AUTOMATSKO TESTIRANJE PROGRAMSKIH RJEŠENJA NA MREŽI

Stankić, Natalie

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

EKONOMSKI FAKULTET

NATALIE STANKIĆ

AUTOMATSKO TESTIRANJE PROGRAMSKIH RJEŠENJA NA MREŽI

ZAVRŠNI RAD

Osijek, 2021.

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

EKONOMSKI FAKULTET

AUTOMATSKO TESTIRANJE PROGRAMSKIH RJEŠENJA NA MREŽI

ZAVRŠNI RAD

Mentor: Doc. dr. sc. Tomisav Jakopec

Student: Natalie Stankić

Predmet: Razvoj poslovnih aplikacija

Osijek, srpanj 2021.

IZJAVA

O AKADEMSKOJ ČESTITOSTI, PRAVU PRIJENOSA INTELEKTUALNOG VLASNIŠTVA, SUGLASNOSTI ZA OBJAVU U INSTITUCIJSKIM REPOZITORIJIMA I ISTOVJETNOSTI DIGITALNE I TISKANE VERZIJE RADA

1. Kojom izjavljujem i svojim potpisom potvrđujem da je završni rad isključivo rezultat osobnog rada koji se temelji na mojim istraživanjima i oslanja se na objavljenu literaturu. Potvrđujem poštivanje nepovredivosti autorstva te točno citiranje radova drugih autora i referiranje na njih.
2. Kojom izjavljujem da je Ekonomski fakultet u Osijeku, bez naknade u vremenski i teritorijalno neograničenom opsegu, nositelj svih prava intelektualnog vlasništva u odnosu na navedeni rad pod licencom Creative Commons Imenovanje-Nekomercijalno- Dijeli pod istim uvjetima 3.0 Hrvatska.
3. Kojom izjavljujem da sam suglasan/suglasna da se trajno pohrani i objavi moj rad u institucijskom digitalnom repozitoriju Ekonomskog fakulteta u Osijeku, repozitoriju Sveučilišta Josipa Jurja Strossmayera u Osijeku te javno dostupnom repozitoriju Nacionalne i sveučilišne knjižnice u Zagrebu (u skladu s odredbama Zakona o znanstvenoj djelatnosti i visokom obrazovanju, NN br. 123/03, 198/03, 105/04, 174/04, 02/07, 46/07, 45/09, 63/11, 94/13, 139/13, 101/14, 60/15).
4. Izjavljujem da sam autor/autorica predanog rada i da je sadržaj predane elektroničke datoteke u potpunosti istovjetan sa dovršenom tiskanom verzijom rada predanom u svrhu obrane istog.

Ime i prezime studenta/studentice: Natalie Stankić

JMBAG: 0010224591

OIB: 19822603336

e-mail za kontakt: natalie.stankic@gmail.com

Naziv studija: Preddiplomski sveučilišni studij (smjer: Poslovna informatika)

Naslov rada: Automatsko testiranje programskih rješenja na mreži

Mentor/mentorica rada: Doc. dr. sc. Tomislav Jakopec

U Osijeku, 2021.godine

Potpis 

SAŽETAK

Tematika ovog završnog rada je automatsko testiranje programskih rješenja na mreži. Prvobitni fokus je postepeno predstavljanje i tumačenje automatskog procesa testiranja uz sve prednosti i mane koje ono sa sobom donosi. Kroz rad se nastoji objasniti vrste i podvrste na koje se testiranje dijeli kroz dvije temeljne podjele na generičke vrste testiranja i vrste testiranja prema životnom ciklusu razvoja programa. Također pristup testiranju iz različitih uglova primjenom raznih metoda, a neke od najpoznatijih su: metoda agilnog testiranja, Ad-Hoc metoda i metode crne, sive i bijele kutije. Pojedinosti, prakse i načini korištenja metoda uz pozitivne i negativne strane detaljno su obuhvaćene kroz poglavlje o metodama testiranja. Procesi i faze provođenja testiranja programskog rješenja opisuju se kroz nekoliko osnovnih koraka od planiranja samog testiranja do zatvaranja testova. S obzirom na sveobuhvatnost tematike automatskog testiranja, detaljno je razjašnjeno što u globalu automatizacija i takav pristup testiranju jesu, koja je svrha i cilj njegovog korištenja, kao i problemi te ograničenosti s kojima se suočava. Spomenuti su i korisni alati za proces automatizacije testnog procesa kao i njihove temeljne mogućnosti uz funkcionalnosti koje pružaju. U konačnici je naveden praktičan primjer provođenja procesa testiranja pomoću Selenium WebDriver alata za automatizaciju.

Ključne riječi: automatsko testiranje, metode testiranja, vrste testiranja, razine i tipovi testiranja programskih rješenja, procesi, ciljevi i principi testiranja programa.

SADRŽAJ

1. UVOD

Digitalizacija, konstantni razvoj i težnja za boljim, kvalitetnijim, bržim i praktičnijim rješenjima obavljanja kako poslovnih tako i svakodnevnih privatnih stvari u užurbanom životu kojeg žive ljudi 21. stoljeća, rezultat su pojave brojnih inačica programskih rješenja. Fascinantna je činjenica koliko je zapravo stvari i mogućnosti dostupno umrežavanjem putem mrežnih stranica/aplikacija, svega u nekoliko klikova na osobnom računalu ili mobilnom telefonu s bilo kojeg mjesta u bilo koje vrijeme. Krajnjim korisnicima često ostaju nevidljivi sav rad, trud, znanje, vrijeme i zalaganje uloženo u projektiranje, osmišljavanje, izradu, testiranje i plasiranje samog programskog rješenja kako bi ono bilo funkcionalno, kvalitetno i sigurno za korištenje te u konačnici zadovoljilo korisničke potrebe. Za ispunjavanje tih uvjeta neizostavnu fazu u životnom ciklusu razvoja projekta čini testiranje programskih proizvoda. To je dugotrajan proces koji obuhvaća sve faze razvoja programskih rješenja od planiranja do primjene, a najbolje ga je provoditi kontinuirano nakon svake faze iako postoji praksa gdje se provodi tek nakon finaliziranja koda. Razlikujemo dvije osnovne podjele testiranja: ručno testiranje i automatsko testiranje. Glavni cilj testiranja je detektiranje što većeg broja grešaka u sustavu, kako bi se iste mogle uspješno ukloniti prije plasiranja proizvoda. Automatizacija nam uvelike pomaže u tom procesu jer štedi vrijeme, novac i resurse. Povećava efikasnost omogućavajući višestruko izvođenje ponavljajućih i učestalih testova, kao i njihovo spremanje te ponovnu upotrebu automatiziranih testnih knjižica, dokumentacije i skripti. Ukoliko je cilj testiranja specifično i jasno postavljen, te su nakon provođenja svih potrebnih testova rezultati u skladu s definiranim korisničkim zahtjevima za programsko rješenje, testiranje se smatra uspješnim, a aplikacija pogodnom za plasman i korištenje.

U ovom radu detaljno su obuhvaćene vrste testiranja i njihova podjela, kao i procesi, faze i metode za provođenje procesa testiranja. Također veći je fokus stavljen na automatsko testiranje i alate za automatizaciju korisne prilikom testiranja izvora na mreži. U konačnici primjenom teoretskog znanja u praksi, prikazan je jednostavan praktičan primjer testiranja.

2. TESTIRANJE

Test/testiranje/ispitivanje/kviz/kolokvij/pregled je namjerna radnja, postupak koji se koristi kako bi se ustvrdila funkcionalnost, ispravnost, učinkovitost, kvaliteta, izvornost i uspješno procijenila sposobnost te određene značajke temeljem kojih se promatranjem i vrednovanjem stvaraju predodžbe i mišljenja o objektu ispitivanja te otklanjaju nastale poteškoće i greške (Hrvatska enciklopedija, 2021).

Testiranje programa jedna je od ključnih i neophodnih faza razvoja programskog rješenja, a osoba zadužena za provođenje tog procesa, kako bi se u konačnici isporučio proizvod/aplikacija/program lišen grešaka i problema koji pruža željene rezultate i ispunjava zadane zahtjeve, naziva se testerom (Jelić, 2020).

Testiranje je proces provjere kvalitete, funkcionalnosti i valjanosti aplikacije s obzirom na prvobitno postavljenu svrhu, ciljeve i očekivanja. Testiranje se provodi na pojedinim dijelovima programa tijekom procesa razvoja ili nakon završetka izrade, kao i cjelokupnom proizvodu njegovim pokusnim izvođenjem. Rezultate testova potrebno je pažljivo i detaljno analizirati. Koristi ih se u svrhu unapređenja i poboljšanja rada aplikacije, otklanjanja nastalih pogrešaka i propusta, te kako bi pospješili učinkovitost, pouzdanost i izdržljivost aplikacije s što manjim utroškom vremena i resursa (Mayers, 2004).

Prema G. J. Myersu: „Svrha testiranja je pokazati da program ispravno izvršava svoje funkcionalnosti“. Dok ISTQB Glossary testiranje i cilj njegovog provođenja opisuje kao: „Proces koji se sastoji od aktivnosti, kako statičkih tako i dinamičkih, koje se bave planiranjem, pripremom i vrednovanjem programskih rješenja i srodnih proizvoda za rad kako bi se utvrdilo jesu li zadovoljili određene zahtjeve, kako bi pokazali da su prikladni za određenu svrhu te kako bi se otkrili njihovi nedostaci.“

Testiranje je korisno u svim fazama razvoja programa, korištenjem mnogo različitih razina, metoda i vrsta dobivamo potpuni uvid u kompletnu strukturu programskog rješenja i sve njegove segmente, to nam omogućuje upravljanje kvalitetom proizvoda i isporuku pouzdanog proizvoda što u konačnici čini temeljni cilj testiranja (Patton, 2006).

Čuvanjem provedenih testnih slučajeva i rezultata olakšava se uvid u stanje aplikacije. Testovi pružaju odlično rješenje jer pomažu u savladavanju i upoznavanju koda (jedinični testovi) te shvaćanju krajnjih ciljeva koje program mora zadovoljiti (sustavni testovi). Testni slučajevi olakšavaju daljnji razvoj, nadogradnju i izmjene programskog koda što uvelike pospješuje održavanje koda u današnje doba brzih izmjena tehnologija (Myers, 2012).

Slika 1. Trošak ispravljanja greške s obzirom na vrijeme detektiranja tijekom životnog ciklusa programskog rješenja

Izvor: Software testing help, 2021.

Zanemarivanjem ili odgodom provođenja testova, trošak testiranja i vrijeme potrebno za uklanjanje poteškoća rapidno rastu. Greške je potrebno pravovremeno detektirati kroz svaku od faza razvoja. U suprotnom troškovi intervencija uzrokovanih greškom u nekoj od završnih faza razvoja programa mogu biti i do nekoliko puta veći u odnosu na detekciju u početnim fazama.

.

3. VRSTE TESTIRANJA

Razlikujemo dvije temeljne vrste testiranja: ručno testiranje i automatsko testiranje. Ručno testiranje jedna je od najprimitivnijih vrsta testiranja, provodi se bez korištenja alata za testiranje, a može se koristiti zasebno ili provoditi prije procesa automatizacije testova. Osigurava pravilan rad mrežnih stranica i detaljnu provjeru performansi, korisničkog sučelja, dizajna i funkcionalnosti uz pronalazak i otklanjanje nedostataka. Provodi se korak po korak od strane testera koji analizira projektnu dokumentaciju i definira testni plan uz jasno napisane testne slučajeve. Njegova glavna zadaća je uočiti dolazi li do odstupanja u radu programa u odnosu na unaprijed zadano očekivano ponašanje. Ovaj način testiranja omogućuje kvalitetan uvid te prosuđivanje kvalitete i zadovoljstva radom programskog rješenja iz ugla krajnjeg korisnika (Hamilton, 2021).

Automatsko testiranje pogodno je za velike projekte jer se provodi uz pomoć alata za automatizaciju čime značajno smanjuje uloženo vrijeme i troškove. Prilikom automatizacije potrebno je formirati uspješnu strategiju i kvalitetne testne skripte koje se kroz proces prilagođavaju. Testovi se na ovaj način mogu ponovo iskoristiti i izvoditi u sličnim i ponavljajućim testnim slučajevima uz istu učinkovitost i točnost (Hamilton, 2021).

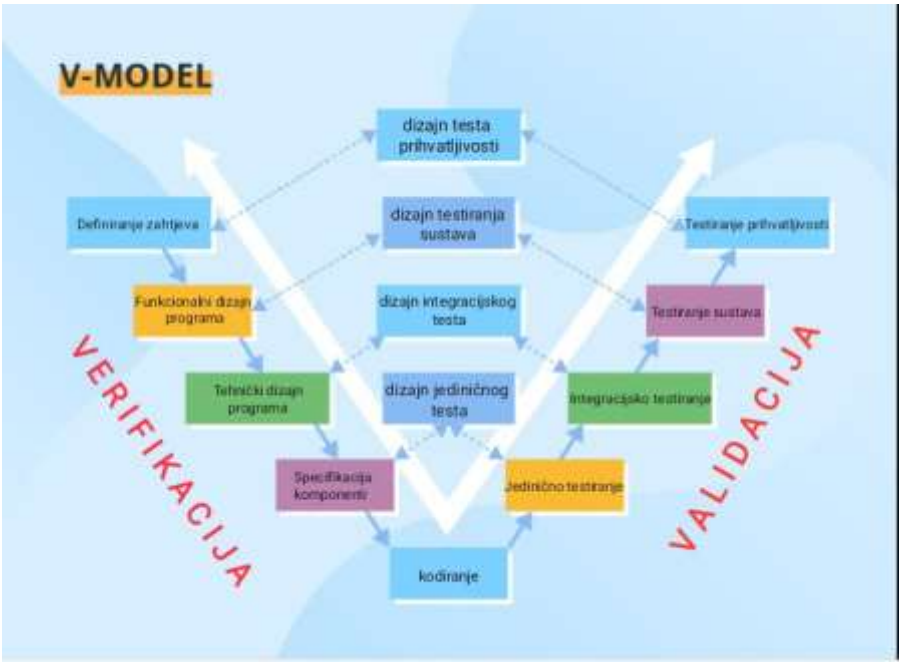
Ovisno o razini specifičnosti i razdoblju kada ih ubacujemo u proces razvoja, testove najčešće grupiramo prema mjestu u životnom ciklusu razvoja programskog rješenja ili prema generičkoj podjeli odnosno osnovnoj podjeli prema vrstama testiranja (GeeksforGeeks, 2020).

Ovakav način podjele prema vrstama detaljno je opisan kroz iduće poglavlje, dok će se na automatsko testiranje osvrnuti nešto kasnije kao zasebnoj temi.

3.1. VRSTE TESTIRANJA PREMA ŽIVOTNOM CIKLUSU RAZVOJA PROGRAMSKOG RJEŠENJA

Kroz svaku fazu razvoja programa provode se procesi analize i provjere poznatiji kao verifikacija i validacija. Verifikacija je provjera gradimo li proizvod na pravilan način u skladu s specifikacijama sustava na kraju razvoje faze. Koristi se kao jednostavniji i brži način testiranja programskih rješenja kojim ispitujemo funkcionalne i nefunkcionalne zahtjeve proizvoda. Validacija je analiza kojom ustvrđujemo gradimo li pravi proizvod koji zadovoljava očekivanje kupaca iz perspektive konačnog proizvoda. Potrebno ju je provoditi u ranim fazama kako bi se izbjegli veliki troškovi u kasnijim fazama razvoja. Program nastaje postepeno kroz faze te je zbog toga verifikacija koju provodimo nakon završetka pojedine faze, jednostavniji i

brži način testiranja programskih proizvoda, dok se validacija programa izvršava nakon izgradnje više funkcionalnosti programskog rješenja provjeravajući cjelokupni proizvod (Majstorović, 2011).



Slika 2. Shema V-modela testiranja proramskog rješenja

lijevo – životni ciklus integracije novih komponenti i testiranja programskog rješenja, desno – životni ciklus razvoja programskog rješenja

Izvor: Shiklo, 2019.

3.1.1. JEDINIČNO TESTIRANJE

Jedinično ili komponentno testiranje (eng. *Unit testing*) bazirano je na testiranju funkcionalnosti i točnosti pojedinih komponenti/jedinica programa. Ovisno o programskom jeziku jedinice možemo prepoznati kao klase, metode, funkcije i sl., a mogu se odnositi i na pojedini postupak ili program (Naik, Triphaty, 2008).

Cilj jediničnog testiranja je izolirati najmanje dijelove programskog rješenja, otkloniti greške te odrediti kompatibilnost funkcionalnosti s početno definiranim zahtjevima aplikacije. Na taj način osigurava se zaseban rad pojedinih komponenti programa. Testeru je tijekom provođenja testiranja dostupna unutarnja logika i struktura programa što upućuje na korištenje metode bijele kutije (eng. *white box*) (Spillner i dr., 2014).

Velika prednost jediničnog testiranja je implementacija promjena u kodu bez utjecaja i promjena na ostatku koda, osiguravanje kvalitetnog temelja za daljnji razvoj programa i detekcija grešaka u ranijim fazama razvoja što smanjuje vjerojatnost njihovog pojavljivanja u kasnijim fazama razvoja. Nedostatak jediničnog testa je što ne može provjeriti funkcionalnost cijelog programskog rješenja, već samo pojedinih komponenti programa (Dev.to, 2020).

3.1.2. INTEGRACIJSKO TESTIRANJE

Integracija je formiranje podsustava programa povezivanjem više programskih jedinica (Spillner, Linz, Schafer, 2014).

Integracijskim testiranjem provjerava se učinkovitost u radu i međusobna interakcija programskih jedinica, fokus je testirati komunikaciju i zajednički rad dva ili više dijelova sustava, te pronaći greške između različitih funkcionalnosti. Iako su prethodno provedeni jedinični testovi potrebno je provesti i integracijsko testiranje za provjeru ispravnosti suradnje različitih jedinica njihovim povezivanjem jedan po jedan ili istodobnim dodavanjem kosturu programskog rješenja. Stoga, vrlo je važno odabrati ispravan pristup i način integracije novih jedinica u smislene cjeline, većim opsegom integracije povećava se rizik i otežava izoliranje grešaka (Spillner i dr., 2014).

Najpoznatiji pristupi dodavanja programskih jedinica u sustav su: Inkrementalna integracija, integracija odozgo prema dole (eng. *Top-Down*), odozdo prema gore (eng. *Bottom-Up*), Sendvič integracija i istovremena integracija (eng. *Big-bang*) (Naik, Tripathy, 2008)

3.1.3. TESTIRANJE SUSTAVA

Testiranjem sustava ispitujemo kompletno programsko rješenje. Cilj testiranja sustava je provjeriti ispunjava li integrirani sustav specificirane korisničke zahtjeve, kompatibilnost aplikacije s funkcionalnim i nefunkcionalnim zahtjevima klijenta te je li programsko rješenje u skladu s propisanim standardima kvalitete. Ovom vrstom testiranja omogućava se validacija i verifikacija poslovnih zahtjeva i arhitekture programskog rješenja uz ispitivanje pouzdanosti, sigurnosti, opterećenja i performansi aplikacije. Testeri u ovoj fazi testiranja su uglavnom osobe koje nisu uključene u razvojni proces programskog rješenja stoga im je unutarnja struktura i logika samog koda nepoznata (Babbar, 2017). Prilikom testiranja koriste se metodom crne kutije (eng. *Black box*) provodeći osnovne, regresijske i funkcionalne testove, testove

interoperabilnosti, performansi i dr., dodatno provjeravajući program iz perspektive krajnjeg korisnika (Naik, Tripathy, 2008).

Glavni nedostatak testiranja sustava predstavljaju nedovoljno definirani, nejasni ili nepotpuni korisnički zahtjevi koji onemogućuju testerima kvalitetnu provjeru rada, reakcija i funkcionalnosti programa čime se propuštaju identificirati i ispraviti greške u programskom rješenju što znatno otežava zadovoljavanje korisničkih zahtjeva i traženih specifikacija (Spillner i dr., 2014).

3.1.4. TESTIRANJE PRIHVATLJIVOSTI

Prije puštanja programskog rješenja na masovno korištenje i proizvodnju provodi se testiranje prihvatljivosti, zadnje u životnom ciklusu programa. Najčešće ga provodi krajnji korisnik kako bi procijenio prihvatljivost programskog rješenja. Ispituje ispunjava li program sve specifikacije, zahtjeve i potrebe iz ugovora o isporuci programskog rješenja, te utvrđuje konačno zadovoljstvo i spremnost aplikacije za korištenje. Međutim, u procesu provjere prihvatljivosti, testiranje programskog rješenja provodi se i s poslovne strane. Testeri ili programeri utvrđuju uspješnost prolaska korisničkog testiranja, a redoslijed ispitivanja definira se ad-hoc (bez pripreme i planiranja) tijekom procesa koristeći se metodom crne kutije. Ispunjava li sustav uvjete za rad provjerava se operativnim testom prihvatljivosti, a funkcionalnost i spremnost sustava za korištenje potvrđuje se korisničkim testom. Testiranje prihvatljivosti s obzirom na okolinu dijeli se na dvije faze: interno/alfa i eksterno/beta testiranje (Spillner i dr., 2014).

Tablica 1. Usporedba internog i eksternog testiranja

INTERNO TESTIRANJE - ALFA	EKSTERNO TESTIRANJE - BETA
Provode ga testeri obično interni zaposlenici organizacije a nisu sudjelovali u kreiranju programa	Provode ga klijenti ili krajnji korisnici
Pri dubinskom Alfa testiranju ne provodi se ispitivanje sigurnosti	Tijekom Beta testiranja provjeravaju se pouzdanost, sigurnost i robusnost
Zahtijeva kontrolirano okruženje laboratorija ili okruženje za testiranje.	Ne zahtijeva kontrolirano okruženje. Programsko rješenje je dostupno javnosti u stvarnom vremenu.
Kritični problemi ili ispravci mogu se odmah dojaviti programerima.	Većina problema ili ispravak grešaka implementirat će se tek u budućim verzijama proizvoda.
Izvodi se na mrežnoj lokaciji za razvojne programere	Provodi se na lokaciji klijenta ili krajnjeg korisnika proizvoda
Uključuje tehnike bijele kutije (engl. white box) i crne kutije (engl. black box)	Beta testiranje obično koristi tehniku crne kutije
Dugotrajni ciklus izvođenja testiranja	Za provedbu testiranja potrebno je nekoliko tjedana.
Osigurava kvalitetu proizvoda prije prelaska na Beta testiranje	Također koncentrirano na kvalitetu proizvoda, okuplja korisničke doprinose o proizvodu i osigurava prilagođenost programa stvarnim korisnicima.

Izvor: Hamilton, 2021.

3.2. GENERIČKE VRSTE TESTIRANJA

Generičke vrste testiranja dijelimo na: funkcionalno testiranje, nefunkcionalno testiranje, testiranje strukture programskog proizvoda i regresijsko testiranje (Spillner i dr., 2014)

3.2.1. FUNKCIONALNO TESTIRANJE

Funkcionalno testiranje provodi se metodom crne kutije, a podrazumijeva: jedinično testiranje, integracijsko testiranje, testiranje sustava, testiranje prihvatljivosti, regresijsko testiranje, eng. *sanity testing* – površno testiranje većih dijelova sustava nakon učinjenih izmjena na izvornom kodu, eng. *smoke testing* – testiranje većih dijelova sustava kako bi se ustanovila ispravnost rada i dr. (Norville, 2017).

Za provođenje funkcionalnog testiranja potrebna je specifikacija funkcionalnih zahtjeva sustava koja pruža jasan uvid na koji način bi programsko rješenje trebalo funkcionirati. Nakon određivanja funkcionalnosti koje ulaze u testiranje definiraju se ulazni testni podaci na temelju kojih se formiraju očekivani izlazni podaci te izrađuju potrebni testni scenariji. Na kraju izvođenja testnih slučajeva uspoređuje su dobiveni rezultati s očekivanim rezultatima ponašanja sustava, ako testni slučajevi odgovaraju željenoj funkcionalnosti testiranje je završeno (Tutorialspoint, 2021).

Razlikuju se pozitivno i negativno funkcionalno testiranje. Pozitivno – potvrda funkcionalnosti sustava u skladu s specifikacijama. Negativno – pronalazak grešaka u funkcionalnosti sustava (Hamilton, 2021).

3.2.2. NEFUNKCIONALNO TESTIRANJE

Nefunkcionalno testiranje fokusira se na razmatranje atributa koji opisuju ponašanje funkcionalnosti sustava, karakteristike i kvalitetu izvedbe (Spillner i dr., 2014).

Cilj je postići jednostavnost i ugodnost korištenja sustava, bolje performanse i kvalitetu te povećati zadovoljstvo klijenata kroz ispunjavanje korisničkih očekivanja. Zahtjevi nefunkcionalnog testiranja moraju biti precizni i kvalitetno definirani kako bi se osigurala pouzdanost, upotrebljivost, učinkovitost, kompatibilnost i sigurnost programskog rješenja u skladu s percepcijom istih s gledišta krajnjeg korisnika (ISO/IEC, 2019).

Ovom vrstom testiranja ne provjeravaju se funkcionalnosti već se osigurava ispravnost i ugodnost korištenja istih. Ukoliko korisnički zahtjev glasi kako je za prijavu u sustav potrebna identifikacija, a implementirano je korisničko ime i lozinka te funkcionalnost kao takva postoji, može doći do korisnikovog nezadovoljstva ako je očekivana i neka do metoda biometrijske identifikacije. Testiranje uz precizne i jasne zahtjeve ključno je kako bi isporučeni proizvod zadovoljio.

Nefunkcionalno testiranje podrazumijeva (TestingWhiz, 2019): testiranje sigurnosti, performansi, volumena, opterećenja, stres testiranje, testiranje različitih konfiguracija, testiranje dokumentacije, kompatibilnosti, robusnosti, održivosti, upotrebljivosti i dr.

3.2.3. TESTIRANJE STRUKTURE PROGRAMSKOG PROIZVODA

Testiranjem strukture programskog proizvoda provjerava se struktura sustava, kako sustav obavlja svoje zadatke te što i kako se događa unutar same aplikacije. Ova metoda testiranja najčešće se koristi kod jediničnih i integracijskih testova. Prilikom kojih se osigurava postepeno testiranje svakog elementa u strukturi programskog rješenja, nakon svake nove implementacije elemenata s određenom svrhom u postojeću strukturu. Testiranje se izvodi uz poznavanje strukture i logike programa, metodom bijele kutije. Što je ujedno i nedostatak s obzirom kako se podrazumijeva poznavanje načina implementacije i dobro znanje programskog koda od strane testera. Funkcija testera je provjera uspješnosti obavljanja zadataka sustava. Prednost testiranja je što programerima omogućava lakše i brže otkrivanje greška ili nepotrebnog koda, te odabir uspješnijih i učinkovitijih metoda implementacije uz pisanje ispravnog koda (Spillner i dr. 2014).

3.2.4. TESTIRANJE POVEZANO S PROMJENAMA I REGRESIJSKO TESTIRANJE

Testiranje povezano s promjenama poistovjećuje se s nadogradnjama, izmjenama i implementacijama novih funkcionalnosti u postojeći sustav, a izvodi se radi sprječavanja grešaka u programskom proizvodu tijekom izmjena. Provodi se nakon ispravljanja grešaka u programskom rješenju kako bi se osiguralo ispravno funkcioniranje sustava nakon odrađenih izmjena, testirajući samo identificirani problem .

Regresijsko testiranje obuhvaća sve funkcionalnosti sustava, ponovno testirajući stare dijelove sustava, kao i novo integrirane kako bi se otklonila mogućnost utjecaja noviteta na ispravnost rada postojećeg sustava prije implementacije novih dijelova. Testiranje vršimo nakon potvrde o uspješnom rješavanju pronađene greške čime se ustvrđuje kako nije došlo do pojave novih grešaka u sustavu (Tutorialspoint, 2021). Provodimo ga na funkcionalnom, nefunkcionalnom i strukturom testiranju, a rezultira lakšim upravljanjem novim verzijama, ublažavanjem rizika i boljom kontrolom nad greškama.

4. METODE TESTIRANJA

„Metoda (grčki *μέθοδος:* put istraživanja, način, postupak), planiran ili unaprijed smišljen postupak za postizanje određenoga teorijskog ili praktičnog cilja.“ (Hrvatska enciklopedija, 2021.)

Ovisno o razvojnoj fazi programskog rješenja razlikuje se nekoliko metoda testiranja koje opisuju gledište i zadaće testera prilikom kreiranja potrebnih testnih slučajeva. Neke od metoda provjeravaju funkcionalnost sustava i njegovu izvedbu dok druge ispituju unutrašnju strukturu i dizajn odvijanja procesa. U većini slučajeva metode testiranja se dijele na:

- Agilno testiranje
- Ad-hoc testiranje
- Testiranje metodom crne kutije (eng. *black box testing*)
- Testiranje metodom bijele kutije (eng. *white box testing*)
- Testiranje metodom sive kutije

4.1. AGILNO TESTIRANJE

lat. *Agilis*: brz, živ, spretan, poduzetan, koji odlikuje voljom za rad i akciju (Jezikoslovac, 2021). Agilno testiranje je metoda testiranja koja slijedi princip agilnog razvoja aplikacije gdje se razvojne i testne aktivnosti odvijaju paralelno (Software testing fundamentals, 2021).

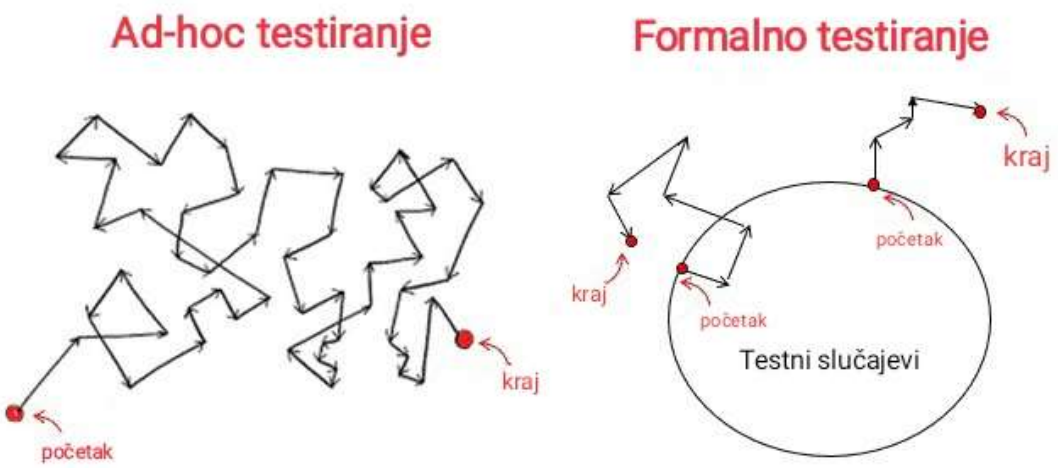
Agilni razvoj programskog rješenja podrazumijeva brzu reakciju i prilagodbu na promjene, automatizaciju testova, korištenje istraživačkih testova i određivanje prioritetnih funkcionalnosti za testiranje. Također kontinuiranu suradnju svih članova razvojnog tima od samog početka rada na projektu pa kroz cijeli životni ciklus projekta, balansirajući razvoj programskog koda i učestalo testiranje. Glavni cilj tima je učestalim razvojem postići visoku kvalitetu, stvoriti prilagodljiv, suradnički, evolucijski i fleksibilan sustav te osigurati ranu isporuku programskog rješenja i time udovoljiti zahtjevima klijenta. Ključna je i bliska suradnja s klijentom s obzirom kako agilna metoda omogućuje češći uvid i isporuku programa, te je stoga moguće učiniti tražene promjene tijekom faza razvoja kroz Iteracije (vremenski period agilnog testiranja, 1-4 tjedana) (Hr.education-wiki, 2021).

Slika 3. Proces agilne metode testiranja sustava

Izvor: Jovanović, 2018.

4.2. AD-HOC TESTIRANJE

Ad-hoc testiranje neslužbeni je način pronalaska grešaka i nedostatak koje se provodi bez plana, projektne dokumentacije i kreiranja testnih slučajeva s ciljem provjere različitih nepredviđenih funkcionalnosti aplikacije. Testira se nasumično, proizvoljnim istraživanjem aplikacije korak po korak, stvarajući nesvakidašnje i neočekivane scenarije uz praćenje ponašanja sustava i određivanja njegove prihvatljivosti. Nedostatak ove vrste testiranja je teško identificiranje i reproduciranje grešaka bez dokumentacije i testnih slučajeva, no prednost leži baš u tom odmicanju od formalnih metoda testiranja. Odmicanje od klasičnog testiranja omogućuje pronalazak grešaka i smanjuje propuste među koracima unutar aplikacije, kasnije nasumično izvedenim od strane korisnika, što se često propušta detektirati (Test Automation Resources, 2018).



Slika 4. Usporedba procesa Ad-Hoc testiranja i provođenja standardnog procesa testiranja

Izvor: Gupta, 2019.

4.3. METODA CRNE KUTIJE

Metoda crne kutije poznata još kao i funkcionalna metoda testiranja ili bihevioralno testiranje je metoda testiranja programskog rješenja bez poznavanja unutarnje strukture, logike i radnji unutar sustava. Fokusira se na eksterni prikaz aplikacije odnosno testiranje funkcionalnosti programa uz poznate ulazne i izlazne parametre. Služi detektiranju neispravnih ili nedostajućih funkcija sustava, pogrešaka u strukturi i ponašanju sustava, problema s bazom podataka i pohranjivanjem. Kao i uočavanju pogreška korisničkog sučelja i odstupanja od specifikacija, kako bi svi funkcionalni zahtjevi koje je korisnik naveo bili zadovoljeni (Software testing fundamentals, 2021).

Metoda crne kutije koristi se prilikom integracijskog testiranja, testiranja sustava i testiranja prihvatljivosti (Test automation resources, 2018). Prednost korištenja ove metode su: pristup kodu i unutarnjoj strukturi nije potreban, jednostavno se otkrivaju greške, lako se izvode testovi, tester ne mora nužno imati vještine programiranja i razumjeti kod, te je pogodna za testiranje velikih dijelova programskog rješenja. Nedostaci su: ograničeno znanje testera što može rezultirati pretpostavljanjem i pogađanjem grešaka, velik broj testnih slučajeva koje je

teško dizajnirati kada specifikacija sustava nije precizna i jasna i velika vjerojatnost ponavljanja testova koju su već izvršeni od strane programera (Myers i dr., 2012).

Slika 5. Metoda testiranja crne kutije

Crna kutija u sredini predstavlja programsko rješenje koje se testira. Testeru nisu poznate informacije o unutarnjoj strukturi već samo ulazni i izlazni parametri.

Izvor: Limbachiya, 2018.

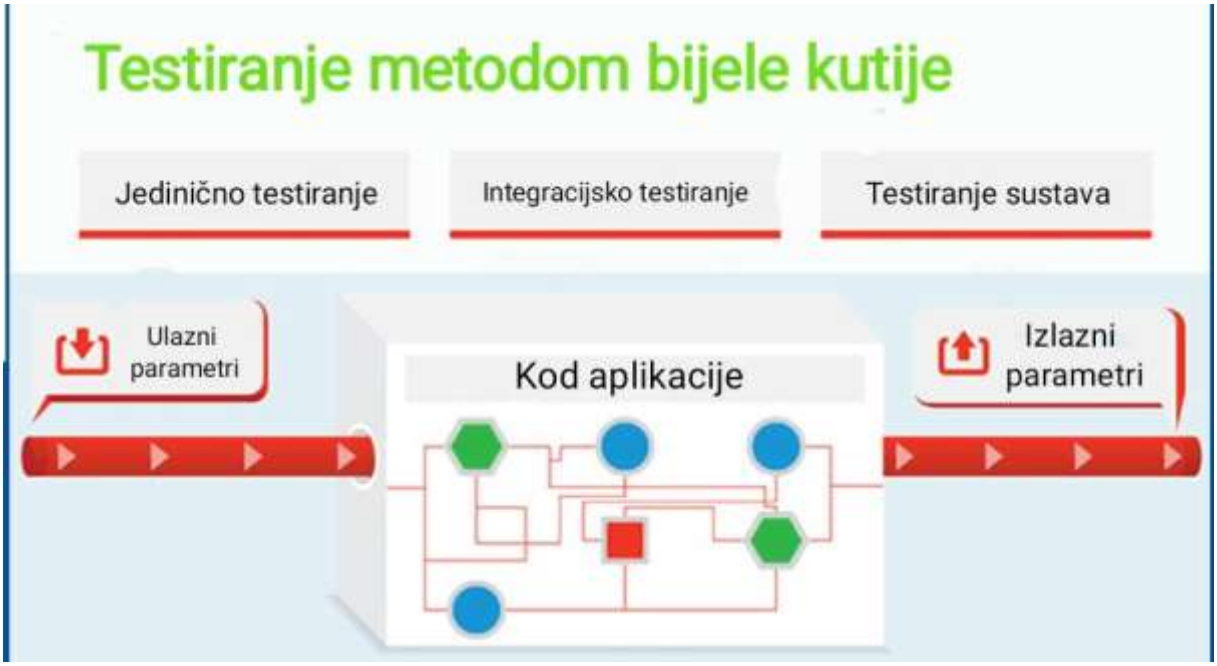
4.4. METODA BIJELE KUTIJE

Testiranje metodom bijele kutije ili strukturno testiranje obuhvaća testiranje cjelovitog koda, interne logike i strukture sustava te dizajna (Hamilton, 2021). Ovu vrstu testiranja najčešće provode programeri upućeni u implementaciju svih komponenti programa, no ukoliko ga provode testeri od iznimne važnosti je posjedovanje informatičkih znanja o programiranju i dobro poznavanje internih procesa. Tester određuje scenarij testnog procesa nakon promatranja protoka podataka i slijeda izvršavanja koda te izabire načine testiranja najvažnijih aspekata sustava (Software testing fundamentals, 2021).

Testiranje metodom bijele kutije temeljit je dugotrajan i iscrpan proces koji povećava upotrebljivost i sigurnost sustava kroz detaljnu analizu proceduralnog dizajna i potanko

testiranje programa (Spillner i dr., 2014). Uz testiranje sustava ovu metodu možemo primijeniti i kod integracijskog i jediničnog testiranja (Software testing fundamentals, 2021).

Prednosti metode su: poznavanje koda od strane testera, lakša optimizacija programskog koda, testiranje rizičnijih dijelova koda s fokusom na dijelove koda sklonije greškama, mogućnost otkrivanja grešaka u ranijim fazama razvoja te lakše definiranje testnih podataka. Dok nedostatke čine: veliki troškovi testiranja, nedostupnost alata za provjeru implementacije na različitim platformama, nemogućnost ispitivanja svih dijelova programskog rješenja i teško održavanje ukoliko često dolazi do izmjena koda (Myers i dr. 2012).



Slika 6. Model metode bijele kutije

ilustracija sustava, funkcioniranja, procesa i podognih testiranja za korištenje metode

Izvor: Singh Gill, 2018.

4.5. METODA SIVE KUTIJE

Metoda sive kutije poznata još kao prozirno testiranje kombinacija je spoja metoda testiranja crne i bijele kutije koja pretražuje greške nastale nepravilnim funkcioniranjem programa i lošom strukturom koda. Testiranje provode osobe različitog znanja i vještina, djelomično upoznate s unutarnjom strukturom i procesom rada aplikacije, najčešće s korisničkog stajališta. Testerima je dostupan pristup projektnoj dokumentaciji, bazi podataka, internoj strukturi i algoritmima prilikom kreiranja testnih podataka i scenarija (Acharya, Pandya, 2012).

Prioritet prilikom provođenja testiranja je smanjiti dugotrajan proces testiranja i omogućiti programerima dovoljno vremena za ispravljanje grešaka, a najčešće se koristi za testiranje aplikacija na mreži te kod integracijske razine testiranja (Software testing fundamentals, 2021).

Pozitivne strane korištenja metode sive kutije su: ne oslanjanje na kod već na definiciju sučelja i funkcionalne zahtjeve, omogućavanje kvalitetnog dizajniranja testnih slučajeva s obzirom kako su testeri upoznati s podacima u programskom rješenju i maksimalno iskorištavanje prednosti testiranja crne i bijele kutije služeći se testiranjem s gledišta korisnika, a ne programera ili dizajnera. No ograničena mogućnost proučavanja logike programskog koda, nepoznavanje kompletne strukture i logike proizvoda uz nepokrivanje i ponavljanje testova neke su od negativnih strana korištenja ove metode (Acharya, Pandya, 2012).

Tablica 2. Usporedba metoda crne, bijele i sive kutije

Testiranje metodom crne kutije	*Testiranje metodom bijele kutije*	*Testiranje metodom sive kutije*
Poznavanje interne radne strukture (kôda) nije potrebno za ovu vrstu ispitivanja. Za testne slučajeve potreban je samo GUI (grafičko korisničko sučelje).	Za ovu vrstu ispitivanja nužno je poznavanje interne radne strukture (kôda).	Djelomično je potrebno poznavanje interne radne strukture.
Manje je zahtjevna od metoda bijele i sive kutije.	Najzahtjevnija je metoda testiranja.	Djelomično zahtjevna; ovisi o vrsti test slučajeva koji se temelje na kodiranju ili GUIu.
Osnova ovog ispitivanja su vanjska očekivanja, interno ponašanje nije poznato.	Osnova ovog testiranja je kodiranje odgovorno za unutarnji rad proizvoda.	Ispitivanje na temelju dijagrama baza podataka na visokoj razini i dijagrama protoka podataka.

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

Izvor : JavaTpoint, 2021.

5. PROCES I FAZE TESTIRANJA

Proces i faze kroz koje program prolazi prilikom kreiranja ovise o modelu razvoja sustava čime se određuje i način testiranja. Testiranje može biti jedna od faza razvoja programskog rješenja (vodopadni model) ili se provoditi nakon svake faze razvoja (V-model) (Shiklo, 2019).

Ključno je osigurati pouzdanost i ispravnost programskog rješenja što se postiže kvalitetnim planom provedbe koji uključuje planiranje testova i vrijeme potrebno za provođenje, dizajniranje testnih scenarija, njihovu provedbu i krajnju evaluaciju (Graham i dr., 2008).

S obzirom na sličnosti uočene prilikom provođenja testnih procesa primjenom različitih pristupa, prema *International Software Testing Qualifications Bordu* faze testiranja svedene su u 5 osnovnih univerzalnih koraka:

- Planiranje i kontrola testiranja
- Analiza, dizajn i projektiranje
- Implementacija, provedba i izvršenje testiranja
- Vrednovanje izlaznih kriterija i izvješćivanje
- Aktivnosti zatvaranja testova

5.1. PLANIRANJE I KONTROLA

Planiranje i kontrola prva su faza testiranja u procesu razvoja programa.

Planiranjem se definiraju pravila i željeni pristup procesu testiranja, odnosno politika i strategija testiranja kojom će se voditi kroz cjelokupni proces. Kako bi se formirala jasna misija i plan testiranja ključno je razumijevanje potreba klijenta i cilja projekta. Planiranjem se kreira specifikacija aktivnosti testiranja, pristup plan i raspored testova prioritetnih funkcionalnosti. Osigurava razumijevanje ciljeva i rizika testiranja, uz određivanje potrebnih resursa i vremena za izvršenje (Graham i dr., 2008).

Kontrola omogućuje uvid u trenutno stanje procesa testiranja konstantnom usporedbom planiranog napretka sa stvarnim napretkom. Izvještavanje o statusu, promjenama i odstupanjima od plana omogućava pravovremenu reakciju ukoliko se javi potreba za poduzimanjem određene akcije, kako bi u konačnici projektni ciljevi bili zadovoljeni (Spillner i dr., 2014). Glavni zadaci kontrole testiranja su analiza rezultata testiranja, dokumentacija i praćenje izlaznih kriterija, napretka i pokrivenosti testova (Graham i dr., 2008).

5.2. ANALIZA I PROJEKTIRANJE

Fazu analize provodimo prije kreiranja testnih slučajeva, pregledavanjem planiranog testnog plana, kako bi provjerili strukturu sustava, funkcionalne zahtjeve, dizajn i analizirali rizik prije kreiranja uvjeta za projektiranje i provođenje testova (Spillner i dr., 2014). Analizom i projektiranjem: ispituju se specifikacije sustava provjeravajući temelj za testiranje, identificiraju ključni uvjeti u procesu testiranja, osmišljavaju reprezentativni testovi za pojedine aspekte programskog rješenja uz provjeravanje isplativosti testiranja, te se određuju potrebni alati i infrastruktura uz dizajniranje testnog okruženja (Graham i dr., 2008).

5.3. IMPLEMENTACIJA I IZVRŠENJE

Implementacija i izvršenje proces su provjere kreiranih testova u testnom okruženju (Spillner i dr., 2014).

Zadaća implementacije je raspisati uvjete za provođenje testova, odnosno pomoću izrade testnih slučajeva plasirati ih u testno okruženje. Uspješno izvršavanje testova pospješuje se kreiranjem zbirke testnih slučajeva temeljem koje se uspješno izrađuju skripte za automatizaciju testiranja. Nakon kreiranja testnih podataka i uputa za provođenje, testovi se implementiraju i ispituju (Graham, 2008).

Izvršavanje testova provodi se prema planiranom rasporedu provođenja, najprije provjeravajući prioritetne funkcionalnosti koje mogu imati značajan utjecaj na rad programa. Testovi se izvršavaju ručno, najčešće kod pojedinačnih testnih slučajeva ili automatizirano kroz testne pakete i odgovarajuće alate. Dokumentacija cjelokupnog procesa i rezultata pravilno izvršenih testova je od ključne važnosti. Omogućuje nam uvid u detektirane greške, uspješno ili neuspješno izvršene testove, korištene alate i verzije programskog rješenja (Spillner i dr., 2014). Uzastopno testiranje popraćeno novim verzijama programa kroz cjelokupni proces testiranja ubrzo bi rezultiralo kaosom stoga je dokumentiranje od iznimne važnosti.

5.4. EVALUACIJA I IZVJEŠĆIVANJE

Evaluacija je proces ocjenjivanja i utvrđivanja stupnja zadovoljenja izvršenih testova. Rezultate testova uspoređuje se s ciljevima definiranim prije početka testiranja. Prilikom ispunjavanja svih kriterija piše se završno izvješće o provedenom testiranju. Pri evaluaciji se vrednuju izlazni

kriteriji individualno kreirani za svaki projekt te se na temelju njih uspostavlja zadovoljavajući kriterij testiranja temeljen na procesu procjene rizika (Graham i dr., 2008).

Svaki detektirani problem potrebno je ispraviti uz ponavljanje testiranja, ukoliko testiranje ne ispunjava zadani kriterij potrebno je provesti dodatne testove. Također u nekim slučajevima originalni kriteriji mogu biti manjkavi i neprovedivi te se tada uvode novi dodatni kriteriji. Nakon svih provedenih testova i maksimalnog ispunjavanja završih kriterija formira se za izvještaj testiranja s obavljenim testiranjima i njihovim ishodima (Spillner i dr., 2014).

5.5. AKTIVNOSTI ZATVARANJA TESTOVA

Zatvaranje testova konačna je faza u testiranju programskog rješenja. Zahtjeva provođenje analize cjelokupnog procesa testiranja čime je cilj obuhvatiti i objediniti kompletne rezultate testiranja uključujući analizu brojeva i činjenica, provjeru i popunjavanje testova te usporedbu planiranih sa stvarnim vrijednostima testiranja. Dokumentira se prihvaćanje ili odbijanje programskog rješenja i sveobuhvatna procjena provedenog testiranja, finalizira i arhivira testna infrastruktura te vrši provjera izvješća otklanjanja grešaka s ciljem pospješivanja uspješnosti i efikasnosti budućih procesa testiranja. Zatvaranjem projektnog testiranja programsko rješenje se predaje organizaciji zaduženoj za održavanje (Graham i dr., 2008).

6. AUTOMATSKO TESTIRANJE

Automatsko testiranje možemo opisati kao ubrzani proces provjere i kontrole aplikacije korištenjem automatiziranih testnih skripti, raznih alata za automatizaciju testiranja i programskih rješenja koji testiraju kontrolu izvršavanja testova, uspoređujući stvarne s predviđenim ishodima (Ammann, Offutt, 2017).

Automatizacija omogućava učinkovitiju provedbu testiranja s obzirom kako se jednom kreirani testni slučajevi mogu pohraniti i ponovno pokretati prema potrebi čime se povećava brzina izvršavanja testova i pokrivenosti programskog rješenja testom (Hamilton, 2021). Iako se automatizacijom testiranja testovi obavljaju brže, efikasnije i jednostavnije ono iziskuje više utrošenog vremena, potrebnih ljudskih resursa i veće troškove prilikom automatiziranja testnog paketa. Kao i nabavljanje programa i alata za automatizaciju, ispravljanje grešaka u testnoj skripti i održavanje testova (Ammann, Offutt, 2017).

Automatizirani alati omogućuju ispravno pohranjivanje podataka testiranja u bazi čime izbjegavamo greške uzrokovane ljudskim faktorom. Testni slučajevi pogodni za

automatiziranje su ponavljajući, teško izvodivi i visoko rizični testovi čijom se automatizacijom značajno olakšava proces testiranja smanjujući broj ručno pokrenutih testnih slučajeva (Smartbear, 2021). Također automatizacija je poželjna i kod ponavljajućih regresijskih testova prilikom dodavanja novih funkcionalnosti u sustav kao i kod testiranja performansi i opterećenja sustava (Fernandes, Di Fonzo, 2017).

6.1. SVRHA AUTOMATSKOG TESTIRANJA

Temeljna svrha automatskog testiranja leži u unapređenju, pospješivanju i ubrzavanju procesa testiranja programskog rješenja (Myers i dr., 2012). Omogućuje češće pokretanje i provedbu postojećih testova: pod različitom konfiguracijom hardvera, na različitim operacijskim sustavima i bazama podataka, na novim programskim rješenjima i različitim verzijama programa uz skraćivanje vremena potrebnog za njihovo izvršavanje. Češćim pokretanjem procesa testiranja povećava se ispravnost i sigurnost sustava, a ukoliko su testovi automatizirani može ih se pokrenuti u određeno željeno vrijeme s obzirom na samostalnu aktivaciju procesa testiranja.

Automatsko testiranje iznimno je korisno kod velikih sustava s povećim brojem korisnika ili višejezičnih sustava gdje ručno testiranje nailazi na problem provođenja i nemogućnost osiguravanja ispravnog rada sustava. Iako je ova vrsta testiranja u samom početku skupa, kasnije rezultira uštedom vremena i resursa s obzirom da se automatiziranjem jednostavnijih i ponavljajućih zadataka smanjuje potreba ručnog testiranja. Rad koji je uložen na odlučivanje o vrsti testiranja, dizajniranje testova i njihovu izradu te osiguravanje pouzdanosti isplativ je količinom višestrukog ponavljanja testova, a time se postiže konzistentnost kod procesa testiranja (Fewster, Graham, 1994).

6.2. PROBLEMI I OGRANIČENOST AUTOMATSKOG TESTIRANJA

Uz brojne prednosti automatizacije postoje i njena ograničenja te problemi koji dolaze s time. Organizacije često nailaze na nesigurnosti prilikom odlučivanja za uvođenjem automatskih testova jer iziskuje velika ulaganja i edukaciju djelatnika, a ne garantira uspjehom. Automatsko testiranje nikada u potpunosti neće zamijeniti ručno testiranje, niti je osmišljeno s tom namjerom. Služi olakšavanju, a ne rješavanju svih tegoba programskog razvoja i provjere programskog rješenja, što često rezultira nerealnim očekivanjima od procesa automatizacije.

Kvalitetno odrađen proces evaluacije i izvršenja testiranja ključan je kod automatizacije, u suprotnom će testovi biti loše kvalitete i neće donositi željene rezultate, a njihovo ispravljanje zahtjeva puno vremena, resursa i novaca. Automatske testove bitno je održavati i unaprjeđivati u skladu s promjenama sustava kako ne bi postali neefikasni, a alati za testiranje neiskoristivi. Testiranjem automatskim testovima detektira se mali broj grešaka stoga je ručno testiranje funkcionalnosti prije automatizacije neophodno. Ispravno izvršeni automatski testovi nisu stopostotna sigurnost potpune ispravnosti sustava, već se ono također ispituje ručnom metodom testiranja(Fewster, Graham, 1994).

Prilikom donošenja odluke o primjeni automatiziranih testova valja dobro odvagnuti sve prednosti i nedostatke koje nam takav proces testiranja donosi te odlučiti što je u skladu s interesima i mogućnostima projektnog tima i organizacije.

6.3. ALATI ZA AUTOMATSKO TESTIRANJE

Ključ uspješnosti automatizacije testnih slučajeva leži u ispravnom identificiranju i kvalitetnom odabiru potrebnih alata za automatizaciju koji su u skladu s potrebama testiranja programa. Ukoliko je taj kriterij zadovoljen automatizacija testova i mogućnost njihovog ponovnog korištenja rezultirati će unapređenjem učinkovitosti i djelotvornosti programskih rješenja (Anderson, 2021). Potrebno je detaljno i temeljito razumijevanje specifičnosti programskog rješenja uz razmatranje svih dostupnih mogućih opcija alata testiranja koji su u skladu s potrebama i kriterijima adekvatnim za projekt, kako bi se odabrao onaj najprikladniji za automatizaciju testova projekta.

Postoji čitav niz različitih alata s raznolikim značajkama, funkcijama i mogućnostima. Osvrćući se samo na automatsko testiranje u programskom okruženju nailazimo na: alate za testiranje aplikacija na mreži, alate za regresijsko/funkcionalno testiranje, testiranje GUI-a na mreži, alate za ispitivanje performansi, naprezanja i opterećenja, alate za testiranje upravitelja veza, za provjeru sigurnosti stranica, za testiranje u više preglednika i brojne druge (Myservername.com, 2021).

U globalu ih možemo svrstati u prilagođene, komercijalne i alate otvorenog koda (eng. *Open source*). Prilagođene alate koristimo isključivo kod specifičnih projekata koji zahtijevaju individualniji pristup, dok su alati otvorenog tipa kreirani za sve faze u procesu razvoja programa i svima javno dostupni na korištenje (Anderson, 2021).

6.3.1. SELENIUM

Selenium je jedan od najpopularnijih besplatnih *open source* paketa alata za automatsko testiranje u mrežnom okruženju namijenjen testiranju funkcionalnosti. Pruža fleksibilnost testiranja aplikacija na mreži putem različitih preglednika i drugačijih platforma. Velikim brojem testnih funkcionalnosti omogućava zadovoljavanje različitih potreba i pristupa automatizaciji programskih rješenja (Selenium, 2021).

Selenium se grana na nekoliko različitih skupina alata za testiranje. Selenium IDE (*Integrated Development Environment*) najlakši je i najjednostavniji za upotrebu i učenje. Omogućuje pohranu obavljenih akcija testiranja i njihovo ponovno pokretanje te se upravo zbog toga najčešće koristi kao prototip za izgradnju testnih skripti, dodatak je za Firefox i Google Chrome preglednike. Selenium RC (*Remote Control*) podržava različite programske jezike: C#, Rubin, Java Script, Perl, Phyton i PHP. Prije pojave Selenium WebDrivera aktivno je održavan čineći vodeći testni paket cjelokupnog Seleniuma. Selenium WebDriver je najnoviji dodatak koji primjenjuje moderniji pristup uz pružanje brojnih novih mogućnosti automatizacije i omogućava stabilnu komunikaciju između testnih skripti i preglednika. Podržava tehnologiju i programske jezike koji rade na različitim preglednicima (Firefox, Chrome) i sistemskim okruženjima (Linux, Windows) kao Selenium RC. Kojim se uz korištenje Selenium Grida omogućava paralelno izvođenje testova u različitim okolinama i povećavanje performansi velikih skupova testova. Glavni nedostatak Seleniuma je neophodna edukacija ljudi kako bi se alate učinkovito koristilo i veliki utrošak vremena na razvoj testnih skripti potrebnih za automatizaciju (Hamilton, 2021).

6.3.2. APPIUM

Appium je jednostavan *open source* alat temeljen na arhitekturi klijent/poslužitelj. Appiumom se mogu automatizirati: aplikacije napisane pomoću SKD – paketa za razvoj programa (eng. *Software development kit*), aplikacije na mreži kojima se pristupa putem preglednika na mobilnom telefonu i hibridne aplikacije koje 'imaju omot' oko internet prikaza, odnosno oko izvorne kontrole koja omogućuje interakciju s web sadržajem. Podržava Safari za iOS, Google Chrome i ugrađene aplikacije preglednika na Androidu. Appiumom je moguće pisanje testova na više operacijskih sustava (Windows desktop,iOS, Andriod) koristeći isto aplikacijsko programsko sučelje – API, čime omogućuje ponovnu upotrebu testnog koda između testnih programa (Appium, 2021).

6.3.3. RANOREX

Ranorex je moćan, intenzivan i iznimno učinkovit alat za automatizaciju testiranja vizualnih elemenata aplikacije (GUI–a). Kvalitetom parira Seleniumu te nerijetko biva češće prakticiran. Razlog tome je jednostavnost primjene i korištenja od strane organizacija jer omogućava jednostavno svladavanje i ne zahtjeva stručnu edukaciju testera i programera. Ranorex funkcionira na način da bilježi korisnikovu aktivnost i radnje na programskom rješenju, nakon čega ih pohranjuje stvarajući mogućnost ponovne reprodukcije i kreiranja testnih skripti, te u konačnici automatiziranje procesa. Koristi se kao alat za automatizaciju testiranja grafičkog korisničkog sučelja kod *desktop*, mobilnih i *web* aplikacija s sposobnosti paralelnog izvođenja testova među različitim preglednicima (Ranorex GmbH, 2021).

6.3.4. KATALON

Katalon je također jedan od poznatijih i značajnijih besplatnih alata za automatizaciju testiranja. Izgrađen je na Selenium i Appium okvirima za automatizaciju, a koristan je alat prilikom procesa implementacije i integracije testnih knjižica i okvira (Anderson, 2021).

Služi stvaranju i reproduciranju automatiziranih testnih skripti korisničkog sučelja. Omogućava automatizaciju testiranja *web*, mobilnih (Android i iOS) i *desktop* aplikacija kao i sučelja za programiranje aplikacija API-a. Možemo ga pokrenuti putem Google Chrome, Mozilla Firefox i Microsoft Edge preglednika, kao i na Linux, Microsoft Windows i macOS sustavima, no jedino putem Groovy programskog jezika. Za razliku od Seleniuma koji podržava mnoštvo skriptnih jezika ima uži skup integracije i kompliciraniji proces implementacije. Glavna prednost Katalona je fleksibilnost prilikom zadovoljavanja raznolikih potreba testera i projekata. Pruža mogućnost složenijeg pristupa skriptiranju uz isticanje sintakse, prijedloga koda i uklanjanje pogrešaka kao i jednostavnijeg pristupa bez potrebe za pisanjem koda (Altexsoft, 2021).

6.3.5. SOAPUI

Prema SoapUI: „SoapUI je najrašireniji alat za funkcionalno testiranje aplikacijskog programskog sučelja (eng. *application program interface – API*) na svijetu“.

SoapUI je alat za funkcionalno testiranje otvorenog koda: uslužno orijentirane arhitekture - SOAP (eng. *simple object access protocol*), web servisa i prijenosa reprezentativnog stanja - REST (eng. *representational state transfer*), aplikacijskog programskog sučelja, pokrivenosti opisnog jezika *web* servisa (eng. *web services description language*) i izmjenu testova (SoapUI, 2021).

Testovi se pišu putem Java Script programskog jezika, a pospješuje jednostavnost i brzinu izvođenja regresijskih, funkcionalnih testova i testova opterećenja sustava (Myservername.com, 2021).

6.3.6. WEB-DRIVER-IO

WebdriverIO prilagođena je implementacija Seleniumovog WebDriver API-a (eng. *application programming interface*). WebdriverIO predstavlja uslužni *open source* alat namijenjen kreiranju automatiziranih testova za *web* i mobilnu automatizaciju, napisan u JavaScriptu dok testove pokreće putem Node.js-a. Automatsko testiranje web aspekata omogućava se slanjem i obrađivanjem naredbi i zahtjeva pogodnih automatizaciji na Selenium putem Webdriver protokola. Putem WebdriverOI alata preglednikom ili mobilnom aplikacijom upravlja se jednostavnim i sažetim kodom, dok se integracijom *test runner-a* (aplikacija koja pokreće izvršavanje integriranih funkcionalnih testova i dopušta isporuku rezultata) omogućava pisanje naredbi i kreiranje zahtjeva u različitom vremenskom i prostornom okruženju. Ovakav pristup iznimno olakšava rad s funkcionalnim elementima na *web* mjestu s obzirom da se za njihovo dohvaćanje ili ponavljanje upotrebljavaju izvorne JavaScript funkcije. Osim u svrhu testiranja ovaj alat koristi se i za dinamično praćenje i spremanje podataka procesa testiranja, te njihovu integraciju s alatima kao što je Appium uz ponovno iskorištavanje (Webdriver IO, 2021).

6.3.7. UTF ONE

UTF One nova je proširena inačica UTF alata za automatizaciju, proširen je dodatnim mogućnostima za testiranje API-a i podržava više platformi za AUT (eng.*Aplication Under Test*). Pruža prikladan izbor za testiranje aplikacija dostupnih na osobnim računalima, *webu* i mobilnim uređajima. Popularan je alat za testiranje *web*, *desktop*, mobilnih i RPA aplikacija (eng. *Robotic Process Automation*). UFT One pruža nekoliko naprednih mogućnosti za otkrivanje pametnih objekata i detekciju te ispravljanje objekata temeljenih na slici. U studenom 2020. Microsoft je objavio najnoviju verziju 'UFT (v15.0.2)' koja nudi nove značajke i poboljšanja, pojednostavljuje proces testiranja, poboljšava učinkovitost i održava kvalitetu testiranja uz bolju vremensku efikasnost (Anderson, 2021).

Postoji još čitav niz alata koji su od velike važnosti i koristi prilikom automatizacije procesa testiranja kao što su: QA wolf, Cucumber, Testlink, Sikulil, Robotium, Apache Jmater, Capybara i brojni drugi.

7. PRIMJER TESTIRANJA

Primjerom je prikazan jednostavan praktičan primjer testiranja poznate AliExpress stranice za kupovinu na mreži. Provjerava se dohvaćanje stranice putem naznačenog linka, pronazak elementa stranice pomoću lokatora (Id) i upisivanje traženog sadržaja u polje lociranog elementa. Nastoji se objasniti i prikazati kompletan proces od instalacije potrebnih alata i njihove implementacije do izrade testnog slučaja i provjere uspješnosti testiranja. Za pisanje koda korišten je Visual studio code 2019 community, test je pisan u #C programskom jeziku. Za pokretanje i razvoj testova upotrebljavao se programski okvir .NET Core i Selenium Webdriver alat za automatizaciju testova. Test se pokreće putem Google Chrome preglednika.

Visual Studio integrirano razvojno okruženje služi za izradu i uređivanje koda, koristi se za različite vrste razvoja programa. Nudi mogućnost razvijanja stranica, aplikacija i usluga na mreži. Sadrži niz alata za olakšavanje procesa razvoja programskih rješenja (Incredibuild, 2021).

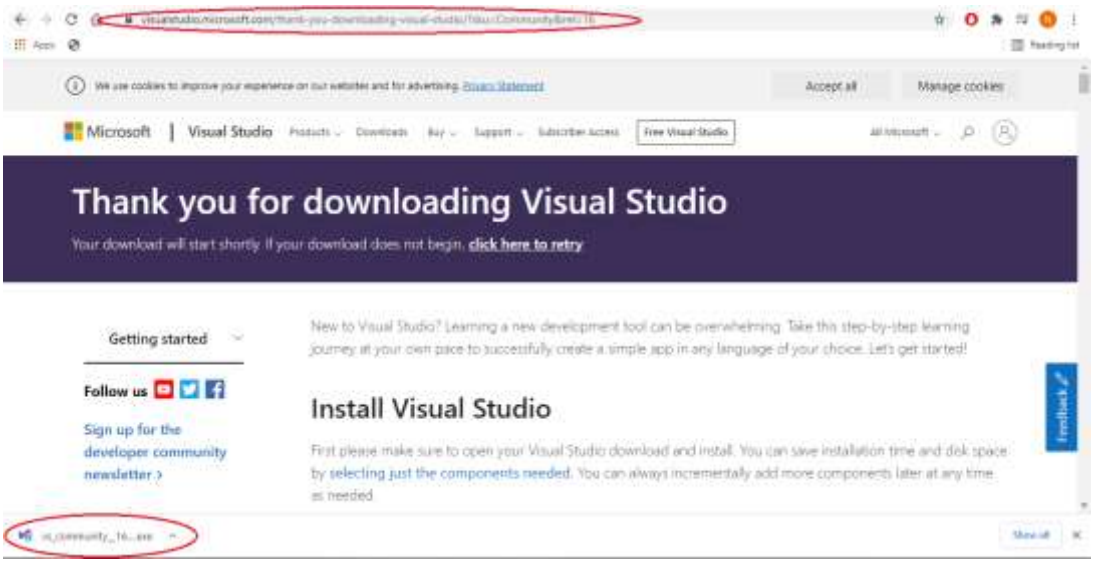
.NET Core je besplatna zajedničkih biblioteka kodova i aplikacijskih programskih sučelja kojima programeri mogu pristupiti i koristiti ih prilikom razvoja programskog rješenja, fokusirana na razvoj aplikacija na mreži (TutorialsTeacher, 2020).

7.1. INSTALACIJA ALATA

U ovom poglavlju detaljno je prikazana instalacija i implementacija alata potrebnih za provođenje procesa testiranja.

7.1.1. VISUAL STUDIO 2019 COMMUNITY I .NET CORE PROGRAMSKI OKVIR

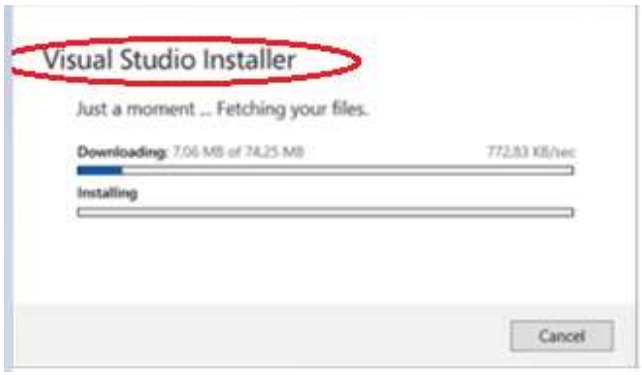
Preuzimanje Microsoft Visual Studio Community 2019 moguće je putem službene Microsoft internetske stranice.



Slika 7. Preuzimanje Visual Studio 2019 Community

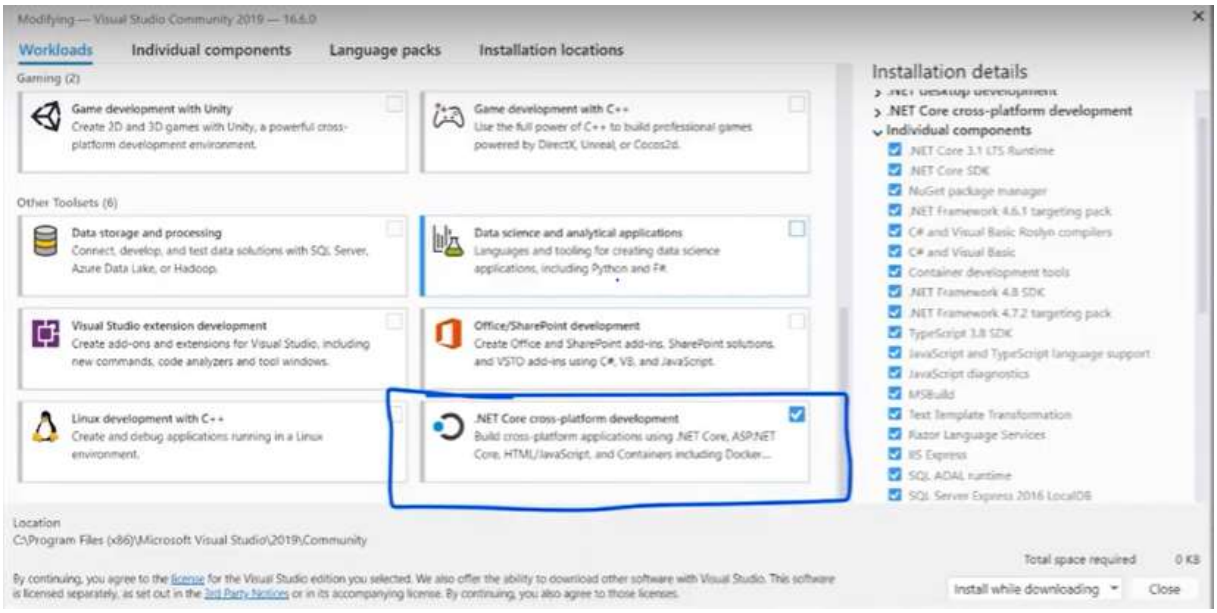
Link stranice preuzimanja i preuzimanje alata

Klikom na instalaciju prvobitno se preuzima Visual Studio Installer kojim će se, nakon odabira potrebnih radnih opterećenja (eng. *Workloads*), Visual Studio 2019 Community uspješno preuzeti na računalo.



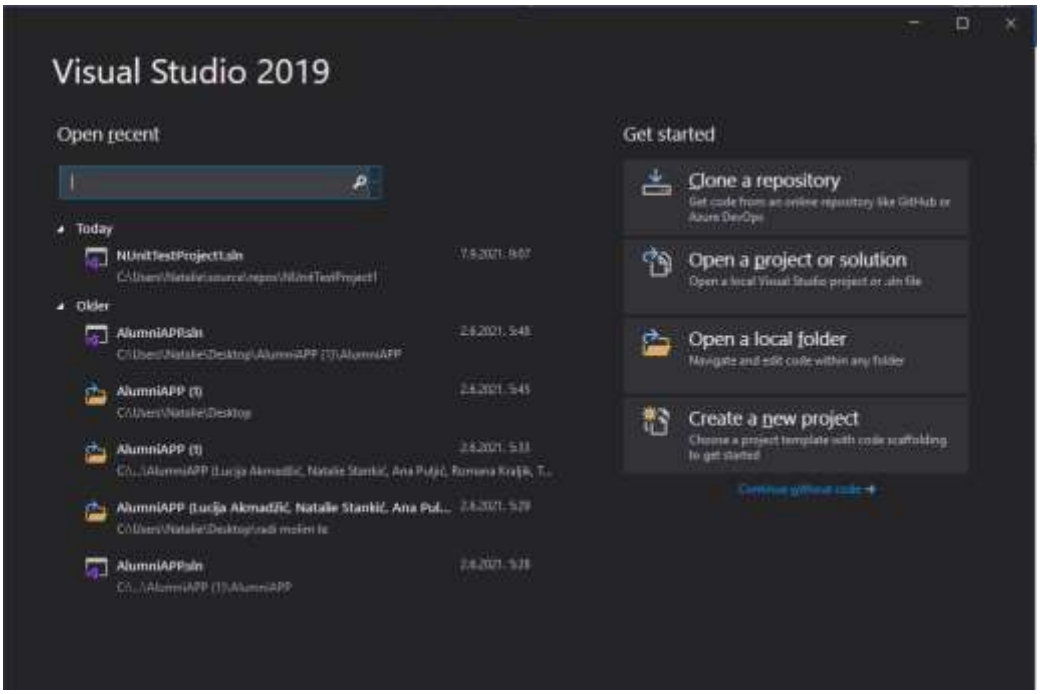
Slika 8. Visual Studio Installer

Ukoliko je Microsoft Visaul Studio Community 2019 već prethodno instaliran sa svim željenim i potrebnim značajkama. Kako bi uspješno proveli automatizaciju testova koristeći .NET Core Framework, potrebano je unutar Visaul Studio Installera provjeriti dali je set alata ".NET Core cross-platform development" preuzet. U suprotnom potrebno je dodatno naknadno preuzimanje.



Slika 9. Visaul Studio Community 2019 - Izmjene

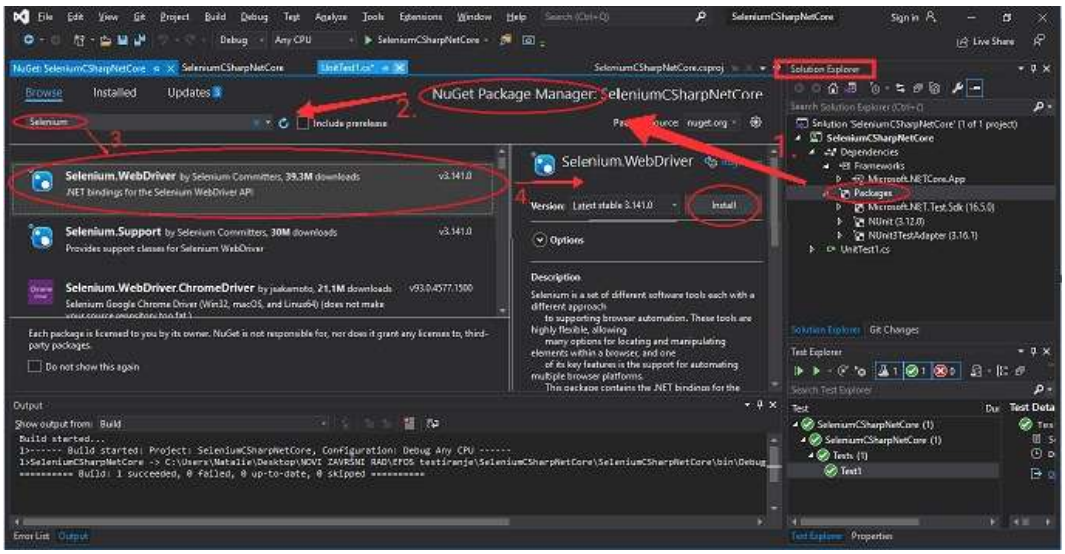
Nakon instalacije svih potrebnih komponenti, Visual Studio 2019 spreman je za korištenje.



Slika 10. Visaul Studio 2019

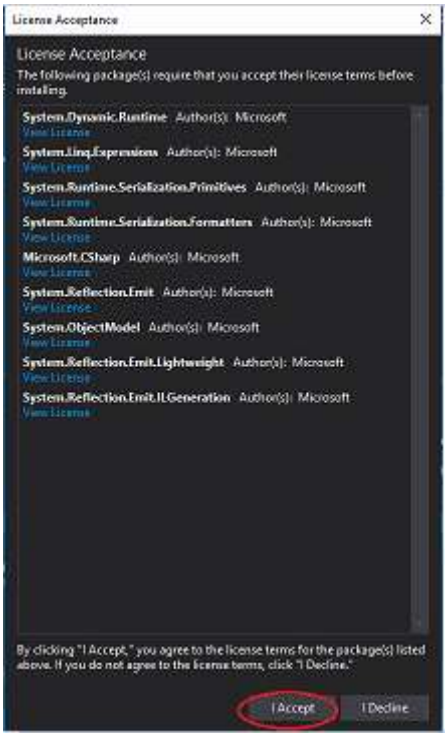
7.1.2. SELENIUM WEBDRIVER

Instalacija Selenium WebDriver biblioteke: U prozoru *'Solution Explorer*' pronalazi se '*Packages*', lijevim klikom na '*Packages*' odabire se *'Manage NuGet packages'*. Nakon otvaranja NuGet prozora, za lakše snalaženje u tražilicu se upisuje naziv željene biblioteke, u ovom slučaju Selenium.WebDriver. Potrebno je provjeriti dali je verzija kompatibilna s ostatkom sustava, ukoliko je, kliknuti *'Install*'.



Slika 11. Selenium.WebDriver

Za završetak instalacije potrebno je potvrditi licence klikom na *'I Accept'*.



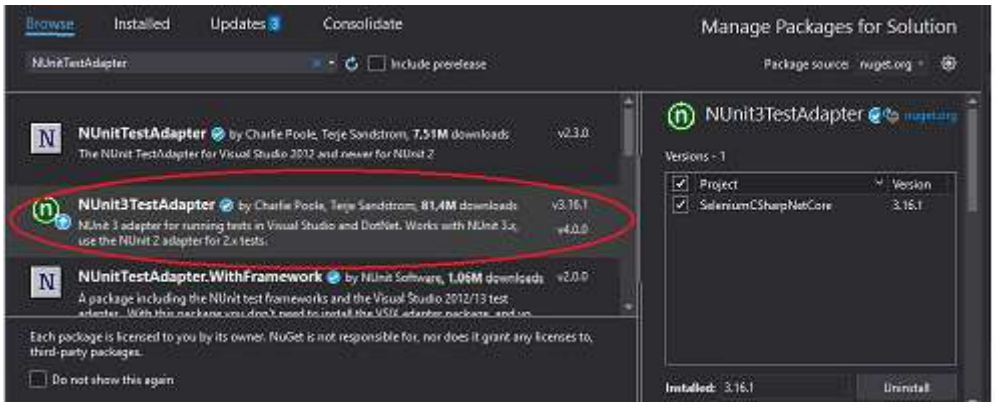
Slika 12. Selenium - potvrda licence.

7.1.3. ADAPTER ZA TESTIRANJE

NUnit test adapter potrebno je instalirati za mogućnost pokretanja NUnit testova unutar Visual Studia.

Popstupak instalacije:

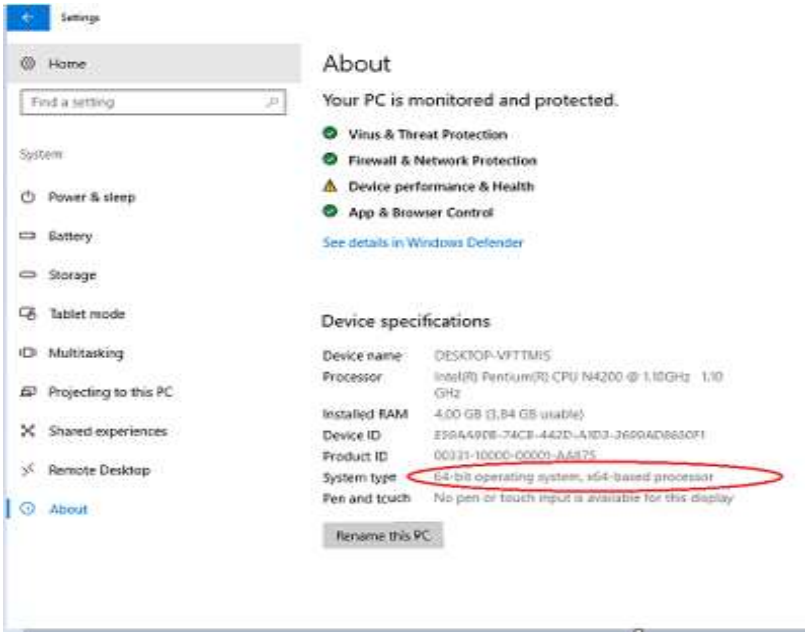
Tools > NuGet Package Manager > Manage Packages for Solution > NUnit Test Adapter
Odabire se projekt na kojem se želi koristiti, zatim klik na '*Install'*.



Slika 13. NUnit adapter za testiranje

Nunit3TestAdapter koristi se bez potrebe za ručnim uključivanjem u proces testiranja kao što je to bilo potrebno kod prethodnih verzija, pogodan je za instalaciju i korištenje na 64-bitnim operacijskim sustavima.

Operacijski sustav računala provjerava se: *Settings* > *System* > *About* > *System Type.*



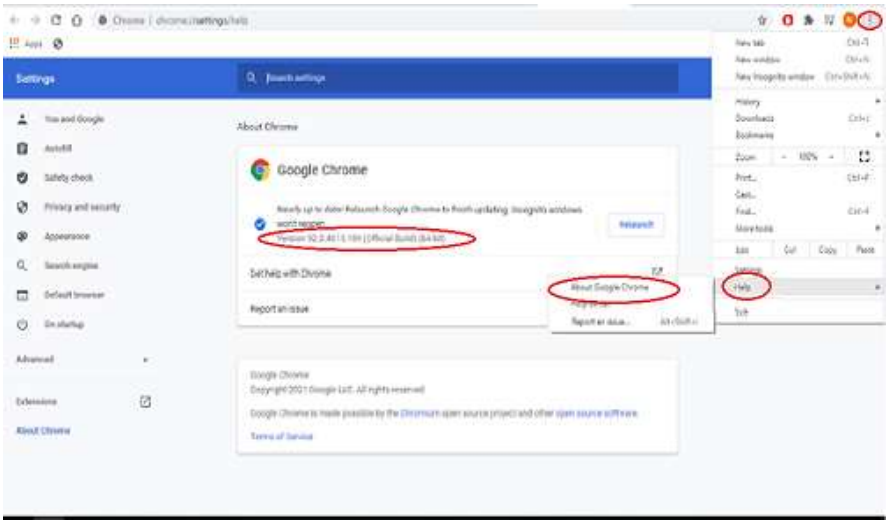
Slika 14. Provjera operacijskog sustava računala

7.1.4. CHROME WEBDRIVER

Za uspješno pokretanje testova putem Google Chrome preglednika potrebno je preuzeti i impementirati Chrome WebDriver, prema idućim koracima:

1. Provjera verzije Google Chrome preglednika aktivne na računalu.

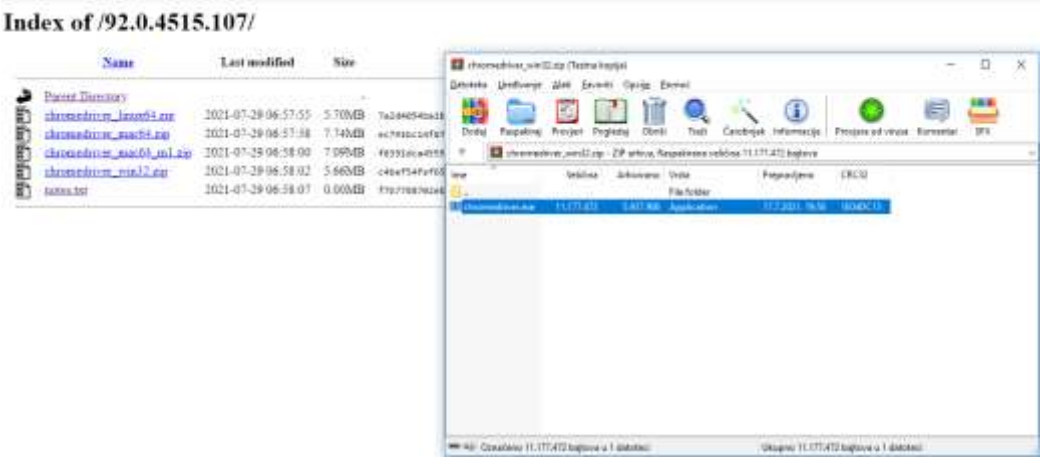
   Klikom na 3 točkice u gornjem desnom kutu Google Chrome preglednika otvara se skočni izbornik. U izborniku izabiremo *Help* > *About Google Chrome*.



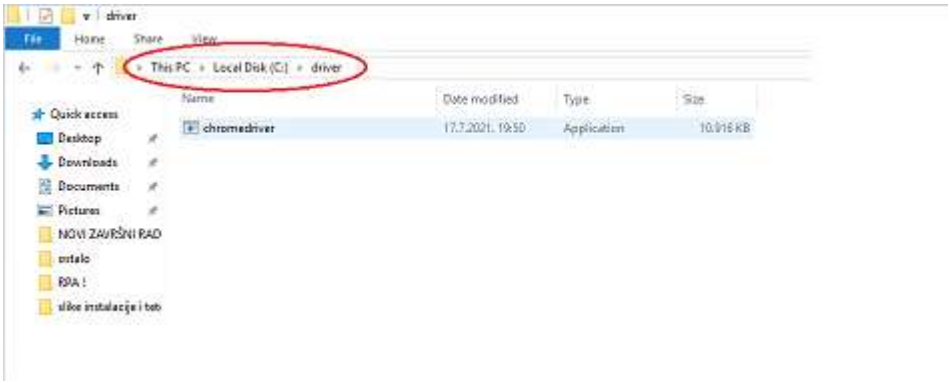
Slika 15. Provjera verzije Google Chrome preglednika

2. Preuzimanje kompatibilne verzije ChromeDrivera s verzijom preglednika.

   Nakon preuzimanja zip datoteke koja odgovara operativnom sustavu, istu je potrebno raspakirati na željeno mjesto na računalu za dohvaćanje izvršne datoteke "chromedriver.exe".



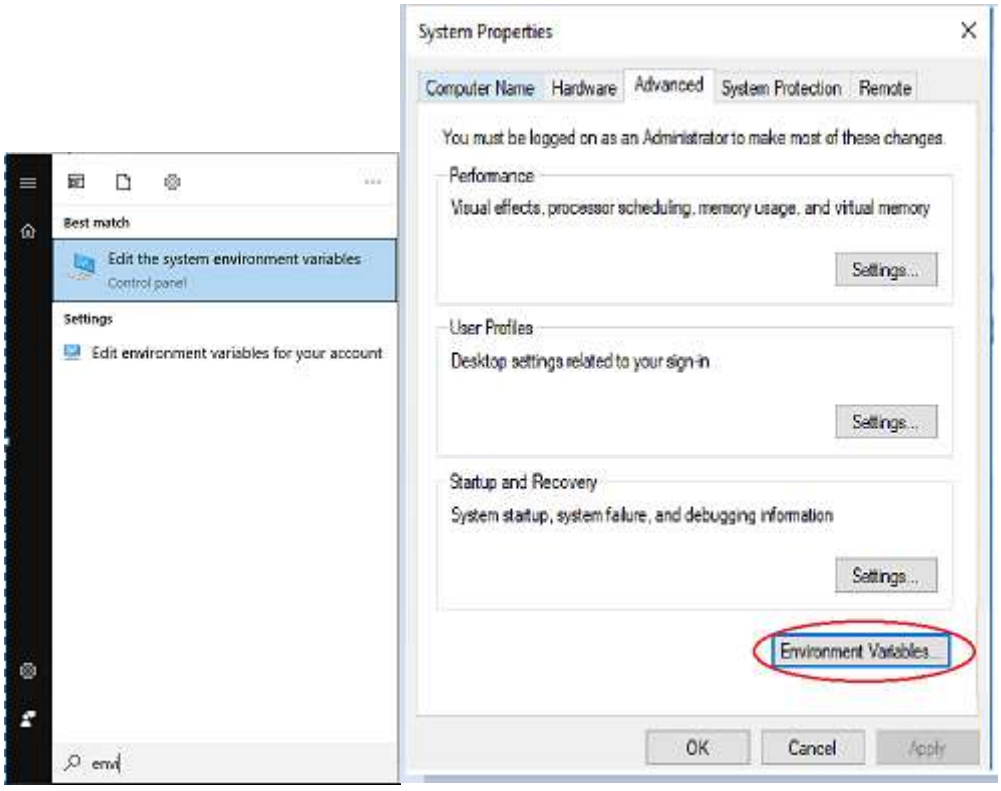
Slika 16. ChromeDriver



Slika 17. chromedriver.exe - lokacija na računalu

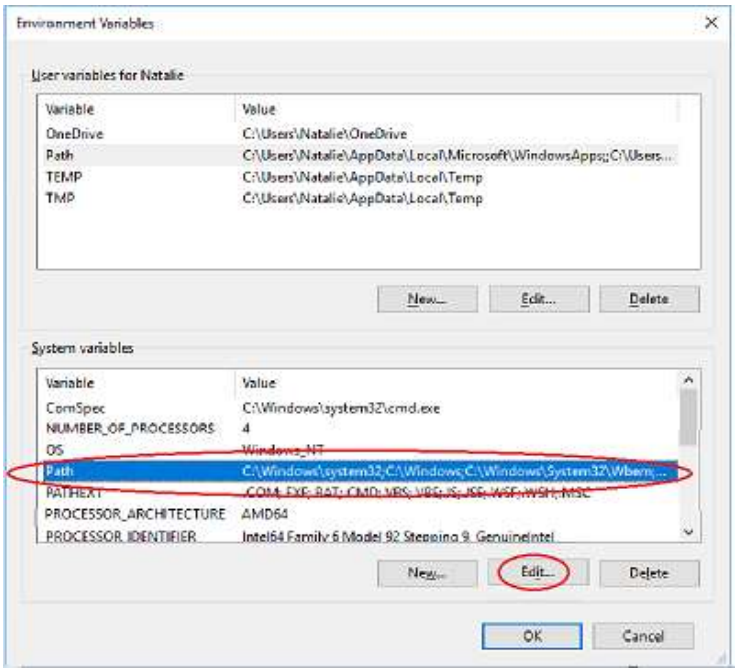
3. Postavljanje u verijable okruženja sustava (eng. *Environment variables*).

   Edit the system environment variables > Environment Variables.



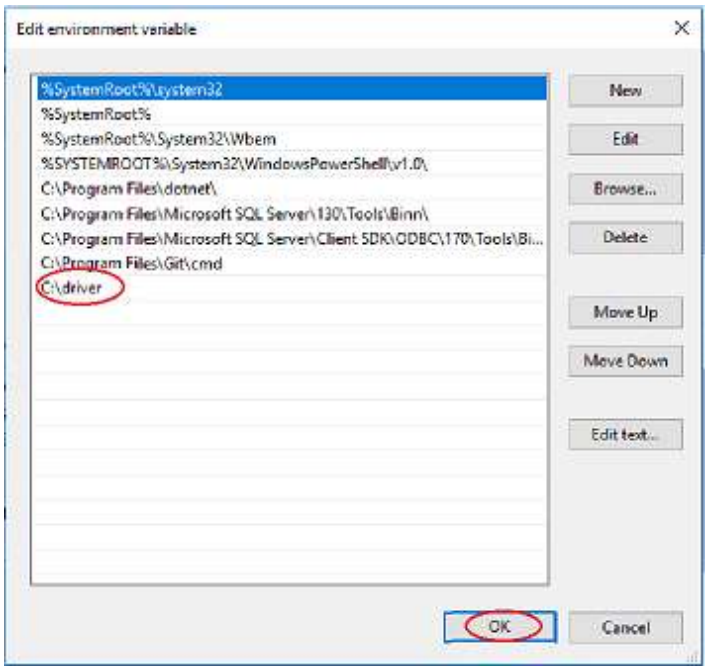
Slika 18. Varijable okruženja

U sistemskim varijablama odabir '*Path*' i klik na '*Edit*'.



Slika 19. Postupak dodavanja varijable okruženja

Na kraju niza potreban je unos ';' i lokacije datoteke "chromedrive.exe" ili ručno dodavanje lokacije u *'Edit'* prozoru.

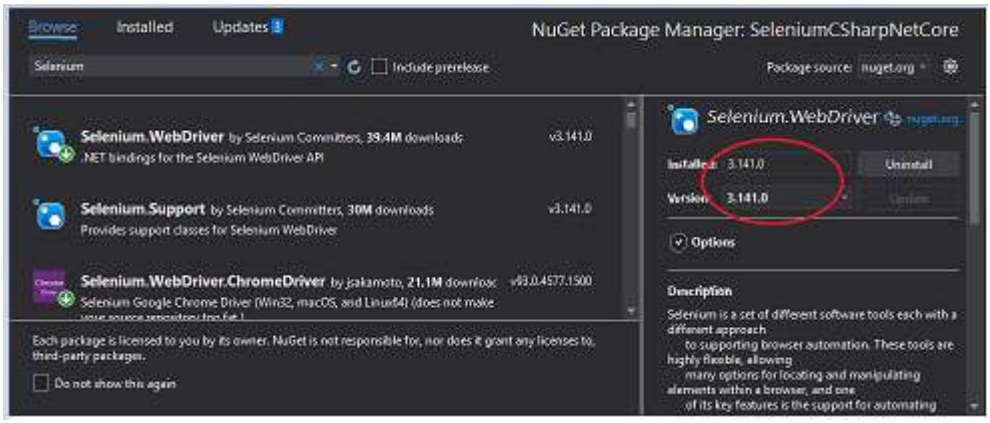


Slika 20. Dodavanje varijable okruženja

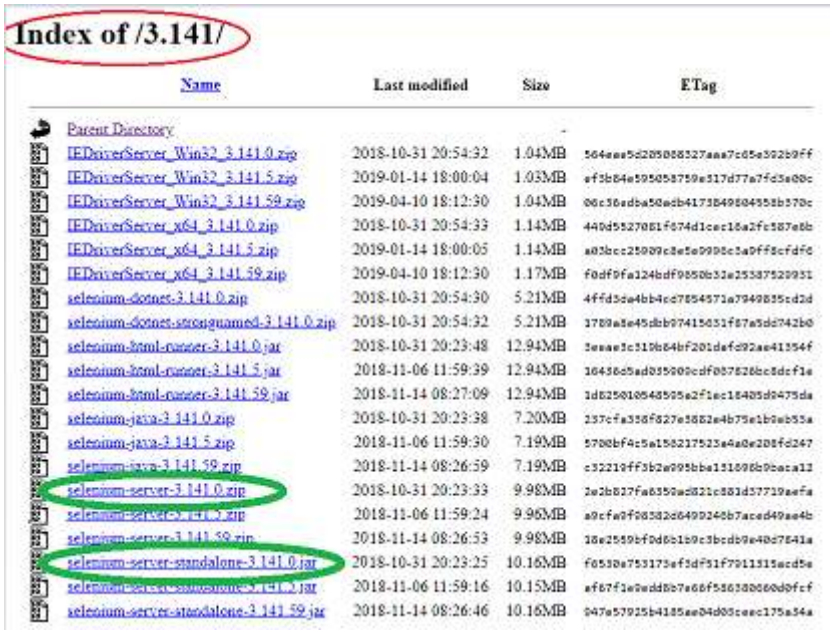
7.1.5. SELENIUM JAR DATOTEKE

Za pokretanje Selenium poslužitelja potrebna je Selenium JAR datoteka koja predstavlja skupinu API-a (eng. *Application programming interface*) za različite programske jezike.

Verziju je potrebno uskladiti s korištenom verzijom Selenium WebDrivera (označeno crvenom), te preuzeti prikladne potrebne datoteke "Selenium Server" i "Selenium Server Standalone" (označeno zeleno).



Slika 21. Provjera verzije Selenium.WebDrivera

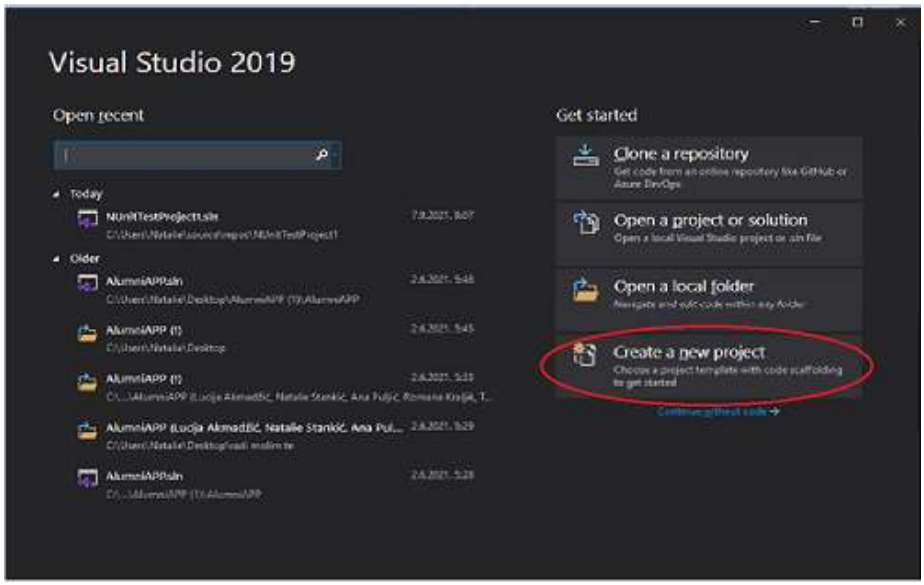


Slika 22. Preuzimanje "Selenium Server" i "Selenium Server Standalone" datoteka

Nakon preuzimanja, iste je potrebno raspakirati prema željenoj lokaciji na računalu. Implementacija u varijable okruženja sustava vrši se na isti način kao kod prethodno prikazanog Chrome WebDrivera.

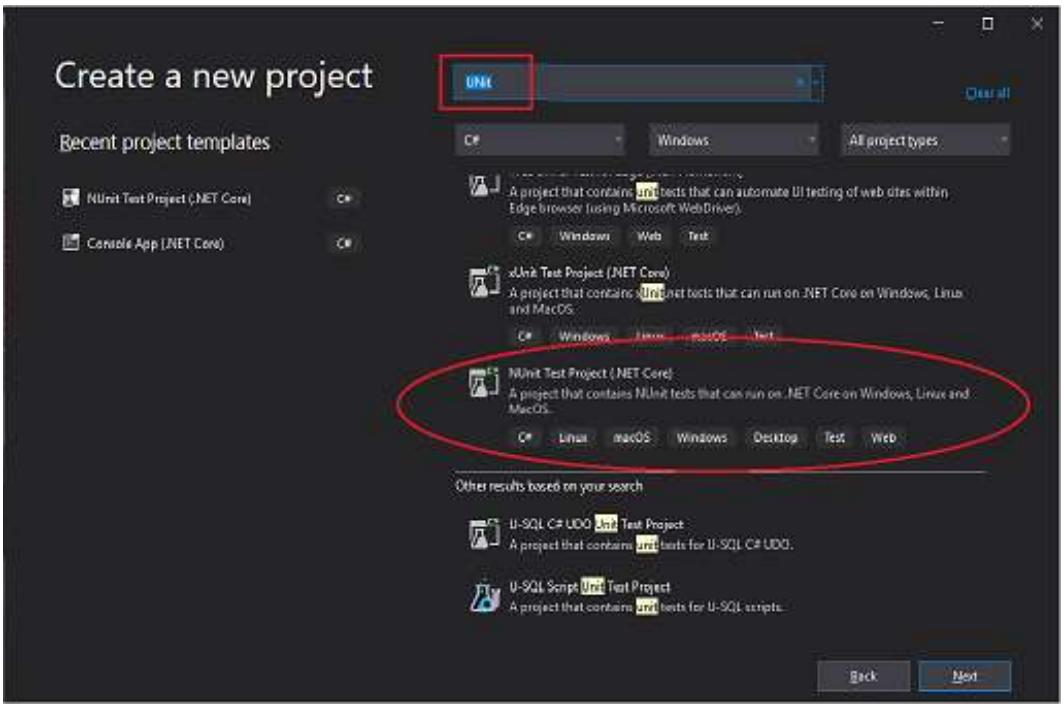
7.2. POSTUPAK IZRADE TESTA

Izrada novog projekta pokreće se klikom *na 'Create a new project'*.



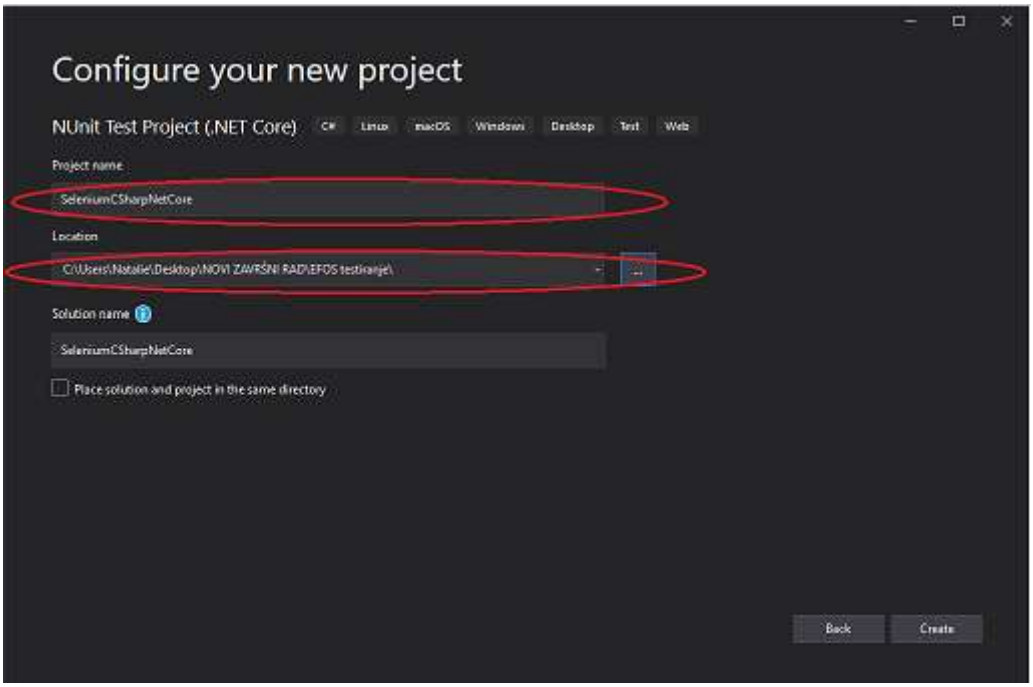
Slika 23. Kreiranje novog projekta

Prvi korak je odabir predloška projekta. S obzirom na opis željenog testiranja, za uspješnu izradu i pisanje testova odabire se "NUint Test Procject (.NET Core)".



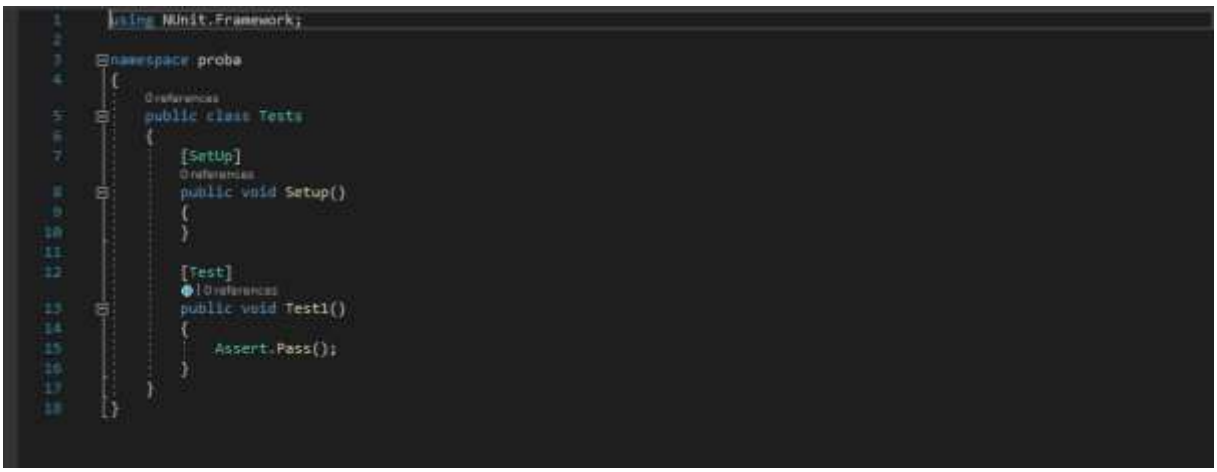
Slika 24. Odabir predloška projekta

Nakon odabira predloška projekta, projektu se dodjeljuje ime i odabire se lokacija pohrane na računalu. Klikom na *'Create'* stvara se novi projekt.



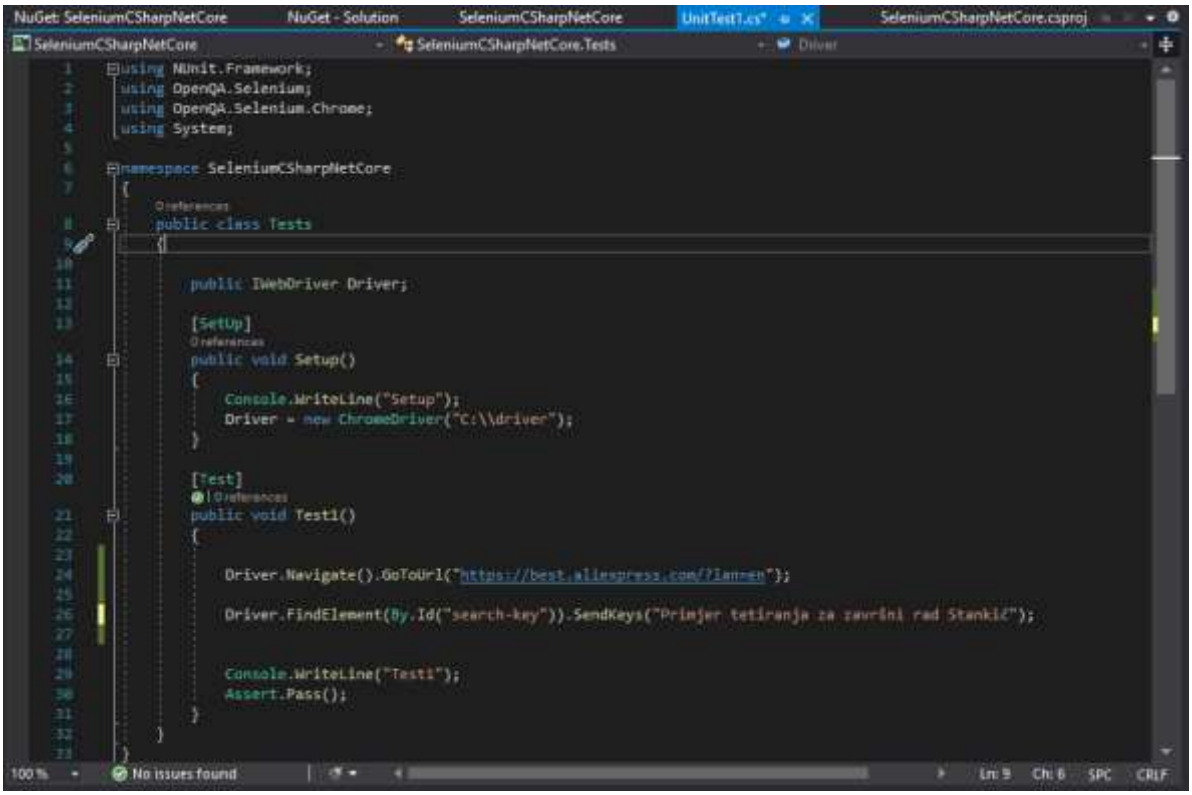
Slika 25. Konfiguracija novog projekta

Nakon klika na 'Create' otvara se slijedeći prozor:



Slika 26. Početno kreirani projekt

Kod se kreira unutar klase pod nazivom "Tests". Izvršava naredbe: dohvaćanja stranice "AliExpress" putem Google Chrome preglednika, pronalaska elementa stranice pomoću lokatora (Id), upisivanja sadržaja u polje lociranog elementa i ispisa rezultata testiranja.



Slika 27. Programski kod

Objašnjenje koda:

1. Metoda za komunikaciju s operativnim sustavom, upis teksa i ispis argumenta niza na konzoli.

```
Console.WriteLine();
```

2. Služi za definiranje sučelja (eng.Interface) putem kojeg se kontrolira preglednik.

```
public IWebDriver Driver;
```

3. Instanciranje objekata. Služi kako bi napisane skripte bile pokrenute pomoću odabranog preglednika.

```
Driver = new ChromeDriver("C:\\driver");
```

4. Izvršava dohvaćanje internetske stranice preko linka putem Chrome preglednika.

```
Driver.Navigate().GoToUrl("https://best.aliexpress.com/?lan=en");
```

5. Pronalazi željeni element na stranici putem lokatora elementa (Id) i upisuje određeni sadržaj u polje.

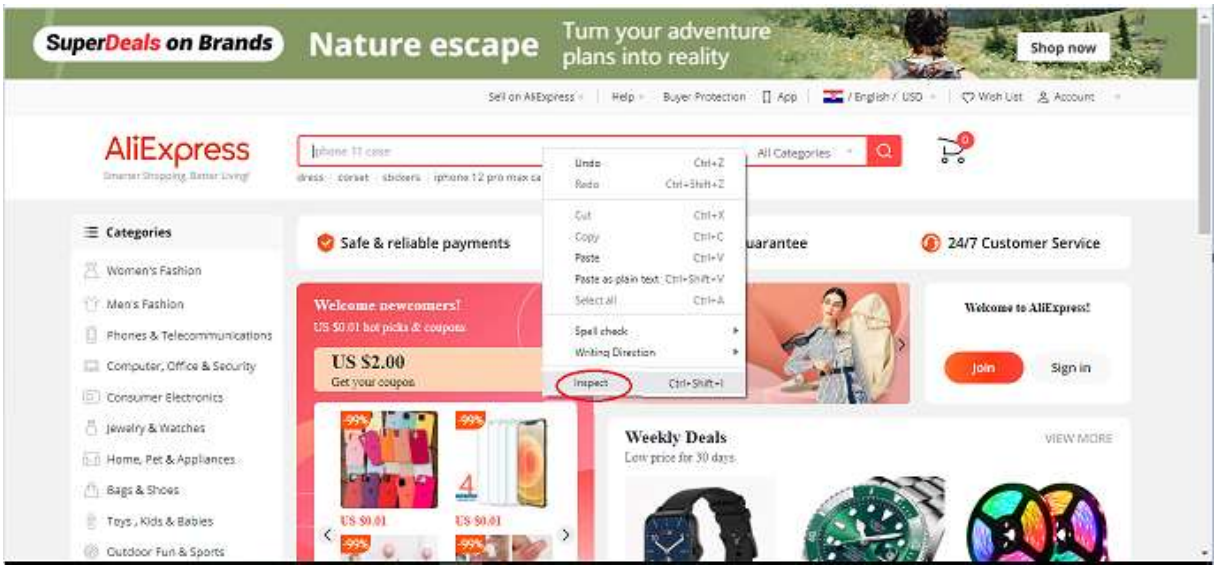
```
Driver.FindElement(By.Id("search-key")).SendKeys("Primjer tetiranja za završni rad Stankić");
```

6. Koristi se za provjeru valjanosti testnog slučaja, dali je test uspješan.

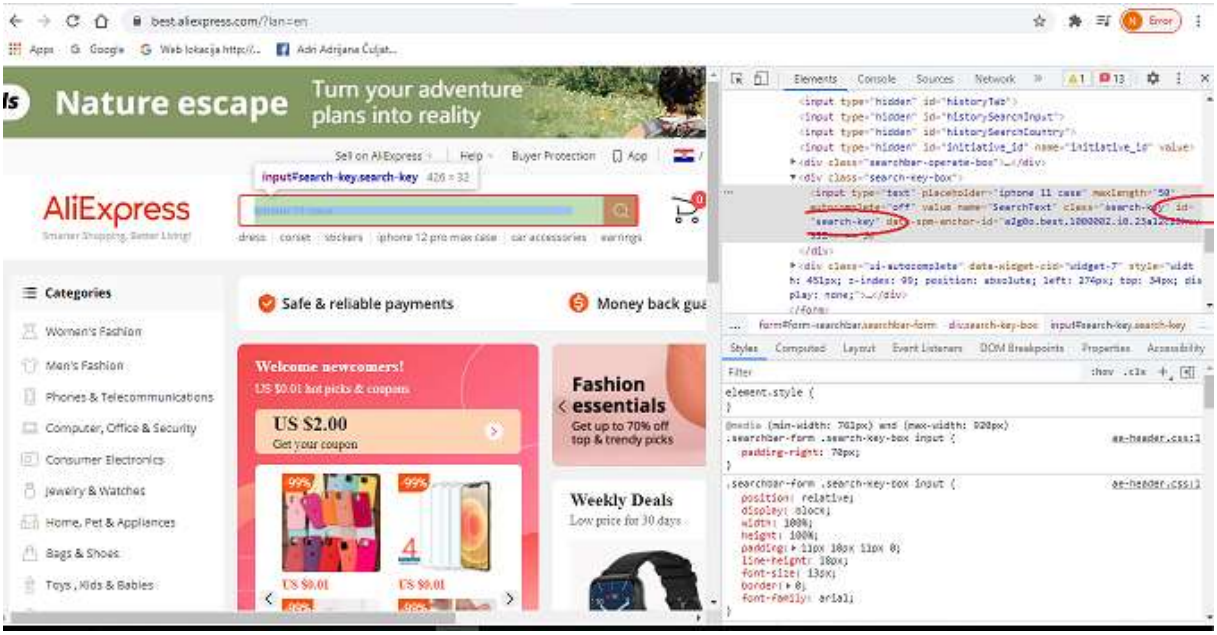
```
Assert.Pass();
```

Proces pronalaska lokatora (ID) elementa stranice: :

Desni klik na željeni element > *Inspect*



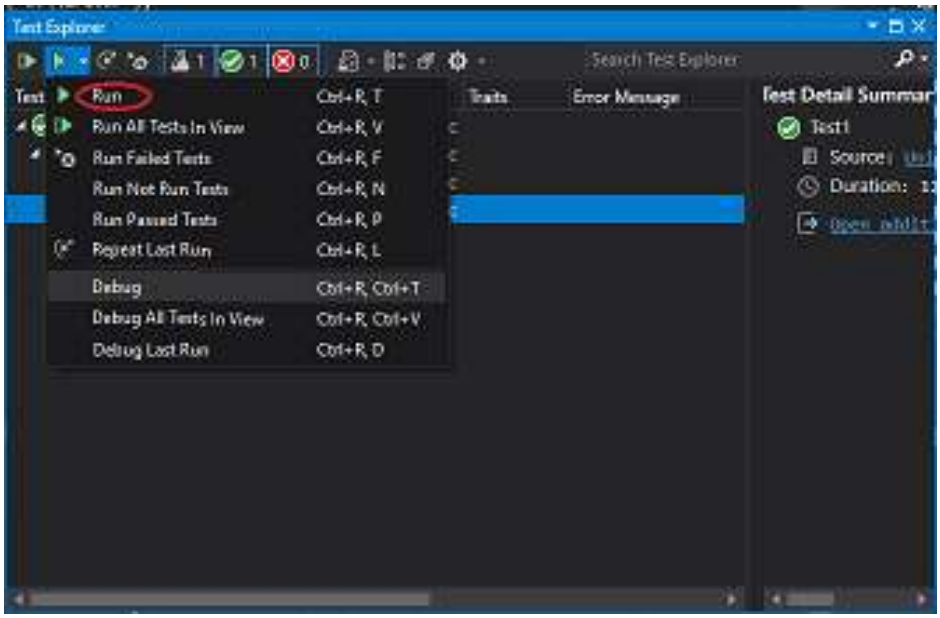
Slika 28. Postupak pronalaska lokatora elementa



Slika 29. Identifikacija ID lokatora

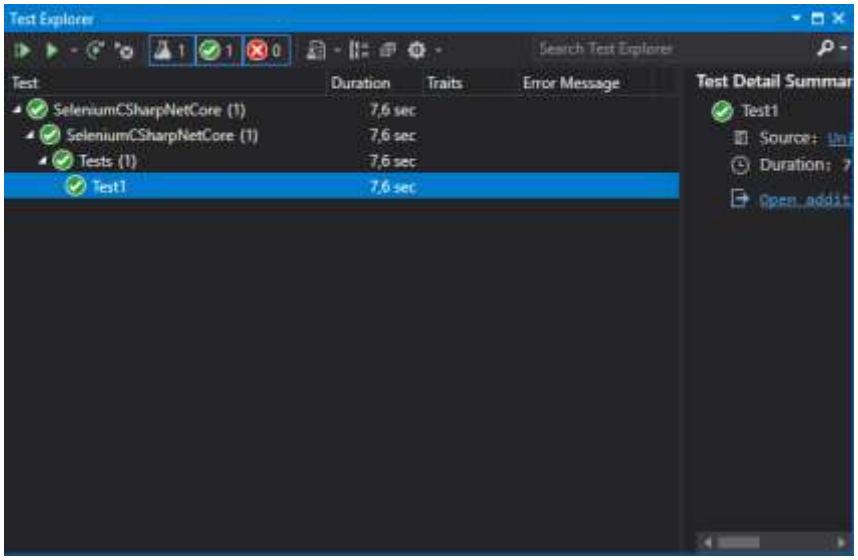
7.3. REZULTATI PROVEDENOG TESTIRANJA

Provjera kod testnih slučajeva pokreće se putem "Test Explorer" prozora klikom na *'Run'*.

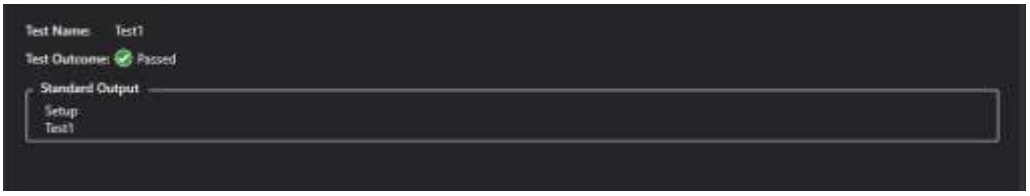


Slika 30. Pokretanje testnog slučaja

Testiranje je uspješno provedeno bez detektiranih grešaka u sustavu. Sustav ispunjava očekivane značajke: učitava navedenu stranicu, pronalazi željeni elemenent i ispisuje tekst.

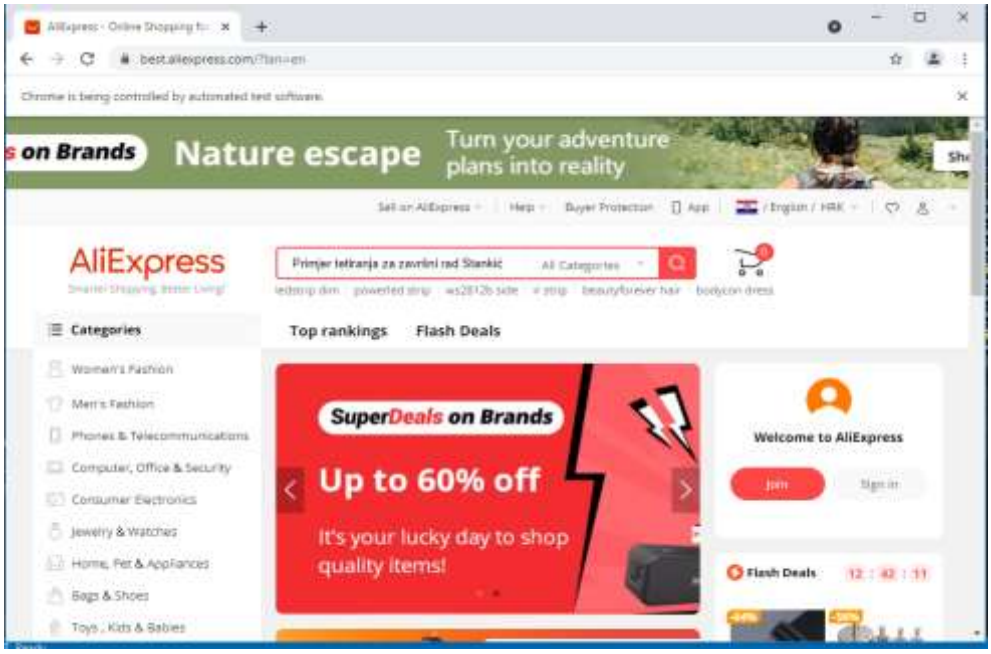


Slika 31. Provjera testnog slučaja



Slika 32. Prolazak testa

Test je uspješno izvršen putem Google Chrome preglednika.



Slika 33. Uspješan test

8. ZAKLJUČAK

Čitav niz malih odluka prilikom izbora procesa, faza, alata i brojnih drugih mogućnosti tijekom razvoja sustava može imati itekako velik utjecaj na ishod procesa testiranja. U globalu važno je samo jedno, ispuniti zacrtani cilj uvjeta kvalitete i sigurnosti te željenog izgleda i ključnih funkcionalnosti, bez obzira na sredstva korištena pri ostvarenju tog cilja. U suprotnom sve u što su uložen rad, trud i vrijeme postaje uzaludno i beskorisno. Testiranje mora biti pomno isplanirano od strane testera, prije kretanja u izvođenje testova, kako bi strategija izrade testnih skripti, scenarija, izvršavanja testova i njihovog vrednovanja bila jasna. Različite vrste, metode i procesi testiranja uz njihov ispravan odabir osiguravaju široku pokrivenost različitih značajki funkcionalnosti programa. Programska rješenja potrebno je provjeriti i osloboditi grešaka, a rano otkrivanje grešaka štedi resurse i vrijeme. Ručno testiranje je dugotrajan proces testiranja funkcionalnosti programa koji se provodi bez korištenja alata, dok automatsko testiranje koristi alate, ali je u početku skupo i neisplativo. Jednom kada su testovi automatizirani štedi se na vremenu i resursima ponavljajući ih kod sličnih ili ponavljajućih testova i koristeći ih na novim projektima. Alate se odabire na temelju specifičnosti i zahtjeva projekta. Ključna je njihova korisnost i kompatibilnost potrebama projekta kako bi se uvelike poboljšala kvaliteta i produktivnost testiranja programskog rješenja. Ukoliko je svaki segment uključen u proces testiranja kvalitetno razmotren, isplaniran i odrađen, rezultat i sveukupno zadovoljstvo na kraju u konačnici čine testiranje neizostavnim dijelom razvojnog procesa programa.

9. POPIS LITERATURE

Knjige:

1. Dustin, E., Garrett, T., Gauf, B. (2009). *Implementing automated software testing*. Addison-Wesly professional.
2. Fewster, M., Graham, D. (1994). *Softwere Test Automation: Techniques for Automating Test Execution.* London: ACM Press Books.
3. Graham, D., Veenendaal, E.V., Evans, I. (2008). *Foundations of Software Testing: ISTQB Certification.* UK: Gaynor Redvers-Mutton.
4. Myers, G.J. (2004). *The Art of Software Testing - Second Edition.* New Jersey: Word Association, Inc.
5. Myers, G.J., Badgett, T., Sandler, C. (2012). *The art of software testing - Third edition.* New Jersey: Hoboken. N.J., USA: John Wiley & Sons.
6. Naik, S., Tripathy, P. (2008). *Software testing and quality assurance: Theory and practice.*New Jersey: John Wiley & Sons, Inc.
7. Patton, R. (2006*). Software testing - Secound Edition.* Indianapolis: SAMS.
8. Spillner, A., Linz, T., Schaefer, H. (2014). *Software testing foundations*. Rocky Nook Computing.

Kvalifikacijski radovi:

9. Acharya, S., Pandya, V. (2021). *Bridge between black box and white box – gray box testing technique.* International Journal of Electronics and Computer Science Engineering.
10. Ammann, P., Offutt, J. (2017). *Introduction to software testing*. Cambridgeshire: Cambridge University Press.
11. Babbar, H. (2017). *Software testing: techniques and test cases*. Landran, Chandigarh group of colleges.
12. Majstorović, Z. (2011). *Validacija, verifikacija i testiranje programske opreme.* Varaždin, Sveučilište u Zagrebu: FOI.
13. Pranić, M. (2010). *Alati za testiranje softvera*. Split, Sveučilište u Splitu: FESB.

Prezentacije:

14. Fernandes, J., Di Fozno, A. (2017). *When to automate your testing (and when not to)* Dostupno: https://silo.tips/download/when-to-automate-your-testing-and-when-not-to
15. McCown, F. (2013). *A short history of computing*. Dostupno: https://slideplayer.com/slide/4999918/

Internet izvori :

16. Alex, K. (2020). Pros&Cons of Unit Testing. Dostupno na: https://dev.to/hiretester/pros-cons-of-unit-testing-2nn9 [pristupljeno 5. lipnja 2021.].
17. AltexSoft (2018). Comparing Automated Testing Tools: Selenium, TestComplete, Ranorex, and more. Dostupno na: https://www.altexsoft.com/blog/engineering/comparing-automated-testing-tools-selenium-testcomplete-ranorex-and-more/ [pristupljeno 29. svibnja 2021.].
18. Altexsoft (2021). The good and the bad of Katalon studio automation testing tool. Dostupno na: https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-katalon-studio-automation-testing-tool/ [pristupljeno 20. lipnja 2021.].
19. Anderson, B. (2021). Best automation testing tools for 2021 (Top 15 reviews). Dostupno na: https://briananderson2209.medium.com/best-automation-testing-tools-for-2018-top-10-reviews-8a4a19f664d2 [pristupljeno 15. kolovoza 2021.].
20. Appium (2021). Introduction to Appium. Dostupno na: https://appium.io/docs/en/about-appium/intro/ [pristupljeno 26. kolovoza 2021.].
21. GeeksforGeeks (2020). Types of Software Testing. Dostupno na: https://www.geeksforgeeks.org/types-software-testing/ [pristupljeno 28. svibnja 2021.].
22. Gupta, R. (2019). Ad-hoc testing vs Exploratory testing. Dostupno na: https://www.webomates.com/blog/manual-testing/ad-hoc-testing-vs-exploratory-testing/ [pristupljeno 27. srpnja 2021.].
23. Hamilton, T. (2021). Alpha Testing Vs Beta Testing: What's the Difference?. Dostupno na: https://www.guru99.com/alpha-beta-testing-demystified.html [pristupljeno 28. svibnja 2021.].

24. Hamilton, T. (2021). AUTOMATION TESTING Tutorial: What is, Process, Benefits & Tools. Dostupno na: https://www.guru99.com/automation-testing.html [pristupljeno 28. svibnja 2021.].
25. Hamilton, T. (2021). Positive testing and negative with examples. Dostupno na: https://www.guru99.com/positive-and-negative-testing.html [pristupljeno 17. srpnja 2021.].
26. Hamilton, T. (2021.). Manual testing tutorial: What is, concepts, types and tool. Dostupno na: https://www.guru99.com/manual-testing.html [pristupljeno 6. lipnja 2021.].
27. Hr.education-wiki (2021). Agilno testiranje. Dostupno na: https://hr.education-wiki.com/3829874-agile-testing [pristupljeno 4. srpnja 2021.].
28. Incredibuild (2021). What is Visual Studio. Dostupno na: https://www.incredibuild.com/integrations/visual-studio [pristupljeno 2. rujna 2021.].
29. ISO/IEC (2019). International standard ISO/IEC 9126. Dostupno na: https://www.iso.org/home.html [pristupljeno 6. lipnja 2021.].
30. ISTQB Glossary (n.d.). Testing. Dostupno na: https://glossary.istqb.org/en/search/testing [pristupljeno 19. srpnja 2021.].
31. JavaTpoint (2021). Black box testring vs white box testing vs grey box testing. Dostupno na: https://www.javatpoint.com/black-box-testing-vs-white-box-testing-vs-grey-box-testing [pristupljeno 27. svibnja 2021.].
32. Jelić, U. (2020). Zašto je posao QA testera baš za tebe. Dostupno na: https://www.helloworld.rs/blog/Zasto-je-posao-QA-testera-bas-za-tebe/10684 [pristupljeno 23. svibnja 2021.].
33. Jovanović, J.S. (2018). Agilno testiranje. Dostupno na: https://blog.itkonekt.com/2018/06/07/agilno-testiranje/ [pristupljeno 5.srpnja 2021.].
34. Leksikografski zavod Miroslav Krleža (2021). Hrvatska enciklopedija, mrežno izdanje: metoda. Dostupno na: https://www.enciklopedija.hr/Natuknica.aspx?ID=40437 [pristupljeno 20. kolovoza 2021.].
35. Limbachiya, N. (2018). Black box testing techniques for security – your guide. Dostupno na: https://www.kiwiqa.com/black-box-testing-techniques-for-security-your-guide / [pristupljeno 28. svibnja 2021.].
36. Meerts, J., Graham, D. (2019). The history of software testing. Dostupno: http://www.testingreferences.com/testinghistory.php [pristupljeno 2. lipnja 2021.].

37. Myservername.com (2021). Najpopularniji alati za testiranje web aplikacija. Dostupno na: https://hr.myservername.com/top-30-web-application-testing-tools-2021 [pristupljeno 14. srpnja 2021.].
38. Norville, L. (2017). 25 functional testing types – examples, tips and more. Dostupno na: https://www.qasymphony.com/blog/functional-testing-types/ [pristupljeno 14. srpnja 2021.].
39. Ranorex GmbH (2021). Ranorex studio: Functional UI Test Automation. Dostupno na: https://www.ranorex.com/ [pristupljeno: 16. kolovoza 2021.].
40. Selenium (2021). About Selenium. Dostupno na: https://www.selenium.dev/about/ [pristupljeno 15. kolovoza 2021.].
41. Shiklo, B. (2019). 8 software developing models: sliced, diced and ogranized in charts. Dostupno na: https://www.scnsoft.com/blog/software-development-models [pristupljeno 14. kolovoza 2021.].
42. Singh Gill, N. (2018). White box testing techniques and advantages. Dostupno na: https://www.xenonstack.com/insights/what-is-white-box-testing [pristupljeno 28. svibnja 2021.].
43. Smartbear (2021), What is automated Testing ?. Dostupno na: https://smartbear.com/learn/automated-testing/what-is-automated-testing/ [pristupljeno 2. kolovoza 2021.].
44. SoapUI (2021). Soap testing. Dostupno na: https://www.soapui.org/docs/soap-and-wsdl/ [pristupljeno 16. kolovoza 2021.].
45. Software testing fundamentals (2021). Agile Testing. Dostupno na: https://softwaretestingfundamentals.com/agile-testing/ [pristupljeno 4. srpnja 2021.].
46. Software testing fundamentals (2021). Black Box Testing. Dostupno na: http://softwaretestingfundamentals.com/black-box-testing/ [pristupljeno 27. svibnja 2021.].
47. Software testing fundamentals (2021). Gray Box Testing. Dostupno na: http://softwaretestingfundamentals.com/gray-box-testing/ [pristupljeno 27. svibnja 2021.].
48. Software testing help (2021). What is Automation Testing (Ultimate Guide To Start Test Automation). Dostupno na: https://www.softwaretestinghelp.com/automation-testingtutorial-1/ [pristupljeno: 21. kolovoza 2021.].

49. Test Automation Resources (2018). 5 software testing methods. Dostupno na: https://testautomationresources.com/software-testing-basics/software-testing-methods/ [pristupljeno 6. srpnja 2021.].
50. TestingWhiz (2019). Types of non functional testing ans its objectives. Dostupno na: https://www.testing-whiz.com/blog/types-of-non-functional-software-tests [pristupljeno 6. lipnja 2021.].
51. Turorialspoint (2021) Software testing tutorial. Dostupno na: https://www.tutorialspoint.com/software_testing/index.htm [pristupljeno 17. srpnja 2021.].
52. TutorialsTeacher (2020). .NET Core overview. Dostupno na: https://www.tutorialsteacher.com/core/dotnet-core [pristupljeno 5. rujna 2021.].
53. Vasylyna N. (2019). How to involve non-functional testing. Dostupno na: https://blog.qatestlab.com/2011/04/01/how-to-involve-non-functional-testing-in-your-project/ [Pristupljeno 13. kolovoza 2021.].
54. Webdriver IO (2021). WebDriver bindings for Node.js. Dostupno na: http://v4.webdriver.io/ [pristupljen 21. kolovoza 2021.].

10. POPIS SLIKA

11.POPIS TABLICA