TALLINNA TEHNIKAÜLIKOOL

Infotehnoloogia teaduskond

Nikita Kums 179097IADB

Kasutajaõiguste testimise automatiseerimine ettevõttes Fujitsu Estonia AS

Bakalaureusetöö

Juhendaja: Aleksei Talisainen
MSc
Kaasjuhendaja: Dmitri Lepp
PhD

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: Nikita Kums

15.05.2021

Annotatsioon

Käesolev bakalaureusetöö lahendab aktuaalset probleemi AS Fujitsu Estonia Documentum meeskonnas, kus antud meeskonna poolt arendatava projekti kasutajaõiguste testimine on teostatud manuaalselt, mis oma mahukuse poolest ei võimalda manuaaltestimise abil katta kõiki testimist vajavaid kasutajaõiguste poolt mõjutatud rakenduse osad. Antud probleem esineb juba ühe organisatsiooni jaoks seadistatud rakenduse korral, kuid selliseid, erinevalt seadistatud, rakendusi on meeskonna käsitluses kümneid. Selle tulemusena ei rahulda tarnitav tarkvaralahendus klientide nõudmisi ning kannatab tarkvaralahenduse kvaliteet.

Lähtudes antud probleemidest, on käesoleva bakalaureusetöö eesmärgiks automatiseerida kasutajaõiguste testimine, kasutades selleks analüüsi tulemustel valitud tööriistu. Kuigi lõputöö käigus valminud rakendus ei olnud veel lõplik, osutus antud automaattestimissüsteem suurt abi pakkuvaks tööriistaks kasutajaõiguste testimisel. Selle tulemusena vähenes kasutajaõiguste testimise peale kulutatav aeg, suurenes tarkvaralahenduse testidega kaetus ning tõusis tarkvaralahenduse kvaliteet.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 35 leheküljel, 7 peatükki, 20 joonist, 5 tabelit.

Abstract

Automation of User Rights Testing in Fujitsu Estonia AS

This bachelor's thesis solves an existing problem in AS Fujitsu Estonia Documentum team where the user rights testing of the project developed by the given team is performed manually, which, due to it's scope, does not allow manual testing to cover all parts of the project affected by user rights that need to be tested. Given problem already occurs with the project configured for one organization whereas there are dozens of such configurations for different organizations at the team's disposal. As a result, the quality of the software delivered suffers and does not meet customer's requirements.

Based on this problem, the aim of this thesis is to automate user rights testing using tools selected during analysis. Although the final user rights automation application created during the thesis was not yet finalized, it provided great help in user rights testing. As a result, the time spent testing user rights decreased, software's coverage by automated tests was increased, further increasing the quality of the software and taking a step towards satisfying customer's requirements towards user rights.

The thesis, first, introduces the project under development by the Documentum team to allow further explanation of the user rights testing problem. It then describes the test automation process which is later used in the fourth chapter which focuses on planning and implementing the user rights test automation application. Before implementation, right tools are selected by analysing which of them satisfy the most needs of the application. Fifth chapter describes the developed user rights test automation application. Sixth, being the last chapter, compares user rights test automation application possibilities with previous, manual, testing and formulates further plans.

The thesis is in Estonian and contains 35 pages of text, 7 chapters, 20 figures, 5 tables.

Lühendite ja mõistete sõnastik

Annotatsioon	*Annotation* - märksõna, mis kujutab endast metaandmeid, tähistades mingit lisainformatsiooni [1]
API	*Application Programming Interface* - rakendusliides, mis võimaldab teisel programmil esimese teenuseid kasutada [2]
Code-review	Koodi ülevaade. Kaasprogrammeerija poolt muudetud koodi ülevaatamine veendumaks, et tehtud muudatused ei too endaga kaasa vigu programmi [3]
Docker	Tööriist loodud selleks, et hõlbustada rakenduste loomise, juurutamise ja käivitamise, kasutades selleks konteinereid [4]
Identifikatsioon ehk id	Miski, mis võimaldab tuvastada midagi või kedagi
JSON	*JavaScript Object Notation* - formaat andmete esitamiseks, salvestamiseks ja andmevahetuseks [5]
Kasutajaõigused	Kasutajale antavad õigused, mis võimaldavad või piiravad antud kasutaja võimalust pääseda ligi teatud ressurssidele [6]
Metaandmed	Andmed teiste andmete kohta, mis lihtsustavad antud teiste andmete leidmise ja nendega töötamise. Näiteks: autor, loomise kuupäev, muutmise kuupäev, faili suurus
Otspunkt	*Endpoint* - koht, kus API (*Application Programming Interface*) päringute korral teostatakse mõlema programmi vahel andmevahetus [2]
Testjuht	*Test case* - tegevuste kogum, mille käivitamise abil valideeritakse tarkvaralahenduse komponente või funktsionaalsust [7]
Testkeskkond	*Test environment* - keskkond, kus testimine teostatakse [8]
URL	*Uniform Resource Locator* - veebis oleva ressursi unikaalne aadress ehk veebiaadress [9]

Vaadin	Vaadin *framework* ehk Vaadin raamistik, mis lihtsustab Java veebirakenduste arendamise ja haldamise [10]
Valideerimine	Millegi õigsuse kontrollimine
XML	*Extensible Markup Language* - formaat andmete esitamiseks, salvestamiseks ja andmevahetuseks [5]

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Tarkvaramaailmas on kahte liiki testimisi – käsitsi testimine ja automatiseeritud testimine. Mõni käsitsi testimise liik, näiteks avastav testimine ja kasutatavuse testimine, on hindamatu väärtusega. Käsitsi saab teostada ka teisi liike teste, kuid inimeste jaoks on üsna raiskav tava sama asja korduvalt teha. Samade testide kordamine toob esile antud testide automatiseerimise vajaduse [8].

Ettevõtte Fujitsu Estonia AS arenduses oleva projekti kasutajaõiguste testimine toimub manuaalselt, nõudes mitu kuud tööd, mis ei võimalda teostada antud testimist pidevalt, ei välista inimvea olemasolu testandmete loomisel, testimise sammude läbimisel ega testimise tulemustes. Samuti ei hõlma manuaalne testimine kõikide võimalike kombinatsioonide testimist ega kõiki kliente, kes erinevad teineteisest projekti seadistatavuse poolest. Nimetatud probleemid ei võimalda rahuldada klientide nõudmisi kasutajaõiguste suhtes.

Lähtudes antud probleemidest, on käesoleva bakalaureusetöö eesmärgiks luua kasutajaõiguste automaattestimise rakendus, mis oleks seadistatav vastavalt kliendile ning pidevalt käivitatav. Antud rakendus võimaldaks vabastada testijad mahukast ning ajakulukast tööst, vähendades seejuures inimvea tõenäosust testandmete loomisel ja testimise tulemustes ning tõsta projekti korrektsust ja kattuvust automatiseeritud testidega.

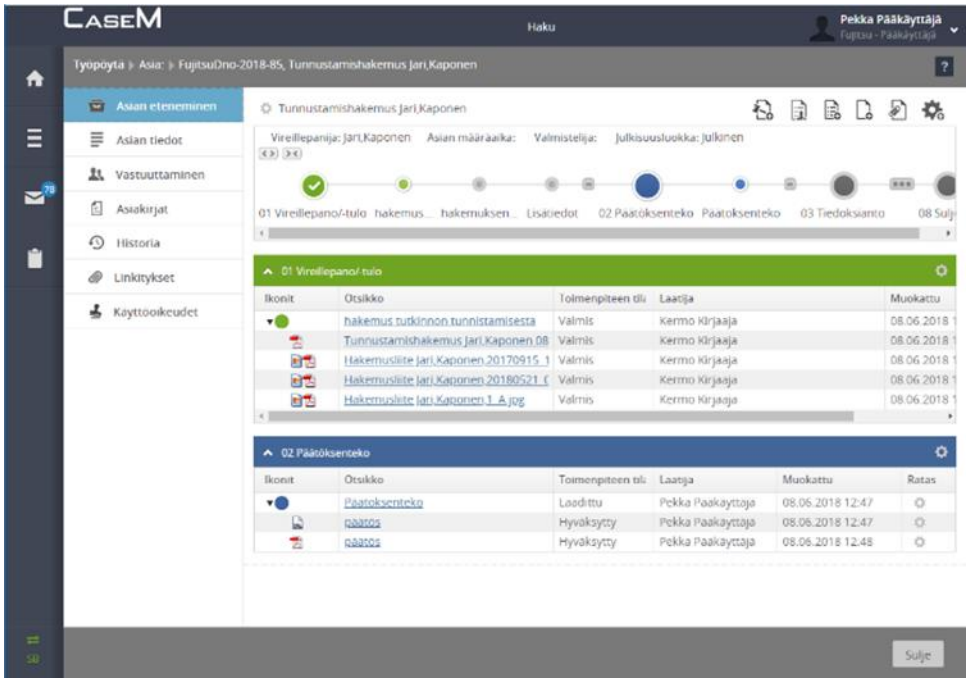
Bakalaureusetöö koosneb viiest osast. Esimeses osas tutvustatakse testitava rakenduse arhitektuuri lõputöö skoobis ning lõputöös käsitletud probleemi täpsemalt. Teises osas tutvustatakse automaattestimise mõistet ning automaattestimise planeerimise metoodikat. Kolmas osa hõlmab endas kasutajaõiguste testimise rakenduse planeerimist, rakendades selleks eelnevalt kirjeldatud metoodikat, mille etappides analüüsitakse võimalike alternatiive. Lõputöö neljandas osas kirjedatakse bakalaureusetöö raames valminud rakendust. Seejärel võrreldakse automaattestimissüsteemi manuaaltestimisega ning sõnastatakse edaspidised plaanid. Konfidentsiaalsuse tagamiseks ei ole lõputöös kasutatud organisatsioonide nimesid.

2 Taust

Antud bakalaureusetöö probleemi põhjalikumaks tutvustamiseks tuleb omada ettekujutust testitava projekti arhitektuurist ning iseärasustest, millest räägivad järgnevad alampeatükid.

2.1 Testitav projekt

Testitavaks projektiks on Fujitsu Estonia AS Documentum meeskonna poolt arendatav juhtumi- ja dokumendihalduse rakendus CaseM. CaseM on teabehalduslahendus, mis lisaks juhtumi- ja dokumendihaldusele võimaldab korraldada e-koosolekuid ja koostööd, avaldada ja allkirjastada dokumente, teostada protsesside ja liigitusreeglite juhtimist ning jälgida protsesside kulgemist reaalajas [11]. CaseM sisaldab palju erinevaid objekte, kuid peamisteks ja enimkasutatud objektideks on juhtumid (inglise keeles *cases*), tegevused (inglise keeles *actions*) ja dokumendid (inglise keeles *documents*) (Joonis 1).



Joonis 1. CaseM avatud juhtum, avatud tegevused (roheline ja sinine) ning dokumendid.

CaseM kasutajaliides on loodud kasutades *Vaadin* raamistikku ning tagarakendus kasutades Documentum'it. Documentum on dokumendihaldustarkvara ehk dokumentide

hoidla, mis hõlmab endas dokumentide metaandmete, versioonide ja binaarse kuju hoidmist ning võimaldab antud andmete juurde ligipääsu teiste süsteemikasutajate poolt [12]. CaseM rakendust kasutavad kümneid erinevaid organisatsioone. Organisatsioonide vajaduste rahuldamiseks on rakenduse seadistamine teostatud läbi XML (*Extensible Markup Language*) failide, mis on loodud eraldi iga organisatsiooni jaoks. Antud XML failid võimaldavad lülitada sisse ja välja rakenduse funktsionaalsust, programmikoodi muutmata.

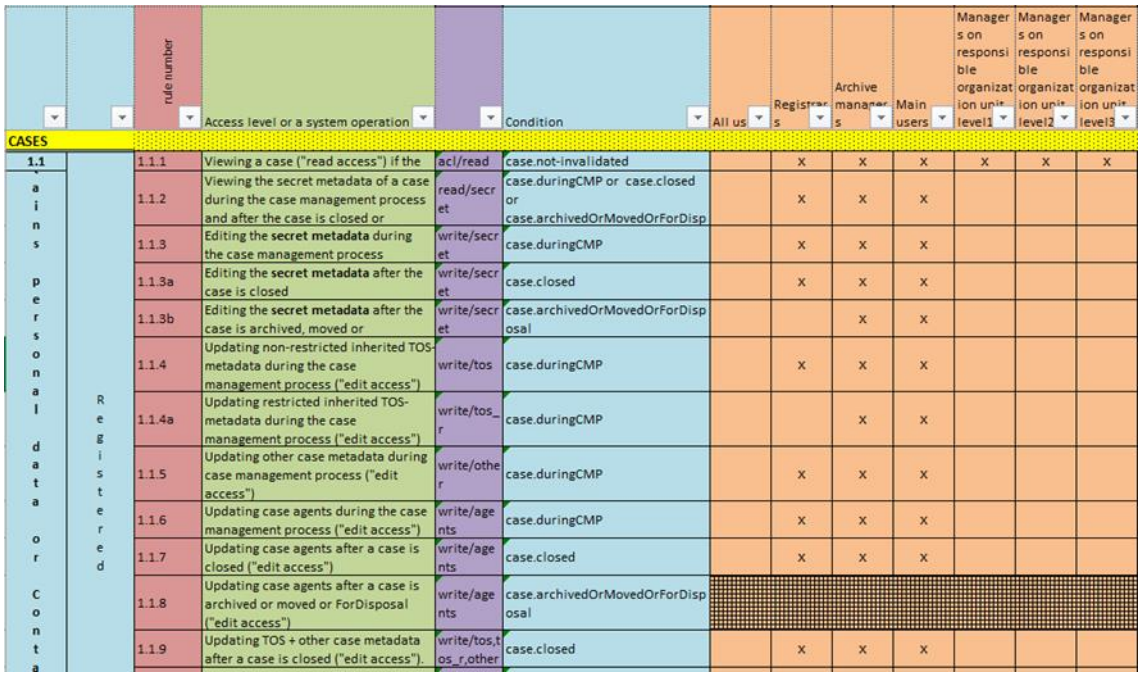
2.2 Projekti kasutajaõigused

CaseM'is antakse kasutajale kasutajaõigusi objektide loomise või muutmise järel sõltuvalt kasutaja rollidest ja gruppidest, kuhu nad kuuluvad. Kasutajaõigused piiravad rakenduse osade juurde ligipääsu või kasutaja tegevusi juhtumitega, tegevustega või dokumentidega. Documentum võimaldab piirata ainult tervete objektide juurde ligipääsu, seega on kasutajaliidese vaadete ja objektide metaandmete eest vastutavate kasutajaliidese osade juurdepääsu piiramine teostatud CaseM koodis. Ligipääsu piiramiseks kasutatakse Documentum'is ACL'e ehk *Access Control Lists*'e. ACL määratakse objekti loomise või arvutatakse uuesti objekti andmete muutmise ajal. ACL sisaldab andmeid kõikidest rollidest ja gruppidest, mis võivad antud objektile ligi pääseda. Documentum'is on kokku seitse erinevat ACL'i (vt Lisa 2), kuid CaseM'is on nendest kasutusel neli: sirvimine, lugemine, kirjutamine ja kustutamine. Rollid jagunevad kaheks: globaalsed rollid, mis antakse kasutajale gruppi kuulumisest tingitult ning rollid, mis leiavad aset objekti kontekstis ning sõltuvad objekti metaandmetest. Kasutaja ligipääsu katsel kontrollitakse, kas kasutaja rollid või grupid on antud objekti ACL'is kirjas ning seejärel antakse kasutajale, vastavalt hierarhiale, kõige kaalukam kasutajaõigus [13]. Näiteks kustutamise õiguse olemasolu korral on kasutajal samuti kirjutamise, lugemise ja sirvimise õigus.

Kõik rollide ja gruppide piirangud kõikide võimalike objektide kombinatsioonide jaoks on määratud rakenduse kasutajaõiguste eest vastutavates XML failides, mis omakorda vastavad eraldiseisvale kasutajaõiguste excel failile, kuhu on dubleeritud XML failide reeglid kasutajasõbralikul kujul. Antud kasutajaõiguste excel faili kasutatakse kliendi poolt rakenduse kasutajaõiguste nõuete ja töökäigu määramiseks.

Kasutajaõiguste excel fail on jagatav tulpade ja ridade kaupa. Parema ülevaate saamiseks on autor tähistanud erinevate funktsioonide eest vastutavad tulbad erinevate värvidega (Joonis 2):

- Sinine – Kõik võimalikud objektide kombinatsioonid;
- Punane – Reeglite numbrid;
- Roheline – Antud reegli kirjeldus;
- Lilla – ACL õigus, millele antud rida vastab;
- Oranž – Kasutajad, erinevates rollides ja gruppides, kus x tulbas tähistab kasutaja õigust antud tehingu või operatsiooni teostamiseks;

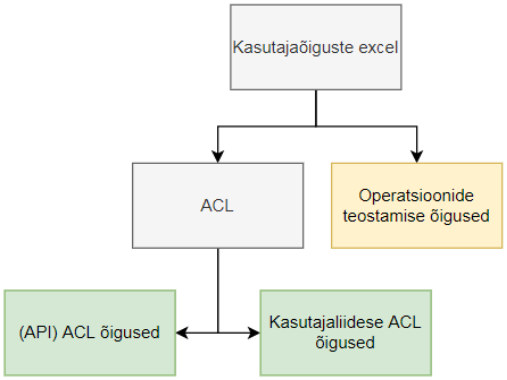


		rule number	Access level or a system operation		Condition	All us	Registrars	Archive managers	Main users	Managers on responsible organization unit level1	Managers on responsible organization unit level2	Managers on responsible organization unit level3
CASES												
1.1	Registered	1.1.1	Viewing a case ("read access") if the	acl/read	case.not-invalidated		x	x	x	x	x	x
		1.1.2	Viewing the secret metadata of a case during the case management process and after the case is closed or	read/secret	case.duringCMP or case.closed or case.archivedOrMovedOrForDisp		x	x	x			
		1.1.3	Editing the **secret metadata** during the case management process	write/secret	case.duringCMP		x	x	x			
		1.1.3a	Editing the **secret metadata** after the case is closed	write/secret	case.closed		x	x	x			
		1.1.3b	Editing the **secret metadata** after the case is archived, moved or	write/secret	case.archivedOrMovedOrForDisposal			x	x			
		1.1.4	Updating non-restricted inherited TOS-metadata during the case management process ("edit access")	write/tos	case.duringCMP		x	x	x			
		1.1.4a	Updating restricted inherited TOS-metadata during the case management process ("edit access")	write/tos_r	case.duringCMP			x	x			
		1.1.5	Updating other case metadata during case management process ("edit access")	write/other	case.duringCMP		x	x	x			
		1.1.6	Updating case agents during the case management process ("edit access")	write/agents	case.duringCMP		x	x	x			
		1.1.7	Updating case agents after a case is closed ("edit access")	write/agents	case.closed		x	x	x			
		1.1.8	Updating case agents after a case is archived or moved or ForDisposal ("edit access")	write/agents	case.archivedOrMovedOrForDisposal							
		1.1.9	Updating TOS + other case metadata after a case is closed ("edit access").	write/tos,tos_r,other	case.closed		x	x	x			

Joonis 2. Kasutajaõiguste exceli osa jaotatud tulpade kaupa.

Ridade kaupa jaguneb kasutajaõiguste excel fail kolmeks osaks (Joonis 3):

1. API põhised ACL õigused;
2. Kasutajaliidese põhised ACL õigused;
3. Operatsioonide teostamise õigused;



Joonis 3. Kasutajaõiguste excel faili jaotus ridade kaupa.

API põhised ACL õigused ja kasutajaliidese ACL õigused moodustavad kasutajaõiguste excelist 83%. Tegelikult on tegu sama kasutajaõiguste exceli osaga, mis mõjutab ja on kontrollitav kahte viisi: läbi API ja läbi kasutajaliidese. Antud osa vastutab ACL tehingute teostamise võimaldamise ning objektide vormide peal väljade nägemise ja muutmise eest. Kasutajaõiguste excel sisaldab keskmisel 410 ACL põhist rida, mille hulk muutub sõltuvalt organisatsioonist.

Operatsioonide teostamise õigused moodustavad kasutajaõiguste excelist 17%. Antud kasutajaõiguste exceli osa vastutab kasutaja õigustest teostada objektide peal operatsioone, näiteks dokumentide allkirjastamine, objekti tööülesandena saatmine teisele kasutajale. Mõned operatsioonid, näiteks objektide loomine, rakenduse seadete muutmine ja otsingu päringute teostamine, ei vaja objekte nende kontrollimiseks. Operatsioonid on kontrollitavad ainult läbi kasutajaliidese. Nagu ka ACL õiguste korral sõltub operatsioonide hulk organisatsioonist, kuid keskmiselt on neid 80 tükki.

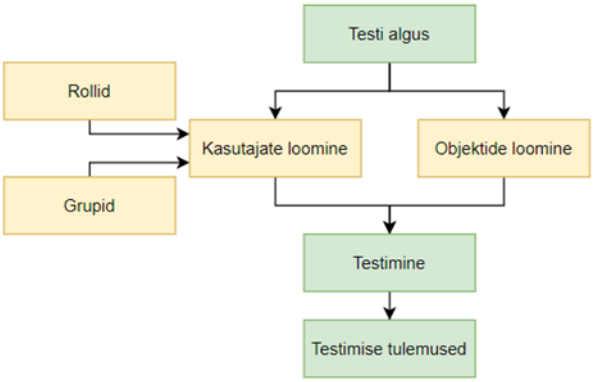
Igat exceli rida võib vaadelda kui eraldi funktsionaalsust. Näiteks võib rida kirjeldada juhtumi andmete muutmist, kuni juhtum pole veel suletud või dokumendi kinnitamise võimaldamist kui dokument ei ole salajane.

2.3 Probleem

Käesoleval hetkel toimub CaseM projekti kasutajaõiguste testimine manuaalselt. Manuaalne testimine ei ole kasutajaõiguste testimise jaoks sobiv lahendus, kuna antud testimist tuleb pidevalt korrata nii leitud vigade parandamise kui ka järjekordse tarnimise korral. Pidev samade kasutusjuhtude kordamine suurendab testija töökoormust, mis

suurendab inimvea tõenäosust testimise tulemustes. Samuti pideva kordamise tagajärjel harjub testija kasutajaliidesega ning võib vead märkamata jätta.

Kasutajaõiguste testimine (Joonis 4) tähendab kasutajaõiguste exceli faili kogu funktsionaalsuse läbiproovimist, kus iga rida kujutab mingit funktsionaalsust. Ehk keskmiselt 410 ACL'i ja 80 operatsiooni eest vastutava rea korral on tarvis teostada 490 erinevat testi, eeldusel, et testitakse ainult ühte kasutajat ning ühte objekti. Juba testandmete loomise etapis ei võimalda manuaalne testimine rahuldada nõutud vajadusi. Testandmete loomine on manuaalselt teostatav küll väikeste koguste korral, näiteks sada erinevat objekti, mille korral manuaalne funktsionaalsuse testimine ei ole juba võimalik, kuid mitmetuhande erineva objekti korral ei ole isegi testandmete loomine manuaalselt teostatav. Samuti ei ole kasutajaõiguste excelis kunagi vähem kui 20 erinevat kasutajat, mis tähendab iga testi kordamist vähemalt 20 korda.



Joonis 4. Kasutajaõiguste testimise protsess.

Nii manuaalsest testimisest tingitud probleemid, vajalik testandmete hulk kui ka erinevate kasutajate hulk kasutajaõiguste testimise teostamiseks ei võimalda rahuldada klientide vajadusi ning vähendab tarkvaralahenduse kvaliteeti.

Käesoleva bakalaureusetöö eesmärgiks on luua kasutajaõiguste automaattestimise rakendus, mis automatiseeriks praegu manuaalselt teostatava kasutajaõiguste testimise protsessi, lahendades antud probleemid.

3 Automaattestimine ja CI/CD

Automaattestimine on tarkvaralahendusest ülevaate saamiseks ning kvaliteedi tagamiseks tarkvaratööriistade rakendamine inimpõhise manuaalse protsessi automatiseerimiseks. Enamus kaasaegseid tarkvaralahendusi sisaldavad automaattestimist juba algusest peale, kuna tarkvaralahenduse manuaalne testimine ei võimalda tänapäeva agiilses keskkonnas rahuldada testimise vajadusi. Ühe enam võetakse kasutusele CI/CD meetodikat [14]. CI/CD ehk pidev integratsioon/pidev edastamine (inglise keeles *Continuous Integration/Continuous Delivery*) on meetod, mis keskendub pidevale tarkvaralahenduse tarnimisele kliendile, tuues automatiseerimise tarkvaralahenduse arendamise etappidesse [15].

Pidev edastamine keskendub klientidele uute funktsionaalsuste tarnimisele nii kiiresti kui võimalik. Antud eesmärgi täitmiseks on automaattestimisel hindamatu väärtus, kuna ei esine võimalust automatiseerida tarnimist, kui esineb manuaalne, palju aega nõudev etapp antud protsessis [14].

Pidev integratsioon on etapp, millest pidev edastamine on sõltuv. Pidev integratsioon vastutab uue funktsionaalsuse automaattestimise eest, kontrollimaks, et tehtud muudatustest ei saanud kannatada juba kasutusel olevad programmi osad ega lisandunud uusi vigu programmi. Pidev edastamine algab, kui pidev integratsioon on läbinud automaattestimise etapi [14].

Antud seos automaattestimise, pideva integratsiooni ja pideva edastamise vahel toob kasu kiirelt arenevatele tarkvaralahendustele ja meeskondadele. Automaattestimine tagab kvaliteedi iga arenduse etapis, kindlustades, et lisatud muudatused ei teinud midagi katki, mis tagab tarkvaralahenduse valmiduse tarnimiseks igal ajal [14].

3.1 Automaattestimise protsess

Automatiseerimine on tõhus viis testimise eesmärkide saavutamiseks etteantud tähtajaks, kasutades teatud hulga ressursse. Kuid testimise automatiseerimine ilma kavandamiseta

või kindla protsessi järgimiseta toob kaasa automaattestimissüsteemid, mis on raskesti hallatavad, sageli ebaõnnestuvad ning vajavad pidevat manuaalset sekkumist. Antud olukorra ärahoidmiseks tuleb testimise automatiseerimise ajal järgida automaattestimise protsessi, mis võimaldab saavutada seatud eesmärgid ning testida rakendust parimal võimalikul viisil. Sellist automaattestimise protsessi tuntakse *Automation Testing Life Cycle*'i all, mis koosneb kuuest etapist [16]:

1. Testimise skoobi määramine;
2. Testimise vahendite valik;
3. Automaattestimissüsteemi kavandamine;
4. Automaattestimissüsteemi juurutamine;
5. Automaattestimissüsteemi käivitamine;
6. Tulemuste raporteerimine;

3.1.1 Testimise skoobi määramine

Automatiseerimise skoobi määramine on esimene etapp automaattestimise protsessis. Antud etapi eesmärk on kindlustada automatiseerimise teostatavuse. Toestatavuse analüüsimisel aitab testjuhtude manuaalne ehk käsitsi läbimine ning järelduste tegemine. Skoobi määramisel tuleb leida vastused järgnevatele küsimustele [16]:

1. Mis rakenduse osad võivad olla automatiseeritud ning millised mitte?
2. Mis testid võivad olla automatiseeritud ning kuidas seda teha?

Testi automatiseerimise teostatavus ei kindlusta selle otstarbekust. Automatiseerimise peamine eesmärk on säästa aega, vaeva ja raha, kuid mõtlemata testide automatiseerimine võib tuua rohkem kahju kui kasu. Seega tuleb testidel vastata mingitele nõuetele [1]. E. Dustin on toonud välja kriteeriumid, millest vähemalt osadele vastavuse korral on antud test hea automatiseerimise kandidaat [17] (Tabel 1).

Tabel 1. Automatiseerimise kriteeriumite tabel.

Automatiseerimise kriteerium	**Jah**	**Ei**
Kas test on korduvalt käivitatav?		
Kas test on sageli kasutatav?		
Kas test katab kõige tähtsaimad funktsioonid?		
Kas testi on võimatu teostada manuaalselt?		
Kas test katab kõige keerulisemad kohad?		
Kas test vajab palju erinevaid testandmete kombinatsioone samade sammude kordamiseks?		
Kas eeldatavad testimise tulemused on pidevad ehk ei muutu iga testi käivituse tagant?		
Kas test on väga ajakulukas?		
Kas test teostatakse stabiilsel rakendusel?		
Kas test peab olema kontrollitud erinevatel tarkvaradel ja riistvaradel?		
Kas testi automatiseerimise investeeringutasuvus on paljutõotav?		

3.1.2 Testimise vahendite valik

Automaattestimine on äärmiselt töövahendist sõltuv. Vahendi valimisel tuleb võrrelda saadaolevaid tööriistu, arvestades eesmärgi saavutamiseks vajalike tegureid, mida antud tööriist peaks pakkuma. Teguriteks, millega arvestada, võivad olla [16]:

1. Projektis kasutatavad tehnoloogiad;
2. Varasem kogemus antud tööriistaga;
3. Tööriista omadused, näiteks toetatud veebilehitsejad;
4. Tööriista kasutusmugavus;
5. Tööriistale pakutud toetus, näiteks kogukond (inglise keeles *community*);

Oluline on valida tööriist, mis toetaks nii praeguseid kui ka tulevaseid funktsioone ning ei esineks vajadus tööriista vahetamiseks või järjekordse tööriista lisamiseks projekti, mis tooks kaasa ebavajaliku ressursikulu.

3.1.3 Automaattestimissüsteemi kavandamine

Kavandamise etapp on kõige tähtsaim automaattestimise protsessis, mis määratleb automatiseerimise lähenemisviisi ning kuidas vajalik eesmärk saavutada [16]. Antud

etapis langetatud otsused peavad olema väga hästi läbimõeldud, kuna on aluseks süsteemile, mida hakatakse edaspidi arendama.

Enamus teste vajavad enda tegevuste teostamiseks testandmeid. Heade ja kvaliteetsete testandmete tagamine testimise läbiviimiseks on sama oluline kui testide kirjutamine. Kvaliteetsete testandmeteta ei paku testide tulemused usaldusväärsust. Seega on oluline, et testandmete loomise jaoks oleks jäetud piisavalt aega, kuna heade testandmete korral on testide kirjutamine kergem ning võimaldab lihtsamini laiendada teste testitava rakenduse uue funktsionaalsuse osas [18].

Meeskonnas töötamine on tarkvaraarenduse lahutamatu osa. Automaattestimissüsteemi kavandamisel tuleks silmas pidada võimalust ühendada loodav süsteem juba kasutusel olevatega [18]. Nii ei ole meeskond jaotatud erinevate projektide vahel, mis põhinevad samasugustel tehnoloogiatel, *code-review*'de või abi vajaduse korral oleksid mõlemad osapooled teadlikud süsteemi omapärasustest ning tehtavatest muudatustest saaksid kasu kõik süsteemid korraga.

Loodavad kasutajaliidese automaattestid peavad olema kasutajaliidese muudatuste suhtes vastupidavad [18]. Probleem võib esineda, kui elementide asukoha leidmiseks kasutatakse koordinaate ning kasutajaliidese muutmisest tingitud elementide ümberpaigutus toob kaasa testi ebaõnnestumise. Antud juhul on suur roll testide tulemustel, mis peavad olema piisavalt täpsed, et oleks arusaadav, kas testi ebaõnnestumise põhjustas viga testitavas rakenduses või automaattestimissüsteem.

3.1.4 Automaattestimissüsteemi juurutamine

Testkeskkonna seadistamine peab olema antud etapi esimene punkt. Testkeskkonna puudumine ei võimalda teostada automaattestimissüsteemi arendamist, kuna ei esine võimalust kontrollida lisatud funktsionaalsust. Testkeskkonna seadistamisel tuleb jälgida, et see ei erineks toodangus olevast keskkonnast, väljaarvatud testandmete osas, vastasel juhul ei võimalda testkeskkonnas käivitatud testid anda usaldusväärseid testimise tulemusi [19].

Automaattestimissüsteemi juurutamisel tuleb järgida testimise parimaid tavasid ning kasutatavaid mustreid, mis aitavad tagada rakenduse hallatavust koodiridade arvu suurenedes.

Testid peavad olema järjekindla ülesehitusega, et edaspidistel arendajatel oleks arusaadav, mida antud test kontrollib. Kõik programmikoodi muudatused peavad läbima *code-review*, et tagada rakenduse kvaliteet ning vähendada juhuslikkude vigade tõenäosust [16]. Testimissüsteemi jaoks on samuti oluline logimise funktsionaalsuse lisamine. Logimine võimaldab saada ülevaadet rakenduse tööst nii arendamise kui ka testimise ajal. Logid aitavad väga palju testimise ajal toimunud vigade ülesotsimisel eriti kui testimine ei toimunud kohalikus masinas. Logimist tuleb lisada juba rakenduse arendamise ajal, vastasel juhul on antud ülesanne väga ajakulukas ning osad kriitilised kohad jäävad siiski vahele, kuna rakenduse koodiridade arv pidevalt kasvab.

3.1.5 Automaattestimissüsteemi käivitamine

Kui testjuhud on juurutatud ning testkeskkond ülesseatud, võib käivitada valminud automaattestimissüsteemi. Automaattestimissüsteemi käivitamisel ei kontrollita ainult testitavat rakendust, vaid ka automaattestimissüsteemi ennast. Testi ebaõnnestumise korral koostatakse põhjalik raport, mis edastatakse arendajate meeskonnale paranduste elluviimiseks, millele järgneb järjekordne automaattestimissüsteemi käivitamine [20].

Testide käivitamine peab tagama võimalikult laia kattuvuse, näiteks testimine erinevatel veebilehitsejatel. Testide paralleelne ehk üheaegne käivitamine aitab kokkuhoida testimiseks kuluvat aega [16]. Tavaliselt ei toimu testide käivitamine kohalikus masinas. Seega peab olema usaldusväärne viis testide tulemuste kättesaamiseks, kuna tulemuste puudumisel ei ole testimise peale kulutatud ressurssidest mingit kasu. Lisaks testimise tulemustele peab olema võimalus saada ülevaade ning vajadusel analüüsida testimise kulgemist, mida saab teostada logimise abil, näiteks tagastades lisaks testimise tulemustele ka logi faili.

3.1.6 Tulemuste raporteerimine

Kui testimissüsteem on enda töö lõpetanud, tuleb raporteerida testimise tulemused. Tulemuste analüüsimine võimaldab tuvastada tarkvaralahenduse koostisosade vigu, millest järeldatakse edaspidised tegevused. Põhjalikud tulemused võimaldavad saada laia ülevaadet rakenduse osadest ning vigade korral kiiresti nende põhjused leida. Testimise tulemused jagatakse huvirühmade vahel, mistõttu on testimise tulemustel oluline roll rakenduse töökindluse ja kvaliteedi tagamisel [16].

4 Kasutajaõiguste testimise rakenduse kavandamine

Antud peatükis kavandas autor kasutajaõigusi testiva rakenduse, kasutades selleks peatükis 3 kirjeldatud automaattestimise meetodikat ning järgides automaattestimise parimaid praktikaid.

4.1 Otsustusmaatriks

Erinevatest valikutest kõige sobivaima leidmiseks tuleb neid omavahel võrrelda. Üheks võimalikuks võrdlemise viisiks on otsustusmaatriks (inglise keeles *decision matrix*). Otsustusmaatriks on mõjukas tööriist olukorras, kus esineb palju häid alternatiive, mille seast valida, arvestades mitmeid erinevaid tegureid. Antud protsess koosneb viiest sammust [21]:

1. Kõikide valikute ja nende ühiste tegurite kirjeldamine.
2. Võrreldavast objektist sõltuvalt tegurite hindamine kasutades 5-pallist skaalat, kus 0 tähistab kehva tegurile vastavust ning 5 suurepärast.
3. Tegurite tähtsuse määramine kaalude abil. Näiteks võivad kaalud olla kujutatud protsendilistes ühikutes, kuid ei tohi sel juhul ületada 100%.
4. Punktis 2 saadud tegurite väärtuste korrutamine punktis 3 määratud kaaludega antud teguri suhtes.
5. Punktis 4 saadud tegurite väärtuste summeerimine iga alternatiivi kohta ning saadud summadest suurima väärtusega valiku valimine.

4.2 Testimise skoobi määramine

Skoobi määramisel lähtus autor tähtsaimast rakendusele esitatud nõudest, milleks oli rakenduse seadistatavus erinevate organisatsioonide jaoks. Antud nõude täitmiseks esines vajadus kõikide andmete, mis võivad organisatsioonist sõltuvalt muutuda, kirjapanekuks eraldi faili. Kuna CaseM rakendust kasutavaid organisatsioone on kümneid, ei osutunud võimalikuks kõikide organisatsioonide jaoks testide samaaegne valmistamine. Sellest lähtudes oli esialgne eesmärk seadistada rakendus ühele organisatsioonile, kaasates

osaliselt ka teist organisatsiooni, et oleks võimalik teha kindlaks organisatsioonist sõltuvad andmed, mis peavad olema eraldi kirjeldatud.

Lisaks skoobi määramisele tuli analüüsida, kas kasutajaõiguste testimise automatiseerimine oli otstarbekas. Selleks rakendas autor eelmises peatükis tutvustatud E. Dustin'i poolt välja toodud kriteeriumite tabelit. Tabeli rakendamisel kasutajaõiguste testimise protsessi osas selgus, et antud tegevus on mõtekas, kuna vastab kõikidele esitatud nõuetele (vt Lisa 3).

4.3 Testimise vahendite valik

Testimise vahendite valimiseks rakendas autor eelnevalt mainitud otsustusmaatriksit. Selleks oli autor väljavalinud enimkasutatud tehnoloogiad ja hindanud nende tegureid, tuginedes enda kogemusele, CaseM projektis kasutatud tööriistadele ning avalikult kättesaadavale materjalile.

Esiteks valis autor programmeerimiskeele ning projektihaldustööriista. Vastavalt CaseM projektile oli valitud *Java* programmeerimiskeel ning *Maven*'i projektihaldustööriist.

Järgmiseks esines vajadus kasutajaliidese testimise raamistiku järele. Sobivaima raamistiku väljaselgitamiseks oli autor valinud välja viis levinud kasutajaliidese testimise raamistikku, millest seejärel oli koostatud otsustusmaatriks (Tabel 2).

Tabel 2. Otsustusmaatriks kasutajaliidese testimisraamistiku valikuks [22]-[26].

Tehno-loogia	***Java* tugi**	**Toetatud veebi-lehitsejad**	**Kogukonna suurus ja doku-mentatsioon**	**Lisateekide vajadus**	**Valideerimine**
Teguri kaal	30%	25%	25%	15%	5%
Selenium	5	5	5	4	3
Selenide	5	5	5	5	5
Cucumber	5	5	4	3	3
Cypress	1	4	5	5	5
Protractor	1	4	5	5	5

Otsustusmaatriksi rakendamise käigus osutus väljavalituks *Selenide* (Tabel 3). Vaatamata sellele, oli autor valinud *Selenium*'i kasutajaliidese testimise teostamiseks. *Selenium*'i valik oli põhjendatud sooviga omada võimalust kasutajaõiguste testimise rakenduse ühendamiseks CaseM'is kasutusel olevate automaattestidega, mis kasutavad *Selenium*'it.

Tabel 3. Kasutajaliidese testimisraamistiku valiku otsustusmaatriksi tulemus.

Tehno-loogia	***Java* tugi**	**Toetatud veebi-lehitsejad**	**Kogukonna suurus ja doku-mentatsioon**	**Lisa-teekide vajadus**	**Vali-deerimine**	**Summa**
Teguri kaal	30%	25%	25%	15%	5%	100%
Selenium	1,5	1,25	1,25	0,6	0,15	4,75
Selenide	1,5	1,25	1,25	0,75	0,25	5
Cucumber	1,5	1,25	1	0,45	0,15	4,35
Cypress	0,3	1	1,25	0,75	0,25	3,55
Protractor	0,3	1	1,25	0,75	0,25	3,55

Kasutajaliidese testimise raamistikuks oli valitud *Selenium*, mis iseenesest ei võimalda mugavat testide käivitamist, andmete haldamist, testimise protsessi osadeks jaotamist ega tulemuste raporteerimist. Antud protsesside teostamist ning *Selenium*'i juhtimist võimaldab testimisraamistik [22], [27].

Testimise raamistiku valimiseks võrdles autor kahte raamistikku, millega ta oli varasemalt kokkupuutunud ning omas kogemust. Antud raamistikeks olid *JUnit* ja *TestNG*. Nii *JUnit* kui ka *TestNG* on testraamistikud, kuid *TestNG* kujutab endast *JUnit*'i raamistiku edasiarendust, millele on lisatud uut funktsionaalsust ning lahendatud *JUnit*'i kitsaskohad [10]. Kasutades otsustusmaatriksit, oli autor määranud mõlema raamistiku tegurid, mis olid tähtsad valmiva rakenduse funktsionaalsuse tagamiseks (Tabel 4).

Tabel 4. Otsustusmaatriks testimisraamistiku valikuks [27], [28].

Testimis-raamistik	**Integreeritud programmeerimis-keskkondade tugi**	**Test-komplektide tugi**	**Kasutaja-sõbralikus**	**Lisateekide vajadus**
Teguri kaal	30%	30%	30%	10%
JUnit	5	5	3	3
TestNG	5	5	5	4

Kasutajasõbralikuse all oli autor vaadelnud järgnevaid omadusi:

1. *TestNG* omas laiemat valikut annotatsioone, mis olid selgemalt nimetatud ning intuitiivsemad.
2. *TestNG* testkomplektide loomine läbi XML failide, mis võimaldas samuti määrata paralleliseerimise seadeid, oli tunduvalt selgem ning hallatavam kui *JUnit*'i korral klasside loomist nõudev viis.

Otsustusmaatriksi alusel osutus väljavalituks *TestNG* testimisraamistik (Tabel 5).

Tabel 5. Testimisraamistiku valiku otsustusmaatriksi tulemus.

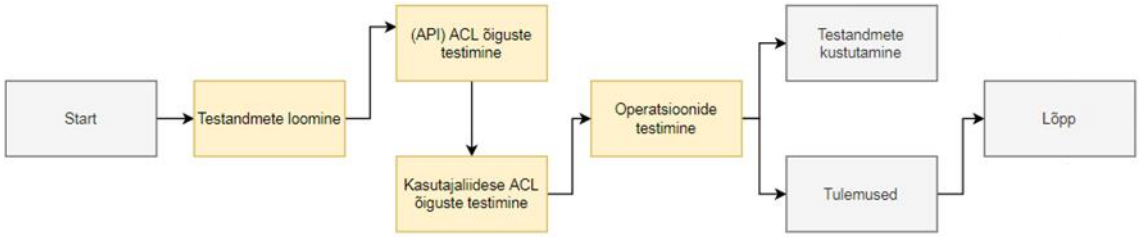
Testimis-raamistik	**Integreeritud programmeerimis-keskkondade tugi**	**Test-komplektide tugi**	**Kasutaja-sõbralikus**	**Lisa-teekide vajadus**	**Summa**
Teguri kaal	30%	30%	30%	10%	100%
JUnit	1,5	1,5	0,9	0,3	4,2
TestNG	1,5	1,5	1,5	0,4	4,9

4.4 Automaattestimissüsteemi kavandamine

Automaattestimisüsteemi protsess oli jaotatud neljaks eraldi protsessiks (Joonis 5):

1. Testandmete loomine;
2. ACL õiguste testimine, kasutades CaseM API't;
3. Kasutajaliidese ACL õiguste testimine;
4. Operatsioonide testimine;

Seejärel teostati testandmete kustutamine testkeskkonna algseisundi taastamiseks ning esitati testimise tulemused.



Joonis 5. Automaattestimissüsteemi testimise protsessi jaotuse kavand.

4.5 Automaattestimissüsteemi juurutamine

Enne automaattestimissüsteemi juurutamist oli vaja seadistada testkeskkond. Testimine pidi olema teostatud juba olemasolevate CaseM rakenduste vastu, mis hõlmasid endas nii andmebaasi, eesrakendust kui ka tagarakendust. Antud rakendused jooksid *Microsoft Azure* peal ning olid seadistatud vastavalt organisatsioonidele.

Automaattestimissüsteemi juurutamisel järgis autor *Selenium'*i dokumentatsiooni soovituste kohaselt leheobjektide mudelit (inglise keeles *page object model*). Antud mudel aitab parandada koodi hallatavust ning vähendab dublitseerimist. Leheobjekt kujutab endast objektorienteeritud klassi, mis täidab automaattestimissüsteemi jaoks liidese funktsiooni veebilehitseja lehekülje vastu. Leheobjektide mudel eelisteks on [22]:

1. Esineb kindel piir testprogrammi koodi ja leheküljele omase koodi, näiteks elementide selektorite, vahel.
2. Kogu lehel teostatav funktsionaalsus on kokkukogutud ühes kohas ehk pole hajutatud testklasside vahel.

Mõlemal juhul võimaldab antud mudel teostada lehekülje funktsionaalsuse muudatuste korral vajalikud parandused ühes konkreetses kohas.

Automaattestimissüsteem pidi olema seadistatav sõltuvalt organisatsioonist. Antud nõude tagamiseks oli juurutamisel kasutatud andmepõhist testimise lähenemist (inglise keeles *data-driven testing*). Andmepõhine testimine kujutab endast testmeetodite ja testandmete üksteisest eraldamist, et võimaldada testimise kulgemise muutmist testmeetodeid muutmata, vähendades seejuures koodiridade arvu ning testide hoolduseks nõudvaid pingutusi. Testandmeid hoitakse antud lähenemise korral keskses asukohas, näiteks JSON (*JavaScript Object Notation*) failis, ning testmeetodid võimaldavad muutuvate andmete sissevoolu [29].

Organisatsioonipõhiste testandmete hoidmiseks oli autor valinud JSON'i andmevahetusvormingu ning kasutusele võtnud *Jackson ObjectMapper*'i, mis pakub JSON'i lugemise ja kirjutamise funktsionaalsust [30]. Lisaks testandmete erinevusele, erinesid organisatsioonid teineteisest kasutajaliidese testide hulgast. Suurem osa kasutajaliidese testidest olid jagatud organisatsioonide vahel, kuid mõnede organisatsioonide korral olid osad testid üleliigsed ning osadel juhtudel esines vajadus

kindla kasutajaliidese testi järele, mis oli kasutatav ainult kindla organisatsiooni korral. Organisatsiooni poolt kasutatavate testide seadistamiseks oli kasutusele võetud XML fail, kus olid kirjeldatud kõik organisatsiooni poolt kasutatavad kasutajaliidese testklassid. XML fail oli antud eesmärgi täitmiseks kasutajasõbralikum, loetavam ning paremini hallatavam, kui JSON. Samuti võimaldas CaseM projektis kasutatav *ConfigNode*'i klass mugavalt nõutud väärtusi XML failist leida, kuna täitis sarnast ülesannet CaseM projektis.

4.6 Automaattestimissüsteemi käivitamine

Automaattestimissüsteemi käivitamise teostamiseks oli autor järginud *TestNG* testimise parimaid praktikaid.

Vaatamata sellele, et *TestNG* pakub mitmeid erinevaid annotatsioone, olid testide käivitamiseks piisavad kolm põhilist annotatsiooni: *@BeforeMethod, @Test* ja *@AfterMethod.*

Testide käivitamine oli teostatav testkomplektide abil, mis kujutavad endast XML faile. Testkomplekt võimaldab grupeerida testjuhte vastavalt vajadusele [7]. Kasutajaõiguste testimise korral, näiteks grupeerida kindla organisatsiooni alamorganisatsioonid ühte testkomplekti ning käivitada nad üheskoos.

Testkomplektide mugavaks käivitamiseks oli autor kasutanud *Maven Surefire Plugin*'i, mis võimaldab rakenduse ehitamisfaasis teste käivitada [31]. Selleks oli *Maven*'i *pom.xml*'i faili, kus hoitakse projekti sõltuvused, lisatud *Maven Surefire Plugin*'i sõltuvus ning seadistatud kausta teekond, kus testkomplektid asusid. Samuti kirjeldas autor *suiteXmlFile*'i muutuja, millele oli võimalik käsureal väärtus lisada (Joonis 6).

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-surefire-plugin</artifactId>
     <version>3.0.0-M2</version>
    <configuration>
        <suiteXmlFiles>
            <suiteXmlFile>
               ${project.basedir}/src/test/java/suites/${suiteXmlFile}
            </suiteXmlFile>
        </suiteXmlFiles>
      </configuration>
</plugin>
```

Joonis 6. *Maven Surefire Plugin*'i sõltuvus ning seadistused.

Seejärel, kasutades *Maven*'i käsurea käske, oli võimalik käivitada vajalik testkomplekt käsuga *mvn –DsuiteXmlFile=GoogleTestSuite.xml test*.

4.7 Testimise tulemuste raporteerimine

Testide jooksutamise järel koostati tulemuste raport. Testide tulemuste kontrollimiseks ei osutunud võimalikuks *assert* lausete kasutamine, kuna samad testklassid oli käivitatud kümneid kordi erinevate andmetega ning vajasid selgeid raporteid, mis objektid olid kasutatud, milline kasutaja oli parasjagu testimisel. Samuti esines vajadus käivitada testid sõltuvalt eelmiste testide tulemustest, mille korral ei piisanud teadmisest, kas test oli õnnestunud või mitte, vaid kindlatel olukordadel. Näiteks dokumendi allkirjastamise vormil olid esialgu täidetud kõik võimalikud väljad. Teatud väljade väärtuste muutmiseks oli dokumendi allkirjastajal vajalik muutmise õigus, mida kasutajal antud dokumendi suhtes ei olnud. Allkirjastamise katse korral kuvati kasutajale veateade. Veateatest sõltuvalt teostati kontroll kasutajaõiguste kohta antud operatsiooni teostamiseks. Kui kasutajal antud omadustega dokumendi muutmise õigust ei olnud, kuid dokumendi allkirjastamise operatsiooni õigus oli olemas, oli antud test arvestatud kui õnnestunud, kuid esines vajadus käivitada sarnane allkirjastamise test, mis ei täitnud muutmise õigust vajavaid välju dokumendi allkirjastamise vormil. Antud funktsionaalsuse teostamiseks oli loodud eraldi klass, mis vastutas testimise tulemuste eest, näiteks õnnestunud, ebaõnnestunud või õnnestunud ja vajalik sõltuva testi käivitamine, ning tulemuste raporteerimiseks kasutusele võetud excel fail.

5 Kasutajaõiguste automaattestimissüsteem

Kasutajaõiguste testimise automatiseerimiseks oli loodud rakendus, mis vastavalt kasutajaõiguste excel failile, täiendavatele seadistuste failidele ning CaseM projektis olevatele XML failidele võimaldas luua nõutud objekte, kasutajaid ning teostada nii API kui ka kasutajaliidese teste vastavalt vajadusele. Sarnaselt testitavale rakendusele olid kasutusele võetud XML failid, mis võimaldasid seadistada kasutajaliidese testide jaoks vajalikude objektide omadused ilma programmikoodi muutmata. Programmeerimiskeelena oli kasutatud *Java*'t ning projektihaldustööriistana *Maven*'it. Lähtudes tähtsaimast nõudest, rakenduse kasutatavusest erinevate organisatsioonidega, olid kõik rakenduse seadistatavad osad, mis võisid erineda organisatsioonist sõltuvalt, üle viidud seadistuste failidesse. Analüüsi tulemustest tulenevalt oli testimise raamistikuks valitud *TestNG* ja kasutajaliidese automatiseerimiseks *Selenium*.

5.1 Testimise protsess

Automaattestimissüsteemi kavandamise käigus oli testimise protsess jaotatud neljaks osaks:

1. Testandmete loomine;
2. API põhised ACL õiguste testid;
3. Kasutajaliidese põhised ACL õiguste testid;
4. Kasutajaliidese põhised operatsioonitestid;

Kõiki eelnevalt kirjeldatud protsesse, väljaarvatud testandmete loomist, oli võimalik vastavalt vajadusele vahele jätta, muutes seadete failis olevaid väärtusi. Samuti oli automaattestimissüsteemi seadistamine käsitletud eraldi etapina testimise protsessis. Järgnevalt tutvustab autor testimise protsessi etappe lähemalt.

5.1.1 Automaattestimissüsteemi seadistamine

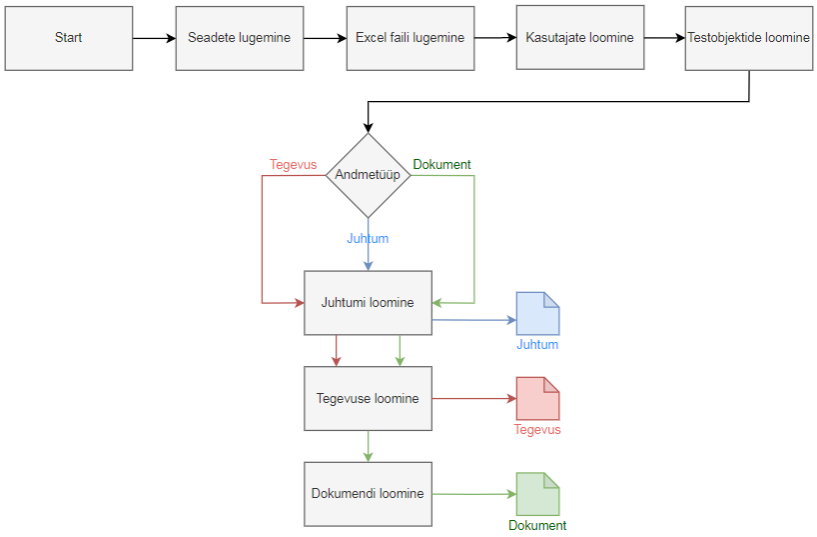
Enne testandmete loomist ja kasutajaõiguste testimisega alustamist oli vaja kirja panna vajalikud seadistamise andmed. Vajalikud seadistamise andmed kujutasid endast:

1. Organisatsiooni spetsiifilised andmed, näiteks organisatsiooni nimi (vt Lisa 4);
2. Testitava rakendusega ühenduse loomiseks vajalikude failide asukohad;
3. Kasutajaõiguste excel faili lugemise jaoks vajalikud andmed;
4. Kasutajad, kasutajate rollid ja grupid (vt Lisa 5);
5. Kasutajaliidese eriliste omadustega, näiteks salajased, väljade nimetused (vt Lisa 6);
6. Testimise tulemuste raporteerimise seaded;

Seadistamise andmed olid kirja pandud JSON faili, mis asus seadete kaustas ning oli nimetatud vastavalt organisatsiooni nimele, mille seadistused asusid antud faili sees. Seejärel esines vajadus luua klass andmete hoidmiseks ning JSON failis antud klassi struktuur soovitud andmetega iga muutuja kohta täita. Rakenduse käivitamisel loodi objekt *TestConfiguration*, mis sisaldas kõiki andmeid etteantud JSON failist. Andmete hoidmine ühes konkreetses objektis lihtsustas uute väljade lisamist ning vajalikude väärtuste leidmist testide loomise ajal.

5.1.2 Testandmete loomine

Testandmete loomisel käsitletakse testobjektidena CaseM'is enimkasutatud objektitüüpe, milleks on juhtumid, tegevused ja dokumendid. Protsess on kujutatud joonisel 7.



Joonis 7. Testandmete loomise protsess.

Kasutajaõiguste excel faili lugemine oli antud protsessi esimene etapp. Kuna organisatsiooni kasutajaõiguste excel failid erinesid kasutajate hulga ning rollide poolest, pidi excel faili lugemine olema dünaamiline ehk ei esinenud võimalust määrata koodis

kasutajatele vastavad tulbad. Antud probleemi lahendamiseks oli esitatud JSON failis olevatele kasutajate nimedele eeldus, et nad vastavad kasutajaõiguste excel failis olevatele tulpade nimetustele. Excel faili lugemise tulemusena oli saadaval *ExcelTable*'i objekt, mis sisaldas kõiki andmeid antud excel failist, ridade kaupa.

Järgmiseks etapiks oli kasutajate loomine. Kasutajate loomine toimus kasutades CaseM API't ja *TestConfiguration*'i objektis olevaid andmeid. Loodavate kasutajate hulk oli võrdeline kasutajaõiguste exceli kasutajate tulpade arvuga, kuna testimine pidi olema teostatud iga erinevas rollis ja grupis oleva kasutaja kohta, mis antud excelis oli kirjeldatud. Ei tohtinud tekkida olukorda, kus kasutaja, lisaks enda tulba rollidele ja gruppidele, kuuluks ka mõne teise tulba rollidesse ja gruppidesse, kuna siis omaks kasutaja laiemaid õigusi ning tooks kaasa vigaseid testimise tulemusi. Esialgu loodi kasutaja etteantud kasutajanimega, kes seejärel lisati testitavasse organisatsiooni ning nõutud rollidesse ja gruppidesse. Et anda kasutajale rollid, mis sõltusid objekti kontekstist või metaandmetest, oli kasutaja lisatud objekti loomise päringu juurde või loodud objekt oli uuendatud nõutud andmetega.

Testobjektide loomine oli teostatud iga automaattestimissüsteemi käivitamise korral, kuna ei esinenud võimalust varem käivitatud automaattestimissüsteemi poolt loodud testobjekte taaskasutada. Taaskasutamine ei olnud võimalik, kuna testimissüsteem kustutas kõik selle töökäigu ajal loodud testobjektid. Testobjektide mittekustutamise korral, eesmärgiga taaskasutada need järgmistel käivitamistel, tooks kaasa vigased testimise tulemused. Nimelt arvutatakse ACL'id objektidele nende salvestamise korral. Kuna CaseM seaded on muudetavad, toob antud tegevus kaasa ka ACL'ide arvutamise töökäigu muutuse. Seega varasema seadistuse alusel loodud ACL'id ei sobinud uue seadistusega testimise jaoks. Samuti ei olnud võimalik kindlustada, et loodud testobjektid ei olnud kellegi poolt muudetud.

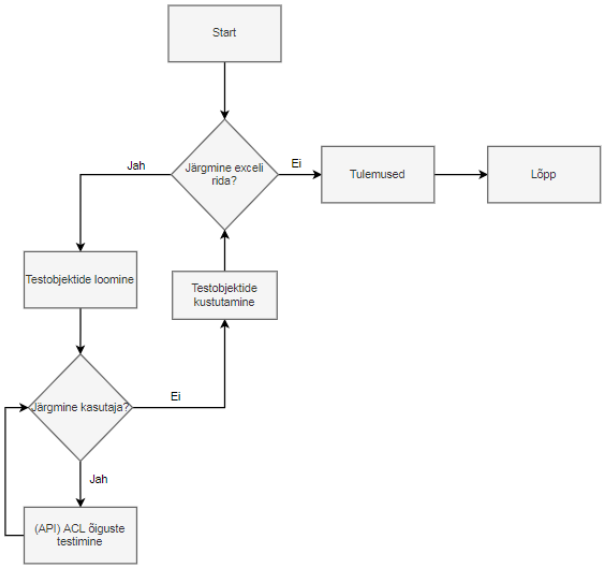
Testobjektide loomine sõltus loodava objekti andmetüübist. Näiteks dokumendi loomise eelduseks on juhtumi ja tegevuse olemasolu. Kõik eelnevalt nimetatud objektid võivad olla erinevate omadustega. Omadused on jaotatud nelja kategooriasse: isiklikud (inglise keeles *personal*) omadused, avalikud (inglise keeles *publicity*) omadused, registreeritud või mitte ja objekti staatus. Iga objekt vajab ühte väärtust iga omaduse juurde. Näiteks võib dokument olla ilma isiklikke andmeteta, salajane, registreeritud ning suletud. Dokumendi korral võib samuti olla sõltuvus juhtumist. Näiteks eelnevalt kirjeldatud

dokumendi omadused ja suletud juhtum. Antud kehtib ka tegevuste ja juhtumite kohta. Antud omadused levivad kõikidele ridadele, mis asuvad omaduste levialas. Seega tuli objektide loomisel arvestada omadustega, mis antud rea kohta kehtisid ning tekitama kõikide võimalikude omaduste kombinatsioonidega objektid. Objektide loomine oli teostatav kasutades CaseM API't sarnaselt kasutajate loomisele.

Kuna erinevaid objekte oli väga palju, nõudis nende loomine kaua aega. Rakenduse tõhususe tõstmiseks teostati testimine, loodud objekte kasutades, iga exceli rea järel. Antud lähenemine võimaldas kindlustada testimise tulemuste olemasolu automaattestimissüsteemi ebaõnnestumise korral.

5.1.3 API põhised ACL õiguste testid

API põhised testid võimaldasid testida kasutajate ACL õigusi eelnevalt loodud objektidele, kasutades API't. API põhiste testide protsess on kujutatud joonisel 8.



Joonis 8. API põhine ACL kasutajaõiguste testimise protsess.

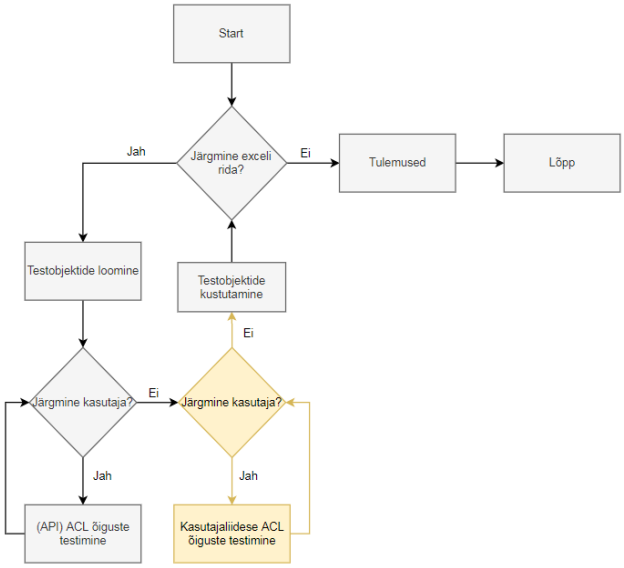
API põhised ACL testid kujutasid endast *ExcelTable* objektist rea kaupa kõikide võimalikude objektide loomist, kuna igat exceli rida võis vaadelda kui eraldi testi, mis omakorda jagunes testideks iga kasutaja kohta. Seejärel, kasutades API't, logiti CaseM rakendusse sisse ning kontrolliti kasutaja õigusi iga loodud objekti peale. Kasutajaõiguste kontrollimiseks esitati päring objekti id'ga, mida soovitakse kontrollida ning kasutajaõiguse tüübiga, näiteks kustutamine. Esitatud päringule saadi vastus, kas õige (inglise keeles *true*) või väär (inglise keeles *false*) vastavalt, kas kasutajal on õigus antud tegevus objektil teostada või mitte. Päringu esitamise käigus oli teada, millise exceli rea

abil oli antud testobjekt loodud. Vastavalt antud exceli reale, valiti välja parasjagu kontrollitav kasutaja ning võrreldi antud kasutaja tulpa antud real. Kui antud tulbas oli märgitud x ning päringu vastus oli *true* või x oli puudu ning päringu vastus oli *false*, oli antud objekti ja kasutaja test õnnestunud. Samas ei tähendanud tulba väärtuse ja päringu vastuse erinevus veel testi ebaõnnestumist. Teadaolevalt seisnes ACL'ide eripära selles, et kõrgema taseme õiguse olemasolu annab kasutajale kõik madalama taseme õigused, näiteks kustutamise õiguse olemasolu annab kasutajale ka lugemise õiguse. Lisaks kõrgema taseme õiguse olemasolule esines olukordi, kus kasutajaõiguste excelis olid sama õiguse kohta mitu rida. Näiteks vastavalt esimese reale, mis vastutas kasutaja kirjutamise õiguse olemasolust objekti peale, ei pidanud kasutajal kirjutamise õigust olema. Järgmine rida, mis vastutas sama objekti salajaste väljade kirjutamise õiguse eest, andis aga kasutajale kirjutamise õiguse. Seega võis esineda olukordi, kus vastavalt esimestele exceli ridadele ei pidanud kasutajal olema õigus, kuid päringu vastuseks oli saadud *true*, kuna kasutajal oli kõrgema taseme või sama õiguse kitsama skoobi õigus, näiteks lihtsalt kirjutamine ja salajaste väljade kirjutamine, sama omadustega objekti peale. Antud olukord oli kokkuleppe kohaselt märgitud kui õnnestunud. Vastasel juhul, kui kasutajaõiguste excel andis kasutajale õiguse, kuid päringu vastus oli *false*, oli antud test ebaõnnestunud.

5.1.4 Kasutajaliidese põhised ACL õiguste testid

Kuna CaseM jaotab objektide metaandmed osadeks, on kasutajaliidesel antud osad kujutatud väljade gruppide kujul ning iga kasutajaõiguste exceli rida vastab teatud väljade grupile. Seega, lisaks ACL funktsioonide testimisele kasutajaliidese kaudu, pidi kasutajaliidese põhine ACL õiguste testimine hõlmama ka objektide väljade testimist, näiteks kasutaja õigust muuta salajane väli. Lisaks ACL'idele on kasutajaliidese korral kasutatud ka *UserRights*. Kui ACL'id arvutatakse objekti loomise või muutmise ajal, siis *UserRights* arvutatakse kasutajaliidese elemendi laadimise ajal. *UserRights* arvutamise reeglid on määratletud eraldi CaseM koodis. Näiteks selleks, et muuta dokumendi salajase välja väärtust, peab kasutajal olema kirjutamise õigus, mis on kontrollitav läbi API ning samuti ei tohi dokument olla allkirjastatud, mis on siinkohal eraldi kontrollitud CaseM koodis. Samas, kui kasutajal on kirjutamise õigus, mis on kontrollitav läbi API, ei tähenda see, et kasutaja saab objekti muuta, kuna muutmine võib olla piiratud järjekordselt CaseM koodis, et saavutada kasutajaõiguste exceli või äriloogika vajadusi.

Selleks, et tõsta testimise tõhusust ja kiirust, oli antud protsess lisatud eelnevas peatükis kirjeldatud API põhise ACL õiguste testimise protsessile (Joonis 9). Selline lähenemine võimaldas taaskasutada eelnevalt loodud objekte, hoides objektide loomise peale kuluvat aega märkimisväärselt kokku.



Joonis 9. Kasutajaliidese põhiste ACL õiguste testimise protsess.

Kuna iga loodud testobjekt vastas kindlale exceli reale, mis omakorda kirjeldas kindlat tegevust antud objekti peal, näiteks salajaste väljade nägemine või muutmine, pidi kasutajaliidese test teostama kindla tegevuse. Kasutajaõiguste excel faili ülesehituse omapära seisnes selles, et iga andmetüübi jaoks oli kirjeldatud kindel hulk tegevusi ning muutujateks olid testobjektide omadused, mis võimaldas, kasutades loodud testobjekti exceli rea reegli numbrit, valida välja õige kasutajaliidese testi klass. Järgnevalt kirjeldab autor antud funktsionaalsuse teostamiseks tehtud tegevusi juhtumi näitel.

Antud funktsionaalsuse teostamiseks oli autor loonud iga organisatsiooni jaoks *acls.xml* faili, milles olid kirjeldatud kõik antud organisatsiooni jaoks võimalikud kasutajaliidese testklassid. Kasutusele oli võetud *Spring* väljenduskeel (inglise keeles *Spring Expression Language*), mis võimaldab teostada objekti peal päringuid programmi töötamise ajal [32]. Loodud *acls.xml* failis oli kasutatud Fujitsu AS XML-skeem. Joonisel 10 on kujutatud *acls.xml* juhtumi osa, kus on kirjeldatud reeglid juhtumi salajaste väljade vaatamise kasutajaliidese testi jaoks. Vastavalt piiratud teostatavate tegevuste hulgale ning muutuvatele andmetele, on antud kasutajaliidese test kasutatav kõikide reeglitega, mis algavad numbriga 1, 2 või 3 ning lõpevad numbriga 2 või 11 (Joonis 10). *Spring* väljenduskeele abil kontrolliti kasutajaliidese testi sobivust kasutades XML elemendi

id'ga *clause* väärtust. Elemendi väärtuses olev meetod *getRuleNr()* tagastas kasutajaõiguste exceli faili reegli numbri, mida kasutati testobjekti loomisel. Seejärel meetodi *isOneOf()* abil kontrolliti, kas esimesel kohal olev väärtus oli võrdne mõne järgmise väärtusega. Kokkusobivuse korral tagastas meetod väärtuse, kas *true* või *false*. Kasutajaliidese testi sobivuse korral, kasutades elementi id'ga *class*, loodi vajalik testklassi objekt. Kõik kasutajaliidese testklassid realiseerisid liidest *IBaseUI*, mis sisaldas abstraktset meetodit *run()*, mida iga alamklass pidi implementeerima ning mis võimaldas käivitada testi, teadmata kindlat testklassi. Eelnevalt kirjedatu oli realiseeritud tehase mustri (inglise keeles *factory pattern*) abil. Tehase muster võimaldab luua objekte, loomise protsessi näitamata, ning loodud objektidele ühise liidese kaudu viidata [33].

```
<CodeBase xmlns="http://fujitsu.com/fujitsu/webservices/2013/01/31">
      <Code Identifier="cases">
            <Code>
                  <Description>Juhtumi salajaste väljade vaatamine
                  </Description>
                  <Value Identifier="clause">isOneOf(getRuleNr(),
                        '1.1.2', '1.1.11', '1.2.2', '1.2.11',
                        '2.1.2', '2.1.11', '2.2.2', '2.2.11',
                        '3.1.2', '3.1.11', '3.2.2', '3.2.11')
                  </Value>
                  <Value Identifier="class">
                        CaseViewSecretMetadata
                  </Value>
            <Code>
      </Code>
</CodeBase>
```

Joonis 10. Juhtumi salajaste väljade vaatamise osa *acl.xml* failist.

5.1.5 Kasutajaliidese põhised operatsioonitestid

Kasutajaliidese põhised operatsioonitestid testisid nii objektide peal teostatavaid operatsioone kui ka objektidest sõltumatuid operatsioone, näiteks otsingumootori kasutamine. Operatsioonide korral on samuti kasutatud *UserRights*, et määrata operatsiooni teostamise võimaldamine kasutaja jaoks. Vaatamata sellele, et antud testid vajasid samuti testobjekte, ei kuulunud nad eelnevalt kirjeldatud kasutajaõiguste testimise protsesside juurde. Antud lähenemine oli kasutatud, kuna mõlemad kasutajaliidese testid teostasid loodud objektidega tehinguid, näiteks muutes objektid avatud olekust suletud olekusse, allkirjastades neid või saates nad arhiivi. Objekti omaduste muutmise tagajärjel ei vastanud antud objekt enam kindlale kasutajaõiguste exceli reale, mille alusel see

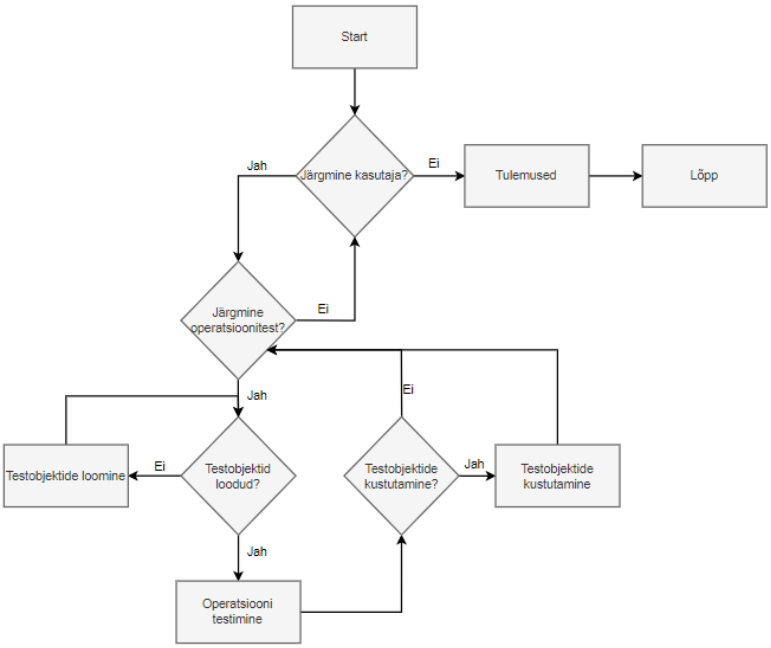
algselt oli loodud. Antud objekti kasutamine testimiseks tooks kaasa vigased testimise tulemused. Samuti ei võimaldanud osad objektidel teostatud operatsioonid korrata antud operatsiooni järjekordselt sama objekti peal, mis oli vajalik teise kasutaja testimise korral, näiteks dokumendi arhiveerimine, juhtumi kustutamine või avamine. Lähtudes antud probleemidest, loodi operatsioonitestide jaoks testobjektid eraldi, kasutades selleks eelnevalt kirjeldatud reeglite numbreid operatsioonide XML failis, mis vastasid kasutajaõiguste excel failis olevatele reeglite numbritele.

Operatsioonide XML fail, *operations.xml*, toimis sarnaselt *acls.xml* failile. Erinevus seisnes selles, et *operations.xml* failis kasutati exceli rea reeglite numbreid vajalike testobjektide loomiseks, kuid testklasside valikuks kasutati CaseM seadistuste failidest leitavaid operatsioonide nimesid. Joonisel 11 on kujutatud juhtumi avamise operatsiooni kontrollimise osa *operations.xml* failist. Meetod *opOneOf()* kontrollis, kasutades *Spring* väljenduskeelt, kas CaseM projektist saadud väärtus vastas meetodile parameetrina antud väärtusele. Vastavuse korral loodi *OpenCaseOperation* klass, mis realiseeris *IBaseUI* liidest. XML element id'ga *requiredObjectRules* sisaldas kasutajaõiguste excel faili reeglite numbreid regulaaravaldise (inglise keeles *regular expression*) kujul, mis pidid antud testiga olema kontrollitud. Regulaaravaldis on sõne, mis võimaldab tekstimustrite järgi sobitada, hallata ning leida erinevaid tekste [34]. Antud elemendi sees oli võimalik kirjeldada kõikide kasutajate jaoks erinevaid reeglite numbreid, kasutades identifikatsioonidena kasutajate nimesid. Joonisel 11 kujutatud tärn ehk korrutusmärk tähistab antud reeglite kasutamist kõikide kasutajate jaoks.

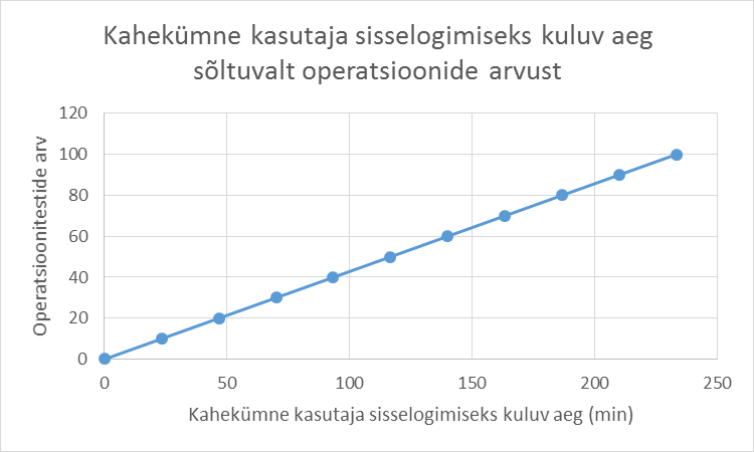
```
<CodeBase xmlns="http://fujitsu.com/fujitsu/webservices/2013/01/31">
      <Code Identifier="10.105">
            <Code>
                  <Description>Juhtumi avamine</Description>
                  <Value Identifier="clause">
                        opOneOf('OpenCaseOperation')
                  <Value>
                  <Value Identifier="class">OpenCaseOperation</Value>
                  <Code Identifier="requiredObjectRules">
                        <Value Identifier="*">'^[1](.*\.(5)$)'</Value>
                  </Code>
            <Code>
      </Code>
</CodeBase>
```

Joonis 11. Juhtumi avamise osa *operations.xml* failist.

Operatsioonide testimise protsess (Joonis 12) teostati loodud kasutajate kaupa, testides kõik võimalikud operatsioonid antud kasutajaga, liikudes seejärel järgmise kasutaja juurde. Antud lähenemine aitas tõsta protsessi kiirust, vähendades testimise jaoks vajalike kasutajaliidese kaudu sisselogimiste arvu, kuna juba kahekümne kasutaja korral, kus iga kasutajaga sisse- ja väljalogimisele kulus ligikaudu seitse sekundit, kasvas operatsioonide arvust sõltuvalt antud tegevuseks kuluv aeg märkimisväärselt (Joonis 13). Samuti võimaldasid mõned teostatavatest operatsioonidest testobjektide taaskasutust teiste kasutajate testimise korral, mis järjekordselt tõstis testimise protsessi kiirust.



Joonis 12. Kasutajaliidese põhine operatsioonide testimise protsess.



Joonis 13. Kahekümne kasutaja sisselogimiseks kuluv aeg sõltuvalt operatsioonide arvust.

5.2 Testide käivitamine

Testide käivitamise jaoks, sõltuvalt organisatsioonist, olid iga organisatsiooni jaoks loodud eraldi testklassid, mis olid *ACLUserRightsAbstractTest* klassi alamklassid ning kirjutasid üle meetodi *getConfFilePath(),* mis tagastas antud organisatsiooni seadete faili asukoha projektis (Joonis 14). Vastavalt seadete failile valiti välja kõik vajalikud failid organisatsiooni kasutajaõiguste testimiseks, näiteks kasutajaõiguste excel fail, organisatsiooni kasutajaliidese testide XML failid.

```
public class FujitsuTests extends ACLUserRightsAbstractTest {
            @Override
            protected String getConfFilePath() {
                  return
                  "/resources/conf/prod/xALM/Fujitsu_conf.json";
            }
}
```

Joonis 14. Organisatsiooni testklassi näide.

ACLUserRightsAbstractTest klass sisaldas kolme meetodit:

1. *beforeMethod*, kus olid teostatud ettevalmistavad tegevused, näiteks seadete faili lugemine, exceli faili lugemine ja kasutajate loomine.
2. *aclRightsTest*, kus leidis aset tegelik testimine. Toimus testandmete loomine, API põhine testimine, kasutajaliidese testimine ning tulemuste raporti koostamine.
3. *afterMethod*, mis vastutas testkeskkonna algseisundi taastamise eest ehk testimise käigus loodud andmete kustutamise eest.

Automaattestimise rakendus oli mõeldud kui osa CI/CD protsessist. Kuna autori meeskonnas kasutatakse Git versioonihaldustarkvara ning GitLab versioonihalduskeskkonda, oli automaattestimise rakenduse käivitamine teostatud GitLab keskkonnas. GitLab CI/CD seadistamiseks oli projekti juurkaustas loodud *.gitlab-ci.yml* fail. Antud faili abil luuakse GitLab'is torujuhe (inglise keeles *pipeline*), mis võimaldab käivitada varamus (inglise keeles *repository*) oleva programmi. Nimetatud torujuhe koosneb tööülesannetest (inglise keeles *jobs*), mis määravad ära, mida tuleb teha ning etappidest (inglise keeles *stages*), mis määravad, kuna tööülesandeid käivitada [35]. Autori poolt kasutavas GitLab'i keskkonnas olid kõik vajalikud tegevused torujuhtme käivitamiseks eelnevalt teostatud ning autoril jäi ainult koostada *.gitlab-ci.yml* fail, mis oleks käivitatav olemasolevate protsesside poolt.

Rakenduse käivitamine pidi olema teostatud käsurealt. Käsureal käivitamise korral pidi olema võimalik määrata vajalikud testid, mida soovitakse käivitada. Selleks oli loodud *TestNG* testkomplekt, kus kirjeldati vajalikud testklassid, mida käivitada (Joonis 15).

```
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >
<suite name = "Fujitsu suite">
   <test name = "Fujitsu">
      <classes>
         <class name ="test.it.runner.prod.FujitsuTests" />
      </classes>
   </test>
</suite>
```

Joonis 15. *TestNG* Fujitsu testkomplekti näide.

Testimissüsteemi seadistamise võimaldamiseks läbi torujuhtme olid lisatud parameetrid *.gitlab-ci.yml* faili, mille väärtused olid määratlevad torujuhtme käivitamisel, kasutades GitLab'i kasutajaliidest (vt Lisa 7). Rakenduse käivitamiseks oli seejärel loodud eraldi torujuhtme etapp, mis kasutas Docker'it, valis, vastavalt torujuhtme parameetritele, testkomplekti, edastas vajalikud parameetrid automaattestimissüsteemile süsteemi omaduste (inglise keeles *system properties*) kaudu ning käivitas selle (vt Lisa 8).

5.3 Testimise tulemused

Teadaolevalt ei olnud testide tulemuste kontrollimiseks võimalik kasutada *assert* lauseid. Antud funktsionaalsuse teostamiseks oli kasutusele võetud *UITestResult* klass, mida kasutati *assert* lausete asemel testide tulemuste tagastamiseks. *UITestResult* klassis oli kasutatud *testResultState*'i muutujat. *TestResultState* oli enum tüüpi klass (vt Lisa 9). Enum on andmetüüp, mis võimaldab hoida eelmääratletud konstante, mida saab valida teatud väärtuste hulgast [36]. Antud väli võimaldas tagastada testimise tulemusi kui ka tekitada sõltuvusi testklasside vahel, mille alusel käivitati testklass vajadusel. Testide jooksutamise käigus koguti kokku kõik testimise tulemused kasutades *Reporter* klassi. Kasutajaõiguste testimise protsessi lõppedes koostas *Reporter* klass kõikidest tulemustest raporti, mis esitati excel faili kujul.

Kasutajaõiguste testimise rakenduse käivitamine toimus GitLab'i torujuhtmes, mis võimaldab tööülesande lõppemise järel üleslaadida arhiivi tööülesande artefaktidega (inglise keeles *artifacts*). Artefaktid kujutavad endast nimekirja failidest ja kaustadest, mis torujuhtme tööülesande täitmise ajal tekivad [8]. Testimise käigus saadud tulemused

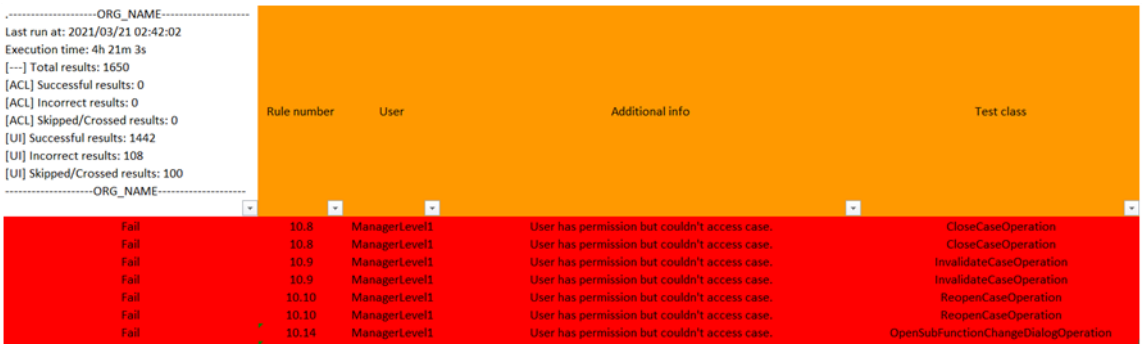
koguti kokku ühte kausta, mida oli vaja artefaktina GitLab'i üleslaadida. Selleks oli lisatud torujuhtme etapp, mis vastutas artefaktide üleslaadimise eest. Antud etapp käivitati peale testimise etappi. Üleslaetud artefaktide arhiiv oli allalaetav GitLab'i kasutajaliidesest ning sisaldas endas logi faili, tulemuste exceli faili (Joonis 16) ning testide ebaõnnestumise korral ekraanipilte. Logi fail ning ekraanipildid võimaldasid jälgida rakenduse töökäiku ning pakkusid eriti suurt abi vigade parandamise korral.

----------ORG_NAME----------
Last run at: 2021/03/21 23:44:02
Execution time: 1h 23m 17s
[---] Total results: 12125
[ACL] Successful results: 8472
[ACL] Incorrect results: 1
[ACL] Skipped/Crossed results: 1620
[UI] Successful results: 0
[UI] Incorrect results: 0
[UI] Skipped/Crossed results: 0
----------ORG_NAME----------

Type	Rule number	Condition	What	Expected permission	Actual permission	Role
Fail	ams_document	5.1.21	record.finished and case.closed	write/tos,tos_r,other	TRUE	FALSE	Registrar
Pass	ams_document	5.1.21	record.finished and case.closed	write/tos,tos_r,other	FALSE	FALSE	AllInternalUsers
Pass	ams_document	5.1.21	record.finished and case.closed	write/tos,tos_r,other	TRUE	TRUE	ArchiveManager
Pass	ams_document	5.1.21	record.finished and case.closed	write/tos,tos_r,other	TRUE	TRUE	ContentMainUser
Pass	ams_document	5.1.21	record.finished and case.closed	write/tos,tos_r,other	FALSE	FALSE	ManagerLevel1
Pass	ams_document	5.1.21	record.finished and case.closed	write/tos,tos_r,other	FALSE	FALSE	ManagerLevel2
Pass	ams_document	5.1.21	record.finished and case.closed	write/tos,tos_r,other	FALSE	FALSE	ManagerLevel3
Pass	ams_document	5.1.21	record.finished and case.closed	write/tos,tos_r,other	FALSE	FALSE	ManagerLevel4
Pass	ams_document	5.1.21	record.finished and case.closed	write/tos,tos_r,other	FALSE	FALSE	SubFunctionBrowser
Pass	ams_document	5.1.21	record.finished and case.closed	write/tos,tos_r,other	FALSE	FALSE	SubFunctionReader

Joonis 16. ACL API testimise tulemuste excel faili osa.

Vigaste tulemuste korral pöörduti esialgu tulemuste exceli lisainfo (inglise keeles *additional info*) tulba poole (Joonis 17). Antud tulbas oli täpsem kirjeldus testi ebaõnnestumisest. Kui kirjeldusest oli arusaadav, miks test ebaõnnestus, käivitati antud test, ebaõnnestunud kasutajaga, kohalikus masinas ning analüüsiti tulemusi. Kui probleemi allikas ei olnud kirjeldusest arusaadav, otsiti vajalik informatsioon logi failist. Kuna tulemuste excelis oli igas reas märgitud ära, mis kasutaja (inglise keeles *user*), testklassi (inglise keeles *test class*) ning reegli numbri (inglise keeles *rule number*) eest antud rida vastutab, oli vajalik koht logi failis kiiresti leitav, sest logisõnum sisaldas alati parasjagu testimises olevate eelnimetatud väljade väärtusi.



----------ORG_NAME----------
Last run at: 2021/03/21 02:42:02
Execution time: 4h 21m 3s
[---] Total results: 1650
[ACL] Successful results: 0
[ACL] Incorrect results: 0
[ACL] Skipped/Crossed results: 0
[UI] Successful results: 1442
[UI] Incorrect results: 108
[UI] Skipped/Crossed results: 100
----------ORG_NAME----------

Rule number	User	Additional info	Test class
Fail	10.8	ManagerLevel1	User has permission but couldn't access case.	CloseCaseOperation
Fail	10.8	ManagerLevel1	User has permission but couldn't access case.	CloseCaseOperation
Fail	10.9	ManagerLevel1	User has permission but couldn't access case.	InvalidateCaseOperation
Fail	10.9	ManagerLevel1	User has permission but couldn't access case.	InvalidateCaseOperation
Fail	10.10	ManagerLevel1	User has permission but couldn't access case.	ReopenCaseOperation
Fail	10.10	ManagerLevel1	User has permission but couldn't access case.	ReopenCaseOperation
Fail	10.14	ManagerLevel1	User has permission but couldn't access case.	OpenSubFunctionChangeDialogOperation

Joonis 17. Ebaõnnestunud operatsioonitestide tulemuste exceli faili osa.

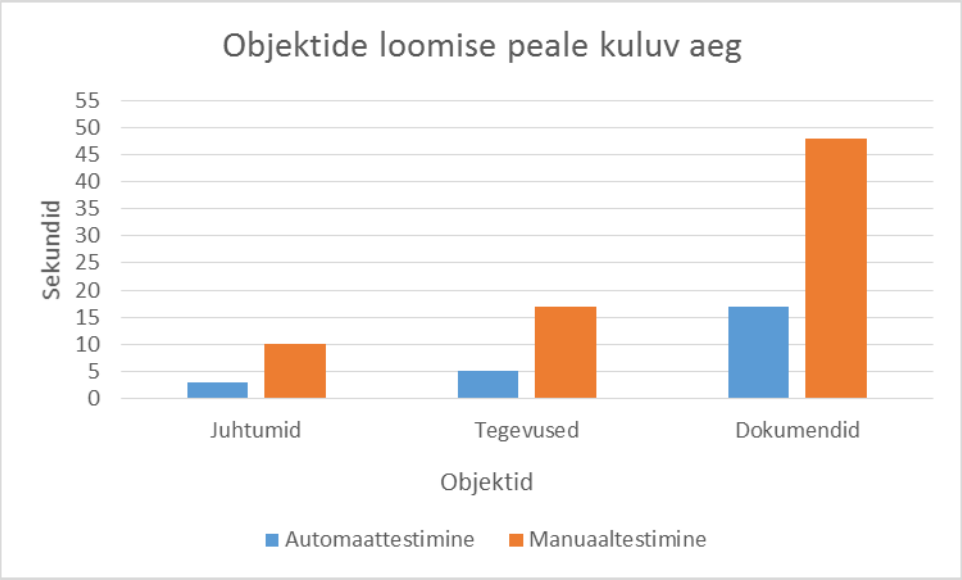
6 Tulemuste analüüs ja edaspidised plaanid

Antud peatükis võrdles autor valminud automaattestimissüsteemi manuaalse testimisega ning sõnastas automaattestimissüsteemiga seonduvad edaspidised plaanid.

6.1 Tulemuste analüüs

Automaattestimissüsteemist saadava kasu võrdlemiseks oli võimalik antud automaattestimissüsteemi võrrelda ainult manuaaltestimisega, kuna kasutajaõiguste testimine oli varem teostatud ainult manuaalselt.

Esiteks oli autor võrrelnud testobjektide loomise peale kuluvat aega. Testobjektideks olid valitud juhtumid, tegevused ja dokumendid. Võrdlemise lihtsustamiseks olid kõikide loodavate objektide korral kasutatud neile vaikimisi määratud staatused. Juhtumite ja tegevuste korral avatud (inglise keeles *opened*) staatus ning dokumentide korral mustandi (inglise keeles *draft*) staatus. Võrdlemise tulemused on esitatud joonisel 18.

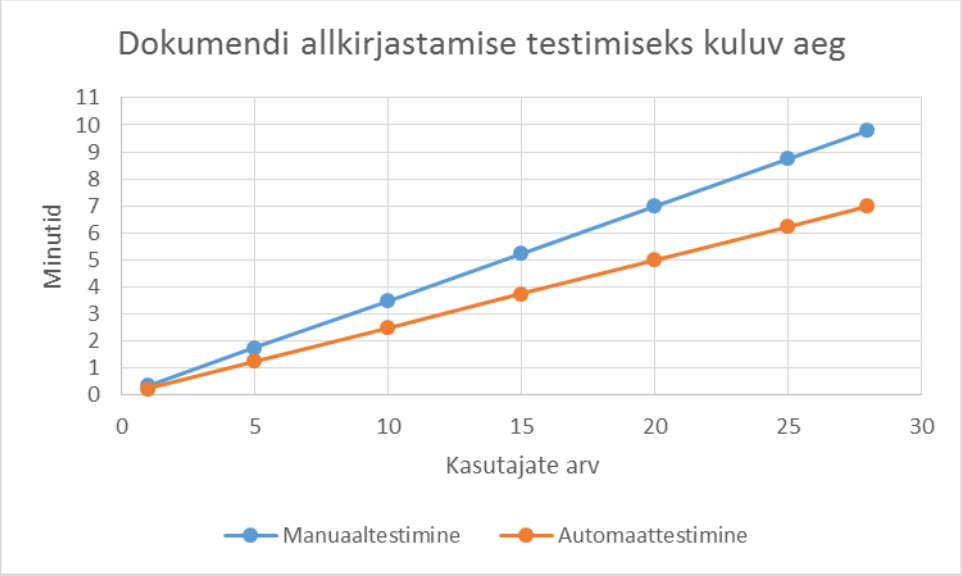


Joonis 18. Testobjektide loomise peale kuluv aeg.

Testobjektide loomise peale kuluv aeg erines väga palju automaattestimise korral manuaaltestimisest, mis kasutas kasutajaliidest. Juhtumi loomise peale kulus automaattestimissüsteemil 3 sekundit, manuaaltestimise korral aga 10 sekundit. Tegevuse loomise peale kulus automaattestimissüsteemil 5 sekundit, manuaaltestimise korral aga

17 sekundit. Siinkohal tuleb märkida, et tegevuse loomiseks pidi olema loodud ka juhtum. Dokumendi loomise peale kulus automaattestimissüsteemil 17 sekundit, manuaaltestimise korral aga 48 sekundit. Nagu ka tegevuse sõltuvus juhtumist, tuli dokumendi loomise jaoks esialgu luua juhtum ja tegevus. Antud tulemustest selgus, et testobjektide loomist teostas automaattestimissüsteem ligikaudu 3 korda kiiremini. Nii suurt erinevust tingis kasutajaliidese laadimise peale kuluv aeg ning vajalikude väärtuste otsimine kõikide objektide loomise vormidest. Automaattestimissüsteem esitas siinkohal kõik andmed koheselt läbi API päringu, jättes vahele kasutajaliidese osa.

Järgmiseks võrdles autor kasutajaliidese testimise jaoks kuluva aja erinevust. Kuna manuaalselt testimise korral ei olnud API põhine ACL'ide testimine teostatav ning ACL kasutajaliidese testimine sarnanes operatsioonide testimisele, oli võrdlemiseks valitud dokumendi allkirjastamise operatsiooni test, mis teostati läbi kasutajaliidese. Antud testi teostamiseks esines vajadus dokumentide järele. Võrdlemise lihtsustamiseks eeldas autor, et kõik vajalikud dokumendid ja kasutajad, kelle arv, lõputöö raames vaadeldud organisatsioonis, oli 28 tükki, olid juba loodud ning kõikide juhtumite, mille sees dokumendid asusid, id'd olid samuti teada, mis võimaldas läbi URL'i (*Uniform Resource Locator*) vajalik juhtum avada. Ehk pidi olema teostatud ainult dokumendi allkirjastamine. Võrdlemine teostati kasutades *Google Chrome*'i veebilehitsejat. Antud võrdlemise tulemused on kujutatud joonisel 19.



Joonis 19. Dokumendi allkirjastamise testimiseks kuluv aeg.

Dokumendi allkirjastamise testimiseks ühe kasutaja jaoks kulus manuaaltestimise korral keskmiselt 21 sekundit, automaattestimise korral 15 sekundit. Antud ajad kujutavad

endast kasutaja sisselogimist, dokumendi allkirjastamist ning kasutaja väljalogimist. Kuigi automaattestimine oli ainult 6 sekundit kiirem, ei ole manuaalselt võimalik alati samasuguse täpsusega antud testimist teostada. Suure hulga kasutajate testimise korral suureneb manuaaltestimisele kuluv aeg testija väsimusest, mis automaattestimist aga ei mõjuta. Automaattestimine, olles ainult 6 sekundit kiirem, kulutas 28 kasutaja korral peaaegu 3 minutit vähem kui manuaalne lähenemine, mis järjekordselt eeldas testija võimekust teostada testimine alati sama kiirusega. Saavutatud ajavõit kujutas endast ainult ühe dokumendi allkirjastamise testimist. Mitmesaja dokumendi korral kasvab automaattestimisega tingitud ajavõit aga märkimisväärselt.

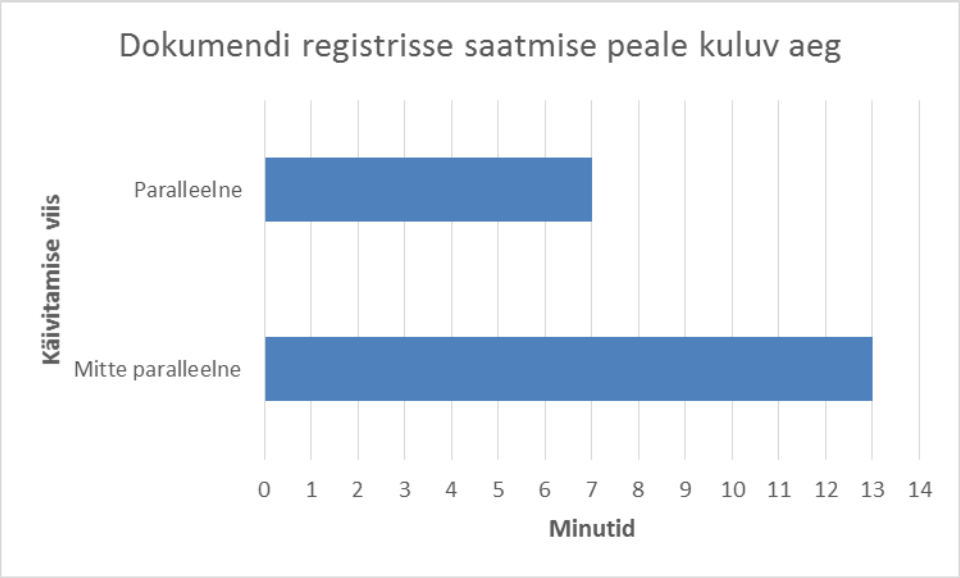
Esitatud võrdlemiste tulemustest järeldas autor, et valminud automaattestimissüsteem ületas manuaalset lähenemist nii testobjektide loomise osas kui ka testide teostamises. Lisaks sellele, et kasutajaõiguste testimise peale kuluv aeg kahanes märkimisväärselt, võimaldas antud automaattestimissüsteem tagada muutumatute andmetega testobjektide loomist, mis omakorda tagas testimise protsessi ja tulemuste järjepidevuse.

6.2 Edaspidised plaanid

Lõputöö raames valminud automaattestimissüsteem ei olnud lõplik ning pidevalt arendatav autori poolt. Esimese eesmärgina kavatseb autor katta kogu lõputöö raames vaadeldud organisatsiooni kasutajaõiguste testide vajaduse. Antud eesmärgi saavutamiseks esineb vajadus kirjutada järelejäänud kasutajaliidese testid ümber nii, et testide poolt kontrollitav funktsionaalsus oleks päriselt teostatud, mitte lihtsalt kontrollitud, kas vastav nupp on vajutatav või mitte. Antud muudatused kavatseb autor teostada selliselt, et oleks võimalik seadistada, millist lähenemist kasutatakse.

Kui esimese organisatsiooni kasutajaõigused on automaattestimissüsteemi poolt kaetud, esineb vajadus lisada antud automaattestimissüsteemi ka järgmised organisatsioonid. Antud automaattestimissüsteem oli arendatud selliselt, et järjekordse organisatsiooni lisamise protsess kujutaks endast ainult seadete failide loomise ja antud organisatsiooni kasutajaõiguste exceli faili lisamist automaattestimissüsteemi projekti. Lõputöö raames vaadeldud teine organisatsioon, rakenduse seadistamise tagamiseks, sisaldas aga lisaks esimese organisatsiooni operatsioonidele ka teisi operatsioone, mis nõuaksid antud operatsioone testivate kasutajaliidese testide lisamist projekti.

Kuna erinevaid organisatsioone on palju, peavad testid olema võimalikult kiired. Testide kiiruse tõstmiseks kavatseb autor kasutada paralleeltestimist ehk testide üheaegset käivitamist. Paralleeltestimise otstarbekuse tõestamiseks võrdles autor dokumendi registrisse saatmise operatsiooni, mis võimaldas taaskasutada varem loodud dokumenti. Võrdlus teostati automaattestimissüsteemi abil, mis käivitati esialgu kõikide kasutajatega ning seejärel jaotati kasutajad kahe testi vahel, mis olid käivitatud paralleelselt. Võrdlus teostati *Google Chrome*'i veebilehitseja abil ning käsitleti kogu testimisprotsessi ehk kasutajate loomise, testobjektide loomise, testimise ja testkeskkonna algseisundi taastamise peale kuluvat aega. Võrdlemise tulemused on kujutatud joonisel 20.



Joonis 20. Dokumendi registrisse saatmise peale kuluv aeg.

Paralleelse käivitamise korral kulus dokumendi registrisse saatmise peale keskmiselt 7 minutit, mitte paralleelse käivitamise korral aga keskmiselt 13 minutit. Vaatamata sellele, et paralleelse käivitamise korral loodi osade kasutajate jaoks samasugused dokumendid, kuna antud kasutajad olid jagatud paralleelselt käivitatud testide vahel, kulus paralleliseeritud testidel peaaegu 2 korda vähem aega kui kõikide kasutajate testimine üksteise järel.

Automaattestimissüsteemi käivitamise hõlbustamiseks kavatseb autor kasutusele võtta pideva integratsiooni tööriista Jenkins'i, kuna praegu kasutatakse selleks GitLab'i kasutajaliidest.

7 Kokkuvõte

Käesolev bakalaureusetöö lahendas aktuaalset probleem AS Fujitsu Estonia Documentum meeskonnas, kus antud meeskonna poolt arendatava projekti kasutajaõiguste testimine oli teostatud manuaalselt, mis oma mahukuse poolest ei võimaldanud manuaaltestimise abil katta kõiki testimist vajavaid kasutajaõiguste poolt mõjutatud rakenduse osad. Antud probleem esines juba ühe organisatsiooni jaoks seadistatud rakenduse korral, kuid erinevalt seadistatud rakendusi oli meeskonna käsitluses kümneid. Selle tulemusena ei rahuldanud tarnitav tarkvaralahendus klientide nõudmisi ning kannatas tarkvaralahenduse kvaliteet.

Lähtudes antud probleemidest, oli antud bakalaureusetöö eesmärgiks automatiseerida kasutajaõiguste testimine. Erinevate organisatsioonide jaoks seadistatud rakenduste arvust tingitud oli bakalaureusetöö raames vaadeldud ühte organisatsiooni ning kaasatud osaliselt ka teist, et kindlustada valmiva automaattestimissüsteemi seadistatavus vastavalt organisatsioonile. Töö käigus tutvustati automaattestimise protsessi, mis oli seejärel rakendatud automaattestimissüsteemi kavandamisel. Kavandamisel analüüsiti tuntumaid kasutajaliidese testimisraamistike, mille seast, kasutades otsustusmaatriksit, osutus valituks *Selenide*. Kuid testitava projekti teises testimissüsteemis kasutatud *Selenium*'i raamistikust lähtudes, oli antud automaattestimissüsteemi aluseks valitud samuti *Selenium*. *Selenium*'i valik nõudis samuti testimisraamisiku valiku, milleks osutus analüüsi käigus *TestNG*. Kasutatud oli *Java* programmeerimiskeel ning *Maven*'i projektihaldustööriist. Valminud automaattestimisüsteem nõudis suurt hulka testandmeid, mille loomist vaadeldi eraldi etapina testimise protsessis, mis oli omakorda jaotatud kolmeks. Samuti oli kasutusele võetud seadistamise failid, mis võimaldasid valminud rakenduse seadistamist vastavalt organisatsioonile programmikoodi muutmata. Automaattestimissüsteem oli käivitatav läbi GitLab kasutajaliidese, kasutades torujuhet.

Valminud automaattestimissüsteem osutus testandmete loomises kolm korda ning kasutajaliidese testimises 30% kiiremaks kui manuaalne lähenemine. Kuigi valminud automaattestimissüsteem vajas veel täiendamist, pakkus see suurt abi kasutajaõiguste testimises.

Kasutatud kirjandus

[1] Oracle, „Lesson: Annotations,“ [Võrgumaterjal]. Saadaval: https://docs.oracle.com/javase/tutorial/java/annotations. [Kasutatud 26 märts 2021].

[2] SmartBear, „API Endpoints – What Are They? Why Do They Matter?,“ [Võrgumaterjal]. Saadaval: https://smartbear.com/learn/performance-monitoring/api-endpoints. [Kasutatud 26 märts 2021].

[3] SmartBear, „What is Code Review,“ [Võrgumaterjal]. Saadaval: https://smartbear.com/learn/code-review/what-is-code-review. [Kasutatud 26 märts 2021].

[4] Opensource, „What is Docker?,“ [Võrgumaterjal]. Saadaval: https://opensource.com/resources/what-docker. [Kasutatud 26 märts 2021].

[5] N.Nurseitov, M.Paulson, R.Reynolds, C.Izurieta, „Comparison of JSON and XML Data Interchange Formats: A Case Study,“ [Võrgumaterjal]. Saadaval: https://www.cs.montana.edu/izurieta/pubs/caine2009.pdf. [Kasutatud 23 märts 2020].

[6] ITProToday, „User Rights,“ [Võrgumaterjal]. Saadaval: https://www.itprotoday.com/compute-engines/user-rights. [Kasutatud 26 märts 2021].

[7] TestMonitor, „Test Case, Test Suite, Test Run, What’s the Difference?,“ [Võrgumaterjal]. Saadaval: https://www.testmonitor.com/blog/test-case-test-suite-test-run-whats-the-difference. [Kasutatud 26 märts 2021].

[8] Testim, „What Is Test Automation? A Simple, Clear Introduction,“ [Võrgumaterjal]. Saadaval: https://www.testim.io/blog/what-is-test-automation. [Kasutatud 25 märts 2021].

[9] Mozilla, „What is a URL?,“ [Võrgumaterjal]. Saadaval: https://developer.mozilla.org/en-US/docs/Learn/Common_questions/What_is_a_URL. [Kasutatud 26 märts 2021].

[10] Vaadin, „Vaadin Framework,“ [Võrgumaterjal]. Saadaval: https://vaadin.com/docs/v8/framework/introduction/intro-overview. [Kasutatud 26 märts 2021].

[11] Fujitsu, „CaseM – Dynamic Information Management in Action,“ [Võrgumaterjal]. Saadaval: https://www.fujitsu.com/fi/Images/casem.pdf. [Kasutatud 18 märts 2021].

[12] Blue Fish, „WHAT IS DOCUMENTUM,“ [Võrgumaterjal]. Saadaval: https://bluefishgroup.com/insights/ecm/what-is-documentum. [Kasutatud 23 märts 2021].

[13] Blue Fish, „DOCUMENTUM ACCESS CONTROL LISTS,“ [Võrgumaterjal]. Saadaval: https://bluefishgroup.com/insights/ecm/what-are-access-control-lists. [Kasutatud 24 märts 2021].

[14] Atlassian, „What is automated testing?,“ [Võrgumaterjal]. Saadaval: https://www.atlassian.com/continuous-delivery/software-testing/automated-testing. [Kasutatud 24 märts 2021].

[15] Red Hat, „What is CI/CD,“ [Võrgumaterjal]. Saadaval: https://www.redhat.com/en/topics/devops/what-is-ci-cd. [Kasutatud 18 märts 2021].

[16] LamdaTest, „All You Need To Know About Automation Testing Life Cycle,“ [Võrgumaterjal]. Saadaval: https://www.lambdatest.com/blog/all-you-need-to-know-about-automation-testing-life-cycle. [Kasutatud 23 märts 2021].

[17] Dustin, E., Garret, T., Gauf, B. (2009). Automated Software Testing: How to Save Time and Lower Costs While Raising Quality. Addison-Wesley Professional.

[18] SmartBear, „Test Automation Best Practices," [Võrgumaterjal]. Saadaval: https://smartbear.com/learn/automated-testing/best-practices-for-automation. [Kasutatud 22 märts 2021].

[19] Testim, „What Is a Test Environment? A Guide to Managing Your Testing," [Võrgumaterjal]. Saadaval: https://www.testim.io/blog/test-environment-guide. [Kasutatud 26 märts 2021].

[20] ProfessionalQA, „Test Execution Cycle," [Võrgumaterjal]. Saadaval: https://professionalqa.com/test-execution-cycle. [Kasutatud 23 märts 2021].

[21] MindTools, „Decision Matrix Analysis," [Võrgumaterjal]. Saadaval: https://www.mindtools.com/pages/article/newTED_03.htm. [Kasutatud 19 märts 2021].

[22] Selenium, [Võrgumaterjal]. Saadaval: https://www.selenium.dev. [Kasutatud 20 märts 2021].

[23] Selenide, [Võrgumaterjal]. Saadaval: https://selenide.org. [Kasutatud 20 märts 2021].

[24] Cucumber, [Võrgumaterjal]. Saadaval: https://cucumber.io. [Kasutatud 20 märts 2021].

[25] Cypress, [Võrgumaterjal]. Saadaval: https://www.cypress.io. [Kasutatud 20 märts 2021].

[26] Protractor, [Võrgumaterjal]. Saadaval: https://www.protractortest.org. [Kasutatud 20 märts 2021].

[27] TestNG, [Võrgumaterjal]. Saadaval: https://testng.org. [Kasutatud 20 märts 2021].

[28] JUnit 5, [Võrgumaterjal]. Saadaval: https://junit.org/junit5. [Kasutatud 20 märts 2021].

[29] Cocchiaro, C. (2018). Selenium Framework Design in Data-Driven Testing. Packt Publishing.

[30] FasterXML, „ObjectMapper," [Võrgumaterjal]. Saadaval: https://fasterxml.github.io/jackson-databind/javadoc/2.7/com/fasterxml/jackson/databind/ObjectMapper.html. [Kasutatud 21 märts 2021].

[31] Maven, „Maven Surefire Plugin," [Võrgumaterjal]. Saadaval: https://maven.apache.org/surefire-archives/surefire-3.0.0-M2/maven-surefire-plugin/index.html. [Kasutatud 26 märts 2021].

[32] Spring, „Spring Expression Language (SpEL)," [Võrgumaterjal]. Saadaval: https://docs.spring.io/spring-framework/docs/3.0.x/reference/expressions.html. [Kasutatud 18 märts 2021].

[33] TutorialsPoint, „Design Pattern - Factory Pattern," [Võrgumaterjal]. Saadaval: https://www.tutorialspoint.com/design_pattern/factory_pattern.htm. [Kasutatud 18 märts].

[34] Computer Hope, „Regex," [Võrgumaterjal]. Saadaval: https://www.computerhope.com/jargon/r/regex.htm. [Kasutatud 18 märts 2021].

[35] GitLab, [Võrgumaterjal]. Saadaval: https://docs.gitlab.com. [Kasutatud 24 märts 2021].

[36] Oracle, „Enum Types," [Võrgumaterjal]. Saadaval: https://docs.oracle.com/javase/tutorial/java/javaOO/enum.html. [Kasutatud 25 märts 2021].

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

Mina, Nikita Kums

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Kasutajaõiguste testimise automatiseerimine ettevõttes Fujitsu Estonia AS“ mille juhendaja on Aleksei Talisainen
   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.

15.05.2021

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.

Lisa 2 – Documentum'i ACL'id [13]

ACL	**Kirjeldus**
Puudub (inglise keeles *None*)	Kasutaja ei saa kunagi teada, et antud objekt on olemas.
Sirvimine (inglise keeles *Browse*)	Kasutaja saab vaadata objekti omadusi, kuid mitte selle sisu.
Lugemine (inglise keeles *Read*)	Kasutaja saab vaadata objekti omadusi ja selle sisu.
Sidumine (inglise keeles *Relate*)	Kasutaja saab lisada objektile märkmeid lisaks eelnevalt nimetatud õigustele.
Versioneerimine (inglise keeles *Version*)	Lisaks eelnevalt nimetatud õigustele saab kasutaja luua objekti uusi versioone, kuid mitte ülekirjutada praegust versiooni.
Kirjutamine (inglise keeles *Write*)	Kõik eelnevalt nimetatud õigused, kuid saab ülekirjutada ehk muuta praegust objekti versiooni.
Kustutamine (inglise keeles *Delete*)	Kõik eelnevalt nimetatu ning objekti kustutamise võimalus.

Lisa 3 – Kasutajaõiguste automatiseerimise otstarbekuse kinnitamine

Automatiseerimise kriteerium	Jah	Ei
Kas test on korduvalt käivitatav?	✓	
Kas test on sageli kasutatav?	✓	
Kas test katab kõige tähtsaimad funktsioonid?	✓	
Kas testi on võimatu teostada manuaalselt?	✓	
Kas test katab kõige keerulisemad kohad?	✓	
Kas test vajab palju erinevaid testandmete kombinatsioone samade sammude kordamiseks?	✓	
Kas eeldatavad testimise tulemused on pidevad ehk ei muutu iga testi käivituse tagant?	✓	
Kas test on väga ajakulukas?	✓	
Kas test teostatakse stabiilsel rakendusel?	✓	
Kas test peab olema kontrollitud erinevatel tarkvaradel ja riistvaradel?	✓	
Kas testi automatiseerimine investeeringutasuvus on paljutõotav?	✓	

Lisa 4 – Organisatsiooni andmete osa seadistuste failis

```
"customerConfigurationPath":
      "/casem/casemutil/src/main/resources/configuration/customers/Fujitsu/
      fujitsu.root.xml",
"excelPath":
      "/acl-automation/src/test/resources/configuration/excel/Fujitsu/
      Fujitsu_käyttövaltuudet_0.0.xlsx",
"adminUsername" : "dmadmin",
"adminPassword" : "password123",
"organizationId": "1.2.246.10.00000001",
"organizationAbbr" : "fujitsu0",
"organization" : "Fujitsu",
"env" : "fujitsu",
"nativeId": "00.00.01",
"nativeId2": "00.00.02",
"sharedFolderId" : "/Yhteiset kansiot/Fujitsu",
"prod": true,
```

Lisa 5 – Kasutajate osa seadistuste failis


```
"testUserDataWrapper": {
    "users": [],
    "password": "password123",
    "testUserData": [
      {
        "username": "Registrar",
        "group": true,
        "processUser": false,
        "groups": [
          "c_#entryOrgAbbrev_all_users",
          "c_#entryOrgAbbrev_registrar",
          "c_#entryOrgAbbrev_registry",
          "c_#entryOrgAbbrev_all_registrars"
        ]
      },
      {
        "username": "SubFunctionModifyer",
        "group": true,
        "processUser": false,
        "groups": [
          "c_#entryOrgAbbrev_all_users",
          "c_#entryOrgAbbrev_se_#nativeIdHere_w",
          "c_#entryOrgAbbrev_se_#nativeId2Here_w",
          "c_#entryOrgAbbrev_all_subfunctionmodifiers"
        ]
      },
      {
        "username": "Draftsman",
        "group": false,
        "processUser": false,
        "amsField": "ams_draftsman",
        "groups": [
          "c_#entryOrgAbbrev_all_users"
        ]
      },
      {
        "username": "RecordSigningTaskReceiver",
        "group": false,
        "processUser": true,
        "amsProcessName": "signing",
        "amsReceiverType": "Action",
        "groups": [
          "c_#entryOrgAbbrev_all_users"
        ]
      }
    ]
  }
```

Lisa 6 – Kasutajaliidese eriliste omadustega väljade osa seadistuste failis

```
"aclUIFields": {
  "caseFields": {
    "secretMetadataField": "ams_restricted_notes",
    "secretMetadataFieldGroup": "basic",

    "nonRestrictedInheritedTosField": "ams_keywordscode",
    "nonRestrictedInheritedTosFieldGroup": "additionalInformation",

    "restrictedInheritedTosField": "ams_retentionreason",
    "restrictedInheritedTosFieldGroup": "conservation",

    "otherMetadataField": "ams_title",
    "otherMetadataFieldGroup": "basic",

    "agentsField": "ams_draftsman"
  },

  "recordFields": {
    "secretMetadataField": "ams_restricted_description",
    "secretMetadataFieldGroup": "basic",

    "nonRestrictedInheritedTosField": "ams_direction",
    "nonRestrictedInheritedTosFieldGroup": "basic",

    "restrictedInheritedTosField": "ams_retentionreason",
    "restrictedInheritedTosFieldGroup": "conservation",

    "otherMetadataField": "ams_delegate",
    "otherMetadataFieldGroup": "reception_information"
  },

  "actionFields": {
    "secretMetadataField": "ams_restricted_notes",
    "otherMetadataField": "ams_maindescription",
    "agentsField": "ams_responsibleperson"
  }
}
```

Lisa 7 – Testimisüsteemi seaditamise parameetrid läbi torujuhtme

Parameeter	Andmetüüp	Väärtus (näide)	Kirjeldus
FULL_TEST	Tõeväärtus	---	Kui tõene, jooksutatakse kõikide organisatsioonide testid
CUSTOMER_TEST	Sõne	FUJITSU_TEST	Kindla organisatsiooni testi käivitamine
ONLY_RUN_ROWS_WITH_RULE_NR	Sõne	1.1.1,2.1.2,3.1.5	Testida ainult antud kasutajaõiguste exceli reeglite numbreid
USERS	Sõne	Registrar,Draftsman	Testida ainult antud kasutajad
RUN_ACL_TESTS	Tõeväärtus	---	Kui väär, ei käivitata API ACL testid
RUN_ACL_UI_TESTS	Tõeväärtus	---	Kui väär, ei käivitata kasutajaliidese ACL testid
RUN_OPERATION_TESTS	Tõeväärtus	---	Kui väär, ei käivitata operatsioonide testid

Lisa 8 – Automaattestimissüsteemi käivitamise torujuhtme etapp

```
Run Fujitsu tests:
  stage: Run Fujitsu tests
  allow_failure: true
  tags:
    - casem_ssh_node02
  script:
    - cd $BUILD_FOLDER_PATH/$CI_COMMIT_SHA/acl-automation
    - docker pull markhobson/maven-chrome:jdk-11
    - docker run --rm --user $(id -u):$(id -g) -v
           "$PWD":/acl-automation   -v
           "$BUILD_FOLDER_PATH/$CI_COMMIT_SHA/casem":/casem -w
           /acl-automation markhobson/maven-chrome:jdk-11
           mvn $ADDITIONAL_MAVEN_CLI_OPTS
           $EDITABLE_PROJECT_CONF_VALUES
           -DsuiteXmlFile=_Fujitsu_.xml test
  only:
    variables:
      - ($CI_PIPELINE_SOURCE == "web" ||
         $CI_PIPELINE_SOURCE == "schedule") &&
         $CUSTOMER_TEST == "FUJITSU_TEST"
```

Lisa 9 – Klassi *TestResultState* struktuur

```
public enum TestResultState {
    PASS("Pass"),
    PASS_RUN_DEPENDENT("Pass run dependent"),
    PASS_CONDITIONAL("Pass conditional"),
    SKIP("Skip"),
    FAIL("Fail"),
    FAIL_RUN_DEPENDENT("Fail run dependent"),
    NOT_RUN("Not run"),
    CROSSED("---")
    ;

    private final String description;

    TestResultState(String description) {
        this.description = description;
    }

    public boolean isFail() {
        return
            this == TestResultState.FAIL ||
            this == TestResultState.FAIL_RUN_DEPENDENT;
    }

    public boolean isPass() {
        return
            this == TestResultState.PASS ||
            this == TestResultState.PASS_RUN_DEPENDENT ||
            this == TestResultState.PASS_CONDITIONAL;
    }

    @Override
    public String toString() {
        return description;
    }
}
```