TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Marilyn Võsu 192003IAAM

# Töö- ja testimisprotsesside parendamine kliendiprojektis ettevõtte Proekspert AS näitel

Magistritöö

Juhendaja: Nadežda Furs
MBA

Ketlin Saksakulm
MSc

Tallinn 2021

# 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: Marilyn Võsu

20.05.2021

# Annotatsioon

Magistritöö eesmärk on koostada põhjendatud muudatusettepanekud töö- ja testimisprotsessidele ettevõtte Proekspert AS kliendiprojektis. Eesmärk on luua testimise strateegia info- ja telekommunikatsioonivaldkonda kuuluva kliendi projektile, mida oleks võimalik rakendada ka teistes sarnastest tarkvaraarendusprojektides.

Magistritöös tehakse manuaaltestide, kasutajaliidese automaattestide ja testimisprotsesside analüüs.

Töö käigus luuakse automaattestide esialgne arhitektuur ning kirjutatakse näidisena ümber osa eelnevalt kirjutatud teste. Töö tulemus on TO-BE vaade testimisprotsessidest, sh. manuaaltestide kirjutamise ja haldamise vaade ning automaattestide kirjutamise ja haldamise vaade.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 61 leheküljel, 6 peatükki, 20 joonist, 4 tabelit.

# Abstract

Improvement of Work and Testing Processes Based on a Client Project of Proekspert AS

The purpose of the thesis is to create justified change propositions for work and testing processes based on a client project of Proekspert AS. The goal is to create a testing strategy for an info- and telecommunications client project, which could be reused in other similar software development projects.

The thesis consists of analysis of manual tests, user interface automated tests and analysis of test processes.

As a result, the first basis of the automated tests is created and a part of the automated tests are refactored as an example. The outcome is a TO-BE draft of test processes, including the writing and maintaining of the manual and automated tests.

The thesis is in Estonian language and contains 61 pages of text, 6 chapters, 20 figures, 4 tables.

# Lühendite ja mõistete sõnastik

| | |
|---|---|
| BDD | *Behavior Driven Development* on tehnika, milles arendajad, testijad ja äripoole esindajad analüüsivad koos tarkvara nõudeid ja koostavad neid ühise keele abil [1] |
| BPMN | *Business Process Model and Notation* on äriprotsesside ülesmärkimise süsteem [2] |
| DevOps | Praktikate kogum, mille eesmärk on automatiseerida ja integreerida tarkvaraarenduse ning tarkvaraarendustiimide protsesse, et tiimid saaksid kiiremalt ja usaldusväärsemalt tarkvara arendada, testida ning avaldada [3] |
| Javascript | Programmeerimiskeel |
| Jira | Tarkvaraarendusprojektide haldustarkvara |
| Jira kasutajalugu | PROJ-märgisega kasutajalugu, mis on loodud ärinõuetele ja vajadustele toetudes |
| Jira testilugu | TEST-märgisega detailne kirjeldus testi sammudest, sisenditest ja oodatud tulemusest, mis on loodud kasutusloole toetudes |
| Kasutajaliidese automaattest | Automatiseeritud testilugu, mida saab vastava tööriistaga automaatselt käivitada |
| Pidev integratsioon | Tarkvara arendusmetoodika, kus arendajad lisavad pidevalt arendatud koodiosi ühisesse repositooriumisse [4] |
| Regressioontestimine | Testimisviis, mille eesmärk on veenduda rakenduse funktsioonide endises toimimises peale uute funktsionaalsuste arendamist |
| Repositoorium | Enamasti koodi, failide või testide hoidla |
| *Scrum master* | Agiilse tarkvaratiimi töökorralduse ekspert |
| TestCafe | Testimisraamistik |

| | |
|---|---|
| Testilugu | Komplekt testisisenditest, sammudest ja oodatavast tulemusest |
| Typescript | Programmeerimiskeel |

# Sisukord

# Jooniste loetelu

# Tabelite loetelu

# Sissejuhatus

Agiilses tarkvaraarenduses on tarkvara testimine selle kvaliteedi hindamiseks ning tagamiseks väga oluline. Iga arendustegevusega käib kaasas ka vastav testimistegevus. Olenemata tarkvaraarenduse elutsüklist, on oluline, et testimistegevused algavad selle elutsükli võimalikult varases staadiumis, et kinni pidada varajase testimise reeglist. [5]

Tarkvaraarenduses on väga olulised eesmärgid ka ennustatavuse parendamine ja produktiivsuse optimiseerimine. [6] Ennustatavuse alusel saab hinnata arendatava toote hinnangulist valmimisaega. Eriti tähtis on ennustatavus tellija seisukohast, mida täpsem on arendatava tarkvararakenduse ajahinnang, seda paremini saab tellija ka enda äriprotsesse planeerida. [7] [8]

Testija ehk kvaliteediinseneri põhilised tööülesanded on tarkvara testimine ning kvaliteedi tagamine. [9] Samas ei ole agiilne testimine pelgalt testide läbiviimine ja tulemuste kontrollimine, vaid see kujutab endast veel palju teisi tegevusi, näiteks nii testide planeerimist, disainimist, kirjutamist kui ka analüüsimist. Samuti kuulub tegevuste hulka testimistulemuste ja -progressi raporteerimine ning testitava süsteemi kvaliteedi hindamine. Tarkvara testimine ja selle süstematiseeritud korraldamine on tarkvara arenduse koha pealt väga oluline, kuna aitab veenduda arendusprojekti kvaliteedis. [5]

Magistritöö on läbi viidud ühe Proeksperdi info- ja telekommunikatsioonivaldkonda kuuluva kliendi projektis. Proekspert on disaini- ja tarkvaraarendusettevõte, mis on asutatud 1993. aastal Eestis. 31.12.2020 seisuga töötab Proeksperdis 155 inimest [10], kellest umbes ⅔ on tarkvarainsenerid. Proekspert on hinnatud tööandja ning on saanud mitmeid auhindu, üks hilisemaid nende seast on näiteks Fontese võrdse palga auhind [11].

Proeksperdil on mitmeid kliente, kellele teostatakse erinevaid tarkvara- ning riistvaralahendusi. Kliendid on mitmetest ärivaldkondadest, näiteks telekommunikatsioon, tööstusautomaatika, pangandus. Endiste ja praeguste klientide hulka kuuluvad nii mitmed eestimaised ettevõtted, näiteks CV Online, Luminor, Skype, Eesti Gaas, Ridango, Tallinna Kaubamaja, Tallink, kui ka suured ja rahvusvahelised ettevõtted, näiteks Microsoft, Danfoss, Alliance Laundry Systems, Telia Company, Elisa,

Swedbank, SEB, ESA (European Space Agency). Proeksperdi põhilised ärivaldkonnad on ärianalüütika, andmeteadus, teenusedisain ja  teenuste arendus.

Ettevõte on tuntud oma ainulaadse juhtimisstiili poolest ning on nö. juhtidevaba, mis tähendab, et otseseid ülemusi ega alluvaid firmas pole, firmas jälgitakse holokraatia juhtimisstiili põhimõtteid. Töötajatelt oodatakse seda, et nad vastutavad nii enda kui ka tiimi heaolu eest ning adresseerivad parandamist vajavaid asjaolusid kui need tekivad. Igaüks on nö iseenda ülemus ning vastutus lasub tiimiliikmete ja kliendi ees. Ettevõttesiseselt saavad töötajad liikuda nii erinevate projektide kui ka rollide vahel, et õppida erinevaid oskusi ning viia enda teadmisi ja kogemusi järgmistesse projektidesse.

Proeksperdi eesmärk on olla oma klientidele hinnatud äriarenduspartner, pakkudes ettevõtetele tipptasemel digitaliseeritud lahendusi. Klientide äriarenduse edendamise ja protsesside digitaliseerimise juures on oluline kliendikeskne lähenemine, protsesside süstematiseeritus, tõhusus ning automatiseeritus. Kvaliteetsete täisteenusete ja toodete pakkumisel on testimise roll kvaliteedi tagamisel väga oluline.

Magistritöö eesmärk on uurida ühe info- ja telekommunikatsiooni valdkonda kuuluva kliendi arendusprojekti üldisi töö- ning testimisprotsesse. Seejuures on eesmärk luua testimisstrateegia, mida saaks rakendada ka teistes sarnastes tarkvaraarendusprojektides. Autor on kaasatud ka kliendiülesesse tarkvaraprojektide edendamise töögruppi, mille eesmärk on tiimide töö- ja testimisprotsesse parendada. Töögrupp edendab tarkvaratiimides testimisalaseid teadmisi ja praktikaid.

Magistritöö on jagatud 6 peatükiks.

Esimeses peatükis püstitab autor magistritöö probleemi, määratleb skoobi ja kirjeldab enda rolli magistritöös.

Teises peatükis annab autor ülevaate valitud meetoditest, kirjeldab ja analüüsib kliendiprojekti töö- ja testimisprotsesse.

Kolmandas peatükis analüüsib autor parimate praktikate meetodeid ning uurib tiimis valitud tööriistade kasutusjuhendeid, mille alusel saab koostada ka konkreetse projekti erisusi ja nõudeid silmas pidades tööprotsesside plaani ja testimisstrateegia.

Neljandas peatükis keskendub autor kliendiprojekti töö- ja testimisprotsesside parendamisele, sealhulgas loob autor konkreetse kliendiprojekti huve silmas pidades ning parimatest praktikatest lähtudes testimisplaani- ning strateegia.

Viiendas peatükis teeb autor projekti uurimistulemuste kohta järeldused, riskianalüüsi ning annab ettepanekud projekti töö- ja testimisprotsesside parendamiseks.

Kuuendas peatükis kirjeldab autor projekti edasisi planeeritud ja soovitatud tegevusi.

# 1 Taust ja probleemipüstitus

Selles peatükis antakse ülevaade hetkeolukorrast kliendiprojektis ja ettevõttes, samuti kvaliteediinseneri rollist käesolevas projektis. Autor kirjeldab ka praegust testimise ja üleüldist tiimi toimimise protsessi ning sõnastab probleemi, mida magistritöös käsitletakse.

## 1.1 Probleemipüsitus ja magistritöö eesmärk

Analüüsitavas kliendiprojektis on mitmed töö- ja testimisprotsessid ning praktikad läbi mõtlemata ja süstematiseerimata, samuti puudub hea ülevaade nii manuaal- kui ka automaattestidest. Testimisprotsessides on lünki ning puudub süsteemsus ja ülevaade hetkeolukorrast. Ka mitmed üldised tööprotsessid tiimis on kokku leppimata ning seetõttu võib iga kord probleemi lahendamisel olla see lisa ajakulu, kuna iga uue probleemi ilmnemisel tuleb lahendust välja mõtlema hakata.

Samuti kulub manuaalsele regressioontestimisele palju aega ning see on kvaliteediinseneridele tihti igav ja väga ajakulukas ülesanne. Iga funktsionaalsuse lisamisega kasvab iga kord ka terve rakenduse regressioontestimise aeg. [9] Manuaalse regressioontestimise käigus ei pruugi kvaliteediinsenerid märgata uusi vigu, kuna fookus võib pika ja üksluise testimise käigus kaduda ja tihti ei märgata vigu seal, kus enne pole neid esinenud. Kuna käesolevas tarkvaraarendusprojektis kasvab funktsionaalsuste arv väga kiiresti, on vaja, et rakendusel oleksid peale ühiktestide ja integratsioonitestide ka usaldusväärsed kasutajaliidese automaattestid, mis jookseksid automaatselt mõistliku regulaarsusega ning mille raportid oleksid tervele tiimile kergesti kättesaadavad.

Proeksperdi ning tema ärikliendi eesmärkide hulka kuulub protsesside automatiseerimine, protsesside ühtlustamine projektide lõikes ning loodud lahenduste taaskasutamine. Väga oluline osa tarkvaralahenduse loomisel on selle ajahinnangu ennustatavus. Mida täpsemad on valmiva töö ajahinnangud, seda täpsemini saab ka äriklient enda äri planeerida. Seega on magistritöö lahenduse juures väga oluline ka neid aspekte silmas pidada.

Magistritöö üldine eesmärk on analüüsida töö- ja testimisprotsesse kliendiprojektis, pakkuda tiimile põhjendatud ettepanekud ning seeläbi parendada töö- ja testimisprotsesse. Eesmärk on luua kliendiprojektile kogum testimispraktikaid, mida saaks näidisena kasutada ka teistes sarnastes tarkvaraarendusprojektides. Testimispraktikaid peaks saama kasutada analüüsitava arendusprojektiga sarnastes tiimides, kus on vähemalt 1 kvaliteediinsener või kohandatult ka nendes, kus konkreetne kvaliteediinseneri roll tiimis puudub. Protsesside parendamisel on eesmärk parendada ka tarkvaraarenduse ennustatavust.

Töö konkreetsed eesmärgid on analüüsida praeguseid manuaalseid testilugusid ning sorteerida need testiplaani, analüüsida ja leida parenduskohti testimisprotsessides ning teha vastavad parendusettepanekud. Eesmärk on luua automaattestide kirjutamiseks ja haldamiseks lihtsasti hallatav arhitektuur ning alustada praeguste automaattestide refaktoreerimise ehk funktsionaalsust muutmata koodi ümber kirjutamisega. [12] Lisaks on magistritöö eesmärk anda ettepanekud, kuidas automaattestid regulaarselt ja automaatselt käivitada mõne pideva integratsiooni vahendiga.

## 1.2 Magistritöö skoop

Magistritöö skoopi kuulub kliendiprojekti tööprotsesside, sh. testimisprotsesside analüüs.

Skoopi kuulub:

- Tiimi tööprotsesside analüüs
- Testimisprotsesside analüüs, sh. testimise planeerimise analüüs
- Testilugude analüüs
- Testilugude sorteerimine ja lisamine testiplaani
- Kasutajaliidese automaattestide analüüs
- Autori ettepanekud automaattestide refaktoreerimiseks ja edasiseks kirjutamiseks
- Esialgne kasutajaliidese automaattestide refaktoreerimine

Skoopi ei kuulu:

- Manuaaltestide haldamise tööriistade võrdlus ja valik
- Kasutajaliidese automaattestide raamistike ja tööriistade analüüs ning valik
- Kõigi olemasolevate kasutajaliidese automaattestide refaktoreerimine

## 1.3 Autori roll

Magistritöö autor töötab Proeksperdis 2019. aastast kvaliteediinsenerina. Kvaliteediinseneri igapäevased tööülesanded kliendiprojektis on analüüsida koos tooteomanikuga arendusse jõudvaid ülesandeid ning aidata defineerida ärinõuete teostamiseks vaja minevaid konkreetseid arendustöid, anda plaanitavatele arendustöödele aja- ja mahuhinnanguid testimiseks, kirjutada kasutuslugude alusel testilugusid ning hallata testiplaani. Lisaks sellele veab autor eest projekti, kus parendatakse töö- ja testimisprotsesse ning kokku lepitult on lisanduvad tööülesanded veel teise kvaliteediinseneri poolt kirjutatud testilugude üle vaatamine, analüüs ning kontroll. Samuti kuuluvad tööülesannete hulka regressioontestimiseks funktsionaalsuste kaupa testiplaanide koostamine ja regressioontestimine, automaattestide analüüs, haldus ja kirjutamine, automaattestide käivitamine ning automaattestide raportite analüüs.

Antud projektis on klient info- ja telekommunikatsiooni valdkonnast. Magistritöö alustamise hetkel on projektis peale autori veel 1 kvaliteediinsener, 7 arendajat, 2 disainerit, tooteomanik, *scrum master* ja 2 *DevOps* spetsialisti.

Töö esitamise hetkel on tiimi lisandunud üks ärianalüütik ning tiimist lahkunud teine kvaliteediinsener, üks arendaja ja üks *DevOps* spetsialist.

# 2 Töö- ja testimisprotsesside kirjeldus ning analüüs

Antud peatükis tutvustab autor analüüsiks valitud meetodeid, tehakse huvitatud osapoolte analüüs, antakse ülevaade kliendiprojekti tööprotsessidest, praktikatest ning kasutusel olevatest tööriistadest. Samuti antakse ülevaade testimisprotsessidest ja tuuakse välja probleemkohad.

## 2.1 Analüüsimetoodika valik

Autor on lähtunud parimate praktikate kogumisel rahvusvahelise testimiskvalifikatsioone väljaandva juhatuse ehk The International Software Testing Qualifications Board'i materjalidest ning tunnustatud agiilse testimise kirjandusest ja juhenditest, samuti agiilsete tiimide kasutatavate tööriistade juhenditest. Autor lähtub parendusettepanekute koostamisel agiilse arenduse ja testimise põhimõtetest ning kohandab põhimõtteid ja rakendatavaid praktikaid just konkreetse kliendiprojekti erisusi arvestades. Äriprotsesse analüüsitakse ja visualiseeritakse BPMNi (*Business Process Model and Notation*) abil. Automaattestimise lahenduse analüüsimiseks ja kavandamiseks kasutab autor testimisraamistiku TestCafe POM (*Page object model)* automaattestimise arhitektuuri. [13] [14]

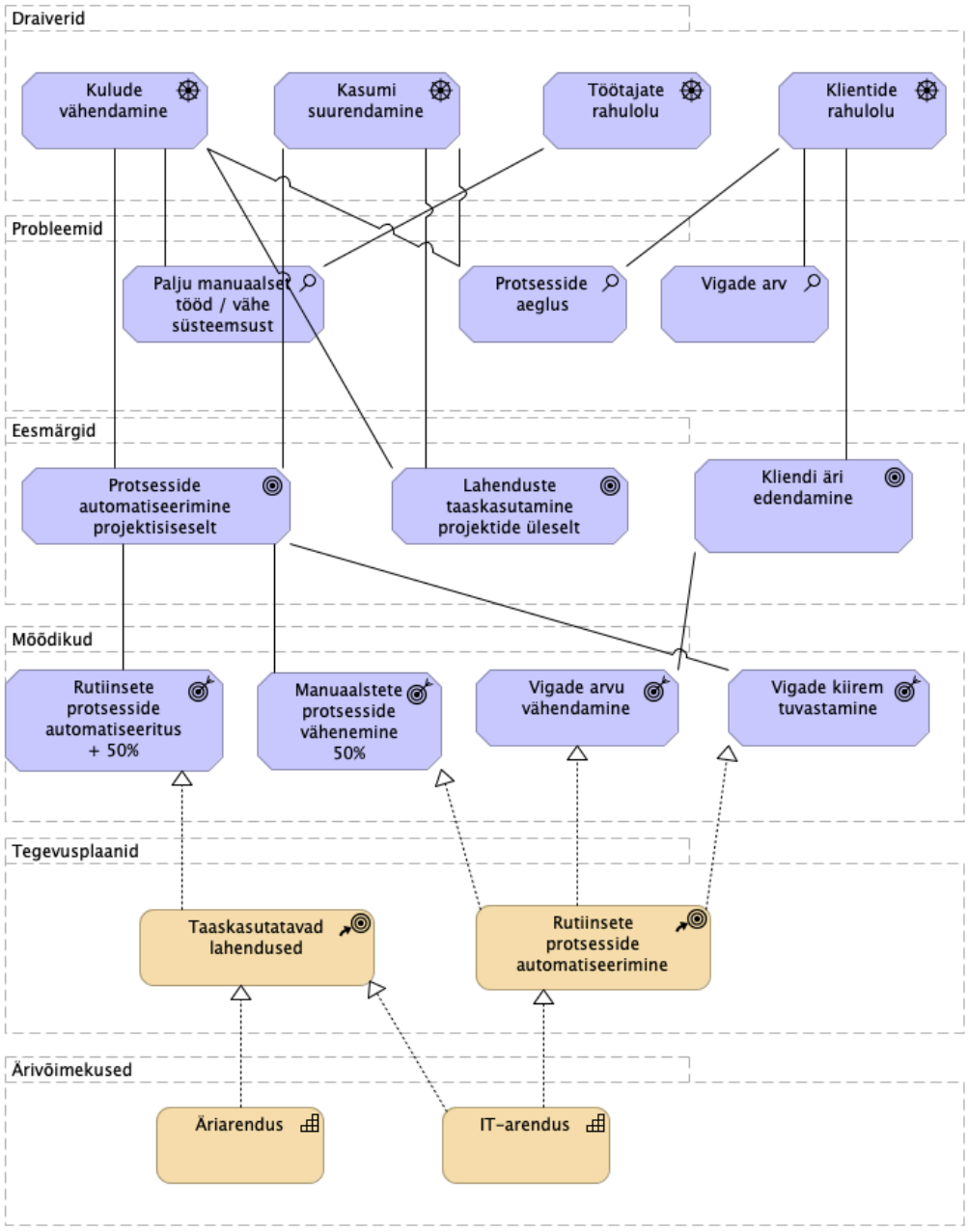
## 2.2 Ärilised eesmärgid

Ettevõtte äriliste eesmärkide ja strateegia edendamiseks on autor analüüsinud kliendiprojekti eesmärkmudeli abil, et kaardistada mõõdikud, mille alusel protsesside analüüsi ja parendamise edukust hinnata. Ettevõtte põhilised eesmärgid on protsesside automatiseerimine, lahenduste taaskasutamine. Samuti on eesmärk edendada kliendi äri. Parendusplaani kokkupanemiseks on hea järgida järgmisi etappe:

- määrata parendusplaani osade vastutavad isikud
- selgitada välja põhilised eesmärgid ja probleemid
- grupeerida probleemid eesmärkide kaupa
- veenduda, et eesmärgid ja probleemid on selgesti mõistetavad ja veenvad

- panna paika eesmärkide prioriteedid
- panna paika eesmärkide mõõdikud [15]

Joonis 1 illustreerib kliendiprojekti eesmärkmudelit:




Joonis 1. Ettevõtte ja kliendiprojekti eesmärkmudel (allikas: autori koostatud)

Joonis 1 ilmneb, et ettevõtte põhilised mõõdikud on rutiinsete protsesside automatiseeritus ning manuaalsete protsesside vähenemine. Samuti on mõõdikuteks

vigade arvu vähendamine ja vigade kiirem tuvastamine. Ettevõtte eesmärgid ja mõõdikud on aluseks antud töö mõõdikute kavandamisel.

## 2.3 Huvitatud osapooled

Kliendiprojekti töö- ja testimisprotsesside muutmisel on oluline võtta arvesse kõiki huvitatud osapooli. Huvitatud osapoolte kaardistamiseks on autor konsulteerinud tiimiliikmete ning Proeksperdi teiste kvaliteediinseneridega. Eri osapoolte vastutuse analüüsimiseks kasutatakse ka RACI maatriksit, mille alusel saab konkreetse initsiatiivi või projektiga seotud isikute vastutust määrata. R märgib vastutavat isikut, A aruandvat isikut, C konsulteeritavat isikut, I informeeritavat isikut. [2] Testimisprotsesside parendamisest huvitatud osapoolte kaardistamiseks viis autor läbi ka RACI maatriksi analüüsi, mille tulemused on alljärgnevas tabelis:

Tabel 1. Huvitatud osapooled (allikas: autori koostatud)

| **Huvitatud osapool** | **Huvi** | **RACI** |
|---|---|---|
| Kvaliteediinsener(id) tiimis | Testiprotsesside läbipaistvus ja organiseeritus, töö optimeerimine | R, A, C |
| Arendaja(d) | Ülevaade automatiseeritud testidest ning võimalus neid peale arendustöö lõppu käivitada, kindlustunne koodimuudatuste ja uute arenduste tegemisel | R, C |
| Disainer(id) | Lõppkasutaja vaate ja kogemuse parem mõistmine, testimispraktikate ja -võtete õppimine, et aidata vajadusel testimisega seotud ülesandeid kanda | R, C |
| Tooteomanik | Testiraportite kättesaadavus, projekti kvaliteedi läbipaistvus ja hindamine | I |
| Lõppkasutaja | Kvaliteetse iseteenindusportaali kasutamine | I |
| Tellija ehk klient | Toote või teenuse müügi suurendamine, kulude vähendamine | I |
| Teised kliendiprojektide või Proeksperdi tiimid | Testistrateegia praktikate kasutamine | C |

Tabelist ilmneb, et kvaliteediinseneril tiimis on nii vastutav, aruandev kui ka konsulteeritav roll. Samuti on vastutavad ja konsulteeritavad isikud tiimis töötavad arendajad.

## 2.4 SIPOC diagramm

Protsesside kaardistamiseks on autor loonud SIPOC diagrammi [16], kus toob välja iseteeninduskeskkonna arendamisega seotud osapooled. Tabel 2 kujutab SIPOC diagrammi:

Tabel 2. SIPOC diagramm (allikas: autori koostatud)

| **Tarnijad** | **Sisendid** | **Protsess** | **Väljundid** | **Kliendid** |
|---|---|---|---|---|
| Arendaja<br>Tooteomanik<br>Disainer<br>Kvaliteediinsener<br>Tellija<br>Klienditugi | Uue funktsionaalsuse vajadus<br>Ärinõude muutmise vajadus<br>Viga süsteemis<br>Disainimuudatus<br>Testide katvuse vajadus | Ärinõude saabumine<br>⌄<br>Tööülesannete kirjeldamine ja analüüsimine<br>⌄<br>Tööülesannetele ajahinnangu andmine<br>⌄<br>Testide kirjutamine<br>⌄<br>Kasutuslugude arendamine<br>⌄<br>Koodiülevaade<br>⌄<br>Testimine<br>⌄<br>Testide täiendamine<br>⌄<br>Vearaportite koostamine<br>⌄<br>Ärinõue rahuldatud | Disaini prototüüp<br>Funktsionaalsuse arendus<br>Tehtud tööde aruanne<br>Vea parandamine süsteemis<br>Testide raport | Äritellija<br>Lõppkasutaja<br>Klienditugi |

SIPOC diagrammi alusel saab järeldada, et protsessi nii sisendid kui ka väljundid on klienditugi, kelle manuaalset tööd proovib iseteeninduspordaali arendamine vähendada. Klienditoelt on võimalik saada sisendit, kuidas mingid konkreetsed protsessid peaksid töötama. Tabelist paistab, et protsesside alla kuulub mitu testimisega seotud tegevust, näiteks testide kirjutamine, testimine, vearaportite koostamine, mis on kõik kvaliteedi tagamisel väga olulised.

## 2.5 SWOT analüüs

Kliendiprojekti praeguse seisu hindamiseks on autor koostanud SWOT analüüsi, mis aitab välja selgitada, milline on parim muudatuste strateegia, mida analüüsitava projekti jaoks kasutada. Samuti on analüüsiga võimalik välja selgitada, millised on projekti tugevused, mille abil saab projekti edendada, nõrkused, mida saab parendada, samuti võimalused ja ohud, mida peab projekti analüüsides ja parendades silmas pidama. [2]
Tabel 3 illustreerib kliendiprojekti SWOT analüüsi:

Tabel 3. SWOT analüüs (allikas: autori koostatud)

| **Tugevused** | **Nõrkused** |
|---|---|
| • Sügavate arendusteadmiste ja -oskustega arendajad<br>• Palju manuaalteste kasutuslugude põhjal defineeritud<br>• Automaattestimise raamistik olemas ning osa automaatteste kirjutatud<br>• Ühiktestide hea katvus | • Tiimis vähe testimisalaseid teadmisi, sh. sügavaid automaattestimise teadmisi ning kogemusi<br>• Automaattestide käivitamine ei ole regulaarne ega automaatne<br>• Automaattestidega pole piisavalt hea testide katvus ning puudub hea ülevaade nende katvusest<br>• Puudulik dokumentatsioon protsessidest<br>• Testimisele kuluvat aega ei arvestada ajahinnangusse<br>• Testiplaan on raskesti hallatav, puudub ülevaade testidest<br>• Testide muutmise korral on palju lisatööd<br>• Ühe kvaliteediinseneri tiimist lahkumisel ei suuda teine üksi õigeaegselt regressioontestimist teha |

| Võimalused | Ohud |
| --- | --- |
| • Erinevate tehnoloogiate ja tööriistade efektiivne kasutamise võimalus<br>• Manuaaltestide automatiseerimise võimalus<br>• Protsesside efektiivsem juhtimine | • Töömahu suurenemisest tulenev ressursipuudus |

Tabelist ilmneb, et kliendiprojekti põhilised nõrkused on puudulik dokumentatsioon ja süsteemsus, samuti testimisele kuluva aja mitte arvestamist tarkvaraarenduse ajaarvestusse.

## 2.6 Kvalitatiivsed ja kvantitatiivsed mõõdikud, mille alusel projekti edukust mõõdetakse

Testimisel on mõõdikute kasutamine on testimise eesmärkide ja efektiivsuse mõõtmiseks oluline. Mõõdikud on hea viis testimistegevuste jälgimiseks ja monitoorimiseks. [17] Mõõdikud põhinevad kvaliteediinseneri igapäevatöö tegevustel ning nende loomisel on arvestatud ka ettevõtte suuremaid eesmärke. Ettevõtte suuremad eesmärgid on protsesside automatiseerimine projektisiseselt, lahenduste taaskasutamine projektide üleselt ning kliendi äri edendamine. Mõõdikud toetavad ettevõtte eesmärke, kuna nende alusel saab hinnata töö tegemiseks kuluvat aega. Mõõdikud, mille alusel projekti edukust mõõdetakse, on järgnevad:

- Ühe funktsionaalsuse manuaaltestide muutmise keskmine aeg
- Ühe funktsionaalsuse automaattestide muutmise keskmine aeg
- Kattuvate manuaaltestide arv kogu testide arvust
- Automaattestide katvuse ülevaade
- Regressioontestiplaani kokkupanekule kuluv aeg
- Kogu manuaalse testiplaani läbimiseks kuluv aeg

- Automaattestide käivitamiseks kuluv aeg
- Testiraportite kättesaadavuse lihtsus ja raportite täpsus ning ajakohasus

## 2.7 Tööriistade ja -protsesside ülevaade

Antud projektis, kus arendatakse info- ja telekommunikatsioonivaldkonda kuuluvale kliendile iseteeninduskeskkonda, kasutatakse mitmeid tööriistu ning programmeerimiskeeli, mille hulka kuuluvad TypeScript, JavaScript, TestCafe, Gherkin. Projekti raames päritakse andmeid mitmetest suurtest andmebaasidest ja süsteemidest.

Arenduses kasutatakse *behavior-driven development* (BDD) praktikaid, mis on osa *test-driven development* (TDD) praktikast, mis tähendab, et tarkvaraarenduses keskendutakse äripoole, arendajate, kvaliteediinseneride koostööle. [1] Analüüsitavas kliendiprojektis on kasutuslugude pealkirjad enamasti ülesehitusega *„As a (user)“, „I want (to do)“* ja *„so that (I can)“,* kus iga lause algusele järgneb kasutajat või rakenduse käitumist ning nõudeid kirjeldav lause. Samuti kasutatakse arendustööde stsenaariumite, kasutus- ja testilugude kirjeldamiseks inimloetavaid lauseid, täpsemalt *Gherkin'*i keelt, kus kasutajatoimingud või sammud on kirjeldatud lausetega, mis algavad tihti märksõnadega *Given, When, Then*. Märksõnade abil kirjeldatakse, millises olukorras peaks teatud osapool midagi süsteemis teha saama. Üks kasutuslugu võib sisaldada mitmeid kasutusstsenaariume.

Projektis kasutatakse agiilse arenduse põhimõtteid, täpsemalt *scrum* metoodikat. Projekti haldamiseks kasutatakse peamiselt Jira tarkvara, dokumentatsiooni jaoks Confluence'i tarkvara. Iteratsioonid on 2-nädalased ning enne iga arendustsükli algust tehakse selle planeerimine. Nädal peale arendustsükli algust tehakse hetkeseisu ülevaade ja arendustsükli lõpus tehakse terve tsükli ülevaade. Arendustsükli lõpus viiakse läbi retrospektiiv ehk tsüklit kokkuvõttev koosolek, kus vaadatakse 2 nädala jooksul õnnestunud asjad ja sündmused üle ning kaardistatakse need teemad, kus võiks parendustöid teha. Samuti toimuvad tiimis igahommikused *stand-up*'id ehk *daily*'d, kus räägitakse, mida eile tehti, mida on täna plaanis teha ning millised on töös esinevad takistused. Tihti tehakse peale hommikust koosolekut väiksemate gruppidega jätkukoosolek, kus arutletakse töös tekkinud küsimuste üle ja lahendatakse koos tehnilisi probleeme.

## 2.8 *Three amigos* sessioon

Koostöös tooteomaniku, teise kvaliteediinseneri ja 1-2 arendajaga tehakse aeg-ajalt *Three amigos* [18] sessioone, kus vaadatakse üle analüüsifaasis olevad kasutuslood ja ärinõuded ning valmistatakse ette arendustööd. Ühel kasutuslool võib olla mitmeid stsenaariume, mis kirjeldatakse *Given, When, Then* ehk BDD-stiilis.

Kasutuslugude kirjutamisel lähtutakse ärinõuetest ja -vajadustest, seega on nende kirja panemisel oluline jälgida, et need ka täidaks etteantud eesmärke. Väga oluline osa on just ärinõuete mõistmine ja analüüsimine, et kasutuslood saaksid võimalikult korrektsed ning oleksid üheselt arusaadavad. Samuti on kasutuslugude kirjutamisel oluline ka see, et arenduse poole pealt oleks arusaadav, kuidas ärinõudetele vastavat lahendust luua. Testimise poole pealt peab uurima, mis võib käesolevate kasutuslugude ja stsenaariumite korral veel juhtuda või kuidas kasutaja võiks antud rakendust alternatiivselt kasutada. Seepärast vaadatakse kasutuslood üle korraga mitmete erinevate rollide poolt, et analüüsida, kas kirja on pandud kõik vajalik nii arendustöö alustamiseks kui ka testimise seisukohast, et mõista, mis selle loo testimise juurde kuulub ja mida silmas peaks pidama.

BDD-sessioonidel hindab arendaja vastava kasutusloo arendamiseks kuluva aja, kuid testimise aega hinnangusse ei arvestata. See aga tähendab, et sprindi planeerimisel arvestatakse vaid arendajate töötundide ning kasutuslugude arendamiseks kuluva ajaga, kuid välja jääb testimisele kuluv tööaeg. **Probleem on selles, et nii on tihti kasutuslugude arendamiseks ennustatav aeg üsna kõikuv ning ebatäpne.**

## 2.9 Arenduslugude analüüs ja testimine

Arendusloo testimisse jõudmiseks on vaja eelnevalt arendaja(te)l funktsionaalsus valmis arendada ning seejärel see ülevaatusesse saata. Koodiülevaatuse ehk *code review* käigus annavad arendajad üksteisele tagasisidet ja soovitusi, kuidas nemad antud arendusülesannet lahendaksid ning kontrollivad, kas arendatud kood töötab nii, nagu oodatud. Kui arendatud kasutuslugu on teiste poolt üle vaadatud ja kontrollitud, võib selle anda tööliinis järgmistele ehk kvaliteediinseneridele analüüsimiseks ja testimiseks.

Projekti raames arendatav iseteeninduskeskkond peab töötama nii veebi- kui ka mobiilivaates. Veebivaates peab keskkond hästi toimima vähemalt veebilehitsejate Chrome, Firefox ja Edge hilisemates versioonides, samuti on oluline testimine

väiksemate ekraanide, näiteks enamlevinumate mobiiliseadmete ja tahvelarvutite peal. Samuti on vajalik testida rakendust eri keeltes, et tõlgete vahetumisel jääks disain sarnaseks ning igas keeles toimiks rakendus ühtemoodi.

Uue funktsionaalsuse testimine on tihti üks kvaliteediinseneri põnevamaid tööülesandeid, sest seal on palju aspekte, mida tuleb jälgida ning testimisel saab kasutada oma loovust. Sageli testib arendaja oma loodud koodiosa just selle pilguga, et see töötaks, seevastu kvaliteediinsener mõtleb pigem nii, kuidas on võimalik arendatud funktsionaalsust lõhkuda ning sellest vigu leida ja millised on rakenduse alternatiivsed kasutuslood.

Samuti saab arenduslugu testides ka disaini uuesti analüüsida ning uurida, kas selles oleks parenduskohti. Uute funktsionaalsuste ärinõutelele vastamisele lisaks on vaja tähelepanu pöörata ka kasutusmugavusele ning samuti mõelda testimise käigus lõppkasutajate peale, kes iseteeninduspportaali kasutama hakkavad. Vajadusel saab disainerite ja ka teiste tiimiliikmetega disainilahenduste üle arutleda ning anda enda soovitusi ja ideid, mida disainerid ka tihtipeale arvesse võtavad. Disainerid on ärinõuete alusel loonud prototüübi, kuid tihti ilmneb kas arenduse või testimise käigus, et algul välja mõeldud disaini oleks vaja parandada või muuta. Seega on kvaliteediinseneri ja disaineri koostöö väga oluline, et lõppkasutajateni jõuaks võimalikult kasutajamugav lahendus.

## 2.10 Testilood ja testiplaanid

Testide haldamiseks on kliendiprojektis kasutusel Jira lisarakendus TestFLO, mille abil saab kirjutada teste, neid repositooriumis hallata ning luua regressioontestiplaane. Antud tööriist on kliendi poolt valitud ning antud töös lähtutakse selle kasutamise jätkamisest.

Hetkel on tiimis praktika, et iga kasutuslooga peavad kaasas käima testlood, mida on võimalik hiljem regressioontestimises testiplaani lisada ning ükshaaval testid üle käia. Tiimis on tööülesanded jagatud järgmiselt: üks kvaliteediinsener kirjutab kasutuslugudele manuaalteste ning arendajad kirjutavad enda arendatud ülesandele ka ühikteste ja manuaaltestidest eraldiseisvaid kasutajaliidese automaatteste. **Probleem on, et sellisel viisil tekib aga kaks eraldi testide komplekti, mis pole omavahel seotud.**

Teste võib kirjutada kasutuslugudele nii enne kui peale selle loo arendusse jõudmist, see oleneb teatud hetke tööülesannete mahust. Kui kvaliteediinseneridel on piisavalt aega, siis kirjutatakse algsed testid kasutuslugudele juba enne nende arendusse jõudmist, kuid

hiljem, kui kasutuslugu on testimiseks valmis, tuleb testid uuesti üle kontrollida ning jälgida, kas midagi on kasutusloo disainis või funktsionaalsuses muutunud. Samuti on vaja üle vaadata, kas testi on vaja lisada täpsemad testandmed, kuna alati ei pruugi funktsionaalsus kõigil kasutajatel ühtemoodi toimida.

Kuna ühel kasutuslool või arendusülesandel on enamasti ka mitu stsenaariumit, tuleb ka ühe kasutusloo kohta kirjutada tihti mitmeid testilugusid. Testilugu kirjutatakse Gherkini keeles ning teste hallatakse Jiras. Testid seotakse ühe konkreetse kasutuslooga, et oleks näha, millised ülesanded ja testlood omavahel seotud on ning kas üks konkreetne kasutuslugu on piisavalt testidega kaetud. Sellise praktika juures aga ei analüüsita eelnevalt kirjutatud testilugusid ning ei seota uusi kasutuslugusid juba olemasolevate testidega**. Probleem seisneb selles, et nii vananevad olemasolevad testilood ning need ei kirjelda enam oodatavat rakenduse käitumist. Seega puudub hea ülevaade, millised testid projektis juba eksisteerivad ning kas oleks vaja testilugusid muuta või parandada.** Tavaliselt hallatakse agiilsetes tiimides testiplaane igapäevaste jooksvate parandustega ning tihti võib just testiplaan olla kõige adekvaatsem rakenduse nõudeid kirjeldav dokument. [5]

Testiloo pealkirjas on kasutusloo number, millele see kirjutatakse, ning see algab enamasti fraasiga „*Verify that*“ või „*Validate that*“, näiteks „PROJ-1234: *Validate that user can see the invoice details page*“, kus „PROJ“ viitab Jira projektile ja „1234“ kasutusloo numbrile. Mõnel testil on pealkirja lõpus lisaks fraas, mis viitab süsteemile, mida testitakse, näiteks „*in self service portal*“. **Probleem seisneb selles, et testilugude pealkirjad on ebavajalikult pikad ja mittevajalike fraasidega. Kui vaadata testilugusid listivaates, on testilugude pealkirjade lugemine aeganõudev ning sealjuures vajamineva testloo leidmine võtab samuti aega. Samuti viitab üks testilugu vaid ühele kasutusloole, mis tegelikkuses ei ole tõsi.**

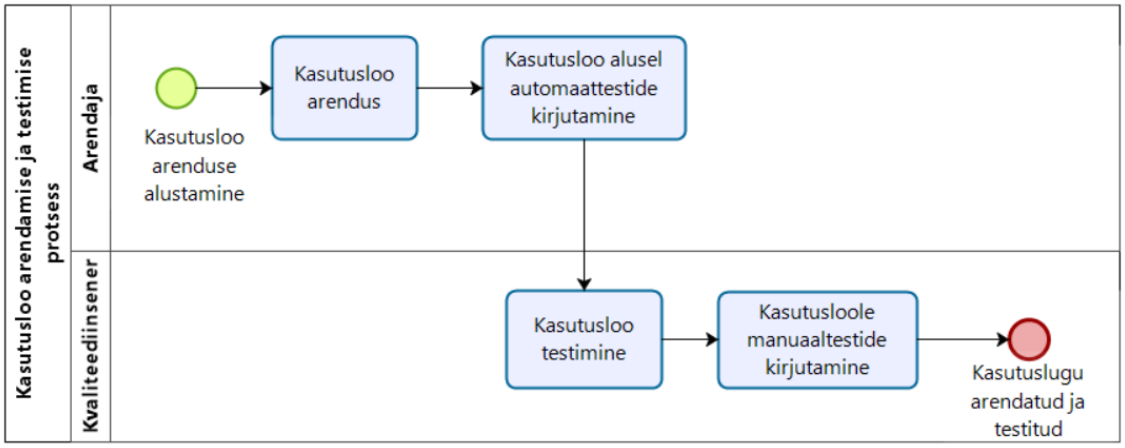
Testiplaanide loomiseks on eelnevalt tiimis olev kvaliteediinsener Jira testilugusid klooninud, et neid lisada uude testiplaani. Seega tekivad ühest testloost mitmed duplikaadid ning sellega ka uued testilugude identifikaatorid ehk järjekorranumbrid. Samuti lisatakse testilugude pealkirja automaatselt sõna „CLONE“. **Probleem seisneb selles, et testilugude läbimisel ei ole head ülevaadet, millisel testimise hetkel mingi funktsionaalsus on ärinõuetele vastav ning millal esineb selles vigu, seega ei ole testilugude kloonimise korral näha põhilise testiloo testimise ajalugu.**

Nagu eelnevalt mainitud, on kõik manuaaltestid seotud vaid ühe konkreetse arendusloogа ning testid kirjutatakse ühe arendusloo kaupa. Teste ei ole võimalik hästi sorteerida ja grupeerida, neid on võimalik leida vaid arenduslugude alt või testloo nime järgi üldisest kaustast. Funktsionaalsuste kaupa ei ole võimalik testilugusid leida või filtreerida.

## 2.11 Automaattestide kirjutamine ja haldamine

Automaattestide kirjutamisega tegelevad vaid arendajad ning üldiselt automaattestid eraldi ülevaatust ei läbi, kuna need kirjutatakse iga kord täpselt kasutuslugude järgi. Tiimis juba eelnevalt olnud kvaliteediinsener automaatteste ei ole kirjutanud, kuna selleks pole tal aega. **Probleem seisneb selles, et puudub täielik ülevaade, millised testid on automatiseeritud ning millised mitte, samuti pole võimalik kindel olla, et automaattestid testivad just seda, kuidas antud ülesanne peaks ärinõudeid täitma.**

Joonis 2 illustreerib analüüsi tegemise hetkeseisu kasutusloo arendamise ja testimise protsessist:




Joonis 2. AS-IS vaade kasutusloo arendamise ja testimise protsessist (allikas: autori koostatud)

Samuti on segane, kes, millal ja kas üldse automaatteste käivitab. Tiimis on reegel, et enne iga arendusülesande üleslaadimist peavad automaattestid olema samuti kirjutatud ning ka toimima. Samas on manuaalselt iga kord automaattestide käivitamine üsna tülikas protsess ning arendajad lähtuvad harva reeglist neid enne iga uue koodimuudatuse üleslaadimist käivitada. Samuti pole selge, kes vastutab testide parandamise või vigade raporteerimise eest kui mõni test ebaõnnestub. **Kuna manuaalselt enda arvutis käivitades ei analüüsita testiraporteid sügavuti ning nende tulemusi ei edastata**

**ülejäänud tiimile, puudub tiimis läbipaistvus ning piisav ülevaade, millal viimati teste käivitati ning mis seisus rakendus täpselt on.**

Esmane kasutajaliidese automaattestide analüüs on tehtud nii ülesehituse ja sõnastuse kohta, meetodite kirjelduse kui ka näiteks testandmete haldamise kohta. Analüüsi alustamise hetkel ehk seisuga 2020. aasta sügis on kokku kirjutatud 117 kasutajaliidese testi, millest nende käivitamisel 61 õnnestuvad ning ülejäänud 56 ei õnnestu, seega umbes pooled testid ei läbi käivitamist edukalt. 117 automaattesti jooksevad keskmiselt 1,5 tundi.

Esmase ebaõnnestunud testide analüüsi käigus on selgunud, et peamised põhjused ebaõnnestumisel on see, et testandmed või funktsionaalsus rakenduses on muutunud, kuid teste ja testisamme pole uuendatud. Samuti on ebaõnnestumise põhjus ka selles, et rakenduses esineb vigu. Ebaõnnestunud testide manuaalsel korduvtestimisel tuleb sageli välja, et rakendus ja testitav funktsionaalsus töötab ootuspäraselt ning automaattest on lihtsalt juhuslikult ebaõnnestunud. See aga muudab testiraportite tulemused ebausaldusväärseks. **Probleem seisneb selles, et testide käivitamine on ajakulukas protsess ning testitulemused ei anna rakendusest adekvaatselt pilti. Samuti on probleem, et testid ebaõnnestuvad vahel juhuslikult.**

Automaattestide analüüsi käigus on selgunud, et kogemuse puudumise tõttu pole tiimis väga ette mõeldud, kuidas peaks teste tulevikus haldama kui testkasutajate õigused, iseteeninduspordi disain või kasutajalood peaksid muutuma ning kui keeruline ja aeganõudev nende muutmine on. Näiteks on testides kirjeldatud üldiselt kasutaja sisselogimist, kuid testis endas pole peale vaadates kohe aru saada, mis kasutajaga on tegemist, vaid selle sai teada vaid teises failis oleva meetodi kirjelduse järgi, kuigi testid kasutasid erinevaid testkasutajaid. **Probleem seisneb selles, et testi ebaõnnestumise korral on keeruline välja selgitada, millise testkasutajaga teatud testilugu ebaõnnestus.**

Testide meetodites pole ka täpselt ja kergesti aru saada, milliseid elemente lehe laadimisel oodatakse ja valideeritakse ning millistele elementidele lehel näiteks vajutatakse. Samuti on mitmete identsete tegevuste jaoks kirjutatud erinevaid meetodeid, mis teeb uute testide kirjutamisel eelneva koodi analüüsimise ja õigete meetodite leidmise aeganõudvaks ning tülikaks. Tegelikkuses võiks kasutada rohkem universaalseid sisendeid kasutavaid

meetodeid kui nende funktsionaalsus on sama tulemusega, mis kindlate väärtustega meetoditel. Meetodites pole enamasti kasutatud ka sisendeid, seega iga uue elemendi valideerimiseks või valimiseks on tihti vaja kirjutada ka uus meetod, mis muudab testide kirjutamise aeganõudvaks. **Selline koodihaldus tekitab aga topelt koodi, mida tegelikkuses saaks lühemalt ja süstematiseeritumalt kirjutada.**

Autor on valinud üheks analüüsitavaks testiks arve detailandmete avamise iseteeninduspportaalis. Konkreetne test on valitud just äriväärtust silmas pidades, kuna arvete maksmine on iseteeninduspportaalis üks olulisemaid toiminguid. Selleks logib kasutaja keskkonda sisse, liigub arvete sektsiooni ning näeb seal arveid listivaates. Seejärel valib kasutaja ühe maksmata arve, avab selle ja näeb arve maksmise nuppu.

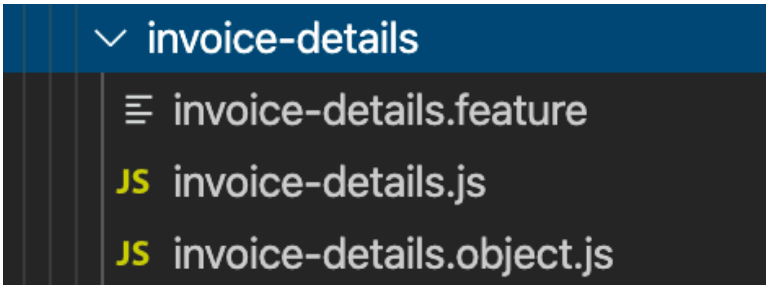
Joonis 3 illustreerib arve detailvaate avamise automaattesti:

```
Feature: Invoice details

    Background: User is in invoice details page
        Given User logs in with valid email address and password
        And User is able to see the dashboard
        And User navigates to "invoices"

    Scenario: User goes to invoice details page
        When User selects an open invoice
        Then User sees invoice main info
        And User sees "PayButton"
```

Joonis 3. AS-IS vaade arve detailvaate avamise automaattestist (allikas: autori koostatud)

Automaattestide koodihaldus on jagatud rakenduse funktsionaalsuse kaupa nii, et igas kaustas on nii ühe konkreetse funktsionaalsuse testifail, meetodite fail kui ka identifikaatorite fail.

Joonis 4 illustreerib ühe funktsionaalsuse, arve andmete detailvaate kausta, kus on nii arve andmete testifail, Javascriptis kirjutatud arve andmete testidega seotud meetodite fail ning vastav identifikaatorite fail:




Joonis 4. AS-IS vaade arve detailvaate testide, meetodite ja identifikaatorite kaust ja failid (allikas: autori koostatud)

**Probleem seisneb selles, et kui mõnes teise funktsionaalsuse testis tahetakse kasutada samu meetodeid või identifikaatoreid, mis näiteks arve detailandmete kaustas, on neid sealt tülikas üles leida.**

Seega esineb automaattestide kirjutamisel ja haldamisel mitmeid probleeme: automaattestide kirjutamisel lähtutakse vaid kasutusloo stsenaariumitest, teste käivitatakse vaid enda seadmes lokaalselt ja puudulikult, seejuures ei salvestada testiraporteid, meetodites ei kasutada tihti sisendeid ning puudub ülevaade, millised testid on automatiseeritud.

# 3 Parimate praktikate ülevaade

Antud peatükis kirjeldab ja analüüsib autor parimaid tarkvara testimisega seotud protsesse ning praktikaid. Autor toetub praktikate analüüsimisel suuresti tiimis valitud tööriistade kasutusjuhistele.

## 3.1 Testimisstrateegia

Testimisstrateegia on testimisega seotud pikaajaliste plaanide kogum, mis kujutab endast sageli staatilist dokumenti ning mis muutub ajas vähe. Testimisstrateegia dokumendis peaks olema kirjas kõik tiimi või projekti testimisprotsessidega seotu, näiteks tiimis kasutatavad testimistehnikad ja -praktikad, testimistööriistad, sh. nii manuaal- kui ka automaattestide omad, testitulemuste haldamine, vigade raporteerimine ja haldamine. [9]

## 3.2 Kasutuslugude ajahinnangud

Tarkvaraprojekti valmimise võimalikult täpse ennustatavuse jaoks on oluline tööülesannetele võimalikult täpsete ajahinnangute andmine. Seejuures on oluline arvesse võtta nii arenduseks kui ka testimiseks kuluvat aega. [7] Testimistegevustele ajahinnangu andmisel on hea lähtuda eelnevatest sarnastest projektidest ning ülesannetest. Testimistegevuste all tuleb silmas pidada konkreetselt testimisele kuluvat, manuaal- ja automaattestide kirjutamiseks, muutmiseks ja ülevaatamiseks kuluvat aega. Samuti peab arvestama ka näiteks vastavate testandmete otsimise, leidmise või loomise ajaga. [19]

## 3.3 Manuaaltestide kirjutamine ja haldamine

Järgnevalt kirjeldab autor analüüsitud parimaid praktikaid manuaaltestide kirjutamisest ning haldamisest.

### 3.3.1 Testide kirjutamine, testide ülevaatus ja täiendamine

Testilugusid kirjutatakse lähtuvalt kasutuslugudest ja ärinõuetest, kirjeldades testitava rakenduse või süsteemi käitumist. [9] Testilugu on komplekt testandmetest, testisammudest ja oodatavast tulemusest, mis kirjeldab rakenduse käitumist. [5]

Testide ülevaatus on oluline, et nende kirjutamisel ei jääks sisse kirja- või trükivigu, kataks kasutusloos püstitatud nõudeid, vastaks disainile ja rakenduse arendusele ning oleks ka teistele tiimiliikmetele peale kvaliteediinseneri selgelt arusaadavad.

### 3.3.2 Testiplaanid

Testiplaan on dokument, mis koondab endas testilugusid. Testiplaani võib käsitleda nii tööriista kui ka tootena. Testiplaan tootena võib olla justkui juhend rakenduse testimiseks ja standarditele vastamiseks. Samuti võib testiplaan olla tööriist, sest selle abil on võimalik testimist vajavat rakendust hallata ning leida rakendusest vigu. [20] Erinevalt testimisstrateegiast, mis on enamasti ajas mitte muutuv dokument, on testiplaan nö. elav dokument ja ajas pidevalt muutuv. Seega on oluline, et testiplaani uuendatakse pidevalt vastavalt rakenduse funktsionaalsuse ja ärinõuete muutumisele. Isegi, kui tiimis ei kasutata konkreetset testiplaani kui dokumenti, on siiski vaja testimist planeerida. [9]

Hea testiplaan ja testide dokumentatsioon aitab tehnilisi testimisülesandeid hõlbustada, parandab testimisülesannete ja -protsesside alast kommunikatsiooni ning annab testimist vajava rakenduse haldamiseks, planeerimiseks ja organiseerimiseks vastava struktuuri. [20]

Testiplaani koostamisel on väga oluline teada kõiki rakenduse funktsionaalsusi ja võimalusi. Kui testimisel kasutada testiplaani nimekirja, ei jää osa funktsionaalsusi kahe silma vahele. [20]

Testiplaani kasutamisel on võimalik teistele tiimiliikmetele või huvitatud osapooltele selgitada testimisstrateegiaid ja testija vaatevinklit testimisele. Testiplaaniga on võimalik testimiseks kuluvat aega ja pingutust hinnata, samuti saab sellega uurida ja kindlaks määrata rakenduse testimise katvust ja täpsust. [20]

### 3.3.3 Jira ja TestFLO

Nagu eelnevalt mainitud, kasutatakse projektis testide kirjutamiseks ja haldamiseks Jira TestFLO tööriista. Selle abil on võimalik testlood lisada testide repositooriumisse ehk hoidlasse. Testilugusid saab vastavalt vajadusele sorteerida kaustadesse ja alamkaustadesse. [21]

TestFLO testide hoidlas on võimalik lisada kõiki teste ja kaustu korraga läbitavasse regressioontestiplaani või üksikuid teste ja kaustu eraldi. Vastavast kategoriseerimata

testide kaustast on võimalik testilugusid vastavasse kausta tõsta nii, et ühte testi ei saa üle ühe korda mõnda kausta lisada. Samuti saab hoidla teste otsingumootori abil otsida ja filtreerida vastavalt vajadusele, näiteks vaadata, millised testid vajavad ülevaadet või millised testid on automatiseerimata või automatiseeritud. Kaustade juurde on märgitud ka selles olevate testide arv. [21]

TestFLO repositooriumi abil on võimalik luua regressioontestimise jaoks konkreetseid testiplaane, mis võivad viidata näiteks konkreetsele rakenduse versiooninumbrile. Seega on võimalik olemasolevatest repositooriumi *test case template*'idest luua konkreetsed *test case*'id, mida saab regressioontestimise jaoks läbida. Testide läbimisel on võimalik märkida, kas konkreetne testilugu läbib kõik nõuded või ebaõnnestub mõnel sammul. Ebaõnnestumise korral märgitakse test ära ning testiloosse saab kirjutada kommentaari, mis põhjusel test ebaõnnestus. Samuti on võimalik ebaõnnestunud testi korral luua Jirasse vearaport, mille saab testitud testilooga ära ühendada.

### 3.3.4 Gherkini keel

Gherkin on äripoolele mõistetav keel, millega saab arendusdetailidesse laskumata ärinõudeid kirjeldada. [22]

Gherkinit on võimalik kasutada paljudes keeltes, kuid enamasti kasutatakse inglise keelt. Gherkin kasutab spetsiaalseid märksõnasid, millega antakse testile struktuur ja testisammudele tähendus. Gherkini dokumendis algavad enamus ridu märksõnadega: *Feature, Scenario, Given, When, Then*, *And* ja *Background*, millele järgnevad konkreetsed laused, mis kirjeldavad kasutajatoiminguid, ärinõudeid ja süsteemi käitumist. Samuti kasutatakse spetsiaalseid märksõnu ka kommentaaride (#) ja siltide (@) jaoks. [23]

*Feature*-märksõna kasutatakse omavahel seotud stsenaariumite grupeerimiseks. Iga Gherkini dokumendi esimene märksõna peab olema *Feature*. [23]

*Background*-märksõna kasutatakse testifaili alguses, et vältida testides kasutamast samu, enamasti *Given*-märksõnaga samme. *Background*-sammudes ei soovitata hoida üle 4 sammu, kuna sellele järgnevaid stsenaariume lugedes peab *Background*-samme alati meeles pidama. [23]

Testisammude jaoks kasutatakse enamasti märksõnu *Given, When, Then* ja *And*. *Given*-märksõna kasutatakse süsteemi algse seisundi kirjeldamiseks. *When*-märksõna kasutatakse mingi tegevuse või sündmuse kirjeldamiseks, selleks võib olla nii mõni kasutajatoiming kui ka mõne teise süsteemi poolt vallandatud toiming. *Then*-märksõna kasutatakse oodatava tulemuse kirjeldamiseks, mida on võimalik ka kasutajal vaadelda [23].

## 3.4 Automaattestimine

Automaattestimine on manuaalsete testide automatiseerimine, mille eesmärk on vabastada kvaliteediinsenerid rutiinsest ja aeganõudvast tööst, et nad saaksid keskenduda uute bugide leidmisele ja uurivtestimisele. Manuaalne regressioontestimine on ajakulukas ning ei pruugi leida uusi vigu. [9]

Automaattestimise alla kuulub peale testide kirjutamise ka testiraportite analüüs. Testiraportid annavad käivitatud testide ja testisammude kohta põhjaliku ülevaate ning raport peab sisaldama huvitatud osapoolte jaoks vajalikku informatsiooni. Testiraportid peavad olema kõigile osapooltele ka kättesaadavad, näiteks võib seda saata e-kirjana või võib see olla kättesaadav mõnel projekti haldamise leheküljel, näiteks Jiras. Testiraportite juures on oluline ka nende ajaloo säilitamine, et saaks regressioontestimise jaoks koguda statistikat nende testilugude kohta, mis on sageli ebaõnnestunud. [24]

Testide automatiseerimise tasemed on järgmised: ühiktestid, integratsioonitestid, API-testid ja kasutajaliidese testid. See tähendab, et automatiseerimist peaks alustama ühiktestide tasemelt, eesmärk on katta võimalikult palju ära esimese taseme ehk ühiktestidega ning alles siis liikuma testide automatiseerimisega järgmistele tasemetele. [25] Seega on kasutajaliidese automaatteste kirjutades oluline teada, mida testitakse rakenduses juba eelnevatel tasemetel.

Automaattestimise eesmärkide hulka kuuluvad näiteks:

- testimise tõhususe parendamine
- funktsioonide testidega katvuse parendamine
- testimisele kuluva ressursi vähendamine
- testimise sageduse tõstmine ning samas testimistsüklite ajakulu vähendamine [24]

Automaattestimisel on palju eeliseid, näiteks:

- teste saab jooksutada kiiresti
- teste saab jooksutada mitmetes keskkondades korraga ning paralleelselt
- kiire tagasiside rakenduse kvaliteedi kohta
- testimisressursside tõhus jaotamine [24]

## 3.5 Automaattestide kirjutamine

Regressioontestimise seisukohast on väga oluline, et vähemalt osa olulisi teste oleksid automatiseeritud, et korra juba testitud funktsionaalsuste uuesti testimise koormus oleks võimalikult väike. [9]

Samas ei ole otstarbekas automatiseerida kõiki stsenaariume ja manuaalteste, kuna siis tõuseb ka nende haldamise koormus ning ka nende täitmise aeg on aeglasem. Testide automatiseerimisel tuleb kõige enam keskenduda just nendele rakenduse osadele mis on kriitilised, suurema riskiga ning mille manuaalne testimine võtab kaua aega. [26]

Automaattestide kirjutamisel on väga oluline ka nende hallatavus ja testide muutmise lihtsus, kuna kasutajaliidese muutuste korral on tihti vaja muuta ka testisamme. [9] Seega tuleb automaattestide arhitektuuri loomisel silmas pidada, et teste ei kirjutada ühe korra ning need ei jää igavesti muutumatuks. Testide asjakohasuse säilitamiseks tuleb teste aeg-ajalt uuendada ja kohendada, samuti võivad ajaga mõned testid muutuda aegunuks ning need tuleks testikomplektist eemaldada. [24]

Gherkini testides saab testifailidesse lisada *tag*'id ehk sildid *background'i* ehk tausta või iga funktsionaalsustesti järgi. Nii on võimalik teste jooksutada ühekaupa või funktsionaalsuste kaupa. Samuti on võimalik kasutada erinevaid silte viitamaks väliste süsteemide, näiteks ärinõuete- või testihaldusrakenduste identifikaatoritele. [23]

Testimisraamistiku TestCafe dokumentatsioonis antakse mitmeid soovitusi parimateks praktikateks:

- mitte kirjutada pikki teste, kuna lühemaid stsenaariume on lihtsam analüüsida ning nendes vigu otsida;

- hoida testidega seotud kaust rakenduse koodist eraldi, sealjuures hoia rakenduse elementide identifikaatoreid, üldkasutatavaid meetodeid, testkasutajate ja -rollide infot ning teste eraldi kaustades;

- sarnaste funktsionaalsuste testifaile hoida alamkaustades. [13]

TestCafe *Page Object Model* disainimuster kujutab endast näiteks testide, meetodite, identifikaatorite ja testkasutajate jagamist eraldi kaustadesse. Testid jagunevad omakorda funktsionaalsuste kaupa kaustadesse. Sellisel viisil kaustade ja failide jagamine aitab ära hoida duplitseeritud koodi ning on hea ülevaade juba olemasolevast. Samuti on nii lihtne leida vajadusel üles vastavaid meetodeid või teste, mida on tarvis täiendada või muuta. Joonis 5 illustreerib näidet testiprojekti failistruktuurist, mis kasutab *Page Object Model* disainimustrit:

```
.
├── .testcaferc.json
└── tests
    ├── |- test_group1/
    │   └── |-test1.js
    │       |-test2.js
    ├── |- test_group2/
    │   └── |-test1.js
    │       |-test2.js
    ├── |- page_model/
    │   └── |- page1.js
    │       |- page2.js
    ├── |- helpers/
    │   └── |- helper1.js
    │       |- helper2.js
    ├── |- roles/
    │   └── |- roles.js
    └── |-data
```

Joonis 5. Näide testiprojekti failistruktuurist [13]

Rakenduse elementide identifikaatorid ehk testisammude läbimiseks kasutatavad eesrakenduse atribuudid ei tohiks olla liiga üldised, näiteks ühe suvalise nupu valimine, ega ka liiga täpsed, näiteks konkreetse klassi järgi valimine, kuna rakenduse koodimuudatuste tõttu võivad need tihti muutuda ning ka identifikaator tuleb ümber defineerida. [13]

Hea praktika on identifikaatorite defineerimisel kasutada konkreetseid automatiseerimise atribuute, näiteks *„data-testid“,* kuna koodimuudatuste korral jäävad need enamasti muutumatuks ning seega ei pea ka testikoodis identifikaatoreid ümber kirjutama. [13]

## 3.6 Automaattestide käivitamine

Pideva integratsiooni korral on võimalik rakenduse vigu kiiresti avastada ning nende täpseid ilmnemiskohti määrata. [4]

Automaattestimise edukaks kasutamiseks on vaja läbi viia mitmeid tegevusi:

- testikoodi kergesti hallatavaks kujundamine;
- automaattestide dokumenteerimine, sh. automaattestimise eesmärgid peavad olema selged ning kirjeldama, milliseid rakenduse funktsionaalsusi ja osi testitakse;
- vananenud või ebavajalike testide koodibaasist eemaldamine;
- automatiseeritud testide ajakohasena hoidmine, mis tähendab, et rakenduse funktsionaalsuse või nõuete muutmisel ja seega testide ebaõnnestumise korral ebaõnnestunud testide parandamine;
- raporteerimise võimaluste rakendamine, mis tähendab, et testiraportid peaksid andma vajalikku infot erinevatele osapooltele ning testiraportid peaksid olema kõigile osapooltele ka lihtsasti kättesaadavad. [24]

Automaattestide kasulikkuse seisukohast on väga oluline, et teste käivitataks tihti ning automaatselt. Tavaliselt kehtib reegel, et mida tihemini rakenduse uusi versioone avaldatakse ning sellega ka vastavaid testimistsükleid, seda suurem kasu on automaattestidest. [24]

Tihti kasutatakse testide automaatseks käivitamiseks pideva integratsiooni tööriistu, näiteks Jenkins. TestCafe toetab integratsiooni Jenkinsiga, kus saab määrata testide käivitamise sageduse ning aja, samuti saab Jenkinsi kaudu peale testikomplekti läbimist automaatse testiraporti. [13]

Jenkinsit on võimalik integreerida ka suhtlustarkvaraga Slack, mida tiimis juba ühe peamise suhtlusvahendina kasutatakse. Slacki ja Jenkinsit on omavahel võimalik seadistada nii, et Slacki tulevad testide käivitamise või testiraportite kohta teated. [27]

Automaattestide ükshaaval ehk üksteise järel käivitamine on aeganõudev protsess. Samas on oluline, et testide käivitamine oleks võimalikult kiire protsess ning nende käivitamisel saadud info oleks võimalikult kiiresti olemas. TestCafe võimaldab teste käivitada paralleelselt, mis tähendab, et testid jooksevad mitmes grupis ühel ajal ning seega väheneb kogu testide käivitamise aeg. Näiteks on võimalik seadistada testide käivitamine nii, et testikomplekt jookseb korraga kolme veebilehitseja peal, nii väheneb testide käivitamise aeg kolmekordselt. [13]

# 4 Töö- ja testimisprotsesside parendamine

Antud peatükis esitatakse põhjendatud ettepanekud töö- ja testimisprotsesside muudatustele ning luuakse kliendiprojekti jaoks testimisstrateegia. Autor toob välja kasutusel olevate praktikate ja protsesside parenduskohad ja annab konkreetsed soovitused.

## 4.1 Testimisprotsesside algne analüüs ja protsessimuudatused

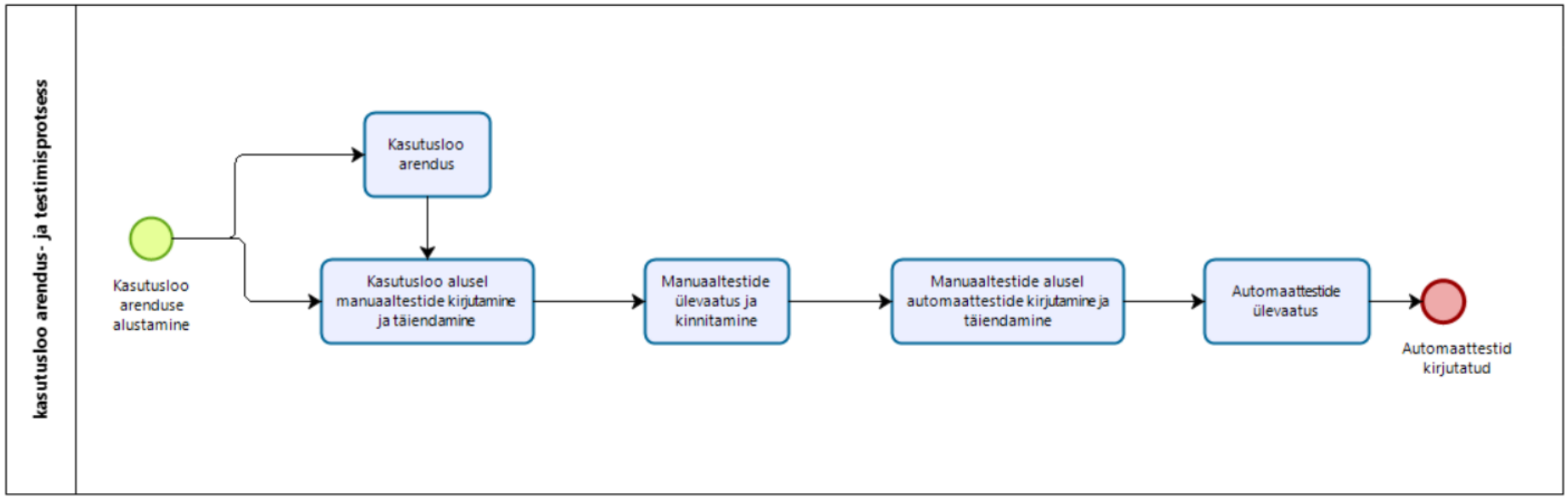
Kuna töö autor on töötanud varem ka mitmetes teistes tarkvaraarendusprojektides, sai ta kohe algul uue tiimiga liitumisel hakata töö- ja testimisprotsesse analüüsima ning võrdlema, mida tehakse konkreetses projektis eelnevates projektides osalenuna teistmoodi. Seega oskas autor juba tiimiga liitudes näha tööprotsessides ja -praktikates potentsiaalseid parendusvaldkondi.

Autor on saanud parendusettepanekuteks ideid näiteks sellest, et mitmes eelnevas projektis oli testide kirjutamise töö jagatud nii, et algul kirjutati rakenduste kasutuslugude, funktsionaalsuste ja ärinõuete alusel manuaaltestid, mis vaadati teise tiimis töötava kvaliteediinseneri poolt üle, nendele anti vajadusel parendussoovitusi või parandati need jooksvalt. Peale testi kinnitamist sai hakata tegelema automaattestide kirjutamisega, mis samuti omakorda mentori või teise tiimis töötava kvaliteediinseneri poolt üle vaadati ning millele jagati omapoolseid soovitusi ja parendusettepanekuid. See garanteeris, et automaattestide katvus üle manuaaltestide oli hõlpsasti kontrollitav ning oli hea jälgida kuidas täpselt järje peal ollakse.

Seega on autor pakkunud ka parimate praktikate soovitusi arvesse võttes just sellise protsessi lahenduse, kus kasutusloo arendamisel peale manuaaltestide kirjutamist ja nende kinnitamist kirjutatakse nende alusel automaattestid.

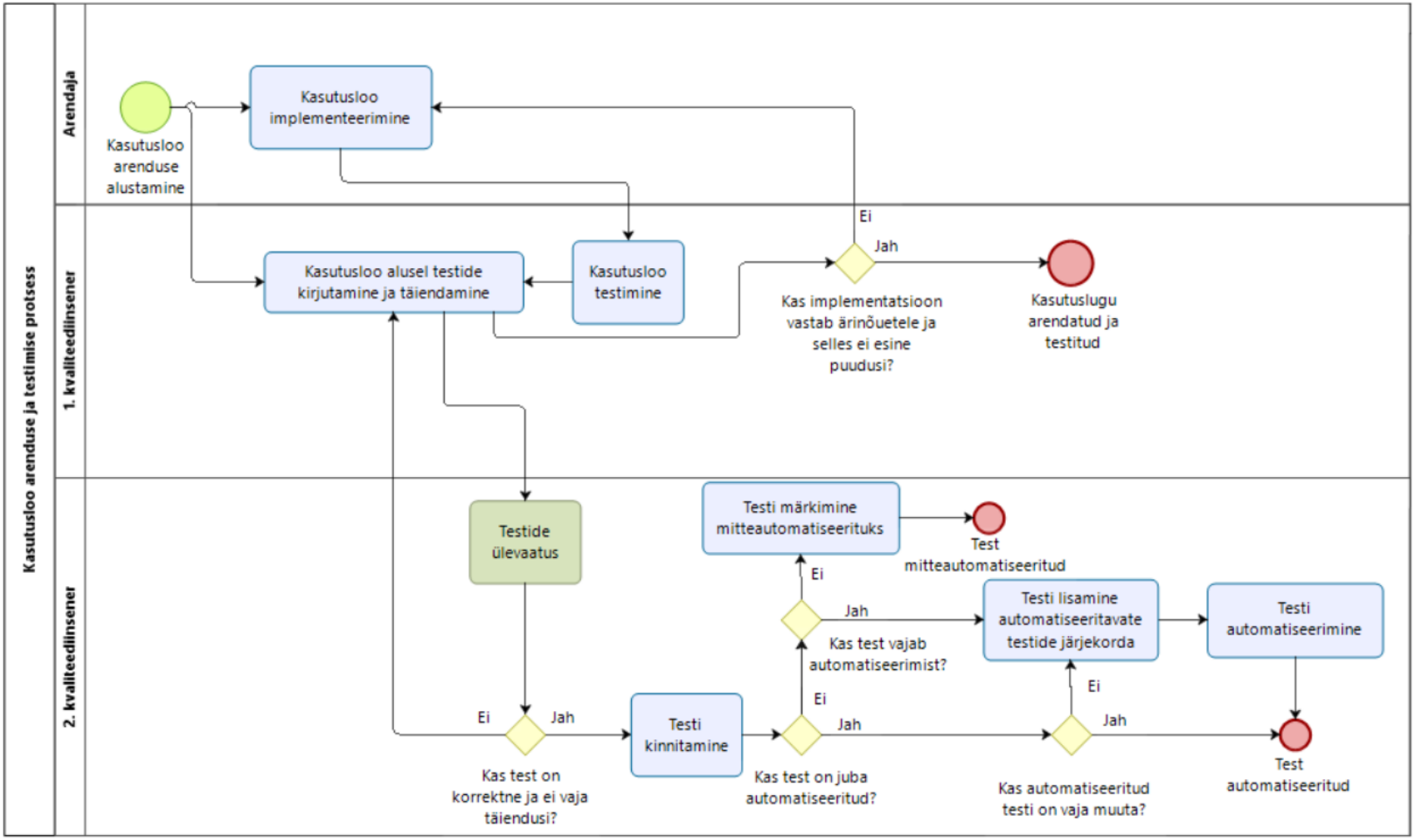
Kasutusloo arendusse lisamise järgselt saab alustada nii kasutusloo arendust kui ka paralleelselt manuaaltestide kirjutamise ja täiendamisega. Seega testide täiendamise ja parandamisega võib tegeleda juba enne seda, kui arendatav kasutuslugu on jõudnud testimisse, kuid kasutusloo päriselt testimisfaasi jõudmisel on vaja kindlasti vaadata eelnevalt täiendatud testid üle ning teha vajadusel vastavad parandused või kirjutada juurde vajalikke teste. Joonis 6 illustreerib kasutusloo arendamise, testimise,

manuaaltestide kirjutamise, nende ülevaatamise ning automaattestide kirjutamise ja omakorda nende ülevaatamise protsessi:



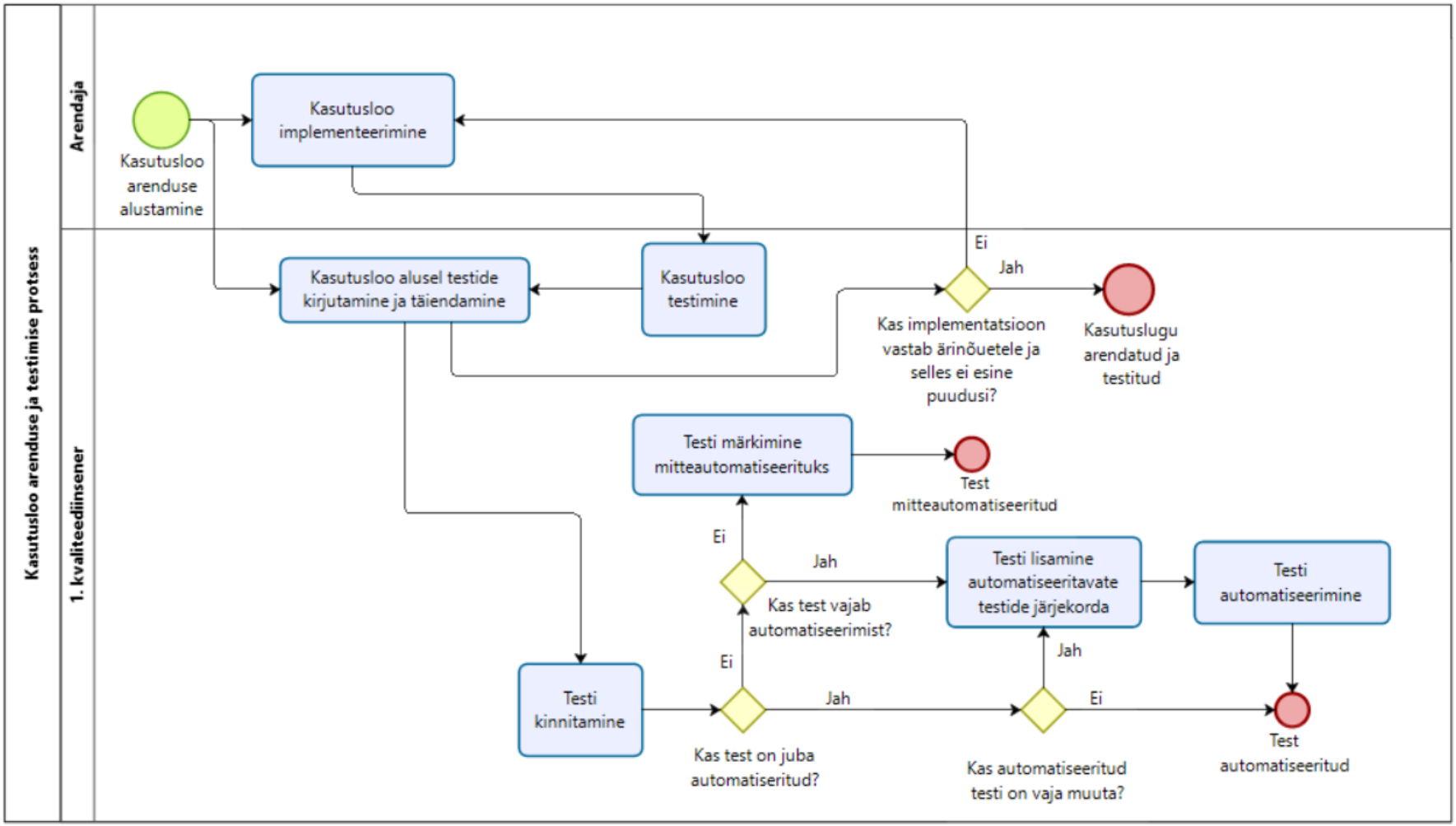

Joonis 6. TO-BE vaade kasutusloo põhilisest arendus- ja testimisprotsessist (allikas: autori koostatud)

Kliendiprojekti koosseisu analüüsi alustamise hetkel arvesse võttes, on autor koostanud tuleviku protsessijoonise, kus kirjeldatakse kasutusloo arenduse, testimise ja testide kirjutamise protsessi. Kasutusloo arendusloo alustamisel saab hakata kasutuslugu arendama ning kasutusloole manuaaltestide kirjutamisega tegelema paralleelselt. Peale kasutusloo arendust liigub kasutuslugu testimisfaasi, kus tuleb ka olemasolevad testilood üle vaadata ning vajadusel täiendada ja parandada. Kui kasutuslugu on testitud, selle arenduses ei esine puudusi ning testid on kirjutatud, saab kasutusloo märkida arendatuks ja testituks ehk tehtuks.  Joonis 7 näitab kahe kvaliteediinseneri korral kasutusloo testimise protsessi:




Joonis 7. TO-BE kasutusloo arenduse ja testimise protsess vähemalt kahe kvaliteediinseneri korral (allikas: autori koostatud).

Arvestades sellega, et tiimis ei pruugi alati olla kaks kvaliteediinseneri, ning ka sellega, et loodavat testimisstrateegiat saaksid kasutada ka teised sarnased tarkvaraarendustiimid, on autor pakkunud välja ka üksikisiku testimisprotsessi. Võrreldes kahe kvaliteediinseneri või kvaliteediinseneri rolli täitva isikuga tiimis, on erinevus selles, et protsess saab toimida ka ilma teise isiku testide ülevaatuseta. Seejuures on aga oht, et ainult ühe inimese vaatevinklist lähtudes on testid ebapiisavad või neis võib esineda rohkem vigu kui siis, kui kvaliteediinsener või mõni tiimiliige need üle vaatab ja kinnitab. Vastavat kirjeldatud protsessi illustreerib Joonis 8:




Joonis 8. TO-BE kasutusloo arenduse ja testimise protsess ühe kvaliteediinseneri korral (allikas: autori koostatud)

Nagu eelnevalt mainitud, võivad kvaliteediinseneri rolli kanda ka teised liikmed tiimis, samuti võib tiimis ühe kvaliteediinseneri korral keegi teise rolliga inimene manuaalteste üle vaadata ning neid kinnitada. Seega võib ka ühe kvaliteediinseneriga tiimis jätta testide ülevaatamise tegevuse sisse ning sellega võib tegeleda näiteks mõni arendaja.

## 4.2 Kasutuslugude hindamine

Kasutuslugude ajahinnangu määramisel arvestatakse edaspidi ka testimisele kuluva ajahinnanguga, kuna nii on võimalik täpsemini kasutuslugude arendusele ja testimisele kuluvat aega paremini ette ennustada. Eelnevalt on kasutuslugude ajahinnangul lähtutud vaid arendamisele kuluvast ajast, kuid testimise ning testimisprotsessidele kuluv aeg on hindamisest välja jäetud. Testimise ajahinnangu lisamine kasutuslugude arendamise ajahinnangusse aitab kasutuslugude ning ka terve arendusprotsessi ennustatavust muuta täpsemaks. Samuti on võimalik magistritöö käigus tehtud analüüsi ja muudatuste abil ajahinnagut paremini määrata, kuna on selgem ülevaade olemasolevatest testidest ning regressioontestimist on lihtsam planeerida, sealhulgas ka kasutuslugude ajahinnanguid.

## 4.3 Manuaaltestide kirjutamine ja haldamine

Antud peatükis kirjeldab autor manuaaltestide kirjutamise ja haldamisega seotud konkreetseid muudatusi, mida ta soovitab projektis läbi viia.

### 4.3.1 Gherkini keele kasutamine

Autor on märganud, et projektis kirjutatakse Gherkini keeles täpsemaid reegleid järgimata, kuid autor teeb tiimile ettepaneku edaspidi kasutuslugude või testide kirjeldamisel neid rohkem järgida. Näiteks peaks Given-samm kirjeldama juba kasutajapoolset tehtud tegevust või olekut, When-samm peaks kirjeldama kasutaja või süsteemi toimingut ning Then-samm kirjeldab tulemust. Testide refaktoreerimisel ning uute testide kirjutamisel oleks autori arvates vaja selguse ja süsteemsuse huvides neid soovitusi arvesse võtta.

### 4.3.2 Testilugude kirjutamine

Manuaaltestid peaksid olema seotud mitmete kasutuslugudega funktsionaalsuste põhiselt. Kasutuslugude testimisel ei peaks iga kord kirjutama uusi teste, vaid peaks üle vaatama

konkreetse funktsionaalsusega seotud juba olemasolevad testid ning neid täiendama, muutma või vajadusel kirjutama uued testid.

Testilugude pealkirjad peaks kirjutama ilma ühe konkreetse kasutusloo numbrita ja ilma „Validate that“ fraasita. Testilood peaks siduma relevantsete kasutuslugudega, et oleks parem ülevaade milliseid teste peaks regressioontestiplaanis läbima, kui mingite konkreetsete kasutuslugudega seotud featuurid vajavad regressioontestimist. Näiteks oleks testiloo nime **„PROJ-1234: Validate that user can successfully see the invoice details page in the self service portal“** asemel lühemat näiteks **„User sees the invoice details page“**, kus on kasutusloo identifikaator ning mittevajalikud fraasid testipealkirjast ära võetud. Nii on testi pealkiri lühem, seda on lihtsam ja kiirem lugeda ning testilugu ei viita enam otseselt ühele kasutusloole.

### 4.3.3 Testilugude ülevaatus

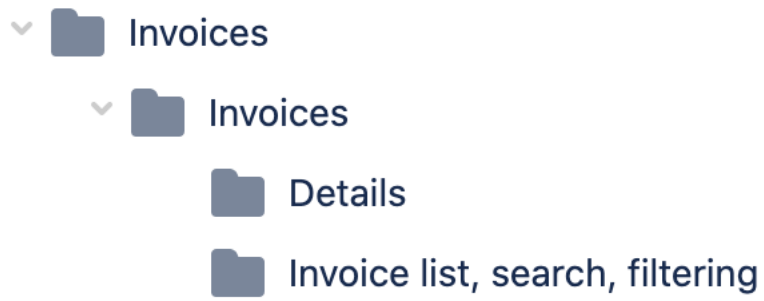
Töö autor on juba teinud oma tiimile mitmeid parendusettepanekuid tööprotsessides ja tööülesannete korralduses. Üheks parendusettepanekuks on see, et manuaaltestidele tehakse ülevaatus ehk *review* ning kvaliteediinsenerid kontrollivad ja vajadusel täiendavad üksteise manuaalteste või annavad selleks soovitusi. Sellega suurendatakse tõenäosust, et testide lugemisel saab ükskõik kes tiimist konkreetsest testist ja selle sammudest üheselt aru ning testide läbimisel ei teki kahetimõistmisi. Samuti on testide kirjutamisel oluline, et kasutuslood ja ärinõuded kaetakse nendes piisavalt regressioontestimise jaoks.

Kui tiimis on vähemal kaks kvaliteediinseneri, siis peaksid nad üksteise testide kirjutamisel tegema ka testide ülevaatuse, et veenduda, et testid ja testandmed oleksid korrektsed ning kasutuslugu oleks piisavalt testidega kaetud. Vajadusel võib teine kvaliteediinsener teste muuta, lisada, parandada või eemaldada testisamme. Samuti võib teine kvaliteediinsener parema katvuse jaoks soovitada teste lisada või kirjutada need ise. Kui testis muudatusi ei tehta või tehakse neid vähesel määral, võib siiski testi märkida kinnitatuks. Kui testides esines aga suuremaid muutusi, on hea tava muudatused uuesti algsel kvaliteediinseneril üle vaadata ning ta võib omakorda testi ise kinnitada.

### 4.3.4 Testilugude kogu

Testid tuleks koondata ühte kogusse ja sorteerida need vastavatesse hierarhilistesse kaustadesse, kasutades Jira TestFLO Test Repository funktsionaalsust. [21] Magistritöö

esitamise ajaks on töö autor vastavad muudatused juba teinud ning olemasolevad testilood sorteerinud ja kaustadesse paigutanud. See aitab ka testide analüüsi ja kinnitamise käigus saada parema ülevaate, millised testid on juba vastava funktsionaalsuse jaoks kirjutatud ning kus esineb veel puudujääke. Teste on funktsionaalsuste kaupa võimalik lihtsasti üles leida. Näiteks on kliendiportaalis arvetega seotud testid jagatud ühte suuremasse kausta, mis omakorda jaguneb eri vaadeteks, näiteks arvete listivaate ning detailvaate kaust. Joonis 9 illustreerib arvetega seotud testide kaustade vaadet TestFLO testilugude repositooriumis:




Joonis 9. TO-BE vaade arvete funktsionaalsuse testikaustad testiplaanis (allikas: autori koostatud)

Eelneval joonisel on näide, kuidas võib arvetega seotud teste kaustadesse jagada. Näiteks on arve listivaate testid ühes kaustas koos otsingu ja filtreerimise funktsionaalsuste testidega, kuna need on ühel kasutajaliidese lehel nähtavad ning seega on neid hea ühes kohas hoiustada. Kui funktsionaalsused ja testide arvud kasvavad väga suureks, on võimalik kaust vajadusel omakorda mitmeks jagada.

### 4.3.5 Testilugude tüübid

Jiras testilugude kirjeldamisel tuleb uue testiloo loomisel valida tüübiks „Test Case Template“ praeguse „Test Case“ asemel, kuna siis ei ole vaja testilugusid kopeerida või kloonida, vaid testilugude mallidest saab teha tulevased testilood, mida saab regressioontestimise käigus eraldi läbida. [21] Joonis 10 illustreerib testiloo tüüpi, millest genereeritakse alamtestilood:


Joonis 10. Testiloo tüüp Jiras, mille alusel regressioonplaani teste luuakse (allikas: autori koostatud)

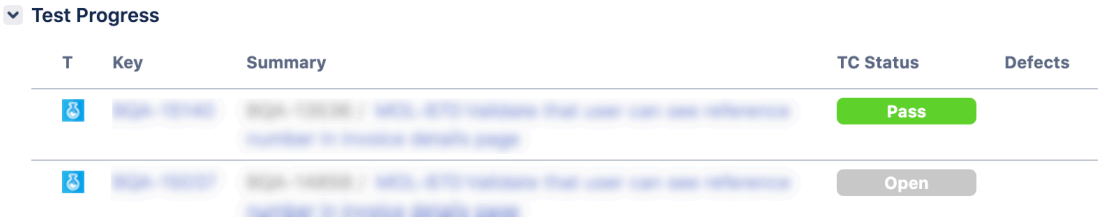
Antud joonisel kujutatud testiloo tüüpi kasutatakse uute testlugude loomisel ning testplaanidesse lisamisel.

Joonis 11 illustreerib testiloo tüüpi, mis luuakse Test Case Template tüüpi testiloost selle konkreetsesse testiplaani lisamisel:


Joonis 11. Testiloo tüüp Jiras, mis luuakse Test Case Template tüüpi testiloost (allikas: autori koostatud)

Antud joonisel kujutatud testiloo tüüpi saab luua Test Case Template testiloo alusel, mis lisatakse ka regressioontestiplaani.

Magistritöö esitamise ajaks on autor koos kaaskvaliteediinseneriga muutnud kõigi olemasolevate testilugude tüübi õigeks ning nüüd saab ka testilugude testiplaani lisamisel ja testide läbimisel näha iga testi ajalugu. Iga testilugu saab selle läbimise käigus märkida läbituks (*Pass*) või ebaõnnestunuks (*Fail*). Kui testilugu on regressioontestiplaanis veel avatud, siis on see märgitud vastavalt *Open.* Joonis 12 illustreerib ühe testiloo läbimise ajalugu, testide identifikaatorid ja pealkirjad on kliendiprojekti konfidentsiaalsuse huvides hägustatud:




Joonis 12. TO-BE vaade testimalli ajaloost ja testilugude progressist (allikas: autori koostatud)

Autori ettepanekul on tehtud lisaks muudatused Jira manuaaltestide haldamisel - näiteks lisatakse edaspidi iga manuaaltesti külge ka vastav silt, mis funktsionaalsusega test seotud on. Siltide sorteerimise alusel on hea teatud funktsionaalsuse kaupa rakendust testida ja ka testiplaani koostada. Samuti saab märkida iga testloo juures ära, kas test on ka automatiseeritud või pole selle automatiseerimine otstarbekas.

### 4.3.6 Testilugude töölaud

Testilugude heaks ülevaatuseks on autor loonud koos teise tiimis töötava kvaliteediinseneriga Jiras töölaua, kus saab erinevate filtrite abil testilugusid sorteerida ja grupeerida. Näiteks on nii võimalik eraldi sorteerida testilood staatuse järgi - millised testilood on kinnitatud või mis vajaksid veel üle vaatamist. Testilugude töölaud annab ka teistele tiimiliikmetele, eriti arendajatele hea ülevaate, millised testid ootavad automatiseerimist. Samuti on võimalik erinevate filtrite abil testilugusid funktsionaalsuste ja siltide kaupa lahterdada.

## 4.4 Automaattestide kirjutamine ja haldamine

TO-BE vaate projekteerimiseks on töö autor peale enamlevinumate parimate praktikate rakendamise konsulteerinud pidevalt ka Proeksperdi ühe vanemkvaliteediinseneriga, kuidas oleks otstarbekam teste refaktoreerida.

Üks parendusettepanek, mille autor varakult peale tiimiga liitumist välja pakkus, oli manuaal- ja automaattestide kontrollimine nii, et juba valmis kirjutatud automaattestile lisatakse samuti Jiras olev unikaalne manuaaltesti number. Samuti on vajalik manuaaltestile panna külge märge, et konkreetne test on automatiseeritud. Ettepanek kiideti tiimi ja teise kvaliteediinseneri poolt heaks ning nii on hetkeseisuga vaadatud umbes pooled automaattestid läbi ning koodis on sisse viidud ka vastavad muudatused. Kuna automaattestide kogu oli töö autori tiimiga liitudes üsna suur, on nende senine analüüs ja kontrollimine olnud aeganõudev ja mahukas töö.

Peale esimeste parendusettepanekute tiimi sisse viimist hakkasid siiski ilmnema veel mõned takistused töös, mida võiks parendada. Näiteks peaksid kasutajaliidese automaattestide eest vastutavad olema siiski kvaliteediinsenerid ning kontrollima, et automatiseerimisel kasutatakse õigeid testandmeid ja testides kontrollitakse täpselt neid asju, mida vaja. Seega on töö autor teinud tiimile ettepaneku muuta automaattestide kirjutamine ja haldamine kvaliteediinseneride vastutada, et nad saaksid kindlustada enda töö kvaliteeti ning samuti täpselt järge hoida, millised testid on automatiseeritud ning millised mitte.

Kuna peale autori esmaste parendusettepanekute kinnitamist on lepitud tiimis kokku, et automaattestide kirjutamise ja haldamise eest vastutavad kvaliteediinsenerid, oli vaja

alustada ka automaattestide kogu analüüsi ja nende refaktoreerimisega parema haldamise ja uute testide kiirema kirjutamise nimel. Automaattestid on eelnevalt kirjutatud arendajate poolt, kellest enamusel eelnev kasutajaliidese automaattestide kirjutamise kogemus puudub, kuid samas on neil pikaajaline kogemus ühiktestide kirjutamisega. Et jätkata arendajate poolt tehtud tööd, on selle tööülesande jätkamiseks olnud vaja eelnevalt teha neile analüüs ja uurida, mis on tehtud õigesti ning mida peaks jätkama, samuti millised koodiosad vajaksid parandust.

### 4.4.1 Automaattestide kirjutamine manuaaltestide alusel

Teine suurem muudatus, mille töö autor on tiimis muudatusena välja pakkunud ning mis on ka tiimis juba kasutusele võetud, on praktika, et automaattestid kirjutatakse manuaaltestide alusel. Samuti on oluline, et unikaalsed numbrid, mis luuakse automaatselt Jiras manuaaltestidele, oleksid kirjas ka automaattestidel. Peale testi automatiseerimist on ka Jiras vaja lisada märge, et test on automatiseeritud. Sel viisil saab lihtsalt nii automatiseeritud kui ka mitte automatiseeritud teste sorteerida ning teistel tiimiliikmetel on ülevaade, millised testid on veel automatiseerimata, kuid vajavad veel automatiseerimist.

Nagu eelnevalt mainitud, kasutatakse tiimis Gherkini keelt, milles kasutajalugusid ning ka teste kirjeldatakse. Sama kehtib ka automaattestide puhul. Lisaks kasutatakse testide kirjutamisel Javascripti ning teste käivitatakse TestCafe automaattestiraamistikuga. TestCafe abil on võimalik teste kirjutada Javascriptis või Typescriptis ning seda on võimalik integreerida mitmete pideva integratsiooni tööriistadega, nagu näiteks Jenkins. TestCafe on valitud projekti, kuna teste kirjutatakse samas keeles, milles kirjutatakse ka rakendust, tööriist on tasuta ja vabavaraline ning toetab enamikke enamlevinud veebilehitsejaid ning operatsioonisüsteeme. [13]

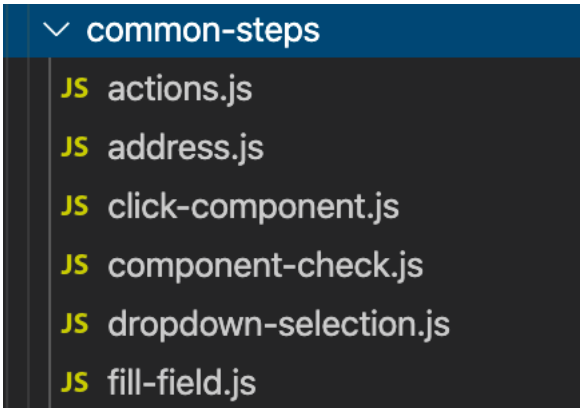
Tööprotsesside muutmisel on oluline pidada silmas eesmärki, milleks oli algul püstitatud kulu ehk *waste'*i elimineerimine ja dubleeritava töö minimeerimine. Kuna manuaaltestide kirjutamise ja ülevaatuse ehk *review*'ga tehakse juba automaattestidele eelnev töö ära, on kasulik automaattestide kirjutamisel need aluseks võtta ning samas ka neile samad identifikaatorid ja pealkirjad anda. Nii on võimalik Jiras manuaalsel testiloo muutmisel leida ka automatiseeritud test hõlpsasti üles ning teha ka selles vastavad muudatused. Samuti on regressioontestiplaani koostamisel võimalik näha, millised testid on juba

automaattestidega kaetud ja ei vaja konkreetse funktsionaalsuse käsitsi testimist, vaid keskenduma peaks pigem funktsionaalsuse disainile.

### 4.4.2 Automaattestidega seotud koodi haldamine

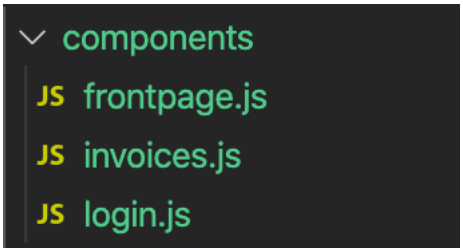
Automaattestide kogus peaksid funktsioonid ehk meetodid, testi- ning identifikaatorifailid olema eraldi kaustades ning sorteeritult, et nendest oleks parem ülevaade. Sel viisil on meetodeid ja identifikaatoreid võimalik paremini uutes testides kasutada kuna neid on võimalik loogilise sorteeringu järgi leida.

Testimeetodite paremaks haldamiseks soovitab autor hoida meetodeid eraldi kaustas ning vajadusel alamkaustades. Meetodite eraldamisel testikoodist annab hea ülevaate, millised meetodid on juba defineeritud ning mida on testide kirjutamiseks vaja veel defineerida. Sarnased meetodid on otstarbekas koondada ühte faili, näiteks lingile või nupule vajutamise meetodid võib hoida ühes failis, samuti välja täitmise meetodid ühes, kui selliste tegevuste jaoks on tarvis mitut meetodit. Joonis 13 illustreerib tulevikuvaadet sellest, kuidas on enamkasutatavad testide meetodid jagatud vastavasse kausta:



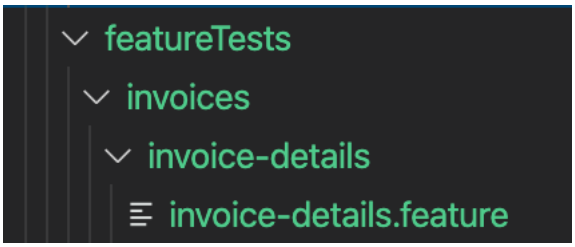

Joonis 13. TO-BE väljalõige enamlevinumate meetodite kaustast (allikas: autori koostatud)

Rakenduse identifikaatorite haldamisel on otstarbekas lähtuda Page Object Model disainist, kust need jagatakse lehe kaupa eri failidesse. Nii on võimalik konkreetse rakenduse vaate testide kirjutamisel leida ka vastavast failist vajaminevad identifikaatorid, mida testis kasutada. Joonis 14 illustreerib tulevikuvaate väljalõiget sellest, kuidas on rakenduse identifikaatorid ehk need elemendid, mida meetodid lehel tuvastavad, failidesse ja kausta jaotatud:




Joonis 14. Joonis 13. TO-BE vaate väljalõige identifikaatorite kaustast funktsionaalsuste kaupa jagatuna (allikas: autori koostatud)

Eelneval joonisel on toodud väljalõige kolmest failist, kus rakenduse eri vaadete identifikaatoreid hoitakse, näiteks on eraldi fail rakenduse esilehest, arvete listivaatest ja sisselogimise vaatest. Joonis 15 illustreerib väljalõiget tulevikuvaadet testikaustade loogikast, joonisel on näitena illustreeritud arve detailvaate avamise testifail:




Joonis 15. TO-BE vaate väljalõige arve detailvaate automaattestide kaustast (allikas: autori koostatud)

Kui eelneva ülesehitusega automaattestid on uue arhitektuuri järgi ümber kirjutatud, on neid ka kergem hallata. Samuti on oluline, et testide refaktoreerimisel ei kirjutata neid pelgalt uuele arhitektuurile ümber, vaid veendutakse testide puhul rakenduse kvaliteedi hindamiseks ka kasutatavates testandmetes, sammudes ja oodatavas tulemuses. Seega on nii arendajatele kui ka äripoolele ja tooteomanikule alati testiraportites näha, millised rakenduse osad ja funktsionaalsuses ning mis põhjustel on vahepeal katki läinud, lakanud töötamast või milliste teiste süsteemidega on integratsioon katkine või puudulik.

### 4.4.3 Sisendid ning taaskasutatavad meetodid

Testide kiiremaks kirjutamiseks ning paremaks haldamiseks pakub autor välja taaskasutatavate meetodite lahenduse, kus iga meetod kasutab vastavat sisendit. Mõnele lehe elemendile vajutamine oleks lahendatud vaid ühe taaskasutatava meetodi abil, mida saab kasutada kõigis testisammudes, kus on tegemist mingile elemendile vajutamisega. Joonis 16 illustreerib lingile vajutamise meetodit, mida on hiljem võimalik omakorda Gherkini sammu sisaldavas meetodis kasutada:

```
16    export async function clickLink(linkId) {
17      await t.click(linkId);
18    }
```

Joonis 16. TO-BE lingile vajutamise funktsioonist (allikas: autori koostatud)

Selle asemel, et kirjutada mitmeid erinevaid meetodeid erinevate elementide vajutamiseks või kontrollimiseks, et see on lehel kasutajale nähtav, soovitab autor koondada need meetodid ning kasutada meetodis konkreetsete testis vaja minevate elementide jaoks sisendeid.

Vajutatavaks elemendiks lehel võib olla näiteks arve, mida kasutaja soovib avada, maksmiseks vajutatav nupp, sõnumi saatmiseks mõeldud nupp või pealehele navigeerimise ikoon. Näiteks selleks, et kirjutada kaks erinevat meetodit, kus ühes kasutaja vajutab arvele, et selle detailvaadet avada ning teises kasutaja vajutab arve nupu peale arve maksmiseks, saaks need lahendada ühe meetodiga. Seega testisammu *„User clicks on an open invoice“* asemel kasutataks hoopis *meetodit* **„***User clicks on „AnOpenInvoice““*, kus sisendiks on üks avatud arve arvete listivaates. Joonis 17 illustreerib meetodit, kus kasutaja vajutab mingile konkreetsele elemendile lehel:

```
When('User clicks on {string}', async (t, params) => {
  const component = getComponent(params[0]);
  await clickLink(component);
});
```

Joonis 17. TO-BE Gherkini sammu meetod elemendile vajutamise funktsioonist (allikas: autori koostatud)

Mõni konkreetne element, mida on vaja kontrollida, kas see lehel esineb, võib olla näiteks arve detailandmete leht või sellel lehel arve maksmise nupp. Selle asemel, et kasutada erinevaid meetodeid konkreetsete elementide lehel ilmnemise kontrollimiseks, oleks otstarbekam kasutada jällegi sarnaseid meetodeid, nagu ka elementide vajutamise korral. Näiteks selle asemel, et kirjutada meetod nimega *„User sees invoice main info“,* oleks võimalik sama tegevust teha ka taaskasutatava sisendit kasutava meetodiga, mis sel juhul oleks „*User sees „InvoiceDetails““,* kus sisendina kasutatakse arve detailvaate identifikaatorit. Joonis 18 illustreerib meetodit, kus kasutaja näeb lehel mõnda konkreetset elementi:

```
Then('User sees {string}', async (t, params) => {
  await t
    .expect(getComponent(params[0]).exists)
    .ok('User does not see ' + params[0]);
});
```

Joonis 18. TO-BE Gherkini sammu meetod elemendi nägemise funktsioonist (allikas: autori koostatud)

Tihti on vaja testidega kontrollida, ega mõni element lehel nähtav ei ole, näiteks kas hüpikaknas sulgemise nuppu vajutades enam hüpikaken nähtav ei ole. Joonis 19 illustreerib meetodit, kus kasutaja ei näe lehel mõnda konkreetset elementi:

```
Then('User does not see {string}', async (t, params) => {
  await t
    .expect(getComponent(params[0]).exists)
    .notOk('User sees ' + params[0]);
});
```

Joonis 19. TO-BE Gherkini sammu meetod elemendi mitte nägemise funktsioonist (allikas: autori koostatud)

Praeguseks on tehtud tiimis töötavate arendajatega paarisprogrammeerimise sessioone, et olemasolevaid teste ja meetodeid refaktoreerida ning leida üheskoos paremad ja hallatavamad lahendused. Nagu eelnevalt analüüsis selgus, on vaja testides täpselt kirjeldada, millise testkasutajaga test käivitatakse, milliseid elemente lehel valitakse ja valideeritakse. Samuti on vaja parema ülevaate nimel jagada lehe elemendid, testandmed, meetodid ja käivitatavad testid erinevatesse failidesse. Elementide valideerimise ja valimise jaoks on nüüdseks kirjutatud peamised universaalsed meetodid, mida on võimalik enamikes testides kasutada.

### 4.4.4 Sildid testifailides

Igale funktsionaalsuse testifailile on vaja käivitamise lihtsuse huvides lisada ka vastavad sildid. Silte võib lisada nii suuremate kui ka väiksemate alamfunktsionaalsuste kohta, näiteks üldisem arvete funktsionaalsuse silt oleks vastavalt *„@invoices“* ning seda täpsustav arve detailvaate silt vastavalt *„@invoiceDetails“*. Sel viisil on võimalik

käivitada nii neid teste, mis hõlmavad konkreetselt arvete detailvaate testimist kui ka kõiki, suuremat funktsionaalsust hõlmavate arvete funktsionaalsusega seotud teste. Samuti on oluline, et igal testil oleks ka eraldi silt vastava Jira testiloo numbriga, mille abil on võimalik iga üksikut testi käivitada, näiteks andes programmis vastava käskluse, mille sisendiks on *„@TEST-12345“*.

Kuna raporti genereerimisel on testid listivaates ainult testide pealkirjadega, mitte testide ja funktsionaalsuste siltidega, soovitab autor kasutada testiloo identifikaatorit ka iga testi pealkirjas, näiteks lisada *„TEST-12345“* testi pealkirja algusesse. Kui test ebaõnnestub, on võimalik identifikaatori abil kiirelt leida üles ka Jira manuaalne test ning uurida sealt täpsemaid testandmeid, sisendeid, testisamme ning oodatavaid tulemusi. Näiteks näeks testilugu automaattestis välja nii: „Scenario: TEST-12345 User sees invoice details“. Joonis 20 illustreerib kavandatavat vaadet arve detailandmete vaate funktsionaalsuse automaattestist:

```
@invoices
@invoiceDetails
Feature: Invoice details

    Background: User is in invoices list page
        Given User has logged in as "Testuser 1" and selected company "Testcompany 1"
        And User navigates to "invoices"

    @TEST-12345
    Scenario: TEST-12345 User sees invoice details
        When User clicks on "AnOpenInvoice"
        Then User sees "InvoiceDetails"
        And User sees "PayButton"
```

Joonis 20. TO-BE vaade arve andmete automaattestist (allikas: autori koostatud)

Antud joonis kujutab arve detailvaate avamist, kus eelnevalt on kirjeldatud *Background* ehk igale testile eelnev taust. Antud test nimega „TEST-12345 *User sees invoice details*“ kontrollib, et kasutaja näeb ühele maksmata arvele klikkides selle detailvaadet ning seal olevat maksenuppu.

### 4.4.5 Testisammude koondamine

Analüüsi teostamise hetkel on testides kirjeldatud eraldi testisamme kus esimene on kasutaja sisse logimine ning teie iseteeninduskeskkonna avalehe nägemine. Kuna mõlemaid samme kasutavad enamikke teste, oleks need sammud otstarbekas ühte lisameetodisse lisada. Seega kahe testisammu *„When User logs in“* ja *„And User is able to see the dashboard“* asemel võiks teise testisammu koondada juba näiteks

sisselogimismeetodi sisse ning kõik testid, mis neid kahte sammu kasutavad, muutuvad ühe rea võrra lühemaks. Samuti on oluline meeles pidada, et kui rakenduses mõni kasutajavoog, antud juhul sisselogimine muutub, saab kasutajaliidese testides teha muudatuse vaid ühes kohas. Eelneva loogikaga oleks pidanud aga rakenduse voo muutumisel pidanud muudatuse tegema põhimõtteliselt kõikides testides. Seega oleksid uued testid vaid testisammuga *„When User logs in“,* mis omakorda sisaldab juba sisselogituna esilehe nägemist.

### 4.4.6 Automaattestide kirjutamise ja haldamise juhend

Töö autor on peale muudatusettepanekute kohta tiimilt kinnituse saamist koostanud ka koodibaasi README faili [28], kus kirjeldatakse ära põhilised testimisprotsessid, reeglid ning kokkulepped, mis on seotud manuaal- ja automaattestide kirjutamisega. Failis kirjeldatakse, et automaattestide kirjutamisel ja muutmisel tuleb aluseks võtta Jiras juba kinnitatud manuaaltestid, mis on automatiseerimiseks valmis või mis on juba automatiseeritud. Automatiseeritud manuaaltestide muutmisel Jiras tuleb teha vastavad muudatused ka koodis ehk automaattestides. Uue testi automatiseerimisel tuleb ka Jiras märkida vastav manuaaltest automatiseerituks, et teistel tiimiliikmetel oleks ülevaade, millised testid on juba automatiseeritud või vajavad seda veel. Failis antakse soovitused ja juhised nii testifailide ja testandmete halduse, elementide defineerimise, kui ka testide kirjutamise kohta. Samuti on failis kirjas, milliste põhimõtete järgi kirjeldatakse vastavate testitavate funktsionaalsuste sildid failides ja testides.

### 4.4.7 Automaattestide siltide fail

Töö autor on teinud ka ettepaneku koostada kõikide funktsionaalsuste siltide koondamiseks eraldi faili, kust saab ülevaate, milliseid sildid on failides kasutusel. Nende siltide alusel saab testide kirjutamisel kasutada juba seal kirjeldatud silte ning arendajad saavad vajadusel vastavate funktsionaalsuste teste käivitada. Uute siltide koodibaasi lisamisel tuleb vastav silt ka siltide nimekirja lisada.

### 4.4.8 Automaattestide käivitamine ja raporti genereerimine

Kuna automaattestide komplektis on palju teste, mis ebaõnnestuvad selle tõttu, et need ei kirjelda enam oodatavat rakenduse käitumist, on vaja need vastavalt rakenduse nõuetele uuendada. Autor soovitab esialgu märkida ebaõnnestuvad testid vastava sildiga, mis eemaldaks need igapäevasest automaatkäivitusest ja seejärel hakata teste ükshaaval

analüüsima, mis on nende ebaõnnestumise taga. Hetkel on autor koos tiimiliikmete abiga testikomplektis ära märkinud kõik ebaõnnestunud testid ja igapäevasesse käivitatavasse testikomplekti on jäänud 79 testi, mille käivitamise aeg on umbes 36 minutit.

Nagu eelnevalt mainitud, siis automaattestide esmase analüüsi käigus selgus, et vahel ebaõnnestuvad testid juhuslikult ning sama funktsionaalsuse manuaalsel testimisel on test edukas. Seega on autor välja pakkunud, et testide käivitamise seadistus peab olema selline, et iga ebaõnnestunud testi käivitatakse veel kaks korda. See muudab kokkuvõttes küll testide käivitamise aeglasemaks, kuid aitab välistada seda, et test ebaõnnestus juhuslikult, näiteks hetkelise aeglase andmebaasipäringu tõttu ning vähendab seda, et testiraportis on ebaadekvaatne info.

Peale ebaõnnestunud testide vastavate välistavate siltidega märkimist saab aegunud teste parandama hakata ning välja selgitada ka need testid, mis tõesti on rakendusest vigu üles leidnud ning seejärel koostada vastavad vearaportid Jirasse.

Automaattestide praegusega võrreldes kiiremaks käivitamiseks soovitab autor teste paralleelselt käivitada. Nii on võimalik saada kiiremat tagasisidet rakenduse seisukorrast. Testide paralleelselt käivitamisel ei oota iga uus test eelneva testi lõppu, vaid korraga käivitatakse mitu testikomplekti. Seega saab testikomplekti käivitamise aega kordades kiirendada.

Autor on tiimile teinud ettepaneku kasutada automaatestide automaatseks käivitamiseks pideva integratsiooni tööriista Jenkins. Automaatteste käivitatakse autori ettepanekul automaatselt igal tööpäevale järgneval ööl, et igal tööpäeva hommikul oleks värske testiraport uue infoga automaatselt koostatud.

Jenkinsiga on võimalik integreerida ka suhtlusvahendiga Slack, mille kaudu on kõigil tiimiliikmetel võimalik igal hommikul testiraportit lugeda. Seega soovitab autor Jenkinsi ning Slacki omavahel seadistada nii, et testide käivitamisel ning testiraportite koostamise kohta tuleks vastavasse Slack'i kanalisse teade, mida kõik tiimiliikmed saavad vajadusel lugeda.

# 5 Järeldused

Käesolevas peatükis annab autor ülevaate tehtud analüüsist, tehtud muudatustest ja töö alguses paika pandud mõõdikute mõõtetulemustest.

## 5.1 Järeldused ja tähelepanekud

Kliendiprojekti analüüsides tegi autor mitmeid tähelepanekuid. Autori hinnangul oleks pidanud enne automaattestide kirjutamise alustamist täpselt välja selgitama nende kirjutamise eesmärgi ning ka tuleviku haldamise loogika. Kui algul ehitada automaattestide kirjutamise arhitektuur raskesti hallatavalt üles, on seda hiljem muuta väga ajakulukas. Automaattestide olemasolul on väga oluline neid ka regulaarselt ja automaatselt käivitada. Samuti pani autor tähele, et tõenäoliselt polnud manuaaltestide kirjutamisel ja haldamisel iseteeninduspportaali funktsionaalsuste kasvamise ja muutumisega väga arvestatud. Projekti alguses võib tunduda, et teste ei ole vaja eraldi hallata ja sorteerida ja uuendada, kuid funktsionaalsuste kasvades võib testikomplekt muutuda väga raskesti hallatavaks, kui neid pidevalt ei hooldata.

## 5.2 SWOT analüüsist lähtuvad tegevused

Toetudes töö algul tehtud projektitiimi analüüsile teeb autor riskianalüüsi vastavalt nõrkuste ja ohtude vähendamiseks, samuti tiimi tugevuste ja võimaluste ära kasutamiseks. Kuna projektitiimi nõrkused võivad viia probleemideni, on oluline teha nende kohta riskianalüüs, mis aitaks riske leevendada. SWOT analüüsist tulenevate tugevuste ära kasutamiseks, nõrkuste vähendamiseks, samuti võimaluste ära kasutamiseks ja ohtude vähendamiseks teeb autor järgnevad ettepanekud:

- **Testiplaan on raskesti hallatav, puudub ülevaade testidest; Automaattestidega pole piisavalt hea testide katvus ning puudub hea ülevaade nende katvusest -** manuaal- ja automaattestide katvuse paremaks ülevaateks on autor teinud protsessimuudatusi, mille abil on võimalik näha, millised manuaaltestid on automatiseeritud ning millised seda vajavad. Samuti on testiplaan eelnevaga võrreldes kergemini hallatav ja parema ülevaatega.

- **Testimisele kuluvat aega ei arvestada ajahinnangusse –** kuna nüüdseks on tiimis antud muudatus tehtud, et testimise aeg arvestatakse samuti kasutuslugude ajahinnangusse, võib öelda, et antud nõrkus ei ole enam aktuaalne.

- **Erinevate tehnoloogiate ja tööriistade efektiivne kasutamise võimalus; Manuaaltestide automatiseerimise võimalus; Protsesside efektiivsem juhtimine –** antud võimalused on autori hinnangul praegu tiimis kasutusele võetud. Tehnoloogiaid ja tööriistu, mis tiimis on valitud, kasutatakse edaspidi efektiivsemalt. Manuaalteste automatiseeritakse süstemaatiliselt ning testimisprotsessid on süstematiseeritud.

- **Ühe kvaliteediinseneri tiimist lahkumisel ei suuda teine üksi õigeaegselt regressioontestimist teha; Töömahu suurenemisest tulenev ressursipuudus; Tiimis vähe testimisalaseid teadmisi, sh. sügavaid automaattestimise teadmisi ning kogemusi –** autor on teinud tiimile ettepaneku koos teste refaktoreerida ning tutvuda koos testiplaaniga. Samuti tehakse mitmekesi uute funktsionaalsuste testimissessioone. Kuna vahepeal on tiimist lahkunud üks kvaliteediinsener ning vaid üks arendaja, ei ole teise tiimi jäänud kvaliteediinseneri töökoormus tasakaalus tiimi arendajate arvuga. Seega tuleb regressioontestimisel kasutada ka arendajate, disainerite ja tooteomaniku ressursse, et tagada regressioontestimise õigeaegsus. Testimisprotsesside, sh automaattestimise teadmiste edendamiseks on töö autor pidevalt konsulteerinud tiimiväliste spetsialistidega, kes on samuti aidanud kaasa tiimi protsesside parendamisele.

## 5.3 Kvalitatiivsed ja kvantitatiivsed mõõdikud, mille alusel projekti edukust mõõdetakse

Töö algul loodud mõõdikutele, mille alusel projekti edukust mõõdetakse, tehti nii töö alustamise kui ka lõpetamise ajal hinnangulised mõõtmised. Samuti püstitas autor hinnangulised eesmärgid vastavatele mõõdikutele.

Tabel 4 illustreerib mõõdikute algseid keskmisi mõõtetulemusi, praeguseid keskmisi mõõtetulemusi ning tulevikueesmärgi keskmisi mõõtetulemusi. Samuti on tabelis toodud vastavalt mõõdikule planeeritav kokkuhoid ajas.

Tabel 4. Kvalitatiivsed ja kvantitatiivsed mõõdikud koos mõõtetulemustega (allikas: autori koostatud)

| **Mõõdik** | **AS-IS mõõtetulemus** | **Praegune mõõtetulemus** | **TO-BE eesmärk** |
|---|---|---|---|
| 1. Ühe funktsionaalsuse manuaaltestide muutmise keskmine aeg | 5h | 1h | 0,5h |
| 2. Ühe funktsionaalsuse automaattestide muutmise keskmine aeg | 3h | 2h | 0,5h |
| 3. Kattuvate manuaaltestide arv kogu testide arvust | 40% | 20% | 5% |
| 4. Automaattestide katvuse ülevaade | puudulik | rahuldav | hea |
| 5. Regressioontestiplaani kokkupanekule kuluv aeg | 8h | 2h | 1h |
| 6. Kogu manuaalse testiplaani läbimiseks kuluv aeg | 16h | 13h | 8h |
| 7. Automaattestide käivitamiseks kuluv aeg | 117 testi / 90 min (1,30 min/test) | 79 testi / 59 min (1,34 min/test) | 1,3 min/test |
| 8. Testiraportite kättesaadavuse lihtsus ja raportite täpsus ning ajakohasus | olematu | regulaarne, lihtsasti kättesaadav | regulaarne, lihtsasti kättesaadav, kiiresti analüüsitav |

1. **Ühe funktsionaalsuse manuaaltestide muutmise aeg:** Autori hinnangul on manuaaltestide kirjutamine kiirem ja lihtsam, kuna uue plaani kohaselt ühendatakse vana testlugu vajadusel uue kasutuslooga ning teste täiendatakse või muudetakse vastavalt. Samuti on testide leidmine kiire ja lihtne, kuna testid on funktsionaalsuste kaupa kaustadesse jagatud. Kuna iga kasutuslooga

ei pea uusi manuaalteste kirjutama, vaid saab täiendada ka olemasolevaid, siis hoitakse selle tegevuse pealt aega kokku. Hinnanguliste mõõtetulemuste käigus oli algne ühe funktsionaalsuse manuaaltestide muutmise keskmine ajakulu 5 tundi, hetkeolukorra mõõtetulemus 1h, eesmärk on saada konkreetne mõõtetulemus 0,5 tunni peale.

2. **Ühe funktsionaalsuse automaattestide muutmise keskmine aeg:** Autori arvates on tulevase arhitektuuriga võimalik automaattestide kirjutamisel ning hiljem vajadusel nende muutmisel suuresti tööaega kokku hoida. Kui mitmed testid kasutavad samu samme, siis luuakse abimeetod, mis kõiki neid samme sisaldab. Testid jäävad selle võrra sammude arvu poolest lühemaks. Seega sama kategooria funktsionaalsuse testide muutmisel on vaja muudatus teha vaid ühes abimeetodis ning nii hoitakse aega kokku paljude testide muutmise pealt. Hinnanguliste mõõtetulemuste käigus oli algne ühe funktsionaalsuse automaattestide muutmise keskmine ajakulu 3 tundi, hetkeolukorra mõõtetulemus 2h, eesmärk on saada konkreetne mõõtetulemus 0,5 tunni peale.

3. **Kattuvate manuaaltestide arv kogu testide arvust:** Manuaaltestide hulgas on nii suuremas kui ka väiksemas osas kattuvaid teste, mida on võimalik optimiseerida nii, et testide arv väheneb, kuid testide funktsionaalsus jääb samaks. Autor on täielikult ja osaliselt kattuvate manuaaltestide arvu hindamisel lähtunud mitmete erinevate funktsionaalsuse testidest ning on hinnanud, et osaliselt kattuvate testide arv on keskmiselt 40%. Praegu on vähemalt osaliselt kattuvate testide arv hinnanguliselt 20%, kuna osa funktsionaalsusi ja teste on juba läbi käidud ning parandatud. Eesmärk on saada kattuvate testide arv võimalikult väikseks, hinnanguliselt 5%, kuna üldse mitte kattuvaid teste ei ole võimalik mõistliku ajakuluga saavutada.

4. **Automaattestide katvuse ülevaade:** Autori hinnangul on automaattestide katvus algse mõõtmise hetkel puudulik, kuna automaattestide kirjutamisel ei ole neid manuaaltestidega seotud. Praeguse mõõtmise käigus hindab autor automaattestide katvuse ülevaadet rahuldavaks, kuna on tehtud vastavad muudatused nii Jiras manuaaltestides, kus automatiseeritud testid on eraldi märgitud kui ka automaattestides, kus on täpselt näha, millise Jira manuaaltesti numbriga see kattub. Autori hinnangul on loodud lahenduse

kasutamise korral parem ülevaade, millised testid on automatiseeritud ja millised vajavad automatiseerimist. Kuna kõiki teste pole veel läbi jõutud käia, pole veel eesmärki „hea“ saavutatud.

5. **Regressioontestiplaani kokkupanekule kuluv aeg:** Autor on hinnanud algse regressioontestiplaani kokkupaneku ajaks umbes 8 tundi, kus iga funktsionaalsuse kasutuslood tuli üles otsida ning nendega seotud testid ükshaaval testiplaani lisada. Testilugude testiplaani lisamisel tehti testidest kloonid. Praegu on teste võimalik testiplaani lisada kõik testid korraga, suuremate ja väiksemate funktsionaalsuste kaupa, vajadusel ka ükshaaval. Konkreetsete funktsionaalsuste kaupa lisades ei pea teste ükshaaval regressioontestiplaani lisamiseks kloonima. Praegu võtab testiplaani kokku panemine hinnanguliselt 2 tundi aega, kuna testid on funktsionaalsuste kaupa sorteeritud ning kasutuslugudele vastavaid teste on kerge üles leida. Hetkel võtab testiplaani kokku panemine veel eesmärgist kauem aega, eesmärk on regressioontestiplaani kokku panemiseks kuluv aeg lühendada umbes 1 tunni peale.

6. **Kogu manuaalse testiplaani läbimiseks kuluv aeg:** Autori hindab, et tiimis on regressioonitestimine kiirem, kuna manuaaltestid on lühemalt ja arusaadavamalt, ilma kordusteta kirjutatud, seega on nende lugemine ja seega läbimine kiirem. Samuti on testide arv tulevikus väiksem ning ühes testis testitakse mitmeid vajalikke rakenduse osi, mida enne tehti jupiti erinevates testides. Autor on mõõtmiseks läbinud ühe funktsionaalsuse regressioonitestid ning teinud vastavad arvutused, et saada hinnanguline terve testiplaani läbimise aeg. Seega on algse mõõtmise tulemusel regressioonitestiplaani läbimise aeg 16 tundi. Esmaste testide refaktoreerimisel on ühe funktsionaalsuse testide arv jäänud kuni 50% vähemaks ehk näiteks algse 30 testi asemel on samal funktsionaalsusel 15 testi. Seega võib selle info põhjal hinnata, et praeguse seisuga, kui funktsionaalsuste arv jääb samas, on regressioontestiplaani läbimise aeg sarnaselt kuni 50% kiirem, hinnanguline aeg 8 tundi. Praeguseks on osa teste juba ümber kirjutatud, seega võib hinnata kõikide testide läbimiseks aega umbes 13 tundi. Kuna aga funktsionaalsuste arv rakendustes kasvab pidevalt, ei ole võimalik regressioontestiplaani läbimise aega täpselt ennustada.

7. **Automaattestide käivitamiseks kuluv aeg:** Automaattestide käivitamine on alghetkel 117 testi korral 90 min, mis tähendab, et ühe testi käivitamine võtab umbes 1,30 min. Arvesse tuleb võtta, et suur osa teste ebaõnnestuvad ning seega on ka nende käivitamise aeg selle võrra pikem, kui test mõnda elementi veebilehel ootab. Hetkeseisuga on testide käivitamine 79 testi puhul 59 minutit, mis on 1,34 minutit ühe testi kohta. Et testi ebaõnnestumise korral olla kindel, et ebaõnnestumine ei olnud juhuslik, on nüüd tehtud muudatus, et seda käivitatakse automaatselt veel kaks korda ning testiraportisse kuvatakse testi keskmine tulemus. Seetõttu võtab praegu testide komplekti käivitamine kokkuvõttes rohkem aega, kuid testide tulemused on täpsemad ja korrektsemad ning ei kulutada liialt aega valede testitulemuste analüüsimisele. Automaattestide käivitamiseks kuluv aeg on hetkel autori hinnangul liiga pikk, kuid autor soovitab panna testid paralleelselt jooksma, mis tähendab, et kogu automaattestiplaani käivitamine võtab vähem aega. Testid käivitatakse korraga mitmes veebilehitsejas, seega on võimalik testide käivitamise aega mitmekordselt lühendada. [29] Eesmärk on ühe testi käivitamiseks aega mitte pikendada, vaid see peaks jääma kuni 1,3 minutit ühe testi kohta.

8. **Testiraportite kättesaadavuse lihtsus ja raportite täpsus ning ajakohasus:** Automaattestide käivitamine ja raportite koostamine on automaatne ning regulaarne, seega saab tiim rakenduse kohta kiiret ülevaadet, mis seisus see on. Praegu käivituvad automaattestid vahetult enne tööpäeva, seega on iga päev võimalik saada rakenduse kohta adekvaatset infot, mis aitab nii kvaliteediinseneridel, arendajatel kui ka tooteomanikul hinnata toote kvaliteeti ja parandamist vajavaid kohti.

# 6 Hetkeolukord ja edasised tegevused

Antud peatükis annab autor ülevaate praegustest tiimi protsessidest, kokkulepetest ja tehtud muudatustest. Samuti annab autor soovitused edasisteks tegevusteks ning tulevikuvaateks.

## 6.1 Hetkeolukord

Manuaaltestide, testiplaani ja automaattestide haldamise eest vastutavad edaspidi peamiselt kvaliteediinsenerid, kuid samuti on edasine eesmärk ka teistele tiimiliikmetele, nii arendajatele, tooteomanikule kui ka disaineritele tööpõhimõtteid selgitada ja tutvustada, et siis vajadusel regressioonitestimisel või automaattestide muutmise vajadusel oskaksid ka vastavate oskustega inimesed kvaliteedi tagamisega seotuid ülesandeid kanda. Näiteks regressioontestimisel väga suure testide arvu ja platvormide korral on hea, kui tiimis leidub peale kvaliteediinseneri(de) veel inimesi, kes oskavad testiloo järgi testisamme testitavas rakenduses läbi viia ning vigu raporteerida.

Manuaaltestide sorteerimine vastavatesse kaustadesse on tehtud ning testilugude analüüsi ja ümberkirjutamisega on alustatud, kaasates vahepeal disainereid ja arendajaid. Samuti on autor teinud mõndade tiimiliikmetega ükshaaval testiplaani refaktoreerimist ehk on käinud järjest mõne funktsionaalsuse teste üle, neid parandanud ja täiendanud ning kattuvaid teste vähendanud. Seda tegevust plaanib autor koos teiste tiimiliikmetega jätkata. Samuti on töö autor tutvustanud tiimile regressioontestimise põhimõtteid ja saab edaspidi loota selles tiimi abile, kuna neil on vastav arusaam, kuidas regressioontestimise protsess välja näeb.

Praegu on arendajate ja disaineritega läbi viidud testimissessioone, kus umbes 1-1,5 tunni jooksul on testitud ühte uut arendatud funktsionaalsust. Samuti on tiimi töö- ja testimisprotsesside parendamisele kaasa aidanud tiimi *scrum master*, kelle abiga on katsetatud erinevad uurivtestimise viise, näiteks ajastatud sessioonid koos arendaja, disaineri ja kvaliteediinseneriga. Tiim soovib ka tulevikus uurivtestimise sessioone jätkata ja rakendada igapäevatöös, kuna sel viisil saab erinevate rollidega koos testides kohe arutada tekkinud küsimusi ning vigade märkamisel nende prioriteete ja parandamise võimalikkust hinnata.

Automaattestide refaktoreerimine ehk ümber kirjutamine on hetkel pooleli ning ümber on kirjutatud mõned tähtsamad testid. Kõik uued testid kirjutatakse lähtudes uuest arhitektuurist ja koodidisainist. Ümber kirjutamine vajab aega, et kõik testid uuele arhitektuurile viia, kuid juba praegu on näha, et uute testide kirjutamine läheb lihtsamini ja kiiremini kui eelneva arhitektuuri korral. Testide muutmiseks on vaja enamasti muuta vaid testisamme või testisammu sisendeid, kuid ei ole vaja muuta meetodeid või elementide identifikaatoreid.

Kuna magistritöö alustamise hetkel ebaõnnestusid automaattestide käivitamisel peaaegu pooled testid, on teinud autor ettepaneku need kas esimesel võimalusel parandada või märkida tulevikus parandamise jaoks need vastava sildiga, mille abil on neid võimalik igapäevasest automaatsest käivitamisest eemaldada. Hetkel on enamus ebaõnnestuvaid teste märgitud vastava sildiga, mis eemaldab need igapäevasest automaatkäivitusest. Ebaõnnestunud testide parandamine on praegu üks tiimi prioriteete ning sellega tegeletakse.

Automaattestide käivitamine on automaatne ja regulaarne, samuti saab tiim regulaarselt Slack'i kaudu testiraporteid, mille ebaõnnestunud tulemusi hommikuste kohtumiste või kõnede ajal arutatakse. Võrreldes algse olukorraga, kus automaatteste ei käivitatud automaatselt, testiraporteid ei tekkinud ning polnud nähtav testiraportite ajalugu, on praegune olukord tunduvalt parem.

Tiimis vähemalt kahe kvaliteediinseneri korral jätkatakse peale tavapäraste kvaliteedi tagamise tööülesannetega ka testide ülevaatusega ning manuaaltestide automatiseerimisega. Kasutajaliidese muutumisel ning testide muutmise vajaduse korral võivad automaatteste muuta kokkuleppel nii testijad kui ka arendajad, sealhulgas tuleb silmas pidada, et automaattesti muutmisel muudetaks ka aluseks võetud Jira testilugu ja vastupidi - juba automatiseeritud manuaalse testiloo muutmisel tuleb muuta ka vastav kasutajaliidese automaattest.

Juhul kui tiimis on vaid üks kvaliteediinsener, võivad manuaalteste üle vaadata teised tiimiliikmed, kuigi seda peaksid reeglina tegema kvaliteediinsenerid. Vajadusel võib ainus kvaliteediinsener ka ise teste kinnitada enda töö paremaks jälgimiseks või arendajatele testi automatiseerimiseks andmiseks. Kui ainsal kvaliteediinseneril tiimis ei jätku kasutuslugude, arenduste ja disainide analüüsimise, manuaaltestimise, testide

kirjutamise ning testiplaani haldamise kõrval aega automaattestide kirjutamise ja haldamisega tegeleda, oleks siiski soovitatav, et ta osaleks automaattestide üle vaatamisel, kui neid kirjutavad tema asemel näiteks arendajad. Samuti on oluline, et kvaliteediinsener annaks automaattestide kirjutamiseks sisendi ehk suunaks just need testilood arendajatele automatiseerimiseks, millele ta on omapoolse prioriteetsuse määranud.

Kui tiimis puudub konkreetne kvaliteediinseneri rolli kandev inimene, on siiski võimalik eelnevalt töös kirjeldatud praktikaid kasutada, nende praktikate rakendamiseks on vaja lihtsalt teistel tiimis olevatel rollidel kanda kvaliteediinseneri tööülesandeid. Näiteks on võimalik ka arendajatel, disaineritel või tooteomanikul täita vähemalt osa kvaliteediinseneri ülesandeid, kui nad on läbinud vastava juhendamise ning saavad aru kvaliteedi tagamise põhitõdedest ning oskavad neid ka oma töös rakendada.

## 6.2 Edasised tegevused

Töö autor jätkab arendustiimile töö- ja testimisprotsesside parendamisettepanekute tegemisega ning tiimile levinumate testimisprotsesside ja -praktikate tutvustamist. Samuti peab töö autor oluliseks ka seda, et tööprotsesse parendatakse pidevalt ning leitakse konkreetse tiimi ja arendatava projekti jaoks parimad lahendused, toetudes parimatele levinud praktikatele, mis on ennast paljudes agiilsetes tiimides tõestanud.

Automaattestide analüüsi käigus märkas autor, et kasutajaliidese objektide identifikaatorite ehk *data-testid* nimetamisel on kasutatud erinevaid stiile, kuid lähtuda võiks pigem ühtsest stiilist. Kuna enamus identifikaatoreid kasutab juba *kebab-case* [30] stiilis nimetamist, oleks hea ka edaspidisel nimetamisel just seda stiili järgida. *Kebab-case* kujutab endast väikeste tähtedega omavahel sidekriipsuga eraldatud nimetusviisi, näiteks *invoice-details*.

Autor on kliendiülese tarkvaraprojektide edendamise töögrupiga koostöös arutanud erinevate tasemete automaattestide läbipaistvuse suurendamist, mis tähendab, et plaanitakse vähendada kasutajaliidese automaattestide taseme mahukust. Kuna kasutajaliidese testide läbimine on üsna aeglane ja seda tehakse pigem rakenduse arendustsükli lõpus, peaks automaattestimist viima võimalikult madalale tasemele. Töögrupp leiab, et peaks keskenduma rohkem ühiktestide ja integratsioonitestide katvuse

parendamisele ning muutma nende testide katvus läbipaistvamaks, et kasutajaliidese testide tasemel ei esineks korduvust. Eesmärk on, et erinevatel tasemetel ei testitaks täpselt samu funktsionaalsusi ning saadaks rakenduse seisukorra informatsioon kiiremini kätte. Seega on vajalik, et erinevate automaattestide tasemete katvus oleks visuaalselt välja toodud.

Edaspidise arendusena oleks autori arvates mõistlik kasutusele võtta ka automaattestide komplekt, mis käivitatakse peale iga uue arenduse lisamist koodi. See annaks arendajale kohe teada, kas tema tehtud arendus on lõhkunud mõne eelneva funktsionaalsuse ära. Seega soovitab autor arenduse käigus eelnevate funktsionaalsuste muutmisel ka automaattestid ära muuta, et need läbitaks endiselt edukalt. Automaattestide muutmisel tuleb muuta ka vastavad manuaaltestid, mis on Jiras kirjeldatud.

Autori üks edaspidisest ülesandest on testimisse arendajaid ja disainereid kaasata, nendele testimisprotsesse ja praktikaid tutvustada. Autori edasine eesmärk on ka teistele kliendiprojektidele tiimi testimisstrateegiat ja -praktikaid tutvustada ning seeläbi aidata ka neil enda töö- ja testimisprotsesse parendada.

# Kokkuvõte

Proeksperdi eesmärk on olla oma klientidele hinnatud tarkvara- ja äriarenduspartner. Tarkvararakenduse kvaliteedi tagamiseks on oluline rakenduse süstematiseeritud testimine. Samuti on äritegevuste planeerimisel oluline tulevaste tööde võimalikult täpne ennustatavus.

Käesolevas töös analüüsiti ning parendati info- ja telekommunikatsiooni kuuluva kliendi iseteeninduspordaali arendava tiimi töö- ja testimisprotsesse. Analüüsi alustamisel olid projektitiimi testimisprotsessid autori hinnangul ebasüsteemsed ning läbimõtlemata.

Töö tulemusena valmis TO-BE vaade testimisprotsessidest, manuaaltestide kirjutamisest ja haldamisest, automaattestide kirjutamisest, haldamisest ja refaktoreerimisest. Töö käigus koostati testimistegevuste süsteem, kus manuaaltestid kirjutatakse ja täiendatakse kasutuslugude alusel ning need omakorda automatiseeritakse vajadusel. Automaattestide kirjutamise ja haldamise jaoks loodi uus testide arhitektuur, mille haldamine on eelnevaga võrreldes süsteemsem ja ülevaatlikum.

Esimeses peatükis püstitati magistritöö eesmärk ja määratleti skoop.

Teine peatükk andis ülevaate valitud meetoditest ning selles analüüsiti kliendiprojekti töö- ja testimisprotsesse.

Kolmandas peatükis analüüsiti parimaid levinumaid testimispraktikaid ja tiimis kasutusel olevate tööriistade juhendeid.

Neljas peatükk keskendus projekti töö- ja testimisprotsesside parendamisele, kus autor lõi kliendiprojektile testimisstrateegia.

Viiendas peatükis keskenduti analüüsitulemuste järeldustele, anti hinnang tehtud tööle ning konkreetsed ettepanekud protsesside parendamiseks.

Kuuendas peatükis kirjeldas autor kliendiprojekti tulevikuplaane protsesside parendamiseks.

Töö autor on teinud tarkvaraarendustiimile mitmeid parendusettepanekuid ning osa neist ka juba koos kolleegide abiga ellu viinud. Autor plaanib projektitiimi protsesse edasi edendada ning tutvustada kasutusel olevaid protsesse ja praktikaid ka teistele tarkvaraarendustiimidele.

Autor lähtus magistritöö efektiivsuse hindamisel töö alguses püstitatud mõõdikutest, millele tehti nii alg- kui ka lõpustaadiumis mõõtmised. Töö tulemusena on projektitiimi töö- ja testimisprotsessid süstematiseeritud ning mõõtmistulemuste alusel saab väita, et muudatused protsessides on muutnud projektitiimi efektiivsemaks ning tegevused läbipaistvamaks. Samuti aitavad protsessimuudatused arendustiimi planeeritavate tööde ennustatavust parandada.

Magistritöös püstitatud eesmärgid on saavutatud.

# Kasutatud kirjandus


[1] International Software Testing Qualifications Board, „Advanced Level Agile Technical Tester,“ [Võrgumaterjal]. Available: https://www.istqb.org/downloads/send/65-advanced-level-agile-technical-tester/289-istqb-ctal-att_syllabus_v1-0.html. [Kasutatud 07. 03. 2021].

[2] International Institute of Business Analysis, BABOK -A Guide to the Business Analysis Body of Knowledge, Toronto, 2005.

[3] Atlassian, „DevOps,“ [Võrgumaterjal]. Available: https://www.atlassian.com/devops. [Kasutatud 23 05 2021].

[4] ThoughtWorks, „Continuous Integration,“ [Võrgumaterjal]. Available: https://www.thoughtworks.com/continuous-integration. [Kasutatud 22. 03. 2021].

[5] International Software Testing Qualifications Board, „Certified Tester: Foundation Level Syllabus,“ [Võrgumaterjal]. Available: https://www.istqb.org/downloads/send/2-foundation-level-documents/281-istqb-ctfl-syllabus-2018-v3-1.html. [Kasutatud 21. 03. 2021].

[6] „Two Measures of Development Effectiveness: Predictability and Optimization,“ [Võrgumaterjal]. Available: https://www.agileconnection.com/article/two-measures-development-effectiveness-predictability-and-optimization. [Kasutatud 11. 05. 2021].

[7] „Software Test Estimation Techniques,“ [Võrgumaterjal]. Available: https://www.softwaretestinghelp.com/software-test-estimation-how-to-estimate-testing-time-accurately/. [Kasutatud 03. 05. 2021].

[8] „Predictive software engineering: the ultimate way to deliver working software,“ [Võrgumaterjal]. Available: https://www.cio.com/article/3252065/predictive-software-engineering-the-ultimate-way-to-deliver-working-software.html. [Kasutatud 11. 05. 2021].

[9] L. Crispin ja J. Gregory, Agile Testing: a practical guide for testers and agile teams, Pearson Education, Inc., 2009.

[10] „Inforegister,“ [Võrgumaterjal]. Available: https://www.inforegister.ee/10331577-PROEKSPERT-AS. [Kasutatud 05. 03. 2021].

[11] „Proekspert,“ [Võrgumaterjal]. Available: https://proekspert.com/blog/proekspert-news/proekspert-wins-equal-pay-award/. [Kasutatud 05. 03. 2021].

[12] M. Fowler, „Refactoring,“ [Võrgumaterjal]. Available: https://refactoring.com/. [Kasutatud 03. 05. 2021].

[13] Devexpress, „TestCafe,“ [Võrgumaterjal]. Available: https://devexpress.github.io/testcafe/. [Kasutatud 21. 03. 2021].

[14] „TestCafe Implementation in DANA- Part 1 : Page Object Model,“ [Võrgumaterjal]. Available: https://medium.com/dana-engineering/testcafe-implementation-in-dana-part-1-page-object-model-29751b63d40e. [Kasutatud 04. 04. 2021].

[15] N. S. Potter ja M. E. Sakry, Making Process Improvement Work, Pearson Education, Inc., 2002.

[16] „SIPOC - A Great Tool for Process Analysis in Six Sigma,“ [Võrgumaterjal]. Available: https://www.edrawsoft.com/sipoc-process-sixsigma.html. [Kasutatud 03. 05. 2021].

[17] „Software Testing Metrics & KPIs,“ [Võrgumaterjal]. Available: https://www.thinksys.com/qa-testing/software-testing-metrics-kpis/. [Kasutatud 19. 05. 2021].

[18] Agile Alliance, „Three Amigos,“ [Võrgumaterjal]. Available: https://www.agilealliance.org/glossary/three-amigos/. [Kasutatud 05 03 2021].

[19] „Software Test Estimation Techniques: Step By Step Guide,“ [Võrgumaterjal]. Available: https://www.guru99.com/an-expert-view-on-test-estimation.html. [Kasutatud 03. 05. 2021].

[20] C. Kaner, J. Falk ja H. Q. Nguyen, Testing Computer Software, Wiley Computer Publishing, 1999.

[21] „Deviniti,“ [Võrgumaterjal]. Available: https://deviniti.com/support/addon/server/testflo-86/latest/test-repository/. [Kasutatud 05. 03. 2021].

[22] Guru99, „Gherkin Language: Format, Syntax & Gherkin Test in Cucumber,“ [Võrgumaterjal]. Available: https://www.guru99.com/gherkin-test-cucumber.html. [Kasutatud 06. 03. 2021].

[23] SmartBear, „Cucumber,“ [Võrgumaterjal]. Available: https://cucumber.io/docs/cucumber/. [Kasutatud 05. 03. 2021].

[24] International Software Testing Qualifications Board, „Certified Tester Advanced Level Syllabus: Test Automation Engineer,“ [Võrgumaterjal]. Available: https://www.istqb.org/downloads/send/48-advanced-level-test-automation-engineer-documents/201-advanced-test-automation-engineer-syllabus-ga-2016.html. [Kasutatud 22. 03. 2021].

[25] SmartBear, „What is Automated Testing?,“ [Võrgumaterjal]. Available: https://smartbear.com/learn/automated-testing/what-is-automated-testing/. [Kasutatud 05. 03. 2021].

[26] Dev Tester, „5 Ways TestCafe Helps Your Tests Run Fast,“ [Võrgumaterjal]. Available: https://dev-tester.com/5-ways-testcafe-helps-your-tests-run-fast/. [Kasutatud 05. 03. 2021].

[27] Jenkins, „Slack Notification,“ [Võrgumaterjal]. Available: https://plugins.jenkins.io/slack/. [Kasutatud 22. 03. 2021].

[28] „Make a README,“ [Võrgumaterjal]. Available: https://www.makeareadme.com/. [Kasutatud 11. 05. 2021].

[29] „TestCafe, Run Tests,“ [Võrgumaterjal]. Available: https://testcafe.io/documentation/402830/guides/basic-guides/run-tests. [Kasutatud 18 05 2021].

[30] „Kebab case,“ [Võrgumaterjal]. Available: https://www.theserverside.com/definition/Kebab-case. [Kasutatud 17. 05. 2021].

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

Mina, Marilyn Võsu

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Töö- ja testimisprotsesside parendamine kliendiprojektis ettevõtte Proekspert AS näitel“, mille juhendaja on Nadežda Furs ning kaasjuhendaja Ketlin Saksakulm.
   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.

20.05.2021

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.