SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

**Mijo Lučić**

PRAKSE TESTIRANJA PROGRAMSKIH PROIZVODA

ZAVRŠNI rad

Varaždin, 2019.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Mijo Lučić

Matični broj: 42281/13–I

Studij: Informacijski sustavi

PRAKSE TESTIRANJA PROGRAMSKIH PROIZVODA

ZAVRŠNI RAD

Mentor:

Doc. dr. sc. Zlatko Stapić

Varaždin, lipanj 2019.

*Mijo Lučić*

Izjava o izvornosti

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

*Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi*

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Sažetak

U ovom radu sam obradio aktuelne prakse, metode, tehnike i alate testiranja programskih proizvoda, pritom fokusirajući se na web i nativne aplikacije. U istraživačkom dijelu rada obrađujem korištenje odabranih praksi testiranja na web aplikaciji multimedijalnog tipa, koja je rađena uz pomoć WebRTC biblioteka. Web aplikacija koju sam izradio omogućuje korisnicima korištenje multimedijalnih mogućnosti preglednika u realnom vremenu na više uređaja, kao što je korištenje web kamere, mikrofona, zvučnika i drugih vanjskih ulaznih i izlaznih uređaja. Primjer jednog slučaja korištenja jeste video dijeljenje. Takva vrsta aplikacije treba imati snažnu podršku raznih preglednika u različitim uvjetima i tehničkih karakteristika. Iz tih razloga, veoma je važno upotrijebiti što više praksi testiranja web aplikacije.

Ključne riječi: razvoj; testiranje; web aplikacije; WebRTC, automatsko testiranje;

Sadržaj

[Sadržaj iii](#_Toc11100644)

[1. Uvod 1](#_Toc11100645)

[2. Metode i tehnike rada 2](#_Toc11100646)

[3. Osnove testiranja programskih proizvoda 3](#_Toc11100647)

[3.1. Ciljevi i načela testiranja 3](#_Toc11100648)

[3.2. Verifikacija i validacija 4](#_Toc11100649)

[3.3. Životni ciklus programskih proizvoda 5](#_Toc11100650)

[3.3.1. Životni ciklus testiranja programskih proizvoda 5](#_Toc11100651)

[3.4. Razine testiranja 6](#_Toc11100652)

[3.5. Metode i tipovi testiranja 7](#_Toc11100653)

[3.6. Modeli suvremenog testiranja softvera 7](#_Toc11100654)

[3.6.1. Vodopadni model 7](#_Toc11100655)

[3.6.2. Agilni model 7](#_Toc11100656)

[3.6.3. DevOps model 7](#_Toc11100657)

[3.7. Uloge u testiranju 7](#_Toc11100658)

[3.8. Test-driven development 9](#_Toc11100659)

[3.9. Ad-hoc testiranje 9](#_Toc11100660)

[3.10. Testiranje performansi 10](#_Toc11100661)

[3.11. Jedinično testiranje 11](#_Toc11100662)

[3.11.1. Dobre prakse u jediničnom testiranju 13](#_Toc11100663)

[3.11.2. Alati za kreiranje jediničnih testova 14](#_Toc11100664)

[3.12. Integracijsko testiranje 14](#_Toc11100665)

[3.12.1. Pristup odozdo prema gore 14](#_Toc11100666)

[3.13. Testiranje web servisa 15](#_Toc11100667)

[3.13.1. Postman 15](#_Toc11100668)

[3.13.2. Telerik Fiddler 16](#_Toc11100669)

[3.13.3. SoapUI 16](#_Toc11100670)

[3.14. Testiranje korisničkog sučelja 17](#_Toc11100671)

[3.14.1. Dimni testovi 17](#_Toc11100672)

[3.14.2. Alati za UI testiranje 19](#_Toc11100673)

[4. Testiranje u kontekstu kontinuirane integracije, dostave i objave 20](#_Toc11100674)

[4.1. Continuous integration 20](#_Toc11100675)

[4.2. Continuous delivery 20](#_Toc11100676)

[4.3. Continuous deployment 21](#_Toc11100677)

[4.4. CI/CD aplikacije putem Jenkins-a 21](#_Toc11100678)

[5. Testiranje u kontekstu osiguranja kvalitete programskih proizvoda 22](#_Toc11100679)

[5.1. DevOps 23](#_Toc11100680)

[5.2. Standardizacija i certifikacija testiranja softvera 23](#_Toc11100681)

[5.3. Normativne reference testiranja u softverskom inženjerstvu 24](#_Toc11100682)

[6. Implementacija i testiranje WebRTC aplikacije 25](#_Toc11100683)

[6.1. Arhitektura WebRTC okvira 25](#_Toc11100684)

[6.2. Pribavljanje korisničkih medija 26](#_Toc11100685)

[6.3. Testiranje WebRTC aplikacije 27](#_Toc11100686)

[6.3.1. Podrška na web preglednicima 27](#_Toc11100687)

[6.4. Rezultati testiranja multimedijalne aplikacije 27](#_Toc11100688)

[6.5. Testiranje usklađenosti 28](#_Toc11100689)

[6.6. Kontinuirana integracija i dostava 28](#_Toc11100690)

[6.7. Testiranje pomoću alata Kite 28](#_Toc11100691)

[6.8. Automatsko UI testiranje pomoću Selenium-a 29](#_Toc11100692)

[7. Zaključak 30](#_Toc11100693)

[Popis literature 31](#_Toc11100694)

[Popis slika 33](#_Toc11100695)

[Popis tablica 34](#_Toc11100696)

1. Uvod

Svjedoci smo vremena kada su programski proizvodi dio naših svakodnevnih radnji u mnogim aspketima života. Kvaliteta programskog proizvoda u modernom razvoju programskih proizvoda ne može bez sustavnog testiranja. Osiguranje kvaliteta ne može bez testiranja. Projektni rizici često su preveliki da bi se dozvolio rizik štednje na aspektu razvoja, kao što je testiranje. Svjedoci smo kako ova grana unutar IT industrije raste pogotovo ogledajući se kroz potražnju na burzi rada u najrazvijenijim državama. Sama ponuda ne prati potražnju za testerima.

Osnovni cilj projekata jesu zadovoljni korisnici. Voditeljima projekata često se čini kako su klijenti zlonamjerni i neprecizni u svojim željama. Međutim, često su klijenti mnogo oštriji nakon isporuke proizvoda kada naiđu na nekvalitetan proizvod u vidu očitih grešaka. Iz tog aspketa, ulaganje u testiranje programskih proizvoda još u fazi dizajniranja, je neophodno. Pritom imajući u vidu da je svrha testiranja otkriti i ukloniti što više mogućih pogrešaka. U ovom radu tematiziram sve aktuelne i relevantne prakse, metode i tehnike testiranja programskih proizvoda.

1. Metode i tehnike rada

Kao osnova za pisanje završnog rada koristio sam službeni predložak za pisanje radova koje je propisao fakultet. Za citiranje i referenciranje korišten je APA standardizirani stil. Za pisanje i uređivanje teksta koristio sam Microsoft Office Word 2016. Od alata za upravljanje referencama koristio sam ugrađene funkcije Word-a i aplikacije Mendeley Desktop.

Za razvoj web aplikacije u istraživačkom dijelu rada, korištene su moderne tehnologije za izradu web aplikacija, kao što su HTML5, CSS3, Bootstrap 4, JavaScript. Za pisanje koda korišten je JetBrains WebStorm IDE. Dio istraživačkog rada jeste i korištenje sustava za verzioniranje Git na servisu GitHub na način da je kreiran privatni repozitorij. Web aplikacija je testirana na lokalnom okruženju uz pomoć XAMPP lokalnog servera. Za potrebe testiranja web aplikacije korišteni su sljedeći preglednici: Google Chrome, Mozilla Firefox, Opera te Microsoft Edge na uređajima sa operacijskim sustavima Windows 10 i Android 7.

Kada je u pitanju preovladajuća praksa pristupa testiranju, najviše sam koristio testiranje odozdo prema gore (*engl. bottom-up*) pristup, kojeg karakterizira inkrementalni pristup integracijskom testiranju, gdje se komponente najniže razine prve testiraju, zatim se testiraju komponente više razine. Od metoda testiranja najviše problematiziram agilni pristup razvoju programskih proizvoda iz razloga što je najrasprostranjeniji u ovom trenutku.

1. Osnove testiranja programskih proizvoda

Testiranje programskih proizvoda je jedinstven proces koji se koristi kako bi se ispitao kvalitet, ispitala ispravnost i potpunost razvijenog. Cilj testiranja jeste da obezbijedi što manji broj grešaka, troškove održavanja i cjelokupne cijene programskog proizvoda. Testiranje pokušava pronaći kritične dijelove softvera. Testiranje kao dio životnog ciklusa softvera, koristi se često za ispitivanje i utvrđivanje ispunjenosti zahtjeva koji su definirani od strane klijenta.

Testiranje je značajno iz razloga što značajno može prevenirati neuspjeh misije i krajnjih ciljeva programskog proizvoda.

Svjedoci smo vremena u kojem mnogi stručnjaci iz IT sektora, miješaju mnoge pojmove iz aspekta testiranja programskih proizvoda. Kod većine građanstva razumljivo je da postoji nedovoljna upućenost u pojmove. Međutim, radi se o pojmovi koji se u općoj uporabi koriste kao sigurnosni propusti. Svakako, te pojmove treba razjasniti. Neke od tih pogrešaka su nedovoljna informiranost o razlicima i suštini različitih pojmova, koji često aludiraju na pogreške. Pogreška je univerzalniji pojam koji se često odnosi na engleske pojmove *fault, bug, error, defect* itd.

Prema (Hrgarek, 2013) neki od njih su:

* Greška (*engl. error*) je pogreška koju je prouzrokovao programer prilikom pisanja koda
* Propust ili unutarnja greška (*engl. fault*) je pogreška koja dovodi do jedne ili više unutarnjih pogrešaka u softveru
* Vanjska pogreška (*engl. failure*) je pogreška koja se pojavljuje kao razlika ponašanja neispravnog i ispravnog programa.
* Nedostatak ili odstupanje (*engl. defect*) je odstupanje od očekivanog rada
  1. Ciljevi i načela testiranja

Jedan od ciljeva ovog rada jeste pojasniti načela, pricnipe i ciljeve testiranja softvera.

(Padmini, 2006) govori o osnovnim načelima testiranja navodeći

(Padmini, 2006) u svojoj knjizi govori o nekoliko činjenica kada je u pitanju programskog inženjerstva, a koje se između ostalog odnose i na testiranje. Neki od njih je tvrdnja da su najbolji programeri do 28 puta bolji nego loši programeri. Te da je 80% razvoja programskih proizvoda bazirano na intelektu, što znači da je ključ u kreativnosti ljudi. Također, tvrdi da su pogreške pri specificiranju zahtjeva, najskuplje za popravak. Uklanjanje pogrešaka je najduža faza životnog ciklusa. Te na kraju kaže kako nema jednog najboljeg pristupa pri uklanjanju pogrešaka u programima.

* 1. Verifikacija i validacija

<https://dzone.com/articles/test-verification-vs-validation-in-website-testing>

Među često korištenim pojmovima koji mnoge čine zbunjenima kada ih vide zajedno, jesu verifikacija i validacija. U ovoj industriji često su predmeti rasprave i dodatnih definiranja i pojašnjenja. Validaciju na našem jeziku možemo nazivati i provjera ispravnosti ili ocjena valjanosti, dok verifikaciju možemo nazivati ovjera.

U tablici 1 uporedit ću verifikaciju i validaciju kroz definiciju, pojašnjenje i aktivnosti za provođenje.

|  |  |  |
| --- | --- | --- |
| Kriteriji | Verifikacija | Validacija |
| Definicija | „Potvrda kroz ispitivanje i pribavljanje objektivnih dokaza da li su ispunjeni specificirani zahtjevi.“  (ISTQB, 2011, str. 57) | „Potvrda kroz ispitivanje i pribavljanje objektivnog dokaza da su ispunjeni zahtjevi za određenu primjenu ili aplikaciju.“  (ISTQB, 2011, str. 57) |
| Pojašnjenje | Proces vrednovanja softvera tijekom ili na kraju procesa razvoja, kako bi se utvrdilo zadovoljava li navedene poslovne zahtjeve. | Pitanje koje treba postaviti jeste: „Kreiramo li mi pravi proizvod? Što znači da se ispituje da li proizvod zadovoljava potrebe korisnika i da li su zahtjevi ispravno definirani. |
| Aktivnosti | Pregled, prolazak i inspekcija | Testiranje |

Tablica : Razlike između verifikacije i validacije

Verifikacija omogućava da programski proizvod zadovolji propisane zahtjeve koji su oblikovani kroz životni ciklus proizvoda. Validacija je zadovoljenje zahtjeva kupaca na kraju životnog ciklusa proizvoda. Dobra praksa testiranja je koristiti i verifikaciju i validaciju u procesu testiranja programskog proizvoda. Verifikacija je ta koja osigurava kvalitet i održavanje. U tom procesu softver mora pravilno izvršavati sve funkcije koje ne štete sustavu u smislu pogoršanja perfomansi ili okidanja dodatnih problema. Verifikacija omogućava interoperabilnost među više različitih sekcija u dokumentaciji. „Interioperabilnost je sposobnost programskog proizvoda da surađuje s jednim ili više specificiranih komponenti ili sustava“. (ISTQB, 2007, str. 29)

Verifikacija povećava troškove razvoja, ali kada se uzme u obzir cjelokupni životni ciklus proizvoda, onda ona zapravo smanjuje troškove. Ispravke pogrešaka u sustavu mogu koštati do desetinama puta više za fazu izvršavanja i održavanja nego u fazi dizajniranja. To je česti slučaj u testiranju.

* 1. Životni ciklus programskih proizvoda

Životni ciklus programskih proizvoda (*engl. Software development life cycle*) ili samo proces razvoja programskog proizvoda definira faze u kreiranju softvera. Životni ciklus varira od jednog do drugog modela ovisno o vrsti modela razvoja softvera, kao što su vodopadni model, spiralni model, iterativni i inkrementalni razvoj i agilni razvoj poznat i pod nazivima *Scrum* i ekstremno programiranje (*engl. Extreme programming*). (Bazzana, 2015)

Modeli se razvijaju sa vremenom. S obzirom na trendove u programskom inženjerstvu, neki modeli koriste se u većoj mjeri zbog toga što se stalno povećavaju zahtjevi za sve većim brojem naprednih sustava u kojem testiranje dobija sve važniju ulogu. Iz tih razloga, modeli koji se više baziraju na ranom otkrivanju pogrešaka imaju prednost u modernom razvoju softvera. Najaktuelniji model je agilni razvoj programskih proizvoda (Balakrishnan, 2017).

Kako bi primjenili dobre prakse testiranja, potrebno je najprije odabrati adekvatan model razvoja. On uvjetuje koji elementi testiranja će biti bolje odrađeni i u kojim fazama razvoja (Myers, 2004).

* + 1. Životni ciklus testiranja programskih proizvoda

Životni ciklus testiranja programskih proizvoda (*engl. Software testing life cycle*) definira različite faze testiranja softvera. Budući da je testiranje također standardizirano, kao što razvoj softvera ima životni ciklus, testiranje također ima životni ciklus.

Životni ciklus karakterizira nekoliko faza:

Slika : Životni ciklus testiranja

* 1. Razine testiranja

Većina programskih proizvoda sastoji se od tri razine/faze: korisničko sučelje ili prezentacijska razina, servisi i programska logika. Ovisno o metodologiji, tih razina može biti više od tri. Svaka od tih faza se testira zasebno. Te razine čine osnovnu piramidu testiranja.

Razine testiranja koji su standardizirani:

* Jedinično testiranje
* Integracijsko testiranje
* Sustavsko testiranje
* Testiranje prihvatljivosti

Jedinično testiranje podrazumijeva testiranje svake jedinice kao osnovne komponente u cilju određivanja ispravnosti. Integracijsko testiranje karakterizira testiranje svih povezanih komponenti od manje do veće razine apstrakcije. Cilj je da sve komponente funkcioniraju kao cjelina.

Testiranje ili provjera sustava (*engl. system testing*) je proces testiranja integriranog sustava u svrhu provjere da li cjelokupni sustav odgovara specificiranim zahtjevima. Provjerava se cjelokupni integrirani softver (Jorgensen, 2013).

Testiranje prihvaćanja je formalno testiranje programskog proizvoda na osnovu korisnikovih potreba, zahtjeva, potreba poslovnog procesa, koja se provodi da bi se utvrdilo da li sustav zadovoljava potrebe. Kriterije prihvatljivosti postavlja klijent koji donosi krajnju ocjenu. Više o pojedinim razinama testiranja u posebnim poglavljima.

* 1. Metode i tipovi testiranja

Metode testiranja koje su poznate u IT industriji su:

* Metoda crne kutije
* Metoda bijele kutije
* Metoda sive kutije
* Agilno testiranje
* Ad hoc testiranje

Tipovi testiranja prema (Software Testing Fundamentals, 2017):

* Dimno testiranje
* Funkcionalno testiranje
* Testiranje upotrebljivosti (engl. usability testing)
* Testiranje sigurnosti
* Testiranje performansi
* Regresijsko testiranje
* Testiranje usklađenosti (engl. compliance testing)
  1. Modeli suvremenog testiranja softvera

Kako testiranje softvera sve više dobija na važnosti, shodno tome softverska industrija skoro svakodnevno proizvodi nove modele testiranja. Jedna tako dinamična industrija brzo stvara i širi nove modele. Šira zajednica programskih inženjera skroz nekoliko desetljeća je uspjela definirati i široko upotrebljavati nekoliko modela koji su upotrebljivi u kontekstu suvremenih aplikacija.

* + 1. Vodopadni model
    2. Agilni model
    3. DevOps model

asfasdf

<https://leanpub.com/testingindevops>

<https://www.testingexcellence.com/testing-in-devops/>

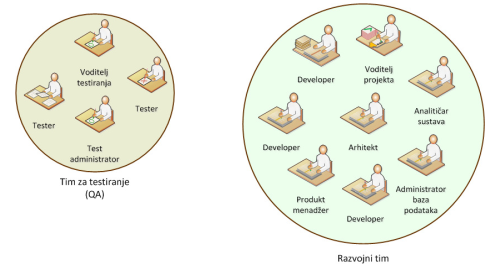
* 1. Uloge u testiranju

Ovo je glavni dio rada u kojem treba razraditi temu, pojasniti istraživanja, prikazati rezultate i slično. Poželjno je na početku poglavlja dati kratki opis strukture poglavlja, kako bi čitatelj/čitateljica rada mogao/mogla lakše pratiti složenu cjelinu.

U teoriji organiziranje uloga u testiranju možemo podjeliti na (Hrgarek, 2013):

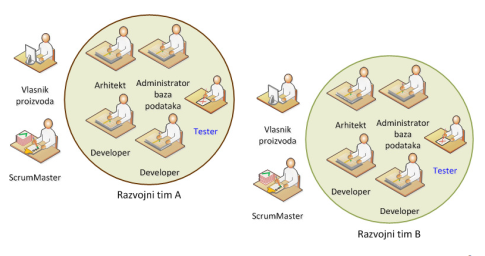
* Voditelja testiranja (engl. test manager/leader)
* Dizajner/analitičar testova (engl. test designer/analyst)
* Automatizer testova (engl. test automatization engineer)
* Administrator testova (engl. test administrator)
* Profesionalni tester (engl. software tester/quality engineer)

U ovom radu bazično ću govoriti iz perspektive dvije općenite neformalne uloge u testiranju programskih proizvoda, a to su testeri i programeri.



Slika: Klasični pristup organizacije testiranja (Hrgarek, 2013)

U agilnom pristupu razvoju programskih proizvoda poznatom pod nazivom *Scrum*, tester je dio tima za razvijanje kao specijalizirani stručnjak za tu fazu razvoja.



Slika : Organizacija testiranja u agilnom pristupu (Scrum)

Testeri se prema nazivlju kompetencija često nazivaju kao osiguravatelji kvalitete (engl. Quality Assurance). Praksa u ostalim inženjerskim strukama koristi ovaj univerzalni naziv.

Programeri imaju svoju značajnu ulogu u testiranju programskih proizvoda. Od njih ovisi mnogo. Najprije, programeri pišu jedinične testove. Više o jediničnom testiranju, u posebnom poglavlju.

* 1. Test-driven development

Razvoj vođen testiranjem ili razvoj usmjeravan testiranjem (*engl. test-driven development*) je je način razvoja programskih proizvoda gdje se testni slučajevi razvijaju i automatiziraju prije nego je razvijena aplikacija. Primarni cilj ovog načina razvoja jeste fokusiranje na specificiranju, a ne validaciji. Taj način tjera nas da dubinski razmislimo o zahtjevima i dizajnu sustava prije nego što napišemo konkretan kod. TDD se oslanja na specifikaciju. Takav jedan pristup ima za rezultat čistiji kod koji ima svoju svrhu, podlogu i osnovu na kojoj postoji.

* 1. Ad-hoc testiranje

Ad-hoc testiranje poznato i kao nasumično testiranje ili testiranje majmuna (*engl. Monkey testing)* je metoda testiranja koja nema formalne karakteristike, kao što su planiranje i dokumentacija. Ova metoda nema formalne pripreme, odnosno nema ne primjenjuju se priznate tehnike oblikovanja testova i nema očekivanih rezultata. Aktivnosti provedbe testiranja vođene su vlastitom voljom onoga koji testira programskih proizvod.

Tester improvizira korake koje proizvoljno izvršava često vođen trenutnim instiktom. Gledano sa formalnog aspekta, ova metoda ima mnoge nedostatke. Najviše iz razloga što nedostaju definirani testni slučajevi. No, ova metoda može iznenaditi sa rezultatima. Poznato je da čovjek svojim nasumičnim pristupom i vođen instiktom može otrkiti nedostatke koji se formalnim metodama teže pronalaze.

Ova metoda često se koristi kod testiranja prihvaćanjem (*engl. acceptance testing*). Kriteriji prihvaćanja su izlazni kriteriji koje komponenta mora zadovoljiti da bi projekat u konačnici bio prihvaćen od strane klijenta ili krajnjih korisnika. Moguće je da klijenti traže izvještaje. Neformalni izvještaji se ne temelji na propisanoj proceduri.

* 1. Testiranje performansi

<http://softwaretestingfundamentals.com/performance-testing/>

Ispitivanje performansi je vrsta testiranja koja pokušava odrediti i prikazati parametre stabilnosti i kvalitete odaziva performansi sustava. Performanse znaju veoma često zadati glavobolju administratorima sustava ukoliko nisu vršili pravovremeno i učestalo testiranja performansi, osobito onda kada se čini da je sustav stabilan i da obavlja svoje zadatke uredno. Najvažnije za sustav jeste da su performanse sustava skalabilne. To je garant da će sustav raditi na dugi period prema strategiji životnog ciklusa softvera. Također, jedan od bitnih principa jeste ne oslanjati se samo neke alate. Potrebno je osigurati zamjenska rješenja i drugačije algoritamske postavke za testiranje performansi sustava. Prosječni rezultati mogu biti najispravniji pokazatelj realnog stanja.

Ovaj tip testiranja se uspostavlja tako što se postavi okolina za testiranje što sličnija produkcijskom okruženju. Testno okruženje mora u svakom segmentu simulirati okolinu produkcije.

Postoji nekoliko tipova testiranja performansi, a nekoliko veoma važnih su:

* Testiranje opterećenja (*engl. load testing*)
* Testiranje otpornosti (*engl. stress testing*)
* Testiranje izdržljivosti (*engl. endurance testing*)
* Testiranje učinka (*engl. spike testing*)
  1. Jedinično testiranje

Jedinično testiranje (*engl. unit testing*) je razina testiranja programskih proizvoda gdje se ispituju pojedine jedinice ili komponente sustava. Cilj i svrha jeste provjeriti da li svaka jedinica radi kako je zamišljeno i definirano u specifikaciji zahtjeva. Jedinica odnosno komponenta je najmanji dio koji se testira. Računalnim jezikom govoreći, jedinica ima obično jedan ili nekoliko ulaza i jedan izlaz. U proceduralnom programiranju, jedinica se može označiti kao funkcija, postupak ili cijelina u programu. U objektno-orijentiranom programiranju, najmanja jedinica je metoda koja može pripadati klasi, apstraktnoj klasi, sučelju ili izvedenoj klasi. Također, postoje prakse u kojima se moduli aplikacije tretiraju kao jedinice koje se testiraju. Takva praksa ima mana u slučaju da postoji mnogo pojedinačnih jedinica unutar modula koje se mogu zasebno tretirati. Prema (ISTQB, 2007) jedinično testiranje je isto što i jedinično testiranje.

Jedinično testiranje izvodi se pomoću metode ispitivanje bijele kutije. Ujedno, ono je prva razina testiranja gledano pristupom odozdo-prema-gore (*engl. bottom-up testing*).

Ovu razinu testiranja obavljaju programeri koji prethodno kreiraju komponente aplikacije. Uloga testera u ovoj razini je minimalna ili nepostojeća, ali je važno da bazično poznavaju proces i svrhu ove razine testiranja. Također, mnogo im govori na čemu trebaju bazirati svoje eksplorativno testiranje (*engl. exploatory testing*). Postoje prakse u kojima programeri i testeri u paru pišu i izvršavaju jedinične testove.

Eksplorativno testiranje je neformalna tehnika kod koje tester aktivno kontrolira oblikovanje testova kada su ti testovi provedeni. Testeri koriste te informacije za unaprjeđenje procesa testiranja u budućnosti.

Tijek izvođenja jediničnog testiranja u cjelokupnom smislu izvodi se kroz pisanje plana, slučajeva jedinica i pisanja koda za ispitivanje valjanosti pojedinih komponenti.

Slika : Piramida automatskog testiranja

Testiranje jedinica sustava povećava povjerenje u kvalitetu koda odnosno u njegovu skalabilnost. Skalabilnost je sposobnost programskog proizvoda za nadograđivanje kako bi podržao dodatno opterećenje i nadogradnju. Skalabilno testiranje je ispitivanje sposobnosti programskog proizvoda za podnošenje dodatnog opterećnja. Ukoliko se jedinični testovi pišu precizno i kvalitetno, moguće je odmah nakon promjena i dodatnih opterećenja uhvatiti i prepoznati pogreške na način da je poznat okvirni uzrok i mjesto nastanka u kodu. Također, ukoliko su promjene već napravljene, a jedinice su malo međusobno ovisne moguće je lakše i brže napisati jedinične testove te je neželjeni učinak promjena u kodu značajno manji. Dobra praksa pisanja koda svakako je pisati dijelove koda što manje ovisne jedne od drugih (Bushnev, 2014).

Mnoge su prednosti prve razine testiranja. Jedna od njih svakako je ponovna iskoristivost, te su više upotrebljivi. Kako bi se moglo sustavno pristupiti pisanju jediničnih testova kodovi trebaju biti što više modularni sa jasnom definicijom, svrhom i jasnošću. To znači da je lakše ponovno iskoristi kod. Jedinično testiranje na neki način „nagrađuje“ dobro pisani kod, ali i tjera programera da piše što čišći kod (Martin, 2002). U principu, ova razina omogućuje brži razvoj. Kada se programeri služe uobičajenim pristupom kao što je često postavljanje točaka prekida (*engl. breakpoints*), ulaze u vrtlog vremena posebno kada je struktura koda kompliciranija. Jedinično testiranje može ubrzati cijeli proces zbog toga što je sigurniji pristup napisati i pokrenuti test dinamički. Također, pisanje ovih testova poboljšava samo razumijevanje vlastitog koda i procedura koje program izvršava.

Napor koji je potreban za pronalaženje i popravljanje nedostataka pronađenih tijekom jediničnog testiranja je veoma manji u odnosu na nedostatke otkrivene u višim razinama aplikacije. Samim time i cjelokupni trošak razvoja je manji. Pod troškove razvoja smatra se uloženo vrijeme, ljudski i drugi resursi. Veoma loša praksa je suočiti se sa pogreškama tijekom aktivnog korištenja ili tijekom testiranja prihvaćanja (engl. acceptance testing).

Ispravljanje pogrešaka je jednostavnije. Kada pojedini test ne uspije, potrebno je samo najnovije izmjene popraviti.

* + 1. Dobre prakse u jediničnom testiranju

Prvo što treba odlučiti kod pisanja jediničnih testova, jeste odabrati odgovarajući alat za testiranje koji je podržan za programski jezik koji se koristi. Alati za testiranje (*engl. frameworks*), slobodne biblioteke ili paketi omogućavaju programerima testiranje jedinica u programskom jeziku kojim programiraju same jedinice. Tako da većina programskih jezika ima određenu vrstu podrške za jedinično testiranje. Primjerice, testiranje aplikacija u C# omogućeno je kroz Visual Studio IDE kroz integrirane biblioteke.

Nema potrebe stvarati slučajeve testiranja. Formalno za testne slučajeve se piše specifikacija na početku procesa testiranja odnosno u fazi planiranja. Specifikacija testnih slučajeva je dokument koji specificira skup ciljeva, ulaza, izlaza i akcija jedinice koja se testira. Dobra praksa je usredotočiti se na jedinice koje uveliko utječu na ponašanje sustave na tzv. kritične točke sustava. Jediničnim testiranjem cilj je postići izolaciju odnosno izdvajanje razvojnog i testnog okruženja programa.

Dobra je praksa korištenje podataka koji će se koristiti u produkciji omogućava rano otrkivanje pogrešaka pogotovo onih koji nisu specifični za programsku logiku nego na izuzetke korištenja. To se posebice odnosi na ulazne podatke. Ukoliko se neki nedostatci slučajno otkriju ručnim testiranjem ili ad-hoc metodom, poželjno je napisati test, pa zatim otkloniti pogrešku, jer time preveniramo mogućnost ponavljanja te ili slične greške. Karakteristika programera jeste da žele što više stvari automatizirati, iz tog razloga ne vole ponavljanje procesa koji stvara nelagodu. To se postiže kontinuiranom nadogradnjom i poboljšanjem pisanja koda.

Kao što sam napomenuo, treba težiti pisanjem testnih slučajeva koji su međusobno neovisni. Primjerice, ako klasa ovisi o bazi podataka, ne pišite slučaj koji ispituje klasu interakcijom sa bazom podataka. Umjesto toga, izradite apstraktnu klasu ili sučelje oko te veze sa bazom podataka i implementirajte ta sučelja tako što ćete je postaviti u ulogu posrednika. Konačni cilj je pokriti sve putanje kroz jedinicu. Posebnu pozornost treba obratiti na uvjetnje petlje.

U konačnici, verzioniranje koda je veoma korisno i u danjašnjem modernom programiranju gotovo pa neizbježno kada su u pitanju veće aplikacije i više učesnika u razvoju. Korištenje sustava za upravljanje verzijama koda kao što je Git uvelike poboljšava učinkovitost programiranja i predstavlja dobru praksu.

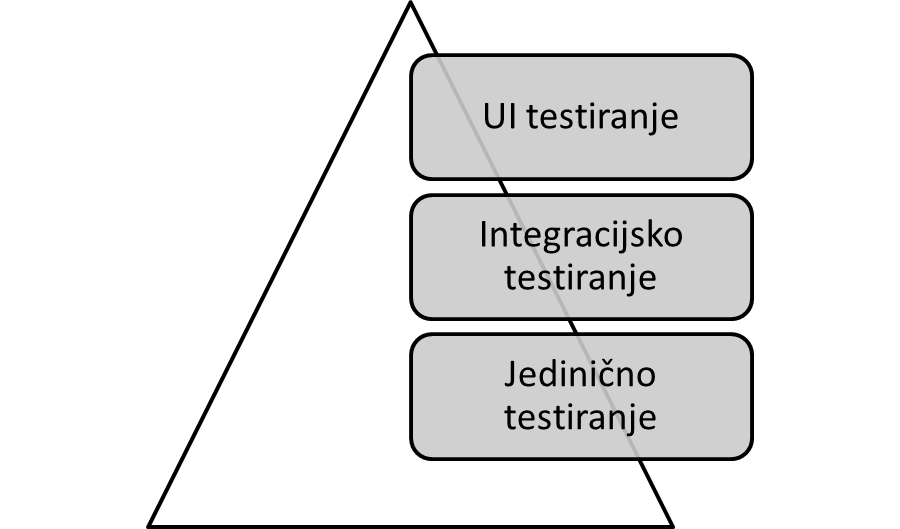
* + 1. Alati za kreiranje jediničnih testova

Postoji cijeli niz kvalitetnih alata koji nude široku lepezu mogućnosti za jedinično testiranje. Neki od popularnih alata za jedinično testiranja za razne programske jezike:

* xUnit.net i NUnit – besplatni alati za testiranje C# .NET aplikacija
* JUnit i NUnit – besplatni okviri/alati za jedinično testiranje aplikacija pisanih u Java programskom jeziku
* PHPUnit – okvir/alat za PHP jezik
* Karma i Jasmine – besplatni alati za testiranje web aplikacija u JavaScript-u
  1. Integracijsko testiranje

Integracijsko testiranje je kao spajanje ključnih točki sustava kroz razne mrežne slojeve aplikacije.

* + 1. Pristup odozdo prema gore



* 1. Testiranje web servisa

Alati za testiranje HTTP zahtjeva na poslužitelju

GET zahtjev.

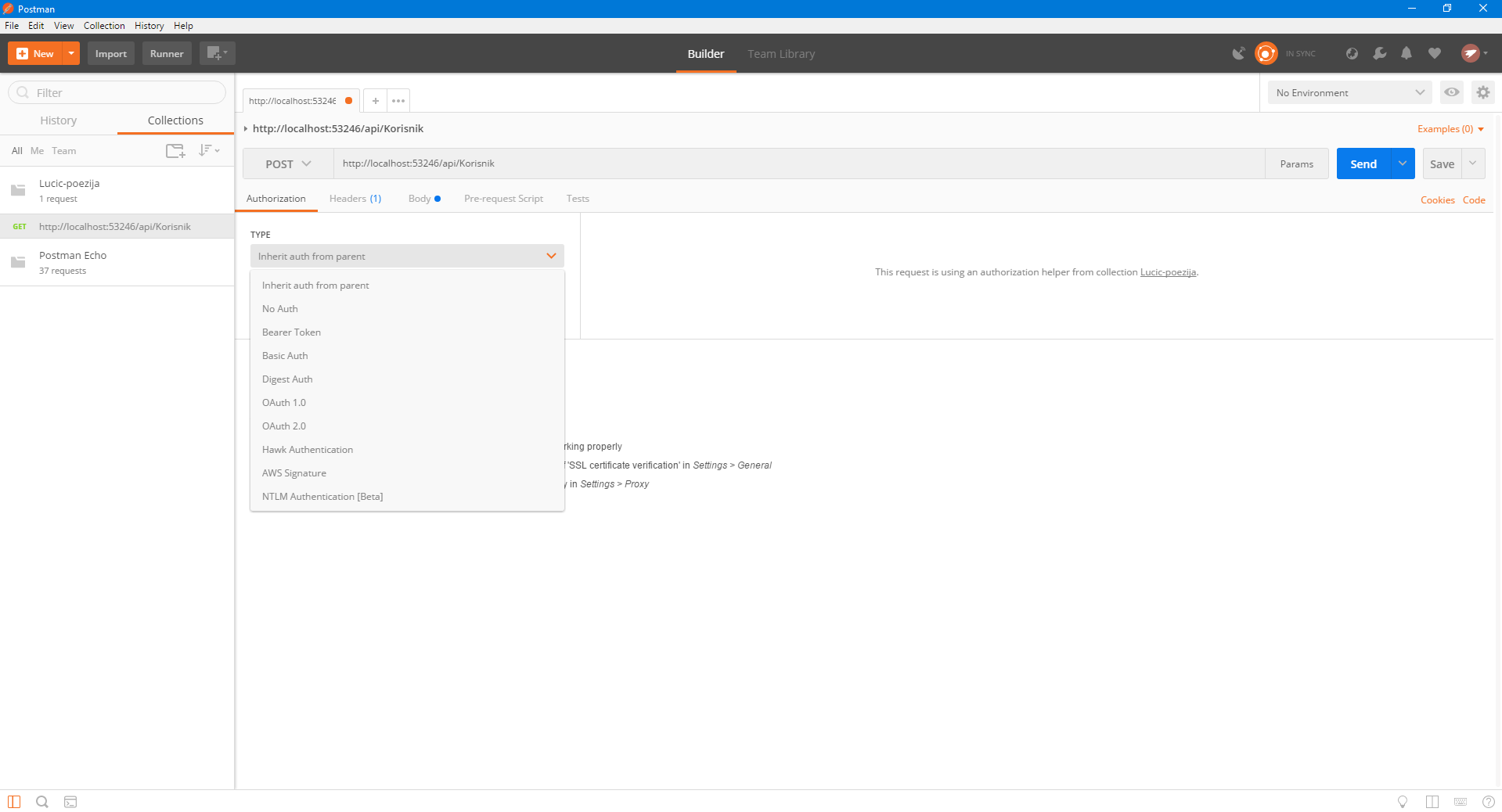
POST zahtjev.

PUT zahtjev.

DELETE zahtjev.

* + 1. Postman

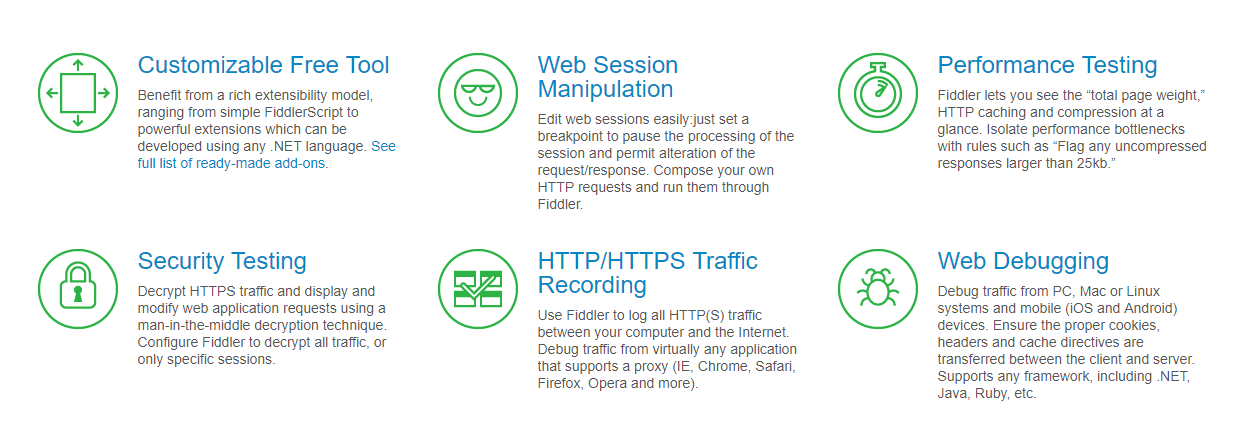
Postman spada u red veoma popularnih i široko korisnih alata koji pomaže programerima i testerima pri izradi i testiranju API-ja, web servisa i RESTful servisa. Ovaj alat se koristi u mnoge svrhe.



Slika : Postman - alat za testiranje API-ja

* + 1. Telerik Fiddler

Asdasd The free web debugging proxy   
for any browser, system or platform



* + 1. SoapUI

Alat za integracijsko testiranje…

* 1. Testiranje korisničkog sučelja

Testiranje korisničkog sučelja, češće nazvano UI testiranje spada u red automatiziranih testova visoke razine apstrakcije. Pišu se skripte koje testiraju aplikacije na isti način na koji to rade krajnji korisnici. Oni klikaju, odabiru i obavljaju sve druge radnje koje svaki prosječni korisnik radi prilikom korištenja web aplikacije. U suštini, ovi testovi su skriptna reprezentacija onoga što rade korisnici. Svrha svakog programskog proizvoda treba biti takva da su krajnji korisnici zadovoljni. Sve ono što se nalazi u pozadini, nije bitno ukoliko su oni nezadovoljni cjelokupnim proizvodom ili uslogom. Taj univerzalni princip tjera testere i programere da koriste ovom vrstom testiranja. S druge strane, sama suradnja testera i programera je veoma bitna da li kroz dobre prakse dizajniranja informacijskog sustava ili kroz svakodnevnu komunikaciju. Većina timova ima česte probleme na oba slučaja.

U piramidi testiranja programskih proizvoda često se postavlja na vrh same piramide, bilo da koristite bottom-up ili top-down pristup. U današnjem programskom inženjerstvu kada se spomene korisničko sučelje često se podrazumjeva grafičko korisničko sučelje, jer sve veći broj aplikacija u potpunosti prelazi u grafičku reprezentaciju umjesto tekstualno orijentiranih instrukcija.

Ovi testovi u suštini prolaze kroz sve slojeve aplikacije „od kraja do kraja“. To znači da prolaze kroz sve različite dijelove, kao što je korisničko sučelje, niže slojeve servisa i integracije sa samom bazom podataka. Prezentacijski sloj često može otrkiti mnoge nedostatke nižih slojeva aplikacije.

* + 1. Dimni testovi

Jedan od pojmova koje koristi jeste „dimni test“ (*engl. smoke test*). Također, poznato i kao testiranje gradnje verifikacija (engl. Build Verification Testing). To su testovi visokog sloja koji provjeravaju neke osnovne razine informacijskog sustava tako što vidi da li su pokrenuti i funkcionalni. Oni osiguravaju da najvažnije funkcije sustava rade stabilno. „Skup svih definiranih/planiranih testnih slučajeva koji pokrivaju glavnu funkciju komponente/sustava u svrhu provjere ključnih funkcija programa, ali bez zamaranja sa finim detaljima. Dnevno izdanje i dimni test spadaju u najbolje prakse širom industrije.„ (ISTQB, 2011.).

Izraz „dimno testiranje“ došao je kao aluzija na sličan tip hardverskog testiranja u kojem uređaju prolaze kroz test u kojem cilj da se „ne zapali“, odnosno da li ostaje funkcionalan nakon prvog uključivanja. Prema (Rasmusson, 2016., str. 21) korisno je zato što:

* provjerava je li aplikacija ispravno isporuče
* provjerava je li okruženje ispravno postavljeno
* i jesu li svi dijelovi arhitekture isptavno povezani i postavljeni pravilno

Dimni testovi garantiraju ispunjenost nekih minimalnih slojeva sustava i njihovog funkcioniranja. Ono čini UI testovima korisnim u cjelokupnom procesu osiguravanja kvalitete programskog proizvoda. Primjerice, ukoliko želimo testirati mogućnost prijavljivanja na neku web aplikaciju unošenjem korisničkog imena, lozinke i klika na gumb, preusmjeravanja na stranicu dobrodošlice, trebamo dobro razmisliti i zapisati korake ponašanja sustava sa strane krajnjeg korisnika. Kada zapišemo korake, potrebno je pretvoriti ih u test koji će to pravilno ispitati.

Mnogi programski jezici imaju već dostupne biblioteke koje olakšavaju ovaj proces. Ukoliko se radi o slučaju korištenja koji se često koristi u raznim sličnim sustavima, poželjno je potražiti rješenja koja su već usvojena kao uzorak dizajna ili kao dobra praksa. U osnovi, UI testovi se mogu bazirati na ključnim alatima za prikaz web aplikacija, kao što su HTML (engl. Hypertext Markup Language) i CSS (engl. Cascading Style Sheets). Za primjer prijavljivanja na sustav, kroz ovaj tip testiranja potrebno je ispitati dohvaćanje elemenata kroz njihov tip podataka i jedinstveni indeks.

Neki od najpopularniji alata za automatsko testiranje (Kazeeva, 2018):

* Selenium IDE i Selenium Web Driver
* Carina
* Google EarlGrey
* Cucumber
* Watir
* Appium
* RobotFramework
* Apache JMeter
* Gauge
* i Robotium

Kada su u pitanju moderni principi UI testiranja treba obratiti pažnju na nekoliko važnih principa (Blazedemo, 2017). Jedan od najvažnijih svakako je ne oslanjati se samo na UI testove. Zatim, treba koristiti okvire za testiranje ponašanja sustava (engl. Behavior-driven development).

Bazični okvir za uspješno testiranje jeste korištenje uzoraka dizajna i principa testiranja. Ono što mnogi zaborave u želji da pokriju što šire mogućnosti korištenja programa, jeste da pokušavaju praviti i pokretati sve testove na svim pretraživačima istovremeno, što je pogrešna praksa i više razloga. Jedno su ograničeni resursi, a drugo nemogućnost detaljnog uvida u pojedinosti rezultata testova na pojedinim pretraživačima.

Između ostalog, potrebno je voditi računa o prenosivosti alata za automatsko testiranje. Korištenje standardiziranih konvecija imenovanja testova je jedan od važnih principa kvaltietnog dizajna testiranja. Spremanje slika ekrana prilikom neuspješnih provjera, jedan je od trikova koji mogu olakšati daljni razvoj testova. Iako je dodavanje komentara veoma korisno u razvoju programskih proizvoda, dobra je praksa pisati pojednostavljene testove. Pored toga, svi testovi treba što manje ovisiti jedan od drugog. Zatim, treba težiti prema korištenju dinamičkim procedurama baziranim na podacima umjesto ponvaljanja. U programskom inženjerstvu uvijek treba težiti automatizaciji procesa, jer ono na duže staze uvijek daje bolje rezultate od ručnog statičkog procesa. Korisno je postaviti detaljno izvještavanje kroz dnevnike rada.

Ova vrsta testiranja omogućuje programerima bolje razumijevanje suštine rada aplikacije sa strane krajnjih korisnika. To znanje je neophodno kako bi programeri testerima omogućili da aplikacija bude što više pogodna za pisanje i nadogradnju UI testova. A sama uloga testera je da pokrije što veći raspon testiranjem „od kraja do kraja“ korištenjem UI testova. Ali treba biti spreman potrošiti dodatnog vremena i resursa na mnoge „lukave dijelove“ aplikacije, korištenjem drugih vrsta testiranja. Stoga, UI testiranje često nije dovoljno, ali zato pokriva najširi spektar.

* + 1. Alati za UI testiranje

Danas postoji niz kvalitetnih i naprednih alata za UI testiranje. Razvoj ovih alata polako mijenja paradigmu, širinu i mogućnosti ove veoma korisne razine testiranja. Neki od tih alata se koriste neovisno o platformi, tehničkim uvjetima, programskim jezicima itd. Postoji niz specijaliziranih alata, od kojih se ističe Selenium.

Slika

Selenium je skup alata (*engl. framework*) za automatsko testiranje korisničkog sučelja web aplikacija (Radotić, S. 2016). On pokriva široki spektar korisnih mogućnosti za kvalitetno testiranje aplikacija na svim široko raprostranjenih pretraživačima. Selenium ima dobru podršku za pisanje testova pomoću različitih platformi. Selenium je svakako najpopularniji i najrasprostranjeniji alat za automatsko UI testiranje. Dio Selenium paketa su Selenium IDE i Selenium Web driver. To je skup alata i dodataka aktuelnim preglednicima.

1. Testiranje u kontekstu kontinuirane integracije, dostave i objave

Kontinuirana integracija (*engl. Continuous integration*) spada u red najboljih praksi dostave i objave aplikacija. Pojmovi kontinuirane integracije, dostave i objave često idu zajedno iz razloga što moderni alati objedinjuju procese tj. nude sve te mogućnosti kroz jedan alat (Maynard, 2018).

Razlike…

* 1. Continuous integration

Continuous integration (u daljnem tekstu CI) je praksa kod testiranja programskih proizvoda koja omogućuje integraciju izvornog koda neprekidno u više iteracija i instanci.

Svrha CI jeste brzo otrkivanje pogreški i lakše pronalaženje izvora odnosno okidača pogreške.

Slika : Prikaz procesa kontinuirane integracije

Neke od dobrih praksi korištenja kontinuirane integracije su:

* Održavanje i pravilno korištenje Git repozitorija
* Automatiziranje pokretanja svih modula projekta
* Provjeravanje ispravnosti izvornog koda
* Konfigurirati okruženje pokretanja prema potrebama i mogućnostima
  1. Continuous delivery

Kontinuirana dostava (*engl. Continuous Delivery*) na neki način objedinjuje sve pakete distribucije sustava u jedan koji će se izvršiti.

Slika : Prikaz procesa kontinuirane dostave

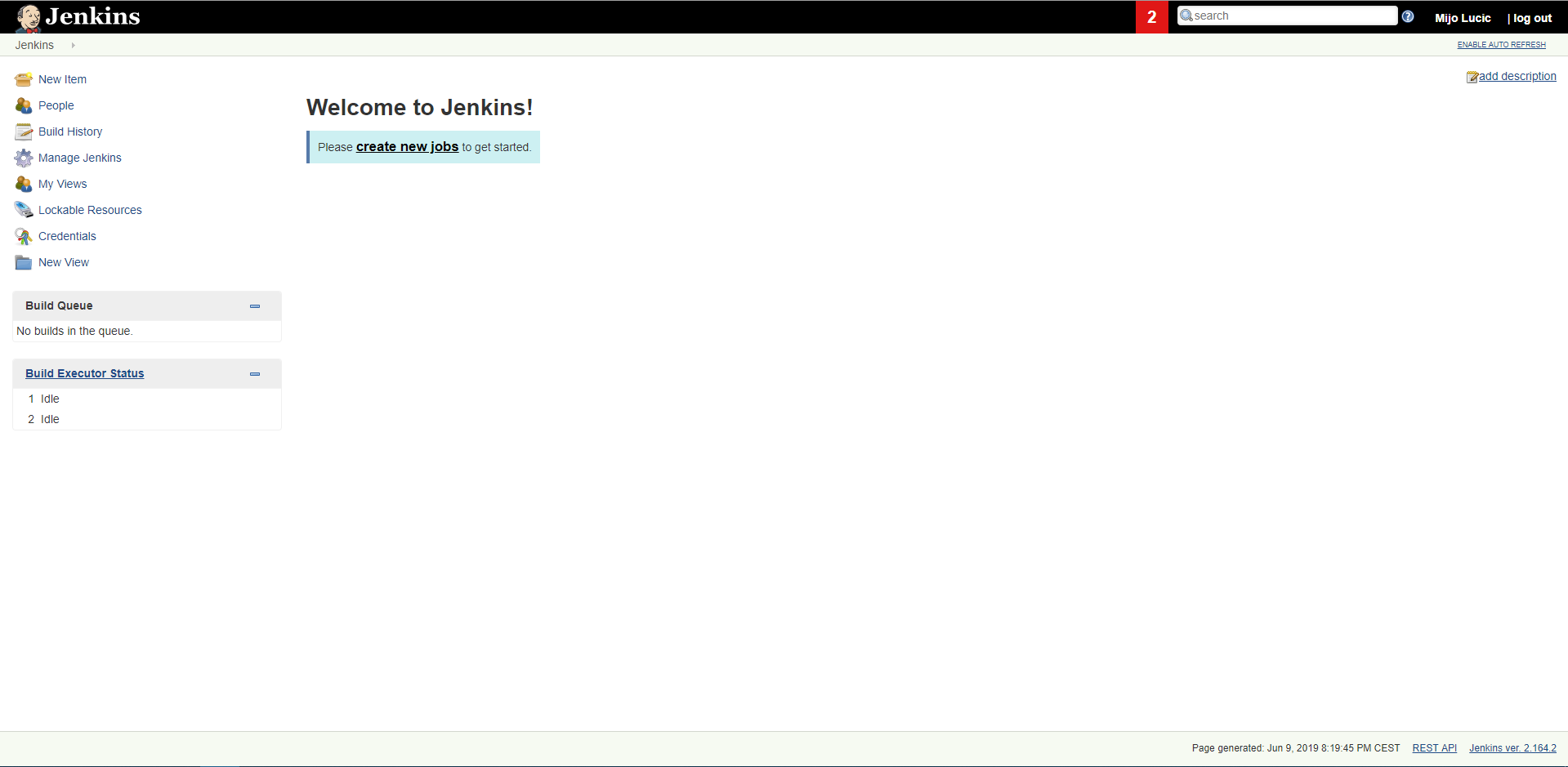
* 1. Continuous deployment

Kontinuirana objava (*engl. Continuous deployment*)

Slika : Prikaz procesa kontinuirane objave

* 1. CI/CD aplikacije putem Jenkins-a

Jenkins je veoma popularni besplatni poslužitelj za kontinuiranu integraciju, dostavu i objavu baziran na Java platformi. Pruža stotine korisnih dodataka koji podržavaju izgradnju, implementaciju i automatizaciju bilo kojeg programskog proizvoda, bez obzira u kojem jeziku pisan. Omogućuje lagani i jednostavnu instalaciju i postavku platforme koja će automatizirati proces testiranja kroz primjenu kontinuirane integracije, dostave i objave. Funkcionira na principu stvaranja paketa projekta kojeg objavljuje povezan sa izmjenama koje se evidentiraju kroz sustav za verzioniranje.



Slika : Jenkins - alat za CI i CD

Instalacija i konfiguracija Jenkins-a

Tekst…

Povezivanje sa sustavom za verzioniranje (Git)

Tekst…

Postavljanje radnika

Tekst…

Postavljanje pipeline

Tekst…

Alternativni alati za CI/CD

Tekst…

Kreiranje Docker slika

1. Testiranje u kontekstu osiguranja kvalitete programskih proizvoda

Ispitivanje ili kontrola kvalitete programskih proizvoda (*engl. Software Quality Control*) je skup aktivnosti za osiguravanje kvalitete u programskih proizvodima. Njen cilj je osigurati da programski proizvodi zadovoljavaju sve zahtjeve i specifikacije (Banjavčić, 2017).

Ona uključuje dvije glavne aktivnosti, a to su pregledi (*engl. reviews*) i testiranje. Aktivnosti testiranja provode se kroz razine, kao što su jedinično, integracijsko, sustavsko i testiranje prihvatljivosti.

Pregledi uključuju niz podaktivnosti, a ključne su:

* Pregled zahtjeva (*engl. requirement review*)
* Dizajn pregleda (*engl. design review*)
* Pregled koda (*engl. code review*)
* Pregled plana razmjene (*engl. deployment šlan review*)
* Pregled testnog plana (engl. test plan review)
* Pregled testnih slučajeva (*engl. test cases review*)

Kontrola kvalitete programskog proizvoda mjeri se na više načina. Mjerenje pogrešaka (*engl. defect age*) u jedinici vremena i po fazama (Software Testing Fundamentals, 2017).

Gustoća pogrešaka (*engl. defect density*) je broj potvrđenih nedostataka otkrivenih u komponentama tijekom određenog razdoblja razvoja podijeljen s veličinom komponente.

Učinkovitost otkrivanja pogrešaka (*engl. defect detection efficiency*) je broj nedostataka otkrivenih tijekom faze otkrivanja, podijeljen s brojem ukupnim brojem otrkivenih pogrešaka.

Trošak kvalitete (*engl. cost of quality*) je mjera koja zbraja trošak kontrole kvalitete i trošak neuspjeha kontrole. Drugim riječima, sažima troškove vezane za prevenciju i otkrivanje nedostataka i troškove zbog kojih se pojavljuju pogreške.

* 1. DevOps

Jakooo bitno!!!

* 1. Standardizacija i certifikacija testiranja softvera

Kako je programsko inžinjerstvo jedno od najmlađih inženjerskih znanosti s vremenom se pojavila potreba za što većom standardizacijom. Kako bi se izbjegao nered koji je bio karakterističan za mladu znanost u svom nastajanju, postavljanje principa, načela i metoda rada bilo je neizbježno, jer se radi o velikoj realnoj potrebi za kvalitetnim proizvodima i uslugama iz ovog sektora. Tako je uporaba informacijskih tehnologija glavna karakteristika modernog društva i neizostavan dio svih velikih industrija. Da bi se osigurala kvaliteta neophodna je standardizacija, a s vremenom i ispitivanje kompetencija ljudi u toj industriji, tzv. certifikacija.

Prema (Padmini, 2006) neke od najraširenijih certifikacija iz domene testiranja softvera su:

* CSQE – ASQ (American Society for Quality)
* CSQA/CSTE – QAI (Quality Assurance Institute)
* ISEB
* ISTQB

ISTQB je skraćenica od Međunarodni odbor za osposobljavanje testera softvera (engl. International Software Testing Qualification Bord). To je međunarodna organizacija…

<https://www.istqb.org/references/surveys/istqb-worldwide-software-testing-practices-report.html>

* 1. Normativne reference testiranja u softverskom inženjerstvu

Prema (ISO/IEC JTC1/SC7, 2014) postoji više svjetskih standarda odnosno normativnih referenci koji se tiču testiranja programskih proizvoda, od kojih su:

* ISO 9000:2005 – Quality Managment System – Fundamentals and vocabulary
* ISO/IEC 29119-1: Concepts & Definitions (published September 2013)
* I[SO/IEC 29119-2: Test Processes (published September 2013)](http://www.softwaretestingstandard.org/part2.php)
* ISO/IEC 29119-3: Test Documentation (published September 2013)
* ISO/IEC 29119-4: Test Techniques (at DIS stage, anticipating publiation in late 2014)
* ISO/IEC 29119-5: Keyword Driven Testing (at CD stage, anticipating publication in 2015)

1. Implementacija i testiranje WebRTC aplikacije

WebRTC je besplatni web okvir (*engl. framework*) koji omogućava komunikaciju u realnom vremenu putem preglednika (*engl. real-time comunications*). To uključuje osnovne elemente za kvalitetnu komunikaciju putem web aplikacija, kao što su audio, video komponente korištene u aplikacijama za glasovni i video chat i streaming.

WebRTC komponente se implementiraju preko preglednika koristeći dostupni JavaScript API. Programeri koriste te komponente kako bi postigli razne multimedijalne mogućnosti svojih aplikacija na većini popularnih preglednika za web i mobitele. Svrha nastanka WebRTC-a je standardizacija multimedijalnih mogućnosti preglednika bazirajući se na API razini W3C (*engl. World Wide Web Consortium*).

Postoje mnoge prednosti korištenja WebRTC API-ja za izradu aplikacija, kao što je video-chat. Ključne prednosti se ogledaju kroz to što su temeljne tehnologije za web, kao što su HTML, CSS, HTTP, TCP/IP protokoli otvoreni i slobodni za daljnu nadogradnju korištenje. Snaga web zajednice jeste u tome što omogućuje svima da se uključe i doprinesu općoj koristi. Trenutno, ne postoji bolje besplatno rješenje za izradu visoko kvalitetnih aplikacija koje se služe komunikacijom u pregledniku. Ono što čini globalnu prednost je kontinuirana integracija sa raznim audio i video uređajima koji su prisutni kod mnogih koji se koriste Internet tehnologijama, drugim riječima krajnjim točkama internetske mreže. Sa sigurnosnog i mrežnog aspekta, WebRTC uključuje NAT i firewall tehnologiju koristeći razne podrške, kao što je RTP-over-TCP, STUN i ICE, te podrška za *proxy* poslužitelje.

WebRTC se gradi na podrškama za preglednike. On sažima signalizaciju nudeći signalni stroj koji izravno prenosi *PeerConnection*. Stoga, programerima se nudi mogućnost odabira protokola za njihove potrebe ograničavajući minimalno.

* 1. Arhitektura WebRTC okvira

WebRTC ima dobro razrađenu i kompleksnu arhitekturu, koja u konačnici programerima omogućuje izradu bogatih multimedijalnih web aplikacija u stvarnom vremenu bez potrebe za raznim dodacima i instalacijama. Opća svrha je omogućiti svima snažne RTC aplikacije da radi jednako na više web i mobilnih preglednika, te na više platformi neovisno o razini mrežnih protokola. Arhitektura u osnovi bazira se na dva sloja: WebRTC C++ API namijenjen razvoju web preglednika i WebRTC Web API namijenjen web programerima za razvoj web aplikacija.

Cjelokupna arhitektura WebRTC-a izgleda ovako:



Slika : Arhitektura RTCPeerConnection komponente WebRTC-a

Komponente sesije su izgrađene kako bi se mogle iznova koristiti bez zahtijevanja xmpp/jingle protokola. RTP stack je mrežni stog za protokol realnog vremena (*engl. Real Time Protocol*). STUN/ICE je komponenta koja dopušta pozive koji koriste STUN i ICE mehanizme za uspostavljanje veza kroz različite tipove mreža. *Session Managment* je apstraktni sloj sesije koja dozvoljava uspostavljanje poziva i sloj organizacije same sesije kroz mrežu. Ono pruža programera da odluči o tome koji će protokol koristiti za uspostavljanje veze.

Prema (WebRTC, 2018) VoiceEngine je okvir za uspostavljanje zvučnog signala od zvučne kartice kompjutera do mreže. VideoEngine je okvir za uspostavljanje videa prolazeći kroz razne slojeve, od vanjskog uređaja kamere do mreže i obrnuto. Kada su u pitanju tehničke karakteristike kvalitete koje omogućuje i na neki način garantira, ona je zadovljavajuća za većinu potreba u okviru korištenja multimedijalnih mogućnosti kroz preglednike.

* 1. Pribavljanje korisničkih medija

Pribavljanje korisničkih medija (*engl. getUserMedia*) predstavlja sinkronizirane struje medija. Jedna od osnovnih uporaba WebRTC API-ja je kroz implementaciju *MediaStream* komponente koja dobiva pristup podatkovnim tokovima, kao što je web kamera i mikrofon. Svakako, korisnik mora odobriti korištenje tih uređaja.

Metoda getUserMedia() uzima parametar objekta *MediaStreamConstraints* i vraća varijablu koja se poziva u *MediaStream* objektu. Svaki *MediaStream* ima oznaku, koja je kriptirana u obliku lozinke. Nizovi *MediaStreamTracks* vraćaju metode *getAudioTracks()* i *getVideoTracks()*.

*RTCPeerConnection* je WebRTC komponenta koja obrađuje komunikaciju podataka strujanjem između krajnjih točaka (*engl. peers*).

* 1. Testiranje WebRTC aplikacije

Jedna od važnih zadaća u ostvarivanju misije WebRTC-a jeste da osiguraju često automatsko testiranje svojih API-ja. Kako se web tehnologije često mijenjaju, potreba za testiranjem je učestala i neophodna.

* + 1. Podrška na web preglednicima

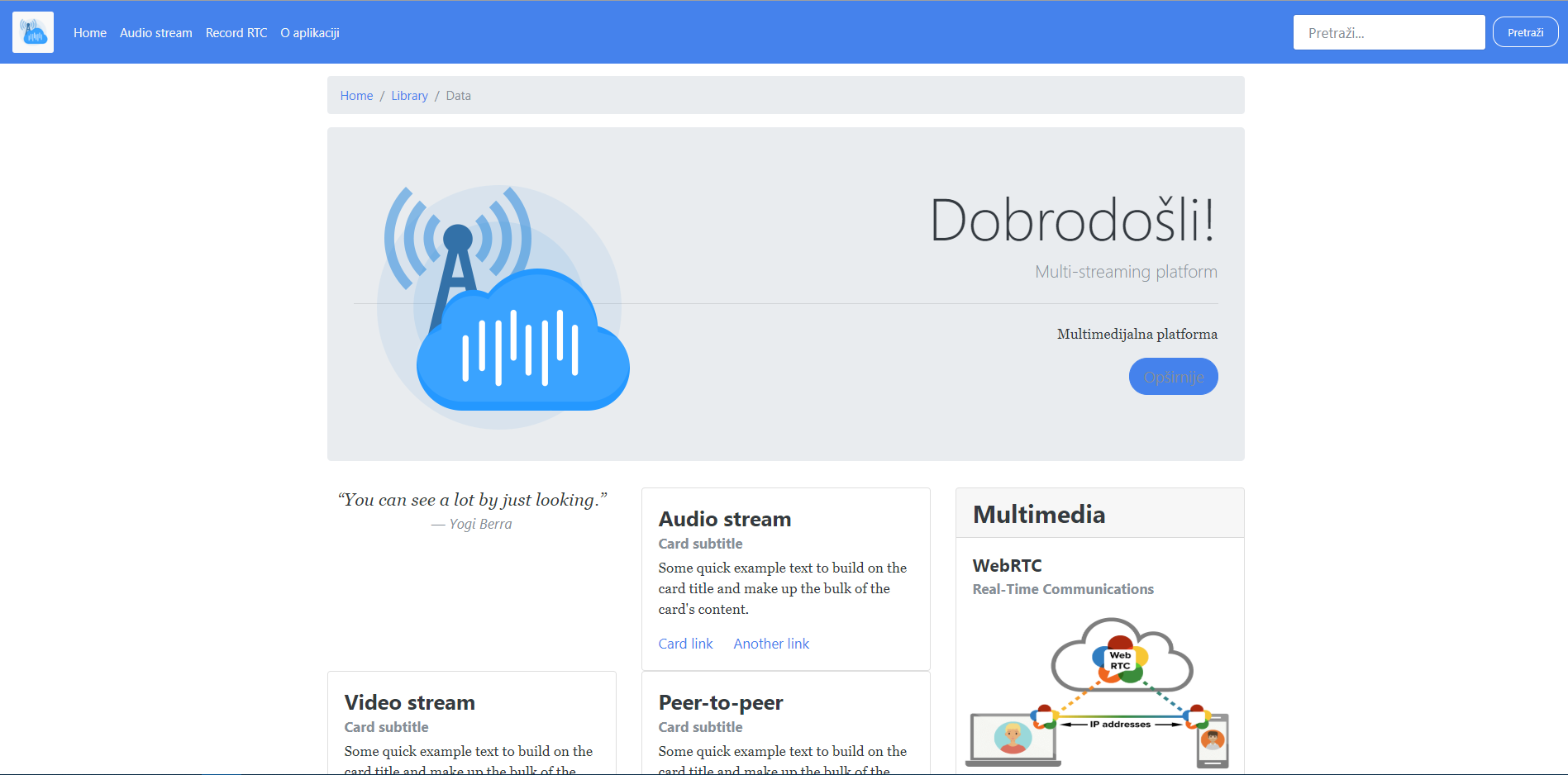
Prema mnogim istraživanjima u posljednjih 10 godina, uz neka odstupanja, vodeći preglednici u svijetu su Chrome i Firefox. Ta dva preglednika interoperabilni su, ali s obzirom na svoje specifičnosti i mala odstupanja od standarda kojeg propisuje W3C, ti preglednici zahtijevaju mali stupanj prilagodbe. Tako da API za pribavljanje korisničkih medija *getUserMedia* (prema W3C) je u Chrome *webkitGetUserMedia*, a kod Firefox-a, *mozGetUserMedia*. Svakako, pri korištenju tih API-ja treba provjeriti da li su aktuelni i da li imaju trenutnu podršku za preglednik kojeg želite koristiti.

WebRTC adapter je dostupan preko *GitHub-a*, hostinga za verzioniranje koda. Taj adapter preko svoje biblioteke *polyfill* olakšava rad sa zastrjelim tehnologijama, te omogućuje programerima korištenje standardnih naziva prema W3C.

Također, treba napomenuti da se sve te mogućnosti mogu koristiti i na mobilnim uređajima kroz nativni API. Za izradu aplikacija koja će imati tu podršku, potrebno je zadovoljiti neke posebne uvjete.

* 1. Rezultati testiranja multimedijalne aplikacije

Za potrebe istraživanja napravio sam web aplikaciju koja nudi korisnicima mogućnost osnovnog audio i video strujanja (*engl. streaming*).



Slika : Slika ekrana web aplikacije napravljene za istraživanje testiranje kompleksnih multimedijalnih aplikacija

* 1. Testiranje usklađenosti

Aplikaciju koju sam izradio u svrhu istraživanja, testirao sam na operacijskom sustavu Windows 10 i na web preglednicima Chrome, Firefox i Edge.

* 1. Kontinuirana integracija i dostava

WebRTC-in-Chrome implementacija je izgrađena na bazičnom kodu koji je pripremio Google tim, a koji uključuje dvije biblioteke: *webrtc* i *libjingle*.

* 1. Testiranje pomoću alata Kite

Kite je besplatni alat za testiranje interoperabilnosti između različitih preglednika. On uveliko olakšava testiranje interoperabilnosti aplikacija i rano otkrivanje regresija. Dizajniran je da bude ponovno iskoristiv, jednostavan za korištenje i nadogradnju automatiziranog okruženja za testiranje. Za sada, on dobro radi na stabilnim verzijama najpopularnijih preglednika, kao što su Chrome, Firefox, Opera, Safari i Edge koji se pokreću na operacijskim sustavima Windows, Linux ili MacOS. Testiranje za mobilne uređaja još uvijek je u fazi razvoja. Nativne aplikacije sve više uzimaju maha, ali one zahtjevaju drugačiji arhitekturalni pristup testiranju, a za to je zadužena suradnja među velikim softverskim kompanijama i konzorciju koji je kreirao WebRTC.

Prema (WebRTC, 2018) postoje javno dostupni testovi putem alata Kite koji ispituju interoperabilnost po kriteriju interoperabilnosti preglednika na različitim operacijskim sustavima:

* *WebAudioInputTest* – provjerava integraciju WebAudio API-ja
* *VP8SDPNegotiationTest* –provjerava sposobnost koristeći samo vp8 kodek
* *SimulcastTest* – provjerava može li preglednik koristiti simulcast za WebRTC
* *RebroadcastTest* – provjerava sposobnost prijenosa primljenog strujanja na drugu međusobnu vezu
* *NoAdapterTest* – provjerava interoperabilnost između preglednika bez adapter.js datoteke
* *MultiStreamTest* – provjerava sposobnost korištenja višestrukih strujanja preko WebRTC-a
* *MediaRecorderAPITest* – provjerava API za snimanje medija
* *IceConnectionTest* – provjerava ICEConnection stanje između dva preglednika koji komuniciraju putem appr.tc biblioteke.
* *H264SDPNegotiationTest* – ovaj test provjerava sposobnost preglednika za rad koristeći samo H-264 medijski kodek
* *CanvasStreamToPcTest* – provjerava *MediaCapture()* funkciju na medijskom platnu (*engl. canvas*)

<https://github.com/webrtc/KITE>

* 1. Automatsko UI testiranje pomoću Selenium-a

asdas

1. Zaključak

Neki od mitova kojima su mnogi ljudi u branši skloni, jeste da je kontrola kvaliteta jednaka testiranju. Ustvari, testiranje je samo jedna komponenta kontrole kvalitete softvera. Primjerice, kontrola kvaliteta uključuje mnoge aktivnosti kao što je pregled (*engl. reviews*). Također, mnogi misle da je cilj testiranja otkriti i otkloniti 100% pogrešaka. Činjenica jeste da je cilj otkloniti što veći mogući broj nedostataka uz osiguravanje da program zadovoljava zahtjeve. Otrkivanje i uklanjanje svih nedostataka je nemoguće.

Ono što mnogi sa vanjske perspektive ne mogu da razumiju, jeste da postoje situacije kada je testiranje veoma težak proces, ponekad i teži od samog kodiranja. Mnogi misle da je lagano testirati i da nije potrebno mnogo tehničkog znanja. No, testiranje je rigorozna disciplina koja zahtjeva široku lepezu vještina iz mnogih aspekata života. Također, zabluda je da automatizirani testovi eliminiraju potrebu za ručnim testovima. Ponekad je ručno testiranje neophodno. Situacija u kojoj je došlo do ogromnog nedostatka se često zna pripisati testerima. Kvaliteta softvera je odgovornost svih članova i dionika uključujući cijeli tim. Kako se razvija programsko inženjerstvo, odgovornost na procesu testiranja postaje sve bitnija. Tu činjenicu potvrđuje sve veći iznos ukupnog ulaganja u ljude i infrastrukturu koji u nekim korporacijama dostiže jednako procesu razvoja.

Popis literature

Alan B. Johnston, D. C. (2014). *WebRTC: APIs and RTCWEB Protocols of the HTML5 Real-Time Web, Third Edition.* St. Louis, MO, SAD: Digital Codex LLC, 3. izdanje.

Balakrishnan, H. M. (2017). *Trends in Software Testing.* Singapore: Springer.

Banjavčić, J. (2017). *Metode vrednovanja kvalitete programskih proizvoda.* Pula: Sveučilište Jurja Dobrile u Puli.

Bazzana, G. Y. (2015 - 2016.). *ISTQB Worldwide Software Testing Practices Report.* Brisel, BEL: International Software Testing Qualifications Board [ISTQB].

Bushnev, Y. (14. Studeni 2017). *Top 15 UI Test Automation Best Practices You Should Follow*. Dohvaćeno iz BlazeMeter: https://www.blazemeter.com/blog/top-15-ui-test-automation-best-practices-you-should-follow

Contributor. (4. Srpanj 2017). *13 Best Practices of Successful Software Testing Projects*. Dohvaćeno iz DevOps: https://devops.com/13-best-practices-successful-software-testing-projects/

FOI. (studeni 2017). *Preporuke citiranja i referenciranja primjenom stila refernciranja APA.* Varaždin: Sveučilište u Zagrebu, Fakultet organizacije i informatike.

Fundamentals, S. T. (n.d.). *Software Quality Control*. Dohvaćeno iz Software Testing Fundamentals: http://softwaretestingfundamentals.com/software-quality-control/

Hrgarek, N. (2013). Testiranje i kvaliteta softvera. Varaždin.

ISO/IEC JTC1/SC7, W. G. (2014). *The International Software Testing Standard*. Dohvaćeno iz Software Testing Standard: http://www.softwaretestingstandard.org/

ISTQB. (2016). *Certification path*. Dohvaćeno iz ISTQB: https://www.istqb.org/certification-path-root.html

Ivandić, M. (2017). *Važnost, tipovi i primjeri testiranja programskih proizvoda.* Split: Sveučilište u Splitu, Ekonomski fakultet.

Jorgensen, P. C. (2013). *Software Testing: A Craftsman’s Approach, Fourth Edition.* Boca Raton, FL, SAD: Auerbach Publications.

Kazeeva, A. (14. Ožujak 2018). *10 Best Open Source Test Automation Frameworks for Every Purpose*. Dohvaćeno iz DZone: https://dzone.com/articles/10-best-open-source-test-automation-frameworks-for

Martin, R. C. (2002). *Agile Software Development, Principles, Patterns, and Practices.* New Jersey, USA: Pearson Education.

Maynard, C. (n.d.). *Software Testing for Continuous Delivery*. Dohvaćeno iz Atlassian: https://www.atlassian.com/continuous-delivery/software-testing

Mohamed Monier, M. M.-m. (2015). Evaluation of automated web testing tools. *International Journal of Computer Applications Technology and Research, 4*(5), 405 - 408.

Myers, G. J. (2004). *The Art of Software Testing.* Canada: John Wiley & Sons, Inc.

Ninović, N. (2. 4 2018). *Kako automatski testovi menjaju pojam kvaliteta softvera*. Dohvaćeno iz Saga: http://www.saga.rs/blog/automatski-testovi

Padmini, C. (2006). *Beginners Guide To Software Testing.* Lulu Enterprises.

Perry, W. E. (2006). *Effective Methods for Software Testing.* New York, NY, SAD: John Wiley & Sons, Inc.

Popović, J. (2012). *Testiranje softvera u praksi.* Beograd: Računarski fakultet.

Rasmusson, J. (2017). *The Way of the Web Tester.* Raleight, NC, SAD: The Pragmatic Bookshelf.

Ristic, D. (2015). *Learning WebRTC.* Birmingham, VB: Packt Publishing.

*Selenium Documentation*. (13. Kolovoz 2018). Dohvaćeno iz SeleniumHQ: https://www.seleniumhq.org/docs/

Sergiienko, A. (2015). *WebRTC Cookbook.* Birmingham, VB: Packt Publishing.

Šljanić, S. (7. prosinac 2016). *Uspesan QA tim - Moje iskustvo u QA*. Dohvaćeno iz TNation: https://www.tnation.eu/blog/uspesan-qa-tim-moje-iskustvo-u-qa/

WebRTC. (2018). *Architecture*. Dohvaćeno iz WebRTC: https://webrtc.org/architecture/

Popis slika

[Slika 1: Životni ciklus testiranja 6](#_Toc11012916)

[Slika 2: Organizacija testiranja u agilnom pristupu (Scrum) 8](#_Toc11012917)

[Slika 3: Piramida automatskog testiranja 12](#_Toc11012918)

[Slika 4: Postman - alat za testiranje API-ja 16](#_Toc11012919)

[Slika 5: Prikaz procesa kontinuirane integracije 20](#_Toc11012920)

[Slika 6: Prikaz procesa kontinuirane dostave 21](#_Toc11012921)

[Slika 7: Prikaz procesa kontinuirane objave 21](#_Toc11012922)

[Slika 8: Jenkins - alat za CI i CD 22](#_Toc11012923)

[Slika 9: Arhitektura RTCPeerConnection komponente WebRTC-a 26](#_Toc11012924)

[Slika 10: Slika ekrana web aplikacije napravljene za istraživanje testiranje kompleksnih multimedijalnih aplikacija 28](#_Toc11012925)

Popis tablica

[Tablica 1: Razlike između verifikacije i validacije 4](#_Toc523195457)