Alati za automatizirano testiranje programskih proizvoda

Grdić, Matea

Sveučilište Jurja Dobrile u Puli
Fakultet informatike u Puli

MATEA GRDIĆ

ALATI ZA AUTOMATIZIRANO TESTIRANJE PROGRAMSKIH PROIZVODA

Završni rad

Pula, rujan, 2019

Sveučilište Jurja Dobrile u Puli
Fakultet informatike u Puli

Matea Grdić

ALATI ZA AUTOMATIZIRANO TESTIRANJE PROGRAMSKIH PROIZVODA

Završni rad

JMBAG: 0303061533, redoviti student
Studijski smjer: Informatika

Predmet: Programsko inženjerstvo
Znanstveno područje: Društvene znanosti
Znanstveno polje: Informacijske i komunikacijske znanosti
Znanstvena grana: Informacijski sustavi i informatologija
Mentor: doc. dr. sc. Tihomir Orehovački

Pula, rujan, 2019

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani _____, kandidat za prvostupnika _____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, _____ dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj završni rad pod nazivom

_____

_____

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

_____

Sadržaj

1. UVOD

Razvoj softvera je skup i dugotrajan proces. Svaki softver mora biti testiran kako bi bili sigurni da će funkcionirati u stvarnom okruženju. Od softvera se očekuje da odgovara karakteristikama kvalitete softvera, tj. da bude funkcionalan, pouzdan, učinkovit, prenosiv te prikladan za održavanje.

Kvaliteta softvera ovisi o jednostavnosti i razumljivosti izvornog koda programa. Od programa se očekuje da bude kvalitetan i da zadovoljava upite korisnika. Testiranje softvera smatra se jednim od najvažnijih procesa razvoja softvera. Ono je nužno u utvrđivanju kvalitete softvera, također je nužno kako bi se krajnjim korisnicima isporučio kvalitetan proizvod koji zahtjeva niže troškove održavanja te rezultira dosljednim i pouzdanim rezultatima. Testiranje softvera mora biti učinkovito u pronalaženju nedostataka, no ono bi trebalo biti izvedeno što je brže i jeftinije moguće. Testiranje softvera se ne smatra samo tehničkim zadatkom, ono uključuje važna razmatranja ljudske psihologije i ekonomije. Vrlo je važno identificirati i prijaviti pogreške u ranim fazama razvoja softvera. Razlog tome je što troškovi popravljanja pogrešaka rastu s vremenom. Zato je organizacija testiranja veoma bitna. Često se smatra da testiranje počinje tek nakon završetka kodiranja, no u životnom ciklusu razvoja konvencionalnog softvera, testiranje započinje u fazi kada su napisane specifikacije proizvoda. Odabir odgovarajuće tehnike koja smanjuje broj testova, jedna je od najvažnijih stvari koje testni tim treba uzeti u obzir pri dizajniranju testnih slučajeva. Test slučajevi moraju biti dizajnirani tako da pokrivaju sve aspekte softvera, kao što su sigurnost, funkcionalnost, korisničino sučelje, baza podataka itd.

Testiranje softvera možemo podijeliti na manualno tj. ručno testiranje i automatizirano testiranje. Ako je dobro provedeno, automatizirano testiranje može značajno smanjiti potreban trud te povećati efikasnost testiranja. Automatizirani testovi se mogu izvoditi u nekoliko minuta što bi za ručno testiranje bilo potrebno mnogo više vremena. Automatizacija testiranja eliminira sitne poslove. Što se testiranje čini „dosadnijim“ potreba za automatiziranim alatom je veća. No, automatizirano testiranje nije uvijek rješenje. Kod aplikacija kod kojih se testiranja provodi rijetko i čije je testove lako izvršiti ručno, nije isplativo ulagati u alate za automatizirano testiranje jer prilagodba tih alata može biti skup i dugotrajan proces. Stoga odluka o automatizaciji testiranja treba biti dobro promišljena. Također, treba dobro razmisliti o odabiru alata za automatizaciju, jer svaki alat neće odgovarati svakoj aplikaciji. Prilikom odabira alata za automatizirano testiranje testni tim može se odlučiti na besplatne „Open

Source“ alate, komercijalne alate te prilagođene alate njihovom projektu. Open Source alati su besplatni ali imaju manje značajki od komercijalnih alata. Kada nijedan od alata otvorenog koda niti komercijalni alat ne odgovara karakteristikama projekta, testni tim se često odlučuje za razvoj prilagođenog alata za automatizirano testiranje. Prilikom odabira alata za automatizirano testiranje važno je temeljito razumijevanje samih zahtjeva projekta. Prilikom odabira alata važno je uzeti u obzir vrstu projekta (web, desktop, mobilna aplikacija), sam opseg projekta te jezik u kojem se programira.

Uspjeh u bilo kojoj automatizaciji testiranja ovisi o pronalasku pravog alata za testiranje. To je veoma težak zadatak s obzirom na to da je danas na tržištu dostupno mnoštvo besplatnih i komercijalnih alata za automatizaciju testiranja. Jedni od popularnijih alata za automatizaciju danas su Selenium, Katalon Studio, UTF, TestComplete i SoapUI. U radu ću navesti njihove glavne karakteristike te za koje vrste projekta je dobro koristiti te alate. Pojasnit ću primjer testiranja u Selenium WebDriver te instalaciju samog alata.

Cilj ovog rada je obrazložiti važne komponente kvalitete softvera, testiranja softvera i vrste testiranja kako bi se detaljno pojasnio svaki od navedenih procesa kao i njihovu važnost u donošenju odluke o automatiziranju testiranja programa.

2. KVALITETA SOFTVERA

Početkom osamdesetih godina dvadesetog stoljeća počinju se javljati zahtjevi za utvrđivanjem karakteristika softvera koje bi se mogle mjeriti. Na taj bi se način mogao uspoređivati i vrednovati određeni softver, kvantificirati njegovi atributi ali i pratiti životni ciklus softvera. Mjerenje karakteristika softvera bitno utječe i na sam razvoj softvera. Kako bi se program mogao ispravno vrednovati i ocijeniti, za to su nam potrebni određeni sustavi, norme i vrijednosti. Oni čine sustav mjerenja kompjuterskih programa -metriku softvera (Oreški, 1990).

Od današnjih programa se očekuje da se odlikuje sljedećim atributima kvalitete, a to su (Manger, 2013):

1. Mogućnost održavanja - program se može mijenjati u skladu s promjenom potreba korisnika.

2. **Efikasnost** - program mora imati zadovoljavajuće performanse, treba upravljati resursima na štedljiv način.

3. Pouzdanost i sigurnost - program ne smije izazivati ekonomske ili fizičke štete, mora se ponašati na predvidljiv način.

4. Upotrebljivost - program treba imati sve funkcije koje korisnici od njega očekuju, mora sadržavati dokumentaciju i sučelje mora biti zadovoljavajuće i lako za korištenje.

Kod većine proizvoda može se reći da se njihova svojstva definiraju u fazi razvoja i oblikovanja, dok su nedostaci najčešće rezultat propusta u proizvodnji. Može se dogoditi da je nedostatak proizvoda i rezultat dizajna, ali to se rjeđe događa, jer se prije puštanja u proizvodnju obavljaju temeljita ispitivanja prototipova (Bevanda i Sinković, 2009).

Kvaliteta je prikladnost za upotrebu. Jedan od načina definiranja kvalitete je zadovoljstvo kupaca. Kupcem možemo nazvati svakog tko je na neki način zainteresiran za proizvod. Postoje dvoje grupe značajki proizvoda koje utječu na zadovoljstvo kupca (Bevanda i Sinković, 2009):

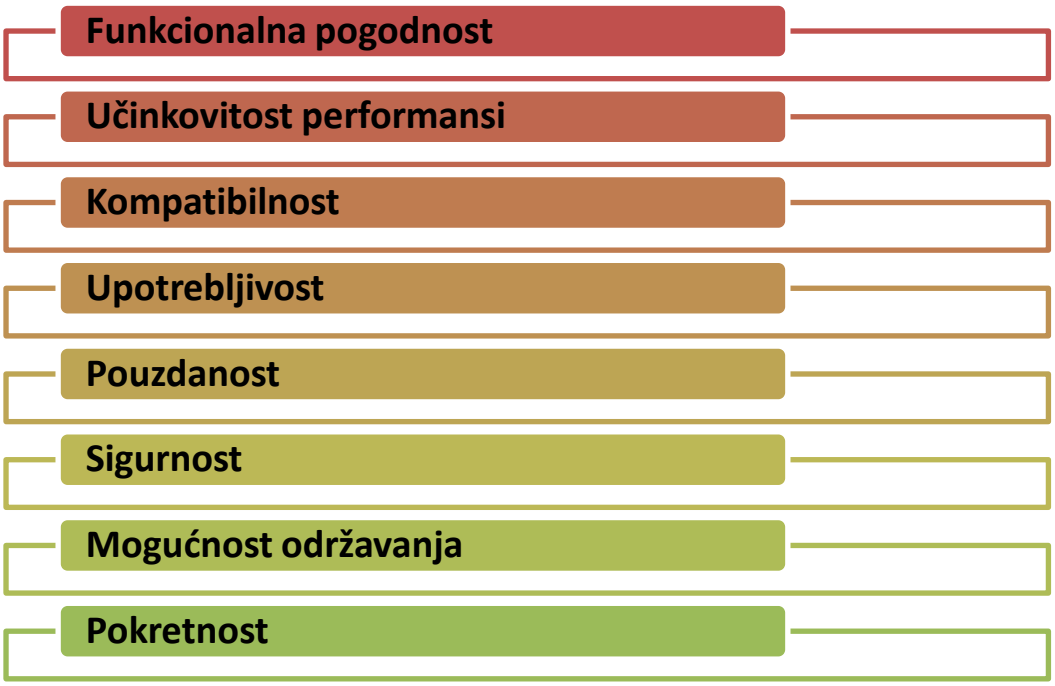
1.Svojstva proizvoda koja je kupac specificirao, naprimjer. želi softver koji je pouzdan, jednostavan za upotrebu, brz, itd.

2.Oslobođenost od nedostataka, odnosno proizvod nema greške, primjerice da softver izvršava se svoje funkcije bez iznimaka.

2.1. ZAHTJEVI KVALITETE ISO 25010: 2011

Model kvalitete proizvoda definiran u ISO/ IEC 25010 sadrži osam karakteristika kvalitete softvera one su prikazane na slici 1 (ISO 25000, n.d.).

Slika 1: Karakteristike kvalitete softvera



Izvor: Prilagođeno prema: ISO 25000 (n.d.)

Karakteristike i potkarakteristike kvalitete softvera (ISO 25000, n.d.):

Funkcionalna pogodnost (eng. Functional Suitability) - ovo svojstvo predstavlja stupanj do kojeg proizvod ili sustav pruža funkcije koje zadovoljavaju navedene i podrazumijevane potrebe kada se koriste u određenim uvjetima. Ova se karakteristika sastoji od sljedećih potkarakteristika:

1. Funkcionalna cjelovitost (eng. Functional completeness). Stupanj kojim skup funkcija obuhvaća sve navedene zadatke i korisničke ciljeve.
2. Funkcionalna ispravnost (eng. Functional correctness). Stupanj do kojeg proizvod ili sustav daje ispravne rezultate s potrebnim stupnjem preciznosti.
3. Funkcionalna prikladnost (eng. Functional appropriateness). Stupanj do kojeg funkcije olakšavaju ostvarenje određenih zadataka i ciljeva.

Učinkovitost performansi (eng. Performance efficiency) - ovo svojstvo predstavlja izvedbu u odnosu na količinu resursa koja se koristi u navedenim uvjetima. Ova se karakteristika sastoji od sljedećih potkarakteristika:

1.Vremensko ponašanje (eng. Time behaviour)**.** Brzina procesiranja i brzina propusnosti proizvoda ili sustava tijekom obavljanja funkcija.

2. Korištenje resursa (eng. Resource utilization)**.** Količina i vrsta resursa koje koristi proizvod ili sustav tijekom obavljanja svojih funkcija.

Kompatibilnost (eng. Compatibility) - Stupanj do kojeg proizvod, sustav ili komponenta može razmjenjivati informacije s drugim proizvodima, sustavima ili komponentama i / ili izvršavati potrebne funkcije, istovremeno dijeleći isto hardversko ili softversko okruženje.

1. Suživot (eng. Co-existence)**.** Stupanj do kojeg proizvod može učinkovito obavljati potrebne funkcije, istovremeno dijeleći okruženje i resurse s drugim proizvodima, bez štetnog utjecaja na bilo koji drugi proizvod.
2. Interoperabilnost (eng. Interoperability**)**. Stupanj do kojeg dva ili više sustava, proizvoda ili komponenata mogu razmjenjivati informacije i koristiti informacije.

Upotrebljivost (eng. Usability) - Stupanj kojem određeni korisnici mogu koristiti proizvod ili sustav da bi postigli zadane ciljeve s učinkovitošću i zadovoljstvom u određenom kontekstu uporabe. Ova se karakteristika sastoji od sljedećih potkarakteristika:

1. Prikladnost prepoznatljivosti (eng. Appropriateness recognizability)**.** Stupanj do kojeg korisnici mogu prepoznati odgovara li proizvod ili sustav njihovim potrebama.
2. Mogućnost učenja (eng. Learnability)**.** Stupanj do kojeg određeni korisnici mogu koristiti proizvod ili sustav kako bi postigli određene ciljeve učenja.
3. Operativnost (eng. Operability)**.** Stupanj kojem proizvod ili sustav ima atribute koji olakšavaju rad i kontrolu.
4. Zaštita od pogreške korisnika (eng. User error protection)**.** Stupanj do kojeg sustav štiti korisnike od pogreške.
5. Estetika korisničkog sučelja (eng. User interface aesthetics)**.** Stupanj kojem korisničko sučelje omogućuje ugodnu i zadovoljavajuću interakciju za korisnika.
6. Pristupačnost (eng. Accessibility**)**. Stupanj do kojeg proizvod ili sustav mogu koristiti ljudi najšireg raspona karakteristika.

Pouzdanost (eng. Reliability) - Stupanj do kojeg sustav ili proizvod obavlja određene funkcije pod određenim uvjetima. Ova se karakteristika sastoji od sljedećih potkarakteristika:

1. Zrelost (eng. Maturity)**.** Stupanj kojem sustav, proizvod ili komponenta zadovoljava potrebe za pouzdanošću u normalnom radu.
2. Raspoloživost (eng. Availability)**.** Stupanj do kojeg su sustav, proizvod ili komponenta operativni i dostupni kada su potrebni za upotrebu.
3. Tolerancija kvarova (eng. Fault tolerance)**.** Stupanj na koji sustav, proizvod ili komponenta djeluju kako je predviđeno unatoč prisutnosti grešaka u hardveru ili softveru.
4. Mogućnost povrata podataka (eng. Recoverability)**.** Stupanj do kojeg proizvod ili sustav može u slučaju prekida ili kvara vratiti podatke koji su izravno pogođeni i ponovno uspostaviti željeno stanje sustava.

Sigurnost (eng. Security) - stupanj do kojeg proizvod ili sustav štiti informacije i podatke tako da osobe, drugi proizvodi ili sustavi imaju stupanj pristupa podacima primjeren njihovim vrstama i razinama autorizacije. Ova se karakteristika sastoji od sljedećih potkarakteristika:

1. Povjerljivost (eng. Confidentiality)**.** Stupanj kojem proizvod ili sustav osigurava dostupnost podataka samo onima ovlaštenim za pristup.
2. Integritet (eng. Integrity)**.** Stupanj kojem sustav, proizvod ili komponenta sprječava neovlašteni pristup ili izmjenu računalnih programa ili podataka.
3. Neosporivost (eng. Non-repudiation)**.** Stupanj do kojeg se može dokazati da su se radnje ili događaji dogodili, tako da se kasnije ne mogu odbiti događaji ili radnje.
4. Odgovornost (eng. Accountability)**.** Stupanj do kojeg se akcije entiteta mogu jedinstveno pratiti.
5. Autentičnost (eng. Authenticity)**.** Stupanj do kojeg se može dokazati identitet subjekta ili resursa.

Mogućnost održavanja (eng. Maintainability) - Ovo svojstvo predstavlja stupanj djelotvornosti i učinkovitosti s kojim se proizvod ili sustav mogu izmijeniti kako bi ga poboljšali, ispravili ili prilagodili promjenama u okruženju. Ova se karakteristika sastoji od sljedećih potkarakteristika:

1. Modularnost (eng. Modularity)**.** Stupanj kojem se sustav ili računalni program sastoji od manjih komponenti tako da promjena jedne komponente ima minimalan utjecaj na ostale komponente.

2. Mogućnost ponovne uporabe (eng. Reusability)**.** Stupanj do kojeg se sredstvo može koristiti u više sustava ili u izgradnji drugih sredstava.
3. Mogućnost modifikacija (eng. Modifiability)**.** Stupanj do kojeg se proizvod ili sustav može učinkovito i djelotvorno izmijeniti bez uvođenja nedostataka ili pogoršanja postojeće kvalitete proizvoda.
4. Mogućnost testiranja (eng. Testability)**.** Stupanj djelotvornosti i učinkovitosti s kojom se mogu utvrditi kriteriji ispitivanja za sustav.

Pokretnost (eng. Portability) - Stupanj djelotvornosti i učinkovitosti s kojim se sustav, proizvod ili komponenta može prenijeti s jednog hardvera, softvera ili drugog operativnog okruženja u drugi. Ova se karakteristika sastoji od sljedećih potkarakteristika:

1. Prilagodljivost (eng. Adaptability)**.** Stupanj kojem se proizvod ili sustav može učinkovito prilagoditi za različita hardverska i softverska okruženja koja se razvijaju.
2. Mogućnost zamjene (eng. Replaceability)**.** Stupanj do kojeg proizvod može zamijeniti drugi navedeni softverski proizvod za istu svrhu u istom okruženju.

2.2. FAKTORI KVALITETE

Svaki softver ima atribute koji ga definiraju. Njih uzimamo kao kriterije procjene te pomoću njih definiramo faktore kvalitete.
Možemo nabrojati faktore kvalitete s pripadajućim atributima procjene (Tablica 2) (Oreški, 1990):

Tablica 1: Faktori kvalitete softvera s pripadajućim atributima procjena

Faktori kvalitete softvera	**Kriteriji procjene softvera**
Korektnost (eng. Correctness)	Sljedivost (eng. Traceability)
	Potpunost (eng. Completeness)
	Dosljednost (eng. Consistency)
Mogućnost održavanja (eng. Maintainability)	Sljedivost (eng. Traceability)
	Dosljednost (eng. consistency)
	Konciznost (eng. Conciseness)
	Modularnost (eng. Modularity)
Pouzdanost (eng. Reliability)	Konciznost (eng. Conciseness)

Efikasnost (eng. Efficiency)	Točnost (eng. Accuracy)
	Tolerancija na pogreške (eng. Error tolerance)
	Jednostavnost (eng. Simplicity)
	Učinkovitost izvršavanja (eng. Execution efficiency)
	Učinkovitost skladištenja podataka (eng. Storage efficiency)
Integralnost (eng. Integrity)	Kontrola pristupa (eng. Access control)
	Revizija pristupa (eng. access audit)
Primjenjivost (eng. Usability)	Operativnost (eng. operability)
	Trening (eng. Training)
	Komunikativnost (eng. Communicativeness)
Mogućnost testiranja (eng. Testability)	Jednostavnost (eng. Simplicity)
	Instrumentacija (eng. Instrumentation)
	Somo-deskripcija (eng. Self-descriptiveness)
Prilagodljivost (eng. Flexibility)	Modularnost (eng. Modularity)
	Općenitost (eng. Generality)
	Proširivost (eng. Expandability)
Prenosivost (eng. Portability)	Modularnost (eng. Modularity)
	Samo-deskripcija (eng. Self-descriptiveness)
	Neovisnost softverskog sustava (eng. Software system independence)
	Neovisnost hardvera (eng. Machine independence)
Obnovljivost (eng. Reusability)	Modularnost (eng. Modularity)
	Općenitost (eng. Generality)
	Neovisnost softverskog sustava (eng.

	Software system independence)
Kooperativnost (eng. Interoperability)	Modularnost (eng. Modularity)
	Zajednička komunikacija (eng. Communications commonality)
	Zajednički podaci (eng. Data commonality)

Izvor: Prilagođeno prema: Oreški (1990)

Pojašnjenje kriterija procjene kojima se opisuju faktori kvalitete (Oreški, 1990):

- **Sljedivost** - mogućnost rekonstrukcije slijeda promjena i utjecajnih faktora iz okoline koji su na promjene utjecali.
- **Kompletnost** - osiguranje potpune implementacije svih svojstva koja su specificirana
- **Dosljednost** - predstavlja primjenu jednostavnih projektantskih notacija i tehnika
- **Točnost** - predstavlja točnost notacije te ponovljivost rezultata
- **Tolerancija na pogreške** - mogućnost rada i pod nepredviđenim uvjetima
- **Jednostavnost** - sustav treba biti implementiran na najrazumljiviji i najjednostavniji način.
- **Modularnost** - Struktura se sastoji od nezavisnih modula
- **Općenitost** - osiguranje generalnosti primjene rješenja
- **Proširivost** - mogućnost povećanja pohrane informacija i podataka te povećanja funkcionalnosti koje će softver obavljati.
- **Samo-deskripcija** - sposobnost jednoznačnog i razumljivog opisa
- **Učinkovitost izvršenja** - izvršavanje uz minimalni utrošak resursa
- **Učinkovitost pohrane** - smanjenje potrebe za vanjskom memorijom
- **Kontrola pristupa i revizija pristupa** - osiguranje kontrole pristupa nad podacima softvera
- **Operabilnost** - formalne procedure i postupci za rad softvera
- **Komunikativnost** - jednostavan način komuniciranja s korisnikom
- **Neovisnost softverskog sustava** - neovisnost softvera od okoline
- **Neovisnost hardvera** - nezavisnost softvera od hardvera
- **Zajednička komunikacija** - uporaba standardnih protokola i rutina za povezivanje
- **Zajednički podaci** - upotreba standardne reprezentacije podataka.

Formula za određivanje rezultata faktora kvalitete (SC_i) (Ošeški, 1990).

$$\boldsymbol{Sci} = \sum_{\boldsymbol{j=1}}^{\boldsymbol{mj}} \boldsymbol{Sij}$$

Mj predstavlja broj kriterija procjene softvera za svaki faktor. *Sij* predstavlja rezultat za *i*-ti faktor kvalitete i *j*-ti kriterij procjene. *Sij* može poprimiti vrijednost od 0 do 5 (Oreški, 1990).

Ukupni rezultat (TSC) možemo dobiti formulom (Oreški, 1990):

$$\boldsymbol{TSC} = \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} \sum_{\boldsymbol{j=1}}^{\boldsymbol{mj}} \boldsymbol{s_{ij}}$$

Uvedemo ponder *w* za koji vrijedi:

$$\sum \boldsymbol{wi} = \boldsymbol{n}$$

Formula za ukupni težinski rezultat(TWSC) glasi (Oreški, 1990):

$$\boldsymbol{TWSC} = \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} \boldsymbol{w_i} \sum_{\boldsymbol{i=1}}^{\boldsymbol{m}} \boldsymbol{m_j}$$

Korisnik na ovaj način može sam dodati težinu određenog faktora kvalitete za koji on smatra da je važan za softver koji promatra (Oreški, 1990).

$$\boldsymbol{TSC_{max}} = \mathbf{5} \sum_{\boldsymbol{i=1}}^{\boldsymbol{m}} \boldsymbol{m_i}$$

TSC_{max} predstavlja maksimalni rezultat za određen broj faktora kvalitete *n* i broj kriterija procjene *m* (Oreški, 1990).

Na kraju dobivamo normalizirani težinski rezultat (NWSC)

$$\boldsymbol{NWSC} = \frac{\boldsymbol{TWSC}}{\boldsymbol{TSC_{max}}}$$

NWSC je unutar intervala 0 i 1. Što je NWSC bliže 1 to je softver kvalitetniji (Oreški, 1990).

$$\mathbf{0} \leq \boldsymbol{NWSC} \leq \mathbf{1}$$

3. TESTIRANJE SOFTVERA

Testiranje softvera definiramo kao proces izvršavanja programa ili aplikacije s namjerom pronalaženja softverske greške. Također se može navesti kao proces provjere valjanosti i provjere programa da bi aplikacija odgovarala poslovnim i tehničkim zadacima te da radi po očekivanjima (Myers, 2004).

„Testiranje softverskih proizvoda se smatra jednim od važnijih procesa razvoja softvera. Testiranje podrazumijeva pokusno izvođenje softvera ili njegovih dijelova na umjetno pripravljenim podacima, uz pažljivo analiziranje rezultata.“ (Myers, 2004). U knjizi „Art of software testing“ Myers ističe da „testiranje i debagiranje potroši 50 posto ukupnog vremena i budžeta stvaranja softvera.“

Myers ističe da se prilikom testiranja softvera previše stavlja naglasak na učinkovitost softvera a nedovoljno na mjerenje pouzdanosti i robusnosti.

Postoje mnogi razlozi koji nam govore zašto je testiranje softvera važan proces i koje su glavne stvari koje bismo trebali uzeti u obzir prilikom testiranja aplikacije i proizvoda.

Testiranje softvera vrlo je važno zbog sljedećih razloga (Myers, 2004)**:**

-Testiranje softvera je potrebno kako bi ukazalo na nedostatke i pogreške koje su napravljene tijekom razvojne faze

-Bitno je jer osigurava pouzdanost i zadovoljstvo korisnika koji je primjenjuju.

-Vrlo je važno u osiguravanju kvalitete proizvoda. Kvalitetan proizvod isporučen klijentima pomaže u stjecanju njihovog povjerenja.

-Testiranje je potrebno kako bi krajnjim korisnicima isporučili kvalitetan proizvod koji zahtjeva niže troškove održavanja i rezultira točnim, dosljednim i pouzdanim rezultatima.

-Testiranje je potrebno za učinkovitu izvedbu softverske aplikacije.

-Važno je osigurati da aplikacija ne uzrokuje nikakve propuste jer će se propusti skupo platiti u kasnijim fazama razvoja softvera.

3.1. NAČELA TESTIRANJA

Postoji sedam načela testiranja. Oni su sljedeći (Guru, n.d.):

1. Testiranje prikazuje prisutnost pogrešaka. Testiranje aplikacije može otkriti jedan ili više nedostataka u primjeni, međutim samo testiranje ne može dokazati da je aplikacija bez grešaka. Stoga je važno osmisliti testne slučajeve u kojima se može pronaći što više nedostataka.

2. Iscrpno testiranje je nemoguće: osim ako aplikacija koja se testira ima vrlo jednostavnu logičku strukturu i ograničen unos. Nije moguće testirati sve moguće kombinacije podataka i scenarija. Stoga se rizici i prioriteti najviše usredotočuju na važne aspekte.

3. Rano testiranje: Čim su zahtjevi za razvoj proizvoda dostupni i sama dokumentacija dizajna, možemo početi s testiranjem. Rano testiranje je veoma bitno jer kada se nađu greške u ranijem životnom ciklusu lakše i jeftinije ih je popraviti.

4. Grupiranje grešaka. Grupiranje pogrešaka navodi da mali broj modula sadrži većinu otkrivenih nedostataka. Oko 80% problema se nalazi u 20% modula. Ako se isti testni slučajevi ponavljaju iznova i iznova, na kraju u istim testnim slučajevima više neće biti bugova.

5. „Paradoks pesticida“. Ponovna upotreba iste mješavine pesticida za iskorjenjivanje insekata će dovesti do toga da kukci s vremenom razviju otpornost na pesticid, čime će pesticid izgubiti djelovanje na insekte. Isto vrijedi i za testiranje softvera. Ako se provodi isti skup ponavljajućih testova, metoda će biti beskorisna za otkrivanje novih pogrešaka. Da bi se to prevladalo, testne slučajeve je potrebno redovito pregledavati i revidirati, dodajući nove i različite testne slučajeve kako bi se pronašlo više grešaka.

6. Testiranje ovisi o kontekstu. Testiranje je ovisno o kontekstu što u osnovi znači da će način na koji se testira web-lokacija za e-trgovinu biti različita od načina na koji se testira reklama na polici dućana. Sve aplikacije nisu identične. Trebao bi se koristiti drugačiji pristup, metodologije, tehnike i vrste testiranja ovisno o vrsti aplikacije.

7. Nepostojeće pogreške - zabluda. Moguće je da je softver koji je 99% bez bugova i dalje neupotrebljiv. To može biti slučaj ako je sustav temeljito testiran zbog pogrešnog zahtjeva. Testiranje softvera nije samo pronalaženje pogrešaka, nego i provjera dali se softver bavi poslovnim problemima. Pronalaženje i popravljanje nedostataka ne pomaže ako je izgradnja sustava neupotrebljiva i ne ispunjava potrebe i zahtjeve korisnika.

3.2. POVIJEST TESTIRANJA SOFTVERA

Testiranje je kao pojam odvojen od debuggiranja 1979 godine je uveo Glenford J.Myers (Pranić, 2010).

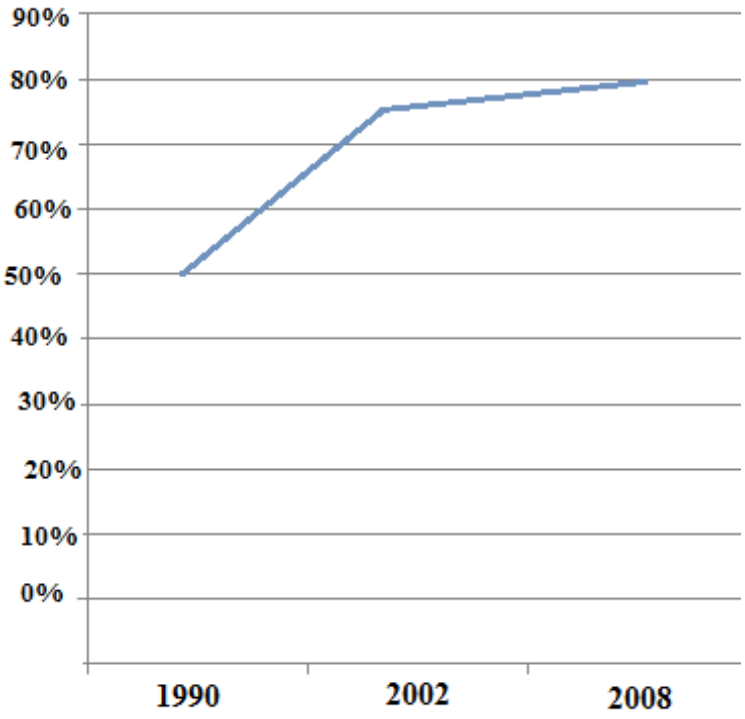
Dave Gelperin i William C. Hetzel svrstavaju ciljeve i faze u testiranju (Elfriede Dustin, n.d.):

- Do 1956 - testiranje orijentirano debuggingu
- 1957-1987 - testiranje orijentirano na demonstraciju
- 1979-1982 - testiranje je orijentirano destrukciji
- 1983-1987 - testiranje se orijentira na procjeni
- 1988-2000 - testiranje je usmjereno prevenciji

Kroz povijest se mijenjalo i vrijeme koje se trošilo na testiranje. Sve više se troši vremena na postupak testiranja softvera. Beizer je 1990.rekao „Pola truda u razvoju softvera bi se trebalo potrošiti na testiranje softvera“. 2002: Hailpern and Santhanam u knjizi „Software debugging, testing, and verification“ navode „postupak otklanjanja pogrešaka, testiranja, validacije uzima između 50 i 75 posto troška razvoja softvera“. 2008: Redmond Developer News kaže „Analitičari procjenjuju da su developeri potrošiti samo oko 20 posto svog vremena na dizajniranje i kodiranje, većina vremena je utrošena na rješavanje problema aplikacije.

Na slici 2 vidimo dijagram koji pokazuje kako se mijenjala važnost testiranja kroz povijest.

Slika 2: Vrijeme utrošeno na testiranje



Izvor:Prilagođeno prema: Elfriede Dustin (n.d.)

3.3. PSIHOLOGIJA TESTIRANJA SOFTVERA

Glenford J.Myers, u knjizi „Art of Software testing“ odvaja testiranje od debuggiranja. Myers navodi da je jedan od glavnih uzoraka lošeg testiranja programa činjenica da većina programera počinje s definicijom pojma: „testiranje je proces dokazivanja da pogreške nisu prisutne“ ili „Svrha testiranja je pokazati da program ispravno izvršava svoje funkcionalnosti“. Navodi da kada se program testira, želi mu se dodati neka vrijednost. Dodavanje vrijednosti kroz testiranje znači podizanje kvalitete ili pouzdanosti programa. Povećanje pouzdanosti programa znači pronalaženje i uklanjanje pogrešaka. Stoga ne testiramo program da bi pokazali da on radi; umjesto toga bi trebalo započeti s pretpostavkom da program sadrži pogreške (pretpostavka za gotovo svaki program), a zatim testirati program kako bi pronašao sve je više moguće pogrešaka.

Myers definira testiranje kao proces izvršavanja programa s namjerom pronalaženja pogrešaka. Razumijevanje prave definicije testiranja softvera može napraviti veliku razliku u uspjehu samog procesa testiranja. Ako je cilj pokazati da program nema pogrešaka, onda ćemo podsvjesno biti usmjereni prema tom cilju, tj. bit ćemo skloni odabrati testne podatke

koji imaju malu vjerojatnost uzrokovanja neuspjeha programa. S druge strane, ako je naš cilj pokazati da program ima pogreške, testni podaci imat će veću vjerojatnost pronalaženja pogrešaka. Taj pristup će dati programu veću vrijednost nego prethodni pristup. Myers navodi kako bi trebalo bolje analizirati upotrebu riječi „ uspješno“ i „neuspješno“, posebno od strane menadžera projekta u kategorizaciji rezultata testnih slučajeva. Većina menadžera testni slučaj koji nije pronašao pogrešku definira kao „uspješan test“, dok se test koji otkriva pogrešku naziva „neuspješnim“. „Neuspješno“ označava nešto razočaravajuće tj. nepoželjno. Mayers navodi kako je dobro proveden i konturiran test dijela softvera uspješan kada pronađe pogreške koje se mogu popraviti. Taj isti test je uspješan kada se konačno utvrdi da nema više pogrešaka. Neuspješan test je onaj koji ne ispituje ispravno softver, u većini slučajeva to je test koji nije pronašao pogreške, budući da je koncept programa bez pogrešaka u osnovi nerealan.

3.4. TEST SLUČAJ (TEST CASE)

Testni slučaj se definira kao skup radnji koje se izvršavaju radi provjere određenog svojstva ili funkcionalnosti softverske aplikacije. Testni slučaj nezamjenjiva je komponenta softvera za provjeru životnog ciklusa softvera koji pomaže u provjeri aplikacije pod testom.
Tipični slučajevi obično trebaju biti mali, izolirani i atomski. Ispitne slučajeve trebalo bi biti lako razumjeti, a korake treba brzo izvršavati. Oni bi trebali biti međusobno neovisni i izvršavati se neovisno jedan od drugog (Softwaretestingclass, n.d.)

Prednosti pisanja testnih slučajeva (Softwaretestingclass, n.d.)**:**

- Testni slučajevi ostvaruju dobru pokretljivost testom
- Pomaže u poboljšanju kvalitete softvera
- Smanjuje troškove održavanja i podrške za softver
- Pomaže provjeriti zadovoljava li softver zahtjeve krajnjeg korisnika
- Omogućuje testeru da dobro razmisli i pristupi testovima iz što više kutova

4. VRSTE TESTIRANJA KROZ ŽIVOTNI CIKLUS SOFTVERA

Testove grupiramo po tome kako ih dodajemo u procesu razvoja softvera ili po razini specifičnosti testa (slika 3).

Slika 3: Vrste testiranja kroz životni ciklus softvera

Izvor:Prilagođeno prema: Guru (n.d.)

4.1. TESTIRANJE JEDINICE (UNIT TEST)

Testiranje jedinice verificira funkcioniranje u izolaciji softverskih dijelova. Ovisno o kontekstu, to mogu biti individualni potprogrami ili veće komponente kreirane od tijesno povezanih jedinica. Testiranje jedinice provjerava funkcioniranje svakog dijela koda i to najčešće na razini funkcije. U objektno-orijentiranom programu, to je na razini klase. Testiranje jedinice osigurava da pojedine komponente softvera mogu raditi zasebno, ali nam ne može provjeriti funkcionalnost cijelog softvera (Pranić, 2012).

4.2. INTEGRACIJSKO TESTIRANJE

Integracijsko testiranje je proces verificiranja interakcije između softverskih komponenti. Cilj integracijskog testiranja je provjeriti sučelja među komponentama. Komponente softvera mogu se integrirati sve zajedno ili iterativnim načinom. Klasične strategije integralnog testiranja su top-down ili bottom-up, one se koriste kod tradicionalnog, hijerarhijski strukturiranog softvera. Moderne strategije su arhitektonski strukturirane, što podrazumijeva integriranje softverskih komponenti na bazi identificiranih funkcionalnih dijelova. Integracijsko testiranje obično se izvodi tako da se sve komponente testiraju zajedno u istom trenutku (Guru, n.d.).

4.3. REGRESIJSKO TESTIRANJE

Regresijsko testiranje je selektivno re-testiranje sistema ili komponenti radi verifikacije da bi provjerili dali su promjene uzrokovale neželjene efekte tj. da bi dokazali da je softver koji je prethodno položio test i dalje radi. Regresije najčešće nastaju kao posljedica nenamjernih promjena programa. Regresijsko ispitivanje će ovisiti o fazi izdavanja procesa i o riziku

dodanih značajki. Regresivno testiranje se može provesti na svakom test nivou. Promjene koje su dodane kasno u proces smatraju „promjenama velikog rizika“, promjene koje su dodaju u ranijim procesima razvoja softvera su „promjene niskog rizika“. Regresivni test je veoma bitan kada postojeći sistem trebamo zamijeniti novim. Regresivni testovi nam garantiraju da su performanse novog sistema jednake performansama starog sistema (Guru, n.d.).

4.4. TEST PRIHVATLJIVOSTI

Test prihvatljivosti provjerava ponašanje softvera da bi utvrdili da li odgovara zahtjevima kupaca. Kupac podnosi tipične zadatke da bi provjerio dali su njegovi zahtjevi ispunjeni, najčešće u svojem laboratorijskom okruženju na vlastitom hardveru (Guru, n.d.).

4.5. ALFA TESTIRANJE

Alfa testiranje je vrsta testiranja prihvatljivosti; da li su se identificirali svi postojeći problemi/ pogreške prije objavljivanja proizvoda kupcima ili javnosti. Fokus ovog testa je simulirati stvarne korisnike pomoću crne kutije i bijele kutije. Alfa testiranje se provodi u laboratorijskom okruženju i obično su testeri interni zaposlenici organizacije. Ovo testiranje se naziva alfa zato što je se provodi rano, pri kraju razvoja softvera i prije beta testiranja (Guru, n.d.).

4.6. BETA TESTIRANJE

Beta testiranje softvera obavljaju „pravi korisnici“ u „stvarnom okruženju“. Beta testiranje se može smatrati oblikom eksternog prihvaćanja korisničkog testiranja. Beta verzija softvera se objavljuje ograničenom broju krajnjih korisnika proizvoda kako bi se dobila povratna informacija o kvaliteti proizvoda. Beta testiranje smanjuje rizike kvara proizvoda te osigurava povećanu kvalitetu proizvoda kroz provjeru valjanosti korisnika. Beta testiranje je konačni test prije slanja proizvoda kupcima. Glavna prednost beta testiranja je izravna povratna informacija od korisnika (Guru, n.d.).

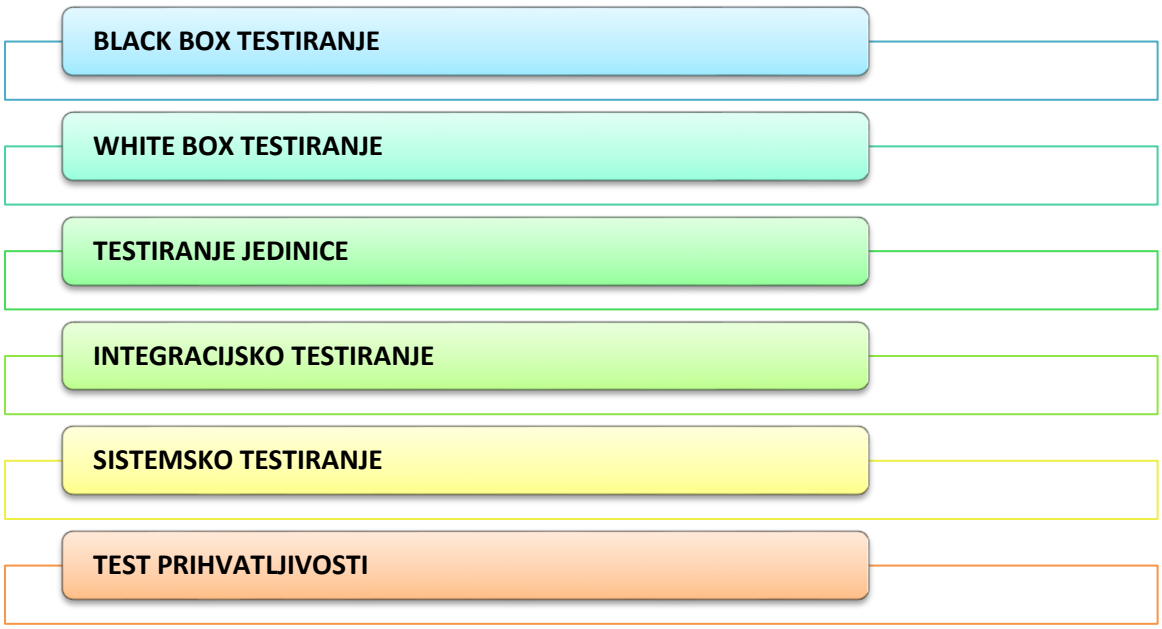
5. MANUALNO TESTIRANJE

Manualno testiranje je vrsta testiranja softvera gdje testeri ručno izvršavaju ispitne slučajeve bez korištenja alata za automatizaciju.

Svaka nova aplikacija se mora ručno testirati prije automatskog testiranja. Ručno testiranje zahtjeva mnogo napora, ali je u većini slučajeva potrebno za provjeru izvedivosti automatizacije. Ručno testiranje ne zahtijeva poznavanje alata za testiranje.
Ključni koncept manualnog testiranja je osigurati da je aplikacija bez pogrešaka te da radi u skladu s navedenim funkcionalnim zahtjevima. Također osigurava da su prijavljeni nedostaci ispravljeni od strane developera, ponovno testiranje obavljaju testeri kako bi potvrdili da su pogreške ispravljene. Manualno testiranje provjerava kvalitetu softvera i omogućava isporuku proizvoda kupcu bez bugova (Guru, n.d.).

5.1. VRSTE MANUALNOG TESTIRANJA

Na slici 4 prikazane su vrste manualnog testiranja.
Slika 4:Vrste manualnog testiranja



Izvor:Prilagođeno prema: Guru (n.d.)

5.1.1. BLACK BOX TESTIRANJE

Black box je tehnika testiranja u kojoj se testira funkcionalnost softvera bez sagledavanja interne strukture koda, detalja implementacije. Ova vrsta testiranja se temelji isključivo na zahtjevima i specifikacijama softvera. U BlackBox testiranju fokus je na ulazu i izlazu, a da se ne bavimo internim znanjem softvera (Guru, n.d.).

Slika 5: Black Box Testiranje

Izvor: Prilagođeno prema: Guru (n.d.)

5.1.2. WHITE BOX TESTIRANJE

Za razliku od BlackBox testiranja čiji je fokus na funkcionalnostima softvera, WhiteBox testiranje se temelji na unutarnjem funkcioniranju aplikacije i svodi se na unutarnje testiranje. WhiteBox testiranje se definira kao testiranje unutarnje strukture, dizajna i kodiranja softverskog koda. Programski kod je kod ove vrste testiranja vidljiv testeru. Glavni fokus ove metode testiranja je na provjeri protoka ulaza i izlaza kroz aplikaciju, poboljšanje dizajna i upotrebljivosti te jačanje sigurnosti. WhiteBox testiranje obično provode programeri (Guru, n.d.).

6. AUTOMATIZIRANO TESTIRANJE

Automatizirano testiranje podrazumijeva korištenje alata za automatizaciju koje izvršava testiranje softvera.

Korištenje automatiziranog testiranja softvera pomaže u izbjegavanju pogrešaka koje čine ljudi. Često dolazi do pogreške kada se ljudi umore od ponavljanja istog procesa. Automatizirani alati testiranja će omogućiti točno pohranjivanje podataka ispitivanja u bazu te će izbjeći greške koje mogu nastati manualnim testiranjem. Rezultati se mogu automatski unositi u bazu podataka i mogu se koristiti za pružanje korisne statistike o tome kako napreduje proces razvoja softvera (Rampure, n.d.).

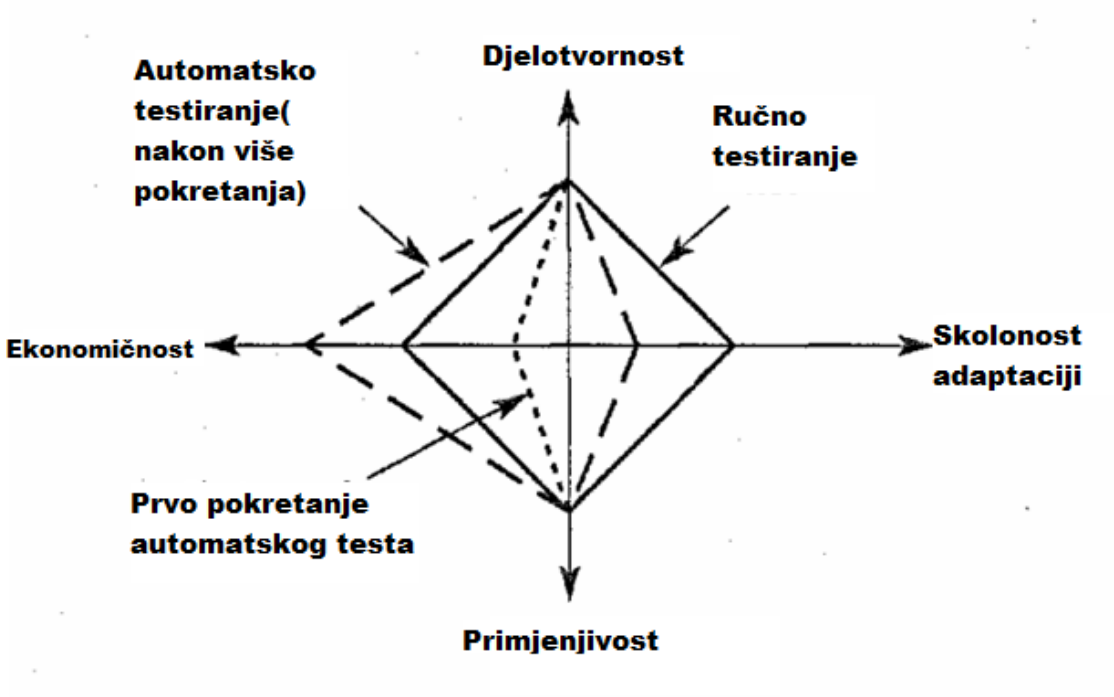
Automatizirano testiranje zahtjeva znatna ulaganja novaca i resursa. Uzastopni razvojni ciklusi zahtijevat će ponavljanje istog skupa testova. Pomoću alata za automatizaciju testiranja moguće je snimiti paket za testiranje i ponovno ga reproducirati prema potrebi. Jednom kada je paket za testiranje automatiziran, nije potrebna nikakva ljudska intervencija.

Cilj automatizacije je smanjiti broj testnih slučajeva koji se ručno pokreću, a ne eliminirati ručno testiranje (Rampure, n.d.).

6.1. AUTOMATIZACIJA TESTIRANJA

Automatizacija testiranja je vještina veoma različita od testiranja. Mnoge organizacije se začude kada vide da je skuplje automatizirati testiranje nego ga jednom izvršiti ručno. Da bi se dobila korist od automatizacije testiranja, testovi koji se automatiziraju moraju biti pažljivo odabrani i implementirani. Automatizirana kvaliteta neovisna je o kvaliteti ispitivanja. Jednom implementiran automatizirani test je općenito mnogo ekonomičniji, a trošak napora je samo djelić napora ručnog vođenja. Međutim, automatizirani testovi općenito koštaju više za stvaranje i održavanje. Na slici 6 prikazana su četiri atributa kvalitete testnog slučaja u Kevinovom dijagramu. Manualno testiranje prikazano je čvrstim crtama. Kada je test prvi puta automatiziran, on će biti manje ekonomičniji (budući da je potrebno mnogo truda da bi se automatizirao). Ali nakon što je automatizirani test pokrenut mnogo puta postat će mnogo isplativiji od testa izvedenog ručno. Osoba koja stvara i održava artefakte povezane s korištenjem alata za automatizirano testiranje je testni automatizator, on može ali i ne mora biti član testnog tima. Moguće je dobro ili loše ispitivanje kvalitete. To je zadatak testera koje određuje kvalitetu ispitivanja. Također je moguće imati dobru ili lošu kvalitetu automatizacije, određivanje kvalitete automatizacije testiranja je zadatak testnog automatizatora. Testni atotomatitazator određuje kako će se lako dodati novi automatizirani testovi, kako će se automatizirani testovi održavati te na kraju krajeva koje će prednosti automatiziranje testiranja pružiti samom testiranju (Fewster i Graham, 1994).

Slika 6: Ekonomska isplativost automatiziranog testiranja

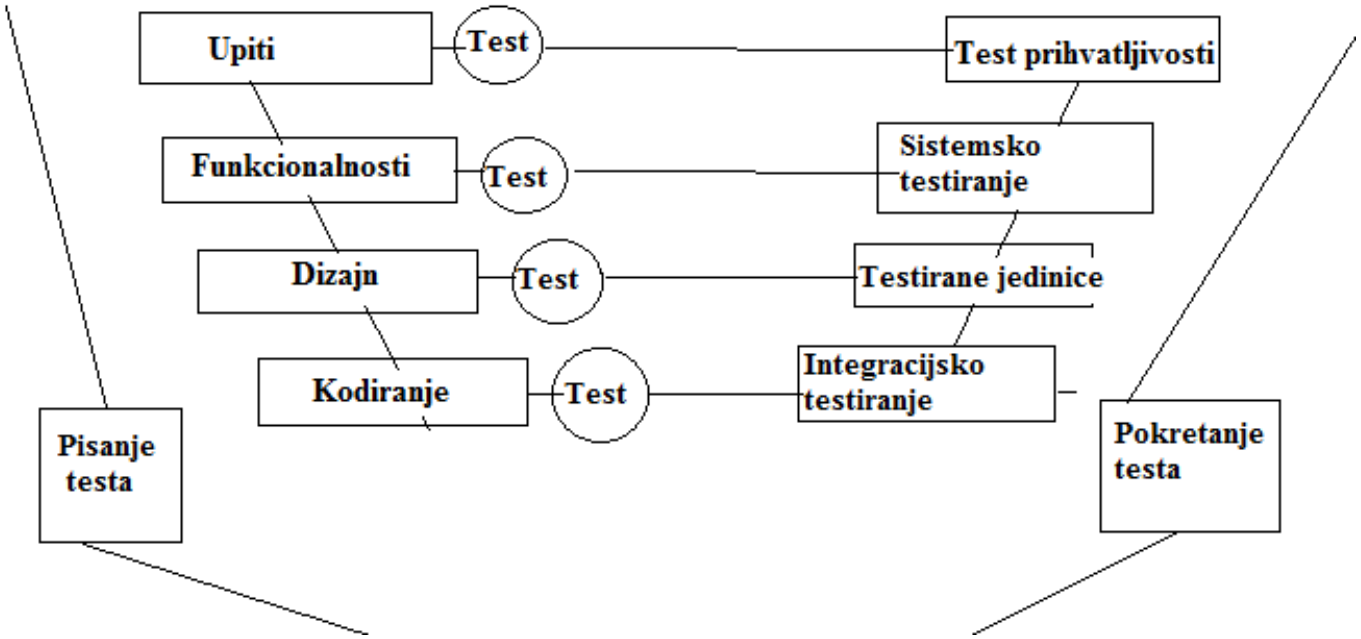


Izvor: Prilagođeno prema: Fewster i Graham (1994)

6.2. V-MODEL

V-model razvoja softvera ilustrira kada se trebaju obaviti aktivnosti testiranja. V-model pokazuje da svaka razvojna aktivnost ima odgovarajuću testnu aktivnost. Na slici 7 vidimo pojednostavljeni V-model s četiri razine razvoja i aktivnosti testiranja. Najvažniji čimbenik za uspješnu primjenu V-modela je pitanje kada su dizajnirani testni slučajevi. Aktivnost projektiranja testa uvijek pronalazi nedostatke u odnosu na testove protiv kojih je dizajniran. Na primjer, projektiranje prihvatljivih ispitnih slučajeva naći će nedostatke u zahtjevima, projektiranje slučajeva testiranja sustava naći će nedostatke u funkcionalnoj specifikaciji, dizajniranje slučajeva integracijskih testova naći će nedostatke u dizajnu i projektiranju, jedinični test slučajevi naći će nedostatke u kodu. Dizajn testa ne mora čekati početak ispitivanja to može biti učinjeno u bilo kojem trenutku nakon što informacije na kojima se ti testovi temelje postaju dostupni. Tada je pronalaženje pogrešaka korisno, a ne destruktivno, jer se defekti mogu ispraviti prije nego što im se poveća broj (Fewster i Graham, 1994).

Slika 7: V-model razvoja softvera



Izvor: Prilagođeno prema: Fewster i Graham (1994)

6.3. ZAŠTO SE ODLUČITI ZA AUTOMATIZIRANO TESTIRANJE?

-Manualno testiranje troši puno vremena i novaca

-Teško je ručno testirati softver pisan na više jezika

-Automatizacija ne zahtijeva ljudski intervenciju (automatizirani test se može pokrenuti bez nadzora)

-Automatizacija povećava brzinu izvršavanja testa

-Automatizacija pomaže povećati pokrivenost testiranja

-Ručno testiranje je sklono ljudskoj pogrešci zbog ponavljanja iste stvari (Guru,n.d.).

6.4. TEST SLUČAJEVI KOJE JE DOBRO I TEST SLUČAJEVI KOJE NIJE DOBRO AUTOMATIZIRATI

Test slučajevi koji su pogodni za automatizaciju (Guru,n.d.)**:**

-Visoko rizični test slučajevi

-Test slučajevi koji se ponavljaju

-Test slučajevi koji se teško izvode ručno

-Test slučajevi koji zahtijevaju puno vremena

Test slučajevi koji nisu pogodni za automatizaciju su sljedeći (Guru,n.d.)**:**

-Test slučajevi koji su novo dizajnirani i ne izvršavaju se ručno barem jedanput

-Test slučajevi čiji se zahtjevi često mijenjaju

-Test slučajevi koji se ne izvode na ad-hoc osnovi.

6.5. FAZE AUTOMATIZIRANOG TESTIRANJA

Na slici 8 vidimo korake od kojih se sastoji automatizirano testiranje (Guru,n.d.):

Slika 8: Faze automatiziranog testiranja

Izvor: Prilagođeno prema: Guru (n.d.)

Odabir alata za testiranje (eng. Test Tool selection) - odabir alata za testiranje uvelike će ovisiti o tehnologiji na kojoj se temelji aplikacija koja se testira. Tako na primjer alat QTP ne podržava Informatica softver (Guru,n.d.).

Definiranje opsega automatizacije (eng. Define the scope of Automation) - određivanje područja automatizacije. Područje automatizacije je područje aplikacije pod testom koje će biti automatizirano.

Sljedeće stavke pomažu u određivanju opsega (Guru,n.d.):

- Značajke koje su važne za poslovanje
- Scenariji koji imaju veliku količinu podataka
- Tehnička izvedivost
- Složenost tehničkih slučajeva

- Sposobnost korištenja istih testnih slučajeva za unakrsno ispitivanje

Planiranje,dizajn i razvoj (eng. Planning, Design, and Development) - tijekom ove faze definira se strategija i plan automatizacije koji sadrži sljedeće pojedinosti (Guru,n.d.):

- Odabrane alate za automatizaciju
- Dizajn okvira i njegove značajke
- Automatizacija unutar i izvan dosega
- Automatska priprema područja koji se testira
- Raspored, vremenska skripta
- Isporuka rezultata automatizacije

Izvršavanje testa (eng. Test Execution) - tijekom ove faze izvršavaju se skripte automatizacije. Skripte trebaju ulazne testne podatke prije postavljanja testa na pokretanje. Nakon izvršenja testa pružaju se detaljna izvješća o testiranju. Izvršenje testa se može izvesti izravno ili pomoću alata „Test Management“ koji će pozvati alat za automatizaciju (Guru,n.d.).

Održavanje (eng. Maintenance) - kako se u sustavu dodaju nove funkcionalnosti pod testom s uzastopnim ciklusima, automatizirane skripte treba dodati, pregledati i održavati za svaki ciklus izdavanja. Održavanje je nužno kako bi se poboljšala učinkovitost skripti za automatizaciju (Guru,n.d.).

6.6. OKVIR ZA AUTOMATIZACIJU

Okvir je skup smjernica za automatizaciju koje pomažu u (Guru,n.d.):
-Održavaju dosljednosti testiranja
-Poboljšanja strukture testa
-Minimalna upotreba koda
-Poboljšanje ponovnog korištenja
-Vrijeme upotrebe alata može se smanjiti
-Uključuje podatke gdje god je to prikladno

Vrste okvira za automatizaciju (Guru,n.d.)**:**

Automatizacijski okvir temeljen na podacima-

- Pohranjuje testne podatke u tablicu ili u oblik proračunske tablice. To omogućava inženjerima za automatizaciju da imaju jedinstvenu testnu skriptu koja može izvoditi testove za sve testne podatke u tablici.
- U ovom okviru ulazne vrijednosti su očitane iz podatkovnih datoteka i pohranjene su u varijablu u testnim skriptama.
- Ulazni podaci se mogu pohraniti u jedan ili više izvora podataka poput xls, xml, csv i baze podataka.
- Okvir za testiranje temeljen na podacima pogodan je za korištenje kada imamo više skupova podataka na kojima trebamo pokrenuti iste testove, jer se ista testna skripta može izvršiti za različite kombinacije ulaznih testnih podataka i generirati rezultate ispitivanja.

Automatizirani okvir za ključne riječi

- Tehnika skriptiranja koja koristi datoteke podataka koje sadrže ključne riječi povezane s aplikacijom koja se testira. Ključne riječi opisuju skup radnji koje su potrebne za izvršavanje određenog koraka.
- Sastoji se od ključnih riječi visoke i niske razine, uključujući argumente ključnih riječi, koji su sastavljeni da opisuju radnju testnog slučaja.
- U testiranju po ključnim riječima prvo se identificira skup ključnih riječi, a zatim se pridružuje radnjama (funkcijama) koja se odnosi na te ključne riječi (Guru,n.d.)**.**

Modularni okvir za automatizaciju

- Dijeli automatizirane testne funkcije na logičke segmente ili module. Moduli su izolirani kako bi stvorili neovisne testove koji se mogu rekombinirati kako bi slijedili plan ispitivanja
- Prednost ovog okvira je podesivost jer pojedinačne promjene u jednom djelu testirane aplikacije ne utječu na sve testove za aplikaciju, već samo na pojedinačne testove.

- Nedostatak ove vrste okvira je složenost. Testovi za svaki modul sadrže ugrađene podatke, tako da se moraju izvršiti promjene na testovima ako se zahtijevaju različiti skupovi podataka (Smartsheet, n.d.).

6.7. PREDNOSTI AUTOMATSKOG TESTIRANJA

Automatizacija testiranja može omogućiti da se testni zadaci obavljaju mnogo učinkovitije nego kada se testiranje obavlja manualno. Osim učinkovitosti postoje mnoge prednosti automatiziranog testiranja, a to su (Fewster i Graham, 1994):

- Pokretanje postojećih (regresijskih) testova na novoj verziji programa. Ovo je jedan od najvažnijih zadataka, osobno u okruženju u kojem se softver često mijenja. Napor uključen u izvođenje skupa regresijskih testova trebao bi biti minimalan.
- Često izvršavanje istog testa jedna je od koristi automatizacije, odnosno mogućnost da se pokrene više testova u manje vremena, stoga se oni mogu pokretati češće.
- Može obaviti testove koje bi bilo teško ili nemoguće napraviti ručno. Provesti test uživo na mreži od 200 korisnika može biti nemoguće, ali unos od 200 korisnika može biti u sistemu automatiziranog testa. Zato što se krajnji korisnici koji definiraju testove repliciraju automatski. Testovi korisničkih scenarija mogu se pokrenuti u bilo kojem trenutku i može ga pokrenuti tehničko osoblje koje ne razumije u zamršenost cjelokupnog testa.
- Bolje korištenje resursa tj. automatiziranje ručnih i dosadnih zadataka, kao što je unos istih testnih ulaza, poboljšava radnu sposobnost i oslobađa kvalificirane testere obavljanja takvih zadataka te oni mogu uložiti više truda u osmišljavanje boljih testnih slučajeva.
- Dosljednost i ponovljivost ispitivanja. Testovi koji se ponavljaju, svaki puta će se točno ponoviti, to daje razinu dosljednosti koju je teško postići ručnim testiranjem. Uvođenjem automatiziranog testiranja možemo osigurati dosljedne standarde kako u testiranju tako i u samom razvoju. Na primjer, alat može provjeriti da li je implementiran isti tip značajke u svakoj aplikaciji.

- Ponovno korištenje testova. Napor uložen u odlučivanje što će se testirati, dizajniranje testova, izgradnju testova može se smanjiti automatizacijom jer ćemo isti test moći koristiti više puta.
- Softver će prije završiti na tržištu. Nakon što je test automatiziran, ponavljanja se obavljaju mnogo brže nego što bi se testiranje ručno obavljalo. Stoga se proteklo vrijeme za testiranje može skratiti.
- Povećano povjerenje - znajući da je opsežan skup automatiziranih testova uspješno pokrenut, može postojati veće povjerenje da neće biti neugodnih iznenađenja kada se softver otpusti u prodaju, pod uvjetom da su testovi koji se provode dobri testovi.

6.8. UOBIČAJENI PROBLEMI AUTOMATIZACIJE TESTIRANJA

Postoje brojni problemi koji se mogu pojaviti u pokušaju automatizacije testiranja. U nastavku su navedeni neki od tih problema (Fewster i Graham, 1994):

- Nerealna očekivanja. Postoji tendencija optimizma o tome što se može postići novim alatom. Dobavljači naglašavaju prednosti i uspjehe koji mogu smanjiti količinu napora potrebnog za postizanje trajne koristi. Ako su očekivanja uprave nerealna, onda, bez obzira koliko dobro se provodi alat s tehničkog stajališta, neće ispuniti očekivanja.
- Loša praksa testiranja. Ako je praksa testiranja loša, sa slabo organiziranim testovima, malom ili nedostatnom dokumentacijom i testovima koji nisu dobri u pronalaženju grešaka, automatizacija testiranja nije dobra ideja. Mnogo je bolje unaprijediti učinkovitost testiranja, nego poboljšati učinkovitost lošeg testiranja. Onda, bez obzira na to koliko se dobro alat provodi s tehničkog stajališta, neće ispuniti očekivanja.
- Očekivanje da će automatizirani testovi naći mnogo novih nedostataka. Alati za izvršavanje testova su alati za ponovnu reprodukciju tj. alati za regresijsko testiranje. Njihova svrha je uporaba u ponavljanju testova koji su već pokrenuti. To je veoma korisna stvar, ali nije vjerovano da će se naći veliki broj novih grešaka, pogotovo kada se pokreću na istom hardverskom i softverskom okruženju kao i prije. Testovi koji ne pronalaze pogreške nisu bezvrijedni, iako dobar test dizajn bi trebao biti usmjeren na pronalaženje pogrešaka. Znajući da je niz testova ponovno prošao, daje povjerenje da softver još uvijek radi dobro kao i prije, te da promjene drugdje nisu imale nepredvidljive učinke.

- Lažan osjećaj sigurnosti. Samo zato što paket za testiranje radi bez pronalaženja pogrešaka, to ne znači da nema pogrešaka u softveru. Testovi mogu biti nepotpuni ili sami mogu sadržavati nedostatke. Ako su očekivani rezultati netočni, automatizirani testovi će jednostavno zadržati te neispravne rezultate na neodređeno vrijeme.
- Održavane automatiziranih testova. Kada se softver promjeni često je potrebno ažurirati neke, ili čak sve testove, kako bi se oni uspješno mogli pokrenuti. To posebno vrijedi za automatizirane testove. Kada je potrebno više napora za ažuriranje testova nego što bi bilo potrebno za ponovno pokretanje tih testova ručno, test automatizacije bit će napušten.
- Tehnički problemi. Komercijalni alati za izvršavanje testova su softverski proizvodi koje prodaju tvrtke dobavljači. Kao softverski proizvodi teće strane, oni nisu imuni na nedostatke ili probleme pogreške. Nerijetko se događa da alat za testiranje nije dobro testiran. Interoperabilnost alata s drugim softverom, bilo vlastitim aplikacijama ili proizvodima trećih strana, može biti ozbiljan problem. Tehnološko se okruženje mijenja tako brzo da je teško prodavačima da drže korak. Mnogi alati izgledaju idealno na papiru ali jednostavno ne uspijevaju raditi u nekim okruženjima. Komercijalni alati za izvršavanje testova su veliki i složeni proizvodi, a potrebno je detaljno tehnološko znanje kako bi se dobilo najbolje od alata. Obuka koju isporučuje dobavljač ili distributer ključna je za sve one koji će koristiti alat izravno, posebno testni automatizatori (osobe koje automatiziraju testove). Osim tehničkih problema sa samim alatom, može doći do tehničkih problema sa softverom koji se testira. Ako softver nije osmišljen i izrađen s obzirom na mogućnost testiranja, može ga biti vrlo teško testirati, ručno ili automatski. Pokušaj upotrebe alata za testiranje takvog softvera dodatna je komplikacija koja može samo otežati testiranje.
- Organizacijska pitanja. Automatizirano testiranje nije trivijalna vježba i mora biti dobro podržano od strane menadžmenta i implementirano u kulturu organizacije. Potrebno je izdvojiti vrijeme za odabir alata, za obuku, za eksperimentiranje i učenje što najbolje funkcionira za promicanje korištenja alata unutar organizacije.

6.9. OGRANIČENJA AUTOMATIZACIJE TESTIRANJA

Uz mnoge prednosti koje nudi automatizacija testiranja, dolaze i mnogi problemi ako postupak uvođenja alata za automatizaciju nije pomno promišljen. Uvijek će biti testiranja koje je mnogo lakše napraviti ručno nego automatski, ili je tako teško automatizirati da nije ekonomično. Testovi koji vjerojatno ne bi trebali biti automatizirani uključuju testove koji:

- Testovi koji se izvode samo rijetko. Na primjer, ako se test provodi samo jednom godišnje, vjerovano nije isplativo automatizirati taj test.
- Kada je softver volatilan. Na primjer, ako se korisničko sučelje i funkcionalnost izmjene iz jedne u drugu verziju, napor potreban za automatizaciju testiranja nije isplativ.
- Testiranja koja se lako provjeravaju od strane čovjeka, ali ih je teško, ako ne i nemoguće automatizirati. Na primjer, prikladnost sheme boja, estetska privlačnost izgleda zaslona.
- Testovi koji uključuju fizičku interakciju, kao što je povlačenje kartice kroz čitač kartica, isključivanje ili uključivanje (Guru,n.d.).

7. KAKO ODABRATI ALAT ZA AUTOMATIZIRANO TESTIRANJE

Uspjeh u bilo kojih automatizaciji testiranja ovisi o identificiranju pravog alata za automatizaciju. Uobičajene vrste testiranja, kao što su regresijsko testiranje, funkcionalno testiranje, testiranje jedinica, testiranje integracije zamjenjuju se sustavnim programima testiranja pomoću alata za automatizaciju. Danas se automatizacija testiranja smatra najučinkovitijim načinom za poboljšanje pokretljivosti, učinkovitosti i djelotvornosti bilo koje softverske aplikacije. To je redefiniranje načina na koji inženjeri izvode operacije testiranja. Međutim, najteži zadatak za svaki projekt je odlučiti je li automatizacija potrebna ili nije. Ako su članovi testnog tima odlučili da je automatizacija potrebna, trebaju utvrditi koji se alati i postupci trebaju koristiti. IT tržište je prepuno alata za automatizaciju, ali neće svaki alat odgovarati preduvjetima projekta. Da bi odabrali pravi alat za automatizaciju moraju se pažljivo ispitati specifičnosti projekta (Shaikh, n.d.).

Tijekom rada na određenom projektu tim „Saviant“ skučio se s preprekama na postojećim alatu za automatizaciju testiranja te je morao pronaći novi alat koji bi najbolje odgovarao njihovim potrebama. Kako bi to učinili, formulirali su strateški pristup kako bi odabrali pravi alat za automatizaciju.

4 koraka za odabir odgovarajućeg alata za automatizirano testiranje (Shaikh, n.d.):

1. Temeljito razumijevanje zahtjeva projekta

Održavanje kvalitete aplikacije te isporuka proizvoda bez bugova ključno je za uspjeh bilo kojeg projekta. Automatizirano testiranje može poboljšati kvalitetu projekta i povećati opseg i dubinu testova. Prije nego započnete proces automatizacije testiranja, treba se dobiti uvid u dubinsko razumijevanje zahtjeva projekta kao što je vrsta projekta (desktop, web, mobilna aplikacija), opseg projekta i snaga postojećeg tima na jeziku na kojem se programira.

2. Razmatranje postojećeg alata za automatizaciju kao mjerilo

Tim „Saviant“ je razmotrio „Selenium Test Automation Tool“ kao mjerilo za procjenu i određivanje najboljeg automatizacijskog alata za svoj projekt. Selenium je besplatan, open source alat dostupan za testiranje web aplikacija i web stranica. Pruža jezičnu podršku za C#, Ruby, Java, JavaScript, Pyton i Node.js ali nema odgovarajuće usluge podrške korisnicima. Testni timovi koriste Selenium ako su zadovoljni tehnikama kodiranja i testiraju web-aplikacije preko korisničkog sučelja. Selenium nije rješenje za samostalnog testera koji mora nadzirati ručno i automatizirano testiranje također izazov korištenja ovog alata je kompatibilnost.

3. Određivanje ključnih kriterija prikladnih za projekt

Postoji mnogo ključnih kriterija koje treba razmotriti prije odlučivanja o najprikladnijem alatu za automatizaciju projekta. U nastavku je nabrojano nekoliko ključnih kriterija koji se koriste za procjenu najboljeg alata:

-Jednostavnost izrade i održavanje skripti tj. razvoj i održavanje test skripti treba biti što je moguće jednostavnije kako bi se smanjilo korištenje ljudskih i vremenskih resursa.

-Jednostavnost izvršenja testa tj. izvedba testnog paketa trebala bi biti jednostavna za svakog člana projekta da ih pokrene kada je to potrebno.

-Podrška za web, desktop i mobilnu aplikaciju: korištenje tri različita alata za tri vrste platformi za automatizaciju testiranja je složen zadatak. Najbolje je odabrati alat koji podržava sve tri platforme.

-Intuitivno izvješće o testiranju. Izvješća o testovima stvaraju povjerenje stoga ona moraju biti intuitivna i jednostavna za upravljanje tako su ih članovi tima lako razumiju.

-Cros Browser Testing. Podrška za Cross browser testiranje je potrebna kada postoji više krajnjih korisnika i nema određenog ograničenja preglednika.

-Tehnička podrška i pomoć. Inženjeri automatizacije često trebaju pomoć pri rješavanju kritičkih problema projekta. Alat koji pruža tehničku podršku i pomoć je od velike pomoći.

-Cijena. Ovisno o gore navedenim kvalitetama i procjenama troškova projekta, treba razmotriti cijenu među ostalim dostupnim alatima za automatizaciju.

4. Korištenje Pugh Matrix tehnike za analizu

Uzimajući u obzir gore od navedenih ključnih kriterija kao vitalne parametre tim „Saviant“, iskoristio je Pugh Matrix tehniku za odabir najboljeg alata za web, desktop i mobilna testiranja (slika 9). Ispod Pughove matrice prikazane su prednosti i mane raznih alata, kao što su Ranorex, Test Complite, Ghost Intpector i Test Studio, zadržavajući Selenium Web Driver kao mjerilo.

Slika 9: Pugh Matrix tehnika za analizu alata za automatizirano testiranje softvera

Pugh Matrix

Legenda koncepta selekcije		Riješenje alternativa					
Pozitivni +		Ocjena važnosti	Selenium Web Driver	Ranorex	Test Complete	Ghost Inspector	Test Studio
Isti S							
Negativni -							
Ključni kriteriji							
Jednostavnost izrade i održavanje skripti		5	S	+	+	-	+
Jednostavnost izvršenja testa		5	S	+	+	+	S
Podrška za web, desktop i mobilnu aplikaciju		5	S	+	+	-	S
Intuitivno izvješće o testiranju		4	S	+	+	S	S
Podrška za Cross browser testiranje		4	S	+	+	-	+
Podrška testiranju na temelju ključnih riječi i podataka		4	S	+	+	S	+
Tehnička podrška i pomoć		3	S	+	+	S	S
Podupire C#		3	S	+	+	-	+
Cijena		3	S	-	-	+	-
Integracija TFS DevOps		3	S	S	S	-	S
Zbroj pozitivnih kriterija				8	8	2	4
Zbroj negativnih kriterija				1	1	5	1
Zbroj istih kriterija				1	1	3	5
Ponderirani zbroj pozitivnih kriterija				33	33	8	16
Ponderirani zbroj negativnih kriterija				3	3	20	3
Ukupni zbroj				30	30	-12	13

Izvor: Prilagođeno prema: Shaikh (n.d.)

Za prvi parametar „jednostavnost razvoja i održavanja skripti“ Selenium se smatra dobrim. Postoji mnogo vrsta alata za testiranje koje upravitelj testa može uzeti u obzir prilikom odabira testnih alata.

U matrici, Ranorex pobjeđuje u utrci za Seleniumom, Test Complete, Test Studio i Ghost Inspectorom. Ranorex je intenzivan i učinkovit alat za automatizaciju web, desktop i mobilnih aplikacija. On je iznimno jednostavan za korištenje UI i snimanje i reprodukciju elemenata. Osim tehničkih plus bodova, Ranorex nudi dobru korisničku podršku putem online foruma i putem e-pošte. S iskustvom stečenim kroz manje treninge ili samo-učenjem, čak i ručni tester može izgraditi okvir koristeći ovaj alat. Nekoliko velikih tvrtki koje koriste Ranorex uključuju Siemens, Motorola, Ericson i Vodafone.

Svaki alat je dobar igrač na svom terenu. Ova analiza nije zbog toga što je alat neprikladan, nego u svjetlu činjenice da Selenium Web Driver treba više vještih programera za izvršavanje testova. To premašuje cijenu bilo kojeg drugog alata u smislu određivanja cijena i jednostavnosti složenosti održavanja. Međutim, Selenium Web Driver je najučinkovitiji alat za opsežne web aplikacije. Ghost Inspector je najprikladniji za automatizaciju malog korisničkog sučelja, dok je Test Studio dobar za testiranje desktop i web aplikacija. Test Complete je slično sposoban kao i Ranorex, ali njegovi troškovi i ažuriranja su mnogo skuplji od Ranorexa (Shaikh, n.d.).

8. ALATI ZA AUTOMATIZIRANO TESTIRANJE

Na tržištu ima mnogo komercijalnih i besplatnih alata koji omogućuju automatizaciju testiranja softvera. Uspjeh automatizacije testiranja ovisi u odabiru alata (Brian, n.d.).

Alate za automatizirano testiranje možemo podijeliti na (Brian, n.d.):

- **Open Source alati** su alati koji su objavljeni za korištenje i/ili izmjenu izvornog dizajna besplatno. Open Source alati dostupni su za gotovo svaku fazu procesa testiranja, od upravljanja testnim slučajevima do praćenja pogrešaka. U usporedbi s komercijalnim alatima Open Source alati često imaju manje značajki.
- **Komercijalni testni alati** su proizvodi za prodaju i oni služe za komercijalne svrhe. Komercijalni alati imaju više podrške i više mogućnosti nego alati otvorenog koda.
- **Prilagođeni alati** - u nekim projektima okruženje za testiranje i proces ispitivanja imaju posebne karakteristike. Kada nijedan Open-Source alat ili komercijalni alat ne može ispuniti potrebe testiranja, menadžer testnog tima treba razmotriti razvoj prilagođenog alata za automatizirano testiranje.

U nastavku su nabrojani alati za automatizirano testiranje (tablica 3)

Tablica 2: Usporedba alata za automatizaciju

Alat	**Selenium**	**Katalon Studio**	**UTF**	**TestComplete**	**SoapUi**
Godina nastanka	2004	2015	1998	1999	2005
Tip aplikacije koji se testira	Web aplikacije	Web (UI,API) i mobilne aplikacije	Web (UI,API),mobilne i desktop aplikacije	Web (UI,API),mobilne i desktop aplikacije	API/Web usluge
Podržane platforme	Windows Linux OS X	Windows Linux OS X	Windows	Windows	Windows Linux OS X
Cijena	Besplatno	Besplatno	$$$$	$$$	$$
Podržani skriptni jezici	Java, Groovy, PHP, Ruby, Perl	Java, Groovy	VB Srcipt	Java Script,Pyton,VB Script, Delphi, C++, C#	Ruby
Potrebne vještine programiranja	Potrebne su napredne vještine za integraciju alata	Nisu potrebne napredne vještine programiranja, ali su preporučene za napredne skripte	Nisu potrebne napredne vještine programiranja, ali su preporučene za napredne skripte	Nisu potrebne napredne vještine programiranja, ali su preporučene za napredne skripte	Potrebne su napredne vještine za integraciju alata
Jednostavnost korištenja	Potrebne su napredne	Jednostavan za instalaciju	Kompleksna instalacija,	Jednostavna instalacija,	Jednostavna upotreba

	vještine za instalaciju i korištenje	i korištenje	potrebna obuka za pravilno korištenje alata	potrebna obuka za pravilno korištenje	i instalacija

Izvor: Prilagođeno prema: Brian (n.d.)

8.1. Selenium

Selenium je open-source automatizirani paket za testiranje web-aplikacija na različitim preglednicima i platformama (Guru, n.d.).

Selenium je jedan od popularnijih alata za automatizirano testiranje. Smatra se industrijskim standardom za testiranje automatizacije web-aplikacija na korisničkom sučelju. Gotovo devet od deset testera koristi ili je koristilo Selenium u svojim projektima, prema anketi o izazovima automatizacije testiranja (Guru, n.d.).

Za programere i testere koji imaju iskustva i vještine u programiranju i skriptiranju, Selenium nudi fleksibilnost koja nije moguća u mnogim drugim alatima automatiziranog testiranja. Korisnici mogu pisati testne skripte na različitim jezicima kao što su Java, Groovy, Python, C#, PHP, Ruby i Perl koji rade na više sistemskih okruženja (Windows, Linux, OS X) i preglednicima (Chrome, Firefox, IE). Kako bi se Selenium učinkovito koristio, korisnici moraju posjedovati napredne vještine programirana i moraju potrošiti dosta vremena na izgradnju okvira automatizacije i knjižnica potrebnih za automatizaciju. To je glavni nedostatak Seleniuma, koji se razmatra u integriranim alatima kao što je Katalon Studio.

Selenium je veoma sličan UFT-u, samo što se Selenium bazira na testiranje web aplikacija (Brian, n.d.).

Selenium nije samo alat već i skup softvera, a svaki od njih zadovoljava različite potrebe organizacije za testiranjem.

Ima četiri komponente (slika 10) (Guru, n.d.):

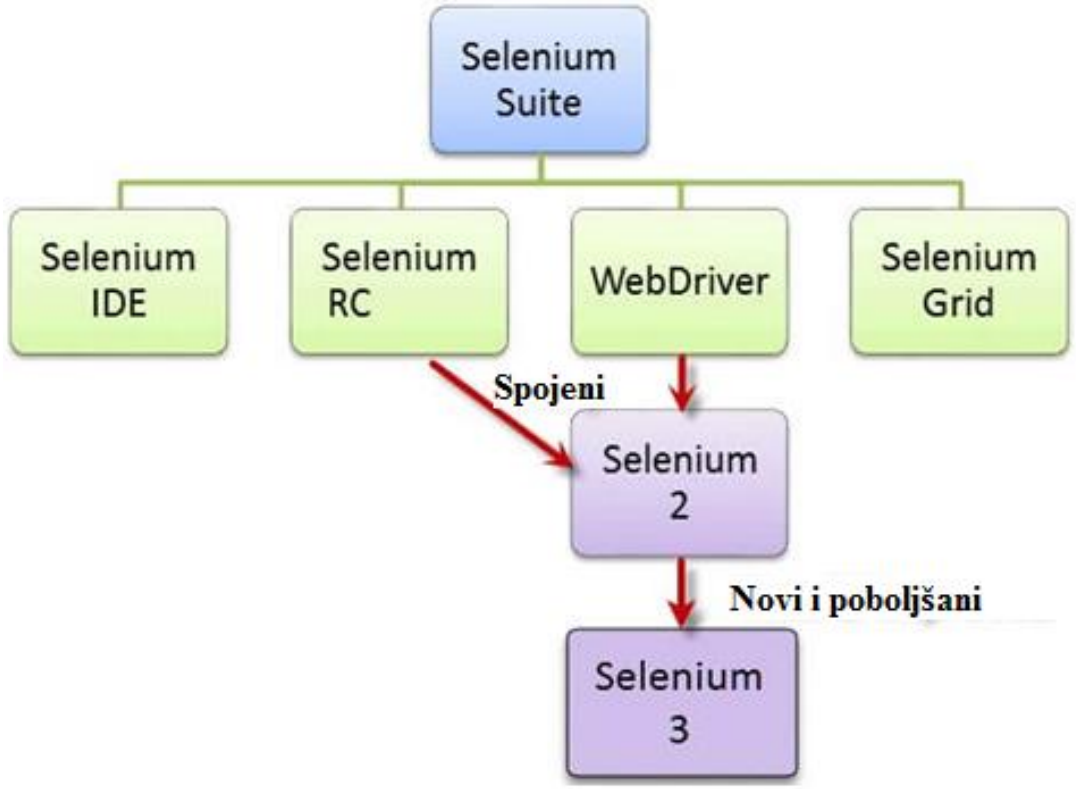
- **Selenium Integrated Development Environment (IDE)**
  - Najjednostavniji paket u Selenium paketu i najlakši za učenje.
  - Zbog svoje jednostavnosti Selenium IDE treba koristiti samo kao prototip. Ako se žele stvoriti napredni slučajevi, preporučuje se koristiti Selenium RC ili WebDriver.
- **Selenium Remote Control (RC)**
  - Dugo vremena je bio vodeći okvir testiranja cijelog projekta Selenium.

- Podržava sljedeće programske jezike: Java, C#, PHP, Pyton, Perl, Rubin.

- **Selenium WebDriver**
- Primjenjuje moderniji i stabilniji pristup u automatiziranju preglednika
- Upravlja izravnom komunikacijom s preglednikom
- Podržava sljedeće programske jezike: Java, C#, PHP, Pyton, Perl, Rubin

- **Selenium Grid**
  - Alat koji se koristi sa Selenium RC za pokretanje paralelnih testova na različitim strojevima u isto vrijeme
  - Iznimno štedi vrijeme

Slika 10: Komponente alata Selenium



Izvor: Prilagođeno prema: Guru (n.d.)

8.2. Katalon Studio

Katalon Studio je moćan i sveobuhvatan alat za automatizaciju testiranja API, Weba i mobilnih aplikacija. Ima velik skup značajki za ove vrste testiranja te podržava više platformi uključujući Windows, Linux, MAC OS (Brian, n.d.).

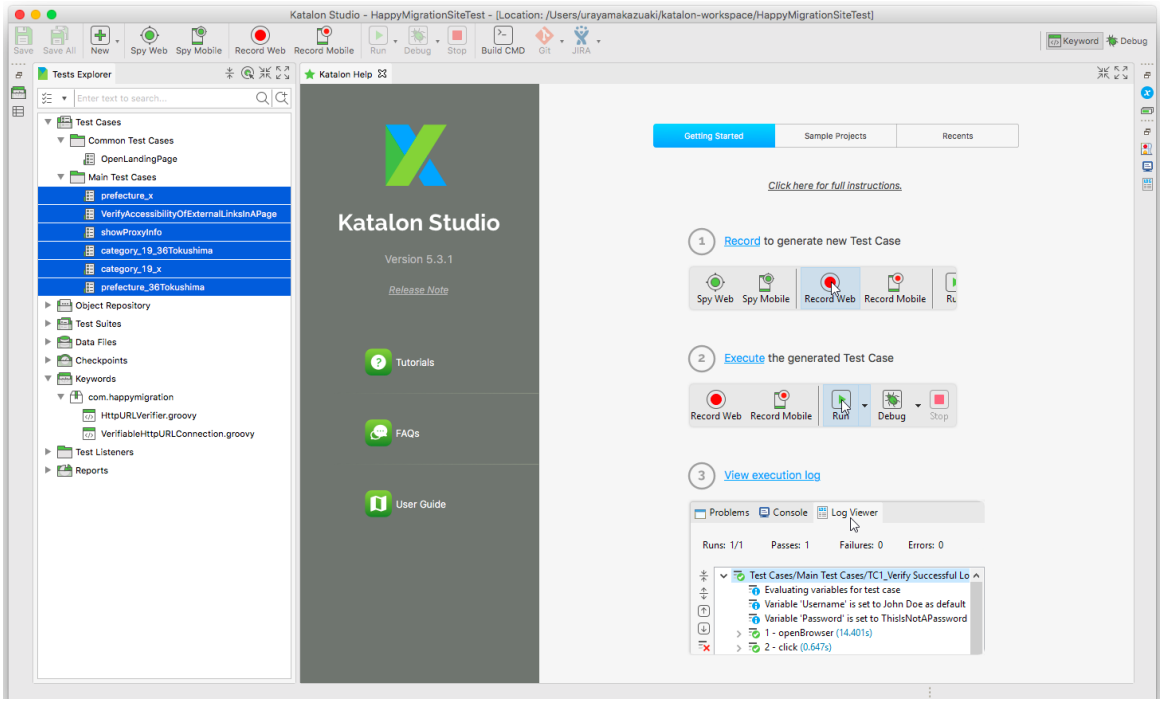
Katalon Studio pruža jednostavno integrirano okruženje za testere koji pronalaze poteškoće u integraciji i implementaciji različitih okvira i knjižnica za korištenje Seleniuma i Appiuma. Katalon Studio je jedan od najznačajnijih alata za automatizirano testiranje (Brian, n.d.).

Katalon Studio je besplatni alat izgrađen 2015 godine. Osmišljen je za stvaranje i ponovnu upotrebu automatiziranih testnih skripti za korisničko sučelje bez kodiranja.

Glavna prednost ovog alata je u tome što ga je lako implementirati i to što ima širi skup integracija u odnosu na Selenium. Katalon Studio ima dvostruke scenarije skripti za korisnike s različitim vještinama programiranja. Testeri s ograničenim tehničkim znanjem mogu upotrebljavati jednostavnije korisničko sučelje koje ne zahtijeva pisanje koda. Iskusniji kosrinici imaju pristup skriptiranju uz isticanje sintakse, prijedlog koda i uklanjanje pogrešaka. Na slici 11 prikazano je sučelje alata Katalon Studio.

Jedan od glavnih nedostataka Katalon Studio alata je nedostatak skriptnih jezika, za razliku od Selenium i TestComplete koji podržavaju mnogo programskih jezika, Katalon Studio podržava samo jedan skriptni jezik: Groovy (AltexSoft, 2019).

Slika 11: Sučelje alata Katalon Studio



Izvor: AltexSoft (2019)

8.3. UFT

UFT (Unified Functional Testing) je popularan komercijalni alat za testiranje desktop,web i mobilnih aplikacija. Alat je proširen tako da uključuje dobar skup za API testiranje. Podržava višestruke platforme za ciljanu aplikaciju koja se testira. UFT pruža nekoliko naprednih mogućnosti za otkrivanje pametnih objekata i detekciju objekata na temelju slike. Na slici 12 prikazano je sučelje alata UTF.

Nekoliko značajki alata UFT alata (Brian, n.d.):

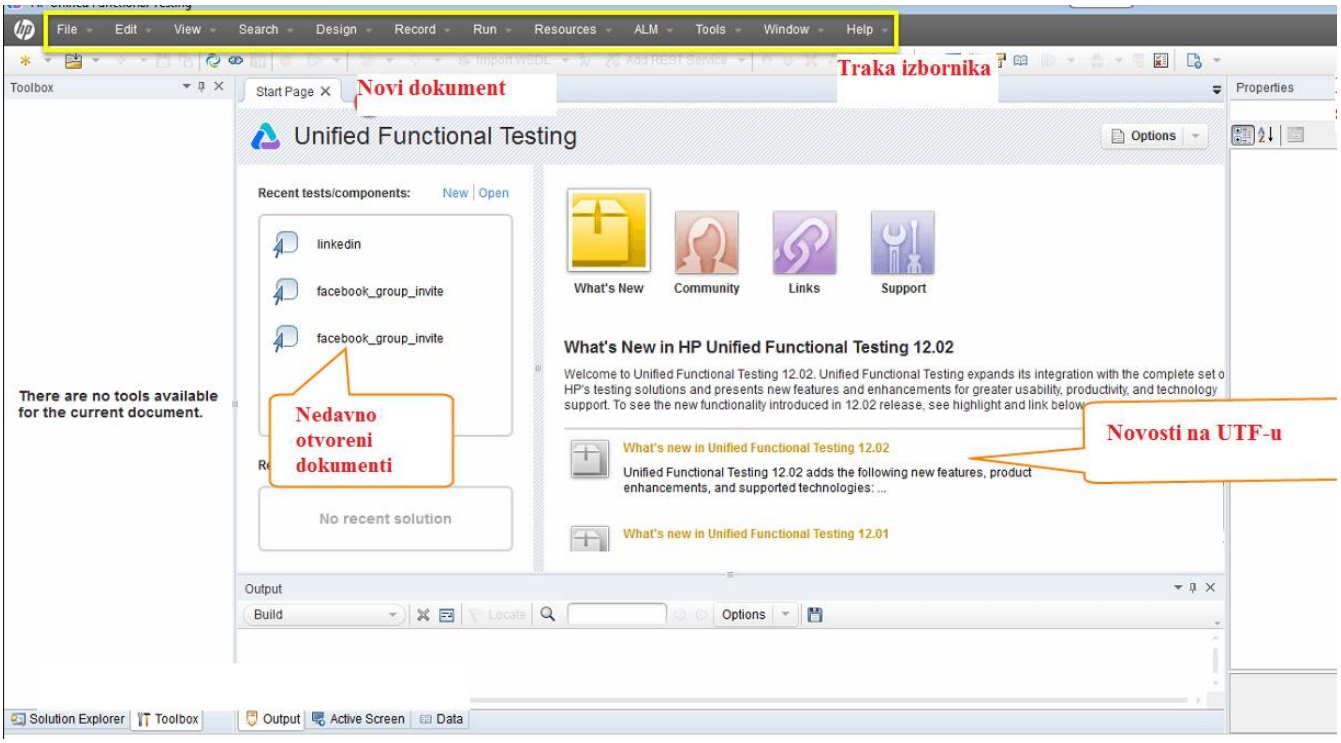
- Intuitivno korisničko sučelje za izradu, izvršavanje API testova
- Podrška generiranju API testova iz WADL dokumenata
- Radnje, aktivnosti i parametri testova mogu se vizualizirati u dijagramima

Prednosti UFT-a (Guru, n.d.)

- Podržava snimanje i reprodukciju
- Koristi aktivni zaslon za snimanje skripti i pomaže testeru u upućivanju na svojstava zaslona
    - Ima veoma dobar mehanizam identifikacije predmeta

- Podržava različite dodatke poput: Oracle, Java, SAP, NET, Web Forms, People Soft itd.
- Podržava popularne okvire automatizacije: pristup testiranju usmjeren na ključne riječi, modularni pristup testiranju, pristup testiranju temeljen na podacima.
- Može se integrirati s alatima za upravljanje testova kao što su: Quality Center, Test Director i Winrunner
- Jednostavan za održavanje

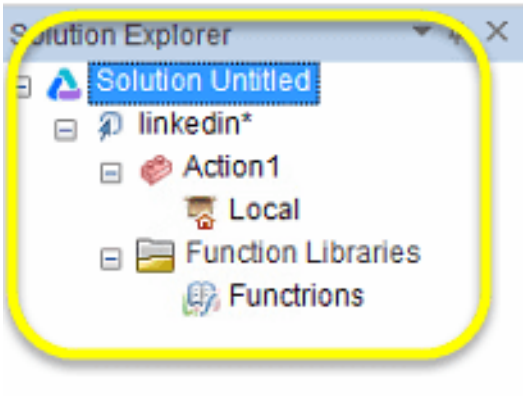
Slika 12: Sučelje alata UTF



Izvor: Prilagođeno prema: Guru (n.d.)

Unutar UTF-a postoji komponenta „Explorer Solution“ (slika 13) u kojoj se nalazi skup svih akcija i testova u trenutnom projektu, s njihovim referencama, događajima i tokovima (Guru,n.d.).

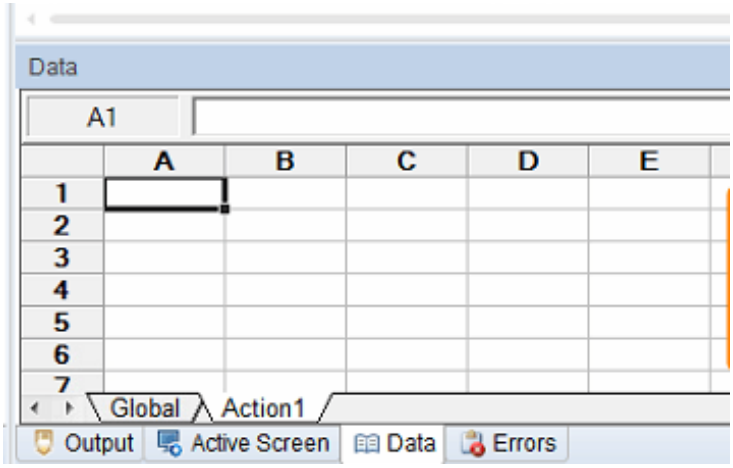
Slika 13: UTF: Explorer Solution



Izvor: Guru (n.d.)

Još jedna važna komponenta UTF alata je tablica podataka (slika 14), svi podaci povezani s testom mogu se unijeti putem te tablice.

Slika 14: UTF: tablica podataka



Izvor: Guru(n.d.)

8.4. TestComplete

TestComplete je na popisu najboljih alata za automatizirano testiranje zbog snažnog i sveobuhvatnog skupa značajki za testiranje weba, mobilnih i desktop aplikacija. Testeri mogu koristiti JavaSkript, VBSkript, Python ili C++ jezike za pisanje test skripti. Testeri mogu jednostavno koristiti TestComplete značajku za snimanje i reprodukciju kao u alatu Katalon Studijo. TestComplete ima mogućnost umetanja kontrolnih točaka u testne korake kako bi

potvrdili rezultate. Kao proizvod tvrtke SmartBear, TestComplete se može lako integrirati s drugim proizvodima koje nudi SmartBear (Brian, n.d.).

Prednosti alata Test Complete su (AltexSoft, 2019):

- Jednostavnost korištenja. Omogućava korisnicima dodavanje i brisanje testova, izmjenu parametara i promjenu redoslijeda ispitivanja.
- Prilagodljivost - omogućuje ručno pisanje i uređivanje skripte
- Pregledna dokumentacija, korisnička podrška te pravovremena ažuriranja

8.5. SoapUI

SoapUI nije alat za testiranje web ili mobilnih aplikacija; ali može biti alat za testiranje API-ja i usluga. To je funkcionalni alat posebno dizajniran za API testiranje.
SoapUI podržava REST i SOAP usluge. Testeri za automatizaciju API-ja mogu koristiti open-source ili pro verziju. Pro izdanje ima prilagođeno sučelje korisniku i nekoliko naprednih značajki kao što je „SQL query builder“ (Brian, n.d.).

SOAP je skraćenica za „Simple Object Access Protocol“. Svojstva SOAP protokola su (Guru, n.d.):

- XML protokol za komunikaciju između dva različita sustava
- Neovisan je o platformi i jeziku (sustav razvijen pomoću jezika Java može komunicirati sa sustavom razvijenim u .NET)
- SOAP zahtjevi prenose se putem HTTP-a

SoapUI nije samo funkcionalan API alat, već nam omogućava i nefunkcionalno testiranje kao što su test performansi i test sigurnosti.

Važne značajke alata SoapUI (Guru, n.d.)**:**

1. Funkcionalno ispitivanje
    - Omogućava testerima da pišu funkcionalne API testove
    - Podržava „drag-drop“ značajku koja ubrzava proces testiranja

- Podržava uklanjanje pogrešaka u testovima i omogućava testerima da razviju testove temeljene na podacima
- Omogućuje napredne scenarije (tester može razviti svoj prilagođeni kod ovisno o scenariju)

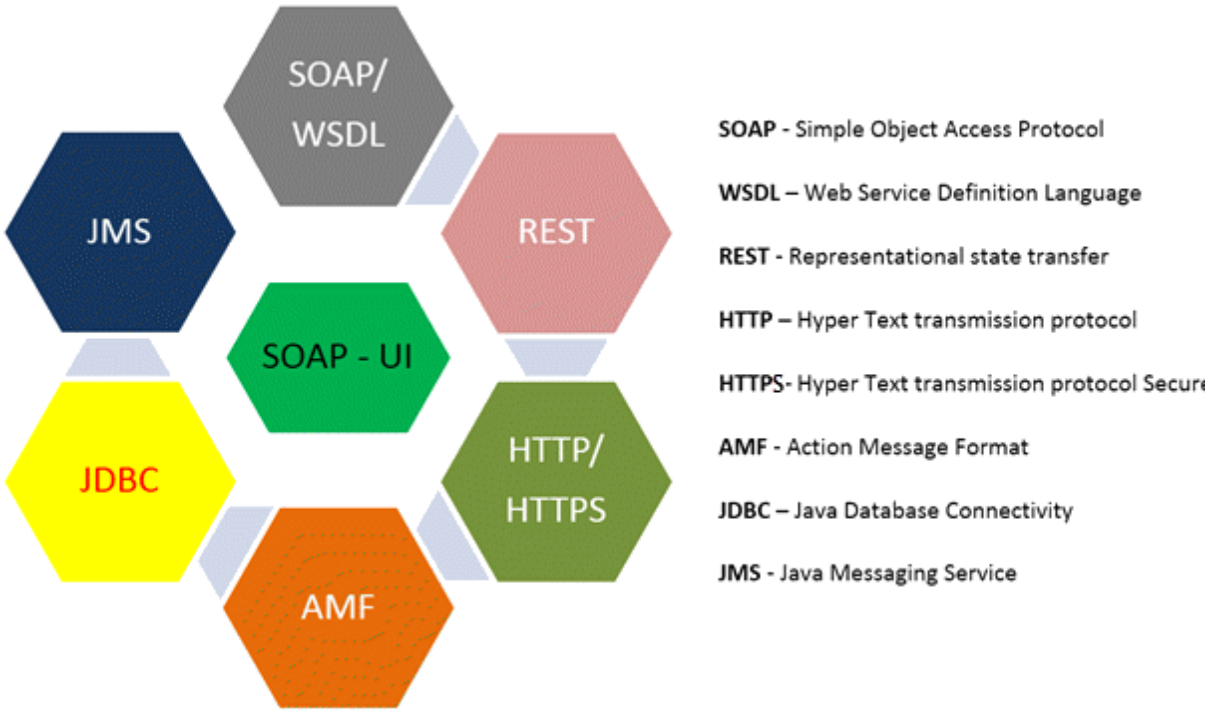
2. Sigurnosno testiranje
    - Ima mogućnost kompletnog skupa skeniranja ranjivosti
    - Sprječava Sql ubrizgavanje radi zaštite baze podataka

3. Učitavanje testiranja
    - Jednostavno simuliranje testiranja velikih količina i stvarnog opterećenja
    - Omogućuje prilagođeno izvješćivanje za snimanje parametara performansi
    - Omogućuje praćenje performansi sustava od početka do kraja

4. Podržani protokoli/ tehnologije

- Od svih nabrojenih alata, SoapUI ima najveći raspon protokola koje podržava, protokoli koje SoapUi možemo vidjeli na slici 15.

Slika 15: Protokoli koje podržava SoapUI



Izvor: Prilagođeno prema: Guru (n.d.)

5. SoapUi se može integrirati s ostalim alatima za automatizaciju (Guru,n.d):

- **Apache Maven** je softverski alat za upravljanje projektima, može upravljati njihovom gradnjom, izvještavanjem i dokumentacijom iz središta spremišta.

- **HUDSON**- kontinuirani integracijski alat temeljen na Javi, integrira se alatima kao što su CVS, Subversion, Git, Perforce, Clearcase i RTC.

- **Junit** je okvir za testiranje jedinica izgrađen na Javi koji može kontrolirati tok testova iz SoapUI-a.

U tablici 4 vidimo usporedbu alata SoapUi s vodećim alatom za automatizirano testiranje Selenium.

Tablica 3: Usporedba alata SoapUI i Selenium

SOAP UI	SELENIUM
Ne koristi se za testiranje korisničkog sučelja. Koristi se samo za WebAPI ili WebService testiranje.	Selenium se koristi za testiranje korisničkog sučelja.
Ima sposobnost testiranja podataka poslanih i primljenih između web preglednika i web poslužitelja. Može testirati protokole poput SOAP i REST.	Ne može testirati protokole, ali može testirati sučelje.
Sposoban za testiranje funkcionalnosti, opterećenja i sigurnosti.	Može obavljati samo funkcionalno testiranje. Moguće je testiranje performansi do određene mjere jer možemo pratiti vrijeme izvršenja s obzirom na izvedbu ali ne možemo testirati višekorisničku i višestranu korist. Ne može se koristiti za testiranje sigurnosti.
Ne ovisi o protokolu i o pregledniku .	Ovisi o mogućnostima preglednika.

Izvor: Prilagođeno prema: Guru(n.d.)

9. SELENIUM WebDriver: Praktični primjer

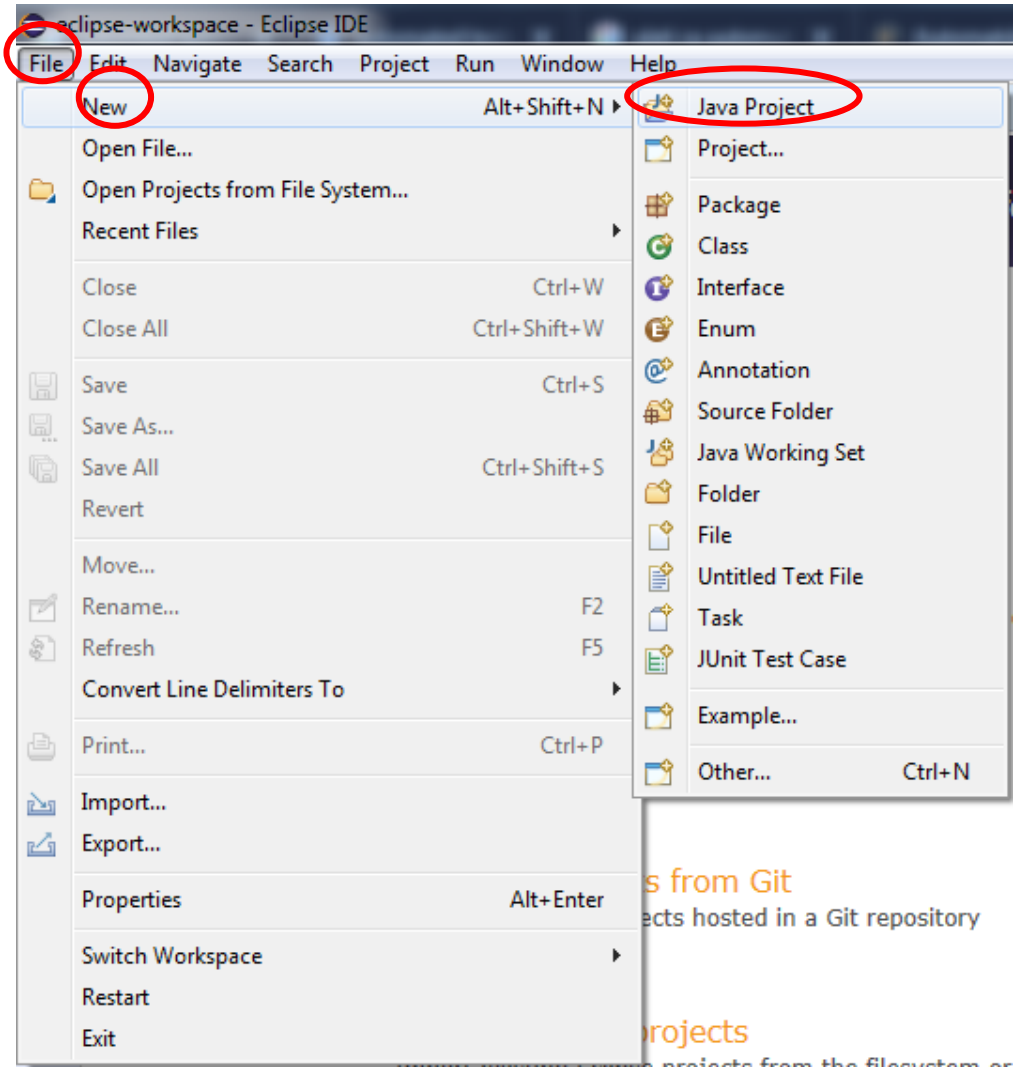
U ovom odlomku objašnjen je alat Selenium WebDriver.

Za instalaciju Selenium WebDrivera potrebno je instalirati: „*Java Software Development Kit*“, „*Eclipse IDE for Java Developers*“ i „*Selenium Java Client Driver*“ (Guru, n.d.).

Nakon što su instalirane ove tri komponente, potrebno je konfigurirati Eclipse IDE s WebDriverom:

Kreiramo novi projekt tako da na traci alata odaberemo *File>New>Java Project* (slika 16).

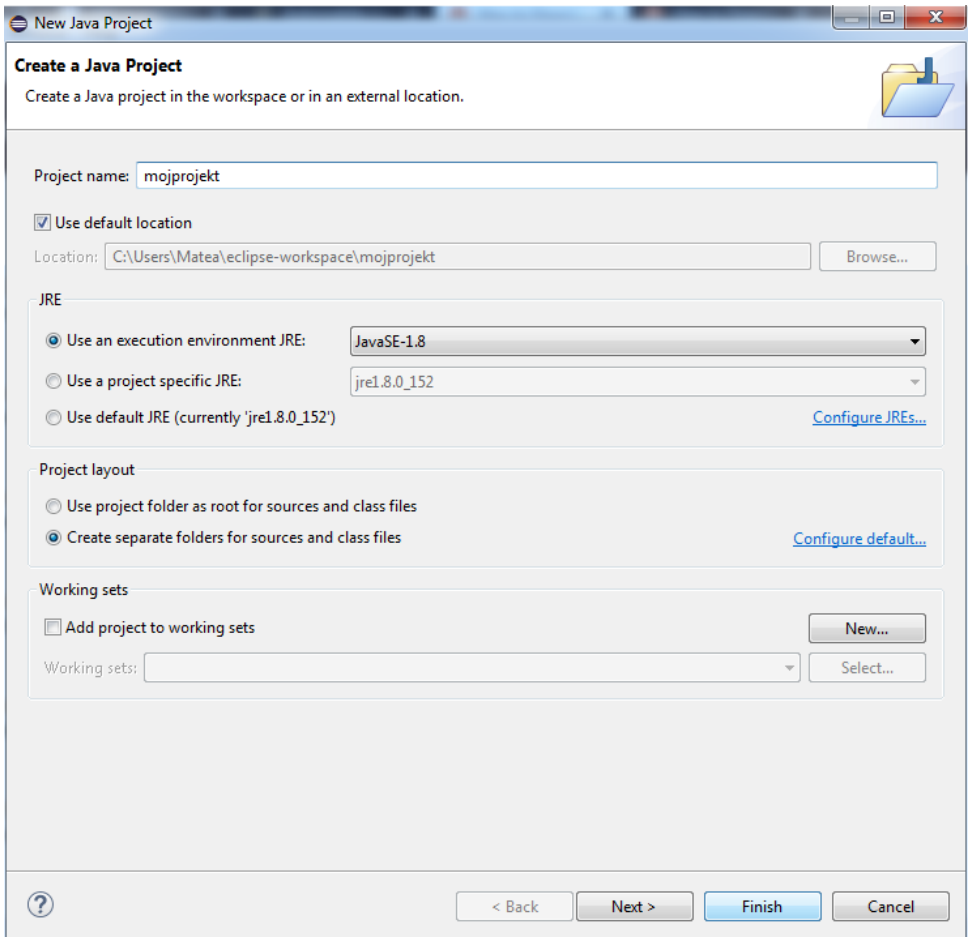
Slika 16: Kreiranje novog projekta



Izvor: Prilagođeno prema: Guru (n.d.)

Zatim odabiremo naziv projekta, mjesto za spremanje projekta, opciju izgleda projekta, klikom na gumb „*Finish*“ stvoren je novi projekt (slika 17).

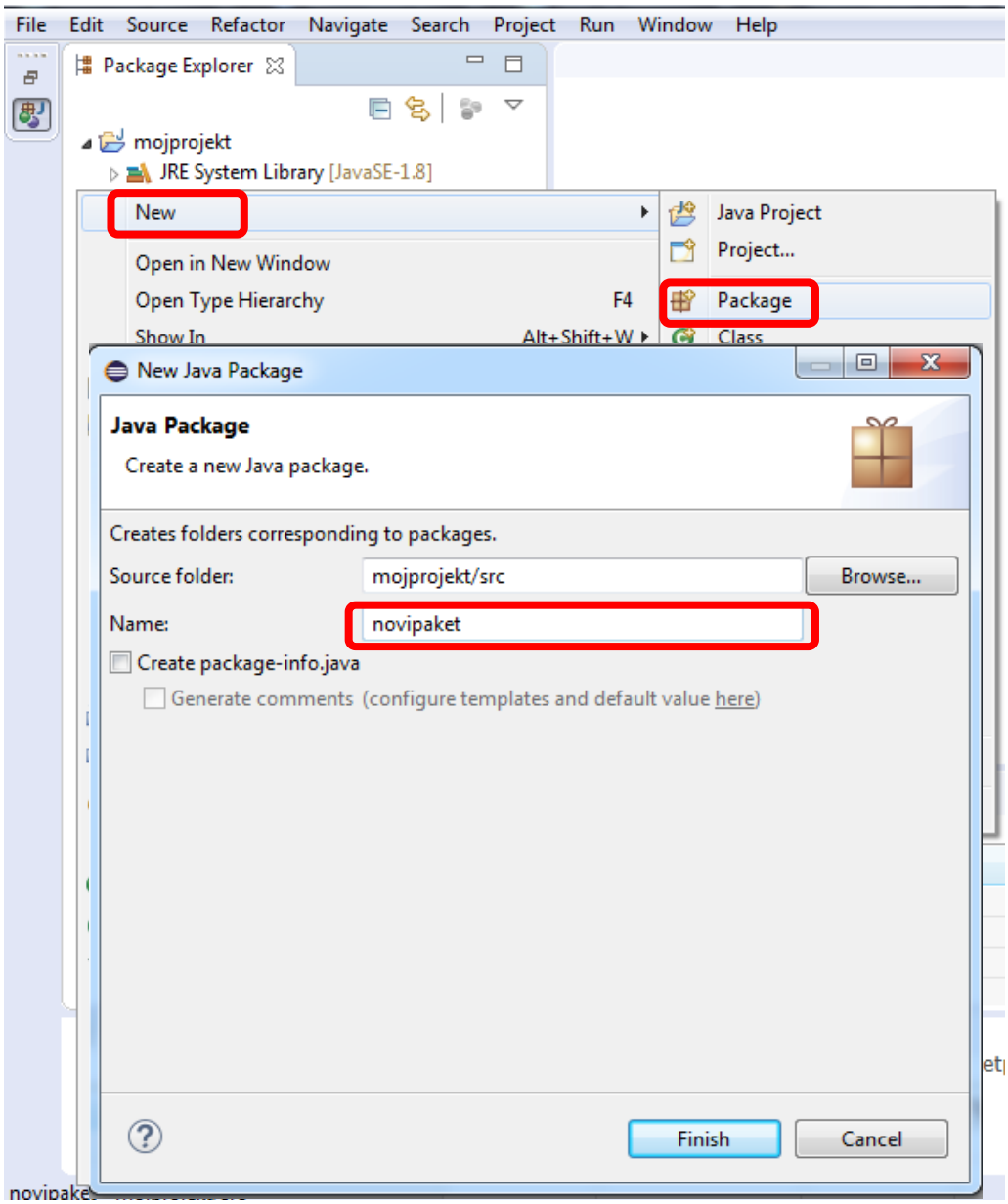
Slika 17: Stvaranje novog projekta



Izvor: Prilagođeno prema: Guru (n.d.)

1. U sljedećem koraku dodat ćemo nove pakete u naš projekt. Desnom tipkom miša kliknemo na novostvoreni projekt odaberemo New>Package te ga imenujemo "novi paket" (slika 18).

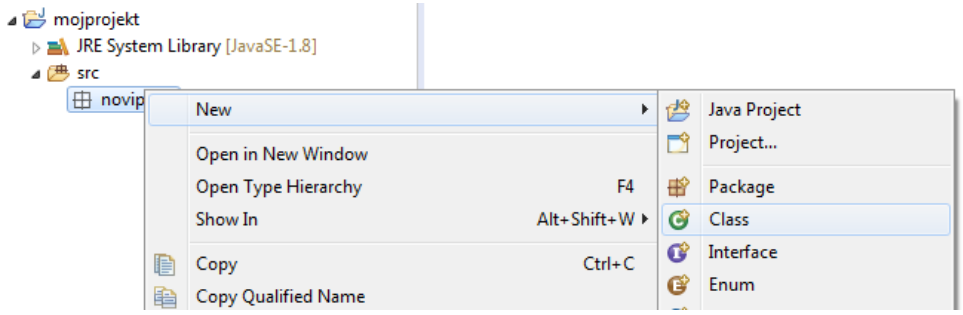
Slika 18: Dodavanje novog Java paketa



Izvor: Prilagođeno prema: Guru (n.d.)

2. Zatim Stvorimo novu Java klasu, desnim klikom na paket koji smo napravili, *New>Class* (slika 19).

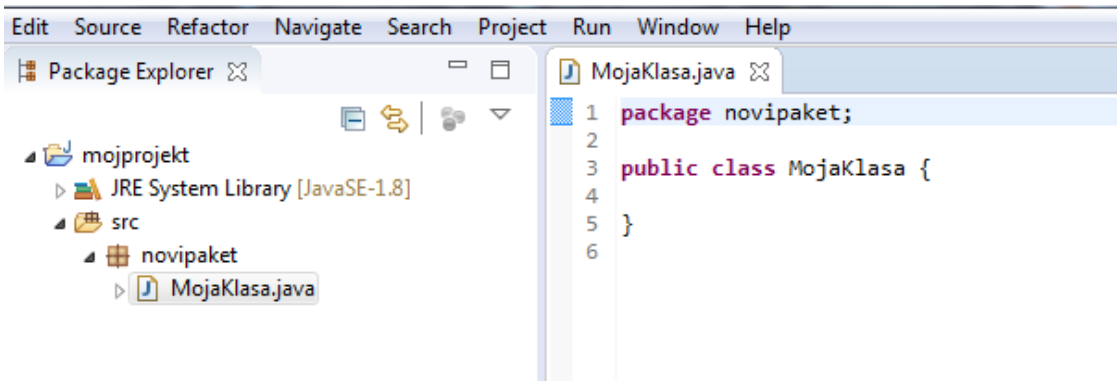
Slika 19: Dodavanje nove klase



Izvor: Prilagođeno prema: Guru (n.d.)

Ovako izgleda projekt nakon dodavanja klase (slika 20).

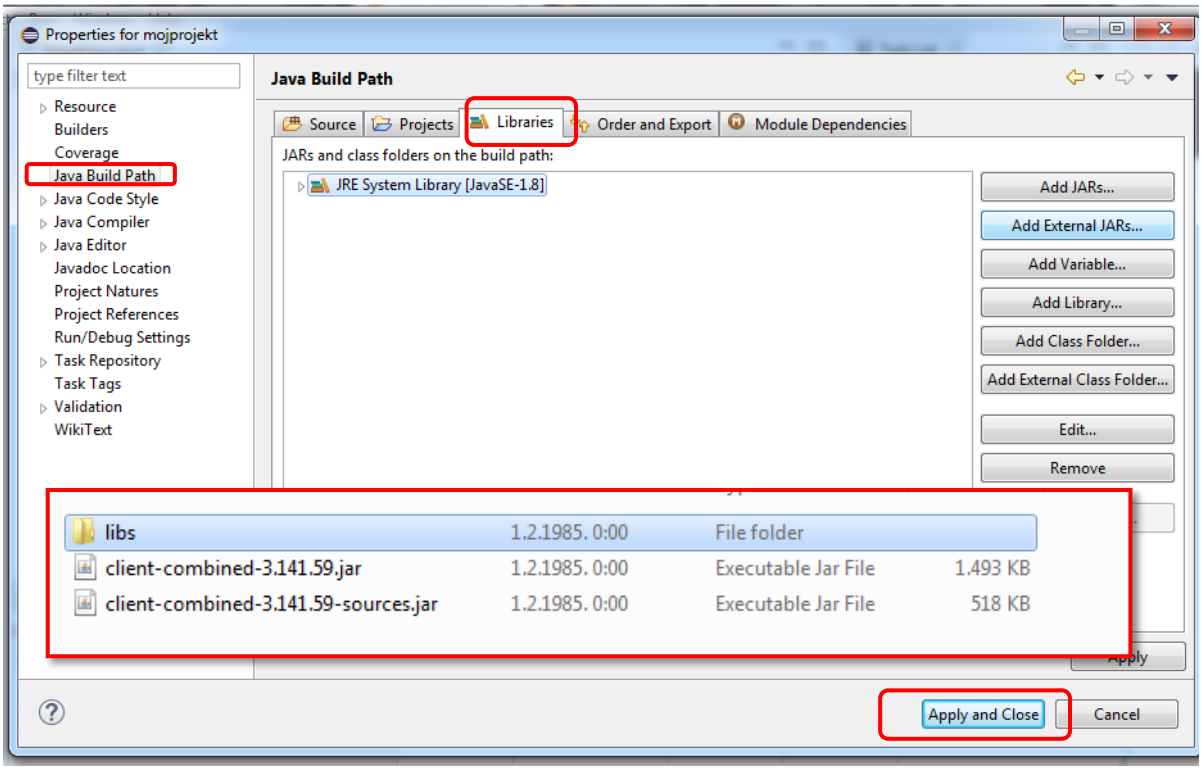
Slika 20: Klasa unutar projekta



Izvor: Prilagođeno prema: Guru (n.d.)

3. Potreban je uvoz Selenium biblioteka. Desnim klikom miša na "noviprojekt" i odaberemo „Properties“. Zatim odaberemo *Java Build Path> Libraries>Add External Jars.* Zatim označino sve Jar datoteke iz „Selenium Java Client Driver“. Nakon završetka kliknemo na gumb „Apply and Close“ (slika 21).

Slika 21: Uvoz Selenium biblioteka



Izvor: Prilagođeno prema: Guru (n.d.)

Unutar klase „MojaKlasa“ koju smo kreirali. Dodajemo kod (implementacija 1) koji će:

- Dohvatiti stranicu „Mercury Tours'“
- Potvrditi naslov
- Ispisati rezultat
- Zatvoriti ga prije završetka programa

Implementacija 1: Provjera naslova stranice

```
package novipaket;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
public class MojaKlasa {
public static void main (String[] args) {
// deklaracija varijabli
System.setProperty ("webdriver.firefox.marionette","C:\\geckodriver.exe");
WebDriver driver = new FirefoxDriver();
String baseUrl = "http://demo.guru99.com/test/newtours/";
```

```
String expectedTitle = "Welcome: Mercury Tours";
String actualTitle = "";
driver.get(baseUrl);
 // uzima stvarnu vrijednost naslova
actualTitle = driver.getTitle();
* usporedite stvarni naslov stranice s očekivanim i ispisujte
* rezultat kao "Uspješan test" ili "Neuspješan test" *
if (actualTitle.contentEquals(expectedTitle)){
System.out.println("Uspješan test");
} else {
System.out.println("Neuspješan test");
}
driver.close();
}
```

Izvor: Prilagođeno prema: Guru (n.d.)

Pojašnjenje implementacije (Guru, n.d.):

- Za početak uveli smo dva paketa:
  - **org.openqa.selenium** – sadrži klasu WebDriver potrebnu za pokretanje novog preglednika s određenim upravljačkim programom.
  - **org.openqa.selenium.firefox.FirefoxDriver** - sadrži klasu FireFoxDriver potrebnu za instanciranje Firefox-ovog upravljačkog programa za preglednik koji je indicirao klasu WebDriver.
- Obično se na ovaj način instanciraju objekti: *WebDriver driver= new FireFoxDriver();*

- Metoda get() WebDrivera koristi se za pokretanje nove sesije preglednika i usmjerava ga prema URL-u koji navedete kao njegov parametar.
  driver.get(baseUrl);

- Klasa WebDriver ima metodu getTitle() koja se uvijek koristi za dobivanje naslova stranice.

actualTitle=driver.getTitle();

- Ovaj dio koda koristi Java if-else strukturu za usporedbu stvarnog koda s očekivanim:

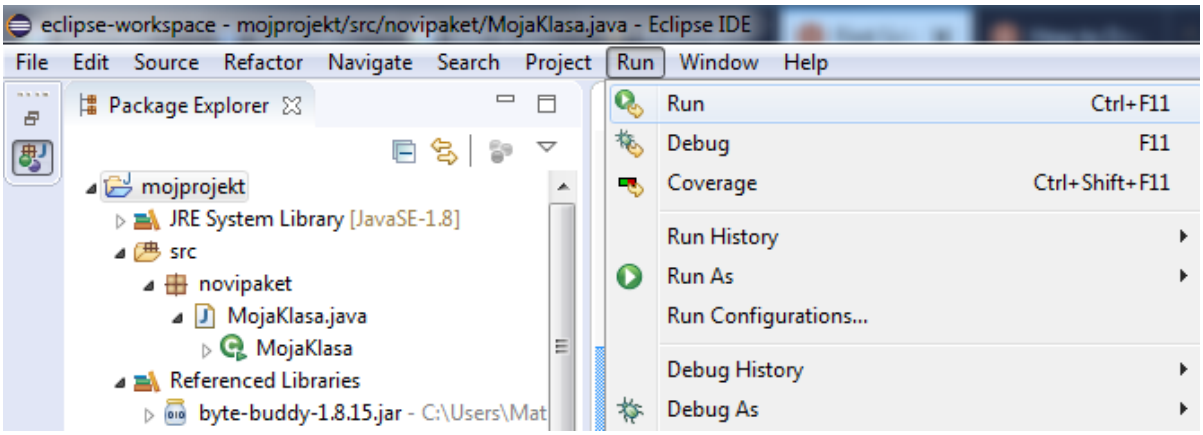
```
if (actualTitle.contentEquals(expectedTitle)){
System.out.println("Uspješan test!");
} else {
System.out.println("Neuspješan test");
}
```

- Metoda „close()“ koristi se za zatvaranje prozora preglednika:

driver.close();

Na kraju pokrećemo test. Na traci alata odaberemo „*Run*“ (Pokreni) ili pritisnemo *Ctrl + F11* (slika 22). Ako je sve ispravno napravljeno, u konzoli Eclipse pisat će „*Uspješan test*“.

Slika 22: Pokretanje testa



Izvor: Prilagođeno prema: Guru (n.d.)

10. ZAKLJUČAK

Na kraju ovog rada možemo zaključiti da automatizacija testiranja može uvelike olakšati postupak testiranja. Ručno testiranje zahtjeva više vremena i napora kako bi se osiguralo da softverski proizvod zadovoljava uvjete kvalitete. Osim toga, testeri moraju zabilježiti svoje rezultate. Za razliku od ručnog testiranja gdje je ljudsko biće odgovorno za samostalno testiranje funkcionalnosti softvera tako da odgovara korisničinim zahtjevima, automatizirano testiranje provodi se putem alata za automatizaciju. Što zahtjeva manje vremena u istraživačkim testovima ali je potrebno više vremena za održavanje test skripti.

Automatizirano testiranje nije pogodno za sve vrste projekata kao na primjer za testove koji se provede rijetko, testovi koji se lako provjeravaju od strane čovjeka, testove koji zahtijevaju fizičku interakciju te testove za projekte čije se korisničko sučelje i funkcionalnosti često mijenjaju iz jedne verzije u drugu. Automatizirano testiranje je pogodno za velike projekte; projekti koji se ponavljaju iznova i iznova i zahtijevaju veliki utrošak vremena, te projekte koji su već prošli kroz ručno testiranje.

Na tržištu postoje mnogi komercijalni i „open source“ alati za automatizaciju testiranja, no neće svaki alat odgovarati svakom projektu. Stoga tim koji provodi testiranje mora biti oprezan prije kupnje jednog od takvih alata. Da bi odabrali pravi alat testni tim treba pažljivo ispitati specifičnosti i zahtjeve projekta. Ako niti jedan od komercijalnih alata ne odgovara potrebama njihovog projekta, često se odlučuju na prilagođenje alate za automatizirano testiranje. No takvi alati mogu biti kompliciran za implementaciju te veoma skupi. Jedni od boljih alata za automatizaciju testiranja dostupnih na tržištu su Selenium, Katalon Studio, UTF, Test Complete, SoapUi. No i oni imaju svojih mana i prednosti stoga se ne bi trebao u potpunosti osloniti na takve alate i zanemariti ručno testiranje. Ljudska stručnost je još uvijek potrebna za određivanje prioriteta testova, procjenu korisnosti testova te u donošenju odluka o testovima koji se nikada ne bi mogli generirati alatom. Alati za automatizirano testiranje nisu fleksibilni i nemaju mašte. Međutim, automatizacija testiranja može značajno povećati kvalitetu i produktivnost testiranja softvera. Automatizirano testiranje ima mnoge prednosti u odnosu na manualno testiranje, kao što je dosljednost ispitivanja, testovi koji se često ponavljaju svaki puta će se točno ponoviti, što daje razinu dosljednosti koju je teško postići manualnim testiranjem. Automatizacija također povećava povjerenje, znajući da je automatiziran test uspješno pokrenut veća je vjerojatnost da neće biti neugodnih iznenađenja kada se softver isporuči kupcu. Iako je automatizacija testiranja u većini slučajeva veoma korisna, postoje brojni problemi koji se mogu pojaviti u pokušaju automatizacije. To se

najčešće događa ako je odabran pogrešan alat ili su očekivanja nerealna. Nerijetko se događa da alati ne uspijevaju raditi u nekim okruženjima. Alati za automatizaciju testiranja su veliki i složeni proizvodi koji zahtijevaju detaljno tehnološko znanje. Kako bi dobili najviše od automatizacije testiranja potrebno je izdvojiti vrijeme za odabir alata, za obuku testnog tima o odabranom alatu te učenje što najbolje funkcionira za promicanje korištenja alata unutar organizacije.

11. POPIS LITERATURE

Knjige:

1. Beizer, B. (1990). *Software Testing Techniques,* 2nd ed., Van Nostrand-Reinhold, 1990, ISBN:0-442-20672-0, New York
2. Fewster,M.,Graham,D. (1994).Softwere Test Automation: Techniques for Automating Test Execution.London: ACM Press Books,.
3. Manger,R.(2005). Softversko Inženjerstvo-skripta.Prvo izdanje.Zagreb
4. Myers,G. (2004)., The Art of Software Testing, Second Edition. New Jersey.: Word Association, Inc.,

Članci:

5. Bevanda,V.,Sinković,G.,(2016). Sustavi Znanja u Potpori Upravljanju Kvalitetom Softverskog Proizvoda. Informatologia.Vol. 42. Str. 284–292
6. Oreški,P.,(1990). Metrika Softvera. Zbornik radova Vol.14. Str.75-87

Kvalifikacijski radovi:

7. Pranic,M.(2010). Alati za Testiranje Softvera (završni rad) Fakultet Elektrotehnike, Strojarstva i Brodogradnje,, Split, Sveučilište U Splitu
8. Rampure,V.(n.d.) Automated software testing, Advanced software engineeringcsc532(Term paper)

Internet izvori:

9. AltexSoft (2019) Comparing Automated Testing Tools: Selenium, TestComplete, Ranorex, and more Dostupno na: https://www.altexsoft.com/blog/engineering/comparing-automated-testing-tools-selenium-testcomplete-ranorex-and-more/ (21.7.2019)
10. AltexSoft (2019) The Good and the Bad of Katalon Studio Automation Testing Tool, Dostupno na: https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-katalon-studio-automation-testing-tool/ (21.7.2019)
11. Brian (2019) Best Automation Testing Tools for 2019 (Top 10 reviews). Dostupno na: https://medium.com/@briananderson2209/best-automation-testing-tools-for-2018-top-10-reviews-8a4a19f664d2 (17.7.2019)
12. Guru (n.d.) Alpha Testing Vs Beta Testing: What's the Difference? Dostupno na:https://www.guru99.com/alpha-beta-testing-demystified.html(14.06.2019)

13. Guru (n.d.) 7 Software Testing Principles: Learn with Examples. Dostupno na: https://www.guru99.com/software-testing-seven-principles.html (13.7.2019)
14. Guru (n.d.) AUTOMATION TESTING Tutorial: What is, Process, Benefits & Tools. Dostupno na: https://www.guru99.com/automation-testing.html (10.07.2019)
15. Guru (n.d.) First Selenium Webdriver Script: JAVA Code Example Dostupno na: https://www.guru99.com/first-webdriver-script.html (22.8.2019)
16. Guru (n.d.) How to Download & Install Selenium WebDriver Dostupno na: https://www.guru99.com/installing-selenium-webdriver.html (23.8.2019)
17. Guru (n.d.) How to use QTP/UFT IDE, Dostupno na: https://www.guru99.com/uft-qtp-ide.html (21.7.2019)
18. Guru (n.d.) Keyword Driven Testing Framework with Example. Dostupno na: https://www.guru99.com/keyword-driven-testing.html (14.7.2019)
19. Guru (n.d.) SoapUI Tutorial: Create a Project, Test Suite, TestCase. Dostupno na: https://www.guru99.com/soapui-tutorial-project-testsuite-testcase.html (21.8.2019)
20. Guru (n.d.) What is Data Driven Testing? Learn to create Framework. Dostupno na: https://www.guru99.com/data-driven-testing.html (10.7.2019)
21. Guru (n.d.) What is Selenium? Introduction to Selenium Automation Testing. Dostupno na: https://www.guru99.com/introduction-to-selenium.html (10.7.2019)
22. Guru (n.d.) What is SoapUI? Introduction to SoapUI Testing Dostupno na: https://www.guru99.com/introduction-to-soapui.html (22.8.2019)
23. ISO 25000(n.d.) ISO/IEC 25010 Dostupno na: https://iso25000.com/index.php/en/iso-25000-standards/iso-25010?limit=3&limitstart=0 (28.8.2019)
24. Shaikh,F (n.d.) Simple Steps to Select the Right Test Automation tool for your Project. Dostupno na: https://www.saviantconsulting.com/blog/4-steps-select-test-automation-tool.aspx (10.07.2019)
25. Smartsheet (n.d.) A Guide to Automation Frameworks. Dostupno na:https://www.smartsheet.com/test-automation-frameworks-software (14.7.2019)
26. Softwaretestingclass (n.d.) How to Write Good Test Cases? Dostupno na: http://www.softwaretestingclass.com/how-to-write-good-test-cases/ (20.8.2019)

Prezentacije:

27. Dustin,E. (n.d.) Automated Software Testing. Dostupno na:http://www.idtus.com

12. POPIS SLIKA, TABLICA

Popis slika:

Popis tablica:

Popis Implementacija:

SAŽETAK

Cilj ovog rada je obrazložiti važne komponente kvalitete softvera, testiranja softvera i vrste testiranja kako bi se detaljno pojasnio svaki od navedenih procesa kao i njihovu važnost u donošenju odluke o automatiziranju testiranja softvera. Svako ponašanje softvera koje nije u skladu sa zahtjevima korisnika predstavlja grešku koju je potrebno prepoznati i ukloniti. U širem smislu testiranje predstavlja sistem kontrole kvalitete kojim se provjeravaju njegove karakteristike i komponente.

Testiranje se može provoditi manualno ili automatizirano. Automatizirano testiranje podrazumijeva korištenje alata za automatizaciju koje izvršava testiranje softvera. Cilj automatizacije je smanjiti broj testnih slučajeva koji se ručno pokreću, a ne eliminirati ručno testiranje. Prilikom odluke o automatiziranju nužno je u obzir uzeti da automatizirano testiranje zahtjeva znatna ulaganja novaca i resursa. Najveće prednosti automatizacije testiranja su: učinkovitost, pokretanje postojećih (regresijskih) testova na novoj verziji programa, mogućnost da se pokrene više testova u manje vremena što omogućuje njihovo češće pokretanje, mogućnost obavljanja testova koje bi bilo teško ili nemoguće napraviti ručno, bolje korištenje ljudskih resursa te dosljednost i ponovljivost ispitivanja. Jedni od najčešće upotrebljavanih automatiziranih alata u testiranju softvera su: Selenium, Katalon Studio, UFT, Test Complete i Soup UI. Ljudska stručnost je još uvijek potrebna za određivanje prioriteta testova, procjenu korisnosti testova te u donošenju odluka o testovima koji se nikada ne bi mogli generirati alatom.

Ključne riječi: Kvaliteta softvera, Testiranje, Automatizirano testiranje, Alati za automatizirano testiranje

Mentor: doc. dr. sc. Tihomir Orehovački

ABSTRACT

The aim of this paper is to explain important software quality components, software testing, and testing types to clarify each of these processes in detail, as well as their importance in deciding on automating software testing.

Any software behavior that does not comply with user requirements is the error that needs to be identified and removed. In a wider sense, testing is a quality control system that checks its characteristics and components.

Testing can be done manually or automatically. Automatical testing implies the use of automatic tools executed by software testing. The goal of automatization is to reduce the number of test cases manually triggered, but not to eliminate manual testing. When deciding on automatization, it is necessary to consider that automatic testing requires substantial investments in money and resources. The biggest advantages of testing automation are: efficiency, launch of existing (regression) tests on a newer version of the program, the ability to run multiple tests in less time enabling them to run more often, to run tests that would be difficult or impossible to do manually, better use of human resources and consistency and repeatability of the test. One of the most commonly used tool for automatic software testing is: Selenium, Katalon Studio, UFT, Test Complete and Soup UI. Although there are great advantages of automatical testing tools, there is still need for expertise to determine the priority of the tests, to evaluate the usefulness of the tests, and to make the. Automated testing tools sometime lack imagination and are not flexible and for that reason at least in some cases manual testing will be preferable.

Key words: Software quality, Testing, Automated Testing, Automated Testing Tools

Supervisor: doc. dr. sc. Tihomir Orehovački