TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Helina Sosi 185654IABB

# E-kaubanduse rakenduste automaattestimise analüüs Opus Online OÜ näitel

Bakalaureusetöö

Juhendaja: Viljam Puusep
MSc

Tallinn 2023

# Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Helina Sosi

17.05.2023

# Annotatsioon

Käesolev bakalaureusetöö käsitleb aktuaalset probleemi, kus enamikel Opus Online OÜ poolt arendatud e-kaubanduse rakendusi testitakse vaid manuaalselt, mille tõttu võib langeda tarkvara kvaliteet. Lisaks sellele on automaatsete testide arendamiseks valitud raamistik vaid ühe töötaja arvamuse põhjal.

Töö käigus analüüsitakse automaattestide vajalikust Opus Online OÜ poolt arendatud e-kaubanduse lahendustele. Lisaks sellele hinnatakse nelja erineva raamistiku sobivust ettevõttes Opus Online OÜ arendatud e-kaubanduse rakenduste automaattestimiseks.

Ettevõte plaanib analüüsist saadud positiivsete tulemuste põhjal rakendada automaatset testimist ka projektidele, kus seda hetkel ei kasutata.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 21 leheküljel, 6 peatükki, 3 joonist, 1 tabelit.

# Abstract

## Analysis of Automated Testing for E-commerce Applications using Opus Online OÜ as an Example

This thesis deals with a current problem, where most of the e-commerce applications developed by Opus Online OÜ are only tested manually, which can lead to a decrease in the quality of the software. In addition, the framework chosen for the development of automated tests is based on the opinion of only one employee.

In the course of the work, the need for automatic tests for the e-commerce solutions developed by Opus Online OÜ is analyzed. In addition, the suitability of four different frameworks for automatic testing of e-commerce applications developed at Opus Online OÜ is evaluated.

Based on the positive results obtained from the analysis, the company plans to introduce automatic testing also for projects where it is not currently used.

The thesis is in Estonian and contains 21 pages of text, 6 chapters, 3 figures, 1 table.

# Lühendite ja mõistete sõnastik

| | |
|---|---|
| API | *Application programming interface*, rakendusliides |
| HTML | *HyperText Markup Language*, hüperteksti märgistuskeel |

# Sisukord

# Jooniste loetelu

# Tabelite loetelu

# 1 Sissejuhatus

Tarkvara testimine on olulisel kohal tarkvara arenduse protsessis. Testimise käigus kontrollitakse tarkvara nõuetele vastavust, töökindlust ning jõudlust, et teha kindlaks tarkvara kvaliteet ja kasutajasõbralikus.

Tarkvara testimine jaguneb peamiselt manuaalseks ning automaatseks testimiseks. Manuaalse testimise käigus viib testija käsitsi läbi test-stsenaariumeid, et kontrollida rakenduse käitumist ja nõuetele vastavust. Automaattestimise puhul kasutatakse erinevaid tehnilisi vahendeid ja metoodikat testide automaatseks läbiviimiseks.

Nii manuaalsel kui automaatsel testimisel on omad plussid ja miinused. Manuaalne testimine on paindlikum tänu inimese võimele paremini tuvastada ootamatut käitumist ning piiripealseid juhtumeid, kuid ajakulu on suurem ning võib tekkida inimlikke eksimusi. Automaatne testimine on efektiivne ja kiire pakkudes täpseid ja järjepidavaid tulemusi igal käivitamisel, aga vajab alustuseks suuremat investeeringut arendusmahu näol. Seetõttu on mõistlik kaaluda automaatse ning manuaalse testimise kombineerimist. [1]

## 1.1 Taust ja probleem

Käesoleva lõputöö aluseks on ettevõtte Opus Online OÜ tarkvara testimise protsess, mis sisaldab endas manuaalset testimist ning automaattestide arendamist. Hetkel testitakse e-kaubanduse tiimis arendusi enne lõppkasutajani jõudmist enamikel projektidel vaid manuaalselt kuna varasemalt pole tiimis olnud testijaid kellel oleks huvi või oskused projektidele automaatteste luua, kuid on ka projekte millele on arendatud automaattestid. Projekte, millele on arendatud automaattestid, testitakse siiski ka manuaalselt üle, et kontrollida disaini ning hinnata kasutajamugavust. Hetkel kasutatakse automaattestide arendamiseks Cypress raamistiku ja JavaScripti.

Peamine probleem automaattestimisega on selle suur ressursi nõudlikus. Automaatsete testide planeerimine, arendamine ja hooldamine nõuavad arvestatava hulga aega. Lisaks

sellele eeldab see kompetentset tööjõudu kellel on oskused ning teadmised automaattestide arendamisest. Tarkvara arendust teenusena pakkuva ettevõtte tehtud arenduste eest maksab klient, seega igasugune lisakulu tuleb põhjalikult põhjendada ning kliendile veenvalt müüa. Seetõttu tuleks ka automaattestimise vajalikust tõestada Opuses poolt arendatud e-kaubanduse rakendustele. Kui automaattestide vajalikus on tõestatud, tuleks valida nende arendamiseks mõistlikult ka õige raamistik paljutest tänapäeval saadaval olevatest valikutest.

## 1.2 Eesmärk

Lõputöö eesmärgiks on analüüsida e-kaubanduse rakenduste automaattestimist võttes aluseks Opus Online OÜ-s kasutatava testimise protsessi. Analüüsi esimeseks osaks on uurida praegust testimise protsessi ning hinnata automaattestide praktilisust ja vajalikust.

Teiseks osaks on hinnata erinevaid raamistike ning nende sobivust e-kaubanduse rakenduste testimiseks. Tänapäeval on palju erinevaid automaattestimise raamistikke, ning oluline on võtta arvesse erinevate raamistike eripära ning uurida nende sobivust Opuse poolt arendatud e-kaubanduse rakendustele.

Selline analüüs on tuleb kasuks Opusele eelkõige seetõttu, et enamasti on kliendid nõus maksma vaid teatud mahu arenduste eest seega efektiivne ajakasutus on äärmisel olulisel kohal. Iga minut arenduskulu tuleb kliendile müüa ja põhjendada ajakulu mõistlikkust. Siiski ei tohi unustada arenduste kvaliteedi tähtsust. Kui automaatestimise vajadus on selgelt põhjendatud, on seda arendust kliendile lihtsam müüa. Ka testimiseks kasutatava vahendi valikut on oluline uurida, sest vale vahendi valimine, mis ei vasta täielikult vajadustele võib tekitada kliendile kahjumit. Kui valitud vahend ei vasta vajadustele, võib testimisele endiselt kuluda suurem ajakulu ja testide arendamisele kulunud arenduskulu on olnud kasutu.

Veebirakenduste automaattestimisest on lõputöid tehtud ka varem, kuid mitte keskendutud e-kaubanduse rakendustega seotud eripäradele. Selle töö eesmärgiks on uurida automaatset testimist võttes arvesse Opuses arendatud e-poodide omadusi.

## 1.3 Ettevõtte taust

Opus Online OÜ on tehnoloogiaettevõtte, mis tegeleb juba üle 10 aasta e-kaubanduse lahenduste ja veebitarkvara arendamisega. Ettevõte pakub oma klientidele projekti nullist üles ehitamist ning selle valmimisel täiemahulist tehnilist tuge. Opuses luuakse Magento 2 abil võimsaid ja hästi töötavaid e-poode. E-commerce tiimi kuuluvad arendajad, analüütikud, testijad ning disainerid. Projektid ehitatakse üles täielikult kliendi sisendi põhjal. See tähendab, et kliendid annavad projektijuhile loetelu oma ärinõuete kohta ning nõudmised disaini osas. Vajadusel pakutakse tehnilist tuge ning arendusteenust ka juba olemasolevatele e-poodidele. Lisaks arendusele pakub e-commerce tiim võimalust tellida auditit oma olemasolevale leheküljele. [2]

## 1.4 Töö struktuur

Töö teine peatükk tutvustab töös kasutatavaid metoodikaid. Tutvustatakse ettevõtet Opus Online OÜ ning räägitakse lähemalt töös kasutatavatest tööriistadest. Lisaks sellele räägitakse töö protsessist ning kirjeldatakse kuidas viiakse läbi analüüsi. Kolmandas peatükkis vaadatakse lähemalt Opus Online OÜ praegust testimise protsessi ning hinnatakse automaattestimise vajalikust selles protsessis. Uuritakse ka erinevaid testimise raamistikke ning vajadusi mida tuleks raamistiku valikul silmas pidada. Analüüsitakse erinevate raamistike võimet katta Opuses arendatud e-poodide spetsiifilisi nõudmisi. Töö tulemusena tuuakse välja automaattestide vajalikus ning mõistlik raamistik testide arendamiseks.

# 2 Metoodika

Äriinfotehnoloogia erialale toetudes on läbi viidud:

- Küsitlus automaatteste teostava töötajaga praeguse automaattestimise protsessi tundma õppimiseks
- Analüüs erinevate automaattestimise raamistike eelistest ja puudustest e-kaubanduse rakenduste testimisel

## 2.1 Tööriistad

Diagrammide koostamiseks on kasutatud vabavara rakendust Bizagi Modeler. Bizagi Modeleri kasutatakse peamiselt graafiliseks diagrammimiseks ja dokumenteerimiseks. [3] Automaattestide raamistikest on töös mainitud Selenium, Cypress, TestCafe ja Gauge. Raamistikude valiku puhul oli kriteeriumideks, et oleks avatud tarkvaraga ja populaarsed. Avatud tarkvaraga raamistiku valmine on oluline, et vältida klientidele lisakulu tekkimist.

## 2.2 Protsess

Automaattestimise kasulikkuse ja vajalikkuse analüüsimise jaoks on viidud läbi kirjalik intervjuu Opuses töötava testijaga, kes on hetkel ainukene automaatteste arendav testija e-commerce tiimis. Intervjuu sisaldas küsimusi praegusest testimise protsessist nii manuaalse kui automaatse testimise kohta ning ka automaattestide arendamise kohta. Selle intervjuu põhjal saab analüüsida ning teha järeldusi automaattestimise vajalikkuse kohta toetudes varasemale kirjandusele

Sobiva vahendi analüüsimise jaoks on aluseks võetud IEEE artikkel „Testing e-commerce systems: a practical guide“ mille autoriks on W.Lam. Antud artikkel on suunatud projektijuhtidele ja testijatele ning sisaldab kontrollnimekirja e-kaubanduse rakenduste valdkondadest, mida vaja silmas pidada testimisel ning kuidas neid testida. Võttes arvesse

artiklis välja toodud valdkondi, saab uurida millised tööriistad katavad testimiseks efektiivselt kõige rohkem valdkondi.

# 3 Testimine

Opuses on testimise koha pealt kõige olulisemad efektiivne ajakasutus ja kõrge testimise kvaliteet. Ajakasutus on eriti oluline, kuna klientidega lepitakse kokku arenduste ajamahud, ning nende ületamine tähendab kliendile kas tähtaegade ületamist või ületundide võrra suuremat arvet. Kõrge testimise kvaliteet on oluline, kuna Opusel on eesmärk luua pikaajalisi ning püsivaid suhteid oma klientidega ja seetõttu soov pakkuda kliendile alati lahendusi mis vastavad täielikult nende ootustele.

## 3.1 Testimise protsess

Testimise protsessi uurimiseks viidi läbi intervjuu Kristo Loiduga kes on hetkel ainukene testija e-kaubanduse tiimis, kes tegeleb automaatsete testide arendamisega. Küsimused mis testijale esitati:

- Milline näeb välja sellise projekti testimise protsess, millel puuduvad automaattestid?
- Milline näib välja sellise projekti testimise protsess, millele on arendatud automaatsed testid?
- Kuidas näeb välja automaattestide arendamise protsess?

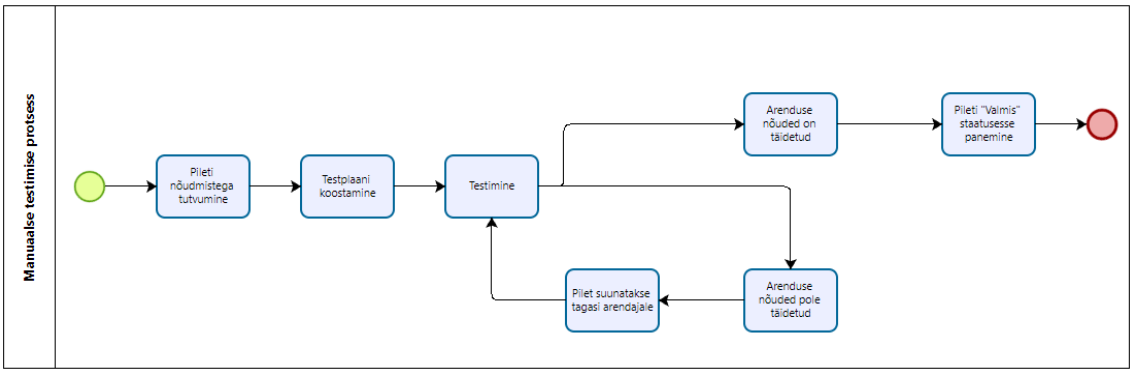
### 3.1.1 Arendusprotsessi tutvustus

Opuses toimub arendus sprintide kaupa. Iga sprinti kohta lepitakse kliendiga kokku millised piletid antud versiooni raames töösse võetakse. Kui versiooni sisu on kinnitatud, võtavad arendajad piletid töösse, ning esimese kontrollina vaatavad koodi üle teised arendajad. Selleks, et pilet jõuaks testimisse, peab vähemalt kaks arendajat kinnitama koodi korrektsuse. Kui see kontroll on läbitud saadetakse pilet testimisse ning testija saab uuendata test-keskkonda ning selle pileti töösse võtta. Enne uue versiooni avaldamist teostakse ka smoke-testid mis kujutab endast rakenduse põhifunktsioonide kontrollimist, et teha kindlaks et mõni uus arendus pole kahjustanud olemasolevaid funktsioone.

### 3.1.2 Manuaalne testimise protsess

Opuses testijana töötav Kristo Loit kirjeldab manuaalse testimise protsessi järgnevalt: „Manuaaltestimine näeb ette kõikide bugiparanduste, uute funktsioonide ja *smoke*-testide tegemist. Kõigepealt kontrollitakse uue versiooni raames uued arendused käsitsi üle. Kui need on korras, siis tuleb teha projektile *smoke*-testid, mis kontrollivad üle poe põhivooga seonduva. Pärast *smoke*-testimist saab teha LIVE uuenduse, kui ei esinenud bugisid.“

Joonis 1 kirjeldab manuaalse testimise protsessi. Arenduspilet, mis sisaldab kliendi poolt soovitud arenduse nõudmisi ning tehnilisi samme selle eesmärgini jõudmiseks, jõuab testimisse kui see on valmis arendatud arendaja poolt, selle koodi on üle vaadanud vähemalt kaks arendajat ning see kood on jõudnud pre-live keskkonda. Manuaalse testimise protsess algab piletis olevate nõudmistega tutvumisega. Seejärel koostab testija testplaani ning alustab seda plaani järgides testimist. Juhul kui arenduse nõudmised pole täidetud suunatakse pilet tagasi arendusse parandamiseks. Peale paranduste lisamist tuleb pilet tagasi testimisse. Juhul kui testimise käigus probleeme ei ilmne, pannakse pilet „valmis“ olekusse, mis tähendab, et arendus on valmis avaldamiseks.




Joonis 1 Manuaalse testimise protsess

### 3.1.3 Automaattestidega projektide testimise protsess

Kristo kirjeldab automaatset testimist järgnevalt: „Automaattestide protsess näeb välja selline, et arendaja või testija on kirjutanud kokku projekti jaoks kogumiku teste, mis kontrollivad projekti põhifunktsionaalsusi (näiteks e-poe puhul oleksid need ostukorvi toodete lisamine/eemaldamine, ostu sooritamine, ostutšeki valideerimine, sisselogimine). Kui need testid on lõpetanud, siis genereeritakse HTML raport, kus on välja toodud:

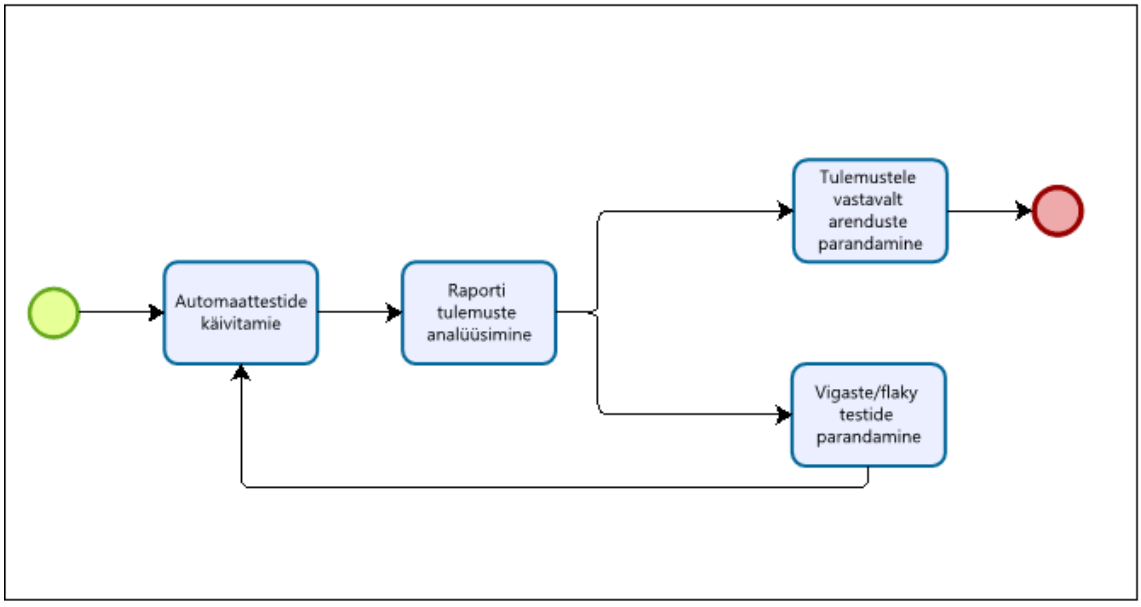
- Millised testid jooksid.

- Mida need testid kontrollisid.
- Kaua testide jooksmine kestis.
- Vead, mis võisid tekkida uute arendustega.
- Vigade olemasolul pilt asukohast, kus test põrus.

Veateade sisaldab veakoodi, mille abil saab kindlaks teha, kas test põrus uue bugi tõttu või on tegu *flaky*/vigase testjuhuga ja *skripti* haldajal/testijal tuleks seda skripti täiendada“

Selle põhjal võib järeldada, et automaattestid annavad märkimisväärse koguse informatsiooni rakenduses tekkivatest vigades. Siiski tuleb arvesse võtta *flaky* ja vigaste testide olemasolu. *Flaky* test kujutab endas testi millel on mittedeterministlikud tulemused. Ehk teisisõnu selline test võib nii edukalt läbi minna kui ka anda veateadet ilma testitavat koodi muutmata. [4]

Joonisel 2 on kirjeldatud automaatse testimise protsessi. Automaatsete testidega testimine algab testide käivitamisest. Seejärel saadakse raport tulemustest. Raporti põhjal saab hinnata tulemusi. Juhul kui raportist selgub, et test sisaldab vigaseid või flaky teste, tulevad need testid tagasi arendusse saata, et need parandada. Juhul kui ei vigaseid teste ei ilmne, saab võtta raportist selgunud tulemused ja nende põhjal teha järeldusi kas midagi on vaja rakenduse koodis parandada.

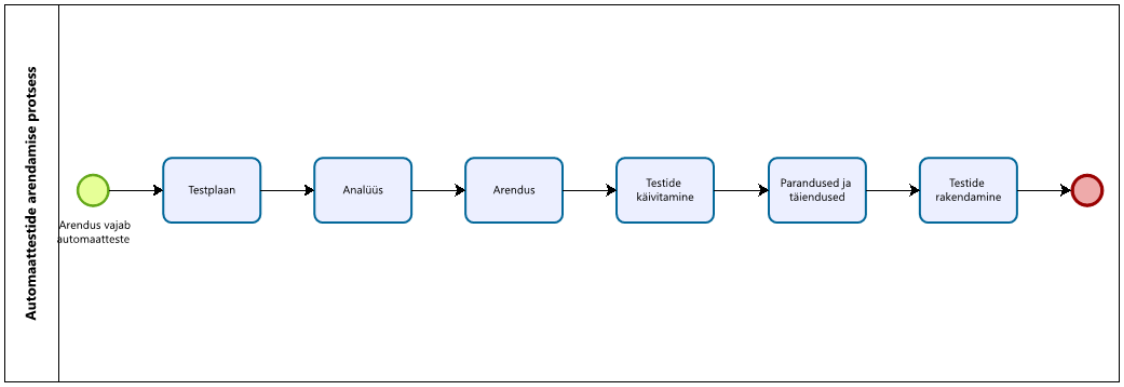



Joonis 2 Automaatse testimise protsess

Automaattestide arendamise tehnilise poole kohta sõnas Kristo: „*Smoke*-testide, ehk põhifunktsionaalsuse kontrolli näitel näeks see välja midagi sellist.

- Testija kaardistab ära kindla funktsionaalsuse test selectorid, mille kaudu skript saab navigeerida e-poe liideses.
    - Kui juhtub, et adekvaatseid selektoreid ei ole, siis ta kaardistab puudulikud selektorid eraldi taskina ja arendaja kirjutab need selektrorid koodi.
- Seejärel võtab testija kätte taski, kus on kirjeldatud, millise flow jaoks on testid mõeldud ja kirjutab selle kohta testid.
- Pärast testide debugimist pannakse kood BitBucketisse
- Projekt ise kasutab testide käitamiseks eraldi Jenkinsi faili, mida kasutatakse:
    - Enne deployd, et kontrollida projekti seisundit enne uuendusi
    - Pärast deployd, et kontrollida projekti staatust pärast uuendusi
- Kui testid avastasid vead, siis luuakse HTML raportid. Seejärel kontrollitakse raporteid ja tehakse kindlaks, kas tegu on bugiga või vigase testiga.“

Joonis 3 kirjeldab automaattestide arendamise protsessi. Automaattestide arendamise protsess algab testplaani koostamisega. Testplaani koostamise käigus hinnatakse nõudmisi mida rakendus peab täitma, ning jagatakse need testitavateks punktides. Järgmiseks sammuks on analüüs, mille käigus täiendatakse testplaani põhjal tehnilised nõudmised automaattestidele. Sellele järgneb testide arendamine. Kui testid on arendatud, on järgmiseks sammuks testide käivitamine, mille järel genereeritakse raport tulemustele. Juhul kui raportis esineb vigu, tuleb hinnata kas tegu on rakenduse koodis oleva probleemiga või vigase testiga. Vigase testi puhul tuleb test tagasi arendusse suunata. Kui viga esineb rakenduse koodis, tuleb luua selle kohta vastav bugi-pilet.




Joonis 3 Automaattestide arendamise protsess

### 3.1.4 Automaattestide vajalikus

Arenduste käsitsi üle vaatamine on kindlasti oluline, aga manuaalne testimine kulutab palju tööjõudu ja on kulukas. Lisaks sellele kipuvad manuaalset testides vead kahe silma vahele jääma ning ühtegi testi pole inimliku faktori tõttu kunagi täpselt sama moodi korrata. Selleks, et peale igat arendust teha manuaalselt kindlaks kas rakendus töötab endiselt nõuetele vastavalt võib kuluda tunde. Selleks, et püüda kinni võimalikult palju vigu ning säästa väärtuslikku aega on mõistlik rakendada automaatteste. [1] Vaadates peale Opuse arendamise ja testimise protsessile oleks mõistlik kombineerida automaatset ja manuaalset testimist. Kuna erinevaid arendusi toodetakse iga sprindi jooksul palju ning nende testimine eeldab tihti visuaalset kontrolli lisaks tehnilisele kontrollile, on manuaalne testimine kindlasti vältimatu. Siiski saaks testimise mahtu oluliselt vähendada asendades näiteks *smoke*-testid käsitsi läbi viimise asemel automaatse testimisega. Sellisel juhul saaks hoida enne igat versiooni uuendust kokku mitmeid tunde, mis kulub nende testide läbi viimisele käsitsi, ning püüda kinni vead mis manuaalsel testimisel märkamata jäid.

## 3.2 Vahendid automaattestide loomiseks

Automaaattestide loomiseks on olemas erinevaid tehnilisi vahendeid. Hetkel Opuses on valitud automaattestide loomiseks Cypress raamistik peamiselt testija varasema kogemuse, oskuste ja arvamuse põhjal.

Opuses töötav testija Kristo Loit sõnas Cypress raamistiku valimise kohta: „Suurimaks põhjuseks osutus kasutaja mugavus, sest Cypressi teste saab jälgida otse veebisirvijas

(seda kutustakse inglise keeles Test Runneriks). Näiteks kui testija on kirjutanud kokku skripti, mis kujutab endast kasutaja sisselogimist, siis Cypressi Test Runneri kaudu saab visuaalse pildi, kuidas skript neid samme läbib. Test Runnerile on ka sisseehitatud funktsionaalsused, mis lihtsustasid: selektorite hankimist skriptide jaoks, testid on Runneris puu kujul, ehk saab käivitada spetsiifilisi testjuhte. Teiseks põhjuseks ma tooks Cypressi dokumentatsiooni tervikluse. Kui kirjutasin oma lõputööd, siis oli dokumentatsioonist väga lihtne aru saada. See aitas õppimise arvelt aega kokku hoida. Kolmandaks põhjuseks ma tooks kogukonna, sest kui raamistiku taga on tervislik ja kogukas kasutajaskond, siis on tagatud see, et on lihtsam saada vastuseid keerulistele küsimustele, mida dokumentatsiooni võibolla ei pruugi rahuldada.“

### 3.2.1 Selenium

Selenium on levinud avatud tarkvaraga testimise raamistik mida kasutatakse automaattestide kirjutamiseks. Selenium pakub tööriistu ja API-sid brauserite automatiseerimiseks, seega on see hea valik veebirakenduste testimiseks. Antud raamistiku kasutus on viimastel aastatel märkimisväärselt populaarsust kogunud, kuna see on paindlik, seda on lihtne kasutada ning see toetab mitmeid erinevaid programmeerimiskeeli (nt Java, Python, Ruby, JavaScript jne). [5] Lisaks sellele saab Seleniumi integreerida ka erinevate raamistikkudega, nagu näiteks TestNG, JUnit ja NUnit tänu millele saavad testijad kasutada neile tuttavaid raamistikke. Selenium lubab ka paralleelset testimist, läbi mille saab mitmeid teste korraga käivitada selle kaudu säästa aega. Seleniumi saab integreerida ka teiste testimise tööriistadega nagu näiteks Jenkins, Docker ja Sauce Labs mille abil testija saab luua enda vajadustele sobiva keskkonna. [6]

### 3.2.2 Cypress

Cypress, mis on hetkel Opuses kasutusel olev raamistik, on populaarne avatud tarkvaraga *end-to-end* testimise tööriist modernsetele veebirakendustele. See põhineb JavaScriptil ja on kasutajasõbralik. Cypressi puhul on eeliseks võime käivitada teste otse brauseris, mis omakorda kiirendab testide jooksutamist ning parandamist. Cypressi üks kasulikumaid omadusi on automaatne ootamine, mis tähendab, et Cypress teab millal leht või element on valmis suhtlemiseks. See omadus eemaldab vajaduse lisada manuaalseid *timeout*-e, tehes testide kirjutamise mugavamaks ning lollikindlamaks. Teine väga kasulik omadus on *time travel debugging* mis lubab testide arendajal käia sammhaaval läbi testide samme ning näha kuidas need käituda, tehes vigade leidmise lihtsamaks. Kolmandaks pakub

Cypress ka kehva või puuduliku internetiühenduse matkimist, mis aitab ennustada kuidas rakendus käitub olukorras kus päringud võivad läbi kukkuda. Cypressi *dashboard* pakub reaalajas testide tulemusi ja videoid, aidates seeläbi probleemide tuvastamist kiirendada. [7]

### 3.2.3 TestCafe

TestCafe on moderne ja kasutajasõbralik testimise raamistik, mille eesmärgiks on eemaldada levinud ebamugavusi automaattestimise protsessist. TestCafe-ga saab teste arendada kasutades tavalist JavaScripti või TypeScripti. TestCafe on ehitatud Promise API-le mis võimaldab kirjutada asünkroonseid teste kasutades modernse JavaScripti süntaksit. Näiteks teeb see lihtsamaks elemendi ilmumise ootamise enne sellega suhtlemist. Raamistik võimaldab testimist erinevates brauserites ja seadmetes, andes seeläbi parema ülevaate rakenduse kvaliteedist. Lisaks sellele võimaldab TestCafe jooksutada teste paralleelselt erinevates brauserites ja seadmetes, säästes selle läbi aega. TestCafe üheks eriliseks omaduseks on selle võima matkida võimalikult palju rakenduse kasutust viisil, kuidas kasutaks seda inimene. Raaamistiku abil saab elementida simuleerida sündmusi nagu klikkimine, kirjutamine ning lehekülje kerimine. TestCafe-d saab integreerida erinevate CI/CD süsteemidega nagu Jenkins, CircleCI ja TravisCI, mille läbi saab automaatset testimist integreerida juba tarkvara arendamise käiku. [8]

### 3.2.4 Gauge

Gauge on avatud lähtekoodiga automaattestimise raamistik mis toetab erinevad programmeerimiskeeli nagu näiteks Java, Python ja C#, lubades testide arendajal valida arenduseks keel mis suhestub käige paremini testitava rakendusega. Gauge on lihtne kasutada ning paindlik. Teste saab käivitada ka käsurealt. Seda raamistikku kasutades kirjutatakse testid lihtsa tekstina ning testid organiseeritakse spetsifikatsioonide failidesse mis sisaldavad endas ühte või rohkemaid teste ehk spetsifikatsioone. Spetsifikatsioonid on nõudmiste kogum mida kasutatakse süsteemi või rakenduse käitumise defineerimiseks. Spetsifikatsioonid ise jaotatakse stsenaariumideks mis on täpsustatud nõuded sellest, kuidas rakendus mingis kindlas olukorras käituma peaks. Ka see raamistik võimaldab paralleelset testimist ehk mitmeid erinevaid teste samaaegselt käivitada. Gauge võimaldab koostada tulemustest raporti ning toetab selleks erinevaid vorminguid nagu HTML, JSON ja XML. [9]

## 3.3 Sobiva vahendi valik

Sobiva vahendi valimiseks vaadatakse millised on üldised e-kaubamajade testimise punktid, ning millise vahendiga saaks neist kõige rohkem kaetud.

### 3.3.1 Brauseriga ühilduvus

Lehe kättesaadavus on kliendile oluline kliendile ning erinevate brauseritega ühilduvus on selle tõttu väga tähtis ja seetõttu on oluline ka selle testimisele aega panustada. Kuna kõik brauserid pole võrdsed, võib juhtuda, et üks rakendus käitub erinevates brauserites erinevalt. [10]

Kõik neli käesoleva töö käigus uuritavat raamistikku sobivad brauseriga ühilduvuse testimiseks. Selenium pakub kõige laiemat seadmete ja brauserite valikut testimiseks, kaasaarvatud mobiilsete seadmete testimiseks. Seleniumi miinuseks on selle keerukus, mille tõttu võib võtta raamistiku õppimine ning arendamine kauem aega. [6] Cypressi sisaldab juba sisse ehitatud tuge testide käivitamiseks erinevates brauserites ning võtab brauserite erinevusi arvesse, kuid ei toeta nii palju erinevaid mobiilseid seadmeid kui Selenium. [7] Ka TestCafe sisaldab sisse ehitatud tuge testide käivitamiseks erinevates brauserites, aga pole nii paindlik kui Selenium ja Cypress. Samuti toetab TestCafe vähem mobiilseid seadmeid kui Selenium. [8] Gauge toetab ka erinevates brauserites testimist, kuid kuna Gauge on loodud pigem tarkvara rakenduste testimiseks siis võib selle kasutamine e-kaubanduse rakendustel võtta kauem aega ning rohkem hooldamist sest tehnilist tuge päris brauserites testimiseks napib. Ka Gauge-i puhul on mobiilsete seadmete valik testimiseks väiksem kui Seleniumil. [9]

### 3.3.2 Lehekülje kuvamine

Lehekülgede kuvamine on järjekordselt üks väga oluline osa, mis vajab testimist. Olulised punktid, mida lehekülgede kuvamise testimisel meeles pidada tuleb on õige lehekülje kuvamise testimine kui ka testida olukordi, kus vajalikke andmeid ei saada andmebaasist kätte, ning vaadata kas selliseid olukordi lahendatakse korrektselt. [10]

Ka lehekülje kuvamist on võimalik testida kõigi nelja raamistikuga. Seleniumi miinuseks on jätkuvalt raamistiku keerukus mille tõttu võib arendamisele ja hooldamisele kuluv aeg pikeneda. [6] Cypressi kasuks räägib tema lihtne kasutatavus. [7] Ka TestCafe puhul on

lihtne kasutatavus poolt argumendiks. [8] Gauge kasutamisel võib jätkuvalt segavaks faktoriks osutuda vähene tugi brauserites testimisele. [9]

### 3.3.3 Kasutatavus

Kasutatavust on keerulisem automaattestimisega hinnata, aga saab testida näiteks otsingut, kontrollides otsingutulemusi otsingusse sisestatud tulemuste vastu. Lisaks saab testida lehtede laadimise aega, kuna kui leht laeb liiga kaua on see kasutajale ebamugav. [10]

Kõik neli raamistikku võimaldavad kasutatavuse testimist. Jätkuvalt on Gauge ja Seleniumi puhul miinuseks hooldamise ja arendamise aeg. Kuna Cypress simuleerib kasutaja interaktsiooni, võimaldab see anda ka paremat ülevaadet olukordadest mis võivad tekkida päris kasutajal. [7] TestCafe haldab automaatselt dünaamilist sisu mis toetab kasutatavuse testimist. [9]

### 3.3.4 Tehingud

Tehingute töötlemine on keskne element e-kaubamajade töös. Tehingute puhul on oluline kontrollida tehingu terviklikkust. Näiteks testida tehingu tegemisel tehtavaid andmebaasi päringuid ning veenduda, et süsteem käitub erinevate stsenaariumite puhul korrektselt ja baasi jõuavad korrektsed andmed. Lisaks sellele oleks hea testida süsteemi vastupidavust suurele kasutajate arvule. [10]

Kõik neli raamistikku toetavad tehingute testimist. Seleniumi keerukust tuleb tehingute testimisel pigem eeliseks kui nõrkuseks, kuna tehingute testimine vajab keerukaid teste. [6] Cypress on küll mugav raamistik mida kasutada ning omab väga võimsat API-t aga cypress võib jääda hätta keerulisemate tehingute töövoogude testimisel. [7] TestCafe ei pruugi olla nii paindlik kui teised raamistikud, ja seetõttu võib kasvatada testide arendamisele kulutatavat aega. [8] Gauge miinuseks on eelpool mainitud vähene tugi päris brauserites testimisel, mis võib tulla takistuseks kui on vaja testida kuidas tehingute tegemine reaalse kasutaja jaoks toimib. [9]

### 3.3.5 Ostlemine, tellimuste vormistamine ja soetamine

Ostlemine, tellimuse vormistamine ja soetamine on põhiline funktsioon e-commerce rakendustel ja selle funktsionaalne testimine võib hõlmata kuskil 25 kuni 50 protsenti testimisele kulutatavast ressursist. Silmas tuleks pidada asju nagu toodete ostukorvi

lisamine, ostukorvi enda funktsionaalsused ja tellimuse vormistamine mis sisaldab kliendilt vajalike andmete kättesaamist ning tehingu sooritamist. Lisaks sellele on enamikel e-poodidel ka võimalus jälgida tellimuse teekonda, mille puhul on oluline testida kliendile kuvatava info korrektsust. [10]

Ostlemise, tellimuse vormistamise ja soetamise puhul tuleb arvesse võtta, et nende testimine on keerukam, kuna töövoog on pikem, tuleb kontrollida palju informatsiooni ning selle protsessi käigus tehakse mitmeid erinevaid päringuid. Seetõttu on nende funktsioonide testimisel Selenium tugev kanditaat, kuna Seleniumiga on võimalik arendada keerukaid teste mis käituvad täpselt vastavalt vajadusele. [6] Cypressile räägib kasuks palju sisse ehitatud funktsionaalsusi päris brauserites testimiseks mis teevad erinevate töövoogude testimise lihtsamaks. Siiski võib Cypress jääda hätta olukorraga kus toimub palju asünkroonseid sündmusi või päringuid. [7] Seevastu TestCafe üheks eeliseks on just see, et see suudab hallata dünaamilist sisu ja asünkroonseid protsesse. Seetõttu on see hea tööriist töövoogude testimiseks mille käigus toimub palju erinevaid sündmusi. [8] Gauge puhul jällegi puudulik tugi selliste olukordade testimisele, mistõttu võib sellega nende funktsionaalsuste testimine olla ajakulukas ja keeruline. [9]

### 3.3.6 Tõlkimine ning keele vahetamine

Enamik Eesti e-kaubamajasid on saadaval lisaks eesti keelsele versioonile ka inglise ning vene keeles. Kuna süsteemitekstid on reeglina inglise keelsed, vajavad eesti ning venekeelsed lehed tõlkeid, ning nende rakendumist ja korrektust. Lisaks tuleks testida anomaaliate puudumist rakenduse keele vahetamisel. [10]

Tõlgete testimine automaatselt on keeruline, kuna eeldab korrektset sisendit mille vastu tõlkeid kontrollida ning seetõttu pole mõistlik. Keele vahetamise testimise saab katta teiste funktsionaalsuste testimisega, tehes kindlaks, et peale keele vahetamist ei tekiks anomaaliaid rakenduses.

### 3.3.7 Operatiivsed äriprotseduurid

Operatiivsete äriprotseduuride alla kuulub näiteks klienditoega ühenduse võtmiseks mõeldud vorm, suhtlusrobotid jne. Selliste funktsioonide puhul on oluline testida, kas lõppkasutaja edastatud informatsioon jõuab ka õigesse kohta kohale. [10]

Operatiivsete äriprotseduuride testimine on väga sarnane peatükis 3.3.3 mainitud kasutatavuse testimisega.

### 3.3.8 Süsteemi integratsioon

Opuse poolt halatavad leheküljed on enamasti integreeritud kolmandate osapoolte loodud laosüsteemidega, kust päritakse infot toodete enda ning tootekoguste kohta. Lisaks sellele saadetakse laosüsteemidesse info tehtud tellimuste kohta, et värskendada tootekoguseid. Äärmiselt oluline on selliseid integratsioone põhjalikult testida, kuna selline info mõjutab raamatupidamist ning toodete kättesaadavust. [10]

Kolmandate osapoolte haldavate rakendustega integratsiooni automaatselt testimine on keeruline, kuna nendele rakendustele pole enamasti testijal ligipääsu.

### 3.3.9 Jõudlus

Lõppkasutajani jõudev jõudlus on tähtis kasutajamugavuse aspektist kuid vähene jõudlus võib rakenduse kasutamise sootuks võimatuks teha. Jõudluse testimise puhul on oluline leida kõige nõrgemad lülid, kus süsteemi komponent võib kokku joosta või töötamise lõpetada. Lisaks sellele tuleb testida ka rakenduse vastupidavust suuremale külastajate arvule, mis on eriti oluline enne suuremaid kampaaniaid. [10]

Selenium toetab tööriistu nagu JMeter ja Gatling mille abil jõudlust testida. Miinuseks jällegi tehniline keerukus, kuna eeldab ka eelpool mainitud tööriistade tundmist lisaks sellele, et Seleniumiga testide arendamine on juba ise tehniliselt keeruline. [6] Cypress omab sisse ehitatud *network throttling* funktsiooni mille abil saab tuvastada ja lahendada jõudlusega seotud probleeme. [7] TestCafe võimaldab paralleelset testimist mille abil saab testida jõudlust olukorras kus leheküljel on suurem koormus. Samuti saab TestCafe abil luua kohandatud brauseri profiile spetsiifiliste olukordade testimiseks. Miinusena tuleks välja tuua, et kuna raamistikul on piiratud tugi *network throttling*-ule mõjutab see testide tulemuse täpsuse hindamist. [8] Sarnaselt Seleniumile toetab Gauge tööriistu nagu JMeter ja Gatling mis aitavad jõudluse testimisel. Miinusena sarnaselt TestCafe-le ei toetata *network throttling*-ut. [9]

### 3.3.10 Autentimine

Enamikes e-kaubamajades on kasutajal võimalik luua endale kasutajakonto. Konto loomise puhul tuleb silmaks pidada nii kasutaja loomise ning sisselogimise protsessi kui

ka süsteemi turvalisust, et kaitsta klienti küber-rünnakute ning pettuste eest. Testida tuleks asju nagu korrektse ning ebakorrektse infoga sisselogimine, parooli unustamine ning vahetamine, ning konto seadete muutmine. [10]

Selenium toetab testimiseks erinevaid autentimise tüüpe nagu näiteks *basic*, *digest* ja NTLM. Seleniumit saab integreerida ka tööriistadega nagu OWASP ZAP et tuvastada levinud turvaauke. Probleemiks on see, et need tööriistad vajavad lisateadmisi mille omandamine võtab aega. [6] Cypress pakub mugavalt kasutatavaid käske mille abil testida kasutaja autentimise töövoogu ning tuge ka erinevate levinud turvaaukude testimisel. Miinuseks on vähene tugi kahe sammulise autentimise testimiseks. [7] Ka TestCafe pakub automaatset autentimise testimise tuge, aga ka vähe tuge kahe sammulise autentimise testimiseks. [8] Gauge-i saab sarnaselt Seleniumile integreerida erinevate kolmanda osapoolte tööriistadega, aga omab vähe tuge turvaaukude ning kahe sammulise autentimise testimisele. [9]

### 3.3.11 Võrdlustabel

Tabel 1 kirjeldab raamistike tugevust e-kaubanduse rakenduste testimise oluliste punktide puhul. Raamistikule on antut hinnang „tugev“ juhul kui antud raamistik sobib väga hästi antud punkti testimiseks. Hinnangut „keskmine“ on kasutatud juhul, kus raamistik on sobilik antud punkti testimiseks, kuid esineb märkimisväärseid miinuseid või takistusi. Hinnangut „nõrk“ on kasutatud juhul kui raamistik ei sobi antud punkti testimiseks.

Tabel 1 Raamistkude võrdlustabel

| **Punkt** | **Selenium** | **Cypress** | **TestCafe** | **Gauge** |
|---|---|---|---|---|
| Brauseriga ühilduvus | Tugev | Tugev | Keskmine | Keskmine |
| Lehekülje kuvamine | Tugev | Tugev | Tugev | Keskmine |
| Kasutatavus | Keskmine | Tugev | Tugev | Keskmine |
| Tehingud | Tugev | Keskmine | Keskmine | Nõrk |

| | | | | |
|---|---|---|---|---|
| Ostlemine, tellimuse vormistamine ja soetamine | Tugev | Keskmine | Keskmine | Nõrk |
| Operatiivsed äriprotseduurid | Keskmine | Tugev | Tugev | Keskmine |
| Jõudlus | Tugev | Tugev | Keskmine | Keskmine |
| Autentimine | Tugev | Tugev | Tugev | Keskmine |

Võrdlustabelist (Tabel 1) selgub, et Gauge sai vaid „nõrk“ või „keskmine“ hindeid ja selle tõttu pole hea raamistik e-kaubanduse rakenduste testimiseks. Kuigi Gauge-i on võimalik kasutada e-poodide testimiseks ei sobi antud raamistik kuna sellel on vähene tugi brauseris testimisele ning seetõttu oleks parem lahendus tarkvarale mis pole veebipõhine. TestCafe rahuldab hinnanguga „tugev“ või „keskmine“ kõiki e-poe testimises olulisi punkte. Selle raamistikuga on võimalik edukalt testida e-kaubanduse rakendusi. Peamiseks miinuseks sellel raamistikul osutus oht suuremale arenduse ajakulule kuna see raamistik pole nii paindlik kui teised töös välja toodud raamistikud, aga tarkvara arendamise teenust müüvale ettevõttele on efektiivne ajakulu oluline.

Cypress ja Selenium said võrdlustabeli põhjal võrdsed hinnangud. Seleniumi peamiseks tugevuseks on võimalus arendada detailseid ja keerukaid teste mis vastavad täpselt rakenduse nõuetele. Seleniumi miinuseks on suurem ajakulu kuna keerukate ja detailsemate testide arendamiseks kulub rohkem aega. Lisaks sellele eeldab Selenium testijalt rohkem teadmisi testide arendamisest, kuna sisse ehitatud funktsioone on vähe ning arendamine on keerulisem. Cypressi eelis on just selle kasutamismugavus. Raamistiku kasutama õppimiseks kulub lühem aeg ning Cypress sisaldab palju sisse ehitatud funktsioone mis kiirendavad arendamise protsessi. Kuid kuna arendatud testid pole nii keerukad, siis võib tekkida olukord kus keerukamaid lahendusi rakenduses on raske testida. Kuna efektiivne ajakasutus on tarkvara arenduse ettevõttes oluline, on selle analüüsi põhjal Cypress testimisraamistikuna mõistlik valik.

# 4 Tulemused

Käesolevas töös analüüsiti automaattestimise vajalikust e-kaubanduse rakendustele ning sobiva vahendi valikut nende arendamiseks. Tulemuste põhjal saab järeldada, kas automaattestid on olulised e-poodide arendamise protsessis. Lisaks sellele on uuritud erinevad populaarseid avatud tarkvaraga automaattestimise raamistikke ja nende sobivust ettevõtte Opus Online OÜ-s arendatavate e-poodide testimiseks.

## 4.1 Automaattestide vajalikus

Analüüsides manuaalse ja automaatse testimise protsessi ettevõttes Opus Online OÜ ilmnes, et enne igat uut versiooni avaldamist teostatakse käsitsi *smoke*-teste. See on ajakulukas ja võimatu on korrata teste iga kord täpselt sama moodi, et vältida ebatäpsuse tõttu tekkivaid tulemuste erinevusi. Manuaalset testimist uute arenduste puhul täielikult eemaldada ei saa, kuna see on oluline rakenduse disaini ning kasutajasõbralikuse testimiseks. Automaatsete testidega oleks mõistlik asendada *smoke*-testimise protsess, mille käigus kontrollitakse enne uue versiooni avaldamist põhifunktsioonide korrektsust. Sellisel viisil saaks kindlustada, et testid läbitakse iga kord sama moodi ning saadakse täpsemad tulemused. Lisaks sellele hoitakse aega kokku, kuna automaattestide käivitamine nõuab vähem aega kui manuaalse testimise protsess.

## 4.2 Sobilik vahend

Sobiva vahendi valiku analüüsimiseks oli valitud neli raamistikku: Selenium, Cypress, TestCafe ja Gauge. Raamistiku valimisel toetuti olulistele punktidele e-poodide testimises ning hinnati raamistike sobivust nende punktide testimiseks. Pärast tulemuste võrdlemist selgus, et kõige paremini sobiv raamistik on Cypress.

# 5 Järeldused

Ettevõttel on mõistlik jätkata automaattestide arendamisega ning laiendada seda praktikat kõikidele projektidele manuaalse testimise toetamiseks. Analüüsi tulemusena selgub, et automaattestimine säästaks kliendile väärtuslikku aega. Aja sööstmiseks on aga vajalik, et automaattestid on arendatud viisil, et manuaalselt peaks arendusi võimalikult vähe üle testima. Kindlasti peaks asendama versiooniuuenduste avalikustamise eelset testimist automaattestimisega. Manuaalset testimist võiks kasutada vaid kasutajasõbralikuse ja disaini testimisel.

Analüüsi käigus selgub et sobiva raamistikuna võib kasutada selleks jätkuvalt Cypress raamistikku. Cypress raamistik katab kõik Opuses arendatud e-poodide testimise vajadused. Siiski võimalusel tuleks kaaluda töötajatele Seleniumi koolitusi, kuna analüüsi käigus selgub, et Selenium saab hakkama keerukamate töövoogude lahendamisega. Näiteks võib Cypress jääda hätta ostlemise, tellimuse vormistamise ja soetamise voogude testimisel kuna tegu on keerukate töövoogudega. Seleniumiga testide arendamise jaoks vajalike oskuste omandamine võib olla küll suurema ajakuluga kui Cypressil, aga selle raamistikuga saab arendada täpselt rakenduse vajadustele kohanduvaid keerukaid teste.

# 6 Kokkuvõte

Käesolev lõputöö käsitleb aktuaalset probleemi tarkvara arendamise protsessis ettevõtte Opus Online OÜ näitel. Automaattestide arendamine nõuab ajalist ja rahalist ressurssi mis tarkvara arendust teenusena pakkuval ettevõttel tuleb kliendile põhjendada. Kui automaatsete testide vajalikus on põhjendatud on oluline õige raamistiku valimine. Kui valida raamistik mis ei sobi projekti vajadustega, suureneb ka rahaline kulu.

Analüüsi raames uuriti manuaalse ja automaatse testimise protsessi ning tehti selle põhjal järeldused automaattestimise vajalikkusest. Raamistiku valikul hinnati olulisi punkte mida tuleb e-poode testides silmas pidada ning hinnati nelja raamistiku sobivust nende punktide testimiseks.

Ettevõte plaanib analüüsist saadud positiivsete tulemuste põhjal rakendada automaatset testimist ka projektidele, kus seda hetkel ei kasutata.

# Kasutatud kirjandus


[1] J. Bosas, „Automated Testing Importance and Impact,“ [Võrgumaterjal]. Available: https://ieeexplore.ieee.org/abstract/document/8532522. [Kasutatud 7 aprill 2023].

[2] „Magento 2 arendus ja e-poodide loomine,“ Opus Online OÜ, [Võrgumaterjal]. Available: https://www.opusonline.co/et/magento-e-poodide-tegemine/. [Kasutatud 5 aprill 2023].

[3] „Bizagi Modeler,“ [Võrgumaterjal]. Available: https://www.bizagi.com/en/platform/modeler. [Kasutatud 12 aprill 2023].

[4] Qingzhou Luo, Farah Hariri, Lamyaa Eloussi, Darko Marinov, „An empirical analysis of flaky tests,“ [Võrgumaterjal]. Available: https://dl.acm.org/doi/abs/10.1145/2635868.2635920. [Kasutatud 29 aprill 2023].

[5] M. K. A. Holmes, „Automating functional tests using Selenium,“ [Võrgumaterjal]. Available: https://ieeexplore.ieee.org/abstract/document/1667589. [Kasutatud 8 aprill 2023].

[6] „The Selenium Browser Automation Project,“ [Võrgumaterjal]. Available: https://www.selenium.dev/documentation/. [Kasutatud 9 aprill 2023].

[7] „Cypress,“ [Võrgumaterjal]. Available: https://docs.cypress.io/. [Kasutatud 12 aprill 2023].

[8] „Why TestCafe?,“ [Võrgumaterjal]. Available: https://testcafe.io/documentation/402631/guides/overview/why-testcafe. [Kasutatud 13 aprill 2023].

[9] „Gauge Documentation,“ [Võrgumaterjal]. Available: https://docs.gauge.org/?os=macos&language=javascript&ide=vscode. [Kasutatud 15 aprill 2023].

[10] W. Lam, „Testing e-commerce systems: a practical guide,“ [Võrgumaterjal]. Available: https://ieeexplore.ieee.org/abstract/document/918215. [Kasutatud 2 aprill 2023].

# Lisa 1 – Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks[1]

Mina, Helina Sosi

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „E-kaubanduse rakenduste automaattestimise analüüs Opus Online OÜ näitel“, mille juhendaja on Viljam Puusep
   1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
   1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.
3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

17.05.2023

1 Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal vastavalt üliõpilase taotlusele lõputööle juurdepääsupiirangu kehtestamiseks, mis on allkirjastatud teaduskonna dekaani poolt, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil. Kui lõputöö on loonud kaks või enam isikut oma ühise loomingulise tegevusega ning lõputöö kaas- või ühisautor(id) ei ole andnud lõputööd kaitsvale üliõpilasele kindlaksmääratud tähtajaks nõusolekut lõputöö reprodutseerimiseks ja avalikustamiseks vastavalt lihtlitsentsi punktidele 1.1. ja 1.2, siis lihtlitsents nimetatud tähtaja jooksul ei kehti.