EKONOMIKAS UN KULTŪRAS AUGSTSKOLA

Studiju programma "Informācijas tehnoloģijas"

Ieva Ozola

AUTOMATIZĒTU TESTĒŠANAS SKRIPTU IZSTRĀDE TĪMEKĻA LIETOTNEI

Bakalaura darbs

Darba zinātniskais vadītājs

Mg.sc.comp. Andrejs Liepiņš

Rīga 2019

Noslēguma darba novērtējuma lapa

Bakalaura darbs "Automatizētu testēšanas skriptu izstrāde tīmekļa lietotnei" (turpmāk teksta – Darbs) ir izstrādāts Ekonomikas un kultūras augstskolas studiju programmā "Informācijas tehnoloģijas".

Es, Ieva Ozola, kā Darba vienīgais autors, atļauju / neatļauju (*vajadzīgo pasvītrot*) Ekonomikas un kultūras augstskolai publiskot savu Darbu pilnā apjomā (ieskaitot arī pilna Darba komplektācijā ietilpstošus Darba failus).

.....
(paraksts,datums)

Darba zinātniskais vadītājs: Andrejs Liepiņš .....
(paraksts,datums)

Darba normkontrolieris: …… .....
(vārds, uzvārds) (paraksts,datums)

Darba recenzents: …… .....
(vārds, uzvārds) (paraksts,datums)

Darbs ir ieteikts aizstāvēšanai Valsts pārbaudījuma komisijā.

Studiju programmas ……. direktors (-e) …… ……

…… darbs aizstāvēts Valsts pārbaudījuma komisijas 20… gada …… sēdē un novērtēts ar atzīmi ( ) ……..

Valsts noslēguma komisijas priekšsēdētājs

…… ……. ……
(vārds, uzvārds) (paraksts,datums)

ANOTĀCIJA

Ieva Ozola. Bakalaura darbs. Automatizētu testēšanas skriptu izstrāde tīmekļa lietotnei. – Rīga: Ekonomikas un kultūras augstskola, 2019.

Bakalaura darba apjoms ir 61 lappuses. Bakalaura darbs sastāv no Ievada, Analītiskā apskata daļas, Situācijas izpētes daļas, Pētījuma rezultātu daļas, Secinājumu un priekšlikumu daļas un Izmantotās literatūras un informācijas avotu saraksta. Darbs satur 2 tabulas, 39 attēlus un 37 izmantotās literatūras un informācijas avotus.

Pētījuma aktualitāte: Uzņēmumam attīstot/uzlabojot pakalpojumu klāstu, arī uzņēmuma atbilstošās tīmekļa vietnes tiek regulāri uzlabotas. Tīmekļa vietnes ir uzņēmuma „seja", to nekorekta darbība var būtiski kaitēt uzņēmuma reputācijai un tēlam. Tāpēc pirms uzlabojumi ir pieejami lietotājiem, tīmekļa darbību nepieciešams pārbaudīt - testēt. Manuālā testēšana aizņem daudz laika un cilvēku resursu, tāpēc, daudz optimālāk ir veikt automatizētus testus.

Pētījuma mērķis: Izstrādāt automatizētus skriptus new.manspasts.lv testiem.

Pētījuma metodes: Datu ieguves metodes - izmantojot aprakstošo pētīšanas metodi, aprakstīta automatizētās testēšanas teorētiskie aspekti un artefaktu analīze – testēšanas rīku salīdzināšanai. Datu apstrādes metodes - testēšanas rīku, testu izpildes rezultātu izpildes laiku salīdzināšanai, izmantota salīdzināšanas metode. Lai grafiski attēlotu izstrādes procesu, testēšanas rezultātus, u.c. procesus izmantota grafiskā pētniecības metode. Secinājumu un priekšlikumu sagatavošana, izmantota loģiski konstruktīvā pētījuma metode.

Sasniegtie rezultāti: Izstrādāti automatizētās testēšanas skripti tīmekļa vietnei.

Atslēgas vārdi: tīmekļa vietnes testēšana; automatizētā testēšana; testēšana; testēšanas rīki.

ANOTATION

Ieva Ozola. Bachelor Thesis. Developing automated testing scripts for a web application. – Riga: University of Economics and Culture, bachelor program „ Information Technologies (Programming)", 2019.

The Bachelor Thesis is written in Latvian. The volume of the Thesis is 61 pages. The Thesis consists of Introduction, Analytical literature review, Results, Conclusions and recommendations, and Bibliography. It comprises 2 tables, and 39 figures. Bibliography consists of 37 information resources.

Research relevance: As the company develops / enhances the range of services, the company's relevant websites are also regularly updated. Websites are the "face" of a business, and their improper performance can seriously harm the company's reputation. Therefore, before improvements are available to users, the web activity needs to be verified, that means testing. Manual testing takes a lot of time and human resources, so it is more optimal to perform automated tests.

Research goal: Develop automated scripts for new.manspasts.lv tests.

Research methods: Methods of data mining are the theoretical aspects of automated testing are described, using the descriptive research method and artefact analysis for comparison of testing tools. Methods of data processing - a comparison method was used to compare the testing tools and the execution time of the test execution results. To graphically illustrate the development process, test results, etc. processes are based on a graphical research method. Preparation of conclusions and suggestions, using method of logical constructive research method.

Results achieved: Developed automated testing scripts for the website

Keywords: website testing; automated testing; testing; testing tools.

АННОТАЦИЯ

Иева Озола. Бакалаврская работа. разработка автоматических скриптов дла веб-приложения**. –** Рига: Высшая школа экономики и культуры, 2019.

Объем работы составляет 61 страниц. Бакалаврская работа состоит из Введения, раздела Аналитического обзора, раздела Результатов исследования, раздела Выводов и предложений и Списка использованной литературы. Работа содержит 2 таблицы, 39 рисунков и 37 источников использованной литературы.

Актуальность исследования: Поскольку компании развивают и расширяют спектр услуг, веб-сайты компании также регулярно обновляются, веб-сайты являются «лицом» бизнеса, и их некорректная работа может нанести серьезный ущерб репутации и имиджу компании. Поэтому, прежде чем улучшения станут доступны пользователям, веб-решения должны быть протестированы. Мануальное тестирование сайта занимает много времени и человеческих ресурсов, поэтому более оптимально проводить автоматизированные тесты.

Цель исследования: Разработка автоматических скриптов для тестов new.manspasts.lv.

Методы исследования: Метод получения данных - используя метод описательного исследования, описаны теоретические аспекты автоматизированного тестирования и анализ артефактов для сравнения инструментов тестирования. Метод обработки данных - метод сравнения был использован для сравнения инструментов тестирования, время выполнения результатов для выполнения теста. Чтобы графически проиллюстрировать процесс разработки, результаты тестирования и др., использован графический метод исследования. Для подготовки выводов и предложений, использован метод логического конструктивного исследования.

Результаты: Разработаны скрипты для автоматического тестирования сайта.

Ключевые слова: тестирование сайта; автоматизированное тестирование; тестирование; инструменты тестирования.

SATURS

IEVADS

Mūsdienu dinamiskajā dzīves ritmā, kad tehnoloģijas ienāk ar vien vairāk mūsu ikdienas dzīvē un neviens vairs sevi nevar iedomāties bez tīmekļa un viedtālruņa, arī programmas tiek rakstītas ar vien vairāk. Uzņēmumi, lai varētu konkurēt ar citiem, ir spiesti ieguldīt finanšu līdzekļus arī programmatūru attīstībā, jo ja mūsdienās, uzņēmumam nav savas mājas lapas, tas netiek uzskatīts par nopietnu tirgus spēlētāju. Potenciālie klienti, izvēloties no kura uzņēmuma saņemt sev vēlamo pakalpojumu, no sākuma apmeklē, tā mājas lapu un izvērtē vai ir vērts ieguldīt savus finanšu līdzekļus tieši šā uzņēmuma sniegtajos pakalpojumos. Protams, vēl aizvien ir spēkā mutvārdu reklāma, kolēģi iesaka kolēģiem, draugi iesaka draugiem, kādu uzņēmumu, ja ir saņēmuši sev vēlamos pakalpojumus labā kvalitātē, bet tā ir ļoti maza tirgus daļa, tāpēc uzņēmumus, kuri vēlas piesaistīt ar vien jaunus klientus, nepieciešams ieguldīt līdzekļus informācijas tehnoloģijās. Mūsdienu klients ir izlutināts, viņš vēlas skaistu, vienkāršu mājas lapu, kurā var ērti atrast sev vēlamo informāciju, vēl papildus bonuss ir tas, ja pakalpojumu ir iespējams pieteikt attālināti – tātad iegādāties e-veikalā. Vēl mūsdienu tehnoloģiju izlutinātajam pircējam ir svarīgi vai ir pieejama arī tīmekļa vietnes mobilā versija.

Uzņēmumi saprotot šo jauno tendenci, investē savus finanšu līdzekļus informācijas tehnoloģijās. Ir uzņēmumi, kuri pie sevi algo programmētājus un ir tādi, kas veic programmēšanas darbu pasūtīšanu no ārpakalpojuma sniedzēja. Visas informācijas sistēmas, kuras tiek piegādātas, vai nu iekšējā izstrādē vai saņemot no ārpakalpojuma sniedzēja, nepieciešams pārbaudīt – testēt. Varbūt var likties, ka saņemot produktu no ārpakalpojuma, testēšanu nav nepieciešams veikt, bet tā nav, jo parasti testēšana (akcepttestēšana) sastāda diezgan lielu izmaksu pozīcijas daļu un tāpēc pasūtītājam (uzņēmuma) būtu pašam jāveic akcepttestēšana. Katrā ziņā, lai arī kurā pusē tiktu veikta testēšana, šim procesam ir nepieciešami laika un cilvēku resursi.

Testēšana mūsdienās ir attīstījusies jau kā atsevišķa zinātne, jo programmatūras testēšanā ir ļoti daudz novirzienu. Attīstoties mobilajām tehnoloģijā, ar vien vairāk tiek pievērsta uzmanība drošības testēšanai. Tiek rīkotas testēšanas konferences visā pasaulē, apmeklējot Programmatūras testēšanas konferenču mājas lapu (Software Testing Conference mājas lapa, 2019), redzams, ka tiek rīkotas konferences visā pasaulē, pulcējot šīs jomas profesionāļus, kuri dalās ar savu pieredzi un labo praksi. Protams, šajās konferencēs neiztikt arī bez reklāmas, tiek reklamēti komerciālie testēšanas rīki, mēģinot uzņēmumiem tos veiksmīgi pārdot. Latvijā šādas konferences rīko LIKTA (Latvijas Informācijas un Komunikācijas Tehnoloģijas Asociācija),

konferences saucas TAPOST. Nākamā šāda konference notiks šā gada oktobrī (Tapost org mājas lapa, 2019). Rīgā notiekošajās konferencēs piedalās testēšanas jomas pārstāvji no visas pasaules. Šeit pulcējas testētāji, kuri jau ir ieguvuši ISTQB (International Software Testing Qualifications Board) sertifikātu, vai plāno to iegūt. ISTQB sertifikāts ir starptautiski atzīts sertifikāts, kas apliecina, ka testētājs ir kvalificēts un zina galvenos testēšanas pamatprincipus (ISTQB org mājas lapa, 2019). Sertifikātu veidi ir dažādi, atkarībā no testēšanas novirziena un pakāpes, ir iespēja iegūt gan pamatsertifikātu (Foundation Level), gan pēc tam iegūt arī augstāka līmeņa sertifikātu. Latvijā zināmi uzņēmumi, kuri nodarbojas ar programmatūru izstrādi/testēšanu, algo testētājus ar šādiem sertifikātiem. Ir uzņēmumi, kuri ir kļuvuši par tā saucamajiem "zelta" partneriem. Tie ir tādi pazīstami uzņēmumi kā TestDevLab (Ventspils augsto tehnoloģiju parks, 2013) un Squalio (Tjarve, 2014). Šis Zelta statuss nozīmē, ka uzņēmums ir ieguvis vismaz 14 sertifikācijas punktus, kuri pieder darbiniekiem vai ārējiem konsultantiem, kuri sniedz pakalpojumus uzņēmumam un velta tam vairāk nekā 70% sava laika, kā arī vismaz vienam darbiniekam jābūt ar augstākā līmeņa sertifikātu (Advanced Level) (ISTQB org mājas lapa, 2019).

Ņemot vērā visu augstāk minēto, redzams, ka testēšanai ir ļoti būtiska loma programmatūras dzīves ciklā un uzņēmumiem, kuriem ir speciālisti, kuri veic šo testēšanu ir svarīga kvalitāte, bet vienmēr ir vēlme samazināt laika resursu, kas tiem tiek veltīts. Bieži vien speciālisti veic manuālo testēšanu, kas aizņem daudz laika resursu, lai samazinātu patērēto laiku testēšanai, risinājums ir veikt automatizētu testēšanu, bet atkal, lai šo veiktu ir nepieciešams ieguldīt resursus darbinieku zināšanās un testēšanas rīkos, jo nav tāda gatava testēšanas rīka, kuru nopērkot tiks atrisinātas visas problēmas, tāpat nāksies sagatavot testēšanas dokumentāciju un veikt izstrādi.

Problēma, kas ir apskatīta šajā darbā ir kā samazināt testēšanai atvēlēto laiku, nepalielinot darbinieku skaitu. Risinājums būtu veikt daļēju manuālo testu automatizāciju. Autore šajā darbā ir izpētījusi kādi ir pieejami testēšanas rīki (atvērtā koda vai bezmaksas), jo konkrētajam uzņēmumam nav līdzekļu, kurus ieguldīt komerciāla rīka iegādei.

Pētījuma objekts ir testējamā vietne new.manspasts.lv un objekta priekšmets ir testēšanas skripti.

Profesionālā bakalaura darba mērķis ir izstrādāt automatizētus skriptus new.manspasts.lv testiem. Lai sasniegtu šo mērķi, tika izvirzīti sekojoši darba uzdevumi:

1. Aprakstīt automatizētās testēšanas teorētiskos aspektus;
2. Raksturot tīmekļa vietni un veikt testēšanas rīku alternatīvu izpēti;
3. Veikt testēšanas skriptu izstrādi;
4. Veikt tīmekļa vietnes testēšanu;

Profesionālā bakalaura darbā tiks izmantotas sekojošas datu ieguves metodes:

1. Artefaktu analīze – testēšanas rīku salīdzināšanai;
2. Monogrāfiskā jeb aprakstošā pētīšanas metode – tiks aprakstīti automatizētās testēšanas teorētiskie aspekti;

Savukārt, datu apstrādei, profesionālā bakalaura darbā tiks izmantotas sekojošas datu apstrādes metodes:

1. Salīdzināšanas metode - tiks salīdzināti testēšanas rīki, testu izpildes rezultātu izpildes laiki;
2. Grafiskā pētījuma metode – dati ar testēšanas rezultātiem tiks atspoguļoti attēlos;
3. Loģiski konstruktīvā pētījuma metode – autors izteiks savus secinājumus un priekšlikumus.

Pētījuma ierobežojumi ir laika periods, kādā tiek veikts pētījums, tas ir profesionālā bakalaura darba izstrādes laika posms no 18.02.2019 – 16.05.2019, papildus ierobežojošais faktors ir tas, ka tiek savstarpēji salīdzināti tikai atvērtā koda testēšanas rīki, neapskatot komerciālo rīku piedāvātās iespējas.

Analītiskā apskata sadaļā autore apskatīs un aprakstīt automatizētās testēšanas teorētiskos aspektus. Šajā sadaļā tiks izklāstīta automatizētās testēšanas vēsture, tiks aprakstīti kādi ir testēšanas standarti un kāda dokumentācija nepieciešama testēšanas procesa ietvaros. Apskatītas tiks arī testēšanas pieejas un to līmeņi, kā arī metodes. Analītiskā apskata beigās tiks aprakstīta automatizētā testēšana.

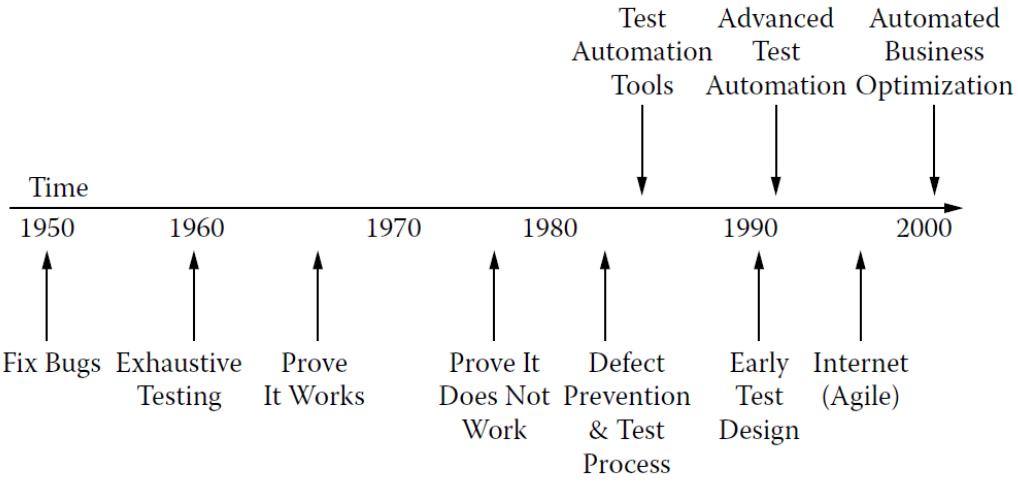
Situācijas izpētes sadaļā autore raksturos testējamo tīmekļa vietni un aprakstīt pašreizējo situāciju, nedaudz pieskarsies uzņēmumam, kuram šī vietne pieder. Tālāk jau tiks apskatīti kādi ir pieejamie atvērtā koda testēšanas rīki. Tie tiks apskatīti katrs atsevišķi un savstarpēji salīdzināti, lai varētu izvēlēties piemērotāko no tiem.

Pēdējā sadaļā – Pētījumu rezultāti, tiks aprakstīts kā veikt izstrādes vides sagatavošanu, testēšanas rīkam, kurš ticis izvēlēts Situācijas izpētes sadaļā. Aprakstīta tiks arī testēšanas skriptu sagatavošana, no plānošanas līdz izstrādei. Tiks aprakstīts kādi ir rezultāti veicot tīmekļa vietnes testēšanu ar izstrādātajiem automatizētajiem skriptiem un kādi ieguvumi salīdzinot testu veikšanu manuāli un automatizēti.

1. ANALĪTISKAIS APSKATS

1.1. Automatizētās testēšanas vēsture

Kopš pirmā datora izgudrošanas 20.gadsimta vidū, un līdz ar to jaunu programmu izstrādi, radās nepieciešamība pēc šo programmu testēšanas. Testēšanas vēstures posmus var aplūkot (1.att.). Sākotnēji testēšana nebija nodalīta no izstrādes un programmētāji paši pārbaudīja savu programmu. Testēšanas vēsturi var izdalīt vairākos posmos, sk. (1.att.). No 1950. līdz 1960. gadam testēšana tika definēta kā "Ko programmētāji darīja, lai atrastu kļūdas savās programmās". Tā vairāk līdzinājās atkļūdošanai un testēšanu un atkļūdošanu lietoja kā sinonīmus (Lewis, 2009).



1. att. Testēšanas vēstures posmi (Avots: Lewis, 2009, 4. lpp.)

Nākamajā posmā sākot ar 1960.gadu – testēšana vairāk līdzinājās revīzijai. Tika veikta programmas pilnīga pārbaude, apskatot visus testējamos ceļus, kodu un visus iespējamos datu ievades variācijas. Šo metodi šodien dēvē kā "Exhautive testing" un viens no šodienas testēšanas pamatlikumiem ir, ka visu notestēt ir neiespējami. Šī metode protams izrādījās neiespējama, jo programmas apjoms bija liels un tika konstatēts, ka ir pārāk daudz iespējamo ceļu, turklāt projektējuma un specifikācijas problēmas bija grūti pārbaudīt (Lewis, 2009).

No 1960.gadu sākuma līdz 1970.gadiem programmu izstrādi dēvēja par datoru zinātni. Savukārt programmu testēšanu definēja "Kas tiek darīts, lai pierādītu, ka programma strādā"

vai arī kā process, kurā tika rasta pārliecība, ka programma darbojas tā kā ir paredzēts. Teorijā šis koncepts bija daudz sološs, bet praksē izrādījās pārāk laikietilpīgs un nepietiekams. Vienkāršos testos bija vienkārši parādīt, ka programma darbojas, kas teorētiski nozīmēja, ka programma arī strādā, tomēr liela daļa no programmām netika testēta, izmantojot šo pieeju lielākā daļa no defektiem tika atklāti programmas lietošanas laikā. Drīz vien tika secināts, ka pareizības pierādīšana nebija efektīva metode programmas testēšanai (Lewis, 2009). Mūsdienās, gan ir saglabājusies vajadzība pēc programmas pareizības demonstrācijas, piemēram akcepttestēšanas.

1970.gadu sākumā tika sākts jauns posms testēšanas vēsturē, kur programmas izpildes mērķis bija atrast kļūdas, nevis pierādīt, ka programma darbojas. Jaunā definīcija noteica to, ka labs testpiemērs ir tāds, kam ir augsta iespējamība atrast vēl neatrastas kļūdas un veiksmīgs tests ir tāds, kura rezultāts ir atklātas vēl neatklātas kļūdas. Šī pieeja ir pilnīgi pretēja iepriekšējai. Lai gan šīs abas pieejas (pierādīt, ka programma strādā un pierādīt, ka nestrādā) bija pilnīgi pretējas, tomēr tām bija kopīgi mērķi:

- Pierādīt, ka produkts darbojas labi;
- Atklāt kļūdas programmā, pirms tas ir pieejams klientiem (Lewis, 2009).

Ja pirmais mērķis ir pierādīt, ka programma strādā, tas nosaka, ka būs neapzināta tendence atlasīt tādus testu datus, kuriem būs zema varbūtība, ka programma nestrādās. Savukārt otrais mērķis ir atklāt kļūdas, bet kā gan var gūt pārliecību, ka programma darbojas, ja mērķis ir pierādīt, ka faktiski programma nedarbojas? Testētāji šodien atzinuši, ka otrais mērķis ir produktīvāks par pirmo, jo pirmajā testētāji neapzināti ignorēs kļūdas un mēģinās pierādīt, ka programma darbojas. Tiek piedāvāti arī šādi labas testēšanas principi (šie ir aktuāli arī mūsdienās):

1. Neatņemama testpiemēra sastāvdaļa ir testa izpildes sagaidāmais rezultāts;
2. Programmētājiem vajadzētu izvairīties no savu programmu testēšanas;
3. Uzņēmumiem, kuri nodarbojas ar programmu izstrādi, nevajadzētu testēt savas programmas;
4. Rūpīgi pārbaudīt katra testa rezultātu;
5. Testpiemēram vajadzētu saturēt derīgus un paredzamus, kā arī nederīgus un neparedzamus nosacījumus un ievaddatus;

6. Programmu pārbauda, lai redzētu vai programma dara to, kas tai jādara, kā arī pārbaude vai tā nedara ko tādu, kas tai nebūtu jādara;
7. Izvairīties no testpiemēru izmešanas līdz brīdim, kad pati programma ir izmesta;
8. Neplānot testēšanu ar domu, ka netiks atrastas kļūdas;
9. Varbūtība, ka programmas daļā ir vairāk kļūdas, ir proporcionāli tajā sadaļā jau konstatēto kļūdu skaitam (Lewis, 2009).

1980.gadu sākumā dienas gaismu ieraudzīja ideja, kas noteica, ka testēšana jāpapildina arī ar defektu novēršanu. Savukārt testu izstrāde ir viena no efektīvākajām kļūdu novēršanas tehnikām. Tika ierosināts veikt testēšanas metodoloģiju un noteikt, ka testēšana un caurskate jāiekļauj visā programmatūras dzīves ciklā un ka testēšanas process ir jāpārvalda. Testēšanas nozīme neaprobežojās tikai ar programmu testēšanu, bet iekļāva arī prasības, projektējumu, kodu, paštestus un programmu (Lewis, 2009).

Testēšana tradicionālajā nozīmē tikai 80.gadu sākumā nozīmēja "Kas tika darīts ar sistēmu, kad tā darbojās un kods tika piegādāts". Tomēr testēšana šodien ir apjomīgāka testēšana, kurā testētājs tek iesaistīts visos programmatūras dzīves cikla posmos. Tiklīdz kods ir piegādāts testēšanai, tas tiek testēts un pārbaudīts, bet ja kaut kas ir nepareizi, jāizpēta iepriekšējie izstrādes posmi. Ja kļūda radusies projektēšanas neskaidrības vai programmētāja kļūmes dēļ, ir vieglāk atrast problēmu tiklīdz tā radusies, nevis gaidīt, kad programma tiks piegādāta produkcijā (Lewis, 2009).

Pētījumi rāda, ka aptuveni 50% no kļūdām tiek radītas prasībās (nosakot, kas programmai jādara) vai projektēšanas fāzē un tās var radīt nopietnu ietekmi un radīt vairāk kļūdu tieši programmēšanas fāzē. Jo agrāk kļūda tiek atrasta programmēšanas dzīves ciklā, jo lētāk šo kļūdu ir izlabot. Pirms veikt programmas pārbaudi un meklēt tajā kļūdas, būtu nepieciešams stingri pārskatīt prasības un projektējums (Lewis, 2009).

1980.gadu vidū tika izveidoti automatizēti testēšanas rīki, lai automatizētu manuālo testēšanu, ar mērķi uzlabot testējamās programmas efektivitāti un kvalitāti. Bija paredzēts, ka dators varētu veikt vairāk programmas testus nekā cilvēks manuāli un ar lielāku precizitātes efektivitāti. Šie rīki sākotnēji bija diezgan primitīvi un tiem nebija uzlabotas skriptu valodas iespējas (Lewis, 2009).

90. gadu sākumā tika atzīts agrīnās testēšanas iesaistes spēks. Testēšana tika pārdefinēta un tagad to saprata kā plānošanu, projektēšanu, izstrādi un testu izpildi. Tā bija testēšanas

kvalitātes nodrošināšanas perspektīva, kas noteic, ka laba testēšana ir pārvaldīts process, kas attiecināms uz visu programmatūras dzīves ciklu. Parādījās arī uzlabotie ierakstīšanas un atskaņošanas testēšanas rīki, kas piedāvāja vairākas skriptu valodas un pārskatus. Tika radīti arī testēšanas pārvaldīšanas rīki, kas palīdzēja pārvaldīt prasības, projektēt testēšanu, izstrādāt skriptus un apkopot kļūdas. Bija pieejami arī komerciālie testēšanas rīki, kas bija paredzēti veiktspējas testiem. Šie rīki testēja programmas stresa un slodzes apstākļos, lai pārbaudītu kuri ir tie kritiskie punkti (datu apjoms, u.c. faktori), pie kuriem sistēma nedarbojas kā plānots. Šos testus varētu saukt arī par kapacitātes plānošanu (Lewis, 2009).

1.2. Testēšanas standarti un dokumentācija

Starptautiskie standarti, kuri attiecas uz testēšanu ir (Spillner u.c., 2014):

- IEEE 829-2008 – standarts programmatūras un testēšanas dokumentācijai;
- IEEE 1028-2008 – standarts programmatūras caurskatīšanai un auditam;
- IEEE 12207-2008 – standarts sistēmas un programmatūras inženierija – programmatūras dzīves cikla procesi;
- ISO/IEC 9126-1:2001 – programmatūras inženierija – programmatūras produkta kvalitāte.

Programmatūras testēšanas dokumentāciju nosaka sekojoši standarti Latvijas standarti:

LVS 70:1996 Informācijas tehnoloģija - Programminženierija - Programmatūras testēšanas dokumentācija, kas nosaka, ka testēšanas laikā var tikt izstrādāti sekojoši dokumenti (Latvijas standarts, 2019):

- Testēšanas plāns – tiek definēta darbības sfēra, testēšanas pieeja, nepieciešamie resursi, plānotais laiks testēšanai, tiek uzskaitīti testējamie vienumi, testējamās raksturiezīmes. Plānā tiek minēti testēšanas uzdevumi un atbildīgās personas.
- Testu projektējuma specifikācija – dokumentā detalizēti tiek aprakstīta testēšanas pieeja un testējamās raksturiezīmes;
- Testpiemēru specifikācija – definēti testpiemēri, kuri minēti testu projektējuma specifikācijā. Šajā dokumentā tiek minēti testējamie vienumi, ievades specifikācija, izvades specifikācija, vides vajadzības un speciālās procedūras prasības, kā arī atkarības starp piemēriem;
- Testēšanas procedūras specifikācija – tiek aprakstīts nolūks, speciālās prasības un procedūras soļi;

- Testējumā vienuma pavadzīme – parasti sastāda izstrādātājs, tiek definēti nosūtāmie vienumi, atrašanās vieta, statuss un apstiprinājumi;
- Testēšanas žurnāls – norāda testu aprakstu, darbības un pierakstus;
- Testa problēmas ziņojums – šajā ziņojumā norāda atklātās problēmas kopsavilkumu, problēmas aprakstu un ietekmi;
- Testēšanas kopsavilkuma pārskats – norāda kopsavilkumu, novirzes, aptvēruma novērtējumu, rezultātu kopsavilkumu, novērtējumu, darbību kopsavilkumu un apstiprinājumu.

LVS 73:1996 Informācijas tehnoloģija - Programminženierija – Programmatūras

Vienībtestēšana (Latvijas Standarts, 2019). Standarts definē integrētu pieeju sistemātiskai un dokumentētai vienībtestēšanai. Šis standarts apraksta testēšanas procesu, ko veido etapu, darbību un uzdevumu hierarhija, un definē minimālo uzdevumu kopu katrai darbībai. Standartā nav definēts atkļūdošanas process. Šo standartu var lietot jaunizstrādātu un modificētu vienību testēšanai.

- Programmatūras vienībtestēšana ir process, kas ietver testēšanas plānošanu, testu kopas iegūšanu un testējamās programmatūras vienības mērīšanu, salīdzinot testējamo vienību ar tai noteiktajām prasībām. Mērīšana nozīmē datu paraugu izmantošanu, lai izpildītu vienību un salīdzinātu vienības faktisko uzvešanos ar to uzvedību, kas specificēta vienības prasību dokumentācijā.

1.3.Testēšanas pieejas un līmeņi

Lai labāk izprastu automatizēto testēšanu ir nedaudz jāapskata testēšana kā tāda. Testēšana konstatē problēmas programmatūrā un atklāj iespējamos riskus, kas saistīti ar programmatūras izplatīšanu ekspluatācijā. Pastāv divu veidu programmatūras testēšanas pieejas.

Pirmo varētu dēvēt par “tradicionālo” (Programmatūras testēšanas rokasgrāmata vadītājiem, 2014) – tā ir balstīta uz programmatūras prasībām un specifikācijām. Lai veiktu testēšanu tiek veiktas sekojošas aktivitātes:

- testēšanas plānošana;
- testpiemēru specificēšana;
- testpiemēriem nepieciešamo datu izveide;
- testpiemēru izpilde.

Šai pieejai, pirms tiek atklāta pirmā kļūda programmatūrā, nepieciešams ieguldīt lielu darbu. Tā ir piemērota lieliem un ilgiem projektiem, testēšanas komanda projektā tiek iesaistīta agri un tas ļauj veikt programmatūras detalizētu analīzi. Izstrādes cikli šai pieejai būs pakāpeniski.

Otra testēšanas pieeja tiek dēvēta par "spējo" (Programmatūras testēšanas rokasgrāmata vadītājiem, 2014) testēšanu. Tiek izvērtēti riski un programmatūras konteksts, pirms nodošanas ekspluatācijā tiek rūpīgi izvērtēts, kādas testēšanas aktivitātes tiks veiktas. Šo metodi izmanto mazāka izmēra programmatūrās ar īsākiem laika periodiem.

Testēšanas definīcija (Spillner u.c., 2014) - testēšana ir process, kurš:

- pārklāj visu programmatūras dzīves ciklu;
- iekļauj statistiskas un dinamiskas aktivitātes;
- sastāv no:
  - plānošanas;
  - sagatavošanas;
  - darba produktu novērtēšanas.
- veikts ar mērķi:
  - nodrošināt, lai darba produkti atbilst prasībām;
  - demonstrēt, ka darba produkti atbilst savam nolūkam;
  - atklāt defektus.

Testēšanas mērķi arī varbūt dažādi, piemēram – atklāt defektus, pārbaudīt produkta kvalitāti, palīdzēt pieņemt lēmumu vai novērst defektus.
Kļūda (*error, mistake*) – cilvēka darbība, kuras rezultātā veidojas nekorekts rezultāts. Savukārt *Defekts* (*defect, bug, fault*) ir sistēmas vai komponenta nepilnība, kas potenciāli var izraisīt sistēmas vai komponenta kļūmi (nepareizu darbību). *Kļūme* (*failure*) – sistēmas vai komponenta darbības novirze no sagaidāmā (Spillner u.c., 2014). Saskaņā ar ISTQB (International Software Testing Qualifications Board) pastāv septiņi testēšanas principi (Spillner u.c., 2014):

1. Testēšana parāda defektu esamību. Tas nozīmē, ka testēšana parāda, ka ir defekti, bet negarantē, ka vairs defektu nav. Testēšana var samazināt defektu esamības varbūtību.
2. Izsmeļoša testēšana nav iespējama, kas nozīmē, ka visu (visus iespējamos ievaddatus, visus pirms nosacījumus un visas kombinācijas) notestēt nav iespējams. Izņemot

protams triviālus gadījumus. Tāpēc testēšanas aktivitātes jāveic atbilstoši riskiem un prioritātēm.

3. Testēšana jāsāk agri, jo agrāk atrastie defekti izmaksā daudz mazāk nekā, ja tos atrod vēlākās fāzēs.
4. Defekti grupējas (klasterizējas). Defekti nav vienmērīgi sadalīti, ja tiek atrasti defekti, visticamāk, ka ir arī citi. Lielākā daļa defektu ir atrodami dažās testa objekta daļās.
5. Pesticīda paradokss – ja tos pašus testus izpilda atkal un atkal, jauni defekti netiks atklāti. Testēšanas piemērus nepieciešams pārskatīt un atjaunot, lai tiktu atklāti potenciālie defekti. Tātad nepieciešams izmantot savādākus testa ievaddatus un datu kombinācijas, kā arī jāsagatavo jauni testēšanas scenāriji.
6. Testēšana ir atkarīga no konteksta – balstoties uz testējamā objekta kontekstu, nepieciešams izvēlēties atbilstošu testēšanas pieeju un stratēģiju.
7. Maldīga defektu neesamība – defektu neesamība nepalīdzēs, ja sistēma neatbildīs lietotāja/pasūtītāja prasībām un gaidām. Risinājums – iepazīstināt lietotāju ar sistēmu ātrāk, rādot sistēmas prototipus.

Dažādos programmatūras izstrādes dzīves cikla posmos, testēšanu veic (Spillner u.c., 2014):

- izstrādātāji – vienībtestēšanu;
- lietotāji – akcepttestēšanu;
- neatkarīgie testētāji – integrācijas un sistēmtestēšanu.

Par testēšanu ir tāds teiciens: “Pēdējo kļūdu nekad neatradīsi”, tas nozīmē, ka pilnībā notestēt programmatūru nav iespējams. Piemēram, ja sistēmā ir operācija ar svara diapazonu no 1-10 kg, tad, lai pārbaudītu pilnīgi visu diapazonu būtu jāvada 1 kg, 2 kg, utt. Tas prasītu ļoti daudz laika un nebūtu arī efektīvi. Tāpēc testēšanā nepieciešams izvēlēties tādus testpiemērus, kuri pārklātu visu testējamo operāciju kopumu, bet būtu arī reāli izpildāmi.

Testēšanu var iedalīt manuālā testēšanā un automatizētā testēšanā. Manuālā testēšanas gadījumā, testēšana tiek veikta manuāli, testētājs izpilda testpiemēros paredzētās darbības – ievada datus un salīdzina programmatūras darbību, ar testpiemērā paredzēto sagaidāmo rezultātu.

Automatizētās testēšanas gadījumā, programmatūras pārbaudei tiek izmantoti skripti, kuros jau ir definētas izpildāmās darbības un sagaidāmais rezultāts.

Sīkāk testēšana iedalās atkarībā no skata punkta, kas tiek testēts un kas veic testēšanu. Tās ir Melnās, Baltās un Pelēkās kastes testēšana. Melnās kastes metode nozīmē – literāriem vārdiem sakot "Sist pa kasti un skatīties, kas notiks", bet Baltās kastes metode nozīme "Skatīties kastē iekšā". Tas nozīmē, ka Melnās kastes testēšanas gadījumā par programmatūru ir zināms kā tai būtu jādarbojas un tiek izmantoti programmatūras prasību apraksti, specifikācijas un cita dokumentācija, bet netiek izmantota informācija par programmatūras pirmkodu. Savukārt Baltās kastes gadījumā testēšanai tiek izmantots programmas kods (Dustin u.c.,1999).

Melnās kastes testēšanas metodes (Dustin u.c.,1999):

- ✓ ekvivalences klases;
- ✓ robežvērtību analīze;
- ✓ lēmumu tabulas;
- ✓ stāvokļu pāreju testēšana;
- ✓ lietošanas piemēru bāzēta testēšana.

Baltās kastes testēšanā izmanto programmatūras pirmkoda informāciju, tā struktūru. Baltās kases testēšanas metodes ietver (Dustin u.c.,1999):

- ✓ operatoru pārklājums;
- ✓ risinājumu pārklājums;
- ✓ nosacījumu pārklājums;
- ✓ risinājumu un nosacījumu pārklājums;
- ✓ kombinatoriskais nosacījumu pārklājums.

Pelēkās kastes testēšanā izmanto abas iepriekš aprakstītās pieejas, tātad Melnās un Baltās kastes testēšanas principus. Šajā metodē testētājam ir pieejamas daļējas zināšanas par programmatūras kodu un šajā metodē testēšana tiek veikta no ārpuses, izmantojot specifikācijas, bet papildus ir arī ierobežotas zināšanas par kodu. Nosaukums "Pelēkā kaste" ir radies no tā, ka programmatūra testētājam ir kā caurspīdīga kaste un testētājs tajā var ieskatīties, bet nevar redzēt pilnīgi visu. Pelēkās testēšana var ietvert arī reverso inženieriju, lai noteiktu piemēra, robežvērtības vai kļūdu paziņojumus (Faulkner, 2009).

Programmatūras dzīves cikla laikā programmatūru iespējams testēt vairākos līmeņos (2.att.), vairākas reizes. Izšķir sekojošus līmeņus (Sandeep & Abhishek, 2016):

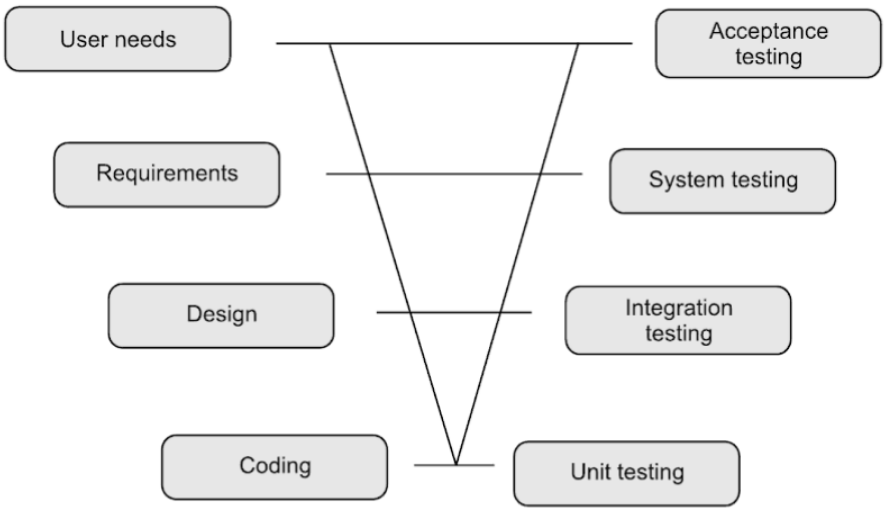
Vienībtestēšana (Unit testing) – nelielas programmatūras daļās pārbaude (procedūras, funkcijas, metodes vai klases), veic izstrādātāji (programmētāji), atrastās kļūdas parasti nereģistrē. Kļūdas tiek novērstas un tiek turpināta izstrāde.

Integrācijas testēšana (Integration testing) – parasti tiek veikti pēc vienībtestiem un tiek pārbaudītas vairāku atsevišķi izveidotu programmatūras daļu savstarpējā sadarbība, piemēram vai sistēmas daļa A saņem datus no sistēmas daļas B. Šos testus arī parasti veic izstrādātāji, vai liela apjoma programmatūras gadījumā – testētāji.

Sistēmtestēšana (System testing) - visas programmatūras pārbaude, apstākļos, kas ir maksimāli pietuvināti ekspluatācijas videi. Šajos testos tiek pārbaudīta sistēmas funkcionalitāte, drošība, veiktspēja u.c. Sistēmas testēšana ir nozīmīgākā un viena no pēdējām fāzēm. Arī šī līmeņa testus veic testētāji.

Akcepttestēšana (Acceptance testing) – pēc veiksmīgiem sistēmas testiem, sistēma ir gatava nodošanai pasūtītājiem un šos testus parasti veic programmatūras pasūtītājs, lai pārliecinātos, vai visa programmatūra darbojas atbilstoši izvirzītajām prasībām. Šajos testos tiek izmantoti scenāriji, kuri atbilst biznesa procesiem, kuriem sistēma tika radīta. Akcepttestēšanu var iedalīt vēl zemākos līmeņos - funkcionālie akcepttesti, produkcijas akcepttesti un lietojamības testi.

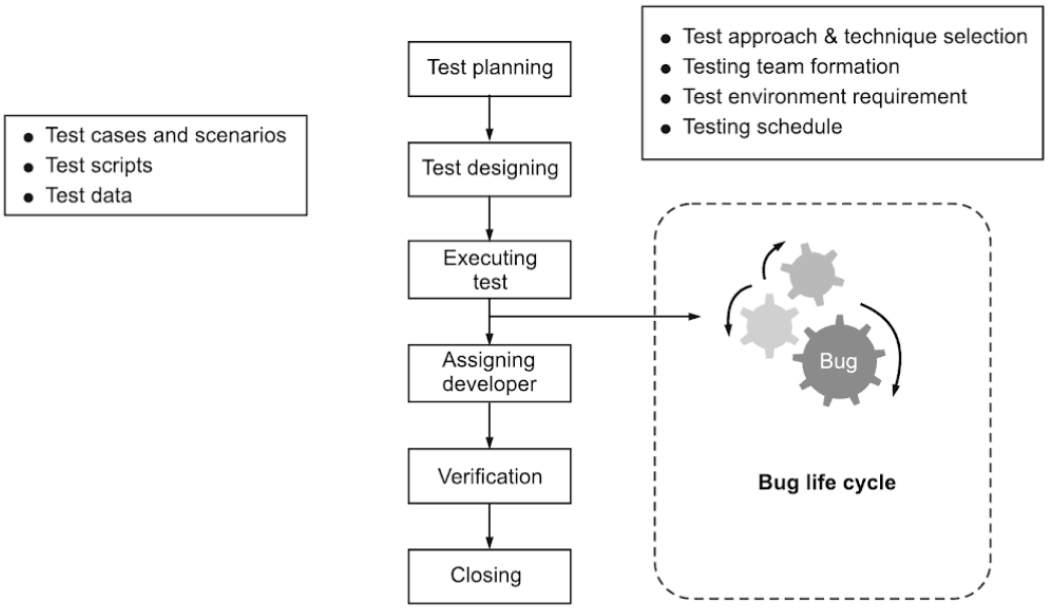
Regresa testi – šos testus veic, ja sistēmai, kura jau ir pieņemta produkcijā, veic izmaiņas. Šo testu mērķis ir pārbaudīt iepriekš testētu funkciju pārtestēšana, lai pārbaudītu vai jaunu operāciju, funkciju realizācija programmatūrā nav sabojājusi esošās "vecās" operācijas un funkcijas, kurās izmaiņas netika veiktas.



2. att. Testēšanas līmeņi (Avots: Sandeep & Abhishek, 2016, lpp.64)

Kā raksta autors R.Chopra savā grāmatā Software Testing: Self-Teaching Introduction, ka Programmatūras testēšana ir kā māksla. Un lielākā daļa metožu un pieeju būtiski nav mainījušās pēdējo 20 gadu laikā un veiksmīga testēšana ir ļoti atkarīga no testētāja kreativitātes, pieredzes un intuīcijas, kas tikai papildina pārējās izmantojamās testēšanas tehnikas (Chopra, 2018).

Testēšanas process ir iekļauts programmatūras dzīves ciklā, bet arī testēšanas procesam kā tādam ir savs dzīves cikls (3.att.). Viss sākas ar testēšanas plānošanu, kurā tiek izvēlēta testēšanas pieeja un tehnikas, tad tiek veikta testēšanas modelēšana, kas sevī ietver testēšanas scenāriju un testpiemēru ar testēšanas datiem sagatavošana. Testu izpildes laikā parasti tiek konstatētas kļūdas un tās tiek nodotas labošanai programmētājiem, pārbaude testēšanas process, tiek pabeigts.



3. att. Testēšanas dzīves cikls (Avots: Sandeep & Abhishek, 2016, lpp.62)

1.4. Automatizētā testēšana

Automatizētā testēšana ir informācijas sistēmu pārbaude, izmantojot testēšanas rīkus vai skriptus. Testu automatizāciju var iedalīt divos līmeņos API (*Application Programming Interface* - lietojumprogrammas saskarne) un GUI (*Graphical User Interface* - grafiskā lietotāja saskarne) (Testēšanas pamati, 2010).

API līmenī tiek testētas iekšējās saskarnes, veikti vienībtesti, integrācijas testi vai sistēmas testi. Emulē citas komponentes. API līmeņa testus parasti veic izstrādātāji. Tie tiek rakstīti

programmēšanas valodā un izmantoti vienībtestēšanas ietvaros (*frameworks*) (Testēšanas pamati, 2010).

Vienībtestēšana – pārbauda funkcionēšanu un meklē defektus vienumos, kas ir atsevišķi trasējami. Parasti veic programmētāji, labojot defektus tiklīdz tie tiek atrasti.

Testu izpilde nav novērojama – redzami tikai rezultāti. Testējamo vienumu rezultāti ir tas, ko atgriež API funkcijas (Testēšanas pamati, 2010). Vienībtestēšanu veic arī izmantojot automatizētus skriptus, kuri veic pārbaudes pret žurnāldrukām, pirms tam ieslēdzot noteiktu kļūdu paziņojumu līmeni.

GUI līmenī tiek testēts lietotāja saskarne, tie ir funkcionālie testi. Emulē cilvēku – testētāju vai lietotāju. Lai veiktu GUI līmeņa testus nepieciešami speciāli rīki. Parasti nav atkarīgi no programmēšanas valodas. Ir atkarīgi no lietotāja interfeisa izmaiņām. Emulē reālu lietotāju, tāpēc testu izpildes laikā var vizuāli novērot veicamās darbības. Testējamo vienumu rezultāti ir tas, kas parādās vizuālajā saskarnē (Testēšanas pamati, 2010).

Jāatceras kāda būtiska lieta – nevar automatizēt procesu, kurš nav sakārtots un atbilstoši definēts. Testpiemēriem jābūt izstrādātiem un aprakstītiem. Ja manuālais process nebūs sakārtots, automatizācija šo problēmu neatrisinās.

Automatizētā testa struktūra sastāv no (Testēšanas pamati, 2010):

Darbībām

- ✓ Soļi, kuros tiek emulētas cilvēka darbības ar sistēmu;
- ✓ Pogu nospiešana, datu ievadīšana, u.c.

Pārbaudes:

- ✓ Soļi, kuros tiek novērtēta lietotāja saskarnes stāvokļa atbilstība sagaidāmajam;
- ✓ Lauku vērtības, vizuālo objektu stāvokļi (piem. aktīvs vai neaktīvs).

Datiem:

- ✓ Ievaddatiem;
- ✓ Sagaidāmajiem rezultātiem.

Vispiemērotākie testi automatizācijai būtu (Testēšanas pamati, 2010):

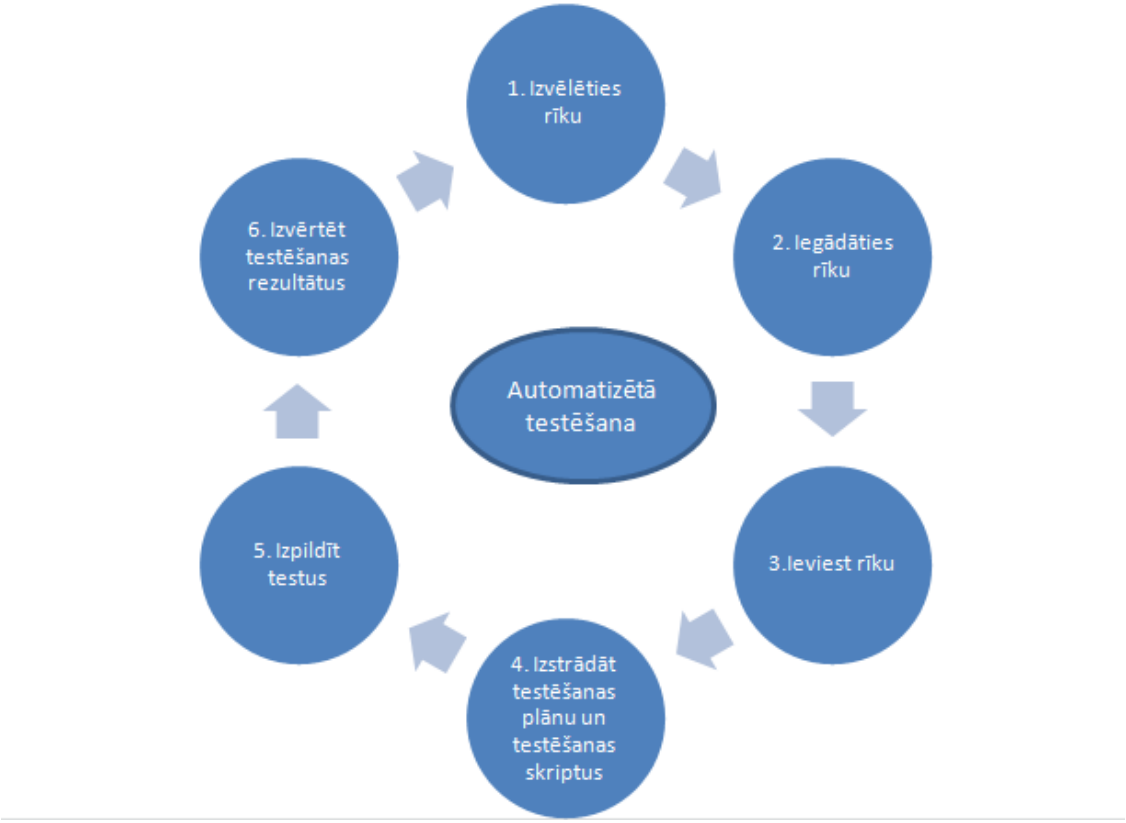
- regresa (atkārtoti darbināmie funkcionalitātes) testi.

- stresa un slodzes testi - pie pietiekami liela datu apjoma un lietotāju skaita šī veida testus manuāli īstenot nemaz nav iespējams, vai arī to izmaksas ir ievērojami lielākas nekā automatizēšanas gadījumā.
- konfigurācijas testi, kas var tikt darbināti uz vairākiem desmitiem un simtiem atšķirīgu konfigurāciju.
- testi, kas izpildāmi bez cilvēka klātbūtnes, piemēram, naktīs.

Vienkāršākie no testu automatizācijas rīku veidiem ir – ierakstošie/nospēlējošie (*capture/replay*) rīki. Rīka darbības princips ir atcerēties (ierakstīt) lietotāja izpildītās darbības, ievadītās vērtības, atskaņojot ierakstu, rīks izpilda un pārbauda iepriekš izpildītās darbības, gadījumā, ja nav iespējams izpildīt ierakstīto darbību, izdod kļūdu.

Cita testu automatizācijas rīku grupa ir rīki, kas analizē programmas kodu. Šie rīki ir piesaistīti pie konkrētām programmēšanas valodām un to versijām (Testēšanas pamati, 2010). Vēl viena grupa ir rīki slodzes un stresa testēšanai. Šīs grupas rīki emulē daudzu lietotāju darbības, radot paaugstinātu servera noslodzi.

Lai veiktu automatizētās testēšanas ieviešanu, nepieciešams izvēlēties testēšanas rīku, atbilstoši vajadzībām un kompetencēm, kāda ir uzņēmumā. Ja testēšanas rīks ir komerciāls, tad iegādāties to, vai arī izvēlēties atvērtā koda rīku par kuru nebūs jāmaksā. Pēc tam nepieciešams izstrādāt testēšanas plānus un testēšanas uzdevumus. Atbilstoši kāds ir šis izvēlētais testēšanas rīks, izstrādāt testēšanas skriptus un izpildīt tos. Pēc tam ir nepieciešams izvērtēt testēšanas rezultātus - tā ieguvumus, noderību, utml., automatizētās testēšanas ieviešanas procesu sk. (4.att.).



4. att. Automatizētās testēšanas ieviešanas process (autores veidots)

Jāatceras, ka nepieciešams arī uzturēt izstrādātos testēšanas skriptus, jo mainoties funkcijām, šie skripti būs jāatjauno, lai tie izpildītos un neatgriezu atbildi "FAIL" tikai tāpēc, ka pašā testējamā programmā kaut kas ir mainījies.

Jāsaprot arī tas, ka automatizētās testēšanas rīks, nebūs burvju nūjiņa, kas atrisinās visas nepieciešamās vajadzības. Daudzi uzņēmumi maldīgi sagaida, ka ieviešot automatizētu testēšanas rīku, tas pats visu testēs un nebūs vairs jāsastāda testa plāni, un jāveic pārējās aktivitātes, tāpēc ļoti būtiski uzņēmuma vadību informēt par automatizācijas procesa ieguvumiem un kādas darbības, atkarībā no manuālās testēšanas turpmāk nāksies veikt. Tas ļaus izvairīties no pārpratumiem un visas iesaistītās puses sapratīs, kas tieši mainīsies ieviešot daļēju automatizēto testēšanu.

2. SITUĀCIJAS IZPĒTE

2.1.Tīmekļa lietotne new.manspasts.lv

Uzņēmumam ir svarīga kvalitatīva tīmekļa vietne, jo to apmeklējot klienti izdara savus secinājumus arī par uzņēmumu, kuram šī vietne pieder.

VAS "Latvijas Pasts" ir valsts akciju sabiedrība, kuras 100% kapitāla daļas pieder valstij.

Sabiedrības pamatdarbības veids ir pasta pakalpojumu sniegšana, ekspresspasta pakalpojumi, finanšu starpniecība, mazumtirdzniecība, abonēto preses izdevumu piegādes pakalpojumi un ar to atbalstu saistītās darbības (VAS „Latvijas Pasts" mājas lapa, 2019).

Atrašanās vieta: Ziemeļu iela 10, Lidosta „Rīga", Mārupes novads, LV-1000.

Saskaņā ar Pasta likumu Pārejas noteikumu 6.punktu VAS "Latvijas Pasts" no 2013.gada 01.janvāra vairs nav spēkā rezervētās īpašās tiesības pieņemt, pārsūtīt un izsniegt vēstuļu korespondences sūtījumus, kuru svars nepārsniedz 50 gramus, līdz ar to Sabiedrība darbojās brīvā tirgus apstākļos visās universālā pasta pakalpojuma grupās (SPRK mājas lapa, 2019).

Atbilstoši spēkā esošo normatīvo aktu prasībām universālā pasta pakalpojuma ietvaros VAS „Latvijas Pasts" ir jānodrošina šādu pasta pakalpojumu pieejamība visā valstī:

- ✓ iekšzemes un pārrobežu vēstuļu korespondences sūtījumu (tajā skaitā ierakstītu un apdrošinātu sūtījumu) savākšana, šķirošana, pārvadāšana un piegāde, kuru svars nepārsniedz divus kilogramus;
- ✓ iekšzemes un pārrobežu pasta paku (to skaita apdrošinātu) savākšana, šķirošana,
- ✓ pārvadāšana un piegāde, kuru svars nepārsniedz desmit kilogramus;
- ✓ pārrobežu pasta paku (to skaita apdrošinātu) piegāde, kuras saņem no citām Eiropas Savienības valstīm un kuru svars nepārsniedz divdesmit kilogramus;
- ✓ abonēto preses izdevumu piegāde.

Tīmekļa lietotne Mans Pasts ir VAS "Latvijas Pasts" bezmaksas pašapkalpošanās vietne, kurā iespējams izmantot dažādus pakalpojumus. Šajā vietnē gan privātpersonas, gan juridiskas personas var veikt sūtījumu noformēšanu pirms došanās uz pasta nodaļu. Šajā vietnē iespējams atrast informāciju par to, ko drīkst sūtīt sūtījumos, kādi sūtījumu veidi ir pieejami. Turklāt, noformējot sūtījumus nav nepieciešams zināt, kādas tieši veidlapas būs nepieciešams aizpildīt, nepieciešams izvēlēties no adrešu grāmatiņas adresātus, izvēlēties sūtījuma veidu un norādīt

visus nepieciešamos papildus parametrus un sistēma pati uzģenerēs gan sūtītāja, gan adresāta gan visas pārējās nepieciešamajam sūtījuma veidam vajadzīgās veidlapas. Ar šādu noformētu sūtījumu pasta nodaļa vairs nenāksies pavadīt ilgu laiku. Sūtījumu vēsture ir nodrošināta iespēja sekot saviem noformētajiem sūtījumiem, kas ir diezgan ērti. Līguma klientiem ir papildus dažādas funkcijas, tādas kā - iespēja pieteikt dažādus pieteikumus (sūtījumu pāradresēšana, adresāta maiņa un citus līdzīgus pieteikumus). Ir iespēja arī veikt līguma izmaiņas, piesakot papildus pakalpojumus, kā arī noslēgt jaunus līgumus. Vietnē iespējams arī uzdod jautājumus un sazināties ar Latvijas Pasts nosūtot ziņu, izmantojot Mans Pasts funkcijas.

2.2. Esošās situācijas apraksts

Kā jau visi uzņēmumi arī Latvijas Pasts pilnveidojas un veic uzlabojumus savu pakalpojumu sniegšanā, un arī vietni Mans Pasts tiek regulāri uzlabota. Mans Pasts izstrādi veic gan ārpakalpojumu sniedzējs, gan Latvijas Pasts programmētāji – atkarībā no ieguldāmā darba apjoma. Pēc katru uzlabojumu veikšanas sistēmā, nepieciešams izmaiņas un funkcijas, kurās tika veiktas un netika veiktas izmaiņas pārbaudīt – testēt. Esošajā situācijā visu izmaiņu pārbaude notiek veicot manuālo testēšanu, izmantojot melnās kastes metodi. Visas sistēmas (funkcijas, kuras minētas testēšanas plānā) testēšana aizņem 7 dienas. Testēšana tiek veikta pēc izstrādātā testēšanas plāna, kurā definētas funkcijas kuras tiek testētas. Septiņas dienas ir ilgs laiks, jo ja testēšanas laikā tiek konstatētas kļūdas (un gandrīz vienmēr tās tiek konstatētas), tiek paildzināts vēlamā pakalpojuma palaišanas produkcijā laiks. Savukārt pēc kļūdu labojumiem, saņemot jaunu nodevumu, to nepieciešams pārbaudīt – testēt, jo kā zināms, katras izmaiņas programmatūras kodā, rada iespēju, ka tiks konstatētas kļūdas, it sevišķi, ja ir sarežģīts aprēķina algoritms, kur ir iesaistīti vairāki moduļi. Lai uzlabotu esošo situāciju un paātrinātu pakalpojuma ieviešanas procesu produkcijā, nepieciešams samazināt testēšanai atvēlēto laiku. Ir vairāki risinājumi – palielināt testētāju skaitu vai veikt regresa testu automatizāciju. Lai palielinātu testētāju skaitu būs jāveic personāla meklēšana, kas pirmkārt aizņems daudz laika un otrkārt radīs papildus izmaiņas. Labākais risinājums, ja testētāju skaitu nav plānots palielināt ir veikt regresa testu automatizāciju. Šis risinājums arī nav vienkāršs un protams, visu uzreiz automatizēt neizdosies, jo arī automatizācijai ir nepieciešami laika un cilvēku resursi, turklāt arī izstrādātie automatizācijas testi ir jāpārbauda un vajadzības gadījumā jāpielabo. Vēl jāņem vērā, ka ja tiks veiktas izmaiņas jau automatizēto testu apgabalā, būs nepieciešams arī pielāgot/mainīt automatizētos testus, tātad automatizācijas rīkam jābūt fleksiblam un viegli papildināmam.

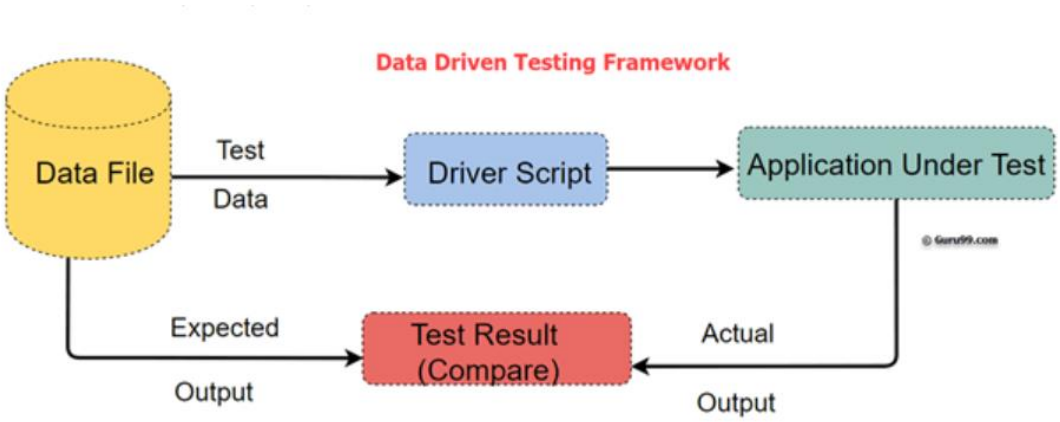
2.3.Testēšanas rīku apraksts un izvēle

Testēšanas rīku skaits mērāms vairākos simtos un tie attiecīgi iedalās pēc to veicamajiem uzdevumiem. Tie ir testēšanas pārvaldības rīki, prasību pārvaldības rīki, pārskatu ģenerēšanas rīki, statiskās analīzes rīki, modelēšanas rīki, kā arī testu specificēšanas rīki un testu izpildes rīki. Ir rīki, kuri paredzēti testēšanas vižu pārvaldībai, pārklājuma pārvaldīšanai, datu salīdzināšanai. Rīki arī paredzēti drošības testēšanai un slodzes testēšanai, kā arī ir daudzi citi rīki. Izpētot pieejamo testēšanas rīku piedāvājumu, autore secina, ka tīmekļa risinājumu testēšanai ir diezgan daudz rīku. Tie attiecīgi iedalās maksas un bezmaksas. Sakarā ar to, ka uzņēmums nav paredzējis izdalīt papildus finansiālos līdzekļus testēšanas rīku iegādei, nepieciešams meklēt bezmaksas vai atvērtā koda risinājuma rīkus.

Meklējot tīmeklī informāciju, autore atrada vietni http://www.qatestingtools.com/, kurā ir apkopota informācija par automatizētās testēšanas rīkiem, kuri standartizēti pēc to pielietojuma. Šajā vietnē iespējams atlasīt rīkus pēc dažādiem parametriem – pēc testēšanas mērķa (aplikācijas veida, piemēram tīmekļa vai darbvirsmas), pēc testēšanas metodes (piemēram, integrācijas testiem) vai regresa testiem. Atlasot pēc mērķa "Web testing" – tīmekļa rīku testēšana tiek atgriezta informācija par 199 rīkiem – tie ir gan komerciālie, gan atvērtā koda rīki. Lai samazinātu pētāmo rīku skaitu, atlases kritērijos tika definēts, ka nepieciešami tikai atvērtā koda rīki, rezultātā tika atgriezts saraksts ar 83 rīkiem, papildus vēl atfiltrējot rīkus, kuri ir aktīvi (tātad tiek nodrošināti rīku uzlabojumi un atbalsts), rīku skaits vēl tika samazināts tādejādi pētāmo rīkus sarakstā palika 16.

Šie testēšanas rīki tiks salīdzināti un tiks izvēlēts piemērotākais testēšanas rīks pēc sekojošiem kritērijiem – vai rīkam ir pieejama IDE (Integrated development environment - Integrētā izstrādes vide), vēlams būtu Eclipse izstrādes vide, jo testu izstrādātājam jau ir pieredze ar šo izstrādes rīku. Svarīga ir arī pieejamā testēšanas metode, kā arī skriptu programmēšanas valoda – vēlams Java, jo autorei ar citu programmēšanas valodu ir mazāk zināšanu. Protams svarīga arī ir pieejamā dokumentācija un atbalsta un apmācības iespējas. Vēl rīkam vajadzētu būt iespējai testēšanas datus ielādēt no EXEL vai cita veida datnes (Data Driven testing – datu vadīta testēšana). Datu plūsmu šai metodei iespējams aplūkot (5.att.). Tajā redzams, ka testa dati un sagaidāmais rezultāts tiek glabāti datu failā un tie tiek padoti skriptam, kurš tālāk ar tiem iedarbojas uz testējamo aplikāciju. Tāpat arī ar sagaidāmajiem rezultātiem, no aplikācijas sagaidāmie rezultāti tiek salīdzināti ar datu failā saglabātajiem. Šī metode nodrošina, ka

iespējams notestēt sistēmu ar dažādiem datiem un tos ir viegli pamainīt, jo nav jāiejaucas paša skripta kodā.



5.att. Datu vadītas testēšanas shēma (Avots: Guru99 mājas lapa, 2019)

Tīmekļa lapu testēšanas rīkam būtu jāspēj testēt tīmekļa vietnes ar vismaz trim populārākajā tīmekļa pārlūkprogrammām – Google Chrome, Mozilla Firefox un Microsoft Edge. Lai salīdzinātu rīkus, no sākuma tie tiks apskatīti katrs atsevišķi.

1. **JSystem**

JSystem ir atvērtā koda rīks, kurš nodrošina testu ierakstīšanu un atskaņošanu. Darbojas gan ar programmām, gan iegultām sistēmām. JSystem arhitektūra sastāv no četriem slāņiem, nodalot testēšanas pamatā esošo loģiku un grafisko lietotāja interfeisu. Atbalsta Java, Python un Visual Design programmēšanas valodas. Atbalsta Eclipse izstrādes vidi, bet nav iespējas veikt datu vadītu testēšanu. Autore neatrada arī informāciju par iespējamu tīmekļa pārlūkprogrammu testēšanu. Ir pieejamas apmācību iespējas un blogs (QA Testing Tools mājas lapa, 2019). Autore konstatē, ka šim rīkam nav savas mājas lapas, bet ir pieejams forums GitHub (vietne, kur vairāk kā 31 miljons cilvēku mācās, dalās un strādājot kopā veic programmatūru izstrādi (GitHub mājas lapa, 2019)). Apskatot rīka bloga aktivitāti, autore secina, ka rīks ir aktīvs un tiek izmantots, kā arī regulāri uzlabots – pēdējās versijas uzlabojumi bijuši 03.martā. 2019.gadā (autore lapu skatījās 14.aprīlī). Šajā GitHub vietne tiek arī atvērti problēmpieteikumi un attiecīgi arī atzīmēti vai tā ir kļūda vai tomēr konsultācija (iespējams lietotājs nav sapratis kā paveikt vēlamo darbību).

2. **SeleniumHQ**

Selenium ir atvērtā koda un bezmaksas programma, paredzēts tīmekļa vietnes funkcionalitātes pārbaudei. Atbalsta tādas skriptu rakstīšanas valodas kā Java, C#, Python un Ruby. Testu izstrādātājam jābūt programmēšanas zināšanām. Atbalsta gandrīz visas pārlūkprogrammas tādas, kā Microsoft Internet Explorer, Mozilla Firefox, Google Chrome, Opera, Safari un Android (QA Testing Tools mājas lapa, 2019). Selenium sastāv no četrām komponentēm:

- Selenium Integrated Development Environment (IDE) – tas ir pārlūkprogrammas spraudnis, kas nodrošina vienkāršu testu ierakstīšanu un atskaņošanu, ja nepieciešami kādi nopietnāki testi, tad būtu jāizmanto Selenium RC vai WebDriver;
- Selenium remote Control (RC) – Selenium galvenais rīks, kas nodrošina testu izstrādi dažādās programmēšanas valodās, tādās kā Java, C#, PHP, Python, Perl un Ruby;
- WebDriver – modernāka un stabilāka pieeja pārlūkprogrammas darbības automatizācijai, WebDriver atšķirībā no Selenium RC, nepaļaujies tika uz JavaScript automatizāciju, bet kontrolē pārlūkprogrammu, sazinoties ar to. Nodrošina testu izstrādi tādās pat programmēšanas valodās kā Selenium RC;
- Selenium Grid – rīks kuru, izmanto kopā ar Selenium RC, nodrošina paralēlās pārbaudes dažādās virtuālās mašīnās un pārlūkprogrammās. Respektīvi šis rīks nodrošina vairāku testu veikšanu uzreiz (Guru99 mājas lapa, 2019).

Selenium atbalsta sekojošas testēšanas metodes - akcepttestēšana, Agile (spējā) testēšana, Datu vadītā testēšana, Funkcionālā testēšana, Atslēgas vārdu vadītā testēšana, Regresa testēšana, Tradicionālā testēšana un Baltās kastes testēšana (QA testingtools mājas lapa, 2019). Izstrādes vižu atbalsts - Eclipse, IntelliJ un MS Visual Studio. Autorei apskatot Selenium mājas lapu, konstatē, ka ir pieejamas plašas apmācību un atbalsta iespējas, gan video, gan cita veida dokumentācija.

3. **Gauge**

Gauge savu produktu reklamē kā “Mazāk koda, mazāk uzturēšanas, vairāk akcepttestēšanas testu”. Tas ir atvērtā koda rīks, kurš atbalsta tādas programmēšanas valodas kā Java, Ruby, C#, Python un JavaScript, kā arī Golang (saukta arī par Go). Golang ir jauna programmēšanas valoda, kura popularitāti ieguva 2016.gada. Šo programmēšanas valodu izstrādāja un atbalsta Google. Gouge darbojas arī ar Selenium WebDriver. Izmantojot speciālus spraudņus, iespējams izmantot izstrādes vides Visual Studio Code, tad iespējamā skriptu rakstīšanas valoda būtu

JavaScript, Ruby vai Python. Vēl iespējams izmantot IntelliJ Idea izstrādes rīku un skriptus rakstīt Java programmēšanas valodā, kā arī Visual Studio, kurā skripti būtu jāraksta C# programmēšanas valodā (Gauge mājas lapa, 2019). Gauge mājas lapā ir pieejams Blogs un dokumentācija par šo rīku, bet sakarā ar to, ka rīks ir diezgan jauns ļoti daudz informācijas tīmeklī autorei neizdodas atrast.

4. **Watir**

Atvērtā koda risinājums, kas ir būvēts uz Ruby bibliotēku bāzes. Watir darbojas ar pārlūkprogrammu tāpat kā reāls lietotājs, spiežot uz saitēm, aizpildot ievadlaukus un apstiprinot ievadītos datus, nospiežot pogas. Veic arī sagaidāmā paziņojumu pārbaudi. Atbalsta visas populārākās pārlūkprogrammas (Watir mājas lapa, 2019). Skriptu rakstīšanas valoda – Ruby. Skriptus iespējams izstrādāt ar izstrādes vides rīku Netbeans (QA Testing tools, 2019). Rīka mājas lapā autore atrod diezgan daudz mācību materiālus, pieejams arī Blogs, kurā šie rīka lietotāji uzdod dažādus jautājumus un saņem atbildes.

5. **FitNesse**

Atvērtā koda risinājums, kurš paredzēts akcepttestēšanas veikšanai. Rīks nodrošina akcepttestēšanas kritēriju definēšanu un pārbaudi. Ar šā rīka starpniecību iespējams nodrošināt dažādu ieinteresēto personu iesaisti, programmatūras piegādes procesā. Wiki serveris nodrošina programmatūras dokumentu saglabāšanu. Ļauj pārbaudīt vai dokumentācija atbilst programmatūrai, nodrošinot dokumentācijas atjaunošanu Atbalsta tāds programmēšanas valodas kā Java, Python un C#. Testēšanu iespējams veikt lietotājiem bez programmēšanas zināšanām (FitNesse mājas lapa, 2019). Rīks atbalsta Datu vadītu testēšanu, un ir pieejama sava izstrādes vide (QA Testing Tools mājas lapa, 2019).

6. **JMeter**

Atvērtā koda rīks, kura pirmā versija tika radīta 1998.gadā. Izstrādāts Java programmēšanas valodā. Rīks ir paredzēts slodzes un veiktspējas testiem. Atbalsta aplikācijas/serverus/protokolus, piemērām: Web – HTTP, HTTPS (Java, NodeJS, PHP, ASP vai .NET), SOAP / REST Webservices, FTP, LPDAP. Rīks darbojas protokolu līmenī un neiedarbojas uz pārlūkprogrammu pa tiešo (Jmeter mājas lapa, 2019).

7. **LDTP**

Rīks paredzēts Linux darba virsmas aplikāciju un web aplikāciju testēšanai, izmanto pieejamās bibliotēkas, lai aktivizētu lietojumprogrammu lietotāja interfeisa logrīkus. Ir iespēja ierakstīt lietotāja darbības, testēšanas piemēriem. Atbalsta Python, Java, Ruby, Perl, C#, VB.Net, PowerShell programmēšanas valodas (LDTP mājas lapa, 2019). Pētot informāciju tīmeklī, autore secina, ka rīks nav populārs un netiek izlaistas jaunas rīka versijas. Rīks neatbalsta datu vadītu testēšanu, bet tam ir sava izstrādes vide, rīka mājas lapā atrodama arī dokumentācija un apraksti.

8. **Karma**

Karma ir JavaScript bāzēts testēšanas rīks, kurš darbojas ar Node.js. Tas ir piemērots AngularJS vai citiem JavaScript projektiem. Iespējams veikt testus uz dažādām ierīcēm (dators, planšetdators vai viedtālrunis). Rīks ir integrēts ar Jenkins, Travis un Semaphore pakotnēm, kā arī ir pieejams daudz spraudņi. Skriptu rakstīšanas valoda – JavaScript. Neatbalsta datu vadītu testēšanu, ir sava Integrētā izstrādes vide (QA testingtools mājas lapa, 2019). Autorei apskatot rīka mājas lapu, secina, ka ir pieejami mācību materiāli, bet to nav pārāk daudz.

9. **Jelly**

Rīks, kurs XML kodu pārvērš par izpildkodu. Rīks ir Java un XML bāzēts rīks skriptu rakstīšanai un izpildei. Skriptus iespējams rakstīt Java un XML programmēšanas valodās (QA TestingTools mājas lapa, 2019). Autorei papētot informāciju rīka Jelly mājas lapā (Apache Comons Jelly, 2019) atrod ierakstu, kas vēsta, ka ir zema aktivitāte no šī rīka izstrādātājiem.

10. **Cerberus**

Atvērtā koda testu repozitorijs, ar lietotājam draudzīgu interfeisu un dažādām tehnoloģijām. Rīka galvenais mērķis ir ļaut jebkuram lietotājam izstrādāt un redzēt savus testus, kā arī veikt to palaišanu (Cerberus mājas lapa, 2019). Salīdzinājumā jauns rīks – radīts 2016.gadā. Piedāvā savu izstrādes vidi, rīks arī pilda testu pārvaldības funkcijas. Interesanti, tas ka šajā rīkā iespējams pievienot Selenium, Appium un Sikuli testēšanas rīku spraudņus (Cerberus testing mājas lapa, 2019). Autore izpētot Cerberus demo testēšanas rīku (Demo Cerberus testing mājas lapa, 2019), secina, ka tas izskatās interesants, taču jāiegulda diezgan daudz laika, lai saprastu, ka tas viss darbojas. Testēšanas piemēri tiek sagatavoti tos nerakstot, bet gan izvēloties no

izkrītošajiem sarakstiem un ievadlaukos definējot ievadvērtības. Autore neatrada iespējas ievaddatus ielādēt no .xlsx vai citu datu faila.

11. **HtmlUnit**

HtmlUnit ir testēšanas rīks, kurš sevi pozicionē kā "mazāk lietotāja saskarnes interfeisa pārlūkos priekš Java programmā" (QA TestingTools mājas lapa, 2019). Izmantojot lietojumprogrammas saskarni (API), rīks neiedarbojoties pa tiešo uz pārlūkprogrammu, modelē HTML dokumentus un veic formu aizpildīšanu, saišu nospiešanu un citas darbības, it kā darbotos ar reālu pārlūkprogrammu. Rīkam ir JavaScript atbalsts un tas darbojas ar AJAX bibliotēkām (QA testing Tools mājas lapa, 2019). Rīka mājas lapā un arī QA Testing tools mājas lapā nav atrodama informācija par izstrādes vidēm.

12. **WCAT**

WCAT – WEB Capacity Analysis Tool (kapacitātes analīzes rīks) radīts Microsoft. Paredzēts tīmekļa aplikāciju veiktspējas testēšanai. Spēj noteikt procesoru noslodzi, atmiņas lietojumu, nosūtīto un saņemto baitu skaitu, u.c. (QA Testing Tools mājas lapa, 2019).

13. **Webrat**

Webrat ir Ruby akcepttestēšanas rīks tīmekļa aplikācijām. Skriptu rakstīšanas valoda – Ruby (QA Testing Tools mājas lapa, 2019). Sakarā ar to, ka jau redzams, ka šis rīks nebūs piemērots (atbalsta tikai Ruby programmēšanas valodā rakstītās aplikācijas), autore sīkāku izpēti neveic.

14. **SeLite**

SeLite (Selenium un SQLLite apvienojums), paredzēts automatizēt tīmekļa lietojumprogrammas datu bāzu līmenī. Tas kalpo tīmekļa lietojumprogrammu testēšanai un citiem mērķiem, piemēram administrēšanai vai datu ieguves/datu manipulāciju veikšanai. Skriptu rakstīšanas valoda – JavaScript. Atbalsta Mozzila FireFox pārlūkprogrammu, citas pārlūkprogrammas neatbalsta. Ir sava integrētā izstrādes vide (QA Testing Tools mājas lapa, 2019).

15. **Minq**

Minq ir atvērtā koda testēšanas rīks, kurš paredzēts tīmekļa lietojumprogrammu testēšanai, kuras rakstītas PHP valodā. Rīka PHP bibliotēka nodrošina tīmekļa lietojumprogrammas

sadarbību ar pārlūkprogrammu. Skriptu rakstīšanas valoda – PHP (QA Testing Tools mājas lapa, 2019).

16. **Robot Framework**

Paredzēts tīmekļa lietojumprogrammu akcepttestēšanas, datu vadītas testēšanas, funkcionālās testēšanas un atslēgas vārdu vadītas testēšanas veikšanai. Skriptu rakstīšanas valodas – lokālā skriptu rakstīšanas valoda, Python (QA Testing Tools mājas lapa, 2019). Pieejami spraudņi priekš Eclipse un IntelliJ izstrādes vides. Pieejami arī mācību materiāli un dokumentācija (Robot Framework mājas lapa, 2019).

17. **Galen Framework**

Rīks paredzēts tīmekļa lapas izkārtojuma testēšanai, imitējot dažādu ierīču izmērus. Rīks nodrošina arī Funkcionālos testus, Vienībtestus. Skriptu rakstīšanas valodas – Java, JavaScript. Atbalsta arī Google Chrome, Mozilla FireFox un Microsoft Edge (QA Testing Tools mājas lapa, 2019). Atbalsta IntelliJ, Sublime, Vim un VS Code izstrādes vides (Galen Framework mājas lapa, 2019).

Pamatojoties uz augstāk minēto rīku apskatu, autore sagatavoja tabulu ar rīkiem, iekļaujot tos rīkus, kuri pa tiešo spēj iedarboties, testēt tīmekļa pārlūkprogrammas, lai būtu vieglāk izvēlēties piemērotāko rīku ar kuru veikt automatizācijas skriptu izstrādi. Tika definēti sekojoši kritēriji:

1. Testēšanas metode – rīkam jāatbalsta funkcionālā, akcepttestēšana, kā arī galvenā metode – datu vadīta testēšana;
2. Izstrādes vide (IDE) – vēlama ir Eclipse izstrādes vide, bet, ja rīks nodrošina savu izstrādes vidi, šis var nebūt ļoti noteicošs kritērijs.
3. Operētājsistēma (OS) – rīkam jāatbalsta Windows operētājsistēma vai jābūt neatkarīgam no operētājsistēmas.
4. Skriptu rakstīšanas valoda = autorei ir pieredze ar Java programmēšanas valodu, tāpēc šis kritērijs ir viens no noteicošajiem;
5. Pārlūkprogramma – vai rīks spēj iedarboties uz pārlūkprogrammām un atbalsta trīs populārākās no tām;
6. Mācību un atbalsta iespējas – rīkam jābūt pieejamiem materiāliem un mācību iespējām, jo uzsākot darbu ar jaunu rīku, iespējams saskarties ar kādām grūtībām un ir ļoti būtiski saņemt atbalstu.

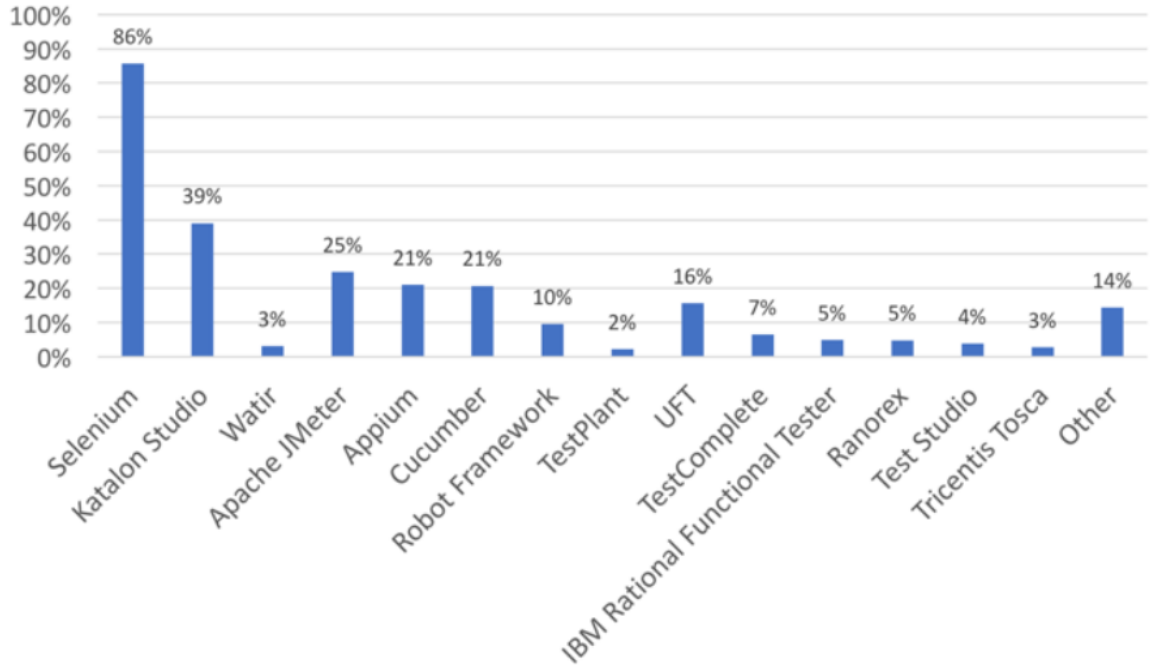
1. tabula. Testēšanas rīku salīdzinājums (Avots: autores veidots)

Testēšanas rīku salīdzinājums						
Kritērijs		SeleniumHQ	Watir	Karma	SeLite	Galen Framework
Testēšanas metodes	Funkcionālā testēšana	X	X	X	X	X
	Akcepttestēšana	X				
	Datu vadītā testēšana	X				
IDE	Eclipse	X				
	IntelliJ					X
OS	Windows	X	X			
	Neatkarīgs no OS	X		X	X	X
	Mac OS	X	X			
Skriptu rakstīšanas valoda	Java	X				X
	Python					
	JavaScript			X	X	X
	Ruby		X			
	Lokāla skriptu valoda			X	X	
Pārlūkprogrammas	Chrome	X	X	X		X
	FireFox	X	X	X	X	X
	MS Edge	X	X	X		X
Mācību un atbalsta iespējas	Apmācību materiāli (Tutorial)	X		X		X
	Dokumentācija mājas lapā	X	X	X	X	X
	FAQ	X	X	X	X	X
	Forums	X	X		X	X
	Video apmācības	X	X			
	Web bināri	X				

Lai būtu pārskatāmāka informācija par salīdzināmo testēšanas rīku piedāvātajām iespējām, pie kritērijiem tika iekļauto šo rīku pieejamās skriptu rakstīšanas valodas un cita informācija. Kā redzams (1.tabula) vienīgais rīks, kurš atbilsta datu vadītu testēšanu, kas ir viens no galvenajiem kritērijiem, ir Selenium. Kā arī Selenium atbilst pārējiem izvirzītajiem testēšanas rīka kritērijiem. Tāpēc, kā piemērotākais testēšanas rīks tiek izvēlēts Selenium.

Pētot informāciju par testēšanas rīkiem autore atrada testēšanas rīku pētījuma salīdzinājumu, kuru veica testēšanas rīka Katalon Studio organizācija. Katalon Studio ir komerciāls rīks tāpēc

šajā rīku pētījumā netika iekļauts. Pētījuma nosaukums "The most striking problems in test automation" (tulkojums – Galvenās problēmas testu automatizācijā). Pētījums izdots 2018. gada maijā. Pētījumā tikuši aptaujāti 2291 respondenti no dažādām organizācijām. Šajā pētījumā secināts, ka populārākais izmantotais testēšanas rīks ir Selenium, pārējo testēšanas rīku popularitāti iespējams aplūkot (6.att.).



6. att. Izmantotie testēšanas rīki

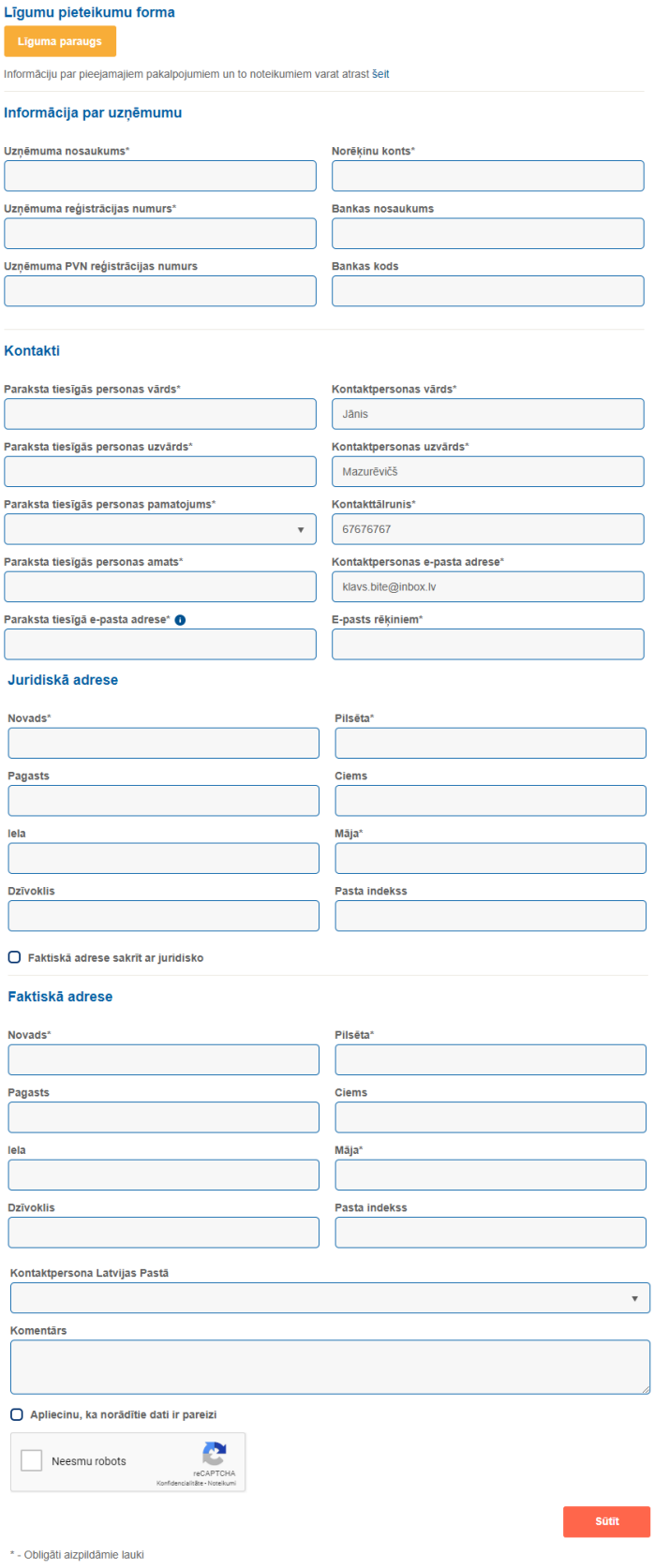
(Avots: The most striking problems in test automation: A survey, 5.lpp, 2018)

Vēl šī pētījuma galvenie atzinumi ir, ka atvērtā koda un bezmaksas rīki ir dominējošie testēšanas automatizācijā. Testēšanas automatizācijas galvenie mērķi ir automatizēt funkcionālos un regresa testus. Būtiskākās automatizācijas problēmas, kuras atklātas šajā pētījumā ir pārāk bieža prasību mainība un resursu trūkums automatizācijai (The most striking problems in test automation: A survey, 2018).

3. PĒTĪJUMA REZULTĀTI

3.1. Izstrādes vides sagatavošana

Pamatojoties uz augstāk minēto rīku pētījumu, Selenium tika izvēlēts kā piemērotākais testēšanas rīks. Būs nepieciešams sagatavot izstrādes vidi un uzsākt skriptu izstrādi, bet sākotnēji nepieciešams noteikt, kas būs tās funkcijas, kuras tiks automatizētas pirmās. Mans Pasts ir diezgan apjomīga vietne ar daudz un dažādām funkcijām, bet sakarā ar to, ka Latvijas Pasts veic līgumu pārslēgšanu ar esošiem un jauniem klientiem, izmantojot Manss Pasts funkciju "Līguma pieteikums" - pirmā potenciālā automatizējamā testu kopa ir – Līguma pieteikuma forma. Šajā formā potenciālie vai esošie klienti var aizpildīt formu un nosūtīt pieteikumu Līguma noslēgšanai. Līgumi tiek slēgti par pakalpojumu saņemšanu no Latvijas Pasta, tādu kā "Pastnieks birojā" – kur pastnieks ierodas pēc sūtījumiem, kurus klients vēlas nosūtīt. Pakalpojums "Adresētais tiešais pasts", kur iespējams nosūtīt reklāmas, kādai konkrētai mērķauditorijai – piemēram tikai sievietēm, no 30 līdz 40 gadiem, kuras interesē skaistuma industrija un kuras dzīvo Rīgā. Vēl Latvijas Pasts piedāvā īpašus tarifus klientiem, kuri vēlās sūtīt sūtījumus vairumā, piemēram skaitā virs 1000 sūtījumiem. Līgums nodrošina arī to, ka Latvijas Pasts piestāda rēķinu pēc pakalpojumu sniegšanas, tātad ar pēcapmaksu, parasti rēķinus izraksta mēneša sākumā, par iepriekšējā mēnesī sniegtajiem pakalpojumiem. Pēc līguma pieteikuma reģistrēšanas Mans Pasts, sistēma ģenerē e-pastu atbildīgajam pakalpojuma vadītājam un tas savukārt sazinās ar lietotāju un noslēdz līgumu. Pirms līguma pieteikuma reģistrēšanas, lietotājs var lejupielādēt un apskatīt līguma paraugu, līguma pieteikuma formu var aizpildīt tikai vietnē Mans Pasts reģistrēts lietotājs. Testēšanas mērķis – pārbaudīt vai sistēma neļauj reģistrēt Līguma pieteikumu, neaizpildot obligātos laukus, par kuru neaizpildīšanu, lietotājam izvadot atbilstošu paziņojumu. Daļa no formas laukiem, tiks aizpildīta automātiski – datus ņemot no lietotāja profila informācijas. Līguma pieteikuma formu, iespējams aplūkot (7.att.). Ja līgumu piesaka lietotājs, kas ir fiziska persona, tad sadaļas "Informācija par uzņēmumu" lauki būs tukši un lietotājam tie būs jāaizpilda. Savukārt, ja juridiskai personai Mans Pasta profilā būs aizpildīti lauki, Līguma pieteikuma formā tie jau būs aizpildīti.



7. att. Līguma pieteikuma forma (Avots: ekrānattēls no Manspasts)

Lai labāk saprastu, kādi ievadlauki ir obligātie un kāds būs kļūdas paziņojums, ja šie lauki nebūs aizpildīti, autore sagatavoja tabulu, kurā ir minēti visi nepieciešamie parametri (2.tabula).

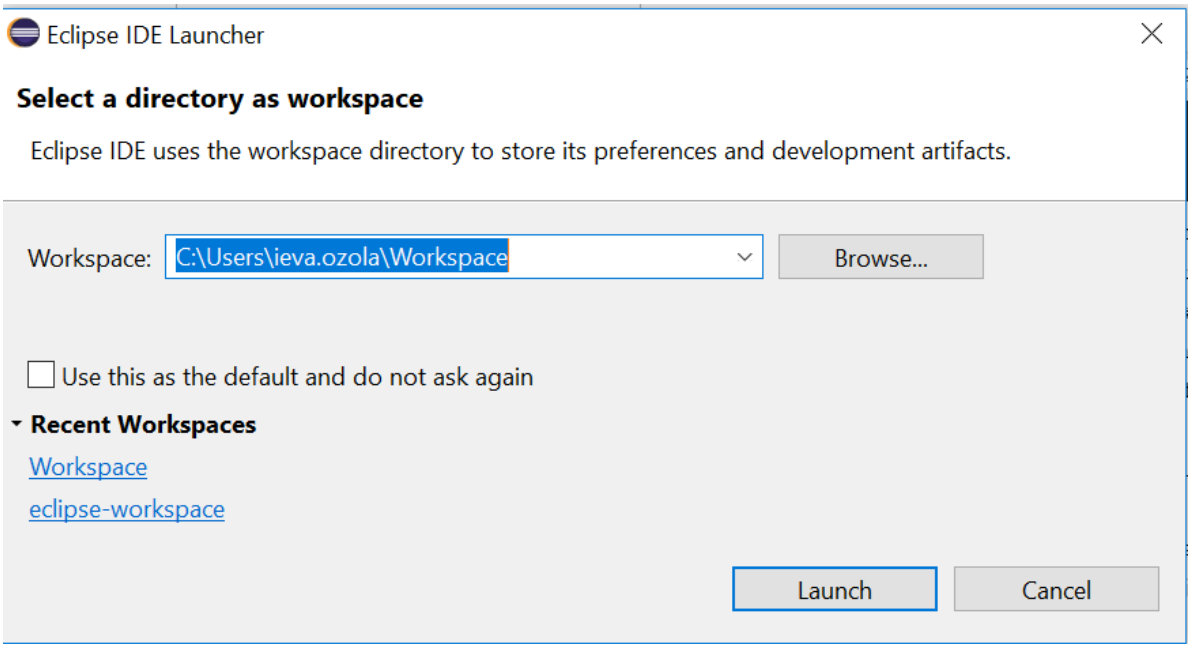
2. tabula. Formas Līguma pieteikums lauki (Avots: autora apkopojums)

Nr.p.k.	Lauks	Obligāts	Komentārs	Paziņojuma teksts, ja lauks nav aizpildīts
1.	Uzņēmuma nosaukums	Jā	Ja lietotājs ir JP, informācija tiek aizpildīta no profila datiem	Ievadiet uzņēmuma nosaukumu
2.	Uzņēmuma reģistrācijas numurs	Jā	Ja lietotājs ir JP, informācija tiek aizpildīta no profila datiem	Ievadiet uzņēmuma reģ. numuru
3.	Uzņēmuma PVN reģistrācijas numurs	Nē	Ja lietotājs ir JP, informācija tiek aizpildīta no profila datiem	
4.	Norēķina konts	Jā	Ja lietotājs ir JP, informācija tiek aizpildīta no profila datiem	Ievadiet norēķinu kontu
5.	Bankas nosaukums	Jā	Aizpildās automātiski no norēķinu konta	Ievadiet bankas nosaukumu
6.	Bankas kods	Jā	Aizpildās automātiski no norēķinu konta	Ievadiet bankas swift kodu
7.	Paraksta tiesīgās personas vārds	Jā		Ievadiet paraksttiesīgās personas vārdu
8.	Paraksta tiesīgās personas uzvārds	Jā		Ievadiet paraksttiesīgās personas uzvārdu
9.	Paraksta tiesīgās personas pamatojums	Jā		Ievadiet paraksttiesīgās personas pamatojumu
10.	Paraksta tiesīgās personas amats	Jā		Ievadiet paraksttiesīgās personas amatu
11.	Paraksta tiesīgā e-pasta adrese	Jā		Ievadiet paraksttiesīgās personas e-pastu
12.	Kontaktpersonas vārds	Jā	Aizpildās automātiski no profila informācijas	Ievadiet kontaktpersonas vārdu
13.	Kontaktpersonas uzvārds	Jā	Aizpildās automātiski no profila informācijas	Ievadiet kontaktpersonas uzvārdu
14.	Kontakttālrunis	Jā	Aizpildās automātiski no profila informācijas	Ievadiet kontaktpersonas tālruni
15.	Kontaktpersonas e-pasta adrese	Jā	Aizpildās automātiski no profila informācijas	Ievadiet kontaktpersonas e-pastu
16.	E-pasts rēķiniem	Jā		Ievadiet e-pastu rēķiniem
17.	Juridiskā adrese	Jā	Aizpildās automātiski no profila informācijas	Lūdzu, norādiet novadu vai pilsētu
18.	Pazīme "Faktiskā adrese sakrīt ar juridisko"	Nē		
19.	Faktiskā adrese	Jā		Lūdzu, ievadiet precīzu pak. Saņemšanas adresi
20.	Kontaktpersona Latvijas Pastā	Nē		
21.	Komentārs	Nē		
22.	Pazīme "Apliecinu, ka norādītie dati ir pareizi"	Jā		Apstipriniet, ka norādītie dati ir korekti
23.	Pazīme "Neesmu robots"	Jā		Neveiksmīgs cilvēktests

Lai veiktu skriptu izstrādi nepieciešams konfigurēt programmatūras izstrādes vidi Eclipse. Šī izstrādes vide sastāv no integrētās izstrādes vides (IDE) un paplašināmas spraudņu sistēmas. Ar Eclipse iespējams izstrādāt programmatūras JAVA un izmantojot dažādus spraudņus arī citās programmēšanas valodās, tādās kā C, C++, COBOLL, Perl, PHP, Python, Ruby, u.c. (Eclipse mājas lapa, 2019).
Resursi programmatūras licencei nav nepieciešami, jo tā ir atvērtā koda programmatūra. Pēdējā stabilā versija ir SimRel 2018-9 (8.att.).

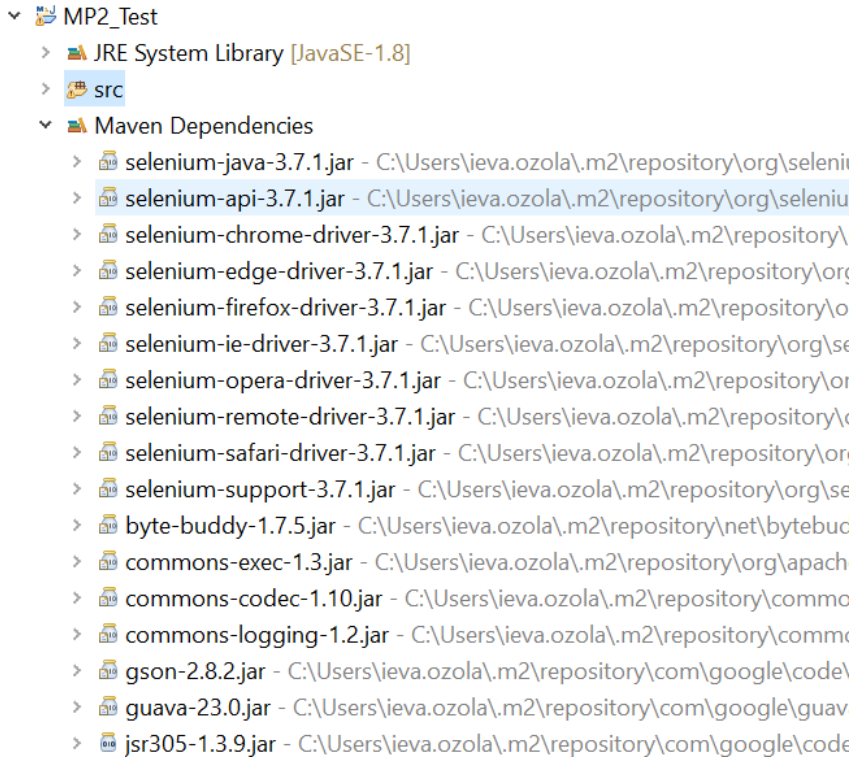
8. att. Eclipse sāknēšanās logs (Avots: autores veidot ekrānattēls)

Tā paredzēta instalēšanai uz datoriem ar dažādām operētājsistēmām (turpmāk – OS) gan uz Linux, gan Mac OS X, Solaris un Windows. Šinī gadījumā programmatūra tika instalēta uz datora ar Windows v.10 Pro operētājsistēmu. Eclipse instalācijas pakotne tika lejupielādēta no https://www.eclipse.org/downloads/ un atbilstoši instalācijas soļiem instalēta. Sākotnēji bija nepieciešams norādīt direktoriju (9.att.), kurā glabāsies visas ECLIPSE projekta datnes.



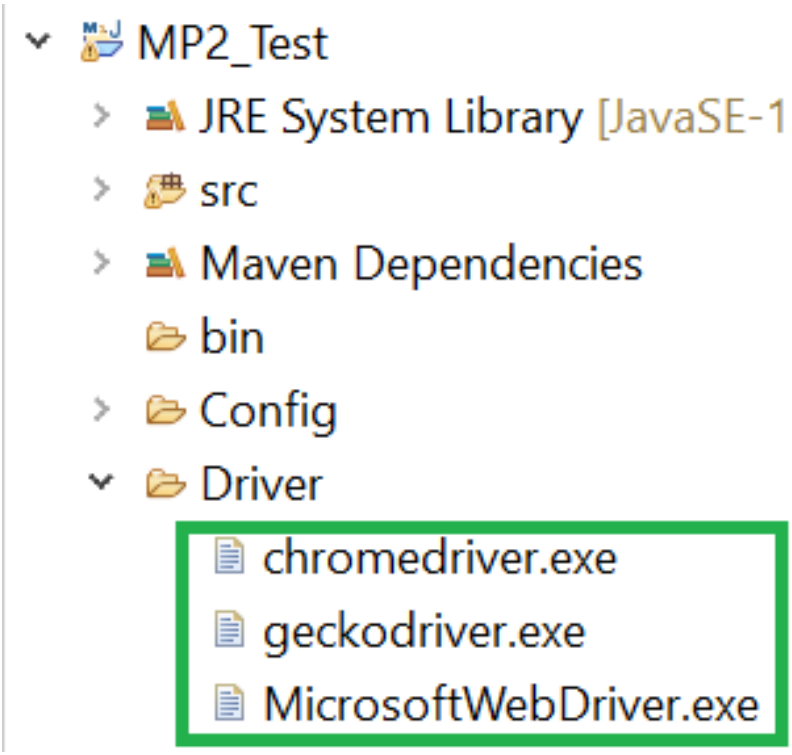
9.att. Eclipse projekta direktorijas norādīšana (Avots: autores veidots ekrānattēls)

Lai nodrošinātu SELENIUM, TESTNG un MAVEN iespēju pielietošanu skriptu izstrādē, tika instalēti arī šo programmproduktu .jar spraudņi un bibliotēkas un attiecīgi pievienoti projektam (10.att.).



10.att. Pievienotie papildus paplašinājumi projektam (Avots: autores veidots ekrānattēls)

Pārlūkprogrammu Google Chrome, Mozilla FireFox un MS Edge darbināšanai arī bija nepieciešams lejupielādēt un instalēt atbilstošos draiverus (11.att.).

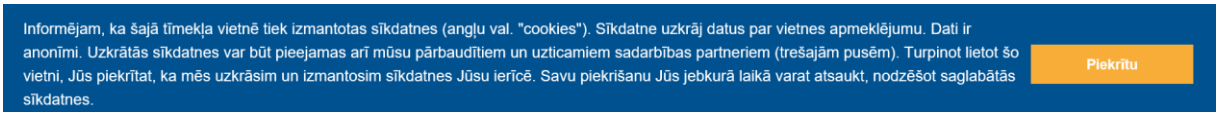


11. att. Pārlūkprogrammu draiveri (Avots: autores veids ekrānattēls)

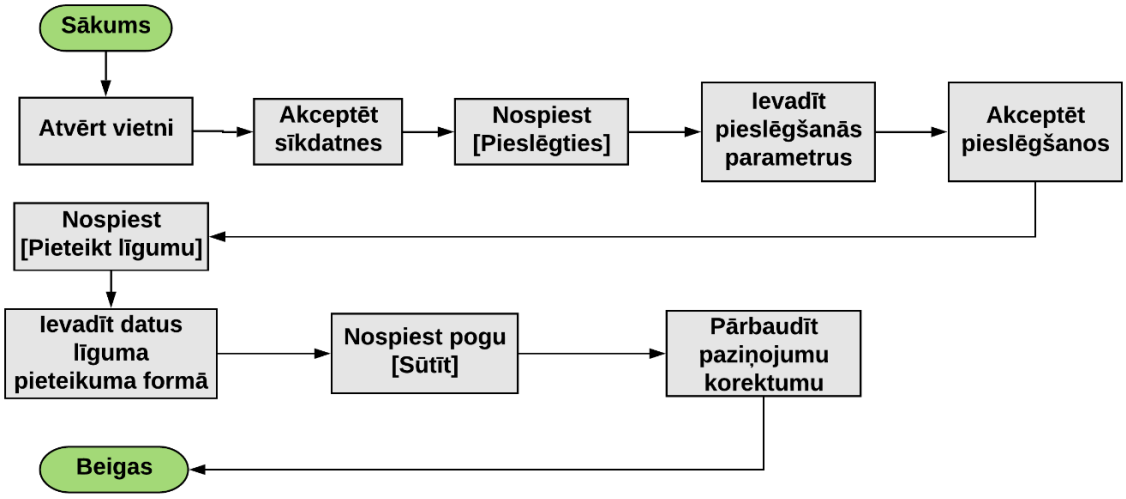
Šie draiveri darbina lietojumprogrammu un imitējot lietotāju darbības, izmantojot testējamās lapas HTML kodu veic darbības, kuras minētas testēšanas skriptā.

3.2. Automatizētu skriptu izstrāde

SELENIUM testēšanas rīks (Seleniumhq.org mājas lapa, 2019) darbojas ar HTML un CSS elementiem, respektīvi, lai nospiestu pogu, vai ierakstītu kādu vērtību ievadlaukā, SELENIUM jāzina kā šis lauks saucas un to atrod vai nu pēc HTML definētā identifikatora, klases, vai Xpath (elementa atrašanās vieta) vai kāda cita unikāla identifikatora. Lai sāktu rakstīt skriptu, no sākuma jādefinē vai jāizmanto jau esošais testēšanas scenārijs. Skripta veicamās darbības redzamas (13.att). Sākumā jāatver testējamā vietne http://mp2test.manspasts.lv/, tad parādīsies sīkdatņu akceptēšanas logs (12.att.), kurā jānospiež poga [Piekrītu].

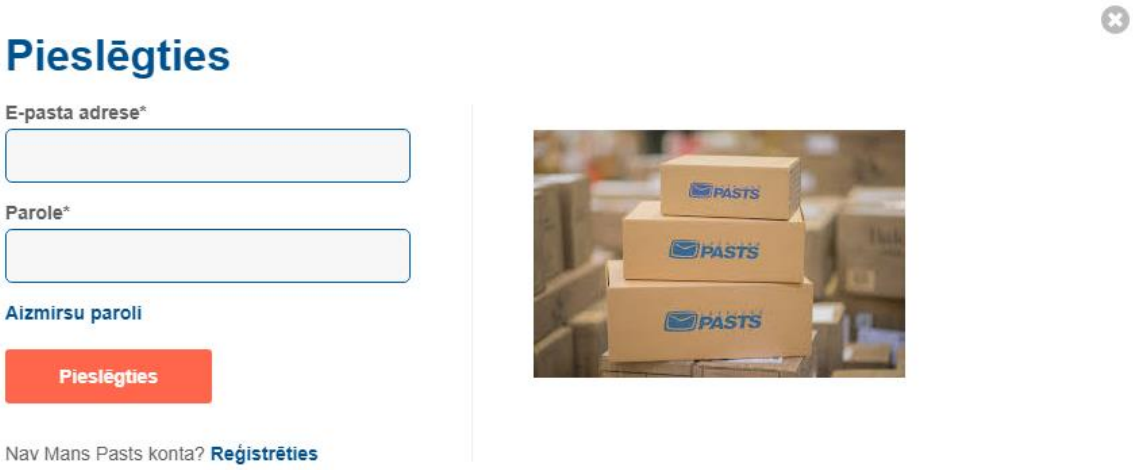


12. att. Sīkdatņu apstiprināšanas logs (Avots: ekrānattēls no testējamās vietnes)



13.att. Skripta veicamās darbības (Avots: autores veidots)

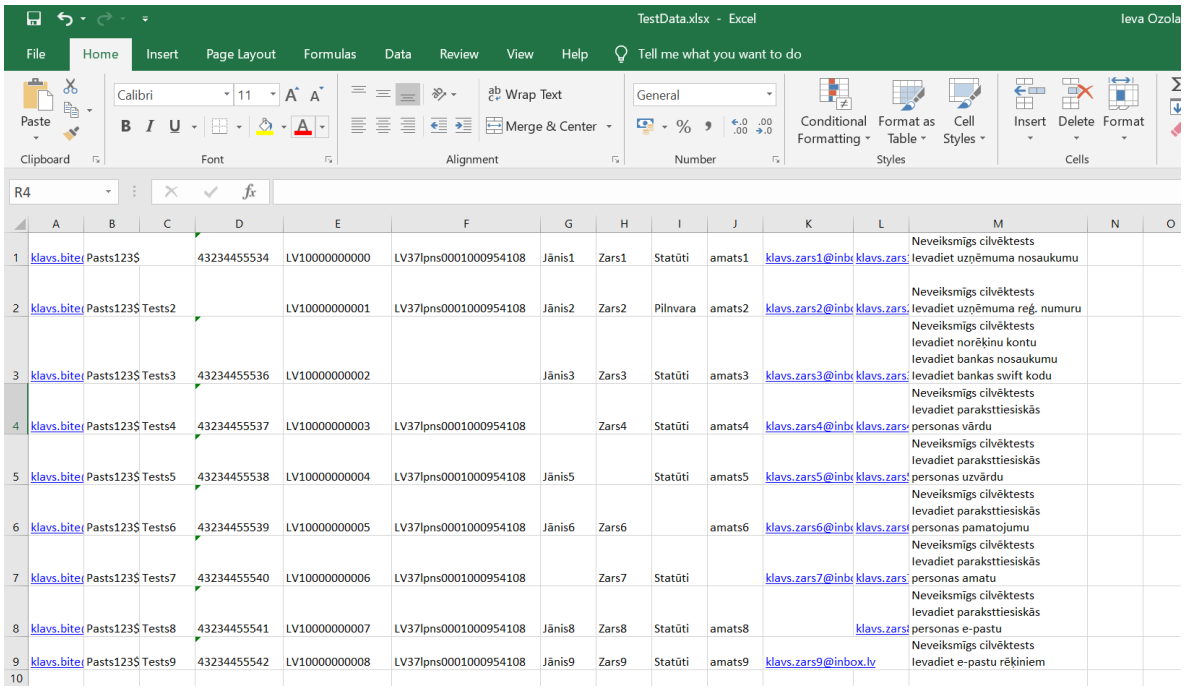
Nākamajā solī nepieciešams nospiest pogu [Pieslēgties] un atvērsies Pieslēgšanās pop-up logs (14.att.), kurā jāievada lietotāja e-pasts un parole, kā arī nospiest pogu [Pieslēgties].



14.att. Pieslēgšanās logs (Avots: ekrānattēls no testējamās vietnes)

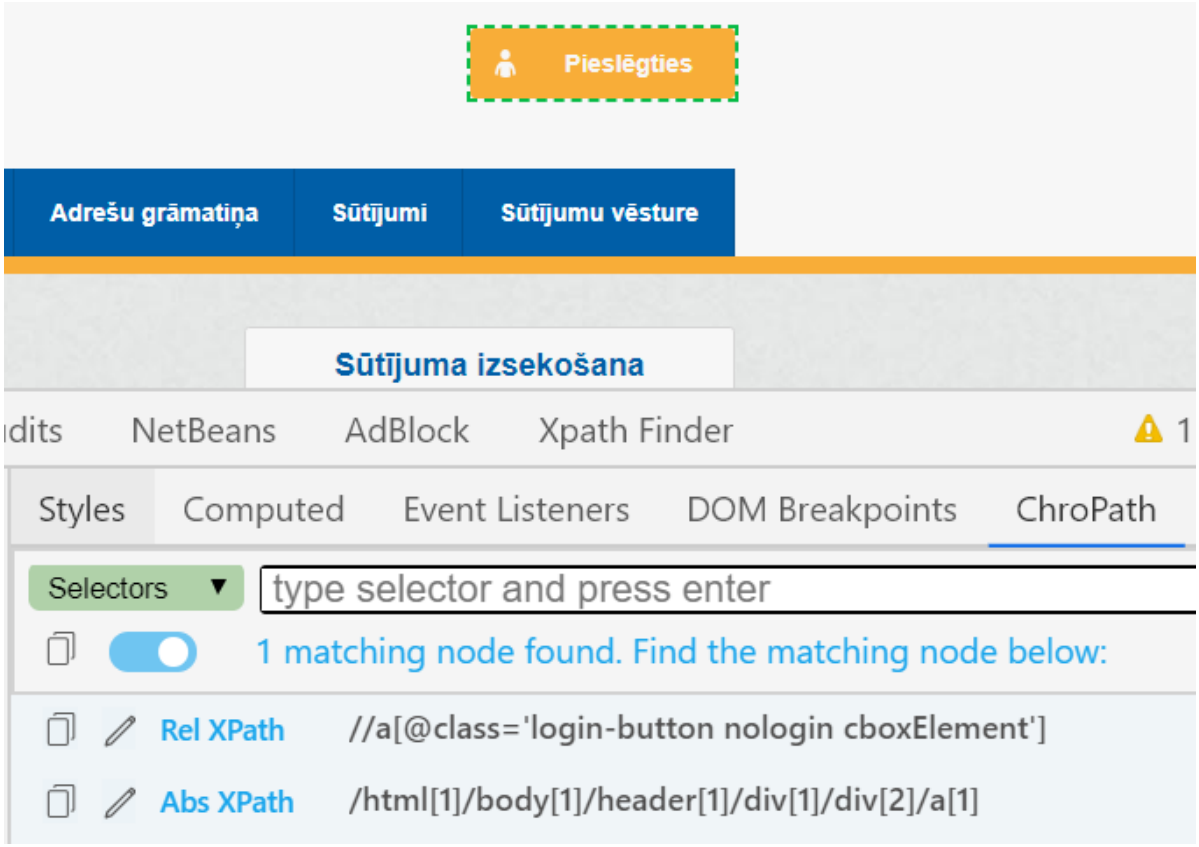
Pēc šiem soļiem, būs iespējams nospiest pogu [Līguma pieteikums] un atvērsies līguma aizpildīšanas forma, kurā aizpildot datus un nospiežot pogu [Sūtīt], būs nepieciešams pārbaudīt sistēmas paziņojumus. Ņemot vērā augstāk minētās formas parametrus un paziņojumus, tika

sagatavoti testpiemēri ar ievadāmajiem datiem .xlsx datnē (15.att.). Šī datne arī tiks izmantota testēšanas skripta ietvaros.



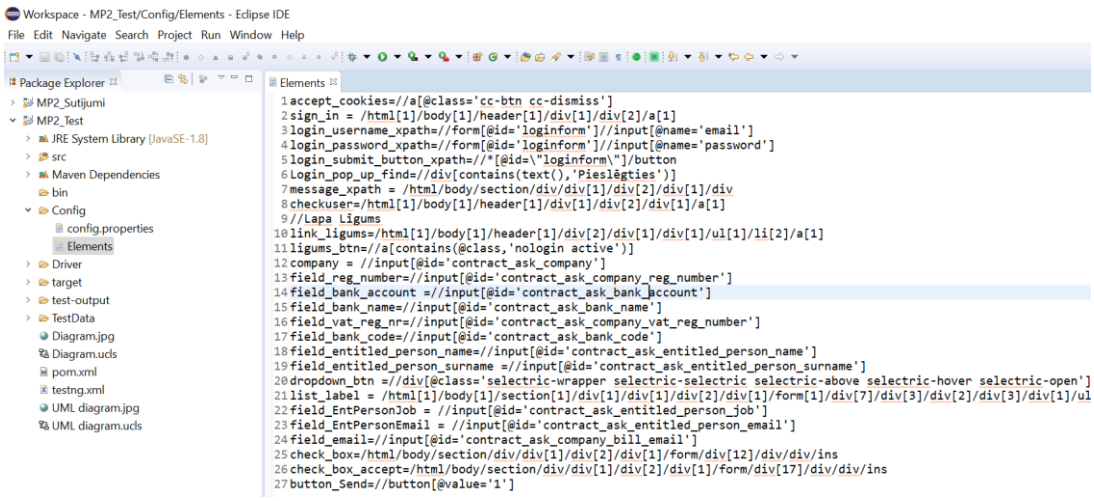
15.att. Ievaddati Līguma pieteikuma formai (Avots: autores veidots)

Lai sāktu rakstīt skriptu ir nepieciešams sagatavot/atrast izmantojamo web elementu identifikatorus. Šeit autore saskarās ar grūtībām, jo pēc element id, vai nosaukuma sākotnēji Selenium skripts nevarēja atrast un tāpēc autore meklēja informāciju tīmeklī un atrada, ka var izmantot web elementu XPath (Guru 99 mājas lapa, 2019). XPath ir veids kā atrast elementa novietojumu web lapā. Pastāv divu veidu XPath adreses absolūtā un relatīvā. Absolūtā Xpath adrese sākās ar vienu "/" un definē adresi sākot ar galveno mezglu. Gadījumā, ja elements tiek pārvietots, to šo adresi vairs nevarēs izmantot, jo elementa adrese būs jau manījusies. Savādāk būs gadījumā, kad tiks izmantota relatīvā XPath, kuras adrese sākas ar "//", kas nozīmē, ka elements tiek meklēts jau par visu tīmekļa lapu un nav nepieciešams rakstīt garo XPath adresi, abu adrešu salīdzinājumu var apskatīt (16.att.), attēlā redzama pogas [Pieslēgties] XPath adrese.



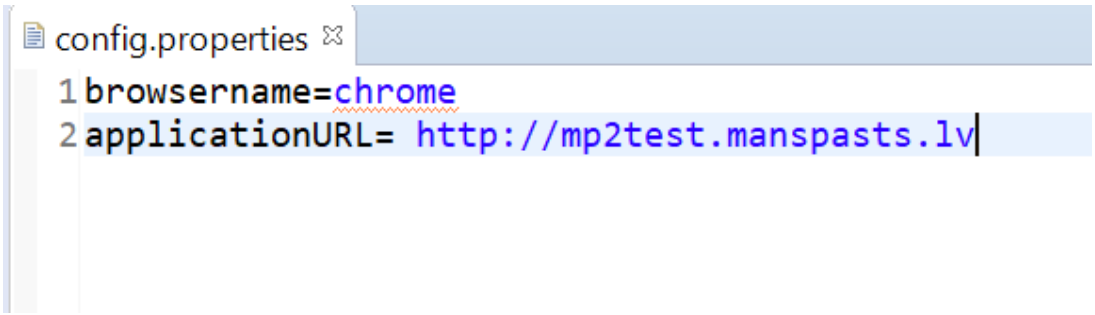
16.att. Pogas [Pieslēgties] XPath adreses (Avots: ekrānattēls no testējamas tīmekļa vietnes)

Lai vieglāk atrastu XPath adrese autore meklēja rīkus/tīmekļa paplašinājumus, kuru sniegtās iespējas nodrošinātu ātrāku XPath adrešu noskaidrošanu. Pēc nelielas izpētes tīmeklī, autore atrada pārlūkprogrammas paplašinājumu ChroPath. Šis paplašinājums ir paredzēts Google Chrome un Mozilla Firefox pārlūkprogrammām (ChroPath mājas lapa, 2019). Visu nepieciešamo elementu XPath ceļi tika saglabāti datnē "Elements" (17.att.)



17. att. Datnes "Elements" saturs un novietojums projektā (Avots: autores veidots)

Kā redzams (17.att.) datne Elements tika izvietota mapē “Config”. Šajā pašā mapē arī tika izveidota un saglabāta datne config.properties (18.att.). Šajā datnē iespējams definēt kāda ir testējamās vietnes adresi un pārlūks ar kuru tiek testēta šī lapa.



18.att. Datnes config.properties saturs (Avots: autores veidots)

Lai šīs abas datnes (Elements un config.properties) varētu izmanot tālākā kodā, tika izveidota klase utility, kurā tika definēti Elements un config.properties izmantošanas parametri (19.att.).

Driver.java | utility.java

```
1  package org.util;
2
3  import java.io.FileInputStream;
6
7  public class utility {
8
9      public static Object fetchPropertyValue(String key) throws IOException
10
11     {
12         FileInputStream file = new FileInputStream ("./Config/config.properties");
13         Properties property = new Properties();
14         property.load(file);
15         return property.get(key);
16
17
18     }
19
20     public static String fetchLocatorValue(String key) throws IOException
21
22     {
23         FileInputStream file = new FileInputStream ("./Config/Elements");
24         Properties property = new Properties();
25         property.load(file);
26         return property.get(key).toString();
27
28
29     }
30
31 }
32
```

19. att. Klases Utiliy saturs (Avots: autores veidots)

Lai varētu izmantot pārlūkprogrammu draiverus, autore izveidoja klasi Driver, kurā tika definēti Google Chrome, Firefox un MS Edge parametri (20.att.). Šajā klasē arī tiek definēta pārlūkprogrammas aizvēršanās pēc testēšanas skripta izpildes.

```
public class Driver
{

    public WebDriver driver;
    @BeforeMethod
    public void initiateDriver() throws Exception
    {
        if (utility.fetchPropertyValue("browsername").toString().equalsIgnoreCase("chrome"))
        {
            System.setProperty("webdriver.chrome.driver", "C:\\Selenium_Drivers\\chromedriver.exe");
            driver = new ChromeDriver();
        }
        else if (utility.fetchPropertyValue("browsername").toString().equalsIgnoreCase("firefox"))
        {
            System.setProperty("webdriver.gecko.driver", "C:\\Selenium_Drivers\\geckodriver.exe");
            driver = new FirefoxDriver();
        }

        else if(utility.fetchPropertyValue("browsername").toString().equalsIgnoreCase("Edge"))
            {
            System.setProperty("webdriver.edge.driver", "C:\\Selenium_Drivers\\MicrosoftWebDriver.exe");
            driver=new EdgeDriver();
            }
        else
            {
            System.setProperty("webdriver.chrome.driver", "C:\\Selenium_Drivers\\chromedriver.exe");
            driver = new ChromeDriver();
            }

        driver.get(utility.fetchPropertyValue("applicationURL").toString());

    }


    @AfterMethod
    public void closeDriver()
    {
```

20. att. Klases Driver saturs (Avots: autores veidots)

Ievaddatu lasīšanai no XLSX datnes tiks izmantotas Maven datu failu apstrādes metodes, šim nolūkam nepieciešams izveidot pom.xml failu un papildus importēt Apache.poi.jar spraudņus (Maven.org mājas lapa, 2019). pom.xml datnes saturs aplūkojams (21.att.).

MP2_Test/pom.xml

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="htt
  <modelVersion>4.0.0</modelVersion>
  <groupId>MP2_Test</groupId>
  <artifactId>MP2_Test</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <build>
    <sourceDirectory>src</sourceDirectory>
    <plugins>
      <plugin>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.8.0</version>
        <configuration>
          <source>1.8</source>
          <target>1.8</target>
        </configuration>
      </plugin>
    </plugins>
  </build>
  <dependencies>
    <dependency>
        <groupId>org.seleniumhq.selenium</groupId>
        <artifactId>selenium-java</artifactId>
        <version>3.7.1</version>
    </dependency>
    <dependency>
        <groupId>org.testng</groupId>
        <artifactId>testng</artifactId>
        <version>6.11</version>
    </dependency>
    <dependency>
```

21. att. Datnes pom.xml saturs (Avots: autores veidots)

Ar to, ka izveidota Maven pom.xml datne nepietiks, nepieciešams arī izveidot klasi, kura veiks .xslx datnes apstrādi un pados tālāk kodā ievadvērtības. Šim nolūkam tika izveidota klase DataGenerator (23.att.), kurā tika definēts, kur atradīsies .xlsx datne un norādīts kurā kolonā atradīsies nepieciešamā ievadvērtība. Ievadvērtībai piešķirts attiecīgs mainīgā nosaukums, kuru tālāk kodā var izmantot, rakstot funkcijas.

MP2_Test/pom.xml | DataGenerator.java

```java
    @DataProvider(name="Ligums")
    public static Object [][] tesDataGegneratorLigums() throws Exception
    {


        FileInputStream file = new FileInputStream ("C:\\Users\\ieva.ozola\\eclipse-workspace\\MP2_Test\\TestData\\TestData.xlsx"
        @SuppressWarnings("resource")
        XSSFWorkbook workbook= new XSSFWorkbook(file);
        XSSFSheet loginSheet = workbook.getSheet("Ligums");
        int numberOfData = loginSheet.getPhysicalNumberOfRows();
        Object [][] testData = new Object [numberOfData][13];

        for (int i=0;i<numberOfData;i++)
        {
            XSSFRow row = loginSheet.getRow(i);
            XSSFCell username = row.getCell(0);
            XSSFCell password = row.getCell(1);
            XSSFCell company = row.getCell(2);
            XSSFCell reg_number= row.getCell(3);
            XSSFCell vat_reg_nr = row.getCell(4);
            XSSFCell bank_account = row.getCell(5);
            XSSFCell EntPersonName = row.getCell(6);
            XSSFCell EntPersonSurname = row.getCell(7);
            XSSFCell reason = row.getCell(8);
            XSSFCell EntPersonJob = row.getCell(9);
            XSSFCell EntPersonEmail = row.getCell(10);
            XSSFCell Email = row.getCell(11);
            XSSFCell message = row.getCell(12);
            testData [i][0] = username.getStringCellValue();
            testData [i][1] = password.getStringCellValue();
            testData [i][2] = company.getStringCellValue();
```

22.att. Klase DataGenerator (Avots: autores veidots)

Lai pārbaudītu vai darbojās izveidotā klase, kas nolasa .xslx datni, datnē TestaData.xslx tika izveidota lapa Login, kura saturēja mazāk ievadvērtības un kas nodrošināja pieslēgšanos testa lapai. Pēc tam tika izveidota klase LoginPage un aprakstītas funkcijas, kas būtu jāveic, lai pieslēgtos (autorizētos testējamās vietnes lapā). Respektīvi, jānospiež poga [Pieslēgties], pēc pop-up loga parādīšanās, jāievada lietotāja e-pasta adrese un parole, kā arī jānospiež apstiprinājuma poga [Pieslēgties]. Klases LoginPage metodes, apskatāmas (23.att.).

LoginPage.java

```
3  import org.openqa.selenium.By;
8
9  public class LoginPage {
10
11     WebDriver driver;
12
13     public LoginPage(WebDriver driver) throws InterruptedException
14     {
15         this.driver=driver;
16
17         Thread.sleep(1);
18     }
19
20     public void acceptcookies() throws Exception
21     {   Thread.sleep(1000);
22         driver.findElement(By.xpath(utility.fetchLocatorValue("accept_cookies"))).click();
23
24     }
25
26
27     public void clickLogin() throws Exception
28     {
29         Thread.sleep(1000);
30         driver.findElement(By.xpath(utility.fetchLocatorValue("sign_in"))).click();
31     }
32
33     public void popupfind() throws Exception
34     {
35         Thread.sleep(500);
36         Actions act = new Actions(driver);
37         act.moveToElement(driver.findElement(By.xpath(utility.fetchLocatorValue("Login_pop_up_find"))));
```

23. att. Klase LoginPage (Avots: autores veidots)

Lai skripts darbotos, bija nepieciešams izveidot arī testa scenāriju, kurš izmantos LoginPage metodes. Klasē TC_001Validate Login (24.att.) norādīts, ka ievaddati tiek padoti no klases DataGenerator (25.att.).

TC_001_ValidateLogin.java

```java
package testcases;
import org.testng.annotations.Test;
import base.Driver;
import pages.LoginPage;


public class TC_001_ValidateLogin extends Driver

{
    @Test (dataProvider="Login", dataProviderClass=DataGenerator.DataGenerator.class)
    public void tc_001_login_funkcionality (String uname, String pass, String message) throws Exception
    {


        LoginPage login=new LoginPage(driver);
        login.acceptcookies();

        login.clickLogin();
        login.enterUsername(uname);
        login.enterPassword(pass);
        login.clickSignin();
        login.checkMessage(message);


    }

}
```

24.att. Klase TC_001_ValidateLogin (Avots: autores veidots)

DataGenerator.java

```java
package DataGenerator;

import java.io.FileInputStream;

public class DataGenerator {
    @DataProvider(name="Login")
    public static Object [][] tesDataGegnerator() throws Exception
    {

        FileInputStream file = new FileInputStream ("C:\\Users\\ieva.ozola\\eclipse-workspace\\MP2_Test\\TestData\\TestData.xlsx"
        @SuppressWarnings("resource")
        XSSFWorkbook workbook= new XSSFWorkbook(file);
        XSSFSheet loginSheet = workbook.getSheet("Login");
        int numberOfData = loginSheet.getPhysicalNumberOfRows();
        Object [][] testData = new Object [numberOfData][3];

        for (int i=0;i<numberOfData;i++)
        {
            XSSFRow row = loginSheet.getRow(i);
            XSSFCell username = row.getCell(0);
            XSSFCell password = row.getCell(1);
            XSSFCell message = row.getCell(2);
            testData [i][0] = username.getStringCellValue();
            testData [i][1] = password.getStringCellValue();
            testData[i][2] = message.getStringCellValue();
        }
        return testData;

```

25.att. Klase DataGenerator Login parametri (Avots: autores veidots)

Savukārt, lai pašu skriptu palaistu bija nepieciešams TestNG .xml fails, kurā iespējams definēt, kurus testpiemērus pēc kārtas, nepieciešams izpildīt.

Izmantojot TestNG sprauduņus, iespējams vienkāršot testēšanu, lai tos izmantotu, tiek pievienoti Eclipse izstrādes videi. TestNG nodrošina atbalstu sākot ar vienībtestēšanu un līdz pat integrācijas testēšanai (testējot vairākas sistēmas, kas veidotas no vairākām klasēm, pakotnēm

un pat vairākām ārējām sistēmām. Testēšanas komplekts sastāv no viena XML faila un tajā var būt vairāki izpildāmie testēšanas piemēri, atzīmēti ar tegu <suite>. Lai sasaistītu XML failu ar izpildāmo piemēru, testpiemēra klasē jābūt norādei @Test. Pirms testa izpildāmās darbības tiek norādītas ar anotāciju @Before un pēc testu darbības tiek atzīmētas ar @After (TestNG mājas lapa, 2019).

Konkrētajā gadījumā, TestNG XML failā tika norādīts viens izpildāmais testpiemērs (26.att.).

testng.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
3  <suite name="Suite">
4    <test name="Test">
5      <classes>
6          <class name="testcases.TC_001_ValidateLogin"/>
7      </classes>
8    </test>
9  </suite>
10
```

26.att. TestNG XML fails (Avots: autores veidots)

Pēc pirmā testa palaišanas tika konstatēts, ka skripts neizpildās kā plānots, respektīvi, tika nospiesta [Pieslēgties] poga, bet ievaddati pop-up logā netika ievadīti. Tika atgriezta kļūda, par to, ka netiek atrasts web elements. Lai risinātu situāciju, procedūrām tika uzlikts gaidīšanas laiks, no brīža, kad tiek nospiesta poga [Pieslēgties], līdz brīdim, kad parādās pop-up logs. Pēc šiem labojumiem, skripts izpildījās kā plānots. Pēc šāda pat principa tika sagatavots tests līguma pieteikuma formai. Ar klasi PieteiktLigumu.java tiek aprakstītas veicamās darbības, tādā pat veidā kā klasē LoginPage. Tikai šajā klasē tiek aprakstītas Līgumu formas lauku apstrāde un ievadāmie parametri, šajā klasē bija daudz vairāk procedūru, nekā klasē LoginPage, jo Līguma pieteikuma forma satur daudz vairāk ievaddatu un veicamo darbību, daļa no klases PieteiktLigumu procedūrām apskatāmas (27.att.).

*PieteiktLigumu.java

```
123
124     public void enterEntPersonJob(String entPersonJob) {
125         try {
126             driver.findElement(By.xpath(utility.fetchLocatorValue("field_EntPersonJob"))).sendKeys(entPersonJob);
127         } catch (IOException e) {
128             e.printStackTrace();
129         }
130
131     }
132     public void enterEntPersonEmail(String entPersonEmail) throws IOException {
133         driver.findElement(By.xpath(utility.fetchLocatorValue("field_EntPersonEmail"))).sendKeys(entPersonEmail);
134
135     }
136
137     public void enterEmail(String Email) throws Exception
138     {
139     //  Thread.sleep(2000);
140         driver.findElement(By.xpath(utility.fetchLocatorValue("field_email"))).sendKeys(Email);
141     }
142
143     public void checkAdress() throws Exception
144     {
145         //Thread.sleep(2000);
146         if (!driver.findElement(By.xpath(utility.fetchLocatorValue("check_box"))).isSelected());
147         {
148             driver.findElement(By.xpath(utility.fetchLocatorValue("check_box"))).click();
149         }
150     }
151
152     public void checkCorrectData() throws Exception
153     {
154         Thread.sleep(200);
155         if ( !driver.findElement(By.xpath(utility.fetchLocatorValue("check_box_accept"))).isSelected());
156         {
157             driver.findElement(By.xpath(utility.fetchLocatorValue("check_box_accept"))).click();
158         }
159     }
160
161     public void clickSubmit() throws Exception
162     {
163         driver.findElement(By.xpath(utility.fetchLocatorValue("button_Send"))).click();
164     }
165
166     public void checkMessage(String message) throws Exception
167     {
168         String actual_error = driver.findElement(By.xpath(utility.fetchLocatorValue("message_xpath"))).getText();
169         String expected_error=(message);
170         Assert.assertEquals(actual_error, expected_error);
171         Assert.assertTrue(actual_error.contains(message));
172         System.out.println(actual_error);
173     }
174
```

27.att. Klase PieteiktLigumu (Avots: autores veidots)

Tika izveidota arī testpiemēra izpildes klase TC_002_Ligums, kurā tika norādīts, ka datus nepieciešams ielādēt no .xslx faila, tāpat tika definēta procedūru secība, kādā nepieciešams veikt darbības. Šajā testpiemērā, tika izmantotas daļa no LoginPage procedūrām, tādas kā sīkdatņu akceptēšana, pogas pieslēgties nospiešana un pieslēgšanās sistēmai (28.att.).

TC_002_Ligums.java

```java
package testcases;
import org.testng.annotations.Listeners;
import org.testng.annotations.Test;

import Listeners.CustomListeners;
import base.Driver;
import pages.LoginPage;
import pages.PieteiktLigumu;
@Listeners(CustomListeners.class)

public class TC_002_Ligums extends Driver

{
    @Test (dataProvider="Ligums", dataProviderClass=DataGenerator.DataGenerator.class)
    public void tc_002_ligums_funkcionality (String uname, String pass, String company, String reg_number, String vat_reg_nr,
            String bank_account, String EntPersonName, String EntPersonSurname, String reason, String EntPersonJob, String EntPersonEmail,  String Email, String message) throws Exception
    {


    LoginPage login=new LoginPage(driver);
        PieteiktLigumu ligums = new PieteiktLigumu(driver);
        login.acceptcookies();

        login.clickLogin();
        login.enterUsername(uname);
        login.enterPassword(pass);
        Thread.sleep(200);
        login.clickSignin();
        ligums.checkUser();
        ligums.clickLigumsBtn();
        ligums.enterCompany(company);
        ligums.enterRegNr(reg_number);
        ligums.enterVatRegNr(vat_reg_nr);
        ligums.enterBank_account(bank_account);
        ligums.enterEntPersonName(EntPersonName);
        ligums.enterEntPersonSurname(EntPersonSurname);
        ligums.listEntPersReason(reason);
        ligums.enterEntPersonJob(EntPersonJob);
        ligums.enterEntPersonEmail(EntPersonEmail);
        ligums.enterEmail(Email);
        ligums.checkAdress();
        ligums.checkCorrectData();
        ligums.clickSubmit();
    ligums.checkMessage(message);

    }

}
```

28.att.TC_002_Ligums (Avots: autores veidots)

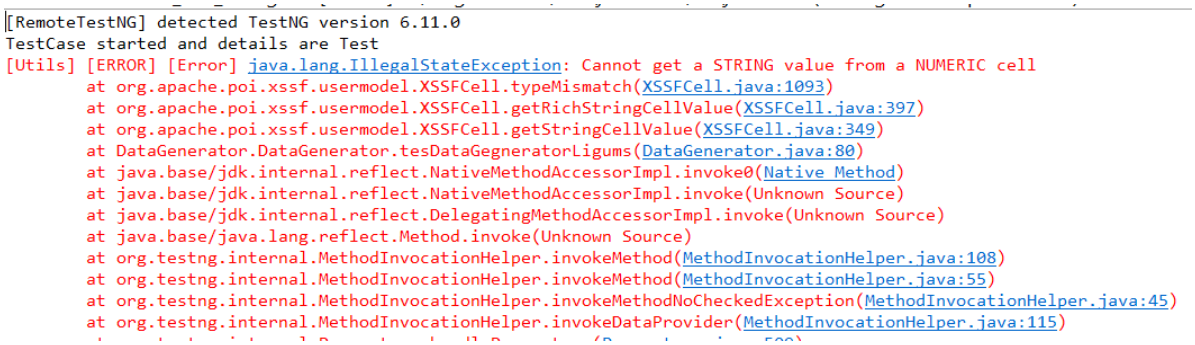
Arī TestNG .xml failā tika nomainīts izpildāmais testpiemērs uz TC_002_Ligums (29.att.).

testng.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite name="Suite">
  <test name="Test">
    <classes>
        <class name="testcases.TC_002_Ligums"/>
    </classes>
  </test>
</suite>
```

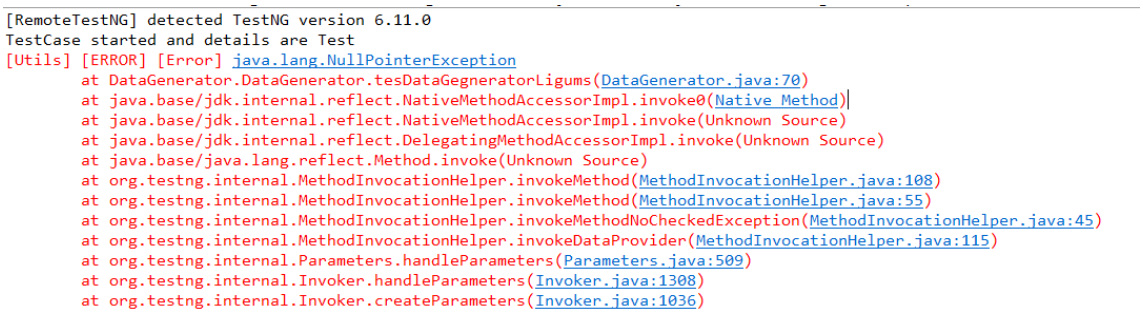
29.att. TestNG XML fails (Avots: autores veidots)

Pirmo reizi palaižot TC_002_Ligums skriptu, tika atgrieztas divas kļūdas viena par to, ka sagaidāma String tipa vērtība, bet saņem Integer tipa vērtība (30.att.).



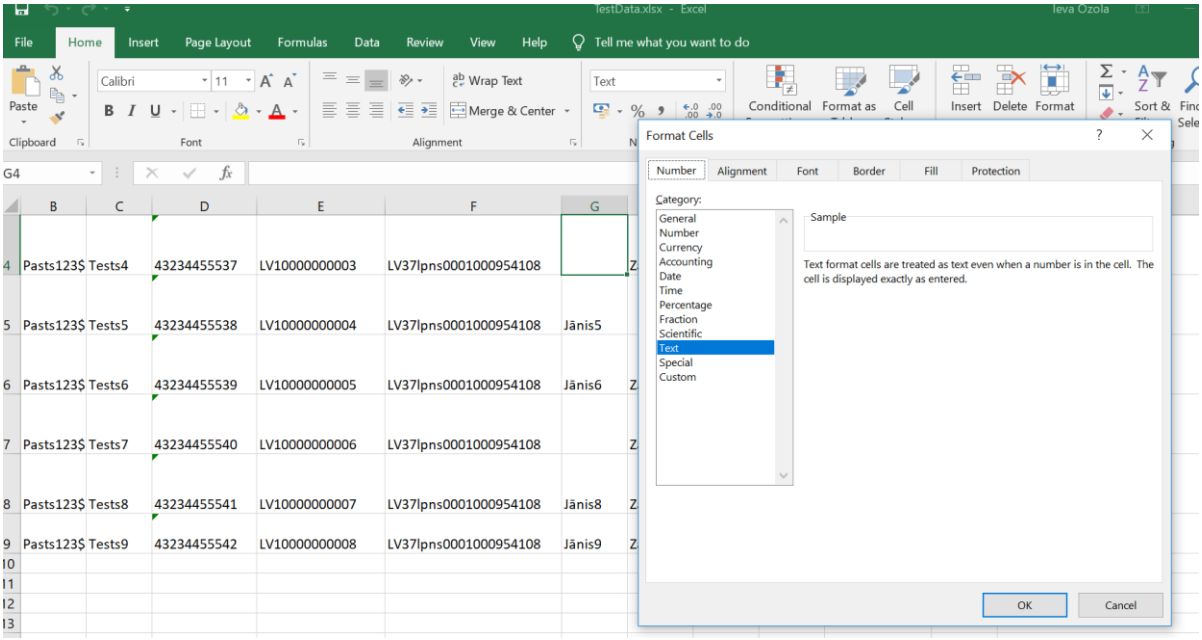
30.att. Kļūda par String tipa vērtību (Avots: autores veidots)

avukārt otra kļūda bija par to, ka fails satur nulles vērtības (31.att.), tas nozīmē, ka skripts nelasīja šūnas, kurās nav norādītas vērtības, skriptam šinī gadījumā vajadzēja neņemt vērā tukšās šūnas, bet turpināt lasīt failu un vadīt ievaddatus ievadlaukos no nākamajās šūnās norādītajām vērtībām.



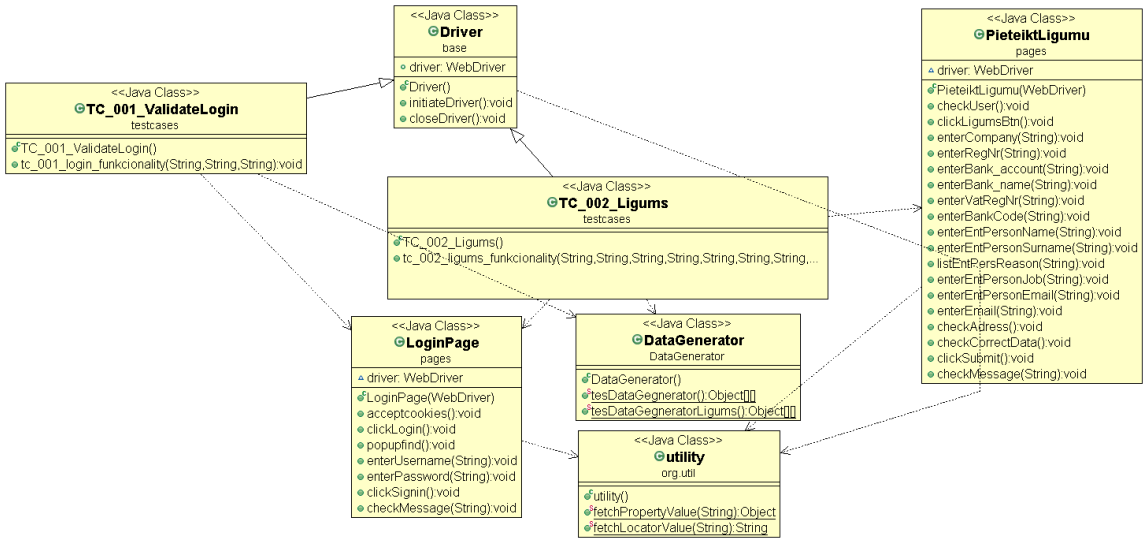
31.att. Kļūda par Nulles vērtībām failā (Avots: autores veidots)

Šinī gadījumā autore meklēja informāciju tīmeklī (izrādās šāda problēma bija ne mazums lietotāju, kuri veica skriptu izstrādi) un atrada informāciju, ka nepieciešams .xslx failā tukšajām šūnām un šūnās, kur atrodas Integer tipa vērtības, nomainīt datu tipu uz Text (32.att.).



32. Tukšo šūnu un šūnu ar Integer tipa datiem formāta maiņa .xslx (Avots: autores veidots)

Pēc šo labojumu veikšanas kļūdas vairs netika konstatētas un skripts izpildījās kā plānots. Visas programmas UML diagramma aplūkojama (34.att).

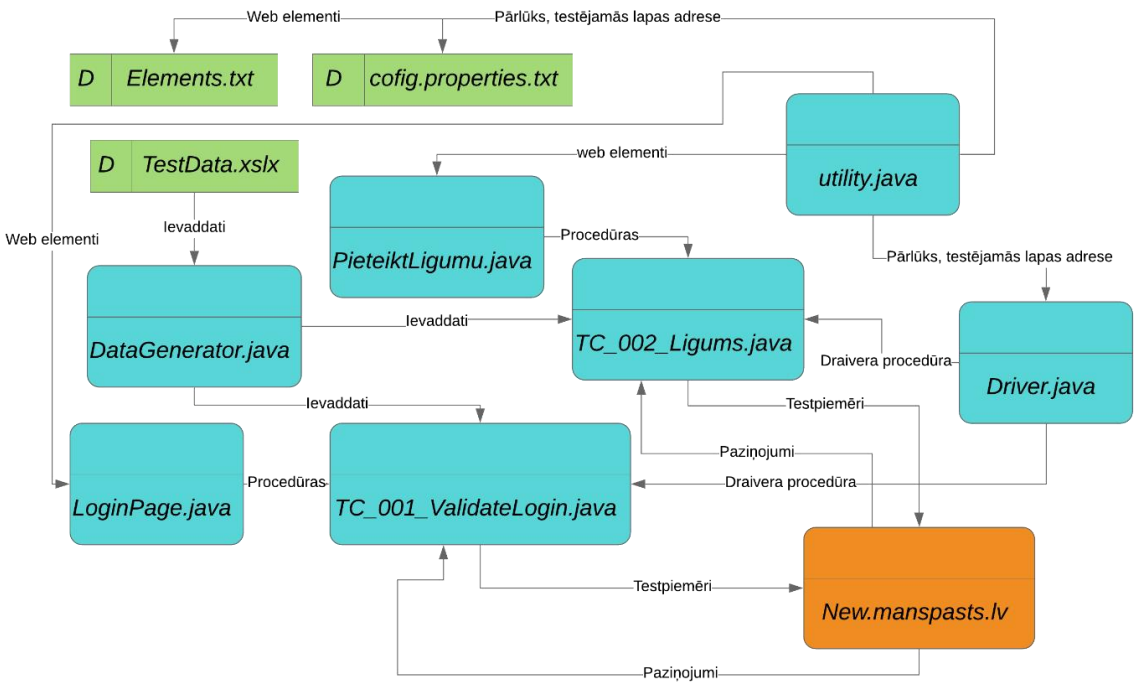


33.att. UML diagramma (Avots: autores veidots)

Tajā redzams, ka pie klases DataGenerator vēršas klases TC_001_ValidateLogin un TC_002_Ligums. Savukārt pie klases LoginPage vēršās gan TC_001_ValidateLogin, gan TC_002_Ligums, jo testpiemēra izpildei izmanto procedūras, kas nodrošina

pieslēgšanos/autorizēšanas testējamajā vietnē. Pārējo klašu attiecības iespējams apskatīt (33.att.).

Datu plūsmas diagrammā (34.att.) var apskatīt kā klases izmanto ārējos failus. Tajā redzams, ka klase DataGenerator izmanto TestData.xlsx failu un tālāk datus nodod klasēm TC_001_Ligums un TC_002_ValidateLogin. Savukārt failu Elements.txt izmanto klases LoginPage un PieteiktLigumu, saņem web elementu XPath adresi caur klasi utility. Klase Driver saņem informāciju kā web pārlūkprogrammas draiveri izmantot un kādu testējamo lapu atvērt, tālāk jau šo informāciju nododot izpildklasēm TC_001_ValidateLogin un TC_002_Ligums. Tālāk jau klases TC_001_ValidateLogin un TC_002_Ligums, izmantojot klasi Driver vēršas pie testējamās lapas.

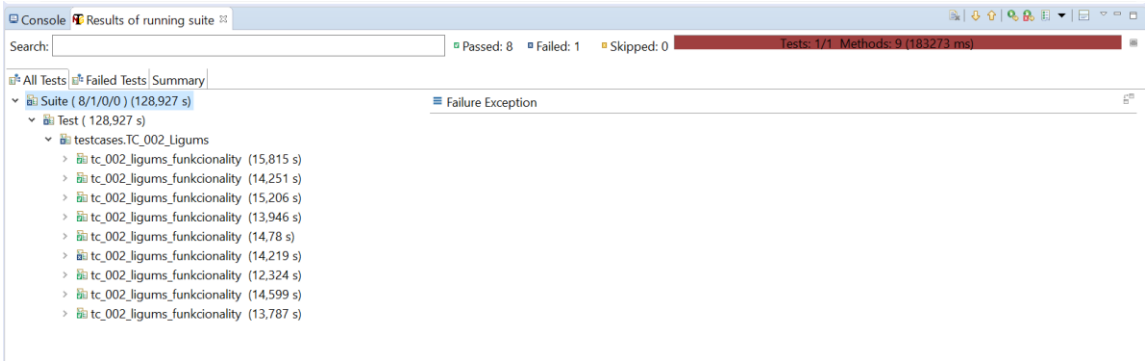


34.att. Datu plūsmas diagramma (Avots: autores veidots)

3.3.Skriptu izmantošanas izvērtējums

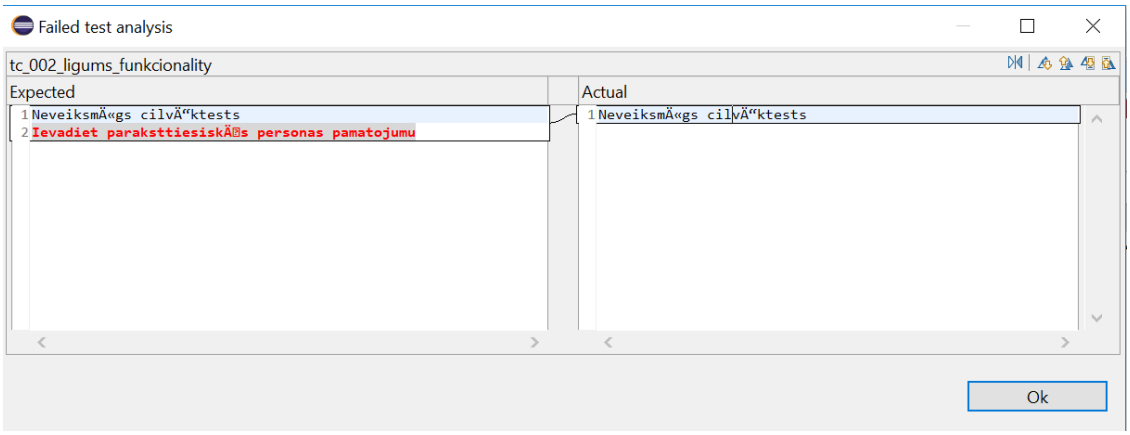
Lai izvērtētu automatizētu skriptu izpildes lietderīgumu, no sākuma jāveic šie paši testi, izmantojot manuālo testu pieejas metodi (tātad šie paši dati jāvada cilvēkam). Salīdzināšanas kritērijs šinī gadījumā būs laiks – cik ātri to spēj izdarīt cilvēks un cik ātri to paveic automatizēts skripts. Izpildot šos testpiemērus manuāli, testu izpildes laiks viena piemēra izpildei aizņēma 2

minūtes. Kopā šo testpiemēru bija deviņi un kopējais patērētais laiks bija 20 minūtes, jo manuāli izpildot testus ir iespēja kļūdīties, vadot datus, kas arī šinī gadījumā notika. Turklāt, lai veiktu laika uzskaiti nepieciešama, kāda programma, kuru palaižot tiek sākta laika uzskaite. Automatizētu skriptu izpildes gadījumā, logfailā redzams cik testi ir izpildījušies pozitīvi un cik negatīvi, kā arī patērētais laiks testiem (35.att.).



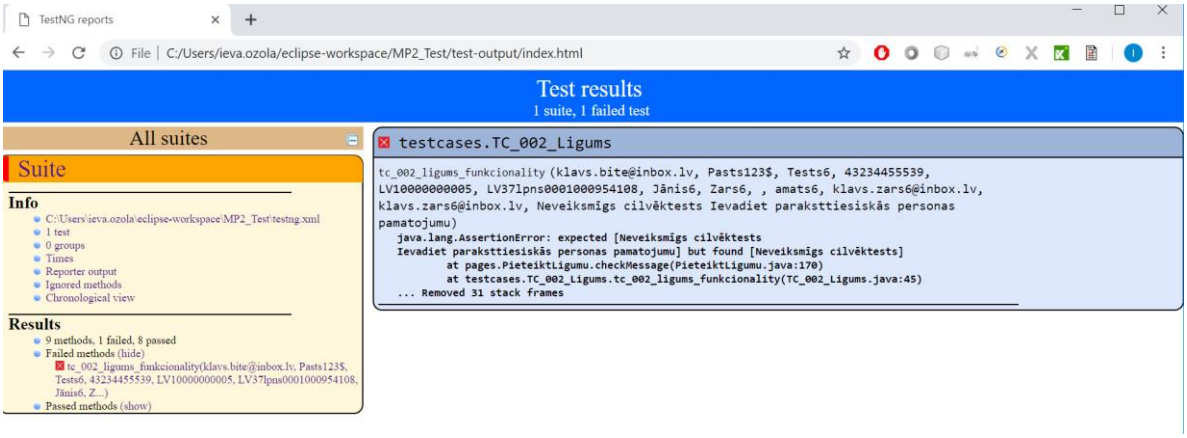
35.att. TestNG logfails ar izpildes failu (Avots: autores veidots)

Konkrētā situācijā redzams, ka 8 no 9 ir izpildījušies, bet viens, ne. Nospiežot uz neveiksmīgā testa ir iespējams redzēt tā detaļas (36.att.). Šajā gadījumā iemesls ir bijis neatbilstošs paziņojums, testā vajadzēja būt diviem paziņojumiem, bet bija tikai viens. Tātad šinī gadījumā tika konstatēta kļūda testējamā vietnē. MP2 nekontrolēja vai ir aizpildīts paraksttiesiskās personas amats.



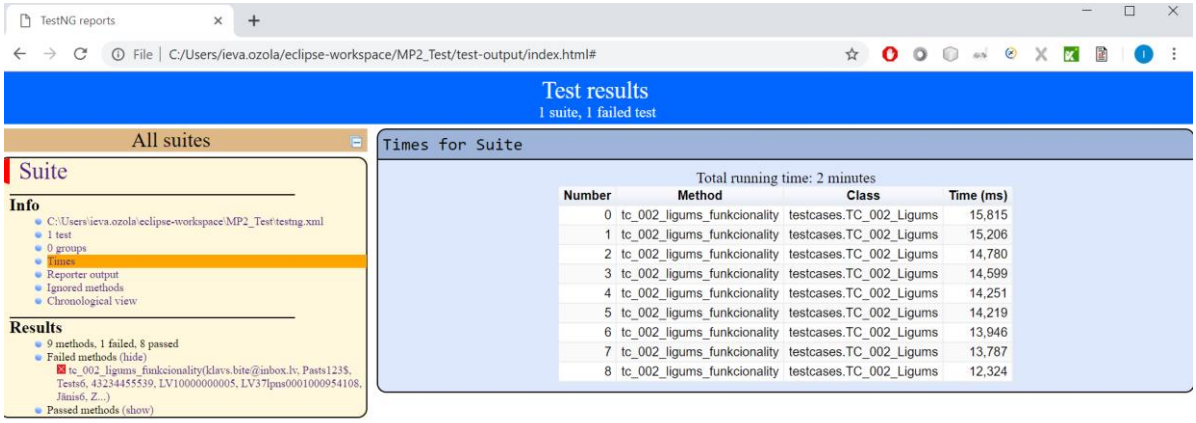
36.att. Neveiksmīgā testa detaļas (Avots: autores veidots)

TestNG ir pieejama arī .html formāta atskaite (37.att.), kurā arī redzams kopējo testpiemēru skaits un rezultāts. Šajā atskaitē, labāk ir redzams iemesls kāpēc tests nav izpildījies.



37.att. TestNG atskaite .html formātā (Avots: autores veidots)

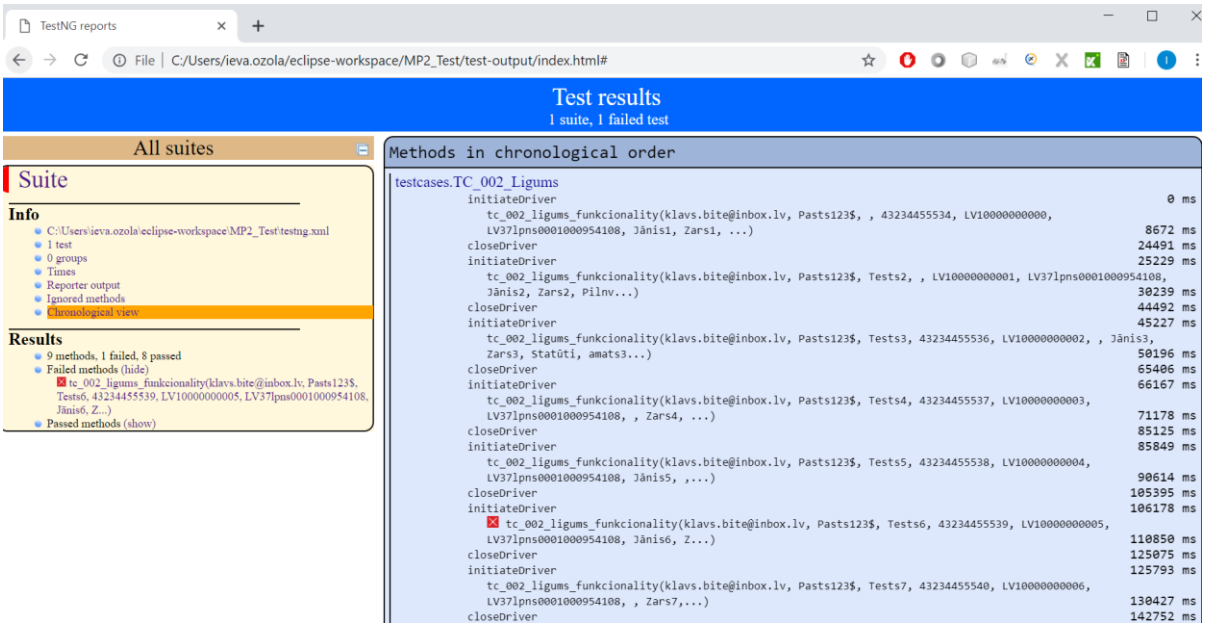
Vēl šajā TestNg atskaitē iespējams apskatīt testa rezultātus dažādos griezumos, piemēram cik ilgā laikā ir izpildījies tests, šajā gadījumā, redzams, ka kopējais izpildes laiks ir 2 minūtes (38.att.).



38.att. TestNG atskaite .html formātā testu izpildes laiki (Avots: autores veidots)

Atskaite arī nodrošina veikto testu hronoloģiskās secības atspoguļojumu (37.att.), kura ir redzams, kādi ievaddati tikuši vadīti un cik daudz laika ir ticis patērēts konkrētajam

testpiemēram. Arī šajā atskaitē redzams, kuri testi ir pozitīvi un kuri negatīvi. Negatīvajam testam ir pamanāms sarkans kvadrāts (39.att.).



39.att. TestNG atskaite hronoloģiskā secībā (Avots: autores veidots)

Ja salīdzina, kāds ir ieguvums testus veicot manuāli un automatizēti, tad konkrētā testēšanas vienuma (9 testēšanas piemēri) izpilde ar automatizētu testēšanas skriptu, aizņēma 2 minūtes. Manuāli izpildot šos pašus testus tika patērētas 20 minūtes. Tātad reālais ieguvums ir 18 minūtes. Tātad automatizētā testēšana veic šos testu par 10 reizēm ātrāk nekā manuālā testēšana. Tas liek izdarīt secinājumus, ka ir vērts turpināt automatizēt manuālo testu kopu, kas savukārt samazinās testēšanai nepieciešamo laiku.

Turklāt testus iespējams uzlabot, rezultātus fiksējot .xlsx datnē, to izpildes laikā, rakstot pie konkrētā testpiemēra PASS vai FAIL. Veicot papildus izpēti un padziļinot zināšanas Selenium un Java izstrādē, iespējams ir izveidot atsevišķu aplikāciju, kuru varētu palaist arī kāds cits lietotājs, jo pašreizējā gadījumā, nepieciešams dators ar Eclipse izstrādes vidi.

Šis automatizētās testēšanas risinājums ir elastīgs, jo ja gadījumā mainīsies šīs formas paziņojumi, vai tiks pievienoti kādi jauni ievadlauki, pazīmes u.c. web elementi, tos iespējams pievienot. Kā arī ir iespējams pievienot vēl testpiemērus, jau esošajiem – ja nepieciešams pārbaudīt ar vēl savādākiem datiem, nekā tie ir definēti šajā izpildes variantā.

SECINĀJUMI UN PRIEKŠLIKUMI

Secinājumi

1. Atvērtā koda testēšanas rīku ir daudz, bet konkrētajai situācijai piemēroti ir tikai pāris.
2. Selenium ir populārākais atvērtā koda testēšanas rīks.
3. Testu izstrāde prasa daudz laika resursu, bet sākotnējais laika patēriņš ir ieguvums, veicot nākamos testus.
4. Veicot testu izstrādi ir būtiski norādīt gaidīšanas laiku, gadījumos, kad lapas elementi līdz galam nav ielādējušies un skriptam pieejami.
5. Salīdzinot automatizētā testēšanas skripta izpildi (2 minūtes) ar manuālu izpildi (20 minūtes), konkrētajā situācijā, ieguvums ir 18 minūtes, kas ir 10 reizes ātrāk nekā manuāla testu izpilde.
6. Izstrādātais risinājums ir fleksibls un piemērots new.manspasts pārējo funkciju testēšanai.
7. Izstrādāto risinājumu, iespējams piemērot arī citai tīmekļa vietnei.

Priekšlikumi

1. Testēšanas vadītājam izstrādāt regresa testēšanas uzdevumu kopu, kurus turpmāk paredzēts automatizēt;
2. Testēšanas vadītājam apmācīt testētājus veikt regresa testēšanu, izmantojot automatizētās testēšanas rīku.
3. Testēšanas vadītājam, uzlabot testēšanas skriptus ar testēšanas rezultāta rakstīšanu .xlsx datnē;
4. Testēšanas vadītājam, izskatīt iespēju uzlabot testēšanas rīka atskaišu atspoguļojumu.
5. Testēšanas vadītājam, uzlabot testēšanas skriptu, izveidojot atsevišķu aplikāciju.
6. Testēšanas vadītājam, izskatīt iespēju, piemērot izstrādāto risinājumu citas tīmekļa vietnes testēšanai.

IZMANTOTĀS LITERATŪRAS UN INFORMĀCIJAS AVOTU SARAKSTS

1. Apache Commons (2019). *Jelly* Elektroniskais resurss [skatīts 09.04.2019]. Pieejams: https://commons.apache.org/proper/commons-jelly/
2. Apache Maven Project (2019). *Welcome to Apache Maven* Elektroniskais resurss [skatīts 09.04.2019]. Pieejams: https://maven.apache.org/
3. Apache JMeter (2019). *About* Elektroniskais resurss [skatīts 02.04.2019]. Pieejams: https://jmeter.apache.org/
4. AutonomlQ (2019). *The Free Time Saver Tool for UI Automation Developers* Elektroniskais resurss [skatīts 09.04.2019]. Pieejams: https://autonomiq.io/chropath/
5. Cerebrus Testing Github (2019). *Introduction to Cerberus* Elektroniskais resurss [skatīts 22.03.2019]. Pieejams:https://cerberustesting.github.io/documentation_en.html
6. Chopra, R.(2018). *Software Testing: A Self-Teaching Introduction*. Sterling (Virginia): Mercury Learning and Information.
7. Demo Cerberus (2019). *Demo Cerebrus Testing* Elektroniskais resurss [skatīts 09.04.2019]. Pieejams: https://demo.cerberus-testing.org/
8. Dustin, E., Rashka, J., & Paul, J. (1999). *Automated software testing: introduction, management, and performance*. New York: Addison-Wesley Professional.
9. Eclipse org (2019). *The Platform for Open Innovation and Collaboration* Elektroniskais resurss [skatīts 11.03.2019]. Pieejams: https://www.eclipse.org/
10. Faulkner, C. (2009). *Software Engineering*. Delhi: Global Media.
11. FitNesse (2019). *FitNesse Features* Elektroniskais resurss [skatīts 02.04.2019]. Pieejams: https://www.upwork.com/hiring/development/golang-programming-language/
12. Free Desktop ogr (2019). *LDTP* Elektroniskais resurss [skatīts 02.04.2019]. Pieejams: https://ldtp.freedesktop.org/wiki/
13. Gauge org (2019). *Gauge Documentation* Elektroniskais resurss [skatīts 09.04.2019]. Pieejams: https://docs.gauge.org/latest/index.html
14. GitHub (2019). *About GitHub* Elektroniskais resurss [skatīts 09.04.2019]. Pieejams: https://github.com/about
15. Guru 99 (2019). *XPAth in Selenium WebDriver: Complete Tutorial* Elektroniskais resurss [skatīts 02.04.2019]. Pieejams: https://www.guru99.com/xpath-selenium.html

16. Guru 99 (2019). *What is Selenium? Introduction to Selenium* Elektroniskais resurss [skatīts 10.04.2019]. Pieejams: https://www.guru99.com/introduction-to-selenium.html
17. ISTQB (2019). *ISTQB Partner program* Elektroniskais resurss [skatīts 12.04.2019]. Pieejams: https://partner.istqb.org/
18. ISTQB org (2019). *Certifying Software Testers Worldwide* Elektroniskais resurss [skatīts 02.03.2019]. Pieejams: https://www.istqb.org/
19. Latvijas Pasts (2019). *Par Latvijas Pastu* Elektroniskais resurss [skatīts 28.02.2019] Pieejams: http://www.pasts.lv/lv/par_mums/
20. Latvijas Standarts LVS 70:1996. Elektroniskais resurss [skatīts 01.03.2019] Pieejams: estudijas.lu.lv/mod/resource/view.php?id=131430
21. Latvijas Standarts LVS 73:1996. Elektroniskais resurss [skatīts 01.03.2019] Pieejams: estudijas.lu.lv/mod/resource/view.php?id=131429
22. Latvijas Unversitātes Datorikas fakultāte (2014). *"SQUALIO" izstrādājis jaunus tiešsaistes rīkus – kvalitātes testu un slodzes testu* Elektroniskais resurss [skatīts 02.04.2019]. Pieejams: https://www.df.lu.lv/zinas/t/10109/
23. Lewis, W. E. (2009). *Software testing and continuous quality improvement*. New York: Auerbach publications.
24. Odo (2014). *Programmatūras testēšanas rokasgrāmata vadītājiem*. Elektroniskais resurss [skatīts 09.04.2019]. Pieejams: https://odo.lv/ftp/docs/squalio_testesanas_rokasgramata.pdf
25. Robot Framework org (2019). *Introduction* Elektroniskais resurss [skatīts 02.03.2019]. Pieejams: https://robotframework.org/#support
26. Sandeep, D., Abhishek, S. (2016). Software Testing: A Practical Approach, Second Edition. Delhi: PHI Learning Private Limited.
27. Software Testing Conferences (2019). *Current Conferences* Elektroniskais resurss [skatīts 20.04.2019]. Pieejams: https://testingconferences.org/
28. Spillner, A., Linz, T., & Schaefer, H. (2014). *Software testing foundations: a study guide for the certified tester exam*.Santa Barbara: Rocky Nook, Inc.
29. SIA Jauno Tehnoloģiju Centrs (2010). *Testēšanas pamati*. (nepublicēts materiāls).
30. Selenium HQ (2019). SeleniumHQ Browser Automation Elektroniskais resurss [skatīts 09.04.2019]. Pieejams: https://www.seleniumhq.org/

31. SPRK gov (2019) *Pārskats par universiālo pasta pakalpojuma kvalitāti 2017.gads* Elektroniskais resurss [skatīts 01.03.2019] Pieejams: https://www.sprk.gov.lv/uploads/doc/PrskatsUPPkvalitte2017.pdf
32. Tapost org (2019). *TAPOST conferences* Elektroniskais resurss [skatīts 20.04.2019]. Pieejams: http://www.tapost.org/
33. TestNG org (2019). *Introduction* Elektroniskais resurss [skatīts 12.04.2019]. Pieejams: https://testng.org/doc/documentation-main.html#introduction
34. The most striking problems in test automation: A survey (2018). Elektroniskais resurss [skatīts 23.03.2019] Pieejams: https://d1h3p5fzmizjvp.cloudfront.net/wp-content/uploads/2018/06/05101901/The-Most-Striking-Problemns-in-Test-Automation-A-Survey.pdf
35. Ventspils (2013). *Ventspils Augsto tehnoloģiju parka Biznesa inkubatora uzņēmums SIA "TestDevLab" nesen ierindojās Starptautiskās Programmatūras testēšanas kvalifikācijas padomes (ISTQB - International Software Testing Qualification Board) zelta patneru sarakstā* Elektroniskais resurss [skatīts 02.04.2019]. Pieejams: https://www.ventspils.lv/lat/ekonomika/53651-sia-testdevlab-iegust-istqb-zelta-partnera-statusu
36. QA Testing Tools (2019). *All about Software Testing Tools* Elektroniskais resurss [skatīts 09.04.2019]. Pieejams: http://www.qatestingtools.com/
37. Watir (2019). *Guides* Elektroniskais resurss [skatīts 02.04.2019]. Pieejams: http://watir.com/guides/

Galvojums

Ar šo es galvoju, ka bakalaura darbs “Automatizēto testēšanas skriptu izstrāde tīmekļa lietotnei” ir izstrādāts patstāvīgi, tajā nav pieļauts citu personu intelektuālā īpašuma tiesību pārkāpums vai plaģiāts – citas personas radošās darbības rezultātu tālākā paušana savā vārdā. No citiem avotiem ņemtajiem darbiem, definējumiem un citātiem darbā ir uzrādītas atsauces. Izmantoti citu autoru pētījumu rezultāti un datu avoti ir norādīti atsaucēs. Darbs nekad nav publicēts un pirmo reizi tiek iesniegts aizstāvēšanai
Valsts noslēguma pārbaudījuma komisijā.

Apliecinu, ka EKA *Moodle* sistēmā augšupielādētā darba teksts ir identisks papīra formātā iesniegtā darba tekstam.

_____ / Ieva Ozola