Testiranje softvera

Mraović, Marko

Sveučilište Jurja Dobrile u Puli

Fakultet informatike u Puli

MARKO MRAOVIĆ

Testiranje softvera

Završni rad

Pula, 2020.

Sveučilište Jurja Dobrile u Puli

Fakultet informatike u Puli

MARKO MRAOVIĆ

Testiranje softvera

Završni rad

JMBAG: 0303069589, redovan student

Studijski smjer: Informatika

Predmet: Programsko inženjerstvo

Znanstveno područje: Društvene znanosti

Znanstveno polje: Informacijske i komunikacijske znanosti

Znanstvena grana: Informacijski sustavi i informatologija

Mentor: izv. prof. dr. sc. Tihomir Orehovački

Pula, 2020.

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani Marko Mraović, kandidat za prvostupnika Informatike ovime izjavljujem da je ovaj Završni rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio Završnog rada nije napisan na nedozvoljen način, odnosno da je prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student

_____

U Puli, _____, _____ godine

IZJAVA

o korištenju autorskog djela

Ja, Marko Mraović dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj završni rad pod nazivom Skladišta podataka i poslovna inteligencija koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama. Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

U Puli, _____(datum)

Potpis

_____

Testiranje softvera

Marko Mraović

Sažetak: U ovom završnom radu objašnjeno je testiranje softvera kroz njegove procese i razine. Navedene su različite vrste testiranja kroz različite metode i alati potrebni za to isto testiranje. Također je pojašnjeno otklanjanje greški koje će se pojavljivati kroz testiranje. U radu je prikazan rad sa automatskim alatom Selenium.

Ključne riječi: softver, testiranje softvera, alati za testiranje, Selenium

Software testing

Abstract: This bachelor thesis explains software testing through its processes and levels. Different types of testing are listed through different methods and tools needed for the same testing. The debugging that will occur through testing has also been clarified. The paper presents the work with the automatic tool Selenium.

Keywords: software, software testing, testing tools, Selenium

Sadržaj

1. Uvod

Tema završnog rada je testiranje softvera. Dva pojma koji u današnjem modernom razvoju softvera moraju ići jedno uz drugo. Softver je korišten i raširen svugdje u svijetu zbog modernizacije i primjene novih tehnologija. Naravno razvila se želja da se taj isti softver usavrši i pripremi za nesmetano korištenje uz što veću kvalitetu. Tu dolazi do izražaja kvalitetno testiranje softvera. U povijesti na testiranje se gledalo kao na sporednu granu razvoja softvera, danas je to ipak postala jedna od primarnih aktivnosti. Testiranje je trenutno implementirano od samog početka u proces razvoj softvera da bi se što više smanjile pojave greški tijekom razvoja i naravno cijena tog istog softvera. Procjenjuje se da je trenutno više od 50% resursa i vremena utrošeno u testiranje tijekom izgradnje softvera.

Kroz rad su prikazane i objašnjene različite vrste testiranja, također i određene razine kroz koje prolazi testiranje softvera, te iste metode koje se koriste da bi se izvršilo testiranje. Prikazani su alati za testiranje koji se najčešće koriste za određenu komponentu u softveru. Tijekom razvoja i nakon završetka izgradnje softvera očekuje se prisutnost grešaka, one su sastavni dio svakog razvoja te ih je nemoguće izbjeći, smatra se da je bolje pronaći greške jer to samo može pomoći pri usavršavanju softvera. Prikazane su različite vrste greški i osnove razlike između njih. Opisan je proces pronalaženja i otklanjanja greški, koji je ujedno jedan od najbitnijih faktora za kvalitetno i temeljito testiranje. Navedene su tehnike i metode koje se koriste za otklanjanje grešaka.

U drugom dijelu rada prikazan je praktični primjer rada sa automatiziranim alatom Selenium, točnije njegovom komponentom WebDriverom. Prikazana je automatizacija na pregledniku, uz dodatan primjer data testiranja.

Također je prikazan i alat JMeter koji se koristi za ispitivanje performansi, obavljeno je testiranje opterećenja.

2. Testiranje softvera

Što je softver te kako ga definirati i objasniti? Postoji mnogo izraza i definicija.

Softverski produkt ili skraćeno softver je skup računalnih programa i pripadne dokumentacije, stvoren zato da bi se isporučio nekom korisniku. Može biti razvijen za sasvim određenog korisnika ili općenito za tržište. Softverski produkt često se kraće naziva softver. Za današnji softver podrazumijeva se da on mora biti kvalitetan. Preciznije, od softverskog produkta očekuje se da se on odlikuje sljedećim atributima kvalitete:
Mogućnost održavanja : Softver se mora moći mijenjati u skladu s promijenjenim potrebama korisnika.

Pouzdanost i sigurnost : Softver se mora ponašati na predvidiv način te ne smije izazivati fizičke ili ekonomske štete.

Efikasnost : Softver mora imati zadovoljavajuće performanse te on treba upravljati računalnim resursima na štedljiv način.

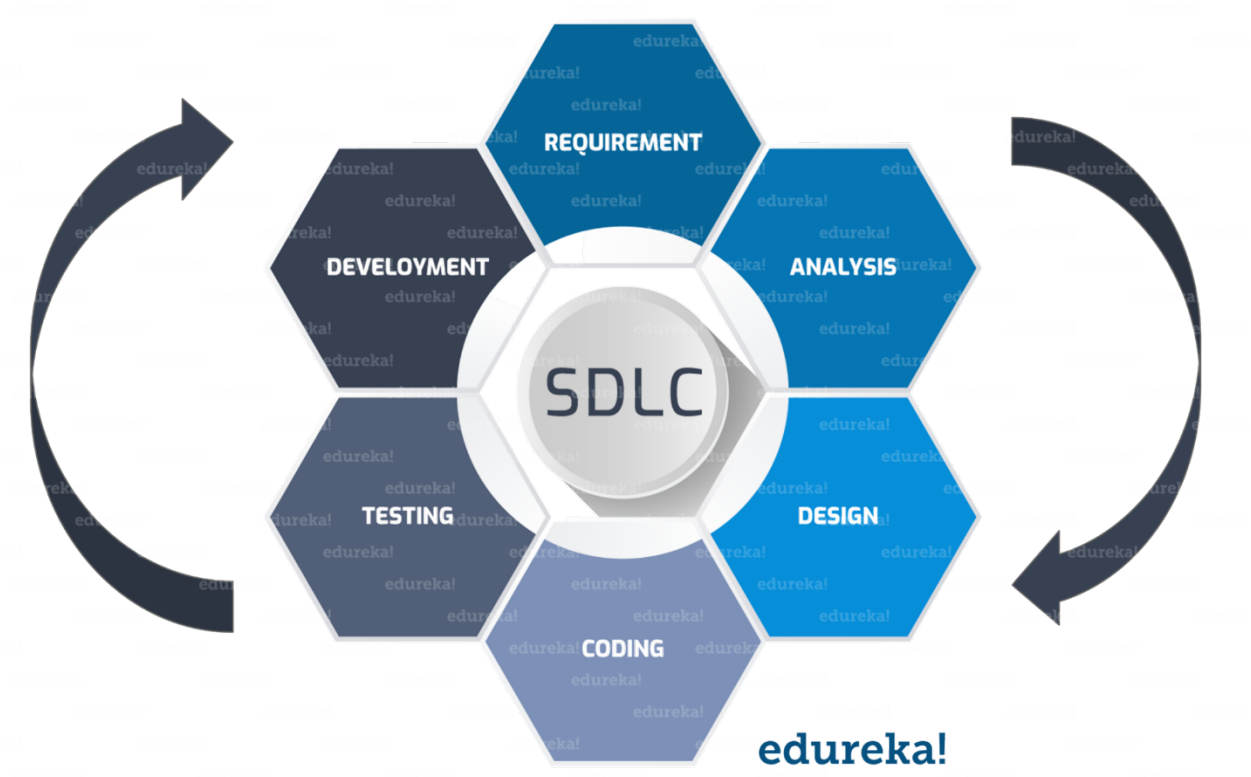
Upotrebljivost : Softver treba raditi ono što korisnici od njega očekuju, sučelje mu treba biti zadovoljavajuće te za njega mora postojati dokumentacija (Manger, 2005).

Institut inženjera elektrotehnike i elektronike (IEEE) definirao je softver kao :
Računalni programi, procedure i povezana dokumentacija i podaci koji se odnose na rad računalnog sustava (ISO19770-2:4.1.25).

2.1 Testiranje u sklopu razvoja softvera

Testiranje softvera primjenjuje se tijekom cijelog razvoja softvera, najčešće od samog početka jer iznimno smanjuje cijenu i moguće greške nastale u daljem razvoju. Životni ciklus razvoja softvera postupak je koji softverska industrija koristi za dizajn, razvoj i testiranje visokokvalitetnog softvera. Cilj mu je proizvesti visokokvalitetni softver koji

ispunjava ili premašuje očekivanja kupaca, postiže završetak u roku i procjeni troškova. Na slici 1 su prikazane različite faze uključene u tom ciklusu (edureka,n.d).



Slika 1. Prikaz razvoja softvera (edureka, n.d)

Kada se pomisli na testiranje, prva asocijacija je testiranje programskog koda ali samo to nije dovoljno, testiranje je jedan širi pojam.
Konkretno testiranje bi mogli definirati kao : „Testiranje je postupak izvršavanja programa s namjerom pronalaženja pogrešaka“ (Myers, 2004).

Test slučaj (eng. Test case) je dokumentirani skup unosa podataka i radnih uvjeta potrebno za pokretanje ispitnog predmeta zajedno s očekivanim rezultatima izvođenja. Očekuje se da tester pokrene program za testnu stavku prema testu slučaja, a zatim usporedi stvarne rezultate s očekivanim, rezultati su zabilježeni u dokumentima. Ako se

dobiveni rezultati u potpunosti slažu sa očekivanim rezultatima, nije prisutna pogreška ili je barem utvrđena (Galin, 2004).

Svrha testiranja je pomoć pri razvoju softvera na način da osigura i stvori razinu povjerenja u ispravnost softvera, a to postiže uz verifikaciju i validaciju.

Iako su ta dva pojma slična po objašnjenju, razlika je velika.

Verifikacija (Are we building the product right?) provjerava da li softver odgovara specifikaciji, dakle dokumentu i njegovim zahtjevima. U dizajnu i razvoju verifikacija se odnosi na postupak ispitivanja rezultata dane aktivnosti kako bi se utvrdila sukladnost s navedenim zahtjevima za tu aktivnost.

Validacija (Are we building the right product?) provjerava da li softver zadovoljava potrebe korisnika. Validacija se obično provodi na konačnom proizvodu pod određenim radni uvjetima (Singh,2019).

2.2 Faze životnog ciklusa testiranja softvera

Životni ciklus testiranja softvera slijed je određenih aktivnosti koje se provode tijekom procesa testiranja kako bi se osiguralo da su ispunjeni ciljevi kvalitete softvera. Suprotno uvriježenom mišljenju, testiranje softvera nije samo pojedinačna aktivnost. Sastoji se od niza aktivnosti koje se provode kao pomoć u certificiranju softverskog proizvoda.

Analiza zahtjeva prvi je korak uključen u životni ciklus testiranja softvera. Tijekom ove faze ispitni tim proučava zahtjeve sa stajališta ispitivanja kako bi identificirao klijentove zahtjeve.

Planiranje ispitivanja najvažnija je faza životnog ciklusa testiranja softvera u kojoj je definirana sva strategija testiranja. U ovoj je fazi uključen test menadžer koji određuje napore i procjene troškova za cijeli projekt, na način da definira cilj i opseg projekta (edureka, n.d).

Razvoj testnog slučaja uključuje stvaranje, provjeru i preradu test slučajeva i testnih skripti nakon što testni plan bude spreman. U početku se podaci o ispitivanju identificiraju, zatim izrađuju i pregledavaju, a zatim prerađuju na temelju preduvjeta. Tada tim započinje proces razvoja testnih slučajeva za pojedine jedinice.

Testiranje okruženja odlučuje o softverskim i hardverskim uvjetima pod kojima se radni proizvod testira. To je jedan od najbitnijih aspekata procesa ispitivanja i može se provesti paralelno s fazom razvoja testnog slučaja.

Izvršavanje testa je faza u kojoj se testiranje izrade softvera vrši na temelju planova ispitivanja i pripremljenih test slučajeva. Postupak se sastoji od izvršenja test skripte, održavanja test skripte i izvještavanja o greškama. Ako se prijave greške, vraća se natrag razvojnom timu radi ispravka i izvršit će se ponovno testiranje.

Faza **zatvaranja testiranja** je završetak izvođenja testa koji uključuje nekoliko aktivnosti poput izvještavanja o završetku testa, prikupljanja matrica završetka testa i rezultata ispitivanja. Članovi ispitnog tima sastaju se, raspravljaju i analiziraju ispitivanje kako bi identificirali strategije koje se trebaju provesti u budućnosti, uzimajući zapažanja iz trenutnog ciklusa ispitivanja (Guru, n.d).

2.3 Zašto testiramo?

Da bi se osigurala usklađenost softverskih aplikacija sa željenom kvalitetom:
Softverske aplikacije razvijene su za rješavanje poslovnih problema određenog kupca. Softverske tvrtke mogu poduzeti dva pristupa za rješavanje problema - prilagoditi već razvijeni proizvod u skladu sa zahtjevima ili mogu razviti novi softver od nule.

Recenzije i ušteda troškova:
Jedna od svrha testiranja softvera je rano otkrivanje grešaka. Kao što znamo, ljudi su uključeni u razvoj softverske aplikacije i očekuje se da će generirati jednu do tri greške

na stotinu redaka koda. To znači da će programske pogreške biti uvođene u svakoj fazi. Ako se te greške ne otkriju na vrijeme, troškovi popravljanja grešaka povećavaju se proporcionalno kašnjenju u pronalaženju grešaka.

Prevencija kvarova:
Prevencija kvarova je postupak koji pomaže u prevenciji programskih pogrešaka. Taj postupak ovisi o postupku testiranja softvera. Analiza nedostataka u svakoj fazi koristi se za stvaranje tehnika, smjernica i metoda za sljedeće faze kako se takvi nedostaci ne bi ponavljali. Prevencija oštećenja je postupak koji se neprestano poboljšava, a čiji uzročni uzroci rezultiraju kratkoročnim i dugoročnim akcijskim planovima. Kratkoročni akcijski planovi provode se odmah, dok se dugoročni akcijski planovi uključuju kao promjene procesa. Te izmjene procesa tada koriste svim sljedećim projektnim testiranjima (Desai i Srivastava, 2016).

2.4 Automatsko i ručno testiranje

Ručno ispitivanje izvodi se ručno. Stručnjaci za osiguranje kvalitete osiguravaju da aplikacije rade ispravno slijedeći uvjete napisane u testnim slučajevima. Unatoč svojoj primitivnoj prirodi, ručno testiranje je i dalje važno, jer se određene funkcije jednostavno ne mogu automatski testirati.

Prednosti ručnog ispitivanja: zahtijeva manje početnih ulaganja u usporedbi s automatskim, pruža točne povratne informacije sučelja zahvaljujući provjerama i analizama inženjera, provjerava čak i male preinake u pokretu i učinkovitije rješava složene slučajeve upotrebe.

Nedostatci su mu: ručno testiranje zahtijeva više vremena ili više resursa, ponekad oboje, ispitivanje performansi nepraktično je u ručnom ispitivanju, ponavljanje istih testova, nije prikladno za velike projekte i vremenski ograničene projekte, usporedba velike količine podataka je nepraktična, često je neprecizan.

Automatsko testiranje koristi alate za automatizaciju za pisanje i izvršavanje testnih slučajeva, pri izvršavanju automatiziranog testiranja nije potrebno ručno sudjelovanje.

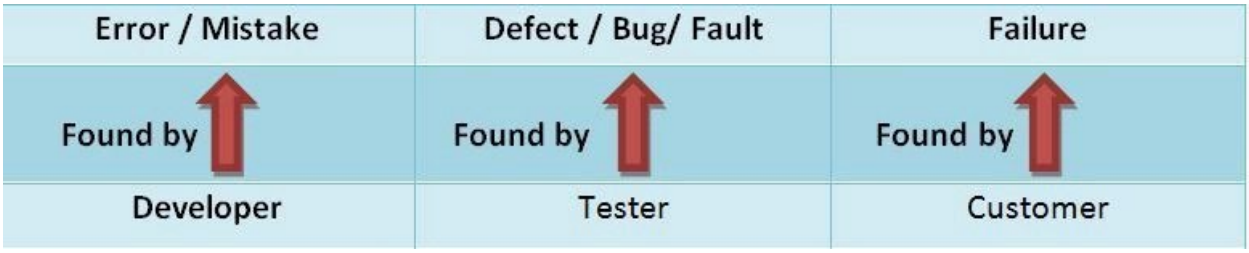
Obično testeri pišu testne skripte i test slučajeve pomoću alata za automatizaciju, a zatim se grupiraju u testne pakete.

To je u osnovi postupak automatizacije ručnog testiranja (Sharma,2014).

3. Greške u softveru

Pogreška(error), nedostatak(bug) i neuspjeh (failure) ovi su pojmovi sastavni dio procesa testiranja softvera i bez njihovog otkrivanja tim testera ne može provjeriti kvalitetu, učinkovitost i funkcionalnost softvera. Slika 2 prikazuje različitost između grešaka.

Sva ova odstupanja različito utječu na sustav, imaju prepoznatljivu kvalitetu i potpuno se razlikuju jedna od druge. Od razloga njihove pojave u softveru do koraka poduzetih za sprečavanje, većina se aspekata povezanih s ovim izrazima razlikuje (Medium,n.d).



Slika 2. Prikaz grešaka u softveru. (medium,n.d)

Pogreška(*error)*

Pogreška je ljudska pogreška. Ne pojavljuje se samo zbog logičke pogreške u kodu koji je napravio programer. Svatko u timu može pogriješiti tijekom različitih faza razvoja softvera.

Poslovni analitičar može pogrešno protumačiti ili pogrešno razumjeti zahtjeve.
Kupac može pružiti nedovoljne ili netočne podatke. Arhitekt može prouzročiti grešku u

dizajnu softvera. Ljudi u timu također mogu pogriješiti zbog nejasnih ili nedovoljnih zahtjeva, vremenskog pritiska ili drugih razloga.

Načini za sprečavanje pogrešaka : Poboljšati kvalitetu softvera pomoću pregleda koda i sustava.

Utvrditi probleme i pripremiti odgovarajući plan oporavka.

Provjeriti ispravke i potvrditi njihovu kvalitetu i točnost.

Usvojiti učinkovite metodologije razvoja softvera.

Nedostatak *(bug):*

Za razliku od pogreške, razlog za nastali nedostatak tijekom implementacije softvera nije zbog pogrešnog izračuna neke vrijednosti ili odstupanja stvarnog i očekivanog rezultata, već je samo dokaz pogreške u softveru. Štoviše, kvar na softverskom sustavu onemogućava mu izvršavanje zadane funkcije i prisiljava sustav na nepredviđeno djelovanje.

Razlozi grešaka mogu biti mnogi:

Pogreška pronađena u razvojnom okruženju softvera. Problem u softveru ili hardveru koji dovodi do kvara. Odstupanja u operativnom sustavu koji koristi program. Nekoliko grešaka čak je uzrokovano zbog pogrešnih kodova koje stvaraju prevoditelji (compilers).

Načini za sprečavanje bugova:

Prilagođavanje inovativnih i učinkovitih metodologija razvoja.

Nuditi podršku programskog jezika te temeljito analizirati kod.

Neuspjeh (*failure)*:

Kada softver nije u stanju izvršiti tražene funkcije i nudi rezultate koji nisu adekvatni i udaljeni su od očekivanih rezultata, to se naziva neuspjehom. Ti su kvarovi netočno vanjsko ponašanje koje vodi softver za isporuku usluga koje nisu u skladu s specifikacijama.

Razlozi neuspjeha:

Neuspjesi se obično događaju kada kvar koji postoji u sustavu izvrši tim, što ga prisiljava na nepredviđene i nepredviđene rezultate i funkcionira na neodgovarajući način.
Ostali razlozi koji dovode do kvara softvera su:

Ljudska pogreška: Pogreške koje je počinio čovjek tijekom interakcije sa softverom i davanjem pogrešnih ili nepotpunih podataka.

Uvjeti okoliša: Još jedan uobičajeni razlog koji prisiljava sustav na neuspjeh su uvjeti okoliša. Oni utječu na hardver i prisiljavaju ga da otkaže. Uz to, mijenja različite varijable okoline.

Korisnici: Neuspjesi se također mogu pojaviti u sustavu ako korisnik pokuša izvesti operacije koje mogu naštetiti funkcionalnosti softvera. Štoviše, oni također mogu izvršavati radnje s namjerom da onesposobe sustav, što će dalje dovesti do kvarova.

Upotreba sustava: Kvarovi se mogu pojaviti i zbog pogrešaka u načinu na koji se sustav koristi.

Načini za sprečavanje neuspjeha:

Identificirati i analizirati pogreške i probleme.

Usvojiti učinkovite preventivne tehnike.

Osigurati ponovno testiranje.

Revidirati i ponovno pregledati specifikacije i zahtjeve (ToolsQA,n.d).

4. Metodologija testiranja softvera

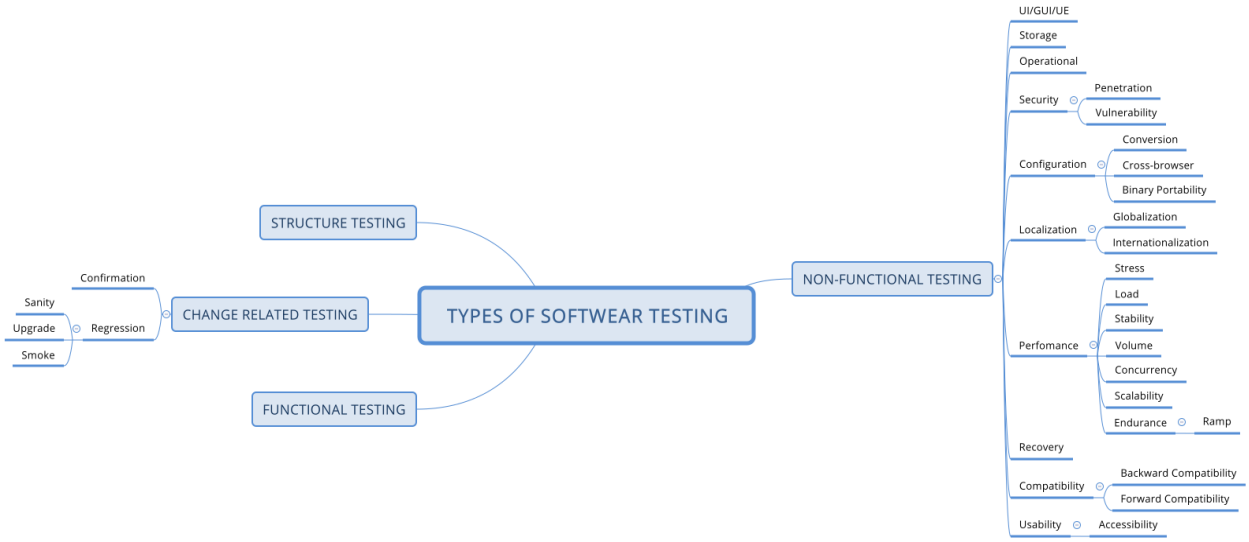
Slijedeći pojmovi iako slični razlikuju se uveliko, često dolazi do zabune između vrsta i razina testiranja, najveća razlika je u kojem se životnom ciklusu razvoja softvera one događaju. Ponekad se krivo interpretiraju i kao metode testiranja softvera.

4.1 Vrste testiranja

Testiranje softvera može uključivati jednu ili više od bilo koje od sljedećih vrsta testiranja, ovisno, naravno o opsegu testiranih aplikacija i proračunu softverske tvrtke. Na slici 3 moguće je vidjeti više vrsta testiranja, te njihovo daljnje grananje kroz vrste.

Razlikujemo 4 vrste testiranja:

- Funkcionalno
- Nefunkcionalno
- Strukturno
- Testiranje ovisno o promjenama



Slika 3. Prikaz vrsta testiranja (EasyQA,n.d)

Funkcionalno: odabir test slučajeva za funkcionalno ispitivanje temelji se na zahtjevu ili projektnoj specifikaciji.

Stoga se ispitivanje funkcionalnosti može provesti pomoću dvije tehnike:

Ispitivanje na temelju zahtjeva: Sadrži sve funkcionalne specifikacije koje su osnova za sva ispitivanja koja će se provesti.

Testiranje temeljeno na poslovnim scenarijima: Sadrži informacije o tome kako će se sustav percipirati iz perspektive poslovnog procesa.

Problem kod funkcionalnog testiranja može da se pojavi u slučaju dvosmislenih zahtjeva i nemogućnosti opisivanja svih načina korištenja softvera.

Nefunkcionalno: testiranje softverske aplikacije na nefunkcionalne zahtjeve. Ono pomaže procijeniti spremnost sustava prema različitim kriterijima koji nisu obuhvaćeni funkcionalnim ispitivanjem, testiranje mora biti mjerljivo.

Na slici 3 prikazuje se mnogo podtipova nefunkcionalnog testiranja, a evo i pojašnjenja za neke od njih.

Testiranje korisničkog sučelja (UI) ima za cilj osigurati da grafičko korisničko sučelje aplikacije zadovoljava specifikacijama. To pomaže u procjeni elemenata dizajna poput izgleda, boja, fontova, veličina fonta, naljepnica, tekstualnih okvira, oblikovanja teksta, opisa, ikona, itd.

Testiranje pohrane potvrđuje testiranu aplikaciju, pohranjuje relevantne podatke u ispravne direktorije i ima dovoljno prostora da spriječi neočekivani prekid zbog nedovoljnog prostora na disku. Pomaže utvrditi koristi li aplikacija više memorije nego što je procijenjeno jer punjenje prostora na disku može uzrokovati znatan zastoj.

Testiranje sigurnosti ima za cilj osigurati da informacijski sustav štiti podatke i održava funkcionalnost kako je predviđeno.

Ispitivanje penetracije i testiranje ranjivosti vrste su vrsta sigurnosnih ispitivanja.

Ispitivanje učinkovitosti namjerava utvrditi kako sustav funkcionira u smislu odziva i stabilnosti pod određenim opterećenjem. Vrste ispitivanja performansi: testiranje otpornosti na stres, ispitivanje opterećenja, ispitivanje stabilnosti, ispitivanje volumena, ispitivanje paralelnosti, ispitivanje skalabilnosti, ispitivanje izdržljivosti itd.

Testiranje oporavka ima za cilj procijeniti sposobnost sustava za oporavak od padova, kvarova hardvera ili drugih katastrofalnih problema. Izvode ga ispitni timovi.

Testiranjem kompatibilnosti provjerava se kompatibilnost aplikacije u različitim okruženjima: hardver, softver, operativni sustav, mrežno okruženje. Dvije su vrste ove vrste testiranja: unatražna kompatibilnost, prosljeđivanje kompatibilnosti.
Testiranje kompatibilnosti unatrag osigurava nastavak rada nove verzije proizvoda sa starijim proizvodom.
Prosljeđivanje kompatibilnosti omogućuje vezu s budućom verzijom proizvoda (edureka, n.d).

Strukturno ispitivanje smatra se više tehničkim nego funkcionalnim ispitivanjem. Pokušava dizajnirati test slučajevi iz izvornog koda, a ne iz specifikacija. Izvorni kod postaje bazni dokument koji se temeljito ispituje kako bi se razumjela unutarnja struktura i ostali detalji provedbe. Također daje uvid u izvorni kod koji se može koristiti kao osnovno znanje za dizajn test slučajeva (Singh, 2012).

Provjerava provedbu programa ili koda ispitivanjem strukture softverskog sustava ili njegovih komponenata. Ispitivač se koncentrira na rad softvera tijekom strukturnih ispitivanja. Može se koristiti na svim razinama testiranja.

Testiranje ovisno o promjenama najjednostavnije rečeno ono se koristi uglavnom u dvije svrhe:

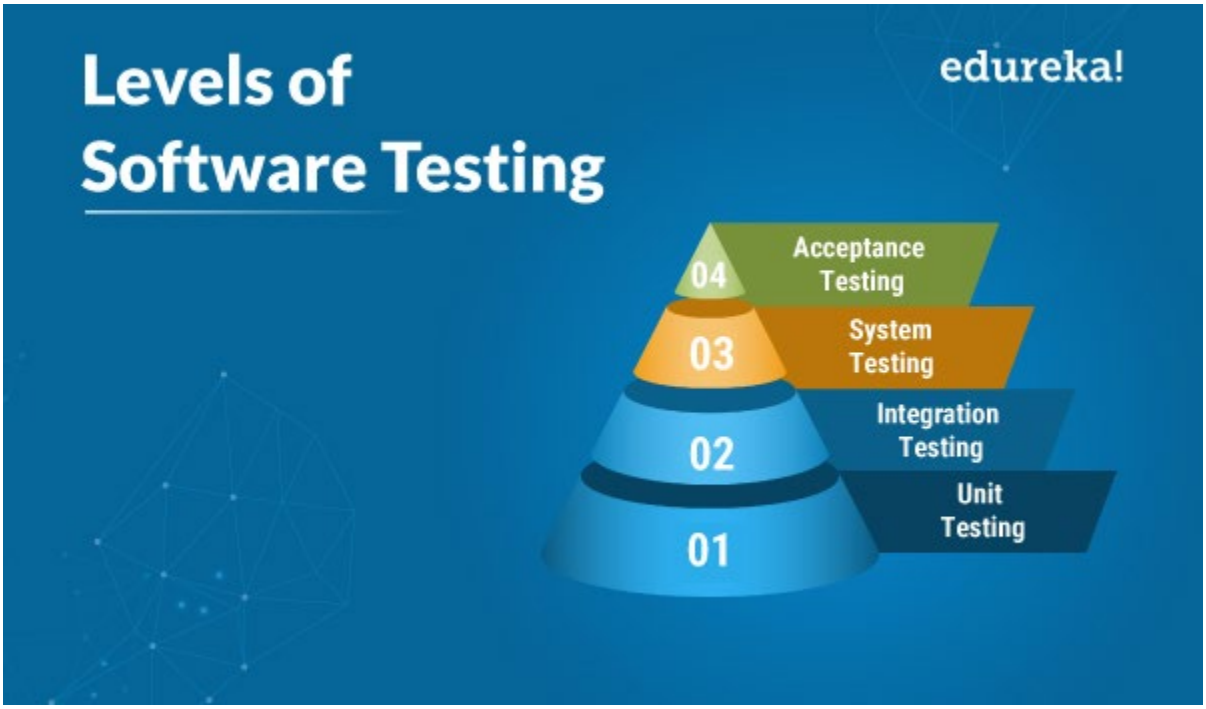
Testiranje potvrde: osiguravanje da su sve otkrivene pogreške uspješno ispravljene.

Test koji je otkrio kvar opet se provodi kako bi se osiguralo da je doista uklonjen.

Ispitivanje regresije: osiguravanje da se novi nedostatci ne pojave nakon izvršenih promjena. Generalno to je ponavljanje testova koji su već izvedeni bez problema kako bi se potvrdilo da novi nedostaci nisu nastali ili otkriveni nakon promjene. Drugim riječima, osigurava se da objekt koji se ispituje nije nazadovao (Gupta,n.d).

4.2 Razine testiranja

Razine testiranja moguće je promatrati pojedinačno jer svaka razina ima svoje značajke i zahtijeva specifičan pristup pri testiranju. Slika 4 prikazuje četiri razine testiranja, svaka od njih odvija se u različitom tijeku razvoja softvera i zbog određene svrhe.



Slika 4. Prikaz razina testiranja(edureka,n.d)

Ispitivanje modula(***unit testing***) postupak je testiranja pojedinačnih potprograma ili postupci u programu. Odnosno, umjesto da se isprva testira program u cjelini, testiranje je najprije usredotočeno na manje građevne dijelove programa. Motivacije za to su trostruke. Prvo, testiranje modula je način upravljanja kombiniranim elementima

testiranja, budući da je pažnja u početku usmjerena na manje cjeline programa. Drugo, testiranje modula olakšava zadatak otklanjanja pogrešaka budući da je pronađena pogreška. Konačno, uvođenje testiranja modula uvjetno je u procesu testiranja programa pružajući nam priliku za testiranje više modula istovremeno (Myers, 2004).

Integracijsko testiranje je testiranje nekompatibilnosti sučelja između inače ispravno radećih komponenata. Odnosno, to je testiranje potrebno za integraciju potkomponente u veću radnu komponentu. U praksi je često zanemareno te se ne obavi na adekvatan način što dovodi do ozbiljnih problema. Ovo testiranje je važno iz razloga da se izbjegnu svi potencijalni problemi koji mogu nastati prilikom spajanja komponenti u cjelinu (Singh,2019).

Testiranje sustava najteže je shvaćen i najteži postupak testiranja. Sistemsko testiranje fokusira se na kompletno integriran softver, da bi se provjerilo da li sve komponente rade zajedno kao cjelina (end to end test), to znači da se softver testira u realnom scenariju upotrebe. Testiranje se obavlja iz korisničke perspektive, sa dodatnom provjerom svakog ulaznog podatka i odgovarajućeg izlaza. Testiranje sustava nije ograničeno na sustave. Ako je proizvod program, ispitivanje sustava je postupak pokušaja da pokaže kako program kao cjelina ne ispunjava svoje ciljeve. Testiranje sustava je nemoguće ako ne postoji pisani i mjerljivi cilj proizvoda. Testiranje najčešće obavljaju najiskusniji testeri, moguće je angažirati nezavisan tim da bi se ostvarila potpuna objektivnost pri testiranju.

Ispitivanje prihvatljivosti je postupak usporedbe programa s početnim zahtjevima i trenutnim potrebama krajnjih korisnika. To je neobična vrsta testa zato što ga obično provodi kupac ili krajnji korisnik programa i obično se ne spada pod odgovornost razvojne organizacije.Ispitivanje prihvatljivosti provodi se samo kada je softver razvijen za određenog kupca (edureka,nd).

4.3 Metode testiranja

Jedna od važnijih metoda testiranja je testiranje u crnoj kutiji, na temelju podataka ili na ulazu / izlazu. Podatci o ispitivanju izvode se iz specificiranih funkcionalnih zahtjeva bez obzira na konačnu strukturu programa. Pri testiranju se koriste različiti ulazi i izlazi , zatim se uspoređuju sa specifikacijama kako bi se potvrdila ispravnost. Ne uzimaju se u obzir detalji implementacije koda. Istraživanje u crnim kutijama uglavnom se fokusira na to kako maksimizirati učinkovitost testiranja uz minimalne troškove.

Metoda bijele kutije suprotna je testiranju crne kutije, softver se gleda kao bijela kutija ili staklena kutija, struktura je vidljiva ispitivaču. Metoda bijele kutije fokusira se na to kako softver radi, može se primijeniti na bilo kojoj razini razvoja ali najčešće se koristi odmah nakon metode crne kutije. Planovi ispitivanja izrađuju se prema detaljima implementacije softvera, kao što su programski jezik, logika i stilovi.

Testiranje bijele kutije može se postići kroz određeni stupanj iscrpljenosti, poput izvršavanja svakog retka koda barem jednom (*coverage)*, prelaze svaku izjavu o grani (*branch coverage*) ili pokrivaju sve moguće kombinacije istinitog i lažnog stanja.

Ispitni slučajevi pažljivo su odabrani na temelju kriterija da su svi čvorovi ili staze pokriveni ili testirani barem jednom. Tako možemo otkriti nepotreban kod odnosno kod koji nema koristi ili se uopće nikad neće izvršiti, što ne može biti otkriven funkcionalnim ispitivanjem (Pan,1999).

5. Alati za testiranje

Na tržištu postoji mnogo alata potrebnih za testiranje softvera, svaki od njih služi za određenu svrhu testiranja.

Podijeljeni su po kategorijama:

Alati za dizajn testa su alati koji pomažu pri odluci koje testove treba izvršiti. Uključivanje alata za projektiranje u fazu dizajniranja testa može rezultirati proizvodnjom učinkovitih test slučajeva, jer su ti alati opremljeni specijaliziranim značajkama za proučavanje i analizu specifikacija i zahtjeva na formalni način kako bi se izvršili određeni zadaci potrebni za svrhe testiranja, poput dizajniranja i izrade testnih slučajeva na visokoj razini, generiranja ulaznih podataka za ispitivanje, ocrtavanja postupaka ispitivanja i mnogih takvih stvari. Ovi testni alati pomažu u smanjenju napora i vremena potrebnog za identificiranje i specificiranje testnih slučajeva, pokrivajući sve aspekte softverskog proizvoda (TryQA, n.d).

Alati za pripremu testnih podataka koriste se kada je za ispitivanje potreban širok raspon ili količina podataka. Vrlo su korisni za ispitivanje performansi i pouzdanosti, gdje je potrebna velika količina realnih podataka. Alati za pripremu podataka za testiranje omogućuju odabir podataka iz postojeće baze podataka ili stvaranje, generiranje, manipuliranje i uređivanje za upotrebu u testovima (TryQA, n.d).

Alati za izvršavanje testa nazivaju se alatima za probno testiranje jer pomažu u učinkovitom izvođenju testova. Alati za izvršavanje testa započinju snimanjem ili zabilježavanjem ručnih testova, stoga se nazivaju i alatima za reprodukciju. Ovi alati zahtijevaju skriptni jezik za pokretanje određenog alata (TutorialRide, n.d).

Alati za pokrivanje pomažu u provjeri koliko je temeljito provedeno testiranje. Alat za pokrivanje prvo identificira elemente ili stavke pokrića. Na razini testiranja komponenata, stavke pokrivenosti mogu biti redovi koda ili izjave koda ili ishodi odluke. Na razini integracije komponenata, stavka pokrivenosti može biti poziv na funkciju ili modul.

Alati za ispitivanje sigurnosti mogu se koristiti za testiranje sigurnosti sustava pokušavajući ga slomiti ili hakirati. Napadi se mogu usredotočiti na mrežu, softver za podršku, aplikacijski kod ili bazu podataka. Značajke ili karakteristike alata za ispitivanje sigurnosti su: identificirati viruse, otkrivanje upada poput napada uskraćivanja usluge, simulacija razne vrste vanjskih napada, identificirati slabosti u datotekama lozinki i lozinkama.

Alati za testiranje opterećenja je softversko testiranje koje se provodi kako bi se razumjelo ponašanje aplikacije pod određenim očekivanim opterećenjem. Ispitivanje opterećenja izvodi se kako bi se utvrdilo ponašanje sustava u normalnim i u ekstremnim uvjetima.

Alati za nadzor koriste se za kontinuirano praćenje stanja sustava koji se koristi kako bi se najranije upozorilo na kvarove, nedostatke ili probleme i kako bi se poboljšali. Postoje alati za nadzor poslužitelja, mreža, baza podataka, sigurnosti, performansi, korištenja web stranica i interneta te aplikacija (TryQA, n.d).

6. Otklanjanje grešaka

Proces prepoznavanja i ispravljanja softverske pogreške poznat je pod nazivom otklanjanje pogrešaka(debugging). Počinje nakon primanja izvješća o neuspjehu, a završava nakon što se osigura da su sve greške ispravljene i da se ta ista greška opet ne pojavljuje kada se isti test izvrši. Otklanjanje pogrešaka prilično je teška faza i može postati jedan od razloga zašto softver nije na vrijeme završen. Svaki postupak otkrivanja grešaka različit je i teško je znati koliko će vremena trebati otkriti i ispraviti grešku. Ponekad možda neće biti moguće otkriti grešku ili ako ga otkrijemo, možda uopće nije izvedivo ispraviti. Sljedeći je korak identificiranje mjesta u kojem se nalazi programska

pogreška izvornog koda i konačno poduzeti korektivne mjere za uklanjanje programske pogreške (Singh,2012).

6.1 Procesi otklanjanja greški

Replikacija greške. Prvi korak u popravljanju greške je njegova replikacija. To znači ponovno stvaranje neželjenog ponašanja u kontroliranim uvjetima. Isti set ulaza treba dati pod sličnim uvjetima i onda bi trebalo proizvesti slično neočekivano ponašanje. Ako se to ostvari, greška se replicirala. U najgorim slučajevima replikacija može biti gotovo nemoguća. Neki od razloga za neuspješno repliciranje greške su: korisnik je pogrešno prijavio problem, program nije uspio zbog hardverskih problema poput prekomjerne memorije, loše mrežne povezanosti, zagušenje mreže, neraspoloživost sistemskih sabirnica, uvjeti zastoja itd. Program nije uspio zbog problema sa sistemskim softverom. Razlog također može biti korištenje druge vrste operativnog sustava, kompajlera, upravljačkih programa itd.

Razumijevanje greške. Nakon replikacije greške, možda će se stvoriti potreba da se nešto više otkrije o njoj. To znači da treba pronaći razloge njenog nastanka. Može postojati jedan ili više razloga i to je općenito aktivnost koja oduzima najviše vremena. Program za ispravljanje pogrešaka također može biti koristan za razumijevanje programa. Ako program za ispravljanje pogrešaka pregledava izjavu programa, možda će moći pokazati dinamiku ponašanja programa pomoću točke prekida. Točke prekida koriste se za pauziranje programa u bilo koje vrijeme, na svakoj točki prekida mogu se promatrati vrijednosti varijabli, sadržaji relevantnih memorijskih mjesta, registri itd. Da bi se razumjela greška, razumijevanje programa je neophodno.

Lociranje greške. Postoje dva dijela izvornog koda koja se trebaju uzeti u obzir za lociranje greški. Prvi dio izvornog koda je onaj koji uzrokuje netočno vidljivo ponašanje, a drugi dio izvornog koda zapravo je netočan. U većini se situacija oba dijela mogu preklapati, a ponekad i oba dijela mogu biti u različitim dijelovima programa.

Ispravljanje grešku i ponovno testiranje programa. Ispravljanje greške je programiranje, a ne uklanjanje pogrešaka. Nakon nužnih promjena u izvornom kodu, potrebno je testirati izvorni kod kako bi se osigurali da je ispravak obavljen na pravom mjestu. Svaka promjena može utjecati na druge dijelove koda također. Stoga je potrebna analiza utjecaja kako bi se identificirao zahvaćeni dio i taj bi dio također trebalo temeljito ponovno testirati. Ova aktivnost ponovnog testiranja se naziva regresijsko testiranje (Singh,2012).

6.2 Metode otklanjanja greški

- **Otklanjanje pogrešaka pomoću Brute Forcea** – najčešća metoda za otklanjanje pogrešaka u programu je metoda "grube sile". Popularna je jer zahtijeva malo razmišljanja i najmanje je mentalno oporezivanje metoda, ali je tneučinkovita i općenito neuspješna. Metode grube sile mogu se podijeliti u najmanje tri kategorije:
  1. Otklanjanje pogrešaka čišćenjem memorije.
  2. Otklanjanje pogrešaka prema uobičajenom prijedlogu za rasipanje ispisa po cijelom tijelu svog programa
  3. Otklanjanje pogrešaka pomoću automatiziranih alata za uklanjanje pogrešaka.

- **Otklanjanje pogrešaka pomoću indukcije** - trebalo bi biti očito da će pažljivo pretraživanje pronaći većinu pogrešaka. Poseban misaoni proces je indukcija, u kojoj se prelazi iz pojedinosti situacije u cjelini. Odnosno, započinje s tragovima (simptomi pogreške, moguće rezultate jednog ili više testnih slučajeva) i potraži

veze među tim istim tragovima.

- **Otklanjanje pogrešaka dedukcijom** - proces polazi od nekih općih teorija ili teza, koristeći procese eliminacije i usavršavanja, kako bi se došlo do zaključka.

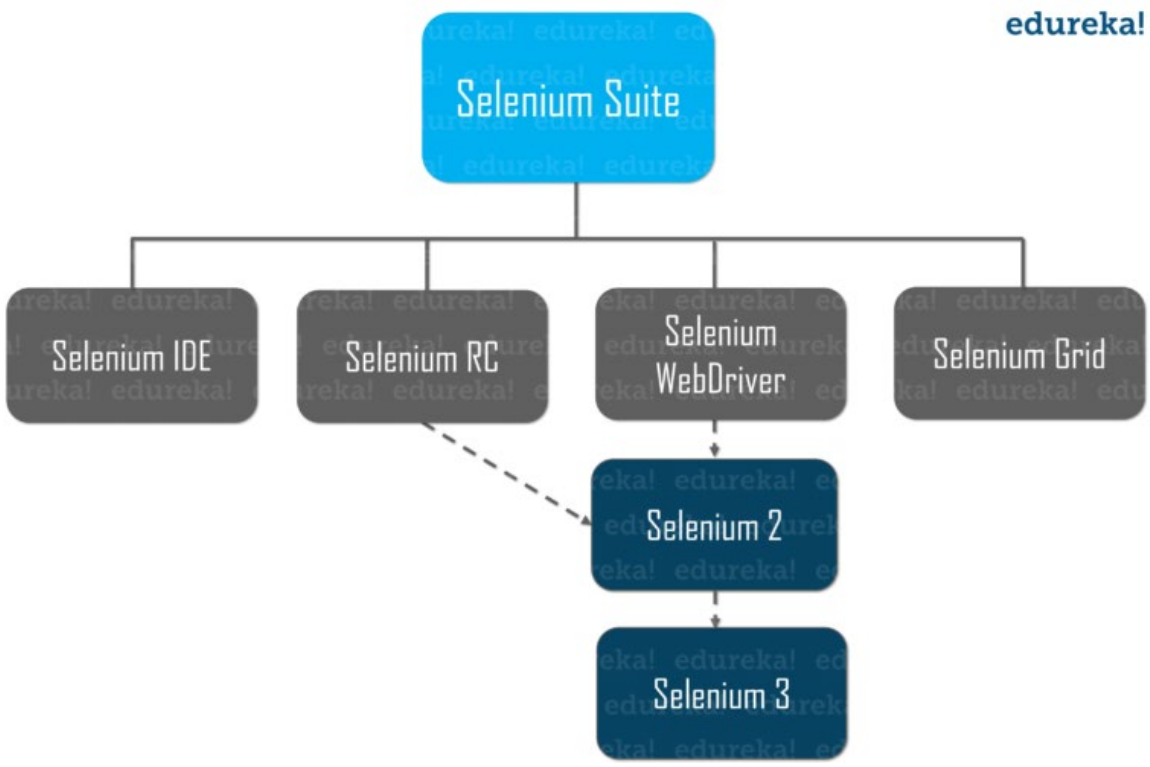
- **Otklanjanje pogrešaka povratnim praćenjem(*backtracking*)** - učinkovita metoda za pronalaženje pogrešaka u malim programima je vraćanje netočnih rezultata kroz logiku programa dok ne pronađete točku u kojoj je logika nestala. Odnosno, započinje na mjestu gdje program daje netočan rezultat. Ovim postupkom traži se mjesto u programu između točke gdje je stanje programa bilo ono što se očekivalo i prva točka u kojem stanje programa nije bilo ono što se ocečivalo.

- **Otklanjanje pogrešaka testiranjem** - posljednja metoda ispravljanja pogrešaka "razmišljanja" je upotreba test slučajeva. Dvjie su vrste test slučajeva: test slučaj za testiranje, gdje je svrha test slučajeva izložiti prethodno neotkrivenu pogrešku i test slučaj za otklanjanje pogrešaka, gdje je svrha pružiti informacije korisne za pronalaženje sumnje na pogrešku. Razlika između njih je test slučajevi za testiranje obično su veći jer se pokušavaju pokriti mnogi uvjeti u malom broju test slučajeva. S druge strane, testni slučajevi za otklanjanje pogrešaka manji su jer želite pokriti samo jedan uvjet ili nekoliko uvjeta u svakom test slučaju (Myers,2004).

7. Selenium

Selenium je alat za web testiranje otvorenog koda koji se koristi za testiranje web preglednika na različitim platformama. Izumio ga je Jason Huggins iz tvrtke ThoughtWorks uvođenjem osnovnog alata nazvanog "JavaScriptTestRunner", kako bi testirao njihovu internu primjenu vremena i troškova. Sada je stekao popularnost među ispitivačima softvera i programerima kao prijenosni okvir za testiranje automatizacije

otvorenog koda. Ima mogućnost automatizacije preglednika s određenim vezama preglednika za automatizaciju web aplikacija u svrhu testiranja. Riječ je o paketu od četiri alata dizajniranih za različite svrhe, koji je ujedno i predočen na slici 5 (Bamotra i Randhawa,2017).



Slika 5. Prikaz Selenium komponenti (edureka,n.d)

Podijeljen je u četiri komponente:

- IDE Selenium IDE, ranije poznat kao Selenium recorder, alat je koji se koristi za snimanje, uređivanje, uklanjanje pogrešaka i ponovnu reprodukciju funkcionalnih testova. Selenium IDE implementiran je kao proširenje preglednika Chrome i

dodatka u pregledniku Firefox. Pomoću dodatka Selenium IDE možete snimati i izvoziti testove u bilo kojem od podržanih programskih jezika kao što su Ruby, Java, PHP, Javascript itd.

- Selenium RC (daljinski upravljač) selenium Core bio je prvi alat u paketu alata. Međutim, zastario je jer je imalo problema u vezi s testiranjem više domena zbog pravila istog podrijetla. Dakle, da bi se prevladao taj daljinski upravljač Selenium (Selenium RC) uveden je nakon Core selenium. RC se pokazao kao pomoć u rješavanju više domena. Ali glavni problem RC-a bilo je vrijeme potrebno za provođenje testa. Kako je Selenium poslužitelj komunicirao pomoću HTTP zahtjeva, to je oduzimalo više vremena. Zbog ovog ograničenja, RC se također sada ne koristi previše.

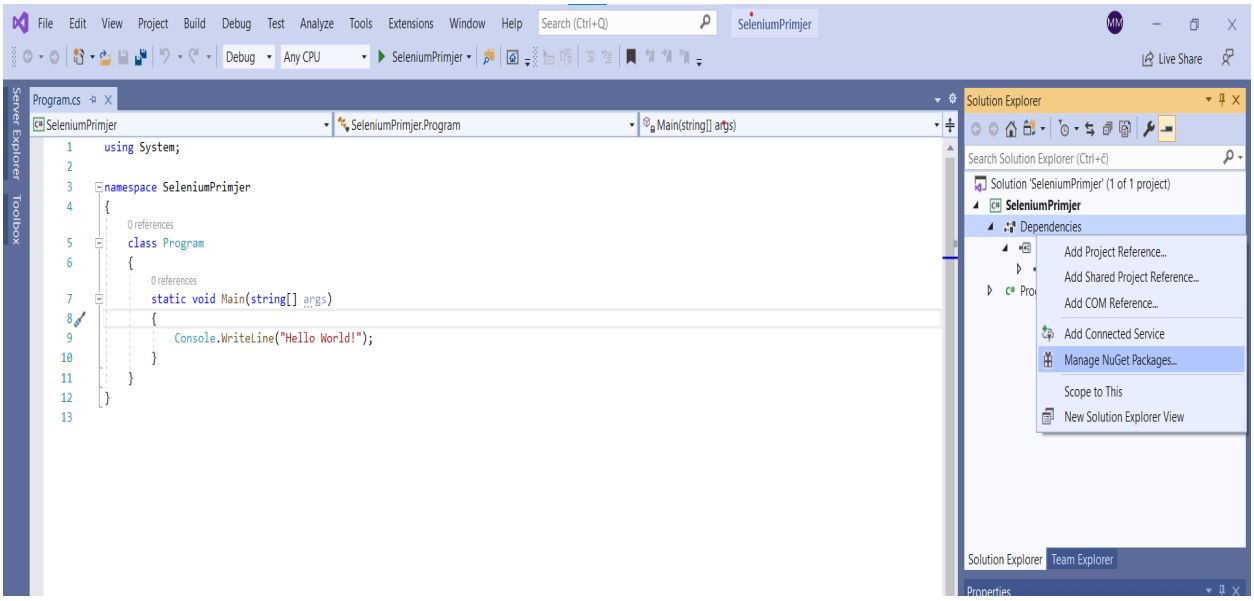
- Selenium Grid temelji se na Hub-node arhitekturi. Pomoću Selenium Grid mogu se pokretati paralelne testne sesije u različitim preglednicima. Hub kontrolira Selenium skripte koje se izvode na različitim čvorovima (određeni preglednik unutar OS-a), a test skripte koje se izvode na različitim čvorovima mogu biti napisane na bilo kojem programskom jeziku. Selenium Grid korišten je s RC za testiranje više testova na udaljenim strojevima. U današnje vrijeme, kako ljudi smatraju da je Webdriver bolji od RC-a, Grid stoga radi i s Webdriverom i s RC-om.

- Web upravljački program (Selenium web driver) je poboljšana verzija Selenium RC i najkorišteniji alat. Prihvaća naredbe putem klijentskog API-ja i šalje ih preglednicima. API Selenium Webdriver pomaže u komunikaciji između jezika i preglednika. Svaki preglednik ima različitu logiku izvršavanja radnji na pregledniku. Jednostavno rečeno, Selenium Webdriver je upravljački program specifičan za preglednik koji pomaže u pristupu i pokretanju različitih preglednika. Pruža sučelje za pisanje i pokretanje skripti za automatizaciju. Svaki preglednik ima različite upravljačke programe za pokretanje testova (Sadiq,2020).

7.1 Primjer Selenium web drivera u C#

Za potrebe primjera koristit će se Microsoft Visual Studio kao integrirano razvojno okruženje.

Prilikom ulaska u Visual studio, odaberemo naredbu da ćemo praviti *Console aplication* i Visual studio nam automatski postavi kod koji ispisuje na ekran „Hello world“. Slika 6 prikazuje taj isti kod koji dobijemo pri samom pokretanju programa te dodavanje paketa potrebnih za rad. Međutim za potrebe rada sa Selenium alatom, potrebno je uključiti naredbe u Visual studio, to se može učiniti na dva načina. Prvi je da se preuzmu odgovarajući paketi sa službene stranice https://www.selenium.dev/ ili da kliknemo desnom tipkom miša na *Dependecies* u desnom kutu i tamo odabaremo opciju *Manage NuGet Packages.*



Slika 6. Prikaz početnog izgleda koda i dodavanje paketa

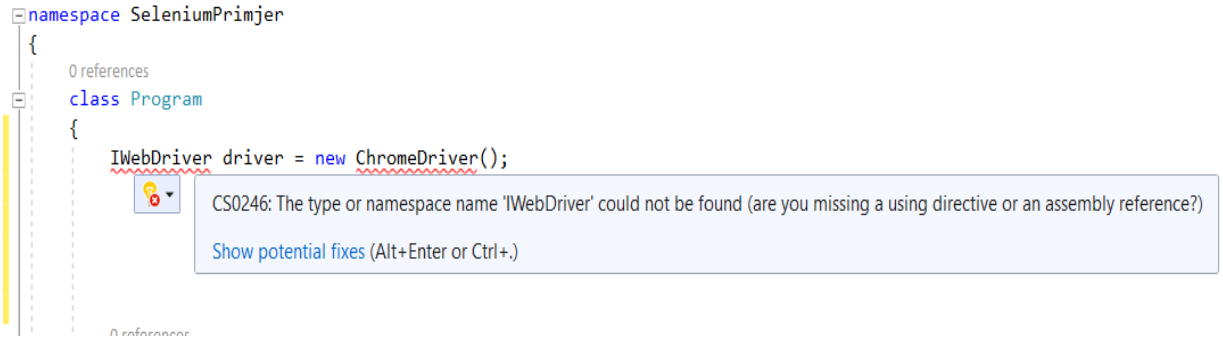
Potrebni paketi su : Selenium Web driver, Selenium WebDriverChrome driver, NUnit, Nunit.ConsoleRunner, NUnit Test Adapter i Microsoft.Net.Test.sdk

Željeni test je bio da posjetimo stranicu Fakulteta Informatike u Puli (https://fipu.unipu.hr/) i zatim koristeći Selenium testiramo njenu ispravnost. Prikaz cijelog koda vidljiv je na slici 7 ali objašnjavanje koda je podijeljeno u dijelove zbog bolje preglednosti.

```
using NUnit.Framework;
using OpenQA.Selenium;
using OpenQA.Selenium.Chrome;
using OpenQA.Selenium.DevTools.Page;
using OpenQA.Selenium.Internal;
using System;
using System.Collections.Generic;
using System.Data;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SeleniumPrimjer
{
    0 references
    class Program
    {
        IWebDriver driver = new ChromeDriver();
        0 references
        static void Main(string[] args)
        {
        }



        [SetUp]
        0 references
        public void Pokrenipreglednik()
        {
            driver.Navigate().GoToUrl("https://fipu.unipu.hr/");
            Console.WriteLine("Otvoren preglednik ");
        }
        [Test]
        0 references
        public void Izvrsitest()
        {
            IWebElement element = driver.FindElement(By.XPath("/html/body/div[3]/div[1]/div[3]/div/div/div/nav/div[2]/ul/li[5]/a[1]"));
            element.Click();
            Console.WriteLine("Test se izvršio ");
        }
        [TearDown]
        0 references
        public void Zavrsi()
        {
            driver.Close();
            Console.WriteLine("Test se uspješno izvršio ");
        }

    }
}

5 %
No issues found
```

Slika 7. Prikaz koda praktičnog primjera

IWebDriver je sučelje za testiranje koje koristimo, ovdje vidimo stvaranje novog objekta ChromeDriver. Trenutno Visual studio nam javlja grešku, jer ne može raspoznati što je jer se trenutno ne nalazi u biblioteci naredbi. Moguće ga je dodati preko ikone ispod njegovog imena ili prečacem preko kombinacija tipki na tipkovnici „ Ctrl “ i „ . “. Potrebno je odabrati „*using OpenQA.Selenium”* i greška će nestati. Prikaz instanciranja drivera i uključivanja biblioteka je prikazan na slici 8.



Slika 8. Prikaz stvaranja sučelja za testiranje i njegovog objekta

Isti slučaj je i za ChromeDriver osim što pri odabiru reference moramo odabrati „*using OpenQA.Selenium.Chrome”.*

Koristi se funkcija Pokreni preglednik koja je tipa void i nema povratnu vrijednost. Koristimo i atribut SetUp koji smo referencirali preko naredbe *„using NUnit.Framework; ”.* SetUp se postavlja prije metode koja će uvijek izvršiti prije svakog testa.
Koristi se metoda „driver.navigate“ u kombinaciji s „GoToUrl“ s kojom se otvara preglednik na određenoj stranici, koja mora biti navedena unutar zagrada s navodnicima.
Naredba „Console.WriteLine“ nam služi da kasnije u Test exploreru prikaže tijek odvijanja testa.

```
[SetUp]
0 references
public void Pokrenipreglednik()
{
    driver.Navigate().GoToUrl("https://fipu.unipu.hr/");
    Console.WriteLine("Otvoren preglednik ");
}
```

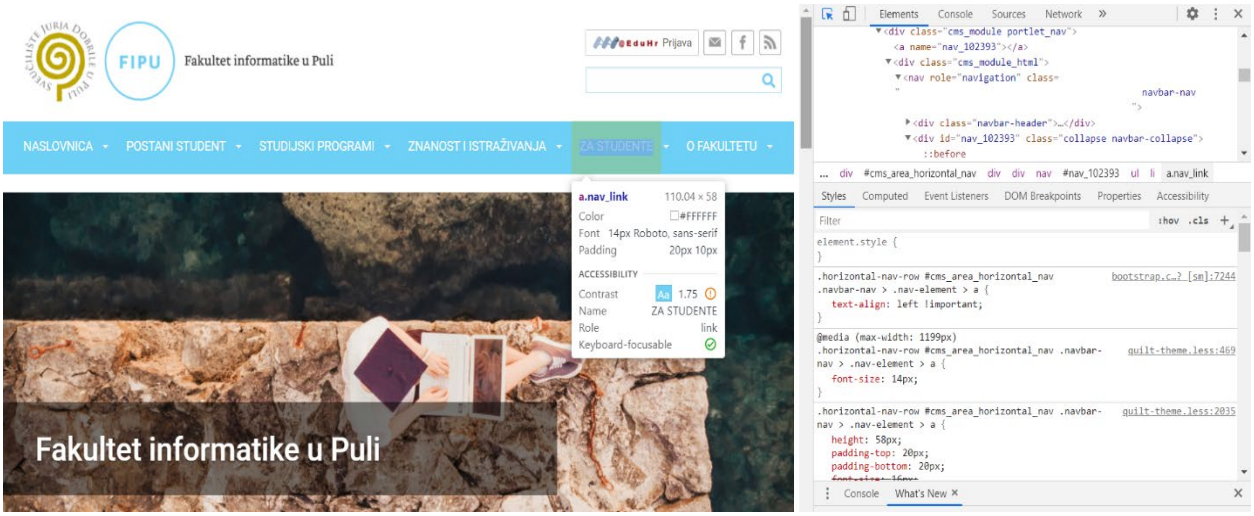
Slika 9. Prikaz otvaranja preglednika

Atribut Test koristi se neposredno prije funkcije kako bi označio metodu koja će se testirati. Metoda driver.FindElement traži na internetskoj stranici taj element, koji smo joj proslijedili preko naredbe By.; u ovom slučaju proslijeđen je parametar „XPath“, ali isto tako mogao se proslijediti „ID“, “Name“ i sl.
Naredba „element.Click“ je metoda s kojom na stranici pozivamo okidač koji će kliknuti na određeni element.

```
[Test]
0 references
public void Izvrsitest()
{
    IWebElement element = driver.FindElement(By.XPath("/html/body/div[3]/div[1]/div[3]/div/div/div/nav/div[2]/ul/li[5]/a[1]"));
    element.Click();
    Console.WriteLine("Test se izvršio ");
}
```

Slika 10. Prikaz pokretanja testa

Kada dođemo iznad određenog elementa kojem želimo pristupiti pritisnemo desnu tipku miša i odaberemo „Inspect“ otvara se dodatni dio prozora i onda možemo vidjeti HTML, CSS i Javascript elemente. Ovisno što želimo saznati ime elementa ili njegov id samo je potrebno kliknuti na strelicu u gornjem lijevom kutu i zatim stisnuti na web stranici na željeni element i označiti će nam se dio koda koji predstavlja taj element. Onda je vrlo lako saznati id, xpath ili ime klase.



Slika 11. Prikaz korištenja inspect alata u pregledniku

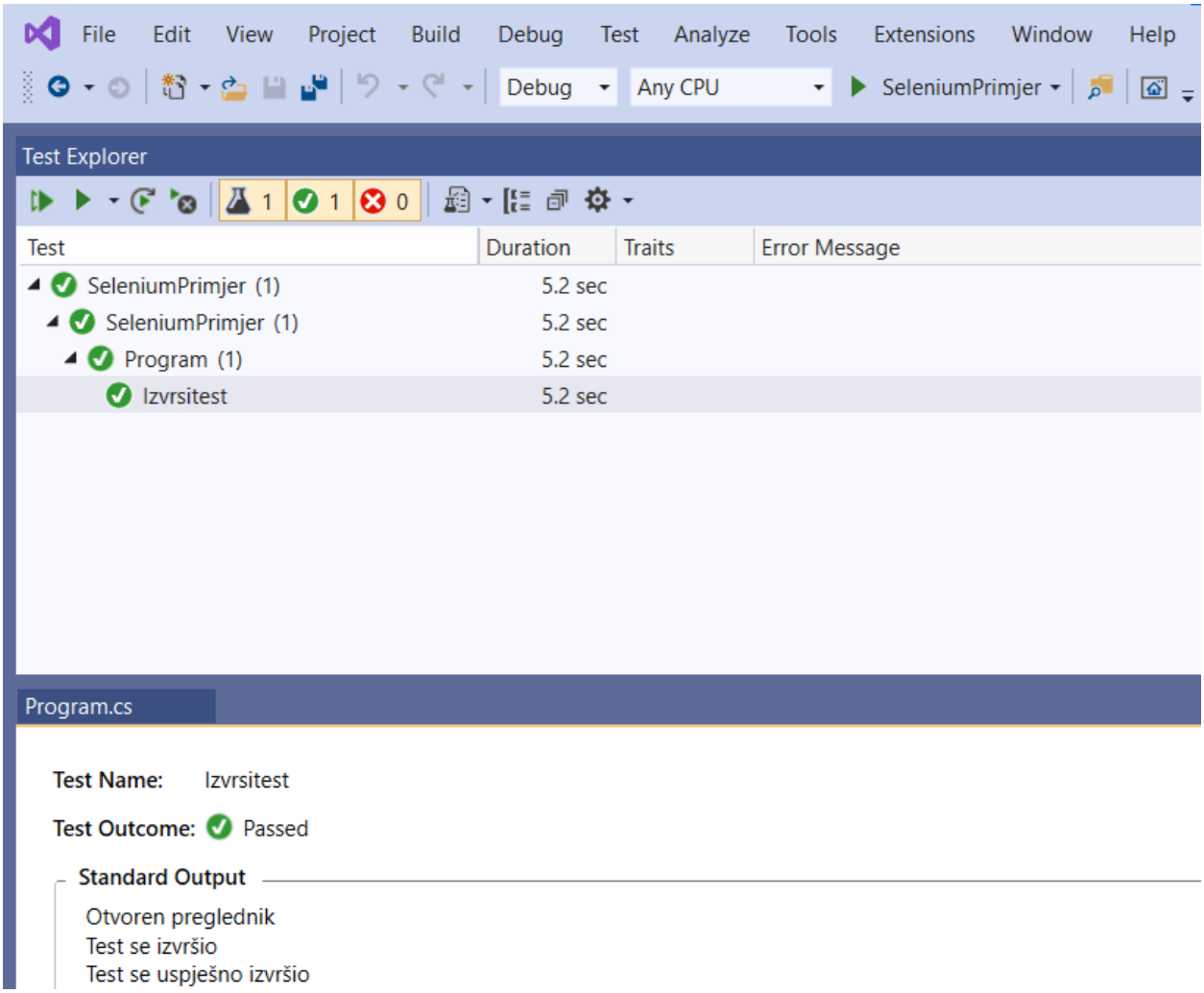
Poziva se funkcija Zavrsi koja ispred sebe ima atribut „TearDown“, on se smije samo jednom pojaviti u kodu, kod će se uspješno kompajlirati ali se testovi neće izvršiti ako se pojavi više puta. Funkcija sadrži metodu driver.Close() koja svojim pozivom zatvara preglednik.

```
[TearDown]
0 references
public void Zavrsi()
{
    driver.Close();
    Console.WriteLine("Test se uspješno izvršio ");
}
```

Slika 12. Prikaz završetka testa

Pokretanje testa se izvršava preko padajućeg izbornika pritiskom na Test>Run all tests.

Kao što se vidi na slici 13 test je uspješno proveden bez ikakvih grešaka. Vidimo rezultate u standardnom outputu koje smo zadali na nam ispiše preko naredbe console.Writeline, koja je prilično korisna kada imamo više testova koji se izvode.



Slika 13. Prikaz završenog testa

Ovakav način automatizacije preglednika može poslužiti kada imamo mnogo repetitivnih radnji. Računalo će to odraditi na brži način i na mnogo efikasniji uz manju mogućnost pogreške. Naravno uvijek je vrlo lako nadograditi testnu skriptu te promijeniti i poboljšati automatizirani proces.

7.2 Data test u Seleniumu

Cilj programa je da povuče podatke iz Excel datoteke i zatim te podatke upotrijebi na određenoj web stranici, koristi se stranica https://fipu.unipu.hr/ i te podatke će test automatizirano upisati na tražilici.

Prilikom ulaska u Visual studio kreiramo novi projekt a odabiremo predložak za konzolnu aplikaciju koristeći C#. Da bi uopće mogli pristupiti excel podatcima i raditi s njima potrebno je također uključiti EPPlus biblioteku u projekt preko *ManageNuGet Packages.* Također uključuju se i *MSTestAdapter* i *MSTestFramework.*

Za rad s podatcima potrebno je dodati excel datoteku s podatcima u projekt. Datoteku je moguće preuzeti s interneta s već gotovim podatcima ili proizvoljno sami da unesemo neke podatke. Excel datoteka se mora nalaziti u izvorišnim podatcima projekta da bi joj Visual studio mogao pristupiti. Slika 14 prikazuje podatke koji su se koristili za potrebe testa.

	A
1	Ime kolegija
2	Napredne tehnike programiranja
3	Dinamičke WEB aplikacije

Slika 14. Prikaz podataka u excel datoteci

U *Solution Exploreru* pod izbornikom *References* potrebno je pronaći excel datoteku koja je kreirana pod imenom Podatci.xlsx, zatim desnim klikom miša otvaramo izbornik i odabiremo naredbu *Include in project* kako bi joj Visual studio mogao pristupiti. Nakon toga opet ponavljamo istu radnju, desnim klikom miša odabiremo naredbu *Properties* kako bi pristupili svojstvima te datoteke i promijenili vrijednost *Copy to Output* i postavili je na *Always*.

Za dohvaćanje i podataka iz excela koristi se metoda Citajexcel, njena implementacija prikazana je na slici 15. Nakon uključivanja EPPlus biblioteke može se koristiti naredba *Excel package* s kojom smo stvorili novu instancu, a naredbu File info moramo referenirati iz biblioteke *SystemIO.* Tako otvaramo datoteku, a njezinu lokaciju preko koje joj pristupamo zapišemo unutar zagrada i navodnika.

Naredba *ExcelWorksheet* nam omogućava da unutar excel datoteke pristupamo određenom listu preko njegovog imena, u ovom slučaju ime lista je List.
Naredba int *rowCount = worksheet.Dimension.End.Row;* koristi se da bi otkrili broj redaka u datoteci, kasnije će se isti taj broj redaka koristiti u for petlji s kojom se krećemo preko datoteke.

U petlji for *(int row = 2; row <= rowCount; row++)* početak je postavljen na drugi redak jer od tamo želimo početi izvlačiti podatke. Petlja će se odvijati dok god ne dođe do zadnjeg reda koji sadrži neki podatak.

Ovdje se odvija povrat i spremanje podataka. yield return new object[] {

worksheet.Cells[row, 1].Value?.ToString().Trim()

```
public static IEnumerable <object[]> Citajexcel()
{
    using (ExcelPackage package = new ExcelPackage(new FileInfo("C:/Users/Marko/Podatci.xlsx")))
    {

        ExcelWorksheet worksheet = package.Workbook.Worksheets["List"];
        int rowCount = worksheet.Dimension.End.Row;
        for (int row = 2; row <= rowCount; row++)
        {
            yield return new object[] {
                worksheet.Cells[row, 1].Value?.ToString().Trim()
        };
            }
    }
}
```

Slika 15. Prikaz metode za dohvaćanje podataka iz excel datoteke

Pristup pregledniku jednak je kao u prošlom primjeru samo što se ovdje koristi MSTest biblioteka pa su imena atributa drukčija. [TestClass] koristi se pri definiranju klase, kao najava da će sadržavati test metode. [TestMethod] se koristi pri označavanju metode za testiranje.

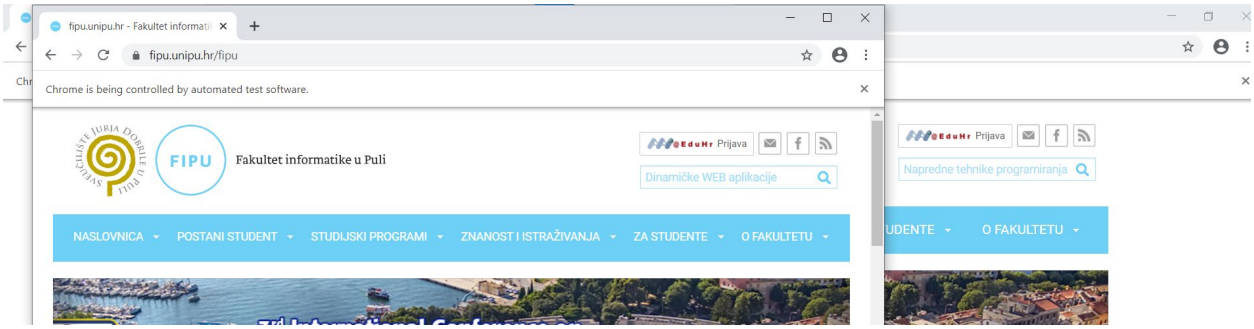
[DynamicData(nameof(Citajexcel), DynamicDataSourceType.Method)] – atribut uzima metodu za testiranje koja se zove Citajexcel.

Nakon toga pokreće se Chrome preglednik i upućuje se prema zadanoj web stranici, tamo se preko Inspect alata koji je spomenut u prethodnom primjeru pristupa određenim elementima na stranici.
Preko naredbe *SendKeys(ime);* proslijedili smo informaciju koju smo dohvatili iz excel datoteke. Detaljan prikaz koda vidljiv je na slici 16.

```
namespace DataTest
{
    [TestClass]
    0 references
    public class UnitTest1
    {

        [DynamicData(nameof(Citajexcel), DynamicDataSourceType.Method)]
        [TestMethod]
        0 references
        public void DataTestiranje(string ime)
        {

            IWebDriver driver = new ChromeDriver();
            driver.Manage().Window.Maximize();
            driver.Url = "https://fipu.unipu.hr/fipu";

            driver.FindElement(By.XPath("/html/body/div[3]/div[1]/div[2]/div[2]/div[2]/div/form/div/input")).SendKeys(ime);
            driver.FindElement(By.XPath("/html/body/div[3]/div[1]/div[2]/div[2]/div[2]/div/form/div/span/button/i")).Click();
            driver.FindElement(By.XPath("/html/body/div[3]/div[1]/div[2]/div[2]/div[2]/div/form/div/span/button")).Click();
            driver.Quit();
        }
```

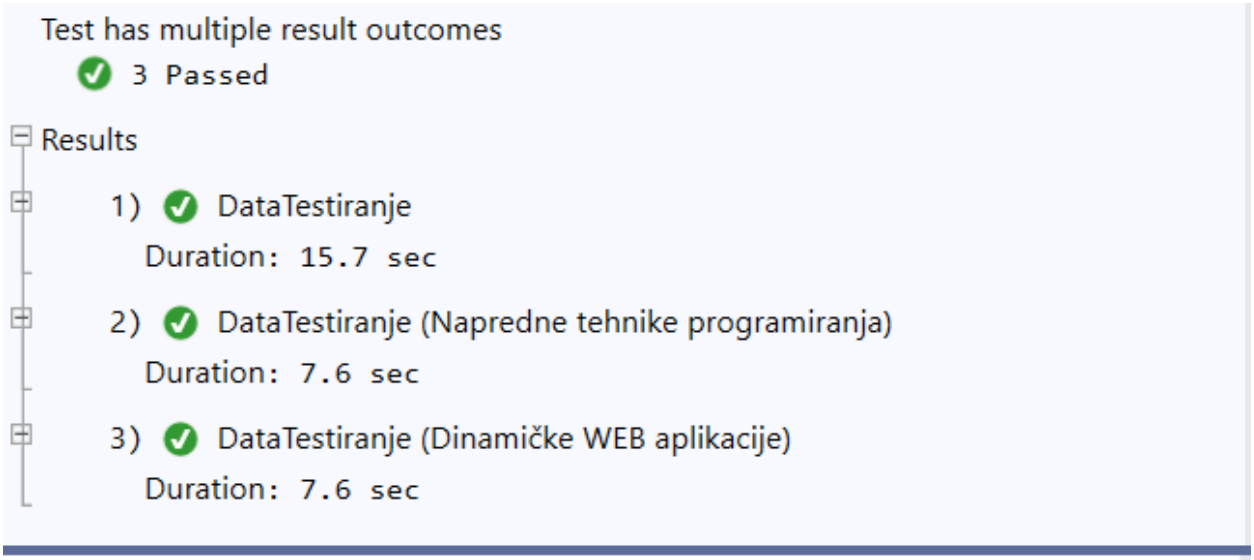
Slika 16. Prikaz koda za pristup i upravljanje stranicom

Test je trebao otvoriti web stranicu Fakulteta informatike u Puli, zatim na tražilici te iste stranice automatizirano upisati podatke koje je dohvatio iz excel datoteke. Iz slike 17 vidljivo je uspješno izvršavanje testa.



Slika 17. Prikaz završenog testa

Odvijanje testa moguće je pratiti preko Test Explorera u Visual Studiu, on na kraju i tijekom izvođenja može pružiti više podataka o uspjehu odnosno neuspjehu testa. U našem slučaju greške nisu pronađene. Slika 18 prikazuje odvijanje testa.



Slika 18. Prikaz outputa u Test Exploreru

Također je moguće zapisivati podatke u Excel datoteku, moguće ih je zapisivati izravno u kodu, ali to nebi bilo prikladno kada bi se zapisivala veća količina podataka. Željene podatke koje poželimo unijeti moguće je odabrati na stranici, koristeći *Inspect* alat odabiremo elemente gdje se nalaze podatci, detaljnije je prikazano na slici 19. Za primjer će se uzeti podatci o prognozi Hrvatske na današnji datum, te podatci o prognozi za sutrašnji datum sa stranice https://meteo.hr/.

Slika 19. Prikaz stranice s koje će se prikupljati podatci

Na slici 20 vidljiv je postupak čitanja podataka sa internetske stranice. Program će pristupiti početnoj stranici te se navigirati do željenih podataka, zatim će te podatke spremiti u tip podatka string *strText.* Zatim se opet kreće do podataka za sutrašnju prognozu i onda te podatke sprema u varijablu *strText2*.

U kodu se koristi naredba *driver.Manage().Timeouts().ImplicitWait = TimeSpan.FromSeconds(10);* koja implicitno naređuje da program "uspori" sa svojim procesom jer preglednik na vrijeme ne otvori stranicu te ne uspije pronaći traženi element.

```
[Test]
0 references
public void Test1()
{
    IWebDriver driver = new ChromeDriver();
    driver.Manage().Window.Maximize();
    driver.Navigate().GoToUrl("https://meteo.hr/");
    driver.Manage().Timeouts().ImplicitWait = TimeSpan.FromSeconds(10);
    var pronadjiRegiju = driver.FindElement(By.XPath("/html/body/header/section/div/nav/div/ul/li[4]/a"));
    pronadjiRegiju.Click();
    var hrvatskaDanas = driver.FindElement(By.XPath("/html/body/header/section/div/div/div[2]/div/div[1]/ul/li[1]/a"));
    hrvatskaDanas = driver.FindElement(By.XPath("/html/body/header/section/div/div/div[2]/div/div[1]/ul/li[1]"));
    hrvatskaDanas = driver.FindElement(By.XPath("/html/body/header/section/div/div/div[2]/div/div[1]/ul/li[1]/a"));
    hrvatskaDanas.Click();
    string strText = driver.FindElement(By.XPath("/html/body/section[4]/div/div/div[1]/div/div[2]/div[1]")).Text;


    driver.Manage().Timeouts().ImplicitWait = TimeSpan.FromSeconds(10);
    var pronadjiRegijusutra = driver.FindElement(By.XPath("/html/body/section[4]/div/div/div[2]/aside/div/ul/li[3]/a"));
    pronadjiRegijusutra.Click();
    string strText2 = driver.FindElement(By.XPath("/html/body/section[4]/div/div/div[1]/div/div[2]/div[1]")).Text;
```

Slika 20. Prikaz koda za čitanje podataka sa stranice

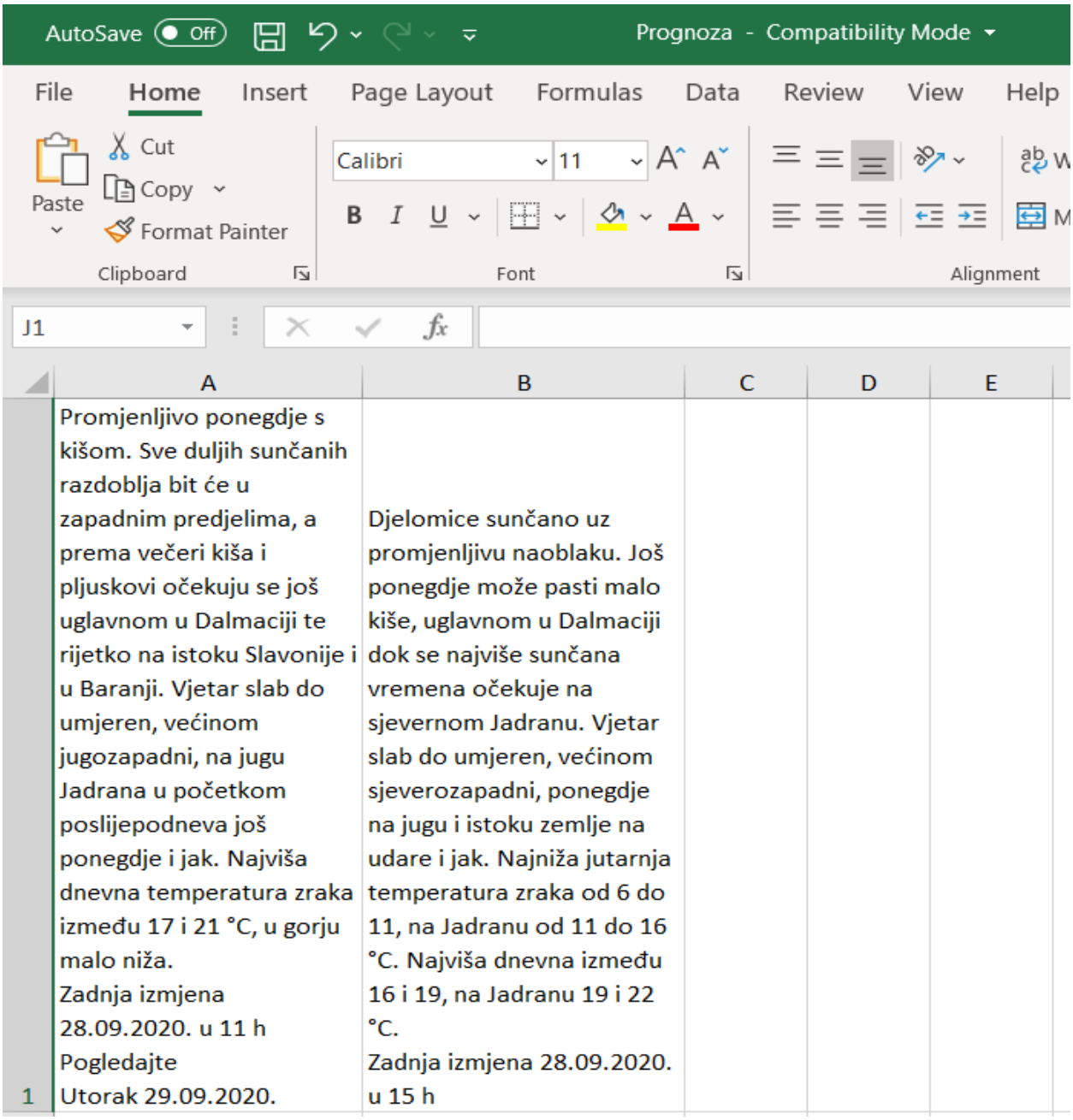
Nakon što su se podatci pročitali i spremili u varijable, potrebno ih je još samo prenijeti u Excel datoteku. Taj tijek može se promotriti na slici 21.

Naredbe excelWorksheet.Cells dio su paketa *Microsoft.Office.Interop.Excel,* koriste se za spremanje podataka preko ćelija u Excelu, odnosno prvi broj u zagradama predstavlja redak a drugi stupac. *excelApp.ActiveWorkbook* je naredba preko koje stvaramo i spremamo datoteku u određenu lokaciju na disku.

```
Excel.Application excelApp = new Excel.Application();
if (excelApp != null)
{
    Excel.Workbook excelWorkbook = excelApp.Workbooks.Add();
    Excel.Worksheet excelWorksheet = (Excel.Worksheet)excelWorkbook.Sheets.Add();



    excelWorksheet.Cells[1, 1] = strText;
    excelWorksheet.Cells[1, 2] = strText2;


    excelApp.ActiveWorkbook.SaveAs(@"C:\Users\Marko\Desktop\Zavrsni\Prognoza", Excel.XlFileFormat.xlWorkbookNormal);

    excelWorkbook.Close();
    excelApp.Quit();
```

Slika 21. Prikaz spremanja podataka u Excel datoteku

Podatci su se spremili u odredišnu datoteku koju smo odabrali i specificirali. Način na koji su spremljeni vidljiv je na slici 22.



Slika 22. Prikaz spremljenih podataka u Excelu

Iz prethodnih primjera s podatcima u Seleniumu vidljivo je da uvelike mogu olakšati posao svakodnevnih autonomnih zadataka. Rad s velikom količinom podataka je iscrpna aktivnost koja ako se može automatizirati olakšala bi i skratila utrošeno vrijeme i resurse organizacije.

Na primjeru testa s podatcima, gdje se koriste podatci iz excel datoteke moguće je dodati generičke podatke u excel, koji bi se mogli koristiti u testiranju nove web stranice kada je potrebno uvijek ispočetka ispunjavati forme te testirati ispravnost i kompatibilnost stranice.

Iz primjera spremanja podataka u excel vidljive su samo male mogućnosti koje se s tim mogu ostvariti. Moguće je spremanje velikog broja podataka s cijele stranice, pretraga određenih podataka koji su nam od važnosti. *Web scraping* ili struganje po internetu je aktivnost kojom se bave mnoge organizacije, ali važno je naglasiti da mnogo stranica to ne dozvoljava te prečesto ili zlonamjerno struganje može dovesti do zabrane pristupu stranici ili aplikaciji.

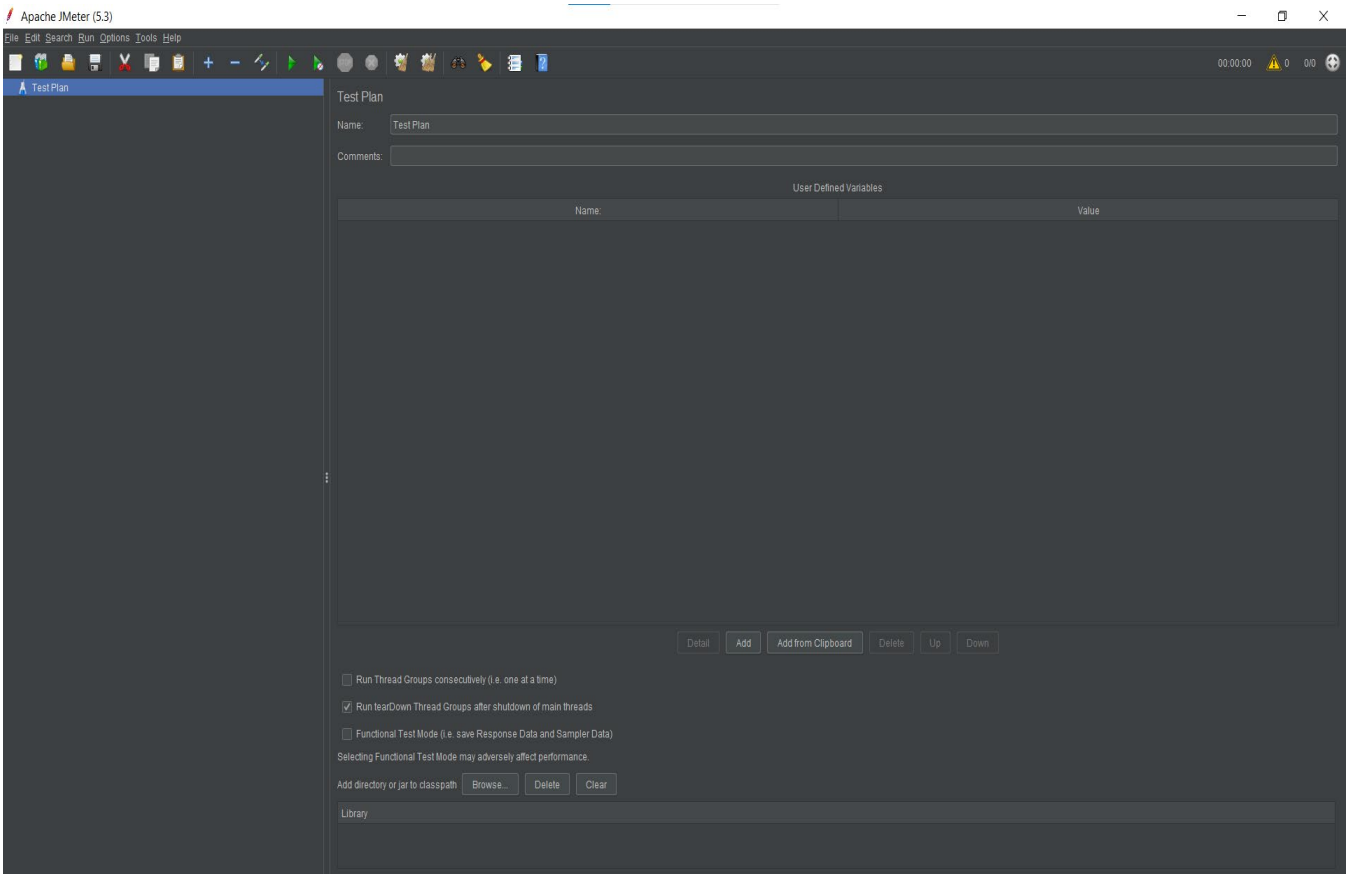
8. Testiranje opterećenja koristeći Apache JMeter

Aplikacija Apache JMeter je softver otvorenog koda, Java aplikacija dizajnirana za učitavanje test funkcionalnog ponašanja i mjerenje performansi. Izvorno je dizajniran za testiranje web aplikacija, ali se od tada proširio na druge funkcije testiranja.

Najbitnije prednosti Jmetera koje se često navode i zbog čega je postao jako popularan i rasprostranjen su:

- Licenca otvorenog koda: JMeter je potpuno besplatan, omogućava programeru upotrebu izvornog koda za razvoj.
- Prijateljski GUI: JMeter je izuzetno jednostavan za upotrebu i ne treba vremena da se s njim krene služiti.
- Neovisan je o platformi: JMeter je Java desktop aplikacija. Dakle, može se izvoditi na više platformi.
- Vizualizacija testa: Rezultati testa mogu se prikazati u drugom formatu, kao što su grafikon, tablica, stablo i datoteka dnevnika.
- Višestruka strategija testiranja: JMeter podržava mnoge strategije testiranja kao što su ispitivanje opterećenja, distribuirano testiranje i funkcionalno testiranje.
- Simulacija: JMeter može simulirati više korisnika istodobnim nitima, stvoriti veliko opterećenje protiv web aplikacije koja se testira
- Podržava više protokola: JMeter ne podržava samo testiranje web aplikacija već i procjenjuje performanse poslužitelja baze podataka. Svi osnovni protokoli kao što su HTTP, JDBC, LDAP, SOAP, JMS i FTP podržani su od strane JMeter
- Snimanje i reprodukcija – Moguće je snimiti korisničku aktivnost u pregledniku i simulirati ih u web aplikaciji pomoću JMetera

Prije samog preuzimanja i korištenja JMetera potrebno je provjeriti imamo li instaliranu Javu, u slučaju da nemamo potrebno je preuzeti zadnju verziju. JMeter je besplatan za korištenje i potrebno je samo preuzeti datoteku s interneta i nakon toga pokrenuti exe. datoteku *ApacheJMeter* i otvara se sučelje JMetera. Slika 23 prikazuje kako izgleda prozor prilikom prvog pokretanja.



Slika 23. Prikaz izgleda JMetera prilikom prvog korištenja

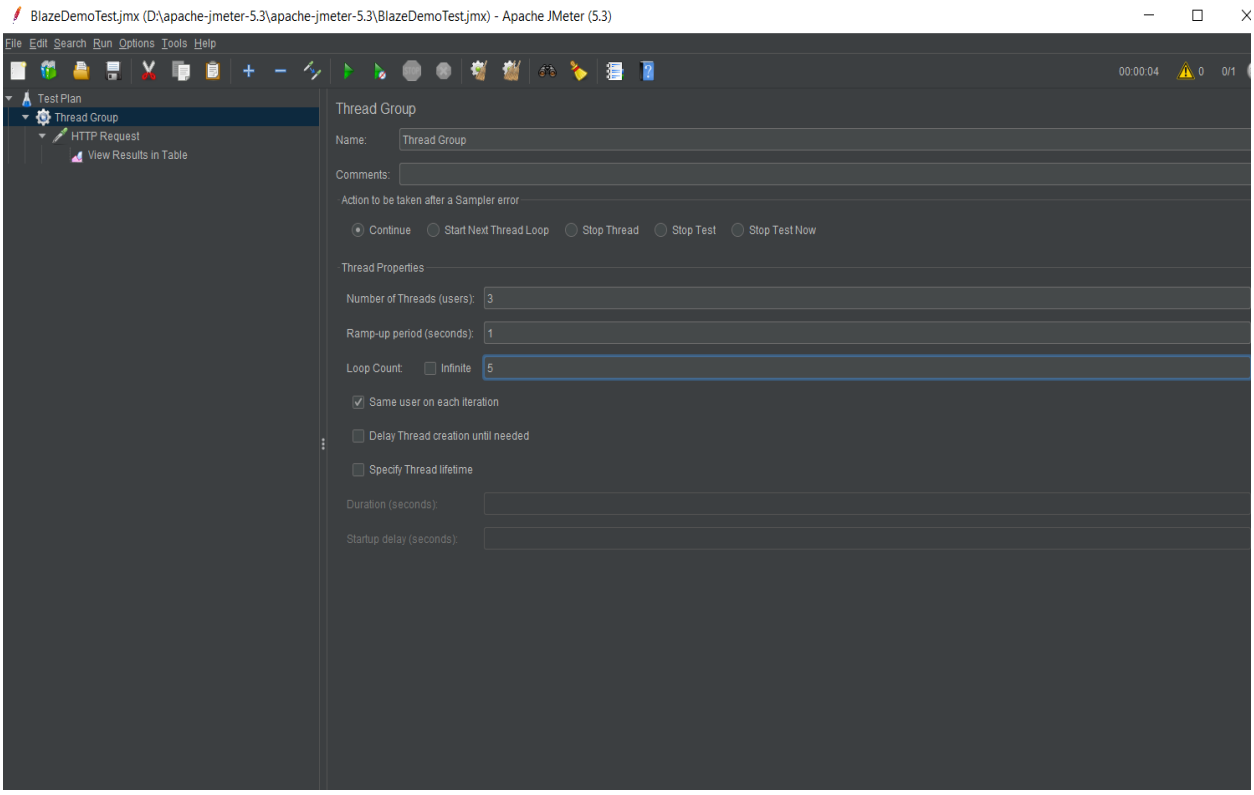
Ispitivanje opterećenja daje povjerenje u sustav i njegovu pouzdanost i performanse. Cilj testiranja je ispitati koliko određena internetska stranica može podnijeti protoka prometa kroz određeni broj korisnika.

Važno je naglasiti da nije dozvoljeno testirati bilo koju stranicu jer se to može krivo protumačiti kao DDoS (D*istributed Denial of Service*) napad.

Na internetu postoji mnogo stranica koje su stvorene upravo zbog svrhe testiranja. A koristiti će se https://blazedemo.com/

Početak pisanja testa započinjemo s dodavanjem *Thread Group,*to ostvarujemo desnim klikom miša na *Test Plan >Add>Threads(Users)>ThreadGroup*. Najbolji opis za *Thread Group* bio kao set instrukcija koje će se odvijati.

Rezultat toga je otvaranje novog sučelja vidljivog na slici 24.



Slika 24. Prikaz Thread Group prozora

Postoji mnogo opcija ali ništa nije promijenjeno osim *Number of Threads* koji smo postavili na iznos od 10,to znači da će JMeter simulirati da 10 korisnika pristupa stranici.

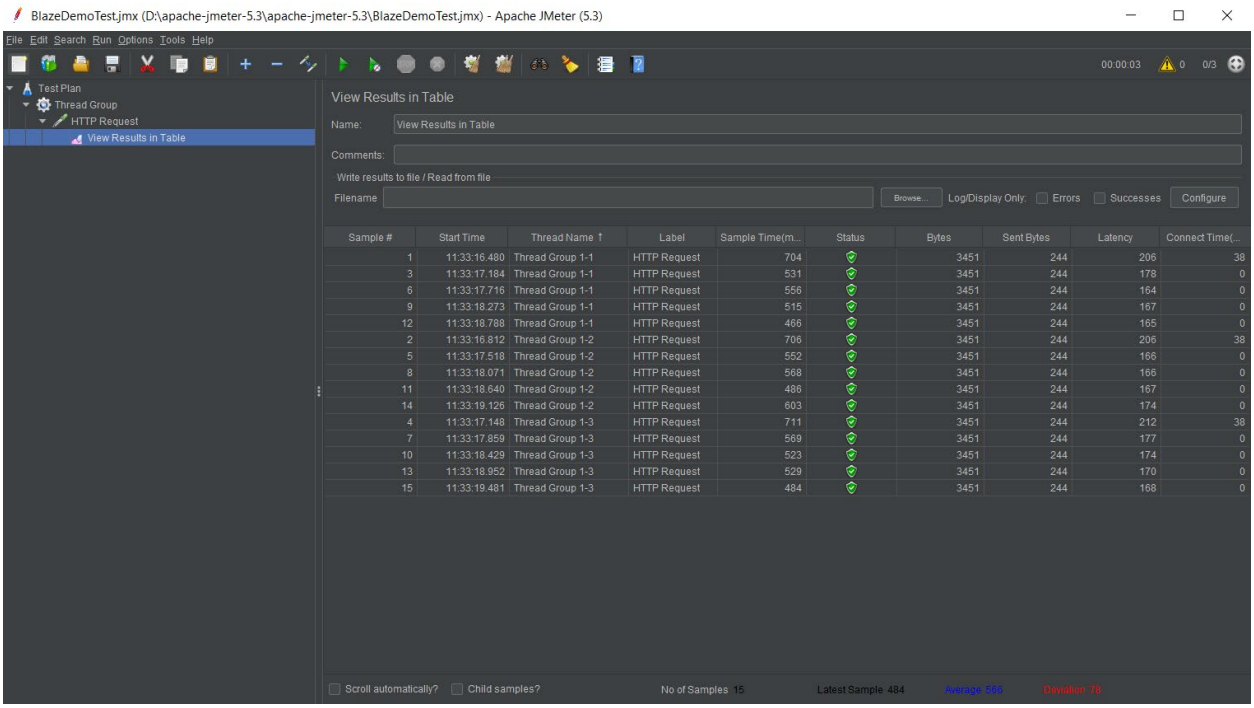
Također je postavljena vrijednost od 3 pod *Loop Count*, što znači da će svaki od tih 10 korisnika posjetiti stranicu 3 puta.

Slijedeći korak je dodavanje HTTP Requesta, koraci su desni klik miša na *Thread Group>Add>Sampler>HTTP Request.* Tamo je potrebno napisati internetsku adresu stranice koja će se testirati.

Sampler naređuje JMeteru da pošalje zahtjeve na poslužitelj i čeka odgovor. Obrađuju se redom kojim se pojavljuju na stablu.

Zatim slijedi dodavanje Listenera, koraci su desni klik miša na HTTP Request>Add>Listener>View Results in Table.

Listeneri služe za grafički prikaz rezultata testa, postoji mnogo mogućnosti za prikazivanje. Slika 25 sadrži rezultate testa.



Slika 25. Prikaz rezultata testa u tablici

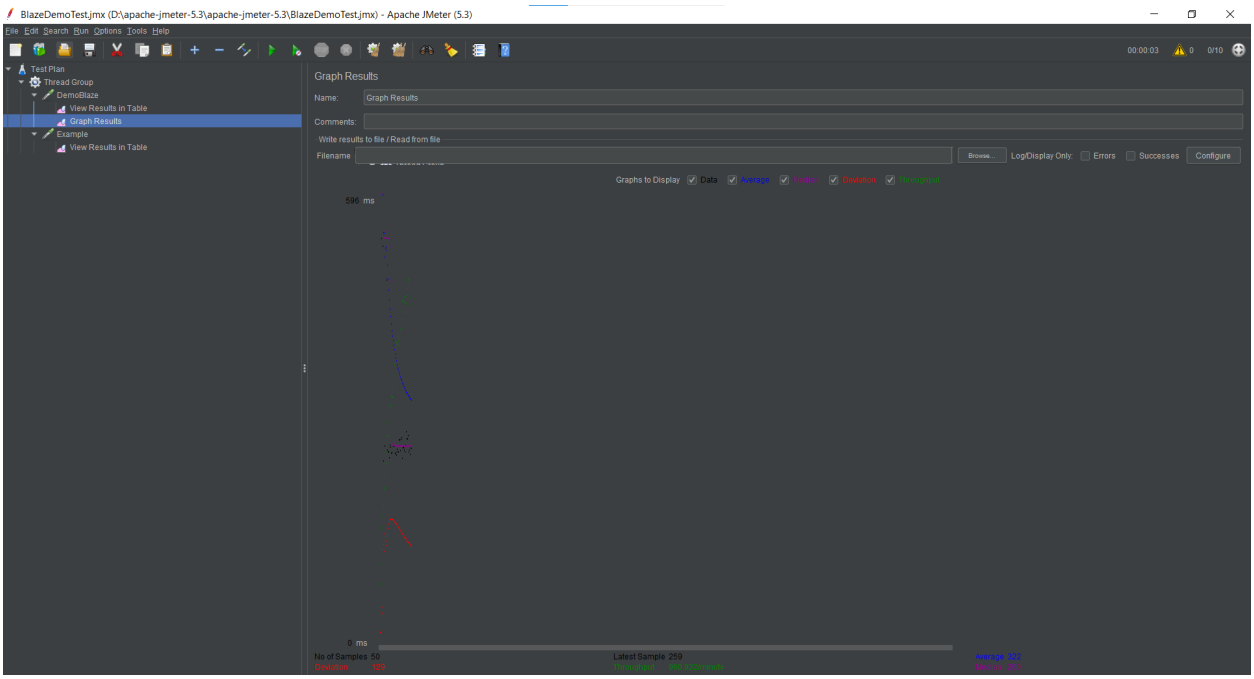
Moguće je dodati više stranica i također više načina prikaza rezultata.

Na slici 26 rezultati su prikazani pomoću grafa, razlikuju se više vrijednosti.

Propusnost (*throughput*): je broj zahtjeva po jedinici vremena koji se šalju poslužitelju tijekom testa.

Propusnost je stvarno opterećenje koje obrađuje poslužitelj tijekom izvođenja, ali ne govori ništa o performansama poslužitelja tijekom istog izvođenja. Vrijeme odziva govori koliko brzo poslužitelj obrađuje određeno opterećenje.

Prosjek (*average*): ovo je prosjek vremena odziva ukupnih uzoraka.
Medijan: ovo je središnja točka distribucije frekvencije.
Odstupanje (*deviation*): Standardno odstupanje mjeri srednju udaljenost vrijednosti do njihovog prosjeka. Daje dobru ideju o disperziji ili varijabilnosti mjera do njihove srednje vrijednosti.

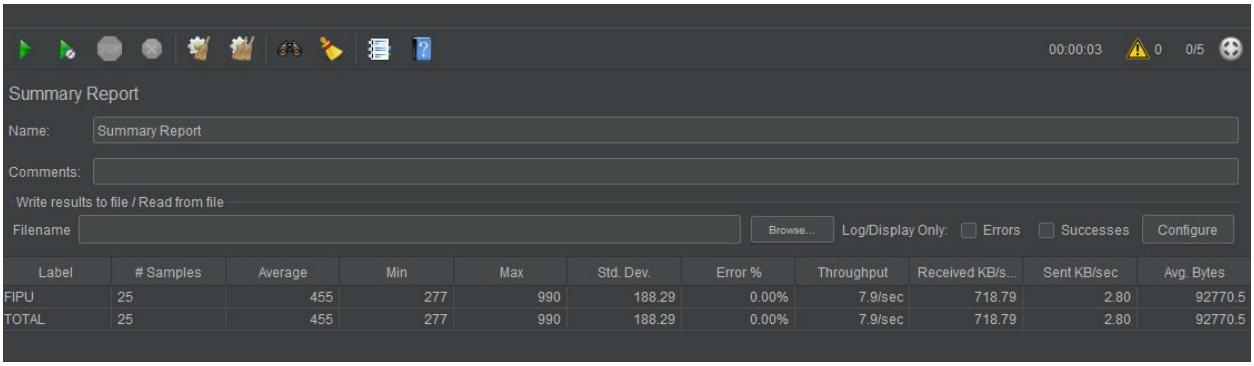


Slika 26.Prikaz rezultata testa koristeći graf

Moguće je dodati *Summary report* iz kojeg se mogu pročitati detaljni rezultati testiranja.

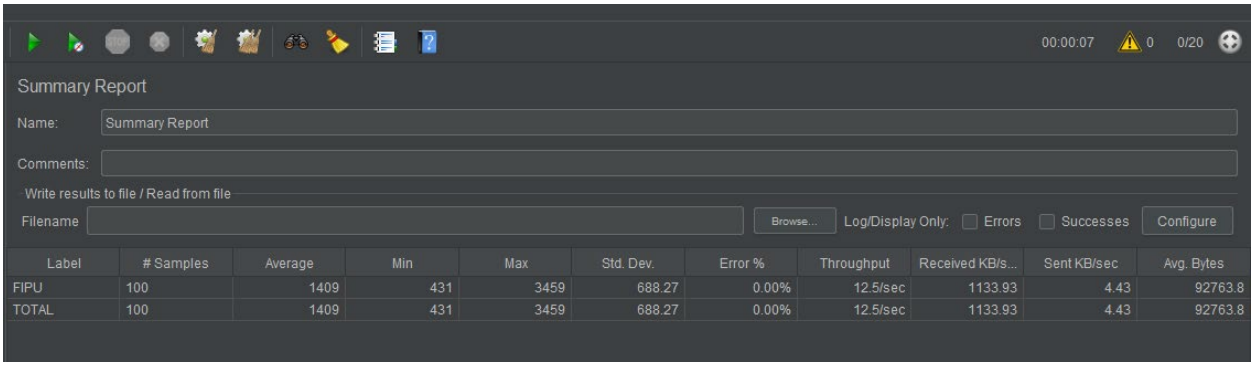
Korak za dodavanje je desni klik miša *ThreadGroup>Add>Listener>Summary report.*

Broj korisnika postavljen je na 5 i svaki od tih korisnika će 5 puta posjetiti stranicu, što nam daje ukupan rezultat od 25 uzoraka, kao što je vidljivo na slici 27.



Slika 27. Prikaz Summary reporta

Kada se poveća broj korisnika vidljiva je promjena u rezultatima. Povećalo se prosječno vrijeme odziva na zahtjev. Propusnost je također isto povećana. Iz slike 28. može se zaključiti da ako povećamo broj uzoraka, mogu se očekivati lošiji rezultati, odnosno sporija obrada.



Slika 28. Prikaz Summary reporta s povećanim brojem korisnika

9. Zaključak

Testiranje softvera smatra se kao najbitnija aktivnost u razvoju kvalitetnog softvera, potrebno je uključiti ga od samog početka razvoja softvera kako bi postigao što bolje efekte i donio uspješnije rezultate, iako dug i skup proces potrebno ga je izvršiti na detaljan i ispravan način jer je od presudne važnosti za stabilno okruženje.

U ovom završnom radu objašnjeni su ključni segmenti testiranja softvera. Pojašnjena je važnost verifikacije i validacije, također s naglaskom na povećanju pozornost prilikom ispunjavanja zahtjeva. Greške u softveru su praktički nezaobilazne, važno je samo prepoznati, locirati i ispraviti te iste greške u što kraćem roku. Pojašnjene su vrste testiranja softvera kroz njegove primjene i ciljeve, također uz moguće pojave problema koje se mogu pojaviti u tijeku. Otklanjanje grešaka konkretno je jedan od najvećih faktora u cijelom testiranju, sam za sebe sveobuhvatan proces koji se sastoji od repliciranja, razumijevanja i lociranja te na kraju i samog ispravljanja greške. Ponuđene su metode za otklanjanje greške poput metoda grube sile, indukcije, dedukcije, backtrackingom i testiranjem. Navedeni su alati za testiranje koje su podijeljeni po kategorijama kao što su alati za korisničko sučelje, alati za učitavanje i izvedbu.

U radu je objašnjen automatski alat za testiranje Selenium, također prikazan je praktičan primjer njegove komponente *Selenium Webdriver*, točnije prikazan je na primjeru automatiziranog internetskog preglednika koji izvršava naredbe preko test skripte koja je napisana preko programskog jezika C#. Također dan je primjer data testa koji uzima podatke iz Excel datoteke i s njima upravlja na pregledniku, isto tako prikazano je „struganje“ po internetskoj stranici te spremanje tih podatka u ovom slučaju u Excel datoteku. Osim toga ponuđen je primjer kako se aplikacija JMeter koristi za testiranje opterećenosti internetske stranice.

10. Literatura

1. Manger,R (2005), Softversko Inženjerstvo-skripta.Prvo izdanje.Zagreb.
2. Myers,J (2004), The Art of Software Testing, Drugo izdanje
3. Galin,D (2004), Software Quality Assurance.
4. Bamotra i Randhawa (2017), Software testing techniques. (http://ijicse.in/index.php/ijicse/article/view/115)
5. Kaur i Gupta (2013), Comparative Study of Automated Testing Tools: Selenium, Quick Test and Professional Testcomplete (http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.448.6743&rep=rep1&type=pdf)
6. Singh i Singh (2019), Software Testing
7. Desai i Srivastava (2016), Software testing: A practical approach, Drugo izdanje
8. Pan,J (1999), Software testing (http://www.sci.brooklyn.cuny.edu/~sklar/teaching/s08/cis20.2/papers/software-testing.pdf)
9. Boštjančić, Stojanović, Nedeljković(2007), Metodologija testiranja softverskog proizvoda (http://2007.telfor.rs/files/radovi/09_03.pdf)
10. Sharma,R.M (2014) Quantitative Analysis of Automation and Manual Testing(https://www.imvcportal.com.au/uploads/study_material/1504593504IJEIT1412201407$%23@_46.pdf)
11. Singh,Y (2012), Software testing
12. What is difference between Defect, Bug , Error(https://medium.com/swlh/what-is-difference-between-defect-bug-error-b477e76b5502)
13. Error, Defect, and Failure(https://www.toolsqa.com/software-testing/istqb/error-defect-failure/)
14. Types of Software testing (https://www.edureka.co/blog/types-of-software-testing/)
15. Životni ciklus testiranja softvera (https://www.guru99.com/software-testing-life-cycle.html)

16. Životni ciklus testiranja softvera (https://www.guru99.com/software-testing-life-cycle.html)
17. Alati za testiranje (http://tryqa.com/what-are-the-different-types-of-software-testing-tools/)
18. Alati za testiranje (https://www.tutorialride.com/software-testing/tools-for-test-execution-and-logging.htm)
19. Sadiq,S (2020) (https://hackr.io/blog/what-is-selenium)
20. Vaidya,N (2020) All You Need to Know About Selenium WebDriver Architecture (https://www.edureka.co/blog/selenium-webdriver-architecture/)
21. Selenium Automation with C# (https://www.youtube.com/playlist?list=PL6tu16kXT9PqKSouJUV6sRVgmcKs-VCqo)
22. Apache JMeter Korisnički priručnik (https://jmeter.apache.org/usermanual/)

11. Popis slika