primijetila značajno korištenje ovakvog načina testiranja. Većina testiranja provodi se kao testiranje crne kutije. U nekoliko slučajeva sam se susrela sa načinom testiranja koji bi okarakterizirala kao testiranje sive kutije. Ovakav slučaj sam iskusila kada mi je bio omogućen pristup bazi podataka te sam testiranje provodila na temelju podataka iz baze. Osim toga, bazu podataka koristila sam za provjeru uspješnosti testnog scenarija. Vjerujem da ovakav način testiranja uvelike olakšava posao testerima, ali i programerima jer je jednostavnije prepoznati uzrok greške.

5.4. Agilno testiranje

Agilno testiranje je metoda testiranja softvera koja prati principe agilnog razvoja softvera.[3]

Prema agilnim principima sustav mora biti suradnički, fleksibilan i prilagodljiv. U ovakvoj okolini potrebno je odrediti prioritetne funkcionalnosti za testiranje, automatizirati testove što je više moguće, povećati korištenje istraživačkih testova te se prilagoditi promjenama. [31]

Prilikom rada u agilnoj okolini testeri moraju ostati u bliskom kontaktu s programerima kako bi surađivali na testiranju kroz cijeli životni ciklus softvera. [31]

Glavni prioritet treba biti zadovoljstvo korisnika. Korisniku je potrebno što češće isporučivati sustav sa traženim promjenama te je potrebno osigurati visoku kvalitetu isporučenog sustava. Zbog ovoga je potrebna bliska suradnja članova razvojnog tima. [3]

U praksi sam radila agilnim načinom rada te smatram da ovakav način osigurava isporuku kvalitetnijeg sustava. Razlog tome je što je sustav napravljen da bude prilagodljiv promjenama koje znaju biti česte. Redovitom isporukom sustava korisniku osigurava se zadovoljstvo korisnika jer je korisniku dana mogućnost provjere manjih promjena u nekoliko iteracija. Na ovaj način testiranje se provodi svakodnevno te smatram da je zbog toga krajnji proizvod kvalitetniji.

5.5. Ad-hoc testiranje

Ad-hoc testiranje je vrsta testiranja koja se provodi bez plana i dokumentacije. Testovi se provode neformalno i nasumično bez formalnih očekivanih rezultata.[1]

Ovaj način testiranja je najmanje formalna metoda zbog toga što tester improvizira korake i provodi ih proizvoljno. Greške sustava pronađene ovom metodom su teže za reproducirati jer ne postoje dokumentirani koraci, ali je moguće pronaći značajne greške koje nije moguće pronaći formalnim metodama. [1]

Ova metoda testiranja se primjenjuje kod testiranja prihvatljivosti sustava.[2]

U praksi sam većinom primjenjivala upravo ovakav način testiranja softvera. Budući da ne postoji plan testiranja, većina testova temeljila se na samostalnom proučavanjem sustava. Ovakav način možemo nazvati i istraživačkim testiranjem jer tester istražuje sve elemente sustava te njihovo ponašanje u neuobičajenim situacijama. Smatram da je ovaj način vrlo koristan jer se pokriju slučajevi koje je teško formalno raspisati, a moguće je da korisnik izvede tu kombinaciju koraka u sustavu. Također, budući da će krajnji korisnici imati slobodu nad aplikacijom vjerujem da je od velike važnosti utrošiti vrijeme na istraživanje različitih kombinacija u sustavu.

6. Standardi testiranja

Danas postoji mnoštvo standarda koji postavljaju ograničenja i definiraju najbolju prasku za razvoj softvera. Standardi se najviše primjenjuju u području upravljanja kvalitetom proizvoda te za testiranje softvera. Zadatak upravljanja kvalitetom je određivanje standarda koji će se primjenjivati prilikom testiranja proizvoda. [33, str. 217]

Postoje različiti izvori standarda. Standard može proizaći iz organizacije te su u njemu definirane interne direktive, procedure i smjernice. Također standard može nastati od strane upravitelja kvalitetom. Ovakvim standardima određuju se minimalni zahtjevi koji moraju biti zadovoljeni. Osim toga postoje i standardi definirani za određeni industrijski sektor. [33, str. 217]

Osim internih standarda postoje i opće propisani standardi. Najpoznatiji ovakav standard je Međunarodno standard testiranja softvera, tj. eng. *The International Software Testing Standard*. [14] U nastavku je opisan ovaj standard te su navedeni još neki standardi testiranja softvera.

6.1. ISO/IEC/IEEE 29119

ISO/IEC/IEEE 29119 Software Testing je međunarodno priznati skup standarda za testiranje softvera koji se mogu koristiti u bilo kojem životnom ciklusu proizvoda ili u bilo kojoj organizaciji. Primjenom ovih standarda organizaciji je pružen visokokvalitetan pristup testiranje koji je prihvatljiv po cijelom svijetu. Trenutno se sastoji od pet standarda: [14]

1. ISO/IEC/IEEE 29119-1: Koncepti i definicije
2. ISO/IEC/IEEE 29119-2: Procesi testiranja
3. ISO/IEC/IEEE 29119-3: Testna dokumentacija
4. ISO/IEC/IEEE 29119-4: Tehnike testiranja
5. ISO/IEC/IEEE 29119-5: Testiranje na temelju ključnih riječi

6.1.1. ISO/IEC/IEEE 29119-1: Koncepti i definicije

Ovaj standard se fokusira na definicije i koncepte korištene u ostalim standardima u skupini 29119 standarda. Ovaj standard služi korisniku kao pomoć pri razumijevanju rječnika koji se primjenjuje u ostalim standardima te daje primjere rada koncepata u praksi. [22] Neke od tema u standardu su uvod u testiranje softvera, testni procesi i potprocesi, prakse u testiranju, automatsko testiranje, itd.[7]

6.1.2. ISO/IEC/IEEE 29119-2: Procesi testiranja

Standard koji se odnosi na procese testiranja bio je dizajniran sa ciljem kreiranja generičkog procesnog modela koji se može koristiti za testiranje softvera. Prema ovom standardu proces testiranja podijeljen je na tri razine – organizacijska, upravljačka i dinamičko testiranje. [22] Ovaj standard je spominjan u poglavlju o procesima testiranja.

6.1.3. ISO/IEC/IEEE 29119-3: Testna dokumentacija

Standard 29119-3 fokusiran je na dokumentaciju pri testiranju te pruža standardizirane predloške koji su dizajnirani za pokrivanje cijelog životnog ciklusa proizvoda. Ovisno o potrebama organizacije dane predloške je moguće uređivati i prilagođavati. Predlošci su sastavljeni tako da odgovaraju testnom procesu iz standarda 29119-2. [22] U ovom standardu su opisani dokumenti poput plana testiranja, izvještaja o testiranju, specifikacije sustava, zahtjeva sustava, itd. [9]

6.1.4. ISO/IEC/IEEE 29119-4: Tehnike testiranja

Četvrti standard u skupini fokusiran je na tehnike dizajniranja testova. Ove tehnike primjenjive su kod pisanja testnih scenarija koji mogu biti dokazi da su ispunjeni svi zahtjevi ili da su pronađeni nedostaci u sustavu. [22] Osim toga, ovaj standard pruža informativne definicije različitih tipova testiranja i primjere testnih scenarija. Prema standardu, tehnike dizajniranja testova podijeljene su na tehnike temeljene na specifikaciji, tehnike temeljene na strukturi te tehnike temeljene na iskustvu. [10]

6.1.5. ISO/IEC/IEEE 29119-5: Testiranje na temelju ključnih riječi

Posljednji standard u skupini 29119 podržava one tehnike i pristupe koji omogućuju testiranje prema ključnim riječima. Ova tehnika primjenjuje se tako da se testni scenariji opišu na temelju unaprijed definiranog skupa ključnih riječi. Ključne riječi su pisane na prirodnom jeziku što olakšava shvaćanje testisnih scenarija. [22] Ovaj standard se sastoji od uvoda u testiranje na temelju ključnih riječi, primjene testiranja, opisa uloga i zadataka te okvira korištenja. [11]

6.2. Ostali standardi

Osim grupe standarada ISO/IEC/IEEE 29119, prilikom testiranja softvera mogu se primijeniti i ostali standardi.

Standard ISO/IEC 25010:2011 je standard orijentirana na zahtjeve i evaluaciju kvalitete softvera. Primjenom ovog standarda omogućeno je kvalitetnije definiranje zahtjeva sustava te načini evaluacije danih zahtjeva što može značajno poboljšati proces testiranja sustava. [21]

Standard IEEE 1028 definira zajednički skup aktivnosti za formalni pregled sustava. U standardu su navedeni različiti tipovi provjere i aktivnosti u procesu testiranja.

Standard ISO/IEC 12207 odnosi se procese životnog ciklusa softvera. Procese definira kao metode i standarde za poboljšanje procesa razvoja, potpornih procesa i procesa upravljanjem.

Standard IEEE 829 je standard koji specificira izgled dokumenata koji se koriste u osam faza testiranja softvera. U standardu su opisani dokumenti te koja je njihova svrha.

BS 7925-1 i BS 7925-2 su rječnici termina korištenih u testiranju softvera. Ovaj rječnik je objavljen u 1998. godini. [13]

Osim navedenih, postoje i standardi IEEE 1061, IEEE 1059, IEEE 1008, IEEE 1012, IEEE 1028, IEEE 1044, IEEE 1044-1, IEEE 830, IEEE 730, IEEE 1061, IEEE 12207, itd. [31]

7. Automatsko testiranje

Testiranje je jedan od ključnih koraka pri razvoju programskih proizvoda. Provedbom testiranja smanjujemo broj pogrešaka u sustavu te osiguravamo isporuku kvalitetnijeg sustava korisniku. Osim što testiranje možemo provoditi ručno, kao što je opisano u prethodnim poglavljima, proces testiranja moguće je automatizirati. U ovom poglavlju opisano je automatsko testiranje programa.

7.1. Pojam automatskog testiranja

Automatsko testiranje pojavilo se krajem 80-tih godina 20.stoljeća razvitkom prvih sustava za automatizaciju. [25] Prema Paulu Ammannu i Jeffu Offuttu, automatsko testiranje je proces korištenja softvera za kontrolu izvršenja testova, usporedbu stvarnih s predviđenim ishodima i za druge funkcije izvješćivanja o testiranju i kontroli. [30, str. 69]

Prema ovoj definiciji i vlastitom znanju o procesu testiranja mogu reći da je automatsko testiranje proces kontrole i brze provjere velikog skupa funkcionalnosti sustava te nam pruža izvještaje o razlici izlaznih i ulaznih parametara.

Paul Ammann i Jeff Offutt smatraju da testiranje softvera može biti skup i intenzivan posao, te je zbog toga cilj automatizirati što veći broj funkcionalnosti sustava. [30, str. 69] S druge strane Mark Fewster i Dorothy Graham navode kao je automatiziranje testiranja skuplje od ručnog testiranja. Kao razlog ovome navode potrebu za opreznim odabirom testova koji će se automatizirati. Smatraju da će test, ako je ispravno automatiziran, biti ekonomičniji za organizaciju. [28, str. 22] Osobno se slažem s ovim načinom razmišljanja, smatram da je prije automatizacije nužno napraviti analizu te odrediti koji će se dijelovi sustava automatizirati.

Kako bi se odredilo koji testovi će se automatizirati potrebno je provjeriti koji su od napisanih testova najpodložniji automatizaciji. Za ovo se upotrebljava pojam ispitljivosti testa (eng. *testability*). Paul Amman navodi da je ispitljivost stupanj do kojega sustav ili komponenta sustava olakšava određivanje testnih kriterija te provođenje testova kako bi se odredilo da li su zadovoljeni ti kriteriji. [30, str. 70]

U današnje vrijeme automatiziranje testiranje je sve češća pojava te skoro sve organizacije imaju bar dio funkcionalnosti pokriven s automatskim testovima. Razlog sve većem značaju automatiziranja testiranja možemo vidjeti iz jednog istraživanja. U svojoj knjizi o budućnosti testiranja Mukesh Sharma iznosi rezultate ispitivanja o povećanju automatizacije: 72 % ispitanika tvrdi da je poboljšano otkrivanje pogrešaka, 70 % ističe bolju kontrolu i transparentnost testova, 39 % je prepoznalo kraće cikluse testiranje te 66 % ističe smanjenje

troškova testiranja. [29, str. 113] Vjerujem da su ovakvi rezultati dovoljni da organizacije uvode automatizaciju testiranja u svoj proces razvijanja sustava.

7.2. Svrha automatskog testiranja

Iako je već navedeno da je osnovna svrha poboljšanje i ubrzavanje procesa testiranja, Mark Fewster i Dorothy Graham naveli su još nekoliko značajna automatizacije.

1. Omogućeno je pokretanje postojećih testova na novoj verziji programa. Ovim načinom napor za obavljanje regresijskih testova je minimalan. Budući da testovi već postoje i provedeni su na ranijoj verziji programa, moguće je odabrati postojeće testove te ih pokrenuti na novoj verziji. [28, str. 26] Osim pokretanja na novih verzijama automatske testove moguće je pokrenuti na različitim okolinama sustava te različitim web preglednicima.
2. Postojanje automatskih testova omogućuje češće pokretanje testova. Jedna od glavnih prednosti automatizacije je mogućnost pokretanja većeg broja testova u kraćem vremenu što omogućava češće pokretanje i provedbu testiranja. Na ovaj način se povećava sigurnost u ispravnost sustava. [28, str. 26] Također je moguće postaviti automatsko pokretanje testova te se na taj način testovi sami pokreću u određeno doba, npr. u ponoć.
3. Automatskim testiranje možemo izvršavati testove koje je teško ili nemoguće provoditi ručno. Primjer takvog testa je provjera rada sustava kada ga koristi nekoliko stotinjaka ljudi. Ručnim testiranjem je gotovo nemoguće simulirati ovu situaciju, ali preko automatskih testova moguće je dobiti simulaciju željene situacije. [28, str. 26] Ovakvi testovi su potrebni za sustave s velikim brojem korisnika kako bi se osigurao ispravan rad sustava za veliki broj korisnika.
4. Automatskim testiranjem omogućuje se bolja raspodjela resursa. Automatiziranjem jednostavnih i ponavljajućih zadataka povećava se ispravnost sustava te je testeru omogućeno vrijeme za ostale zadatke, kao što je pisanje boljih testnih scenarija ili provođenje potrebnog ručnog testiranja. [28, str. 26] Ovakav slučaj se događa ukoliko postoji ponavljajući proces u sustavu koji zahtijeva puno vremena za ručno testiranje.
5. Omogućena je dosljednost i ponovljivost testiranja. Testove koji su automatizirani moguće je pokrenuti više puta, i iako će ulazni parametri biti jednaki, može doći do razlike u izlaznim parametrima. Na ovaj način dobiva se na konzistentnosti testiranja što je teško postići ručnim testiranjem. Osim toga, testove je moguće pokrenuti pod različitom konfiguracijom hardvera, na različitim operacijskim sustavima ili s različitim bazama podataka. [28, str. 27]

6. Moguće je ponovno korištenje automatskih testova. Rad koji je uložen na odlučivanje o testiranju, dizajniranje testiranja i izgradnju testova distribuiran je na mnoga pokretanja tih testova. Za testove koji će se ponovo koristiti potrebno je uložiti dosta vremena kako bi bili sigurni u njihovu pouzdanost. [28, str. 27] U praski se dijelovi automatskih testova mogu ponovo koristiti i na različnim projektima, ukoliko su funkcionalnosti slične.
7. Automatskim testiranje smanjuje se vrijeme potrebno za testiranje. Budući da proces testiranja traje znatno kraće, sustav će moći ranije biti isporučen korisniku. [28, str. 27] Ukoliko su osnovne funkcionalnosti sustava pokrivene automatskim testovima prije isporuke potrebno je pokrenuti testove kako bi se provjerile osnovne stvari, a tester onda ima slobodu baviti se ručnim testiranjem, najčešće istraživačkim testiranjem, kako bi se otkrile posebne greške u sustavu.
8. Automatskim testiranjem povećava se sigurnosti u sustav. Ukoliko dobro definiran skup automatskih testova prođe bez problema, povećava se sigurnost da u sustavu neće biti neugodnih iznenađenja nakon isporuke.

7.3. Problemi i ograničenost automatskog testiranja

Osim navođenja prednosti automatskog testiranja, Mark Fewster i Dorothy Graham naveli su i nekoliko problema kod automatskog testiranja: nerealna očekivanja, loša praksa testiranja, očekivanja da će se pronaći velik broj novih grešaka, lažan osjećaj sigurnosti, održavanje automatskih testova, tehnički problemi i organizacijski problemi. [28, str. 27-29] U nastavku sam svaki od ovih problema objasnila sa vlastitog stajališta i iskustva iz prakse.

1. Nerealna očekivanja
    - Menadžment, ali i članovi razvojnog tima često mogu imati prevelika očekivanja što se tiče automatskog testiranja. Često se vjeruje da će uvođenjem automatskog testiranja nestati potreba za ručnim testiranjem te da će se automatskih testiranjem pokriti cijeli sustav. Problem se javlja kada se shvati da to ipak nije moguće te da automatizacija ne može riješiti sve probleme.
2. Loša praksa testiranja
    - Ukoliko tester nije upoznat s dobrom praksom testiranja, vrlo je vjerojatno da i testovi koje napiše za automatizaciju neće biti kvalitetni. Automatiziranjem ne kvalitetnih testova neće se vidjeti povećanje efikasnosti nego će sav uloženi trud ispasti uzalud. Prije automatizacije potrebno je napisati smislene testove.
3. Očekivanja da će se pronaći velik broj novih grešaka

- Budući da je potrebno svaki sustav testirati ručno, pojavljivanje velikog broja grešaka pri automatskom testiranju nije realno. Smatram da je jedna od najboljih praksi ručno testirati svaku funkcionalnost kada je napravljena te nakon toga ju automatizirati. Ukoliko se radi na ovaj način velika je vjerojatnost da će se pri pokretanju automatskih testova pojaviti jako mali broj novih grešaka.

4. Lažan osjećaj sigurnosti
    - Često je mišljenje da ukoliko su automatski testovi ispravno izvršeni da u sustavu nema grešaka. Na ovaj način dođe do stopostotne sigurnosti u ispravnost sustava iako to nikad nije istina. Smatram da se detaljnim istraživačkim testiranjem puno bolje pokrije sustav nego samim automatskim testovima. Razlog tome je nemogućnost automatiziranja svih slučajeva do kojih se može doći istraživanjem sustava.
5. Održavanje automatskih testova
    - Svakom promjenom na sustavu potrebno je promijeniti automatski test. Često se dogodi da zbog promjene pozicije određenog elementa u sustavu dođe do greške pri pokretanju automatskih testova. Automatske testove potrebno je pisati na način da su jednostavni za održavati jer ukoliko se ne održavaju ubrzo će postati neiskoristivi.
6. Tehnički problemi
    - Do tehničkih problema može doći ukoliko alat za automatsko testiranje nije kompatibilan s testiranim sustavom ili okolinom sustava. Budući da su promjene u softveru česte, mogući su slučajevi kada alati za automatsko testiranje ne mogu pokriti nove elemente sustava. Osim toga, ukoliko sustav koji se testira nije razvijen na najbolji način mogu se dogoditi problemi s testiranjem.
7. Organizacijski problemi
    - Do organizacijskih problema dođe ukoliko menadžment ne shvaća potrebu za automatizacijom te ne želi odobriti resurse za automatiziranje sustava. Kako je već navedeno, inicijalni trošak automatiziranja je veći od troška ručnog testiranja te je čest slučaj da menadžment odbije automatizaciju ne vjerujući u uspjeh. Osim toga, često su potrebne dodatne edukacije zaposlenika zaduženih za automatsko testiranje kako bi bolje obavljali svoj posao, ali menadžment ne želi odobriti navedene troškove.

Unatoč navedenim problemima smatram da je vrlo korisno automatizirati proces testiranja. Automatizacijom dobro sastavljenih testnih scenarija smanjuje se vrijeme potrebno

za testiranje sustava što testeru omogućava kreativnost pri ručnom testiranju. Osim toga, testove koji su automatizirani moguće je provoditi više puta u kratkom vremenu čime se možemo svakodnevno osigurati da funkcionalnosti koje su automatizirane ispravno rade.

7.4. Alati za automatsko testiranje

Alat za automatsko testiranje je temelj automatizacije. Na tržištu postoji velik broj alata za automatiziranje. Neki alati omogućuju samo specifične vrste testiranja i rade samo s specifičnim jezicima, dok drugi podržavaju velik opseg aplikacija i nude velik broj funkcionalnosti i osobina. [38]

Prilikom odabira alata za testiranje potrebno je sastaviti popis zahtjeva od alata koji će se koristiti pri odabiru. Definiranje liste zahtjeva nije jednostavno jer je potrebno u obzir uzeti sve potrebe pri testiranju. [38] Kao primjer mogu se iskoristiti sljedeći zahtjevi: [18]

1. Godina nastanka
2. Tip aplikacija koje se testiraju
3. Cijena
4. Podržane platforme
5. Podržani skriptni jezici
6. Potrebne vještine programiranja
7. Jednostavnost korištenja

Prema navedenim zahtjevima napravljena je usporedba 5 najboljih alata za testiranje u 2018. godini: [38]

Alati	Selenium	Katalon Studio	Unified Functional Testing	TestComplete	watir
Godina nastanka	2004.	2005.	1998.	1999.	2008.
Tip aplikacija koje se testiraju	Web aplikacije	Web aplikacije, mobilne aplikacije	Web aplikacije, mobilne aplikacije, desktop aplikacije	Web aplikacije, mobilne aplikacije, desktop aplikacije	Web aplikacije
Cijena	Besplatno	Besplatno	$$$$	$$	Besplatno
Podržane platforme	Windows, Linux, OS X	Windows, Linux, OS X	Windows	Windows	Windows, Linux, OS X

Podržani skriptni jezici	Java, C#, Perl, Python, JavaScript, Ruby, PHP	Java/Groovy	VBScript	JavaScript, Python, VBScript, Delphi, C++, C#	Ruby
Potrebne vještine programiranja	Potrebne napredne vještine za integraciju alata	Nisu potrebne, preporučene za napredne skripte	Nisu potrebne, preporučene za napredne skripte	Nisu potrebne, preporučene za napredne skripte	Potrebne napredne vještine za integraciju alata
Jednostavnost korištenja	Potrebne napredne vještine za instalaciju i korištenje	Jednostavna instalacija i korištenje	Kompleksna instalacija, potrebna edukacija za korištenje	Jednostavna instalacija, potrebna edukacija za korištenje	Potrebne napredne vještine za integraciju alata

Slika 8. Usporedba alata za automatizaciju [18]

Za potrebe izrade praktičnog dijela diplomskog rada koristila sam alat Selenium koji je detaljnije opisan u nastavku. Osim njega, korišteni su dodatni alati.

7.4.1. Selenium – alat za automatsko testiranje

Selenium je alat za automatizaciju preglednika. Selenium je alat razvijen od strane organizacije ThoughtWorks u 2004. godini u Chicagu. [37] Selenium je skup različitih alata od kojih svaki ima drugačiji pristup automatizaciji softvera. Zbog ovoga omogućeno je velik broj testnih funkcionalnosti prilagođenih za testiranje web aplikacija. Ove operacije su iznimno fleksibilne kako bi se korisniku omogućilo jednostavno pristupanje svim elementima i uspoređivanje očekivanih rezultata s stvarnim ponašanjem. Jedna od glavnih prednosti Seleniuma je podrška pokretanju istih testova na nekoliko preglednika. [36]

Unutar Seleniumovog skupa alata postoji nekoliko sustava: [36]

1. Selenium 2 (Selenium WebDriver)
    - budući smjer projekta i najnoviji dodatak
    - pruža velik broj novih mogućnosti, uključujući i više objektno orijentiran API
    - podržava tehnologiju iz Selenium 1
2. Selenium 1 (Selenium RC)
    - vugo vremena bio je glavni projekt, dok se nije pojavio Selenium 2
    - trenutno nije aktivno održavan
3. Selenium IDE
    - Prototip alata za izgradnju testnih skripti

- dodatak za preglednike Google Chrome i Firefox
- ima mogućnost snimanja obavljenih akcija i njihovog ponovnog pokretanja

4. Selenium Grid
    - omogućava da se rješenja iz Selenium 1 primjene na veće skupine testova i omogućuje pokretanje testova na više okolina
    - povećava performanse velikih skupova testova
    - omogućava paralelno izvođenje na različitim okolinama

Za pisanje automatskih testova koristiti ću Selenium 2, tj. WebDriver, i preglednike Google Chrome i Firefox.

Selenium WebDriver podržava sljedeće preglednike i operacijske sustave: [36]

- Google Chrome
- Internet Explorer
- Firefox
- Safari
- Opera
- HtmlUnit
- phantomjs
- Android
- iOS

7.4.2. IntelliJ IDEA

IntelliJ IDEA je programska okolina ili integrirano razvojno okruženje namijenjeno većinom za programski jezik Java. Razvijen je od strane firme JetBrains u 2001. godini. [40]

IntelliJ omogućuje funkcionalnosti poput automatskog nadopunjavanja koda (nazivi klasa, metoda, polja i ključnih riječi), prepoznaje druge jezike osim Jave (npr. SQL, HTML, JavaScript, itd.), predviđa potrebe i generira ponavljajući programski kod, itd. [26]

7.4.3. Maven

Maven je alat za izgradnju i upravljanje projektima temeljenima na programskom jeziku Java. Maven omogućuje jednostavnu izgradnju procesa, jedinstven sustav izgradnje, kvalitetne informacije o projektu, smjernice za primjenu najbolje prakse i jednostavnu migraciju novih značajki. [16]

Maven se temelji na konceptu modela objekata projekta (eng. POM – *project object model*) koji će se primjenjivati na praktičnom primjeru. [17]

7.4.4. Cucumber

Cucumber je alat koji podržava razvoj temeljen na ponašanju (eng. BDD – *Behavior Driven Development*). BDD omogućava pisanje testova prihvatljivosti dok je sustav u fazi razvoja. Testovi koji su napisani koriste izraze običnog jezika tako da su razumljivi i krajnjim korisnicima. [44]

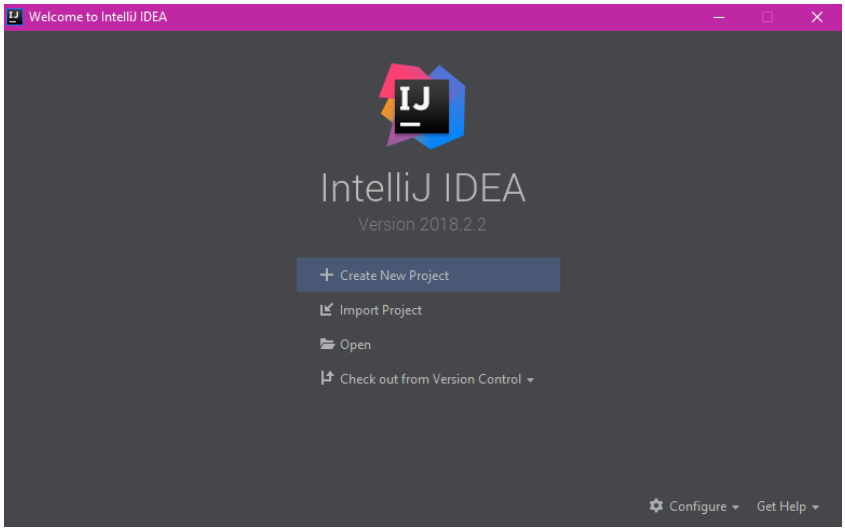
Cucumber je alat koji omogućuje čitanje jednostavnog jezika (npr. hrvatski, engleski) i omogućuje povezivanje sa programskim kodom. Na ovaj način se omogućuje pisanje automatskih testova koji će biti čitljivi svima, neovisno o znanju programskog jezika. [44]

7.4.5. Gherkin

Struktura jezika koji se koristi pri korištenju Cucumber-a naziva se Gherkin. Gherkin omogućava skup ključnih riječi na prirodnom jeziku koje se koriste za označavanje koraka testa. Neke od ključnih riječi su Osobina (eng. *Feature*), Zadano (eng. *Given*), Kada (eng. *When*). I (eng. *And*), Onda (eng. *Then*), Koncept (eng. *Scenario Outline*), Primjeri (eng. *Examples*) itd. [43] Ukupno je podržano oko 70 svjetskih jezika. [39]

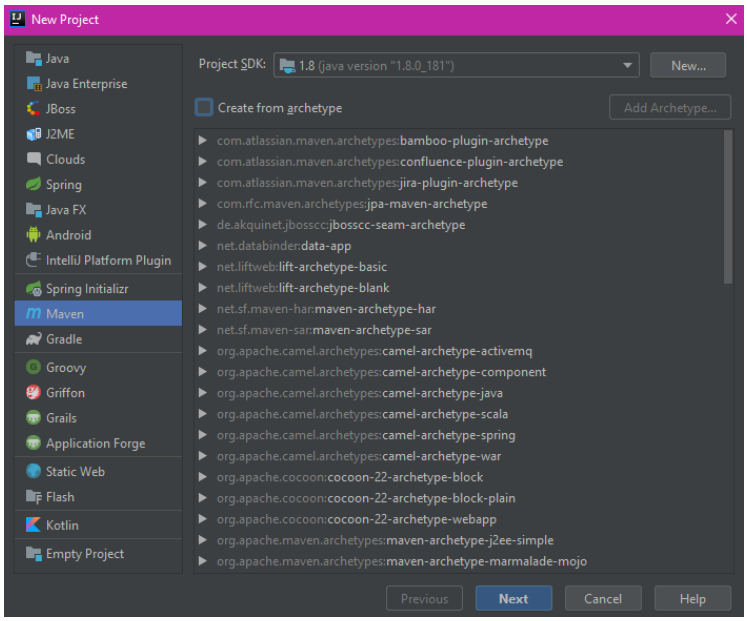
8. Kreiranje novog projekta

Za kreiranje novog projekta potreban nam je alat IntelliJ IDEA. Moguće ga je preuzeti s službene web stranice te instalirati na računalo. Nakon uspješne instalacije može se pokrenuti novi projekt.



Slika 9. IntelliJ - početni ekran

Odabirom opcije za kreiranje novog projekta IntelliJ nam nudi izbor nekoliko različitih projekata. Odaberemo Maven projekt i opciju za sljedeći korak.

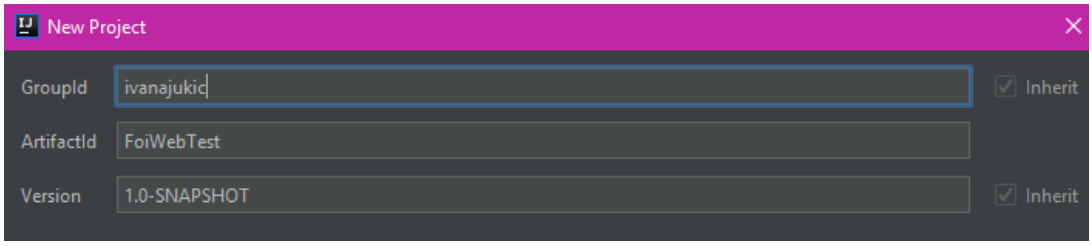


Slika 10. IntelliJ - odabir projekta

Nakon toga potrebno je unijeti sljedeće podatke:

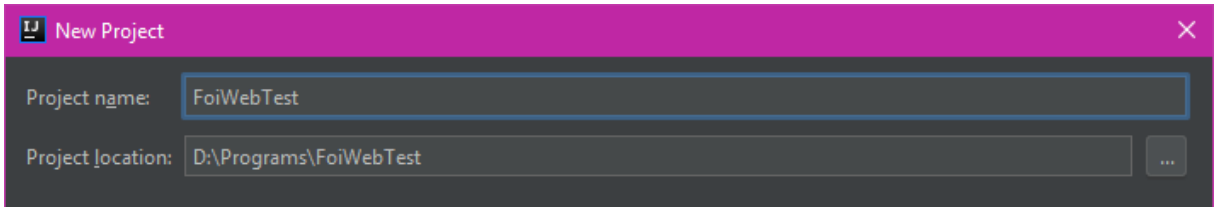
- GroupId – jedinstvena oznaka projekta
- ArtifactId – označava ime projekta

- Version – verzija projekta



Slika 11. IntelliJ – unošenje potrebnih podataka

Na idućem prozoru potrebno je odabrati lokaciju projekta.



Slika 12. IntelliJ - odabir lokacije projekta

Nakon postavljanja otvori se projekt. Prilikom kreiranja projekta Maven je generirao datoteku pom.xml u koju je potrebno dodati potrebne zavisnosti:

- Selenium – omogućuje upravljanje elementima stranice
- Cucumber – omogućuje pisanje testnih scenarija
- JUnit – služi za pisanje i pokretanje testova
- WebDriverManager – omogućuje lakše upravljanje potrebnim dodacima za preglednike

Unutar projekta potrebno je kreirati sljedeće datoteke:

- Google.feature
- TestSteps.java
- SharedSteps.java
- GooglePagePO.java
- runTest.java

8.1. Google.feature

Feature datoteke sastoje se od Gherkin programskog koda. U njoj je zapisan testni scenarij pomoću ključnih riječi. Budući da se sustav sastoji od funkcionalnosti, a svaka funkcionalnost može imati nekoliko testnih scenarija, potrebno je navesti ključne riječi koje označavaju funkcionalnost, odnosno scenarij.

Pisanje testnog scenarija počinje se s ključnom riječi „Osobina“ koja označava funkcionalnost. Nakon osobine piše se „Scenarij“, ili u ovom slučaju „Koncept“. Koncept omogućuje pokretanje scenarija na više načina, tj. u ovom slučaju za dva preglednika. Ukoliko se koristi ključna riječ scenarij nije moguće navesti primjere korištenja te je test moguće pokrenuti samo na jednom pregledniku.

Nakon koncepta pišu se koraci testa. Ključna riječ „Zadano“ označava stanje prije početka testa i u ovom koraku se izvršavaju svi procesi koji moraju biti izvršeni prije početka scenarija (u ovom slučaju je to kreiranje drivera i otvaranje preglednika). Riječ „<preglednik>“ nalazi se u primjerima te će se izvršiti dva testa, od kojih će jedan koristiti Chrome preglednik, drugi Firefox preglednik.

Dobra praksa je pisati testove na način:

- Testovi ne smiju ovisiti jedan o drugome.
- Svaki test provjerava samo jednu stvar.
- Aktivnosti koje se izvršavaju prije testa moraju biti prije koraka „Kada“, tj. u koraku Zadano.
- Test završava korakom „Onda“ gdje se provjerava uspješnost testa.
- Testovi moraju biti kratki i jednostavni. Bolje je imati veliki broj kratkih testova nego mali broj dugačkih testova.

Ukoliko je moguće test bi trebao sadržavati samo jednu ključnu riječ „Kada“ i jednu ključnu riječ „Onda“. Ukoliko se ove riječi ponavljaju velika je vjerojatnost da je test moguće podijeliti na dva manja testa.

```
Google.feature
1   #language: hr
2
3   Osobina: Otvori Google odi na FOI stranicu
4
5     Koncept: Otovori Google i upisi teskt
6       Zadano kreiran je web driver na "<preglednik>"
7       Kada otovorim stranicu google
8       I upisem tekst "FOI"
9       I odaberem gumb za pretrazivanje
10      I kliknem na FOI poveznicu
11      Onda se otovori FOI pocetna stranica
12
13      Primjeri:
14        |preglednik|
15        |Chrome    |
16        |Firefox   |
```

Slika 13. Primjer ispravno napisanog testnog scenarija

8.2. TestSteps.java

Datoteke s nazivom „Steps“ opisuju korake testnog scenarija. Cucumber preko ključnih riječi Gherkina pronalazi odgovarajući programski kod i pokreće određene funkcije. Unutar datoteke TestSteps opisani su svi koraci koji se koriste unutar jedne feature datoteke. Ukoliko se korak pojavljuje u bar dvije feature datoteke on se zapisuje u datoteku SharedSteps.

Unutar ove datoteke koraci testa su anotirani pomoću ključnih riječi Gherkina te se svaki korak odnosi na jednu funkciju. Unutar ovih funkcija moguće je pozivati funkcije za upravljanje elementima sa stranice iz s nazivom „po“ (po = page object) ili odrađivati zadatke koji nisu vezani uz same elemente stranice.

```
TestSteps.java

13
14  public class TestSteps {
15
16      private WebDriver driver;
17
18      @Given("^kreiran je web driver na \"([^\"]*)\"$")
19      public void kreirajDriver(String preglednik ) {
20          if (preglednik.equals("Chrome")) {
21              WebDriverManager.chromedriver().setup();
22              driver = new ChromeDriver();
23              SharedSteps.init(driver);
24          }
25          else {
26              Assert.fail();
27          }
28      }
```

Slika 14. Primjer koraka testnog scenarija

8.3. SharedSteps.java

Kao i datoteka TestSteps.java sadrži korake testnog scenarija. SharedSteps koristi se za zapis koraka koji su jednaki za nekoliko feature datoteka, tj. za nekoliko testnih scenarija. Budući da trenutno postoji samo jedna feature datoteka, u SharedSteps je samo inicijalizacija potrebnih objekata. Kreiranjem novih feature datoteka koraci koji se ponavljaju biti će zapisani u ovoj datoteci.

```
SharedSteps.java
1   package FoiTest;
2
3   import org.openqa.selenium.WebDriver;
4
5   public class SharedSteps {
6
7       public static GooglePagePO googlePage;
8
9       public static void init(WebDriver driver){
10          googlePage = new GooglePagePO(driver);
11      }
12  }
```

Slika 15. Primjer klase SharedSteps

8.4. GooglePO.java

Datoteke s nazivom „PO“ obavljaju akcije nad elementima web stranice. Ovdje se pronalaze elementi stranice te vrše akcije nad njima. Ovakav način odvajanja logike upravljanja elementima od testnih koraka olakšava održavanje automatskog testa jer ukoliko dođe do promjene na stranici koja se testira potrebno je promijeniti samo PO datoteku.

Na primjeru ispod se pomoću anotacije @FindBy i identifikacije pronađe element za upisivanje teksta za pretragu. U funkciji ispod se u pronađeni element upisuje tekst koji mu je proslijeđen

```
GooglePagePO.java
10  public class GooglePagePO {
11
12      @FindBy(id = "lst-ib")
13      private WebElement search;
14
15      public void insertText(String tekst) {
16          search.sendKeys(tekst);
17      }
18
```

Slika 16. Primjer pronalaska i upravljanja elementom

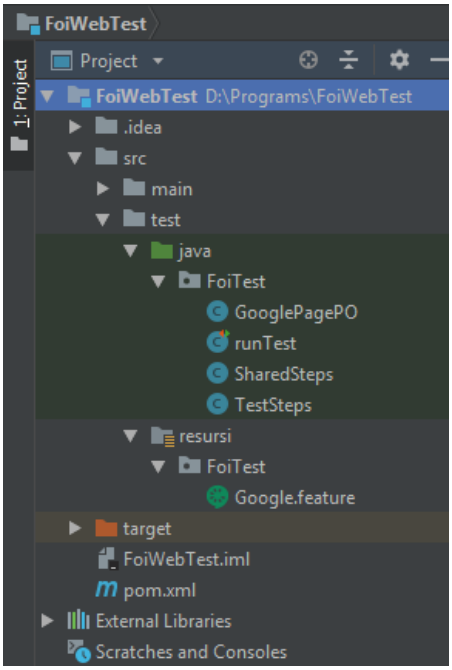
8.5. runTest.java

Posljednja datoteka koju je potrebno kreirati je runTest. Ova datoteka služi za pokretanje testa. Budući da u testu ne postoji funkcija „main“ koja se inače koristi za pokretanje programa, potrebno je dodati programski kod koji će to obavljati. U ovoj datoteci potrebno je dodati anotaciju koja označava da se program pokreće pomoću Cucumbera, tj. prvo će se pokrenuti feature datoteka.

```
runTest.java
1   package FoiTest;
2
3   import cucumber.api.CucumberOptions;
4   import cucumber.api.junit.Cucumber;
5   import org.junit.runner.RunWith;
6
7   @RunWith(Cucumber.class)
8   @CucumberOptions()
9
10  public class runTest {
11  }
```

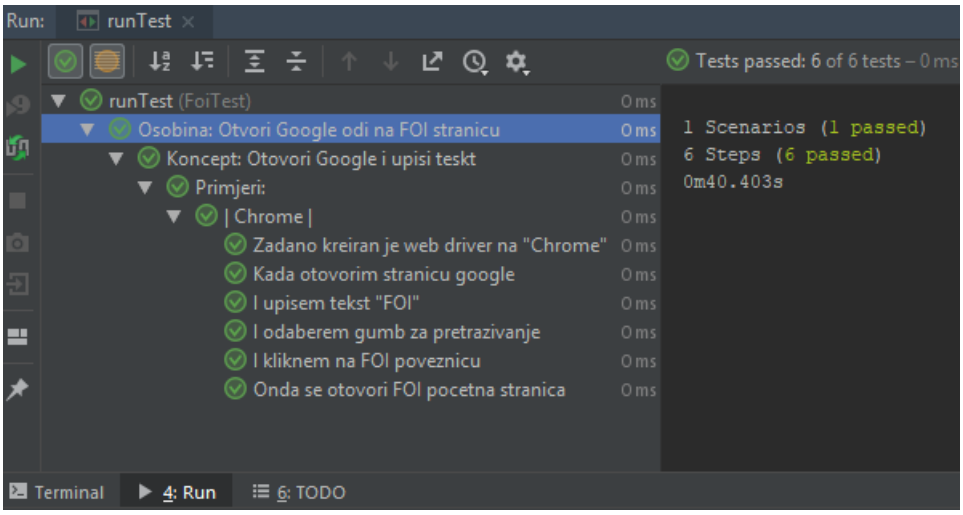
Slika 17. Primjer programskog koda za pokretanje testova

Struktura kreiranih datoteka izgleda kao na sljedećoj slici:



Slika 18. Struktura osnovnog projekta

Nakon kreiranja svih potrebnih datoteka te implementiranja programskog koda moguće je pokrenuti. Nakon pokretanja testa IntelliJ nam prikazuje rezultate po koracima testa.



Slika 19. Prikaz rezultata osnovnog testa

9. Praktični primjer testiranja FOI web mjesta

U praktičnom dijelu diplomskog rada provela sam testiranje nad sljedećim servisima:

- ELF sustav
- E-mail
- E-potrfolio
- Katalog knjižnice
- Kontakti
- Planer studija

Struktura praktičnog dijela podijeljena je na nekoliko elemenata:

- Resursi
- PageObject
- FoiTest

Skupina „Resursi“ sadrži testne scenarije, tj. u njoj se nalaze datoteke s nastavkom .feature. U skupini „PageObject“ se nalaze datoteke koje upravljaju elementima stranice. Sadržaj datoteka podijeljen je na način da svaka datoteka upravlja elementima samo jedne stranice – npr. datoteka „KnjiznicaPO“ upravljati će samo elementima sa servisa knjižnica. Posljednja skupina „FoiTest“ sadrži korake scenarija te datoteku za pokretanje testova. Koraci scenarija podijeljeni su po datotekama tako da koraci koji se odnose na rad na jednoj stranici se nalaze u jednoj datoteci – npr. koraci koji se odnose na kretanje po naslovnoj stranici nalaze se u datoteci „NaslovnaSteps.java“.

9.1. ELF sustav

Testirajući ELF sustav provjerila sam funkcionalnosti prijave u sustav, pokušaja prijave s neispravnim korisničkim imenom i lozinkom te odjave iz sustava. Testni scenariji se nalaze u datoteci „Elf.feature“. Testni scenariji su pisani pomoću Gherkin jezika te su korišten hrvatski jezik. Kao primjer naveden je testni scenarij prijave u sustav.

Koncept: Prijava u ELF sustav

Zadano otvorena je naslovna FOI stranica na pregledniku "**<preglednik>**"
Kada otvorim popis web servisa
I odaberem servis ELF
I odaberem opciju za prijavu
I unesem ispravno korisnicko ime i lozinku
I odaberem gumb za prijavu
Onda sam prijavljen u sustav

Primjeri:
|preglednik|
|Chrome |
|Firefox |

Ključne riječi Gherkin jezika označene su podebljanim slovima. Prvi korak, „Zadano“ otvara FOI naslovnu stranicu na jednome od preglednika – Chrome ili Firefox. U idućim koracima opisan je način pronalaska ELF sustava preko FOI servisa te proces prijave. Posljednji korak, „Onda“ označava provjeru uspješnosti izvršenja testa.

Nakon završetka pisanja testnih scenarija potrebno je definirati ponašanje pojedinog koraka. Za ovo su korištene datoteke „NaslovnaSteps.java“ i „LoginSteps.java“. U ovim datotekama kreirane su funkcije za pojedini korak te se poziva funkcija za rad s elementima stranice. Na primjer, korak „**I** unesem ispravno korisnicko ime i lozinku“ se nalazi u datoteci „LoginSteps.java“ i poziva funkcije za unos korisničkog imena i lozinke.

```
@I("^unesem ispravno korisnicko ime i lozinku$")
public static void unesemIspravnoKorisnickoImeILozinku() {
    SharedObjects.loginPage.unesiKorisnickoIme();
    SharedObjects.loginPage.unesiLozinku();
}
```

Unutar ove funkcije pozivaju se funkcije „unesiKorisnickoIme“ i „unesiLozinku“ koje djeluju nad elementima stranice za prijavu (loginPage).

Nakon definiranja svih koraka potrebno je kreirati funkcije za rad s elementima. Za prijavu u ELF sustav elementi sa stranice se nalaze u datotekama „ElfPO.java“ i „LoginPO.java“.

```
@FindBy(id = "username")
private WebElement korisnickoIme;

public void unesiKorisnickoIme() {
    SharedObjects.wait.until(ExpectedConditions.elementToBeClickable(korisnickoIme));
    korisnickoIme.sendKeys("test");
}
```

Funkcija „unesiKorisnickoIme“ koristi pronađen element „korisnickoIme“ i prosljeđujem mu zadani tekst. Element je pronađen pomoću anotacije „@FindBy“ i identifikacijske oznake elementa. Za provođenje testa korišteni su privatni podaci o korisničkom imenu i lozinki te se iz tog razloga u ovom dokumentu koristi tekst „test“.
Na ovaj način kreirani su svi koraci u testnim scenarijima.

9.2. E-mail sustav

Za sustav E-mail provela sam test prijave. U nastavku je testni scenarij.

Koncept: Prijava u sustav Email

Zadano otvorena je naslovna FOI stranica na pregledniku "**<preglednik>**"
Kada otovrim popis web servisa
I odaberem servis email
I unesem ispravno korisnicko ime i zaporku
I odaberem gumb za prijavu na email
Onda sam prijavljen na sustav email

Primjeri:

|preglednik|
|Chrome|
|Firefox|

Kao i za ELF sustav testiranje je provedeno na dva preglednika te su definirani koraci ponašanja sustava.

9.3. E-portfolio sustav

Za testiranje sustava e-portfolio provela sam testove prijave, kreiranja nove stranice te odjave iz sustava. U nastavku je testni scenarij kreiranja nove stranice.

Koncept: Kreiraj novu stranicu

Zadano prijavljen sam u sustav e-portfolio na pregledniku "**<preglednik>**"
Kada odaberem opciju za kreiranje nove stranice
I zapocnem kreiranje nove stranice
I unesem sve potrebne podatke
I odaberem opciju za spremanje
Onda je kreirana nova stranica

Primjeri:

|preglednik|
|Chrome	
|Firefox	

Testiranje je provedeno na preglednicima Chrome i Firefox. Kod ovog scenarija je zanimljivo primijetiti da je u koraku „Zadano“ definirano da obavljena prijava u sustav. Ovo je iz razloga što je prijava preduvjet testiranja stranice.

9.4. Sustav knjižnice

Sustav knjižnice testirala sam tako da sam provjerila pretragu literature po autoru, po naslovu knjige te po nepostojećem naslovu. Primjer testnog scenarija je prikazan ispod.

Koncept: Pretrazivanje po naslovu knjige

Zadano otvorena je naslovna FOI stranica na pregledniku "**<preglednik>**"
Kada otovrim popis web servisa
I odaberem servis katalog knjiznice
I unesem tekst "**Programiranje 2**" u polje za pretragu
I odaberem opciju pretrazivanja
Onda na popisu rezultata postoji knjiga "**Programiranje 2**"

Primjeri:
|preglednik|
|Chrome |

U ovom testnom scenariju u pretraživač se upisuje naziv „Programiranje 2“ te se vrši pretraga rezultata. Ukoliko je traženi rezultat pronađen test će proći uspješno.

9.4.1. Servis kontakti

U servisu kontakti provjeravala sam pretraživanje djelatnika FOI-a. Napisana su dva scenarija – pretraga po postojećem djelatniku te pretraga po nepostojećem djelatniku.

Koncept: Pretrazivanje po nepostojećem djelatniku

Zadano otvorena je naslovna FOI stranica na pregledniku "**<preglednik>**"
Kada otovrim popis web servisa
I odaberem servis kontakti
I pretrazujem po tekstu "**Djelatnik FOI-a**"
Onda je prikazana poruka da nisu pronađeni rezultati

Primjeri:
|preglednik|
|Chrome |
|Firefox |

U scenariju iznad u pretraživač se upisuje tekst „Djelatnik FOI-a“ te budući da ne postoji djelatnik s tim nazivom pretraga neće dati rezultate.

9.5. Servis planer

Posljednji servis koji sam testirala je planer studija. Ovdje sam provjeravala prijavu u sustav, odjavu iz sustava, registraciju korisnika te kreiranje novog plana.

Koncept: Pokretanje plana

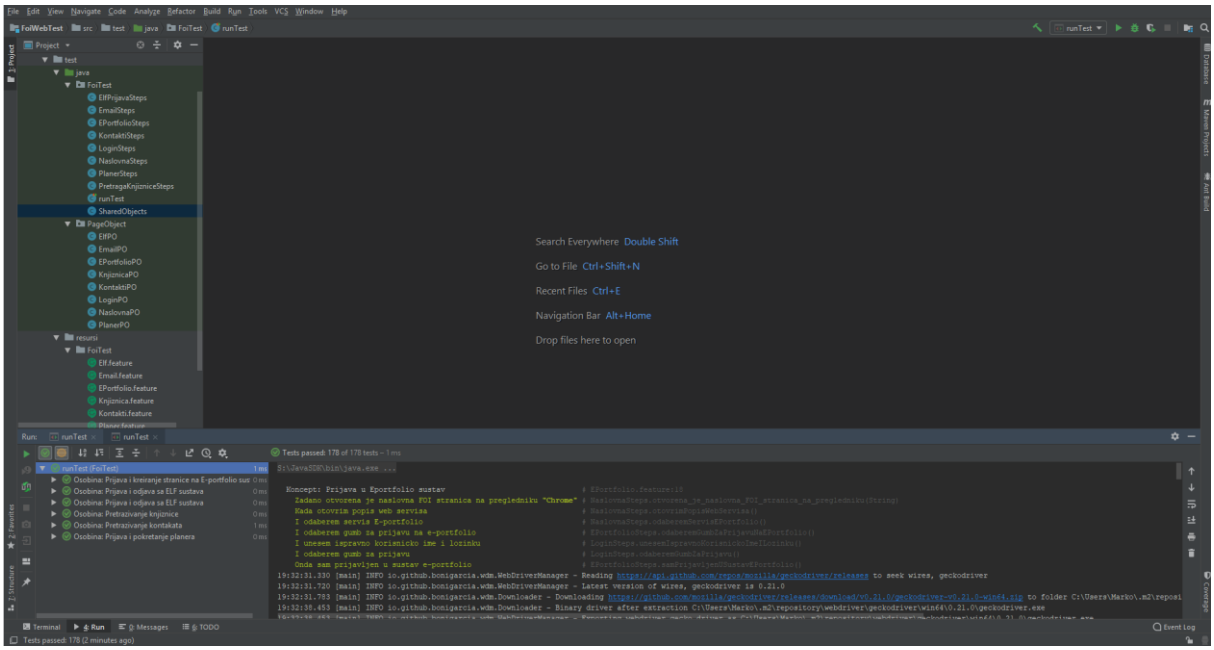
Zadano prijavljen sam na sustav planer na pregledniku "**<preglednik>**"
Kada zapocnem planiranje novog programa
I odaberem verziju plana
Onda je prikazan raspored kolegija

Primjeri:
|preglednik|
|Chrome |
|Firefox |

U ovom testnom scenariju smo prijavljeni u sustav te započinjemo s planiranjem novog programa. Nakon odabira verzije plana, tj. vrste studija, provjerava se postojanje rasporeda kolegija.

9.6. Rezultati testova

Nakon pokretanja testova alat IntelliJ nam omogućuje praćenje koraka scenarija kako bi lakše uočili moguće pogreške. Ukoliko su svi testovi uspješno izvršeni ekran izgleda kao na slici ispod.



Slika 20. Rezultati testova za FOI web servis

U donjem dijelu ekrana vidimo da su svi scenariji označeni zelenom bojom, tj. svi testovi su uspješno prošli.

9.7. Opažanja

Iako je FOI web sustav dizajnom iznimno privlačan, mogu reći da sam naišla na nekoliko problema prilikom testiranja. Smatram da je uvođenje FOI launcher-a dosta otežava rad sa sustavom te pronalaženje pojedinih servisa.

Što se tiče pojedinih servisa, ELF nisam bila u mogućnosti detaljno testirati budući da trenutno nemam dodanih predmeta.

Servis e-mail je vrlo nezgodan za testiranje te nisam bila u mogućnosti provesti testove nad njim. Servis onemogućava pronalaženje elemenata stranice što direktno utječe na nemogućnost provođenja testova.

Servis e-portfolio je bio relativno jednostavan za testiranje te smatram da je ovaj servis dobro napravljen.

Prilikom testiranja servisa knjižnice naišla sam na poteškoće prilikom dohvaćanja rezultata pretrage ali zaključujem da servis ispravno radi te nema potrebe za promjenama.

Na servisu kontakata sam naišla na pogreške pri implementaciji, ali su neznačajne za osnovnu funkcionalnost sustava. Također su se pojavile manje poteškoće pri dohvatu rezultata ali su uspješno riješene.

Posljednji servis, planer studija, stvarao je poteškoće pri pokretanju novog plana. Pretpostavljam da je došlo do zabune pri implementaciji te dva elementa dijele identifikacijsku oznaku. Unatoč tome, testove sam uspješno napisala.

Ponajveće probleme pri provođenju testiranja stvaralo mi je nedostatak identifikacijskih oznaka elemenata. Ukoliko element nema identifikacijsku oznaku, pronalazak i upravljanje elementom je otežano te je potrebno koristiti manje sigurne metode pronalaženja.

10. Zaključak

Unatoč mnogim mišljenjima, razvoj programskih proizvoda ne počinje s implementacijom nego mnogo ranije. Prije početka same implementacije potrebno je detaljno analizirati zahtjeve korisnika te specificirati buduće ponašanje sustava. Ukoliko su ove aktivnosti nedovoljno kvalitetno izvršene, tokom faze implementacije doći će do većih problema.

Osim procesa analize, kada se govori o razvoju softvera često se izostavljaju procesi testiranja proizvoda. Programske proizvode potrebno je detaljno testirati kako bi korisniku bio isporučen sustav s minimalnim nedostacima. Također, ovisnost uspjeha isporučenosti traženog sustava ovisi o fazi analize, tj. kako bi se isporučio ispravan proizvod potrebno je znati sva njegova ponašanja.

Proces testiranja sastoji se od provjeravanja rada sustava, bilo da se provjerava svaki segment ili sustav u cjelini. Ovisno o načinu rada organizacije, proces testiranja će se izvoditi na drugačiji način ali osnovna ideja je ista – pronaći nepravilnosti u sustavu.

Iako na prvu testiranje proizvoda zvuči kao jednostavan zadatak, to nije istina. Tester mora dobro razumjeti sustav, biti upoznat s detaljima funkcionalne specifikacije te moći uočiti razlike u ponašanju sustava od željenog ponašanja. Osim toga, tester blisko surađuje s razvojnim programerima te mu je zadatak pripomoći pri izradi kvalitetnog proizvoda. Tester mora imati mogućnost sagledavanja sustava iz korisničkog pogleda, ali i visok stupanj kreativnosti kako bi se pronašli neuobičajeni slučajevi korištenja.

Kako bi se olakšao posao testera te poboljšala sigurnost u ispravnost sustava, testni scenariji se mogu automatizirati. Automatizacijom se može postići značajno smanjenje ručnog testiranja, ali automatizacija može ispasti samo veliki trošak. Kako bi se osigurali da će uvođenje automatizacije biti vidljivih promjena potrebno je detaljno i kvalitetno specificirati testne scenarije za automatizaciju.

Iz osobnog stajališta mogu reći kako je automatizacija svakako korisna u procesu razvoja programskog proizvoda te da pomaže pri obavljanju ponavljajućih i zamarajućih zadataka.

Kroz ovaj diplomski rad navela sam na koje načine je moguće testirati programske proizvode te kada se koji način primjenjuje. Osim toga, navedene su moguće prednosti i nedostaci automatizacije testiranja. Smatram da je testiranje iznimno važan korak pri izradi

programskih proizvoda te da ukoliko bi se proces testiranja uklonio, isporučivali bi se proizvodi nezadovoljavajuće kvalitete što bi dovelo do gubitaka za organizacije.

11. Literatura

1. „5 Software Testing Methods“ (29. siječanj 2018), Test automation resources [Na internetu]. Dostupno: https://testautomationresources.com/software-testing-basics/software-testing-methods/ [pristupano 05. kolovoza 2018.]
2. „Ad hoc Testing“ (bez dat.), Software testing fundamentals [Na internetu]. Dostupno: http://softwaretestingfundamentals.com/ad-hoc-testing/ [pristupano 05. kolovoza 2018.]
3. „Agile Testing“ (bez dat.), Software testing fundamentals [Na internetu]. Dostupno: http://softwaretestingfundamentals.com/agile-testing/ [pristupano 04. kolovoza 2018.]
4. „Black Box Testing“ (bez dat.), Software testing fundamentals [Na internetu]. Dostupno: http://softwaretestingfundamentals.com/black-box-testing/ [pristupano 08. kolovoza 2018.]
5. „Definition of 'Unit Testing'“ The Economic Times [Na internetu]. Dostupno: https://economictimes.indiatimes.com/definition/unit-testing [pristupano 11. kolovoza 2018.]
6. „Gray Box Testing“ (bez dat.), Software testing fundamentals [Na internetu]. Dostupno: http://softwaretestingfundamentals.com/gray-box-testing/ [pristupano 08. kolovoza 2018.]
7. „ISO/IEC/IEEE 29119-1: Concepts & Definitions“ (10. listopad 2013) ISO/IEC/IEEE 29119 Software Testing [Na internetu]. Dostupno: http://www.softwaretestingstandard.org/part1.php [pristupano 04. kolovoza 2018.]
8. „ISO/IEC/IEEE 29119-2: Test Processes“ (24. listopad 2013) ISO/IEC/IEEE 29119 Software Testing [Na internetu]. Dostupno: http://www.softwaretestingstandard.org/part2.php [pristupano 04. kolovoza 2018.]
9. „ISO/IEC/IEEE 29119-3: Test Documentation“ (24. listopad 2013) ISO/IEC/IEEE 29119 Software Testing [Na internetu]. Dostupno: http://www.softwaretestingstandard.org/part3.php [pristupano 04. kolovoza 2018.]
10. „ISO/IEC/IEEE 29119-4: Test Techniques“ (24. listopad 2013) ISO/IEC/IEEE 29119 Software Testing [Na internetu]. Dostupno: http://www.softwaretestingstandard.org/part4.php [pristupano 04. kolovoza 2018.]
11. „ISO/IEC/IEEE 29119-5: Keyword-Driven Testing“ (24. listopad 2013) ISO/IEC/IEEE 29119 Software Testing [Na internetu]. Dostupno: http://www.softwaretestingstandard.org/part5.php [pristupano 04. kolovoza 2018.]

12. „method“ (26. kolovoz 2018) Merriam Webster. [Na internetu]. Dostupno: https://www.merriam-webster.com/dictionary/method [pristupano 02. kolovoza 2018.]
13. „Testing Standards List“ (21. studeni 2013) Proffesional Testing [Na internetu]. Dostupno: https://www.slideshare.net/dumethvah/testing-standards [pristupano 05. kolovoza 2018.]
14. „The International Software Testing Standard“ (10. rujan 2014) ISO/IEC/IEEE 29119 Software Testing [Na internetu]. Dostupno: http://www.softwaretestingstandard.org/ [pristupano 07. kolovoza 2018.]
15. „White Box Testing“ (bez dat.), Software testing fundamentals [Na internetu]. Dostupno: http://softwaretestingfundamentals.com/white-box-testing/ [pristupano 05. kolovoza 2018.]
16. Apache Maven Project, „Maven“ (03. rujan 2018). [Na internetu]. Dostupno: https://maven.apache.org/what-is-maven.html [pristupano 12. kolovoza 2018.]
17. Apache Maven Project, „Welcome to Apache Maven“ (03. rujan 2018). [Na internetu]. Dostupno: https://maven.apache.org/ [pristupano 12. kolovoza 2018.]
18. Brian, „Best Automation Testing Tools for 2018 (Top 10 reviews)“ (26. listopad 2017). [Na internetu]. Dostupno: https://medium.com/@briananderson2209/best-automation-testing-tools-for-2018-top-10-reviews-8a4a19f664d2 [pristupano 12. kolovoza 2018.]
19. C. Tozzi, „Quality Assurance and Software Testing: A Brief History“, 12. srpanj 2016. [Na internetu]. Dostupno: https://saucelabs.com/blog/quality-assurance-and-software-testing-a-brief-history [pristupano 02. kolovoza 2018.]
20. MTSolutions, blog, [Na internetu] http://www.mstsolutions.com/blog/content/generic-functional-testing-software-testing [pristupano 02. kolovoza 2018.]
21. ISO 9126, Standard [Na internetu] https://www.iso.org/standard/35733.html_25010 [pristupano 02. kolovoza 2018.]
22. TestBytes, Blog [Na internetu] https://www.testbytes.net/blog/software-testing-standards/ [pristupano 02. kolovoza 2018.]
23. ISO 9126 Software Quality Characteristics (bez dat.) „An overview of the ISO 9126-1 software quality model definition, with an explanation of the major characteristics.“. [Na internetu]. Dostupno: http://www.sqa.net/iso9126.html [pristupano 04. kolovoza 2018.]
24. ISQB (bez dat.), „*Standard Glossary of Terms used in Software Testing Version 3.2*“
25. J. Meerts i D. Graham (bez dat.), „The History of Software Testing“ [Na internetu]. Dostupno: http://www.testingreferences.com/testinghistory.php [pristupano 02. kolovoza 2018.]
26. JetBrains (bez dat.) „IntelliJ IDEA“. [Na internetu]. Dostupno: https://www.jetbrains.com/idea/ [pristupano 12. kolovoza 2018.]

27. L. Bradford, „Everything You Need to Know About Software Testing Methods“, 27. kolovoz 2018. Dostupno: https://www.thebalancecareers.com/all-you-need-to-know-about-software-testing-methods-4019921 [pristupano 03. kolovoza 2018.]
28. M. Fewster, D. Graham, *Software Test Automation: Effective use of test execution tools.* London: ACM Press Books, 1994.
29. M. Sharma, *Software Testing 2020: Preparing for New Roles*, Boca Raton: CRC Press, 2017. str 37
30. P. Ammann, J. Offutt, *Introduction to software testing,* Cambridgeshire: Cambridge University Press, 2017.
31. QASymphony (bez dat.) „Agile Methodology: The Complete Guide to Understanding Agile Testing“ [Na internetu]. Dostupno: https://www.qasymphony.com/blog/agile-methodology-guide-agile-testing/ [pristupano 04. kolovoza 2018.]
32. QASymphony (bez dat.) „Functional Testing Types – 25 Best Practices, Tips & More!“ [Na internetu]. Dostupno: https://www.qasymphony.com/blog/functional-testing-types/ [pristupano 04. kolovoza 2018.]
33. A. Spillner, T. Linz i H.Schafer, *Software testing foundations*, Santa Barbara: Rocky Nook Inc., 2014.
34. R. Patton, *Software Testing*, Sams Publishing, 26.srpanj 2005.
35. S. Naik, P. Tripathy, *Software testing and quality assurance: Theory and practice,* New Jersey: John Wiley & Sons, Inc., 2008.
36. SeleniumHQ (bez dat.) „Introduction“. [Na internetu]. Dostupno: https://www.seleniumhq.org/docs/01_introducing_selenium.jsp [pristupano 12. kolovoza 2018.]
37. SeleniumHQ (bez dat.) „Selenium History“. [Na internetu]. Dostupno: https://www.seleniumhq.org/about/history.jsp [pristupano 12. kolovoza 2018.]
38. SMARTBEAR (bez dat.), „Selecting Automated Testing Tools“. [Na internetu]. Dostupno: https://smartbear.com/learn/automated-testing/selecting-automated-testing-tools/ [pristupano 11. kolovoza 2018.]
39. T. Sundberg, „Languages supported by Cucumber-JVM“, (24. siječanj 2018). [Na internetu]. Dostupno: http://www.thinkcode.se/blog/2018/01/24/languages-supported-by-cucumberjvm [pristupano 11. kolovoza 2018.]
40. Techopedia (bez dat.) „IntelliJ IDEA“. [Na internetu]. Dostupno: https://www.techopedia.com/definition/7755/intellij-idea [pristupano 11. kolovoza 2018.]

41. TestingWhiz, „Types of Non Functional Software Testing“, 1. veljače 2012. [Na internetu]. Dostupno: https://www.testing-whiz.com/blog/types-of-non-functional-software-tests [pristupano 04. kolovoza 2018.]
42. TRY QA (bez dat.) „What is Structural testing (Testing of software structure/architecture)?“, [Na internetu]. Dostupno: http://tryqa.com/what-is-structural-testing-testing-of-software-structurearchitecture/ [pristupano 03. kolovoza 2018.]
43. Tutorialspoint (bez dat.) „Cucumber - Gherkins“. [Na internetu]. Dostupno: https://www.tutorialspoint.com/cucumber/cucumber_gherkins.htm [pristupano 09. kolovoza 2018.]
44. Tutorialspoint (bez dat.) „Cucumber - Overview“. [Na internetu]. Dostupno: https://www.tutorialspoint.com/cucumber/cucumber_overview.htm [pristupano 09. kolovoza 2018.]
45. Tutorialspoint (bez dat.) „Software Development Life Cycle“. [Na internetu]. Dostupno: https://www.tutorialspoint.com/software_engineering/software_development_life_cycle.htm [pristupano 01. kolovoza 2018.]
46. Tutorialspoint (bez dat.) „Software Testing - ISO Standards“. [Na internetu]. Dostupno: https://www.tutorialspoint.com/software_testing/software_testing_iso_standards.htm [pristupano 05. kolovoza 2018.]
47. Tutorialspoint (bez dat.) „Software Testing - Methods“. [Na internetu]. Dostupno: https://www.tutorialspoint.com/software_testing/software_testing_methods.htm [pristupano 02. kolovoza 2018.]
48. Tutorialspoint (bez dat.) „Structural Testing“. [Na internetu]. Dostupno: https://www.tutorialspoint.com/software_testing_dictionary/structural_testing.htm [pristupano 02. kolovoza 2018.]

12. Popis slika

Slika 1. Odnos vremena pronalaska greške i troškova ispravka [34, str. 45]..... 6
Slika 2. Vodopadni model [33, str. 33] 8
Slika 3. Poboljšani vodopadni model [45] 9
Slika 4. Proces testiranja prema ISO standardu [8] 12
Slika 5. Model testiranja crne kutije [4] 21
Slika 6. Model metode bijele kutije [1].....23
Slika 7. Model metode sive kutije [1].....24
Slika 8. Usporedba alata za automatizaciju [18] 35
Slika 9. IntelliJ - početni ekran 38
Slika 10. IntelliJ - odabir projekta.....38
Slika 11. IntelliJ – unošenje potrebnih podataka.....39
Slika 12. IntelliJ - odabir lokacije projekta 39
Slika 13. Primjer ispravno napisanog testnog scenarija.....40
Slika 14. Primjer koraka testnog scenarija.....41
Slika 15. Primjer klase SharedSteps.....42
Slika 16. Primjer pronalaska i upravljanja elementom.....42
Slika 17. Primjer programskog koda za pokretanje testova 43
Slika 18. Struktura osnovnog projekta 43
Slika 19. Prikaz rezultata osnovnog testa 44
Slika 20. Rezultati testova za FOI web servis 51

13. Prilog 1 – programski kod

13.1. Elf.feature

```
#language: hr
Osobina: Prijava i odjava sa ELF sustava

  Koncept: Prijava u ELF sustav
    Zadano otvorena je naslovna FOI stranica na pregledniku "<preglednik>"
    Kada otovrim popis web servisa
    I odaberem servis ELF
    I odaberem opciju za prijavu
    I unesem ispravno korisnicko ime i lozinku
    I odaberem gumb za prijavu
    Onda sam prijavljen u sustav
    Primjeri:
      |preglednik|
      |Chrome    |
      |Firefox   |

  Koncept: Prijava u ELF sustav s neipravnim podacima
    Zadano otvorena je naslovna FOI stranica na pregledniku "<preglednik>"
    Kada otovrim popis web servisa
    I odaberem servis ELF
    I odaberem opciju za prijavu
    I unesem neispravno korisnicko ime i lozinku
    I odaberem gumb za prijavu
    Onda se prikazuje poruka o neispravnosti podataka
    Primjeri:
      |preglednik|
      |Chrome    |
      |Firefox   |

  Koncept: Odjava iz ELF sustava
    Zadano prijavljen sam u ELF sustav koristeći preglednik "<preglednik>"
    Kada odaberem opciju za odjavu
    Onda sam odjavljen iz sustava
    Primjeri:
      |preglednik|
      |Chrome    |
      |Firefox   |
```

13.2. Email.feature

```
#language: hr
Osobina: Prijava i odjava sa ELF sustava

  Koncept: Prijava u sustav Email
    Zadano otvorena je naslovna FOI stranica na pregledniku "<preglednik>"
    Kada otovrim popis web servisa
    I odaberem servis email
    I unesem ispravno korisnicko ime i zaporku
    I odaberem gumb za prijavu na email
    Onda sam prijavljen na sustav email
    Primjeri:
      |preglednik|
      |Chrome    |
      |Firefox   |
```

13.3. EPortfolio.feature

```
#language: hr

Osobina: Prijava i kreiranje stranice na E-portfolio sustavu

Koncept: Prijava u Eportfolio sustav
    Zadano otvorena je naslovna FOI stranica na pregledniku "<preglednik>"
    Kada otovrim popis web servisa
    I odaberem servis E-portfolio
    I odaberem gumb za prijavu na e-portfolio
    I unesem ispravno korisnicko ime i lozinku
    I odaberem gumb za prijavu
    Onda sam prijavljen u sustav e-portfolio
    Primjeri:
      |preglednik|
      |Chrome    |
      |Firefox   |

  Koncept: Kreiraj novu stranicu
    Zadano prijavljen sam u sustav e-portfolio na pregledniku
"<preglednik>"
    Kada odaberem opciju za kreiranje nove stranice
    I zapocnem kreiranje nove stranice
    I unesem sve potrebne podatke
    I odaberem opciju za spremanje
    Onda je kreirana nova stranica
    Primjeri:
      |preglednik|
      |Chrome    |
      |Firefox   |

  Koncept: Odjava iz E-portfolio sustava
    Zadano prijavljen sam u sustav e-portfolio na pregledniku
"<preglednik>"
    Kada odaberem opciju za odjavu iz e-portfolia
    Onda sam odjavljen iz sustava e-portfolio
    Primjeri:
      |preglednik|
      |Chrome    |
      |Firefox   |
```

13.4. Knjiznica.feature

```
#language: hr
Osobina: Pretrazivanje knjiznice

  Koncept: Pretrazivanje po autoru knjige
    Zadano otvorena je naslovna FOI stranica na pregledniku "<preglednik>"
    Kada otovrim popis web servisa
    I odaberem servis katalog knjiznice
    I unesem tekst "Danijel Radošević" u polje za pretragu
    I odaberem opciju pretrazivanja
    Onda na popisu rezultata postoji knjiga "Programiranje 2"
    Primjeri:
      |preglednik|
      |Chrome    |
      |Firefox   |

  Koncept: Pretrazivanje po naslovu knjige
    Zadano otvorena je naslovna FOI stranica na pregledniku "<preglednik>"
    Kada otovrim popis web servisa
    I odaberem servis katalog knjiznice
    I unesem tekst "Programiranje 2" u polje za pretragu
    I odaberem opciju pretrazivanja
    Onda na popisu rezultata postoji knjiga "Programiranje 2"
    Primjeri:
      |preglednik|
      |Chrome    |
      |Firefox   |

  Koncept: Pretrazivanje po nepostojecem naslovu
    Zadano otvorena je naslovna FOI stranica na pregledniku "<preglednik>"
    Kada otovrim popis web servisa
    I odaberem servis katalog knjiznice
    I unesem tekst "50 nijansi sive" u polje za pretragu
    I odaberem opciju pretrazivanja
    Onda se pojavljuje poruka da nisu pronađeni rezultati
    Primjeri:
      |preglednik|
      |Chrome    |
      |Firefox   |
```

13.5. Kontakti.feature

```
#language: hr
Osobina: Pretrazivanje kontakata

  Koncept: Pretrazivanje po postojećem djelatniku
    Zadano otvorena je naslovna FOI stranica na pregledniku "<preglednik>"
    Kada otovrim popis web servisa
    I odaberem servis kontakti
    I pretrazujem po tekstu "Danijel Radošević"
    Onda je prikazan trazeni djelatnik
    Primjeri:
      |preglednik|
      |Chrome    |
      |Firefox   |

  Koncept: Pretrazivanje po nepostojećem djelatniku

    Zadano otvorena je naslovna FOI stranica na pregledniku "<preglednik>"
    Kada otovrim popis web servisa
    I odaberem servis kontakti
    I pretrazujem po tekstu "Djelatnik FOI-a"
    Onda je prikazana poruka da nisu pronađeni rezultati
    Primjeri:
      |preglednik|
      |Chrome    |
      |Firefox   |
```

13.6. Planer.feature

```
#language: hr
Osobina: Prijava i pokretanje planera

  Koncept: Pokretanje plana
    Zadano prijavljen sam na sustav planer na pregledniku "<preglednik>"
    Kada zapocnem planiranje novog programa
    I odaberem verziju plana
    Onda je prikazan raspored kolegija
    Primjeri:
      |preglednik|
      |Chrome    |
      |Firefox   |

  Koncept: Prijava u sustav sa postojecim korisnikom
    Zadano otvorena je naslovna FOI stranica na pregledniku "<preglednik>"
    Kada otovrim popis web servisa
    I odaberem servis planer
    I odaberem opciju za prijavu kao student
    I unesem ispravno korisnicko ime i lozinku
    I odaberem gumb za prijavu
    Onda sam prijavljen u sustav planer
    Primjeri:
      |preglednik|
      |Chrome    |
      |Firefox   |

  Koncept: Registracija korisnika
    Zadano otvorena je naslovna FOI stranica na pregledniku "<preglednik>"
    Kada otovrim popis web servisa
    I odaberem servis planer
    I odaberem opciju registracija
    I unesem potrebne podatke
    I potvrdim registraciju
    Onda se prikazuje poruka o uspjesnoj registraciji

    Primjeri:
      |preglednik|
      |Chrome    |
      |Firefox   |

  Koncept: Odjava iz sustava planer
    Zadano prijavljen sam na sustav planer na pregledniku "<preglednik>"
    Kada odaberem opciju za odjavu iz planera
    I potvrdim odjavu
    Onda sam odjavljen iz sustava planer
    Primjeri:
      |preglednik|
      |Chrome    |
      |Firefox   |
```

13.7. EflPrijavaSteps.java

```
package FoiTest;
import cucumber.api.java.hr.Kada;
import cucumber.api.java.hr.Onda;
import cucumber.api.java.hr.Zadano;
import org.junit.Assert;
public class ElfPrijavaSteps {
    private final String URL = "https://www.foi.unizg.hr/";
    @Zadano("^prijavljen sam u ELF sustav koristeći preglednik
\"([^\"]*)\"$")
    public void prijavljenSamUELFSustavKoristećiPreglednik(String
preglednik) {
        SharedObjects.init(preglednik);
        SharedObjects.driver.navigate().to(URL);
        NaslovnaSteps.otovrimPopisWebServisa();
        NaslovnaSteps.odaberemServisELF();
        LoginSteps.odaberemOpcijuZaPrijavu();
        LoginSteps.unesemIspravnoKorisnickoImeILozinku();
        LoginSteps.odaberemGumbZaPrijavu();
    }
    @Kada("^odaberem opciju za odjavu$")
    public void odaberemOpcijuZaOdjavu() {
        SharedObjects.loginPage.odaberiOpcijuZaOdjavu();
    }
    @Onda("^sam odjavljen iz sustava$")
    public void samOdjavljenIzSustava() {

Assert.assertTrue(SharedObjects.loginPage.prikazanaJePorukaOOdjavi());
    }
}
```

13.8. EmailSteps.java

```
package FoiTest;
import cucumber.api.java.hr.I;
import cucumber.api.java.hr.Onda;
import org.junit.Assert;
public class EmailSteps {
    @I("^unesem ispravno korisnicko ime i zaporku$")
    public void unesiKorisnickoImeiZaporku(){
        SharedObjects.emailPage.unesiImeIZaporku();
    }
    @I("^odaberem gumb za prijavu na email$")
    public void odaberemGumbZaPrijavuNaEmail() {
        SharedObjects.emailPage.odaberiGumbZaPrijavu();
    }
    @Onda("^sam prijavljen na sustav email$")
    public void samPrijavljenNaSustavEmail() {

Assert.assertTrue(SharedObjects.emailPage.provjeriDaLiSamPrijavljen());
    }
}
```

13.9. EPortfolioSteps.java

```
package FoiTest;
import cucumber.api.java.hr.I;
import cucumber.api.java.hr.Kada;
import cucumber.api.java.hr.Onda;
import cucumber.api.java.hr.Zadano;
import org.junit.Assert;
public class EPortfolioSteps {
    @I("^odaberem gumb za prijavu na e-portfolio$")
    public void odaberemGumbZaPrijavuNaEPortfolio() {
        SharedObjects.eportfolioPage.odaberiGumbZaPrijavu();
    }
    @Onda("^sam prijavljen u sustav e-portfolio$")
    public void samPrijavljenUSustavEPortfolio() {
        Assert.assertTrue(SharedObjects.eportfolioPage.provjeriPrijavu());
    }
    @Zadano("^prijavljen sam u sustav e-portfolio na pregledniku
\"([^\"]*)\"$")
    public void prijavljenSamUSustavEPortfolioNaPregledniku(String
preglednik) {

NaslovnaSteps.otvorena_je_naslovna_FOI_stranica_na_pregledniku(preglednik);
        NaslovnaSteps.otovrimPopisWebServisa();
        NaslovnaSteps.odaberemServisEPortfolio();
        odaberemGumbZaPrijavuNaEPortfolio();
        LoginSteps.unesemIspravnoKorisnickoImeILozinku();
        LoginSteps.odaberemGumbZaPrijavu();
        samPrijavljenUSustavEPortfolio();
    }
    @Kada("^odaberem opciju za kreiranje nove stranice$")
    public void odaberemOpcijuZaKreiranjeNoveStranice() {
        SharedObjects.eportfolioPage.odaberiOpcijuZaKreiranje();
    }
    @I("^zapocnem kreiranje nove stranice$")
    public void zapocnemKreiranjeNoveStranice() {
        SharedObjects.eportfolioPage.zapoceniKreiranjeStranice();
    }
    @I("^unesem sve potrebne podatke$")
    public void unesemSvePotrebnePodatke() {
        SharedObjects.eportfolioPage.unesiPodatke();
    }
    @I("^odaberem opciju za spremanje$")
    public void odaberemOpcijuZaSpremanje() {
        SharedObjects.eportfolioPage.odaberiGumbZaSpremanjeStranice();
    }
    @Onda("^je kreirana nova stranica$")
    public void jeKreiranaNovaStranica() {

Assert.assertTrue(SharedObjects.eportfolioPage.provjeriSpremanjeStranice())
;
    }
    @Kada("^odaberem opciju za odjavu iz e-portfolia$")
    public void odaberemOpcijuZaOdjavuIzEPortfolia() {
        SharedObjects.eportfolioPage.odaberiOpcijuZaOdjavu();
    }
    @Onda("^sam odjavljen iz sustava e-portfolio$")
    public void samOdjavljenIzSustavaEPortfolio() {
```

```
Assert.assertTrue(SharedObjects.eportfolioPage.provjeriPostojanjeOpcijeZaPr
ijavu());
    }
}
```

13.10. KontaktiSteps.java

```
package FoiTest;
import cucumber.api.java.hr.I;
import cucumber.api.java.hr.Onda;
import org.junit.Assert;
public class KontaktiSteps {
    String djelatnik ="";
    @I("^pretrazujem po tekstu \"([^\"]*)\"$")
    public void pretrazujemPoTekstu(String tekst) {
        djelatnik = tekst;
        SharedObjects.kontaktiPage.pretrazujPoTekstu(tekst);
    }
    @Onda("^je prikazan trazeni djelatnik$")
    public void jePrikazanTrazeniDjelatnik() {
Assert.assertTrue(SharedObjects.kontaktiPage.provjeriDaLiJePronađenDjelatni
k(djelatnik));
    }
    @Onda("^je prikazana poruka da nisu pronađeni rezultati$")
    public void jePrikazanaPorukaDaNisuPronađeniRezultati() {

Assert.assertTrue(SharedObjects.kontaktiPage.provjeriDaLiPostojiPoruka());
    }
}
```

13.11. LoginSteps.java

```
package FoiTest;
import cucumber.api.java.hr.I;
import cucumber.api.java.hr.Onda;
import org.junit.Assert;
public class LoginSteps {
    @I("^unesem ispravno korisnicko ime i lozinku$")
    public static void unesemIspravnoKorisnickoImeILozinku() {
        SharedObjects.loginPage.unesiKorisnickoIme();
        SharedObjects.loginPage.unesiLozinku();
    }
    @I("^odaberem gumb za prijavu$")
    public static void odaberemGumbZaPrijavu() {
        SharedObjects.loginPage.klikniGumbPrijava();
    }
    @I("^odaberem opciju za prijavu$")
    public static void odaberemOpcijuZaPrijavu() {
        SharedObjects.elfPage.odaberiOpcijuPrijava();
    }
    @Onda("^sam prijavljen u sustav$")
    public void samPrijavljenUSustav() {

Assert.assertTrue(SharedObjects.elfPage.provjeriDaLiSamPrijavljen());
    }
    @I("^unesem neispravno korisnicko ime i lozinku$")
    public static void unesemNeispravnoKorisnickoImeILozinku() {
        SharedObjects.loginPage.unesiNeispravnoKorisnickoIme();
        SharedObjects.loginPage.unesiNeispravnuLozinku();
    }
    @Onda("^se prikazuje poruka o neispravnosti podataka$")
    public void sePrikazujePorukaONeispravnostiPodataka() {

Assert.assertTrue(SharedObjects.loginPage.prikazanaJePorukaONeispravnosti()
);
    }
}
```

13.12. NaslovnaSteps.java

```
package FoiTest;
import cucumber.api.java.hr.I;
import cucumber.api.java.hr.Kada;
import cucumber.api.java.hr.Zadano;
public class NaslovnaSteps {
    private static final String URL = "https://www.foi.unizg.hr/";
    @Zadano("^otvorena je naslovna FOI stranica na pregledniku
\"([^\"]*)\"$")
    public static void
otvorena_je_naslovna_FOI_stranica_na_pregledniku(String preglednik) {
        SharedObjects.init(preglednik);
        SharedObjects.driver.navigate().to(URL);
    }
    @Kada("^otovrim popis web servisa$")
    public static void otovrimPopisWebServisa() {
        SharedObjects.naslovnaPage.otvoriPopisWebServisa();
    }
    @I("^odaberem servis ELF$")
    public static void odaberemServisELF() {
        SharedObjects.naslovnaPage.otvoriElf();
    }
    @I("^odaberem servis katalog knjiznice$")
    public void otvorimServisKatalogKnjiznice(){
        SharedObjects.naslovnaPage.otvoriKatalogKnjiznice();
    }
    @I("^odaberem servis email$")
    public void odaberemServisEmail() {
        SharedObjects.naslovnaPage.otvoriEmail();
    }
    @I("^odaberem servis kontakti$")
    public void odaberemServisKontakti()  {
        SharedObjects.naslovnaPage.otvoriKontakte();
    }
    @I("^odaberem servis planer$")
    public static void odaberemServisPlaner() {
        SharedObjects.naslovnaPage.otvoriPlaner();
    }
    @I("^odaberem servis E-portfolio$")
    public static void odaberemServisEPortfolio()  {
        SharedObjects.naslovnaPage.otvoriEportoflio();
    }
}
```

13.13. PlanerSteps.java

```
package FoiTest;
import cucumber.api.java.hr.I;
import cucumber.api.java.hr.Kada;
import cucumber.api.java.hr.Onda;
import cucumber.api.java.hr.Zadano;
import org.junit.Assert;
public class PlanerSteps {
    @I("^odaberem opciju za prijavu kao student$")
    public void odaberemOpcijuZaPrijavuKaoStudent() {
        SharedObjects.planerPage.odaberiOpcijuZaPrijavuStudenta();
    }
    @Onda("^sam prijavljen u sustav planer$")
    public void samPrijavljenUSustavPlaner() {
        Assert.assertTrue(SharedObjects.planerPage.provjeriPrijavu());
    }
    @I("^odaberem opciju registracija$")
    public void odaberemOpcijuRegistracija() {
        SharedObjects.planerPage.odaberiOpcijuRegistracija();
    }
    @I("^unesem potrebne podatke$")
    public void unesemPotrebnePodatke() {
        SharedObjects.planerPage.unesiPotrebnePodatke();
    }
    @I("^potvrdim registraciju$")
    public void potvrdimRegistraciju() {
        SharedObjects.planerPage.potvrdiRegistraciju();
    }
    @Onda("^se prikazuje poruka o uspjesnoj registraciji$")
    public void sePrikazujePorukaOUspjesnojRegistraciji() {

Assert.assertTrue(SharedObjects.planerPage.provjeriPorukuORegistraciji());
    }
    @Zadano("^prijavljen sam na sustav planer na pregledniku
\"([^\"]*)\"$")
    public void prijavljenSamNaSustavPlanerNaPregledniku(String preglednik)
{

NaslovnaSteps.otvorena_je_naslovna_FOI_stranica_na_pregledniku(preglednik);
        NaslovnaSteps.otovrimPopisWebServisa();
        NaslovnaSteps.odaberemServisPlaner();
        odaberemOpcijuZaPrijavuKaoStudent();
        LoginSteps.unesemIspravnoKorisnickoImeILozinku();
        LoginSteps.odaberemGumbZaPrijavu();
        samPrijavljenUSustavPlaner();
    }
    @Kada("^zapocnem planiranje novog programa$")
    public void zapocnemPlaniranjeNovogPrograma() {
        SharedObjects.planerPage.zapocniPlaniranje();
    }
    @I("^odaberem verziju plana$")
    public void odaberemVerzijuPlana() {
        SharedObjects.planerPage.odaberiVerzijuPlana();
    }
    @Onda("^je prikazan raspored kolegija$")
    public void jePrikazanRasporedKolegija() {
```

```
Assert.assertTrue(SharedObjects.planerPage.provjeriPosotjanjeRasporeda());
    }
    @Kada("^odaberem opciju za odjavu iz planera$")
    public void odaberemOpcijuZaOdjavuIzPlanera() {
        SharedObjects.planerPage.klikniNaGumbOdjava();
    }
    @I("^potvrdim odjavu$")
    public void potvrdimOdjavu() {
        SharedObjects.planerPage.potvrdiOdjavu();
    }
    @Onda("^sam odjavljen iz sustava planer$")
    public void samOdjavljenIzSustavaPlaner() {
        Assert.assertTrue(SharedObjects.planerPage.provjeriOdjavu());
    }
}
```

13.14. PretragaKnjizniceSteps.java

```
package FoiTest;
import cucumber.api.java.hr.I;
import cucumber.api.java.hr.Onda;
import org.junit.Assert;
public class PretragaKnjizniceSteps {
    @I("^unesem tekst \"([^\"]*)\" u polje za pretragu$")
    public void unesemTekstUPoljeZaPretragu(String tekst) {
        SharedObjects.knjiznicaPage.unesiTekstZaPretragu(tekst);
    }
    @I("^odaberem opciju pretrazivanja$")
    public void odaberemOpcijuPretrazivanja() {
        SharedObjects.knjiznicaPage.pokreniPretrazivanje();
    }
    @Onda("^na popisu rezultata postoji knjiga \"([^\"]*)\"$")
    public void naPopisuRezultataPostojiKnjiga(String tekst) {

Assert.assertTrue(SharedObjects.knjiznicaPage.provjeriDaLiPostojiRezltat(te
kst));
    }
    @Onda("^se pojavljuje poruka da nisu pronađeni rezultati$")
    public void sePojavljujePorukaDaNisuPronađeniRezultati() {

Assert.assertTrue(SharedObjects.knjiznicaPage.provjeriDaLiPrikazanaPoruka()
);
    }
}
```

13.15. runTest.java

```
package FoiTest;
import cucumber.api.CucumberOptions;
import cucumber.api.junit.Cucumber;
import org.junit.runner.RunWith;
@RunWith(Cucumber.class)
@CucumberOptions(
        features = "src/test/resursi/FoiTest/Elf.feature",
        glue = "FoiTest",
        format = "pretty"
)
public class runTest {
}
```

13.16. SharedObjects.java

```
package FoiTest;
import PageObject.*;
import cucumber.api.java.After;
import io.github.bonigarcia.wdm.WebDriverManager;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.chrome.ChromeOptions;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.firefox.FirefoxOptions;
import org.openqa.selenium.support.ui.WebDriverWait;
public class SharedObjects {
    public static WebDriver driver;
    public static WebDriverWait wait;
    public static NaslovnaPO naslovnaPage;
    public static ElfPO elfPage;
    public static LoginPO loginPage;
    public static KnjiznicaPO knjiznicaPage;
    public static EmailPO emailPage;
    public static KontaktiPO kontaktiPage;
    public static PlanerPO planerPage;
    public static EPortfolioPO eportfolioPage;
    public static void init(String preglednik){
        if(preglednik.equals("Chrome")) {
            inicijalizirajChrome();
        }
        else {
            inicijalizirajFirefox();
        }
        wait = new WebDriverWait(driver, 10);
        naslovnaPage = new NaslovnaPO(driver);
        elfPage = new ElfPO(driver);
        loginPage = new LoginPO(driver);
        knjiznicaPage = new KnjiznicaPO(driver);
        emailPage = new EmailPO(driver);
        kontaktiPage = new KontaktiPO(driver);
        planerPage = new PlanerPO(driver);
        eportfolioPage = new EPortfolioPO(driver);
    }
    private static void inicijalizirajChrome(){
        WebDriverManager.chromedriver().setup();
        ChromeOptions chromeOptions = new ChromeOptions();
        //chromeOptions.addArguments("headless");
        chromeOptions.addArguments("--start-maximized");
        driver = new ChromeDriver(chromeOptions);
    }
    private static void inicijalizirajFirefox(){
        WebDriverManager.firefoxdriver().setup();
        FirefoxOptions firefoxOptions = new FirefoxOptions();
        //firefoxOptions.addArguments("--headless");
        driver = new FirefoxDriver(firefoxOptions);
        driver.manage().window().maximize();
    }
    @After
    public void nakonTesta(){
        driver.close();
```

```
        }
}
```

13.17. ElfPO.java

```
package PageObject;
import FoiTest.SharedObjects;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;
import org.openqa.selenium.support.PageFactory;
import org.openqa.selenium.support.pagefactory.AjaxElementLocatorFactory;
import org.openqa.selenium.support.ui.ExpectedConditions;
public class ElfPO {
    @FindBy(css = "#page-wrapper > nav > ul.nav.navbar-nav.ml-auto >
li:nth-child(3) > div > span > a")
    private WebElement pokreniPrijavuButton;
    @FindBy(xpath = "//*[@id=\"dropdown-1\"]/span/span[1]")
    private WebElement korisnickoImeDropdown;
    public ElfPO(WebDriver driver){
        PageFactory.initElements(
                new AjaxElementLocatorFactory(driver, 10), this);
    }
    public void odaberiOpcijuPrijava() {
SharedObjects.wait.until(ExpectedConditions.elementToBeClickable(pokreniPri
javuButton));
        pokreniPrijavuButton.click();
    }
    public boolean provjeriDaLiSamPrijavljen() {
        return korisnickoImeDropdown.getText().equals("Ivana Jukić");
    }
}
```

13.18. EmailPO.java

```
package PageObject;
import FoiTest.SharedObjects;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;
import org.openqa.selenium.support.PageFactory;
import org.openqa.selenium.support.pagefactory.AjaxElementLocatorFactory;
import org.openqa.selenium.support.ui.ExpectedConditions;
public class EmailPO {
    @FindBy(name = "login_username")
    private WebElement username;
    @FindBy(name = "secretkey")
    private WebElement password;
    @FindBy(xpath =
"/html/body/form/table/tbody/tr/td/center/table/tbody/tr[3]/td/center/input
")
    private WebElement prijavaButton;
    @FindBy(xpath = "/html/head")
    private WebElement pretinci;
    public EmailPO(WebDriver driver){
        PageFactory.initElements(
                new AjaxElementLocatorFactory(driver, 10), this);
    }
    public void unesiImeIZaporku() {

SharedObjects.wait.until(ExpectedConditions.elementToBeClickable(username))
;
        username.sendKeys("test");
        password.sendKeys("test");
    }
    public void odaberiGumbZaPrijavu() {
        prijavaButton.click();
    }
    public boolean provjeriDaLiSamPrijavljen() {
        return true;
    }
}
```

13.19. EPortfolioPO.java

```
package PageObject;
import FoiTest.SharedObjects;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;
import org.openqa.selenium.support.FindBys;
import org.openqa.selenium.support.PageFactory;
import org.openqa.selenium.support.pagefactory.AjaxElementLocatorFactory;
import org.openqa.selenium.support.ui.ExpectedConditions;
public class EPortfolioPO {
    @FindBys({
            @FindBy(id = "sb-loginbox"),
            @FindBy(tagName = "button")
    })
    private WebElement loginBox;
    @FindBy(xpath = "/html/body/header/div/div/div[2]/button[2]")
    private WebElement userMenu;
    @FindBy(id = "home-info-create")
    private WebElement pokreniKreiranjeButton;
    @FindBy(id = "addview-button")
    private WebElement dodajElementButton;
    @FindBy(id = "add-view-button")
    private WebElement kreirajStranicuButton;
    @FindBy(id = "settings_title")
    private WebElement nazivStranice;
    @FindBy(id = "tinymce")
    private WebElement tekstStranice;
    @FindBy(id = "settings_submit")
    private WebElement spremiStranicuButton;
    @FindBy(id = "messages")
    private WebElement porukaOSpremljenosti;
    @FindBy(xpath = "/html/body/header/div/div/div[2]/button[2]/span[2]")
    private WebElement userOpcije;
    @FindBy(xpath = "//*[@id=\"navuser\"]/li[4]")
    private WebElement odjavaButton;
    @FindBy(className = "usermenu")
    private WebElement prijavaNaSustavOpcija;
    public EPortfolioPO(WebDriver driver){
        PageFactory.initElements(
                new AjaxElementLocatorFactory(driver, 10), this);
    }
    public void odaberiGumbZaPrijavu() {

SharedObjects.wait.until(ExpectedConditions.elementToBeClickable(loginBox))
;
        loginBox.click();
    }
    public boolean provjeriPrijavu() {

SharedObjects.wait.until(ExpectedConditions.visibilityOf(userMenu));
        return userMenu.isDisplayed();
    }
    public void odaberiOpcijuZaKreiranje() {
SharedObjects.wait.until(ExpectedConditions.elementToBeClickable(pokreniKre
iranjeButton));
        pokreniKreiranjeButton.click();
```

```
    }
    public void zapoceniKreiranjeStranice() {
SharedObjects.wait.until(ExpectedConditions.elementToBeClickable(dodajEleme
ntButton));
        dodajElementButton.click();
SharedObjects.wait.until(ExpectedConditions.elementToBeClickable(kreirajStr
anicuButton));
        kreirajStranicuButton.click();
    }
    public void unesiPodatke() {

SharedObjects.wait.until(ExpectedConditions.visibilityOf(nazivStranice));
        nazivStranice.clear();
        nazivStranice.sendKeys("Testna stranica");
    }
    public void odaberiGumbZaSpremanjeStranice() {
        spremiStranicuButton.click();
    }
    public boolean provjeriSpremanjeStranice() {

SharedObjects.wait.until(ExpectedConditions.visibilityOf(porukaOSpremljenos
ti));
        return porukaOSpremljenosti.isDisplayed();
    }
    public void odaberiOpcijuZaOdjavu() {

SharedObjects.wait.until(ExpectedConditions.elementToBeClickable(userOpcije
));
        userOpcije.click();

SharedObjects.wait.until(ExpectedConditions.elementToBeClickable(odjavaButt
on));
        odjavaButton.click();
    }
    public boolean provjeriPostojanjeOpcijeZaPrijavu() {
        return prijavaNaSustavOpcija.isDisplayed();
    }
}
```

13.20. KnjiznicaPO.java

```
package PageObject;
import FoiTest.SharedObjects;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;
import org.openqa.selenium.support.PageFactory;
import org.openqa.selenium.support.pagefactory.AjaxElementLocatorFactory;
import org.openqa.selenium.support.ui.ExpectedConditions;
import java.util.List;
public class KnjiznicaPO {
    @FindBy(id = "upit")
    private WebElement pretragaPolje;
    @FindBy(id = "trazi")
    private WebElement pretragaButton
    @FindBy(xpath = "//*[@id=\"printReady\"]/table[1]/tbody")
    private WebElement tablicaRezultata;
    @FindBy(xpath = "//*[@id=\"d1\"]/table/tbody/tr[2]/td")
    private WebElement porukaONepostojecimRezulattima;
    public KnjiznicaPO(WebDriver driver){
        PageFactory.initElements(
                new AjaxElementLocatorFactory(driver, 10), this);
    }
    public void unesiTekstZaPretragu(String tekst) {

SharedObjects.wait.until(ExpectedConditions.elementToBeClickable(pretragaPo
lje));
        pretragaPolje.sendKeys(tekst);
    }
    public void pokreniPretrazivanje() {

SharedObjects.wait.until(ExpectedConditions.elementToBeClickable(pretragaBu
tton));
        pretragaButton.click();
    }
    public boolean provjeriDaLiPostojiRezltat(String tekst) {
        List<WebElement> lista =
tablicaRezultata.findElements(By.tagName("tr"));
        System.out.println("ISPINT: " + lista.size());
        for(WebElement red : lista) {
            if(red.getText().contains(tekst)) {
                return true;
            }
        }
        return false;
    }
    public boolean provjeriDaLiPrikazanaPoruka() {
        return porukaONepostojecimRezulattima.isDisplayed();
}
}
```

13.21. KontaktiPO.java

```
package PageObject;
import FoiTest.SharedObjects;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;
import org.openqa.selenium.support.FindBys;
import org.openqa.selenium.support.PageFactory;
import org.openqa.selenium.support.pagefactory.AjaxElementLocatorFactory;
import org.openqa.selenium.support.ui.ExpectedConditions;
import java.util.List;
public class KontaktiPO {
    @FindBy(className = "search")
    private WebElement search;
    @FindBys({
            @FindBy(xpath =
"/html/body/div[2]/div[2]/div/div/div/div[2]/div/div[2]/div/div/div/div/mai
n/div[2]"),
            @FindBy(tagName = "li")
    })
    private List<WebElement> djelatnici;
    @FindBy(className = "fail")
    private WebElement porukaORezultatima;
    public KontaktiPO(WebDriver driver){
        PageFactory.initElements(
                new AjaxElementLocatorFactory(driver, 10), this);
    }
    public void pretrazujPoTekstu(String tekst) {

SharedObjects.wait.until(ExpectedConditions.elementToBeClickable(search));
        search.clear();
        search.sendKeys(tekst);
    }
    public boolean provjeriDaLiJePronađenDjelatnik(String tekst) {
        for(WebElement djelatnik : djelatnici) {
            if(!djelatnik.isDisplayed()) {
                continue;
            }
            if(djelatnik.getText().contains(tekst)) {
                return true;
            }
        }
        return false;
    }
    public boolean provjeriDaLiPostojiPoruka() {

SharedObjects.wait.until(ExpectedConditions.visibilityOf(porukaORezultatima
));
        return porukaORezultatima.isDisplayed();
    }
}
```

13.22. LoginPO.java

```
package PageObject;
import FoiTest.SharedObjects;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;
import org.openqa.selenium.support.PageFactory;
import org.openqa.selenium.support.pagefactory.AjaxElementLocatorFactory;
import org.openqa.selenium.support.ui.ExpectedConditions;
public class LoginPO {
    @FindBy(id = "username")
    private WebElement korisnickoIme;
    @FindBy(id = "password")
    private WebElement lozinka;
    @FindBy(name = "submit")
    private WebElement prijavaButton;
    @FindBy(css = "#fm1 > div")
    private WebElement porukaONeispravnosti;
    @FindBy(id = "dropdown-1")
    private WebElement userDropdown;
    @FindBy(id = "actionmenuaction-6")
    private WebElement odjavaButton;
    @FindBy(id = "content")
    private WebElement porukaOOdjavi;
    public LoginPO(WebDriver driver){
        PageFactory.initElements(
                new AjaxElementLocatorFactory(driver, 10), this);
    }
    public void unesiKorisnickoIme() {

SharedObjects.wait.until(ExpectedConditions.elementToBeClickable(korisnicko
Ime));
        korisnickoIme.sendKeys("ivajukic");
    }
    public void unesiLozinku() {

SharedObjects.wait.until(ExpectedConditions.elementToBeClickable(lozinka));
        lozinka.sendKeys("278xeHus");
    }
    public void klikniGumbPrijava() {

SharedObjects.wait.until(ExpectedConditions.elementToBeClickable(prijavaBut
ton));
        prijavaButton.click();
    }
    public void unesiNeispravnoKorisnickoIme() {

SharedObjects.wait.until(ExpectedConditions.elementToBeClickable(korisnicko
Ime));
        korisnickoIme.sendKeys("test");
    }
    public void unesiNeispravnuLozinku() {

SharedObjects.wait.until(ExpectedConditions.elementToBeClickable(lozinka));
        lozinka.sendKeys("test");
    }
    public boolean prikazanaJePorukaONeispravnosti() {
```

```
        return porukaONeispravnosti.isDisplayed();
    }
    public void odaberiOpcijuZaOdjavu() {

SharedObjects.wait.until(ExpectedConditions.elementToBeClickable(userDropdo
wn));
        userDropdown.click();

SharedObjects.wait.until(ExpectedConditions.elementToBeClickable(odjavaButt
on));
        odjavaButton.click();
    }
    public boolean prikazanaJePorukaOOdjavi() {
        return porukaOOdjavi.isDisplayed();
    }
}
```

13.23. NaslovnaPO.java

```
package PageObject;
import FoiTest.SharedObjects;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;
import org.openqa.selenium.support.PageFactory;
import org.openqa.selenium.support.pagefactory.AjaxElementLocatorFactory;
import org.openqa.selenium.support.ui.ExpectedConditions;
import java.util.List;
public class NaslovnaPO {
    @FindBy(id = "block-block-78")
    private WebElement webServisi;
    @FindBy(linkText = "ELF 2018/2019")
    private WebElement otvoriElfButton;
    @FindBy(linkText = "Katalog knjižnice")
    private WebElement katalogKnjiznice;
    @FindBy(linkText = "E-mail")
    private WebElement email;
    @FindBy(linkText = "Kontakti")
    private WebElement kontakti;
    @FindBy(className = "menu-icon")
    private List<WebElement> dodatno;
    @FindBy(linkText = "Planer studija")
    private WebElement planer;
    @FindBy(linkText = "E-portfolio")
    private WebElement otvoriEportfolioButton;
    public NaslovnaPO(WebDriver driver){
        PageFactory.initElements(
                new AjaxElementLocatorFactory(driver, 10), this);
    }
    public void otvoriPopisWebServisa(){

SharedObjects.wait.until(ExpectedConditions.elementToBeClickable(webServisi
));
        webServisi.click();
    }
    public void otvoriElf(){

SharedObjects.wait.until(ExpectedConditions.elementToBeClickable(otvoriElfB
utton));
        otvoriElfButton.click();
    }
    public void otvoriKatalogKnjiznice() {

SharedObjects.wait.until(ExpectedConditions.elementToBeClickable(katalogKnj
iznice));
        katalogKnjiznice.click();
    }
    public void otvoriEmail() {

SharedObjects.wait.until(ExpectedConditions.elementToBeClickable(email));
        email.click();
    }
    public void otvoriKontakte() {

SharedObjects.wait.until(ExpectedConditions.elementToBeClickable(dodatno.ge
```

```
t(0)));
        dodatno.get(0).click();

SharedObjects.wait.until(ExpectedConditions.elementToBeClickable(kontakti))
;
        kontakti.click();
    }
    public void otvoriPlaner() {

SharedObjects.wait.until(ExpectedConditions.elementToBeClickable(dodatno.ge
t(2)));
        dodatno.get(2).click();

SharedObjects.wait.until(ExpectedConditions.elementToBeClickable(planer));
        planer.click();
    }
    public void otvoriEportoflio(){
SharedObjects.wait.until(ExpectedConditions.elementToBeClickable(otvoriEpor
tfolioButton));
        otvoriEportfolioButton.click();
    }
}
```

13.24. PlanerPO.java

```
package PageObject;
import FoiTest.SharedObjects;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;
import org.openqa.selenium.support.PageFactory;
import org.openqa.selenium.support.pagefactory.AjaxElementLocatorFactory;
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.Select;
import java.util.Date;
public class PlanerPO {
    @FindBy(id = "loginButton")
    private WebElement prijavaButton;
    @FindBy(id = "foiLoginButton")
    private WebElement studentButton;
    @FindBy(id = "logoutButton")
    private WebElement odjavaButton;
    @FindBy(id = "registerButton")
    private WebElement registracijaButton;
    @FindBy(id = "username")
    private WebElement username;
    @FindBy(id = "password")
    private WebElement password;
    @FindBy(id = "confirmPassword")
    private WebElement confirmPassword;
    @FindBy(id = "name")
    private WebElement name;
    @FindBy(id = "email")
    private WebElement email;
    @FindBy(id = "register")
    private WebElement registerButton;
    @FindBy(id = "flash")
    private WebElement porukaORegistraciji;
    @FindBy(id = "planerButton")
    private WebElement planerButton;
    @FindBy(id = "studyVersion")
    private WebElement verzijePlanaDropdown;
    @FindBy(xpath = "//*[@id=\"studyPlanVersions\"]/input")
    private WebElement odaberiButton;
    @FindBy(id = "newStudyPlan")
    private WebElement studiji;
    @FindBy(xpath = "//*[@id=\"newStudyPlan\"]/input")
    private WebElement pokreniButton;
    @FindBy(id = "currentStudy")
    private WebElement trenutniPlan;
    @FindBy(id = "logoutConfirmButton")
    private WebElement potvrdaOdjaveButton;
    @FindBy(id = "content")
    private WebElement porukaOOdjavi;
    public PlanerPO(WebDriver driver){
        PageFactory.initElements(
                new AjaxElementLocatorFactory(driver, 10), this);
    }
    public void odaberiOpcijuZaPrijavuStudenta() {
```

```
SharedObjects.wait.until(ExpectedConditions.elementToBeClickable(prijavaBut
ton));
        prijavaButton.click();

SharedObjects.wait.until(ExpectedConditions.elementToBeClickable(studentBut
ton));
        studentButton.click();
    }
    public boolean provjeriPrijavu() {
        return odjavaButton.isDisplayed();
    }
    public void odaberiOpcijuRegistracija() {
SharedObjects.wait.until(ExpectedConditions.elementToBeClickable(registraci
jaButton));
        registracijaButton.click();
    }
    public void unesiPotrebnePodatke() {
        //DateFormat dateFormat = new SimpleDateFormat("yyyy/MM/dd
HH:mm:ss");
        Date date = new Date();

SharedObjects.wait.until(ExpectedConditions.visibilityOf(username));
        username.clear();
        username.sendKeys("NoName" + date);
        password.clear();
        password.sendKeys("NoName");
        confirmPassword.clear();
        confirmPassword.sendKeys( "NoName");
        name.clear();
        name.sendKeys("No Name");
        email.clear();
        email.sendKeys( "noname@gmail.com");
    }
    public void potvrdiRegistraciju() {
        registerButton.click();
    }
    public boolean provjeriPorukuORegistraciji() {
        return porukaORegistraciji.isDisplayed();
    }
    public void zapocniPlaniranje() {

SharedObjects.wait.until(ExpectedConditions.elementToBeClickable(planerButt
on));
        planerButton.click();
    }
    public void odaberiVerzijuPlana() {

SharedObjects.wait.until(ExpectedConditions.visibilityOf(verzijePlanaDropdo
wn));
        try {
            Thread.sleep(1000);
        } catch (InterruptedException ex) {
            Thread.currentThread().interrupt();
        }
        Select odaberiVerziju = new Select(verzijePlanaDropdown);
        odaberiVerziju.selectByVisibleText("1.2");
        odaberiButton.click();
        SharedObjects.wait.until(ExpectedConditions.visibilityOf(studiji));
        Select odaberiStudij = new
Select(studiji.findElement(By.tagName("select")));
```

```
        odaberiStudij.selectByVisibleText("Organizacija poslovnih
sustava");
        pokreniButton.click();
    }
    public boolean provjeriPosotjanjeRasporeda() {
        return trenutniPlan.isDisplayed();
    }
    public void klikniNaGumbOdjava() {

SharedObjects.wait.until(ExpectedConditions.elementToBeClickable(odjavaButt
on));
        odjavaButton.click();
    }
    public void potvrdiOdjavu() {
SharedObjects.wait.until(ExpectedConditions.elementToBeClickable(potvrdaOdj
aveButton));
        potvrdaOdjaveButton.click();
    }
    public boolean provjeriOdjavu() {

SharedObjects.wait.until(ExpectedConditions.visibilityOf(porukaOOdjavi));
        return porukaOOdjavi.isDisplayed();
    }
}
```