TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
TTÜ IT KOLLEDŽ

Piret Samuel 201309IDSR

MANUAALSE TESTIMISE OSALINE AUTOMATISEERIMINE SYNERALL AS TIIMIS

Diplomitöö

Juhendaja: German Mumma
Magister

Tallinn 2020

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: Piret Samuel

17.05.2020

Annotatsioon

Käesoleva diplomitöö eesmärgiks on kommunaalteenuseid pakkuvatele ettevõtetele loodud kliendiinfo- ja arveldussüsteemi Synerall testimise protsessi tõhustamine.

Testimist on siiani läbiviidud manuaalselt. Mitmed testimistegevused Synerallis on korduva iseloomuga ja võtavad märkimisväärse osa testimisele kuluvast ajast. Rutiinsete tegevuste pidev kordamine ei mõju hästi ka inimese rahulolule oma tööga. Neist põhjustest tulenevalt otsustati need tegevused automatiseerida.

Käesoleva diplomitöö käigus analüüsitakse, milliste vahenditega on mõistlik automatiseerimine teostada ja millised tegevused on kasulik automatiseerida. Töö käigus valmis lahendus, mis teostati kasutades automatiseerimisvahendit Cypress. Koodi dubleerimise vältimiseks ja parema hallatavuse eesmärgil järgiti *Page Object* mudelit. HSY Syneralli keskkonda AD-kontoga sisselogimiseks kasutati eraldi pluginat cypress-ntlm-auth. Syneralli eri instantside erinevad väärtused otsustati salvestada keskkonnamuutujatena vastava keskkonna konfiguratsioonifaili ja käivitada Cypress koos vastava konfiguratsiooniga.

Pärast lahenduse valmimist mõõdeti korduvtegevuste automatiseeritult sooritamiseks kuluvat aega ja võrreldi seda ajaga, mis kulub tegevuse sooritamiseks manuaalselt. Lahendus osutus inimesest keskmiselt 5 korda kiiremaks. Saavutatud ajaline võit oli keskmiselt 23 minutit päevas. Tehtud töö tulemusena on vähenenud rutiinse töö määr testija töös ja testija saab keskenduda testloo sisulisele poolele. On loodud vundament kasutajaliidese suitsu- ja regressioonitestide loomiseks.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 46 leheküljel, 7 peatükki, 19 joonist, 4 tabelit.

Abstract

Partial Automation of Manual Testing in Synerall Team

A purpose of the current thesis is an enhancement of a testing process of a customer information and billing system Synerall.

Until now, testing in Synerall has been conducted manually. However, some testing activities in Synerall have a repetitive nature and performing those consumes considerable amount of time. Repeating routine tasks for a long time also affects job satisfaction. Due to those reasons it was decided to automate the repetitive testing activities.

In the course of the current thesis it was analyzed what tools are optimal for implementing the automation and what flows exactly should be automated.

In the course of the current thesis a solution was developed using test automation tool Cypress. In order to avoid code duplication and to ensure better code maintenance *a Page Object Model* was followed. To log into HSY Synerall using AD account, a separate plugin – cypress-ntlm-auth – was used. Different values of Synerall instances were decided to be saved as environment variables into a configuration file of a respective environment and respective configuration to be specified each time when opening Cypress.

After the flows were automated, it was measured how much time it takes to perform the repetitive activities automatically and how it compares to performing the same activities manually. The automation solution turned out to be about 5 times faster than human. Time gained *via* automation was about 23 minutes per day.

As a result, the amount of routine activities testers perform at work has been reduced. Testers can now better focus on test cases and substantial issues. A foundation for creating smoke and regression tests has been created.

The thesis is written in Estonian and contains 46 pages of text, 7 chapters, 19 figures, 4 tables.

Lühendite ja mõistete sõnastik

AVP	Andmevahetusplatvorm
Brauseriülene testimine	Veebisaidi või veebirakenduse testimine erinevates veebilehitsejates, aga ka eri operatioonisüsteemides töötavates veebilehitseja versioonides garanteerimaks, et veebilehe renderdamine vastab kavatsetule [1].
CIS	*Customer Information System*
EDIEL	*Electronic Data Interchange for the Electricity Industry*
Elektritarnija	Elektritarnija varustab elektri tarbijat elektriga. Tarbijal on õigus tarnijat valida kõigi tarnijate seast, kes antud võrgupiirkonnas teenust pakuvad.
Funktsionaalne testimine	Tarkvara nõuetekohaste funktsionaalsuste testimine.
HER	*Herrfors Ab*
Heuristik	Lihtne ja tõhus, ent siiski mitte eksimatu suunis, mis aitab lahendada probleeme ja teha otsuseid [2].
HSY	*Helsingin seudun ympäristöpalvelut*
IDE	*Integrated DevelopmentEnvironment*
IntelliSense	Katustermin, mis hõlmab erinevad koodiredigeerimise funktsionaalsusi nagu näiteks koodi automaatlõpetamine, kiirinfo ja parameetriinfo [3].
ISS	*Imatran Seudun Sähkö*
KER	*Keravan Energia*
KOK	*Kokkolan Energia*
KSAT	*Koillis-Satakunnan Sähkö*
KSOY	*Kymenlaakson Sähkö*
Liitumisleping	Liitumisleping luuakse kommunaalteenuse tarbija ja võrguoperaatori vahel uue liitumise, liitumise muudatuse ja ka omanikuvahetuse puhul.
Manuaalne testimine	Testimise teostamine manuaalselt ilma automatiseerimise vahendeid kasutamata.
MIT litsents	Tarkvara litsents, mis lubab tarkvara kommertskasutamist, modifitseerimist, levitamist, privaatset kasutamist, hõlmamist piiravama litsentsiga süsteemi tingimusel, et autoriõiguse ja litsentsi viide on lisatud igasse koopiasse või modifitseeritud

	töösse. Autor oma tarkvara kasutamisest tuleneda võivate kahjude eest vastutav ei ole [4].
npm	Node paketihaldur JavaScript programmeerimiskeele jaoks.
NuGet	NuGet on paketihaldur, mis võimaldab arendajatel luua, jagada ja kasutada kasulikke .NET teeke.
Planning poker	Konsensusel põhinev tehnika tööde mahu hindamiseks tarkvaraarenduses [5].
PO	*Page Object*
POM	*Page Object Model*
Regressioonitestimine	Testimine, mille eesmärk on tagada, et uus lisatud kood või olemasoleva koodi muudatus pole teinud katki olemasolevat funktsionaalsust [6].
Suitsutestimine	Testimine, mille eesmärk on tagada, et testitava tarkvara kõige elementaarsem funktsionaalsus töötab korrektselt [7].
Elektri tarneleping	Leping, mille elektri tarbija sõlmib elektritarnijaga.
TFS	*Team Foundation Server*
Elektri võrguleping	Leping, mille elektri tarbija sõlmib elektri jaotusvõrgu operaatoriga, kes hakkab pakkuma võrguteenust.
Võrguoperaator	Kohaliku energiataristu haldaja. Igale kohalikule võrgupiirkonnale vastab üks võrguoperaator – seda tarbija valida ei saa.

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Synerall on kliendiinfosüsteem ja arvelduslahendus, mis on optimeeritud kommunaalteenuseid pakkuvate ettevõtete jaoks. Enamus Syneralli klientidest on hetkel Soome elektrienergia- ja kaugkütteettevõtted.

Klientide arv on viimase paari aasta jooksul oluliselt tõusnud ja ka arendustiim vastavalt kasvanud. Et püsida konkurentsivõimelisena, on tarvis kohandada ja optimeerida tarkvaraarendusprotsesse.

Käesoleva töö teemaks on testimisprotsessi tõhustamine: hetkel täielikult manuaalse testimisprotsessi osaline automatiseerimine.

Töö teise osa esimeses alajaotuses antakse ülevaade ettevõttest ja arendatavast tootest, teises alajaotuses kirjeldatakse arendus- ja testimisprotsessi hetkeolukorda ning kolmandas alajaotuses kirjeldatakse probleem ja sõnastatakse töö eesmärk.

Töö kolmandas osas analüüsitakse, milline võiks olla optimaalne viis püstitatud eesmärgi saavutamiseks: esimeses alajaotuses kaalutakse, millised vahendid on antud ülesande teostuseks optimaalsed ning teises alajaotuses kirjeldatakse meetodit, mida ülesande lahendamisel järgitakse.

Neljandas osas otsustatakse, millised tegevused automatiseerida. Seejärel teostatakse automatiseerimine.

Viienda osa esimeses alajaotuses viiakse läbi mõõtmised ja võrreldakse manuaalse ja automatiseeritud tegevuse sooritamise peale kuluvat aega. Teises alajaotuses valideeritakse tehtud töö seatud eesmärkide suhtes.

Kuuendas osas tehakse ettepanekud võimalike edasiarenduste osas.

Seitsmendas osas tehakse diplomitöö tulemustest kokkuvõte.

2 Taust

Selles peatükis antakse ülevaade ettevõttest ja arendatavast tootest, kirjeldatakse arendustiimi, arendusmetoodikat ja testimisprotsessi hetkeolukorda. Seejärel kirjeldatakse probleem ja püstitatakse töö eesmärk.

2.1 Ettevõtte ja toote kirjeldus

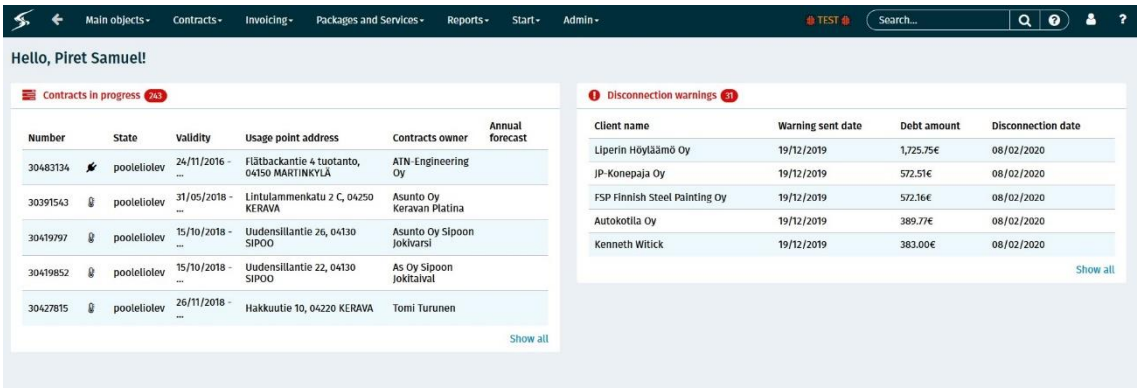
Syneralli tarkvara arendamist alustati aastal 2007 tarkvaraarendusfirma Net Group OÜ Syneralli tiimis. 2016. aasta augustis loodi Syneralli tiimist eraldi tütarettevõte, kliendihaldus- ja arvelduslahenduste pakkuja Synerall AS, kus töötab hetkel 14 inimest.

Eesti klientideks on Imatra Elekter AS, VKG Elektrivõrgud OY ja VKG Soojus AS ning Soome klientidest kasutavad Syneralli tarkvara Oy Herrfors Ab, Keravan Energia Oy, Vihreä Älyenergia Oy, Koillis-Satakunnan Sähkö Oy, Kokkolan Energia Oy, Kymenlaakson Sähkö Oy, Ilmatar Windpower Oyj, Imatran Seudun Sähkö Oy ja HSY.

Synerall tarkvara koosneb kommunaalteenuseid pakkuvate ettevõtete töötajatele mõeldud kliendiinfosüsteemist (CIS) ja kommunaalteenuste tarbijaile loodud iseteeninduspordaalist.

2.1.1 CIS

CISi avalehel (Joonis 1) kuvatakse kasutajale erinevaid infoplokke vastavalt sellele, kuidas ta on avalehe oma profiililehel seadistanud: antud puhul näidatakse pooleliolevaid lepinguid ja katkestusteateid.



Joonis 1. CISi avaleht.

Veebilehe ülaosas on veebisaidi peamenüü, mis kuvatakse veebisaidi igal veebilehel. Erinevates Syneralli instantsides võivad menüüpunktid mingil määral varieeruda. Kuvatavad menüüpunktid sõltuvad näiteks sellest, kas Syneralli instants on kasutusel ettevõttes, mis on ainult elektrimüüja või ettevõttes, mis on nii elektrimüüja kui ka võrguoperaator. Lisaks sõltub menüüpunktide kuvamine ka ettevõtte poolt pakutava kommunaalteenuse tüübist ja kasutaja rollile antud õigustest.

Esimese menüüpunkti „Põhiobjektid“ alt on võimalik navigeerida näiteks klientide, objektide, katkestuste, arvesti mudelite ja arvestite nimekirja. Nimekirja elemendid on lingitud, viies vastava elemendi detailvaatele. Nimekirja vaatel on võimalik luua uusi põhiobjekte (vastavalt siis kliente, objekte jne).

Teine menüüpunkt „Lepingud“ võimaldab navigeerida näiteks müügilepingute, liitumislepingute ja katkestusteadete nimekirja ja sealt edasi vastavatele detailvaadetele.

Kolmas menüüpunkt „Arveldus“ võimaldab navigeerida näiteks arvete, maksete, arvestinäitude ja teostatud teenuste nimekirja ja sealt edasi vastavatele detailvaadetele; lisaks on selle menüüpunkti all lingid arvete masskoostamise veebilehele, arvete massväljastamise, maksete impordi ja näitude impordi veebilehele.

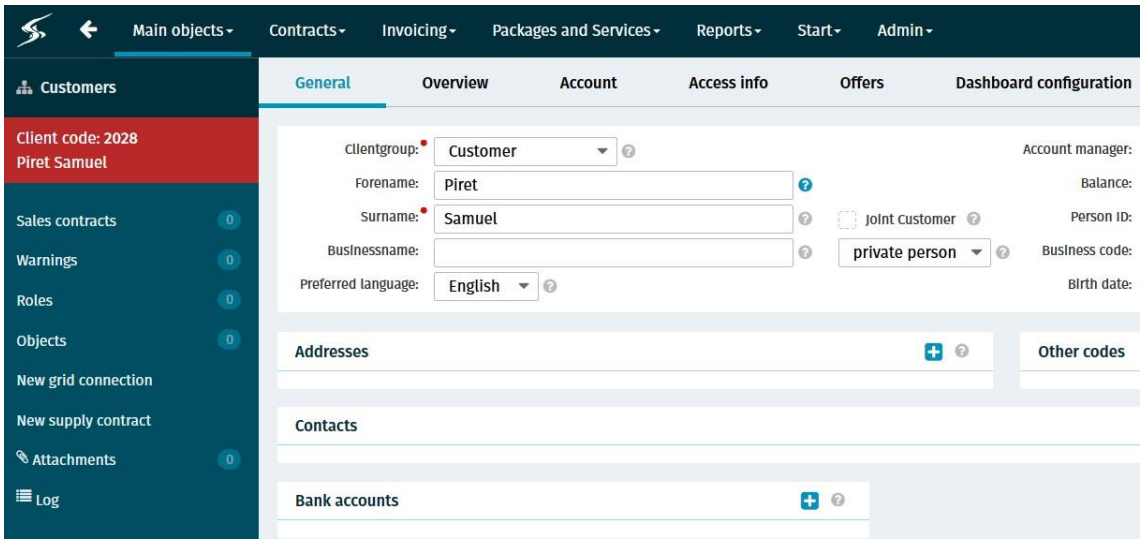
Neljas menüüpunkt „Paketid ja teenused“ sisaldab linke toodete ja teenustega seotud veebilehtedele. Näiteks on siit võimalik navigeerida võrgutoodete hinnakirjade nimekirja, tarnelepingu toodete nimekirja, kampaaniate, toodete, teenuste ja tootekomponentide nimekirja.

Viies menüüpunkt „Aruanded“ sisaldab erinevaid raporteid.

Kuuenda menüüpunkti „Start“ kaudu on võimalik navigeerida näiteks rakenduse sessioonide, süsteemi logi, teavituste ja EDIEL/AVP sõnumite nimekirja.

Seitsmes menüüpunkt „Admin“ kuvatakse ainult administraatori õigustega kasutajale ja sisaldab linke erinevatele konfiguratsiooniga seotud veebilehtedele, muuhulgas rakenduse seadistuste, tõlgete lehele, taustatööde ja rakenduses kasutatavate mallide lehele.

Peale peamenüü on Synerallis kasutusel ka kontekstimenüüd, mis asuvad detailvaate vasakservas. Näiteks kliendi detailvaate (Joonis 2) kontekstimenüü kaudu on võimalik navigeerida selle kliendi lepingutele, näha teda puudutavaid katkestushoiatusi, selle kliendi rolle ja talle kuuluvaid objekte, alustada uue võrgu- ja tarnelepingu loomist, salvestada ja avada manuseid ning vaadata selle kliendiga seotud tegevuste logi.



Joonis 2. Kliendi detailvaade.

2.1.2 Iseteenindusportaal

Iseteenindusportaal võimaldab kommunaalteenuse tarbijal luua uusi tarnelepinguid, sisestada näite, saada ja aktsepteerida uusi pakkumisi, näha oma lepinguid ja arveid, tarbimise ajalugu ja hallata isiklikku kontaktinfot. Iseteenindusportaal on ainult elektriharu klientidel, HSY-l portaal puudub.

2.2 Ülevaade arendus- ja testimisprotsessi hetkeolukorrast

Testimisprotsess on osa üldisest tarkvaraarendusprotsessist. Tarkvaraarendus Synerallis kasutab agiilset lähenemist. Agiilne, nii nagu ka traditsiooniline lähenemine, on katustermin, mis hõlmab erinevaid metodoloogiaid ja raamistikke: näiteks XP, Scrum, Kanban ja Crystal Clear.

2.2.1 Kasutatav tarkvara arendusprotsessi raamistik

Synerallis on kasutusel Scrumban, mis on hübriid Scrumist ja Kanbanist. Scrumi ja Kanbani kohta võib lähemalt lugeda lisadest: Lisa 1 ja Lisa 2.

Scrumist lähtuvalt on Synerallis kasutusel 3 Scrumi sündmust, milleks on Sprindi planeerimine, Sprindi tagasivaated ja igahommikused püstijalakoosolekud ning 3 Scrumi *artefakti*, milleks on Toote kuhi, Sprindi kuhi ja Sprindi langustrendi graafik. Lisaks on esindatud Scrumi 2 rolli: toote omanik ja arendustiim: 3ndat rolli – Scrum meistrit – pole kedagi täitma määratud. Iteratsiooni, mida Scrumi puhul nimetatakse sprindiks, pikkus Synerallis on üks nädal.

Kanbanist lähtuvalt on Synerallis kasutusel Kanban tahvel, mis aitab visualiseerida töövoogu. Kanban tahvli igal töövoo jaamal on oma WIP limiit, mis aitab voogu tasakaalustada ja vältida olukorda, kus korraga on liiga palju tööd käsil. Kui üks töö lõpetatakse, siis tõmmatakse järgmine kõrgeima prioriteediga töö tegevusse.

2.2.2 Tiim ja kasutusel olevad töövahendid

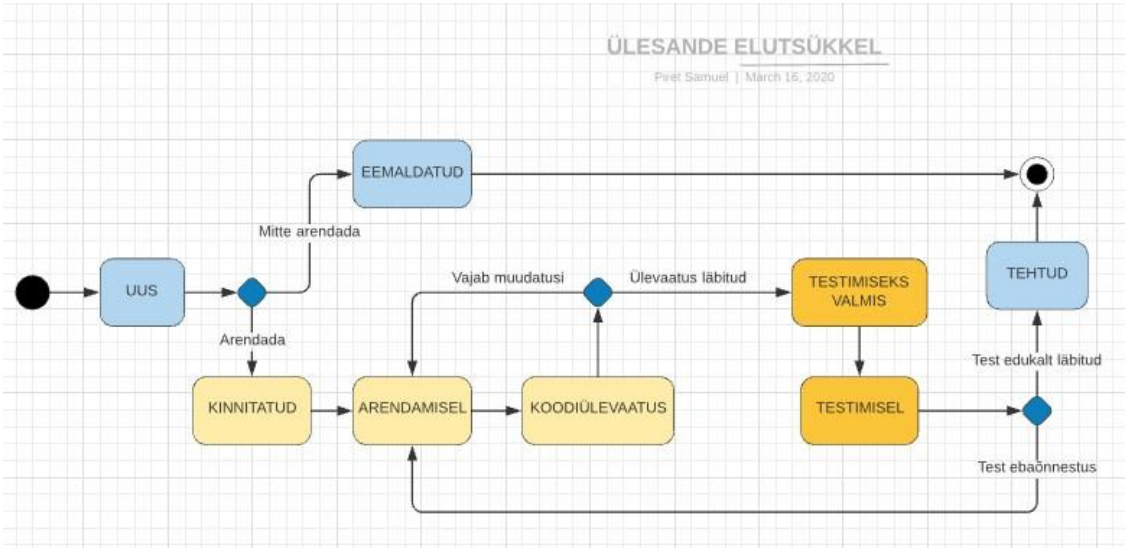
Synerallı tiimis on hetkel 2 põhikohaga testijat; lisaks kasutab Synerall võimalusel ettevõtte ASA Quality Services testija teenust osalise koormusega vastavalt vajadusele ja võimalusele. Testija põhiülesandeks on süsteemi manuaalne funktsionaalne testimine ning ühtlasi kasutajajuhendite loomine ja haldamine. Tiimi peale on 8 arendajat. Mõned Synerallı tiimi arendajad on Net Group OY töötajad, kelle teenuseid Synerall kasutab. Meeskonnas on 1 arhitekt-analüütik, kes on ühtlasi tooteomaniku rollis ja kes on peamine ühenduslüli kliendi ja tiimi vahel. Lisaks on Synerallis 3 kasutajatoe pakkujat, kes igapäevaselt tegelevad tootmiskeskkondadest pärit probleemide ja intsidentide haldamisega. Vastavalt vajadusele ja võimalusele on kasutajatugi abiks ka testimisel. Tiimi liikmed töötavad reeglina ühes ruumis.

Arendusraamistikuks on Synerallis .NET. Peamine arenduskeel on C#, aga *front-endi* puhul ka Javascript. Relatsioonilise andmebaasi haldussüsteemiks on MS SQL Server.

Peamised töövahendid Synerallis on TFS (Kanban tahvel, tootekuhi, sprindikuhi, versioonihaldus), MS Teams (suhtlemiseks, koosolekute pidamiseks), Outlook (e-mail), Visual Studio koodiredaktor, SQL Server Management Studio ja Sharepoint (dokumentatsiooni haldamiseks).

2.2.3 Synerallі testimise protsessi kirjeldus

Agiilsele arendusele omaselt toimub jooksvalt tihe koostöö arendajatega. Esmaspäevahommikuti toimub sprindi planeerimine, mille käigus valitakse tootekuhjast uue sprindi tööd, iga töö analüüsitakse läbi ning kaalutakse, milline on parim lahendus töö teostamiseks. Seejärel kirjutatakse töödele vastuvõtukriteeriumid ja antakse hinnang töö keerukusele *(planning poker*). Hinnatud töö liigutatakse Kanban tahvli jaamast „Uus“ jaama „Kinnitatud“, kust arendajad sprindi jooksul töid ette võtavad, liigutades need edasi jaama „Arendamisel“. Kui töö on arendatud, nihutatakse see koodiülevaatuse jaama. Pärast koodiülevaatuse läbimist on töö testimiseks valmis. Testija tõmbab uusi pileteid jaamast „Testimiseks valmis“ jaama „Testimisel“. Kui arendatud töö ei vasta vastuvõtukriteeriumitele, siis liigutatakse see tagasi jaama „Arendamisel“. Kui töö läbib testimise edukalt, suunatakse see jaama „Tehtud“. Ülesande elutsüklit illustreerib Joonis 3.



Joonis 3. Ülesande elutsükkel.

Hetkel on Synerallis kasutusel 2 kanban tahvlit: üks on HSY projekti jaoks ja teine katab kõik ülejäänud projektid. See tähendab seda, et testija jälgib pidevalt 2 tahvlit.

Valdav osa Syneralli testimisest on manuaalne funktsionaalne testimine. Regressioonitestimist regulaarselt ei tehta. Testimiseks vajalik info on vastava TFS pileti all kirjas: funktsionaalsuse kirjeldus või defekti kordamise sammud, vastuvõtukriteeriumid ja manused, milleks on näiteks selgitavad ekraanitõmmised või malli näidised. Testlugusid eraldi ei dokumenteerita: keerulisemate funktsionaalsuste korral testija kirjutab vastavalt vajadusele erinevad testlood omale ise välja, kas faili või paberile.

Üldjuhul sooritatakse testimine kasutades üht brauserit, näiteks Firefoxi: põhjuseks on see, et brauseri eripäradest tulenevaid vigu tuleb ette siiski harva ja reeglina pole need kriitilised vead. Samas jällegi on oluline testida funktsionaalsus läbi erinevates Syneralli instantsides: seda tuleb aeg-ajalt ette, et ühes instantsis testides toimib kõik ootuspäraselt, aga teises mitte. Osad vead, mis erinevates keskkondades testimise käigus ilmnevad, pole otseselt testitava funktsionaalsuse vead. Näiteks testides 6 soome kliendi keskkonnas viga, mis on seotud Soome e-arve formaadiga Finvoice, tuli 2 testitavas keskkonnas välja, et teenusarve salvestamisel andis rakendus vea. Antud puhul otsustati teenusarve neis keskkondades teha sel põhjusel, et tarbimisarve tegemiseks puudusid keskkonnas vajalikud andmed, andmete importimine oleks olnud aeganõudev ja teenusarve oli selle vea testimiseks piisav. Võimalik, et see viga poleks selle taski testimise käigus ilmnenud, kui oleks piirdutud vaid 1-2 keskkonnas testimisega.

2.3 Probleemi- ja eesmärgipüstitus

Nagu juba eelpool sai öeldud, on Synerallis siiani testimine toimunud manuaalselt. Testija igapäevatöös kulub keskmiselt ligi pool tundi päevas korduvatele tegevustele.

Valdav osa Syneralli vigadest ja uuest funktsionaalsusest on seotud kas lepingute või arveldamisega. Selleks, et saaks testida näiteks elektri võrgulepingu funktsionaalsust, tuleb testijal eelnevalt teha läbi järgmised sammud:

1. luua uus klient (erakliendi isikukood ja ärikliendi registrikood genereeritakse veebirakenduses)

2. lisada kliendile vajalikud kontaktid,

3. luua objekt: oma võrgu kasutuskoha puhul hõlmab see samm objekti üldandmete, liitumispunkti andmete ja kasutuskoha andmete sisestamise,

4. luua mõõdik ja paigaldada see objektile,

5. sisestada võrgulepingu üldandmed,

6. luua võrgulepingule mõõtelahendus,

7. sisestada lepingu teenuste kontaktandmed.

Elektri tarnelepingu funktsionaalsuse testimiseks tuleb oma võrgu kasutuskoha puhul luua kõigepealt võrguleping ja alles seejärel tarneleping. Elektri liitumislepingu loomisel on tarvis, et oleks olemas klient ja objekt, millele on paigaldatud arvesti (so. sammud 1-4). Arveldamise testimine eeldab sageli nii võrgu- kui ka tarnelepingu olemasolu. Nagu näha, tuleb praegu suure osa testimise puhul teha manuaalselt läbi eelnimetatud sammud 1-7. See ei ole aja efektiivne kasutamine.

Allpool esitatud tabelis (Tabel 1) on toodud põhilised korduvtegevused, ühe tegevuse ühekordseks sooritamiseks kuluv aeg, tegevuse keskmine kordumise sagedus päeva peale ja aeg, mis keskmiselt kulub päevas selle tegevuse sooritamiseks. Elektriharu ja veeharu (HSY) on mõõdetud eraldi, sest kuigi kliendi ja arvesti loomine ja arvesti paigaldamine on mõlemas üsna võrreldavad ning ka elektriharu võrguleping ja veeharu tarbimisleping on analoogsed, siis nende objektide ja lepingute loomise vormid on ikkagi piisavalt erinevad, mistõttu on neid mõistlik eraldi mõõta. Tarneleping on kasutuses ainult elektriharus. Sagedus päevas on arvutatud 2020. aasta 5 esimese sprindi alusel. Selles ajaperioodis oli põhirõhk HSY arendustöödel, mistõttu elektriharu arvud selles tabelis on väiksed. Näiteks elektri tarnelepingute testimine on reeglina üsna sage tegevus, aga mitte sellel perioodil.

Kuna tegemist on Syneralli elementaarsete objektide loomisega, siis see, et antud perioodil on proportsioonid tugevalt HSY kasuks, ei mängi siin suurt rolli – ainus märkimisväärne erinevus veeharu ja elektriharu vahel on tarnelepingute puudumine veeharu puhul. Muus osas on erinevused minimaalsed.

Mõõtmised on läbi viidud nii, et tegevusi on mõõdetud üksteisest sõltumatult. See tähendab seda, et objekti loomisel ei ole sisse arvestatud arvesti loomise ega paigaldamise aega ja lepingute puhul on eeldatud, et objekt koos installeeritud arvestiga ja omanikuga on juba olemas. Liitumismuudatuse ja omanikuvahetuse puhul on eeldatud, et objektil on juba kehtiv liitumisleping. Tulemused on ümardatud 2 kohani pärast koma.

Tabel 1. Korduvtegevuste peale kuluv aeg päevas.

Tegevus	Aeg (min)	Sagedus	Aeg*Sagedus
ELEKTRIHARU			
Kliendi loomine	1,77	0,88	1,56
Mõõdiku loomine	0,42	0,88	0,37
Oma võrgu objekti loomine	1,28	0,88	1,13
Arvesti paigaldamine	0,53	0,88	0,47
Võrgulepingu loomine	1,6	0,72	1,15
Tarnelepingu loomine	1,17	0,68	0,8
Liitumislepingu loomine (uus liitumine)	1,92	0,08	1,15
Liitumislepingu loomine (liitumise muudatus)	2,07	0,08	0,17
Liitumislepingu loomine (omanikuvahetus)	1,43	0	0
VEEHARU (HSY)			
Kliendi loomine	1,75	3	5,25
Arvesti loomine	0,35	2,55	0,89
Objekti loomine	1,08	2,55	2,75
Arvesti paigaldamine	0,52	2,55	1,33
Vee tarbimisleping	1,47	3,45	5,07
Liitumislepingu loomine (uus liitumine)	1,83	2,55	4,67
Liitumislepingu loomine (liitumise muudatus)	1,97	0,85	1,68
Liitumislepingu loomine (omanikuvahetus)	1,43	0,5	0,72
Korduvate tegevuste peale keskmiselt kuluv aeg päevas kokku:			**29,16 minutit**

Antud diplomitöö fookuses olev probleem on Syneralli infosüsteemi manuaalse testimise ebatõhusus: teatud tegevusi tuleb päeva jooksul sooritada korduvalt. Orienteeruv aeg, mis päevas selliste tegevuste sooritamise peale kulub, on peaaegu pool tundi. Terve kuu peale kulub nii juba üle 9 tunni ehk rohkem kui 1 tööpäev. Manuaalse testimise üheks

puuduseks on lisaks see, et inimene aeg-ajalt eksib, eriti kui pikk päev juba seljataga. Ühtlasi kipub samade tegevuste kordamisel mõte minema autopiloodile ja nii võib kaduda fookus testloo testimisel. Kui mitte otseselt, siis kaudselt mõjutab testimise protsessi ka rutiinsete tegevuste pikaajalise sooritamise psühholoogiline pool: 2019. aasta Diffblue arendajate uuringu tulemuste [8] põhjal võib väita, et rutiinsete tegevuste osakaal tööl mõjutab arendaja tööga rahulolu määra: 86% arendajaist tunnistasid, et rutiinsete ülesannete automatiseerimise võimalus on faktor, mis nende tööga rahulolu mõjutab. Testija on samasugune inimene nagu arendaja ja eelistab samamoodi tegeleda pigem ülesannetega, mis loovad väärtust ja mille kaudu on võimalik edasi areneda. Tööga rahulolev inimene on suurema tõenäosusega tõhusam testija.

Kirjeldatud probleemist lähtuvalt on käesoleva diplomitöö eesmärgiks:

- suurendada hetkel manuaalse testimisprotsessi efektiivsust korduvsooritatavate tegevuste automatiseerimisega ehk minna üle osaliselt automatiseeritud testimisele. Automatiseeritult võiks kuluda selliste tegevuste peale umbes 5 korda vähem aega,
- vähendada inimlikest eksimustest tulenevaid vigasid,
- võimaldada testijal keskenduda eelkõige testimise sisulisele poolele,
- parandada Syneralli testija tööalaseid arenemisvõimalusi ja seeläbi tööga rahulolu: lisaks testitava rakenduse äriloogika tundmaõppimisele, mis on kindlasti väga tähtis manuaalse testija töös, on automatiseerides võimalik arendada ka tehnilisemat laadi teadmisi ja oskusi,
- luua alus selleks, et Syneralli testijad ka edaspidi püüaks leida viise, kuidas oma tööd automatiseerides tõhusamaks ja huvitavamaks muuta,
- luua alus Syneralli kasutajaliidese suitsu- ja regressioonitestide loomiseks.

3 Lahenduse valik

Automatiseerimine on tehnoloogia kasutamine sooritamaks ülesandeid ilma inimese sekkumiseta. Automatiseerimist teostatakse tootmisvaldkonnas, robootikas, sõidukite kontrollimisel ja muudes valdkondades, sealhulgas IT-maailmas [9].

Tarkvara testimise valdkonnas võib eristada 2 terminit – testide automatiseerimine ja automatiseeritud testimine. Testide automatiseerimine on testide arendamine, aga automatiseeritud testimine on arendatud automaattestide jooksutamine [[10], lk 16]. Käesoleva töö kontekstis see tähendab seda, et 4. peatüki teemaks on testide osaline automatiseerimine, mille tulemusena on võimalik üle minna manuaalselt testimiselt osaliselt automatiseeritud testimisele.

Kuna töö eesmärgiks on automatiseerida korduvsooritatavaid manuaalseid testimistegevusi, mis moodustavad mingi osa kogu testloost, siis on mõistlik viia läbi automatiseerimine samuti kasutajaliidese tasemel. Brauseri automatiseerimine on siiski lähedasem viisile, kuidas lõppkasutaja süsteemiga suhtleb. See väldib ka ohtu, kus mingid tegevused sooritatakse pidevalt kasutajaliidest kaasamata ja märkamata võib jääda mõni defekt, mis ilmneb ainult kasutaja interaktsioonis rakendusega brauseri kaudu.

Töö skoobis on:

- 8 soome elektriharu klienti. Elektriharu klient võib olla ainult elektritarnija (näiteks Ilmatar Windpower Oyj ja Vihreä Älyenergia Oy) või nii tarnija kui ka jaotusvõrguoperaator (näiteks Oy Herrfors Ab, Keravan Energia Oy, Vihreä Älyenergia Oy, Koillis-Satakunnan Sähkö Oy, Kokkolan Energia Oy, Kymenlaakson Sähkö Oy ja Imatran Seudun Sähkö Oy)
- 1 soome veeharu klient (HSY)

Kuigi ka iseteenindusportaali kaudu on võimalik luua elektritarnelepinguid, jääb see antud töö skoobist välja - skoobis on ainult CIS. Eesti kliendid Imatra Elekter ja VKG

Elekter jäävad selle töö skoobist välja, sest testimise osakaal neis keskkondades on praegu väga väike.

3.1 Kasutatavad töövahendid

3.1.1 Automatiseerimise vahendi valik

Testide automatiseerimise vahendite valik tänapäeval on väga lai. UI automatiseerimise valdkonna standardiks on tänaseks kujunenud Selenium WebDriver, aga laialdaselt kasutatakse ka muid vahendeid nagu näiteks Katalon Studio, Test Studio, Ranorex, TestComplete, Puppeteer, TestCafe, Capybara, Cypress ja Watir jne.

Arvestades eelarve nappust, otsustati *open source* vahendite kasuks. Vahend peaks kindlasti olema hästi dokumenteeritud, et hoida kokku aega, mis kulub testijal selle vahendi õppimiseks. Kuna Syneralli arenduses kasutakse .NET arendusraamistikku, siis võiks olla ka testide kirjutamise keeleks kas C# või JavaScript, sest vastav kompetents on tiimis olemas. Hea vahend on ühtlasi see, mille installeerimine ja seadistamine on kiire ja lihtne. Vahendil võiks olla arvestatav kogukonna tugi juhuks kui tekib küsimusi, millele ei leia dokumentatsioonist vastust. Vahendil peaks olema ka tulevikuperspektiivi, et selle arendus ei katkeks lähiaastatel.

Nimetatud kriteeriumite alusel valiti kaks kandidaati, milleks on Selenium WebDriver ja Cypress. Hõlbustamaks valikut nende kahe vahel, on allikate [11] - [18] põhjal koostatud Cypressi ja Selenium WebDriveri võrdlustabel (Tabel 2), mille alusel seejärel analüüsitakse, kumb on sobivam vahend käesoleva töö teostamiseks.

Tabel 2. Vahendite Cypress ja Selenium WebDriver võrdlus.

Võrdlusobjekt	CYPRESS	WEBDRIVER
Testide kirjutamise keel	Ainult JavaScript ja TypeScript on toetatud [11].	Paljud keeled on toetatud, sh. Java, C#, Python, Javascript ja Ruby [11].

Võrdlusobjekt	CYPRESS	WEBDRIVER
Brauseri-ülene testimine	Chrome, Chromium ja Electron olid pikka aega ainsad toetatud brauserid. Alates Cypressi versioonist 4.0 lisati tugi ka brauseritele Edge ja Firefox. Firefoxi tugi on hetkel beetas. [12]	Toetatud on Chrome, Firefox, Opera, Safari ja Internet Explorer. WebDriver on end juba pikalt tõestanud selles valdkonnas [11].
Kiirus	Kuna Cypressi testid jooksevad brauseri enda sees, siis Cypress on väga kiire. Käskude sünkroonsus on teine põhjus, miks Cypressi testid jooksevad kiiresti [11].	Seleniumi WebDriveri testid võtavad rohkem aega kui Cypressi omad. Käsud on asünkroonsed [11].
Dokumen-tatsioon	Väga põhjalik ja detailne [11].	Hea, aga Cypressi oma on põhjalikum. [11].
Hind	*Cypress Runner* on open-source ja tasuta allalaetav. *Cypress Dashboard* teenusel on tasuta versioon ja 3 erinevat hinnastatud versiooni [13].	Tasuta allalaetav, open-source [14].
Vahendi töökorda seadmine	Väga kiiresti installeeritav näiteks npm paketihalduri kaudu, kasutades käsku *npm install cypress* [15].	Selenium Webdriveri installeeri-mine näiteks Visual Studio kaudu kasutades NuGet paketihaldurit ei ole samuti keeruline [16].
Automaatne elemendi ootamine	Cypress otsib automaatselt elementi niikaua, kuni selle leiab või kuni määratud *timeout* kätte jõuab [17].	Automaatset elemendi ootamist ei ole. Ootamised tuleb koodi kirjutada [18].

Selenium Webdriveri erinevate programmeerimiskeelte toetus on lai, Cypressi teste on võimalik kirjutada ainult kasutades JavaScripti ja TypeScripti. Arvestades seda, et tiimi

arendajad tunnevad kõige paremini C#, aga ka JavaScripti, sobivad seega sellest aspektist vaadatuna mõlemad vahendid. JavaScripti kasuks räägib näiteks see, et tegemist on ühega populaarseimatest programmeerimiskeeltest [19].

Brauseriülese testimise vaatevinklist on momendil küll WebDriver Cypressist üle: ühelt poolt sellega, et toetab peale Chrome'i ja Firefoxi ka Safarit, Internet Explorerit ja Operat, teiselt poolt sellega, et on end selles valdkonnas juba aastaid tõestanud. Cypress on algusest peale toetanud Chrome-perekonna brausereid ning alles versioonist 4.0 on lisatud brauserite Edge ja Firefox tugi. Arvestades seda, et käesoleva töö eesmärk on automatiseerida korduvalt sooritatavaid tegevusi testija töös, siis selle jaoks piisab täiesti ka ühe brauseri toetusest. Kuna Syneralli puhul esineb brauseri spetsiifikast tulenevaid vigasid pigem harva ja need pole reeglina kriitilised, puudub mõjuv vajadus automatiseeritud brauseriülese testimise järele ka tulevikku silmas pidades. Seetõttu võib Cypressi lugeda ka selles osas piisavaks.

Nii Selenium Webdriver kui ka Cypress on tasuta. Cypress pakub küll ka tasustatud Dashboardi teenust, aga käesoleva töö jaoks pole Dashboard-teenus vajalik. Dashboard võimaldab teste organiseerida ja käivitada paralleelselt üle erinevate masinate. Lisaks on sel on ka tasuta versioon olemas kuni 3 kasutajale, mis võimaldab salvestada kuni 500 testi. Ehk siis kui edaspidi võiks see kasulikuks osutuda, siis see tasuta versioon on tõenäoliselt piisav.

Mõlemad vahendid on suhteliselt lihtsad installeerida. Cypressi saab siiski kiiremini tööle. Paketihalduri npm kaudu võtab Cypressi installeerimine aega ainult mõned minutid ja kõik vajalik on installeeritav ühe käsuga. WebDriveri puhul on tarvis installeerida WebDriver ja siis veel eraldi spetsiifilise brauseri draiver, nt ChromeDriver.

Cypressi dokumentatsioon on põhjalikum kui WebDriveri dokumentatsioon. Kuigi Tayar kirjutas artikli [11] ajal, kui Seleniumi veebisait oli veel uuendamata, on Cypressi dokumentatsioon endiselt põhjalikum ja paremini navigeeritav. Kuna Selenium WebDriver on loodud kasutamiseks paljude eri programmeerimiskeeltega ja brauserite tugi on laiem, siis materjali, mida katta, on ka rohkem.

Cypressi üks suur eelis WebDriveriga võrreldes on automaatne elemendi ootamine. WebDriveri puhul on tarvis ootamised koodi kirjutada, aga Cypress teeb selle töö taustal

ise ära, oodates elementi kuni selle leiab või kuni *timeout* kätte jõuab. Seega Cypressi kood on selle võrra puhtam ja lihtsam kirjutada.

Kõike eelnevat arvesse võttes osutus Cypress antud töö teostamiseks sobilikumaks vahendiks. Cypressi ja WebDriverit võib vaadata vastavalt kui klassikalist „Vähem on rohkem“ ja Šveitsi nuga [11]. Cypress on kergekaaluline, aga piisav. WebDriver pakub rohkem võimalusi, mis aga ei osutu alati vajalikuks, nagu ka antud projektis.

3.1.2 Testandmete genereerimise vahendi valik

Selleks, et luua Synerallis põhiobjekte, on tarvis testandmeid:

- Erakliendi loomiseks: eesnimi, perenimi, isikukood (Soome ja Eesti),
- Ärikliendi loomiseks: ettevõtte nimi, äriregistrikood (Soome ja Eesti),
- Arvesti loomiseks: arvesti nimi.

Testandmete genereerimiseks saab kasutada selleks otstarbeks spetsiaalselt loodud tarkvara, milleks on andmegeneraatorid. Andmete genereerimise protsess mängib olulist rolli arvutiteaduse erinevates valdkondades, sealhulgas tarkvara testimises. Genereeritud andmete oluline aspekt on see, et andmed oleksid realistlikud, aga mitte tõelised, et oleks tagatud konfidentsiaalsus ja privaatsus [20].

Üks tuntumaid testandmete generaatoreid on veebirakendus Mockaroo, mis võimaldab luua era- ja ärikliendi nimesid, e-posti aadresse ja arvesti nimesid, aga mitte soome ja eesti isikukoode ega firma registrikoode. Andmeid saab salvestada erinevates formaatides, näiteks CSV (komaga eraldatud), JSON, SQL, XML, XLSX ja TXT (TABiga eraldatud). 1000 rea genereerimine on tasuta.

Soome ja eesti isikukoodide genereerimiseks on tarvis vaja midagi muud.

Veebirakendus *id-check.artega.biz* [21] võimaldab luua eesti ja soome isikukoode. Korraga luuakse 910 isikukoodi, mis kuvatakse veebilehel ühes tulbas, kust need on võimalik edasi salvestada faili. Kasutamine on tasuta.

Veebirakenduse *Fake Name Generator* [22] abil on võimalik aga genereerida peale eesti ja soome isikukoodide ka erakliendi nimed, ärikliendi nimi, e-posti aadress, telefon,

arvesti nimi. Võimalikud formaadid on näiteks CSV (komaga eraldatud), SQL, TXT (TABiga eraldatud) ja XLSX. Korraga on võimalik genereerida kuni 100000 rida tasuta. Fail saadetakse meili teel.

JavaScripti mikroteek *finnish-ssn*, mis valideerib ja genereerib Soome isikukoode. Kasutatud on MIT litsentsi. See on installeeritav npm paketihalduri kaudu ja leitav Github'ist [23].

Eelnimetatule analoogne Javascript mikroteek *isikukood* leidub ka Eesti isikukoodide genereerimiseks. Kasutatud on MIT litsentsi. See on installeeritav npm paketihalduri kaudu ja leitav Github'ist [24].

Soome äriregistrikoodide loomiseks saab kasutada generaatorit *Finnish Business Ids*, mis on leitav Github'ist [25]. See on samuti JavaScripti mikroteek soome äriregistrikoodide valideerimiseks ja loomiseks ja litsentseeritud MIT litsentsiga.

Lisaks on hea, kui kliendile saab lisada ka IBAN koodi – seda on tarvis HSY Synerallis hüvituste funktsionaalsuse testimisel. Soome IBANi genereerimiseks saab kasutada mikroteeki *finnish-bank-utils*, mis on litsentseeritud MIT-litsentsiga ja leitav Github'ist [26].

Nimetatud generaatoritest on kõige universaalsem veebirakendus *Fake Name Generator*. Ainus, mida see vajalikest andmetest ei genereeri, on äriregistrikoodid ja võimaldab korraga luua kõige rohkem ridasid.

Lõplik valik osutus siiski JavaScript teekide kasutamise kasuks: Soome isikukoodid genereeritakse *finnish-ssn* mikroteeki kasutades, Eesti isikukoodid genereeritakse mikroteeki *isikukood* kasutades, Soome äriregistrikoodid genereeritakse mikroteegi *finnish-business-ids* abil. Eraklientide nimed saab genereerida lihtsalt järgmisel viisil: Era12345 Isik45678, kus 12345 ja 45678 on juhuslikult genereeritud täisarvud mingis fikseeritud vahemikus. Ärikliendi nime ja arvesti saab luua analoogselt.. Sellega on kõik genereeritavad andmed kaetud peale eesti äriregistrikoodi: selle loomiseks ei leidunud mingit head vahendit. Kuna aga Eesti keskkonnad jäävad selle töö skoobist välja, sest testimise maht pole seal suur, siis võib sellele lahendust otsida hiljem, kui on plaanis luua regressiooniteste. Teekide kasutamise peamiseks plussiks on teiste nimetatud

generaatorite kasutamise ees see, et pole mingit vajadust tekitada ja hallata andmefaile, sest vastavad andmed genereeritakse vahetult programmi täitmisajal.

3.1.3 Koodiredaktori valik

Tiimi programmeerijad kasutavad Visual Studio IDEd, mis on võimas vahend ja mõeldud eelkõige .NET lahenduste loomiseks. Testide automatiseerimiseks JavaScriptis leidub sobivamaid vahendeid, mis on piisava funktsionaalsusega, aga samas kergema kaaluga. Stack Overflow 2019. aasta uuringu tulemuste [27] kohaselt on populaarseimaks arenduskeskkonnaks Visual Studio Code – 50,7% vastanutest pidasid seda populaarseimaks. Visual Studio Code on kergekaaluline, aga siiski võimas koodiredaktor, millel on olemas sisseehitatud JavaScripti tugi ja mis pakub JavaScripti puhul ka *IntelliSense* funktsionaalsust. See on arendatud Microsofti poolt, selle lähtekood on vaba ja see on tasuta allalaetav.

3.1.4 Versioonihaldusvahendi valik

Antud töö skoobis ei ole versioonihalduse koostöö juurutamine tiimis, küll aga eelduste loomine selleks. See tähendab seda, et tuleb valida sobivad vahendid ja lahenduse arendamise käigus versioonihaldust ise kasutada.

Versioonihaldussüsteem salvestab failidesse tehtud muudatused nii, et hiljem on võimalik tagasi pöörduda soovitud varasema projekti või faili versiooni juurde, võimaldab võrrelda faili erinevaid versioone ja näha, kes on teinud potentsiaalselt probleemi põhjustava muudatuse. Versioonihaldussüsteem võib olla kas lokaalne, tsentraliseeritud või hajus [28]. Detailsemalt on versioonihaldussüsteemide tüüpe kirjeldatud Lisas 3.

Valitud koodiredaktoris Visual Studio Code on integreeritud versioonihaldus ja see hõlmab vahetut Git'i tuge. Muud versioonihalduse pakkujad on ka toetatud, aga ainult laienduste kaudu. Git on hajus versioonihaldussüsteem, mis töötab kiirelt ja tõhusalt nii väikeste kui suurte projektide puhul. See on hästi dokumenteeritud: Pro Git raamat on kas veebis loetav, elektroonsel kujul allalaetav või ostetav paberraamatuna [29].

Git võimaldab inimestel töötada koos ühe projekti heaks. Kasutades Git'i koostööks, tuleb pidada kinni ühest kesksest reeglist: põhiharu (*master branch*) peab olema alati tarnitav. Seda võimaldab uute harude loomine ja nende liitmine põhiharuga [30]. Näiteks kui üks testija automatiseerib üht tegevust ja teine testija samal ajal automatiseerib mingit

teist tegevust, siis kumbki võib oma arvutis lokaalselt luua vastava haru ja kui valmis, laadida oma haru kaughoidlasse (*remote repository*) ja teha sellele koodiülevaatuse taotluse (*pull request*). Ülevaataja teeb ülevaatuse ära ja otsustab, kas haru võib liita või on eelnevalt tarvis teha parandusi. Kui muudatusi pole vaja teha, siis liidetakse haru põhiharuga.

Git-hoidla majutusteenuseid on mitmeid, aga üks tuntumaid neist on GitHub. See on veebipõhine ja hästi dokumenteeritud [31]. Üks GitHub'i peamisi konkurente on GitLab, mis pakub sarnaselt GitHub'ile tasuta versiooni [32]. GitHubi dokumentatsioon tundub GitLab'i omast oluliselt paremini struktureeritud ja intuitiivsem, vähemalt selle kasutamise esimeste sammude tegemisel. Kui varem, erinevalt GitLab'ist, olid GitHub'is privaatsed hoidlad tasulised, siis tänase päeva seisuga hõlmab GitHub'i tasuta versioon samuti piiramatul arvul privaatseid hoidlaid [33].

Versioonihaldustarkvara valik osutus seega Git'i ja Github'i kombinatsiooni kasuks.

3.2 Kasutatav metoodika

Töös järgitav meetod koosneb järgnevatest osadest:

1. Identifitseeritakse aeganõudvad korduvtegevused testija töös.
2. Otsustatakse, millised tegevusvood automatiseerida.
3. Tegevusvoogude automatiseerimine kasutades peatükis 3.1 valitud vahendeid.
4. Automatiseerimislahenduse kiiruse mõõtmine ja võrdlemine manuaalse sooritusega.
5. Automatiseerimislahenduse valideerimine peatükis 2.3 seatud eesmärkide suhtes.

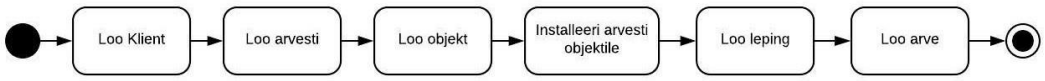
4 Automatiseerimine

Neljanda osa esimeses alamosas otsustatakse, millised tegevusvood automatiseerida. Teises alamosas kirjeldatakse, kuidas automatiseerimine teostatakse.

Automatiseerida on eeskätt mõtet tegevusi, millel on korduv iseloom, sest automatiseerimise implementeerimine ja juurutamine võtab ka aega ja kui automatiseeritud tegevust sooritatakse harva, siis ei pruugi selle automatiseerimisest olla palju tulu. Seega reeglina on kasulik automatiseerida lihtsalt automatiseeritavaid tihti kasutatavaid funktsionaalsusi. Mida ärikriitilisem funktsionaalsus, seda rohkem on põhjust seda automatiseerida. Ühtlasi mängib rolli aeg, mis selle tegevuse manuaalselt sooritamiseks kulub – näiteks kui mingit tegevust sooritatakse päevas umbes 7 korda, aga see võtab vaid paar hiireklikki, siis selle automatiseerimisel ei pruugi olla mõtet. Teiselt poolt tegevuse, mida sooritatakse vaid üks kord päevas, aga mis võtab manuaalsel sooritamisel palju aega, automatiseerimist võib kaaluda. Arutledes selle üle, mida tasub automatiseerida ja mida mitte, lähtub raamatu „The „A“ Word“ autor Alan Page ka heuristikust „mul on igav“ [[34], lk 5]. Sellest heuristikust lähtuvalt võib olla põhjust automatiseerida isegi tegevust, mille puhul ajaline võit pole suur. Näiteks kui testija kordab teatud tegevust juba mitmendat aastat järjest, siis isegi kui selle regulaarse tegevuse sagedus pole just suur ja automatiseerimise teel saavutatav ajaline võit on minimaalne, siis võib selle tegevuse automatiseerimine siiski olla õigustatud. Peale ajalise võidu aitab automatiseerimine vähendada vigasid, mida inimene teeb kas väsimusest või töö rutiinsest iseloomust tulenevalt.

4.1 Automatiseeritavad stsenaariumid

Enamus korduvaid tegevusi Syneralli testimises tehakse lepingute või arvelduse testimisel. Arveldamine on kahtlemata Synerallis üks ärikriitilisemaid funktsionaalsusi. Järgnev joonis (Joonis 4) illustreerib väga üldisel kujul seda tegevuste voogu kliendi loomisest kuni arveldamiseni välja.



Joonis 4. Arvelduse testimise tegevuste jada üldistatud kujul.

4.1.1 Elektriharu

Klient võib Synerallis olla kas eraklient, äriklient, grupiklient, FIE, linn või kontsern. Enamus testimisest sooritatakse era-, äri- ja grupiklientidega. Seega tasub ka automatiseerida just nimetatud 3 tüüpi klientide loomine. Erakliendi loomine eeldab isikukoodi sisestamist ja ärikliendi loomine eeldab registrikoodi sisestamist – siiamaani on testijad võtnud need koodid veebirakendusest, mis võtab omajagu aega. Kliendile tuleb lisada kontaktid, mida kasutatakse arvete, lepingu kinnituste, arve tasumise meeldetuletuste saatmise ja muu sellise tarvis – nendeks kontaktideks on postiaadress, e-post ja mobiiltelefon. Vaikimisi on loodava kliendi keeleks soome keel. Samas on osa soome kommunaalteenuste tarbijaist kliendid, kelle eelistatud keel on rootsi keel. Seepärast on kliendile saadetavad teated, kinnitused ja meeldetuletused samuti vastavalt kas soome- või rootsikeelsed. Aegajalt tuleb testides ette seda, et on vaja testida mingit rootsikeelset malli, aga kuna kliente rootsi keele eelistusega on Synerallis vähe, siis ei pruugi olemasolevaid testandmeid selleks mõnikord ollagi. Kliendi keelt saab küll muuta manuaalselt igal ajal soome keele pealt rootsi keele peale ja arve malli testimiseks sellest piisab, sest uue arve tegemine pole raske, aga osade taustatööde puhul aitaks palju, kui rootsikeelseid kliente ja neile loodud lepinguid ja arveid oleks Synerallis rohkem. Näiteks elektrilepingu pikenemise teate või tarbimise aastaaruande jaoks rootsikeelsete testandmete tekitamine on palju aeganõudvam. Seepärast on kasulik automatiseerida ka rootsikeelse kliendi loomine: see aitab kaudselt tagada seda, et Synerallis oleks mallide testimine hõlpsam.

Arvesti võib Synerallis olla kas koht- või kaugloetav. Koht- ja kaugloetava arvesti puhul on võrgulepingu mõõtelahendused erinevad ning sellest, kas kasutuskohal on koht-või kaugloetav arvesti, sõltub ka see, milliseid tooteid tarnelepingu koostamisel pakutakse – näiteks SPOT-tooteid on võimalik valida vaid siis, kui kasutuskoht on kaugloetav. Seega tasub automatiseerida mõlemad variandid.

Objekt võib olla Synerallis kas oma võrgu objekt oma või kuuluda välisvõrku. Kui on tegemist välisvõrgu objektiga, siis sellele arvestit ei installeerita ja ainus leping, mida sinna teha saab, on tarneleping. Seega on testimises raskuskese ikkagi oma võrgu objektil, mis tuleb kindlasti automatiseerida. Skoobist jäetakse välja soojuse ja gaasi objektid, mida on võimalik luua näiteks Kerava ja Herrforsi Synerallis, sest nende kommunaalteenuste testimise osakaal on üsna väike. Kui see peaks mingil hetkel muutuma, siis saab need hiljem lisada. Soodustamaks rootsi keele eelistusega klientidega testimist, automatiseeritakse objekti loomine ka era- ja -ärikliendiga, kelle keel on rootsi. Et aga vältida voogude arvu liiga suureks kasvamist, tehakse seda ainult kohtloetava arvestiga objekti puhul.

Elektri võrguleping luuakse oma võrgu kasutuskohale. Kaug- ja kohtloetav mõõtelahendus on erinevad, mistõttu automatiseeritakse mõlemad versioonid. Võrgulepingu puhul mängib rolli lepingu staatus: kuni leping on ainult salvestatud, saab seda veel muuta. Kui leping on juba kliendile väljastatud ehk kinnitatud, siis seda enam muuta ei saa. Sellest tulenevalt on kasulik automatiseerida nii salvestatud lepingu loomine kui ka kinnitatud lepingu loomine. Kuigi salvestatud mõõtelahendust on võimalik kustutada, on mõistlik automatiseerida ka võrgulepingu „toorik“: Tooriku all on mõeldud seda, et täidetakse ainult lepingu üldandmed: mõõtelahendust ja kontakte ei lisata üldse. See on kasulik, kui on teada, et mõõtelahendus tuleb teha manuaalselt või kui on teada, et installeeritav arvesti peab olema erinev sellest, mis on konfiguratsioonifailis määratud.

Elektri liitumislepinguid on 3 tüüpi: uus liitumine, liitumise muudatus ja omanikuvahetus.

Ka liitumislepingute puhul mängivad lepingu staatused rolli. „Salvestatud“ staatuses saab lepingus muudatusi teha. Pärast väljastamist muudatusi teha ei saa. Liitumislepingu puhul tuleks automatiseerida lisaks ka lepingu allkirjastamine ja aktiveerimine. Uue liitumislepingu tegemiseks kasutuskohale peab eelmine liitumisleping olema staatuses „Ühendatud“. Seega selleks, et automatiseerida liitumismuudatust ja omanikuvahetust, mis järgnevad uuele liitumisele, on vaja, et uus liitumine oleks allkirjastatud ja ühendatud. Arvesti tüüp ei oma suurt rolli liitumislepingute puhul, sest liitumislepingu arveldamine sellest ei sõltu – seepärast piisab, kui automatiseerida ainult näiteks kohtloetava arvestiga objektile liitumislepingu loomine.

Elektri tarnelepinguid loob testija võrdlemisi sageli. Sellele vaatamata otsustati need käesoleva töö raames mitte automatiseerida. Põhjuseks on see, et tarnelepingu tegemine on suhteliselt lihtne tegevus ja reeglina on siin oluline, millise tootega leping koostatakse: tarnelepingu toodete valik on aga väga suur. Just toote valimisel on kasulik, et seda saaks testija oma silmaga teha.

Ka arvete koostamist otsustati mitte automatiseerida, sest see on üldjuhul selline etapp, kus on samuti kas vaja täpsustada arveldamisperioodi, maksetähtaega, kontrollida arveridasid või midagi muud sellist. Teiseks – kuna Synerall koostab arve automaatselt – vaikimisi väärtustega arve tegemine võtab vaid paar klikki, siis võib seda vabalt ka manuaalselt teha.

Et lisada lahendusele paindlikkust, luuakse täispikkadele stsenaariumitele lisaks ka nö „osalised“ vood, mis eeldavad juba mingite elementide olemasolu. Näiteks kui täispikk voog loob kliendi, arvesti, objekti, paigaldab arvesti objektile ja loob võrgulepingu, siis „osaline“ voog eeldab objekti olemasolu, kuhu on loodud arvesti juba paigaldatud ja kliendi olemasolu, kes on objekti omanik. See võimaldab kasutada juba olemasolevaid elemente. Lisaks võimaldab see näiteks luua kõigepealt objekt, seda vastavalt vajadusele modifitseerida ja siis seejärel jätkata selle objektiga lepingute tegemist. Rootsi keele eelistusega klientidele saab sellisel viisil teha lepinguid luues esmalt objekti rootsikeelse kliendiga ja seejärel käivitades „osalise“ voo lepingu tegemiseks loodud objektiga.

Eelneva põhjal otsustati elektriharus automatiseerida järgmised tegevusvood (tegevusvood on grupeeritud kategooriatesse ja liitumislepingu puhul ka alamkategooriatesse):

1. Klient:
    a. Erakliendi loomine, kelle keel on soome
    b. Erakliendi loomine, kelle keel on rootsi
    c. Ärikliendi loomine, kelle keel on soome
    d. Ärikliendi loomine, kelle keel on rootsi
    e. Grupikliendi loomine, kelle keel on soome

2. Arvesti:
    a. Kaugloetava arvesti loomine
    b. Kohtloetava arvesti loomine
3. Objekt:
    a. Oma võrgu objekti loomine ilma arvestita, kus:
        i. objekti omanik on eraklient, kelle keel on soome
        ii. omanik on äriklient, kelle keel on soome
        iii. objekti omanik on grupiklient, kelle keel soome
        iv. kliendi olemasolu on eeldatud („osaline“ voog)
    b. Oma võrgu objekti loomine kohtloetava arvestiga, kus:
        i. objekti omanik on eraklient, kelle keel on soome
        ii. objekti omanik on eraklient, kelle keel on rootsi
        iii. objekti omanik on äriklient, kelle keel on soome
        iv. objekti omanik on äriklient, kelle keel on rootsi
        v. objekti omanik on grupiklient, kelle keel on soome
    c. Oma võrgu objekti loomine kaugloetava arvestiga, kus:
        i. objekti omanik on eraklient, kelle keel on soome
        ii. objekti omanik on äriklient, kelle keel on soome
        iii. objekti omanik on grupiklient, kelle keel on soome
    d. Oma võrgu objekti loomine, kus arvesti ja kliendi olemasolu on eeldatud („osaline“ voog)
    e. Võõra võrgu objekti loomine

4. Võrguleping:

    a. Arvestita võrgulepingu „tooriku“ loomine, kus lepingu omanik on eraklient, kelle keel on soome

    b. Arvestita võrgulepingu „tooriku“ loomine, kus lepingu omanik on äriklient, kelle keel on soome

    c. Kohtloetava arvestiga „Salvestatud“ staatuses võrgulepingu loomine, kus:

        i. lepingu omanik on eraklient, kelle keel on soome

        ii. lepingu omanik on äriklient, kelle keel on soome

        iii. lepingu omanik on grupiklient, kelle keel on soome

        iv. kohtloetava arvestiga ja omanikuga objekti olemasolu on eeldatud („osaline“ voog)

    d. Kohtloetava arvestiga „Väljastatud“ staatuses võrgulepingu loomine, kus:

        i. lepingu omanik on eraklient, kelle keel on soome

        ii. lepingu omanik on äriklient, kelle keel on soome

    e. Kaugloetava arvestiga „Salvestatud“ staatuses võrgulepingu loomine, kus:

        i. lepingu omanik on eraklient, kelle keel on soome

        ii. lepingu omanik on äriklient, kelle keel on soome

        iii. lepingu omanik on grupiklient, kelle keel on soome

        iv. kaugloetava arvestiga ja omanikuga objekti olemasolu on eeldatud („osaline“ voog)

    f. Kaugloetava arvestiga „Väljastatud“ staatuses võrgulepingu loomine, kus:

        i. lepingu omanik on eraklient, kelle keel on soome

        ii. lepingu omanik on äriklient, kelle keel on soome

5. Liitumisleping:
    a. Liitumislepingu tüüp „Uus liitumine“:
        i. Liitumislepingu loomine arvestita objektile, kus lepingu omanik on eraklient ja lepingu staatus on „Salvestatud“
        ii. Liitumislepingu loomine arvestita objektile, kus lepingu omanik on äriklient ja lepingu staatus on „Salvestatud“
        iii. Liitumislepingu loomine kohtloetava arvestiga objektile, kus lepingu staatus on „Salvestatud“ ja:
            1. Lepingu omanik on eraklient, kelle keel on soome
            2. Lepingu omanik on äriklient, kelle keel on soome
            3. Lepingu omanik on grupiklient, kelle keel on soome
        iv. Liitumislepingu loomine kohtloetava arvestiga objektile, kus lepingu staatus on „Ühendatud“ ja:
            1. Lepingu omanik on eraklient, kelle keel on soome
            2. Lepingu omanik on äriklient, kelle keel on soome
            3. Lepingu omanik on grupiklient, kelle keel on soome
        v. Uue liitumise lepingu loomine, kus objekti – arvestiga või arvestita – olemasolu on eeldatud („osaline“ voog)
    b. Liitumislepingu tüüp „Liitumise muudatus“:
        i. Liitumislepingu loomine kohtloetava arvestiga objektile, kus lepingu staatus on „Salvestatud“ ja:
            1. Lepingu omanik on eraklient, kelle keel on soome
            2. Lepingu omanik on äriklient, kelle keel on soome
            3. Lepingu omanik on grupiklient, kelle keel on soome

ii. Liitumislepingu loomine kohtloetava arvestiga objektile, kus lepingu staatus on „Ühendatud“ ja:

1. lepingu omanik on eraklient, kelle keel on soome
2. lepingu omanik on äriklient, kelle keel on soome
3. lepingu omanik on grupiklient, kelle keel on soome

iii. Liitumismuudatuse lepingu loomine, kus on eeldatud juba olemasolevat omanikuga objekti, millel on liitumisleping staatusega „Ühendatud“ („osaline“ voog)

c. Liitumislepingu tüüp „Omanikuvahetus“:

i. Liitumislepingu loomine kohtloetava arvestiga objektile, kus lepingu staatus on „Salvestatud“ ja:

1. lepingu omanik on eraklient, kelle keel on soome
2. lepingu omanik on äriklient, kelle keel on soome
3. lepingu omanik on grupiklient, kelle keel on soome

ii. Liitumislepingu loomine kohtloetava arvestiga objektile, kus lepingu staatus on „Ühendatud“ ja:

1. lepingu omanik on eraklient, kelle keel on soome
2. lepingu omanik on äriklient, kelle keel on soome
3. lepingu omanik on grupiklient, kelle keel on soome

iii. Omanikuvahetuse lepingu loomine, kus on eeldatud juba olemasolevat omanikuga objekti, millel on liitumisleping staatusega „Ühendatud“ („osaline“ voog)

4.1.2 Veeharu

Kliendi loomine on analoogne elektriharule, millest tulenevalt on mõistlik automatiseerida era-, ära- ja grupikliendi loomine. Elektriharuga samal põhjusel

automatiseeritakse ka rootsi keele eelistusega kliendi loomine. Kontaktidena lisatakse postiaadress, e-post ja mobiiltelefon. Kuna HSYs hüvituste funktsionaalsus eeldab ka IBANi olemasolu, kui kliendil on valitud ’Hüvitamine pangakontole’, siis on mõistlik ka lisada kliendile ka IBAN.

Arvesti loomine on analoogne elektriharule selle erinevusega, et luuakse lihtsalt kohtloetav arvesti, sest hetkeseisuga kaugloetavaid arvesteid veeharu Synerallis ei kasutata.

Objekti loomine erineb elektriharust selle poolest, et siin oma võrgu ja võõra võrgu erisust pole. Võimalik on luua vee objekte, rendiobjekte ja sprinklerobjekte. Kuna valdav osa testimisest hõlmab veeobjekte, siis käesoleva töö raames piirdutakse veeobjektide loomisega. Soodustamaks rootsi keele eelistusega klientidega testimist, automatiseeritakse objekti loomine ka era- ja -ärikliendiga, kelle keel on rootsi. Et aga vältida voogude arvu liiga suureks kasvamist, tehakse seda ainult arvestiga objekti puhul.

Tarbimisleping on analoogne elektriharu võrgulepingule, mis hõlmab mõõtelahendust. Tarbimislepingut on võimalik luua objekti vaate vasakmenüü kaudu. Teine võimalus on luua kõigepealt liitumisleping, mille väljastamisel Synerall loob objektile tarbimislepingu tooriku.

Liitumislepingud on tüüpide poolest analoogsed elektriharu liitumislepingutele. Erinev on see, et kui elektriharus peab olema järgmise liitumislepingu loomiseks eelnev liitumisleping allkirjastatud ja pingestatud, siis veeharus peab olema eelmine liitumisleping allkirjastatud ning eelmise liitumislepingu arve peab olema tasutud. Nimelt tehakse liitumisarve uue liitumise ja liitumismuudatuse puhul. Seega selleks, et automatiseerida liitumismuudatust ja omanikuvahetust, mis järgnevad vahetult uuele liitumisele, on vaja, et algne liitumisleping oleks allkirjastatud ning selle arve makstud.

Eelneva põhjal otsustati veeharus automatiseerida järgmised tegevusvood (tegevusvood on grupeeritud kategooriatesse ja liitumislepingu puhul ka alamkategooriatesse):

1. Klient:
    a. Erakliendi loomine, kliendi keel: soome
    b. Erakliendi loomine, kliendi keel: rootsi

c. Ärikliendi loomine, kliendi keel: soome

d. Ärikliendi loomine, kliendi keel: rootsi

e. Grupikliendi loomine

2. Arvesti:

    a. Arvesti loomine

3. Objekt:

    a. Arvestiga objekti loomine, kus:

        i. objekti omanik on eraklient, kelle keel on soome

        ii. objekti omanik on eraklient, kelle keel on rootsi

        iii. objekti omanik on äriklient, kelle keel on soome

        iv. objekti omanik on äriklient, kelle keel on rootsi

        v. objekti omanik on partnerklient, kelle keel on soome

        vi. kliendi ja arvesti olemasolu on eeldatud

    b. Arvestita objekti loomine, kus:

        i. objekti omanik on eraklient, kelle keel on soome

        ii. objekti omanik on äriklient, kelle keel on soome

        iii. objekti omanik on partnerklient, kelle keel on soome

        iv. kliendi olemasolu on eeldatud („osaline“ voog)

4. Tarbimisleping:

    a. „Salvestatud“ staatusega tarbimislepingu loomine arvestiga objektile, kus:

        i. objekti omanik on eraklient, kelle keel on soome

ii. objekti omanik on äriklient, kelle keel on soome

iii. objekti omanik on partnerklient, kelle keel on soome

b. „Kinnitatud“ staatusega tarbimislepingu loomine arvestiga objektile, kus:

i. objekti omanik on eraklient, kelle keel on soome

ii. objekti omanik on äriklient, kelle keel on soome

iii. objekti omanik on partnerklient, kelle keel on soome

c. Tarbimislepingu loomine juba eelnevalt olemasolevale omaniku ja arvestiga objektile („osaline“ voog)

5. Liitumisleping:

a. Liitumislepingu tüüp: „Uus liitumine“:

i. Liitumislepingu loomine arvestita objektile, kus lepingu staatus on „Salvestatud“ ja:

1. Lepingu omanik on eraklient, kelle keel on soome

2. Lepingu omanik on äriklient, kelle keel on soome

3. Lepingu omanik on grupiklient, kelle keel on soome

ii. Liitumislepingu loomine arvestita objektile, kus lepingu staatus on „Kinnitatud“ ja

1. Lepingu omanik on eraklient, kelle keel on soome

2. Lepingu omanik on äriklient, kelle keel on soome

3. Lepingu omanik on grupiklient, kelle keel on soome

iii. Liitumislepingu loomine arvestiga objektile, kus lepingu staatus on „Salvestatud“ ja:

1. Lepingu omanik on eraklient, kelle keel on soome

2. Lepingu omanik on äriklient, kelle keel on soome

3. Lepingu omanik on grupiklient, kelle keel on soome

iv. Liitumislepingu loomine arvestiga objektile, kus lepingu staatus on „Kinnitatud“ ja

1. Lepingu omanik on eraklient, kelle keel on soome

2. Lepingu omanik on äriklient, kelle keel on soome

3. Lepingu omanik on grupiklient, kelle keel on soome

v. Liitumislepingu loomine staatusega „Kinnitatud“ arvestiga objektile koos „Salvestatud“ staatusega tarbimislepinguga, kus:

1. Lepingu omanik on eraklient, kelle keel on soome

2. Lepingu omanik on äriklient, kelle keel on soome

3. Lepingu omanik on grupiklient, kelle keel on soome

vi. Liitumislepingu loomine staatusega „Kinnitatud“ arvestiga objektile koos „Kinnitatud“ staatusega tarbimislepinguga, kus:

1. Lepingu omanik on eraklient, kelle keel on soome

2. Lepingu omanik on äriklient, kelle keel on soome

3. Lepingu omanik on grupiklient, kelle keel on soome

vii. Liitumislepingu loomine eelnevalt olemasolevale omanikuga objektile („osaline“ voog)

b. Liitumislepingu tüüp: „Liitumise muudatus“ (eeldatud on allkirjastatud ja tasutud arvega liitumislepinguga objekti olemasolu):

i. Liitumislepingu loomine staatusega „Salvestatud“, kus lisatakse liitumise tüüp „sademevesi“ („osaline“ voog)

ii. Liitumislepingu loomine staatusega „Kinnitatud“, kus lisatakse liitumise tüüp „sademevesi“ („osaline“ voog)

iii. Liitumislepingu loomine staatusega „Salvestatud“, kus muudetakse objekti tüüpi („osaline“ voog)

iv. Liitumislepingu loomine staatusega „Kinnitatud“, kus muudetakse objekti tüüpi („osaline“ voog)

v. Liitumislepingu loomine staatusega „Salvestatud“, kus muudetakse objekti pindala („osaline“ voog)

vi. Liitumislepingu loomine staatusega „Kinnitatud“, kus muudetakse objekti pindala („osaline“ voog)

c. Liitumislepingu tüüp: „Omanikuvahetus“ (eeldatud on allkirjastatud ja tasutud arvega liitumislepinguga objekti olemasolu):

i. Liitumislepingu loomine staatusega „Salvestatud“, kus:

1. Uus omanik on eraklient, kelle keel on soome („osaline“ voog)

2. Uus omanik on äriklient, kelle keel on soome („osaline“ voog)

3. Uus omanik on grupiklient, kelle keel on soome („osaline“ voog)

ii. Liitumislepingu loomine staatusega „Kinnitatud“, kus:

1. Uus omanik on eraklient, kelle keel on soome („osaline“ voog)

2. Uus omanik on äriklient, kelle keel on soome („osaline“ voog)

3. Uus omanik on grupiklient, kelle keel on soome („osaline“ voog)

iii. Omanikuvahetuse lepingu loomine, kus lisaks eelneva allkirjastatud ja tasutud arvega liitumislepinguga objektile on eeldatud kliendi olemasolu, keda määrata uueks omanikuks („osaline“ voog)

d. Liitumislepingu allkirjastamine („osaline“ voog, kasutatakse omanikuvahetuse puhul)

e. Liitumislepingu allkirjastamine, arveldamine ja arve eest tasumine („osaline“ voog, kasutatakse liitumismuudatuse puhul)

4.2 Teostus

4.2.1 Cypressi installeerimine ja käivitamine

Cypressi töövalmis seadmine npm paketihalduri kaudu käib kiiresti. Kui npm paketihaldurit veel pole, siis tuleb laadida alla ja installeerida Node.js: sellega koos installeeritakse ka npm [35].

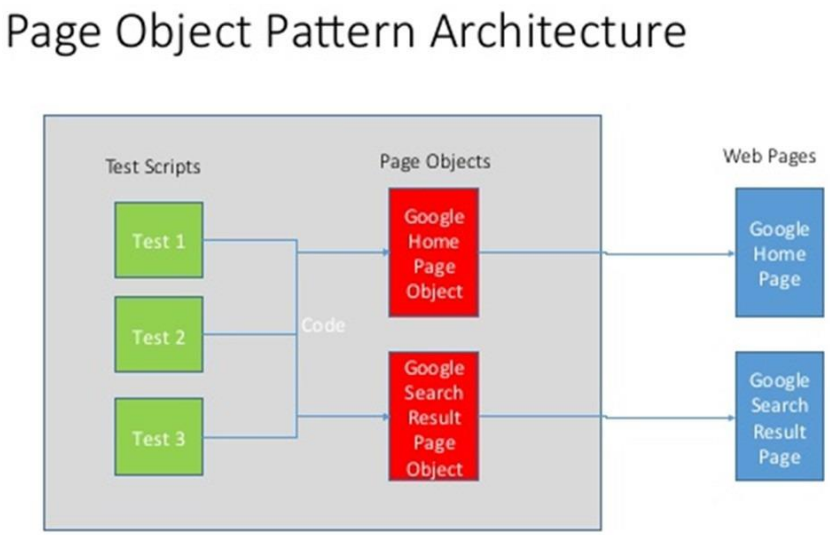
Järgnevalt tuleb luua projekti kataloog ja seejärel projekt initsialiseerida. Selleks tuleb käsureal minna vastava projekti kataloogi ja kasutada käsku `npm init`, mis tekitab package.json faili. Cypressi installeerimiseks tuleb kasutada käsku `npm install Cypress --save-dev`. Kui Cypress on installeeritud, võib Cypressi avada käsuga `npx cypress open`. Cypressi avamiseks on rohkem kui üks käsk, aga käsu npx kasutamine on kõige lühem viis seda teha ja see võimalus on olemas, kui kasutada paketihalduri npm versiooni 5.2 või hilisemat [15]. Ekraanitõmmist Cypressi graafilisest kasutajaliidesest on võimalik näha Lisas 4.

4.2.2 *Page Object* mudel

Automatiseerimislahendus on teostatud kasutades Page Object mudelit (POM).

POM on disainimuster, mis on testide automatiseerimise valdkonnas laialt kasutatav, kuna aitab teste hallata ja vähendab koodi dubleerimist. Testid kasutavad *page object* (PO) klassi meetodeid, kui selle veebilehe kasutajaliidesega suhtlevad. Kui veebilehe kasutajaliides muutub, siis seda mudelit kasutades teste muutma ei pea – koodimuudatus tuleb teha ainult vastava veebilehe PO klassis. POM aitab hoida koodi puhta, hallatava ja

arusaadavana. Testlood on lühikesed ja optimeeritud, sest testfailis kasutatakse taaskasutatavaid POM klassi meetodeid [36]. Reeglina vastab veebilehele üks PO. Testskripti (Cypressi puhul spec.js failid) imporditakse kõik POd, mida vastav skript kasutab. POMi illustreerib Joonis 5.



Joonis 5. *Page Object disainimustri arhitektuur* [37].

Lisaks konkreetsele veebilehele vastavatele POdele on käesolevas töös kasutatud veel üht kihti, milleks on *Base Page Object* ja mida kasutatakse selleks, et vältida dubleerimist PO failides. PO klassid laiendavad *BasePage* klassi, mis käesoleva töö puhul sisaldab näiteks sisselogimise funktsiooni.

4.2.3 Syneralli erinevad keskkonnad ja konfiguratsioonid

Konfiguratsiooniväärtusi hoitakse cypress.json failis. Siin on võimalik näiteks üle kirjutada Cypressi vaikimisi konfiguratsiooniväärtused ning lisada baas-URL ja keskkonnamuutujaid [38].

Kuna Synerallis on aga palju erinevaid kliendikeskkondi, kus mitmed väärtused, mida antud lahendus kasutab, on erinevad: näiteks erinevad tooted ja arvestimudelid, siis on vaja, et igal keskkonnal oleks oma konfiguratsioonifail, kuhu sellised väärtused saab salvestada keskkonnamuutujatena.

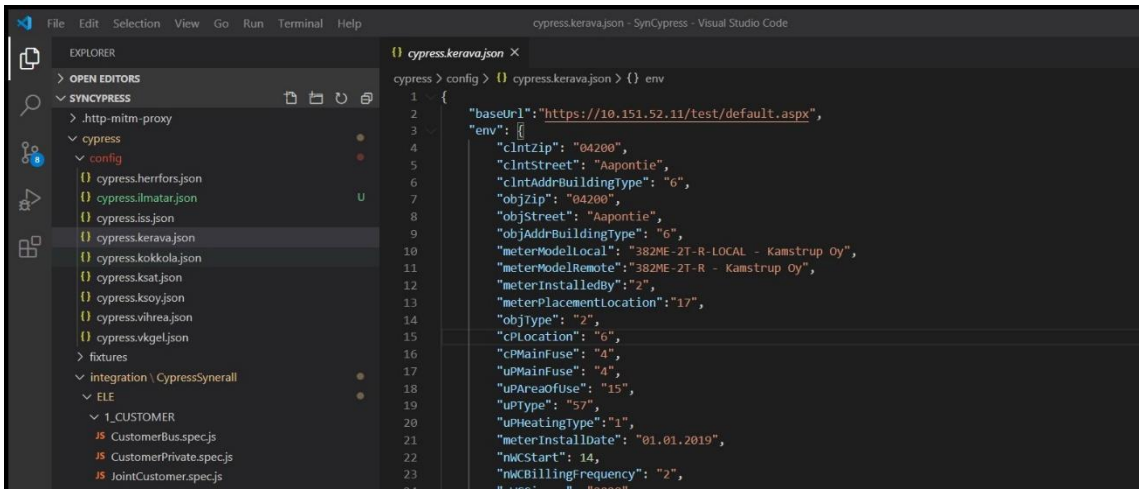
Cypress võimaldab salvestada mitmeid konfiguratsioonifaile ja nende vahel ümberlülituda [39]. Selle tagab järgneval joonisel (Joonis 6) esitatud kood plugins/index.js failis.

```
// promisified fs module
const path = require('path');
const fs = require('fs-extra');

function getConfigurationByFile(file) {
  const pathToConfigFile = path.resolve(
    'cypress/config',
    `cypress.${file}.json`
  );

  return fs.readJson(pathToConfigFile);
}

module.exports = (on, config) => {
  // accept a configFile value or use local by default
  const file = config.env.configFile || 'local';
  return getConfigurationByFile(file);
};
```

Joonis 6. Kood konfiguratsioonide vahel ümberlülitumiseks [40].

See kood kasutab *fs-extra* moodulit, mis on litsentseeritud MIT litsentsiga. Moodul ise on installeeritav npm paketihalduri kaudu.

Nüüd on võimalik luua iga keskkonna jaoks eraldi konfiguratsioonifail cypress/config kataloogi alla: seda illustreerib Joonis 7.



Joonis 7. Konfiguratsioonifailid.

Selleks, et avada Cypress runner nii, et see kasutaks konfiguratsioonifaili cypress.kerava.json, tuleb käsku täpsustada: `npx cypress open --env configFile=kerava`.

Selle lahenduse puhul tuleb aktsepteerida fakti, et iga kord, kui konfiguratsioonifaili muuta, tuleb Cypress uuesti käivitada soovitud konfiguratsiooniga. Kui peaks osutuma vajalikuks korduvalt käivitada mingit stsenaariumit erinevate keskkonnamuutujate väärtustega, siis konfiguratsioonifaili muutmise alternatiiviks on koodi muutmine vastavas PO failis, kommenteerides keskkonnamuutuja välja ja kirjutades väärtuse otse koodi. Niiviisi ei pea Cypressi taaskäivitama.

Niikaua kuni HSY on ainus keskkond, kuhu sisselogimine toimub AD autentimist kasutades, kasutatakse HSY konfiguratsioonifailina Cypressi vaikimisi konfiguratsioonifaili cypress.json – see annab selle eelise, et pole vaja konfiguratsioonimuudatuse jõustumiseks Cypressi taaskäivitada.

4.2.4 Sisselogimine

Sisselogimine on iga stsenaariumi eelduseks. Reeglina logitakse Syneralli testkeskkonda sisse kasutajanime ja parooliga. Erandiks on HSY Synerall, mille puhul kasutatakse AD autentimist.

Selleks, et Cypress oskaks AD autentimisega toime tulla, tuleb kasutada eraldi pluginat *cypress-ntlm-auth*, mis on litsentseeritud MIT litsentsiga, seega kasutatav ka kommertseesmärkidel. Plugin on npm paketihalduri kaudu installeeritav käsuga `npm install --save-dev cypress-ntlm-auth.` Plugina konfigureerimiseks tuleb modifitseerida 3 faili: cypress/plugins/index.js, cypress/support/index.js ja package.json vastavalt detailsetele juhistele, mis on antud npm paketihalduri veebisaidi *cypress-ntlm-auth* plugina veebilehel [41].

HSY Syneralli sisselogimise funktsioon kasutades *cypress-ntlm-auth* plugina käske on väga kompaktne. Seda illustreerib Joonis 8.

```
login() {
    cy.ntlmReset()
    cy.ntlm(
        "http://tlldev5.netgroupdigital.com",
        Cypress.env('username'),
        Cypress.env('pw')
    )
    cy.visit('http://tlldev5.netgroupdigital.com/HSY/default.aspx')
}
```

Joonis 8. Funktsioon HSY keskkonda sisselogimiseks.

Kui Package.json faili on tehtud muudatus eelnimetatud juhise kohaselt, siis Cypressi käivitamine NTLM-autentimisega on tehtav käsuga `npm run cypress-ntlm`. See käivitab nii ntlm-proxy eraldi protsessina kui ka Cypressi. Kui Cypress lõpetab, lõpetatakse ka ntlm-proxy protsess.

4.2.5 Kasutatavad teegid

Töös on kasutatud erinevaid npm paketihalduri kaudu installeeritavaid teeke: Soome id-koodi generaatori mikroteek *finnish-ssn* [23], Soome äriregistrikoodi mikroteek *finnish-business-ids* [25], Soome IBANi genereerimise mikroteek *finnish-bank-utils* [26] ja kuupäevade funktsionaalsust pakkuv *date-fns* teek [42]. Kõik eelnimetatud teegid on litsentseeritud MIT litsentsiga ja kasutatavad ärilistel eesmärkidel.

4.2.6 Juhuslike arvude genereerimine

Kasutatud on developer.mozilla.org veebisaidil kuvatavat funktsiooni juhuslike täisarvude genereerimiseks soovitud vahemikus, kus miinimumväärtus on kaasaarvatud ja maksimumväärtus väljaarvatud (Joonis 9).

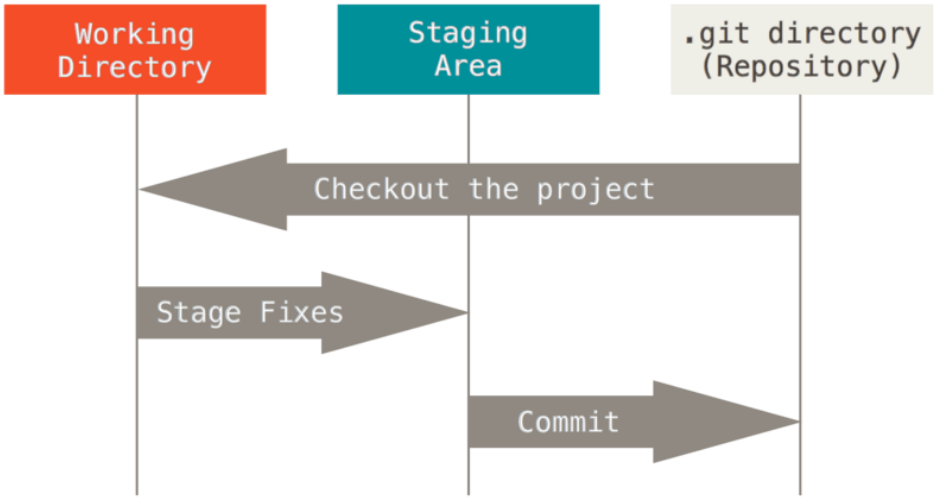
```
function getRandomInt(min, max) {
      min = Math.ceil(min);
      max = Math.floor(max);
      return Math.floor(Math.random() * (max - min)) + min;
}
```

Joonis 9. Funktsioon juhuslike täisarvude genereerimiseks [43].

4.2.7 Versioonihaldus

Versioonihalduse ülesseadmiseks tuleb alla laadida, paigaldada ja konfigureerida Git. Seejärel liikuda käsureal projekti kataloogi ja initsialiseerida see projekt käsuga `git init`, mille tagajärjel luuakse uus alamkataloog .git/. Selleks, et Git hakkaks faile jälgima, tuleb failid lisada töökataloogist (*working directory*) ettevalmistusalasse (*staging*

area) käsuga `git add`. Ettevalmistusalast saadetakse failid kohalikku hoidlasse (*repository*) käsuga `git commit`. Seda illustreerib Joonis 10.



Joonis 10. Töökataloog, ettevalmistusala ja Git'i kohalik hoidla [44].

Kui Git vastutab selle eest, mis toimub projekti failidega kohalikus masinas, siis Git-hoidla majutusteenus, milleks antud töös valiti GitHub, kannab hoolt projekti kaughoidla eest. Selleks, et saaks saata projektifailid GitHub'i, tuleb teha GitHub'is endale kasutajakonto, seejärel luua seal uus hoidla, hoidla vaatel nupu „Klooni või lae alla" kaudu kopeerida hoidla URL. Seejärel tuleb käsureal lisada Git'i see kaughoidla käsuga, mida illustreerib Joonis 11.

```
piret.samuel@NGA-PiretS2017 MINGW64 ~/dev/SynCypress (master)
$ git remote add origin https://github.com/pirsam/SynerallCypress.git
```

Joonis 11. Kaughoidla lisamine Git'i.

Nüüd on võimalik saata kohalikku hoidlasse tehtud muudatused kaughoidlasse käsuga `git push origin master [45].`

5 Tehtud töö tulemused

Käesoleva diplomitöö tulemused esitatakse 2 osas: esiteks esitatakse mõõtmistulemused ja teiseks valideeritakse töö tulemused peatükis 2.3 seatud eesmärkide suhtes.

5.1 Mõõtmistulemused

Cypress kuvab aega, mis kulub käivitatud programmi täitmiseks. Mõõtmistel kasutatigi Cypressi enda kella. Ainsaks erandiks oli mõõdiku paigaldamisele kuluva aja mõõtmine, sest paigaldamiseks pole eraldi spec.js faili. Paigaldamine on osa spec.js failist, milles luuakse arvestiga objekt. Seetõttu paigaldamisele kuluva aja mõõtmiseks kasutati Cypressi käsku `cy.pause()` ja stopperit. Tegevuse automatiseeritud sooritamise peale kuluvat aega mõõdeti 3 korda, millest võeti aritmeetiline keskmine, mis on esitatud alljärgnevates tabelites.

Mõõtmised viidi läbi igas Syneralli keskkonnas eraldi. Elektriharu tulemused kuvatakse järgnevas tabelis (Tabel 3), kus automatiseeritud tegevuste peale kuluv aeg on esitatud minutites. Viimases tulbas kuvatakse 6 elektriharu keskkonna mõõtmistulemuste aritmeetiline keskmine, mida kasutatakse võrdlustabeli (Tabel 4) elektriharu osas. Mõõtmisi ei viidud läbi Ilmatari ja Vihreä Energia Synerallis, sest kuna need on ainult elektritarnijad, siis nende keskkondades on mõõdetavatest tegevustest võimalik teha vaid kliendi loomist. Võõra võrgu objekti loomine on küll automatiseeritud, kuid kuna selle loomine on pigem harv ja lihtne tegevus, siis ei pidanud vajalikuks seda eraldi mõõta.

Tabel 3. Mõõtmise tulemused elektriharu klientide Syneralli keskkondades.

TEGEVUS	KER	KSAT	KOK	ISS	HER	KSOY	**Avg**
Kliendi loomine	0,39	0.44	0.45	0,4	0.41	0.39	**0.41**
Mõõdiku loomine	0,07	0.08	0.1	0.08	0.08	0.08	**0.08**
Objekti loomine (oma võrk)	0,29	0.32	0.32	0.28	0.29	0.32	**0.30**
Mõõdiku paigaldamine	0,07	0.08	0.1	0.09	0.07	0.1	**0.09**
Võrgulepingu loomine	0,22	0,24	0.24	0,2	0.22	0.23	**0.23**
Liitumislepingu loomine	0,2	0.21	0.21	0.19	0.18	0.19	**0.20**
Liitumise muudatus	0,26	0,26	0.31	0,25	0.26	0.28	**0.27**
Omanikuvahetus	0,15	0.16	0.18	0.15	0.17	0.16	**0.16**

Järgnevalt on esitatud võrdlustabel (Tabel 4), milles kõrvutatakse korduvtegevuste peale kuluv aeg manuaalselt ja automatiseeritult sooritatuna. Aeg kuvatakse minutites. Tabeli viimasel real kuvatakse korduvsooritatavate tegevuste peale keskmiselt kuluv aeg päevas manuaalselt ja automatiseeritult. Sagedused pärinevad diplomitöö esimesest tabelist (Tabel 1). Kõigi mõõtmiste ja arvutuste tulemused ümardati 2 kohani pärast koma.

Tabel 4. Tegevuste peale kuluv aeg (minutites) manuaalselt (M) ja automatiseeritult (A).

TEGEVUS	Aeg		Sagedus	Aeg päevas	
	M	A	päevas	M	A
ELEKTRIHARU					
Kliendi loomine	1,77	0,41	0,88	1,56	0,36
Arvesti loomine	0,42	0,08	0,88	0,37	0,07
Objekti loomine (oma võrk)	1,28	0,3	0,88	1,13	0,26
Arvesti paigaldamine	0,53	0,09	0,88	0,47	0,08
Võrgulepingu loomine	1,6	0,23	0,72	1,15	0,17
Liitumislepingu loomine – uus liitumine	1,92	0,2	0,08	1,15	0,02
Liitumismuudatus	2,07	0,27	0,08	0,17	0,02
Omanikuvahetus	1,43	0,16	0	0	0
VEEHARU (HSY)					
Kliendi loomine	1,75	0,3	3	5,4	0,9
Arvesti loomine	0,35	0,05	2,55	0,89	0,13
Objekti loomine	1,08	0,29	2,55	2,75	0,74
Arvesti paigaldamine	0,52	0,09	2,55	1,33	0,23
Vee tarbimislepingu loomine	1,47	0,22	3,45	5,07	0,8
Liitumislepingu loomine – uus liitumine	1,83	0,38	2,55	4,67	0,97
Liitumismuudatus	1,97	0,46	0,85	1,68	0,39
Omanikuvahetus	1,43	0,28	0,5	0,72	0,14
Korduvate tegevuste peale keskmiselt kuluv aeg päevas:				**28,36**	**5,28**

Järeldused:

- Ajaline võit: 28,36 min – 5,28 min = 23, 08 minutit päevas.

- Automaatselt kulus päeva lõikes korduvtegevuste sooritamiseks 28,36:5,28=5,37 korda vähem aega kui neid manuaalselt sooritades.

5.2 Töö valideerimine seatud eesmärkide suhtes

Selles peatükis võetakse ette kõik eesmärgid, mis seati peatükis 2.3 ja valideeritakse lahendust nende suhtes.

Eesmärk 1: suurendada hetkel manuaalse testimisprotsessi efektiivsust korduvsooritatavate tegevuste automatiseerimisega ehk minna üle osaliselt automatiseeritud testimisele. Automatiseeritult võiks kuluda selliste tegevuste peale umbes 5 korda vähem aega.

Peatükis 5.1 esitatud mõõtmistulemuste kohaselt kulub automatiseeritult korduvtegevustele päevas 5, 37 korda vähem aega kui seda manuaalselt tehes.

Ajalist võitu ei saa kindlasti aga võtta sõnasõnaliselt, et nüüd on testijal iga päev just 23 minutit rohkem aega. Iga päev on erinev, iga sprint on erinev ja testitakse seda, mida parasjagu arendatakse. On perioode, mille vältel lepingute ja arveldamise testimist on vähem ja perioode, millal seda on rohkem. Cypressi käivitamine võtab aega. Kõige mõistlikum on kohe hommikul Visual Studio Code koodiredaktoris projekt avada ja kui tuleb testimist, mille puhul saab kasutada antud lahendust, saab koodiredaktori terminali kaudu Cypressi avada ligikaudu 20 sekundiga. Iga kord, kui on tarvis muuta konfiguratsiooni, tuleb Cypress taaskäivitada. Lisaks peab arvestama sellega, et lahenduse koodi on tarvis aegajalt ka hallata, mis omakorda võtab aega. Kuna käesolevas töös oli veebielementide lokaatoriks valdavalt elemendi id, siis veebilehe elementide paigutuse muudatused ei mõjuta lahenduse koodi toimimist. Siiski võib ette tulla seda, et konfiguratsioonifailis määratud väärtus, näiteks arvesti mudel, on andmete uuendamise käigus kustutatud – see olukord tõepoolest eeldab hetkel manuaalset sekkumist.

Kuigi nüüd tuleb ette loodud lahenduse haldamisega seotud lisategevusi, mis võtavad teatava osa testimise ajast, siis teisest küljest jällegi võib lugeda vabanenud ajaks ka selle aja, mil Cypress uut objekti loob – testija saab seda aega kasutada produktiivselt. Need 5,28 minutit päevas, mis Cypressil kulus korduvtegevuste loomiseks, on orienteeruv aeg, mis võib päevas keskmiselt kuluda tegevuste peale nagu lahenduse koodi haldamine ja Cypressi käivitamine. Koodi haldamine on aga tänu POM disainimustri kasutamisele

lihtne, sest koodimuudatuse koha otsimisele kulub vähe aega ja muudatus tuleb teha reeglina ühes kohas.

Käesoleva diplomitöö raames see oht, et automatiseerimine vähendaks loovusliku lähenemise vähenemise tõttu vigade leidmise arvu, pole tõsine. Näiteks kui on tarvis varieerida väärtusi kliendi loomisel, saab kliendi kõigepealt Cypressi abiga valmis luua, teha manuaalsed muudatused ja salvestada see klient *clientid.t*xt faili, seejärel jätkata objekti loomise osalise vooga, mis võtab selle kliendi sealt failist. Analoogselt saab toimida arvestiga – objekti loomisel võetakse arvesti failist *meterid.txt* ning ka objektidega – lepingute loomisel võetakse objekt failist *objectid.txt.* Teiste sõnadega – mänguruumi on sama palju kui enne ja aega eri variantide läbitegemiseks on rohkem kui enne.

Järelikult võib öelda, et korduvtegevusi automatiseeritult sooritades on nüüd testimine efektiivsem ja korduvtegevuste sooritamine võtab vähemalt viis korda vähem aega kui manuaalne sooritamine.

Eesmärk 2: vähendada inimlikest eksimustest tulenevaid vigasid.

Vigade tegemine manuaalsel testimisel nimetatud korduvtegevuste kontekstis puudutab peamiselt lepingute loomist: kliendi ja arvesti loomisel reeglina eksimisvõimaluste tõenäosus suur pole. Üks selline viga, mida manuaalselt testides ette tuleb, on näiteks see, et võrgulepingu ühtlase arveldamise väärtus jääb vormil valimata ning kui see leping nii väljastada, siis seda väärtust enam kasutajaliideses lisada ei saa ja tuleb otsast peale alustada või logida andmebaasi sisse ja muuta seal. Nüüd saab aga lasta Cypressil teha võrguleping valmis nii, et see ainult salvestatakse ja testija peab vaid lisama selle ühe väärtuse ning lepingu salvestama ja kinnitama. Seega võib öelda, et teataval määral väheneb korduvtegevusi automatiseerides ka inimlikest eksimustest tulenevate vigade arv.

Eesmärk 3: võimaldada testijal keskenduda eelkõige testimise sisulisele poolele.

Sel ajal, kui lahendus loob näiteks uut lepingut, on nüüd testijal võimalus mõelda juba testloo järgmistele sammudele, teha vajalikke märkmeid ja muud sellist: see aga just hoiabki mõtte olulisel, laskmata kõrvale kalduda ebaolulistesse detailidesse. Seega -

manuaalse testimisega võrreldes on nüüd paranenud testija võimalus kontsentreeruda olulisele.

Eesmärk 4: parandada Syneralli testija tööalaseid arenemisvõimalusi: lisaks testitava rakenduse äriloogika tundmaõppimisele, mis on kindlasti väga tähtis manuaalse testija töös, on automatiseerides võimalik arendada ka tehnilisemat laadi teadmisi ja oskusi.

Testijal on nüüd võimalus arendada oma teadmisi JavaScript'ist, versioonihaldussüsteemist Git, Cypress'ist ja kõigest muust, millega ta teste automatiseerides ja hallates kokku puutub – seega on diplomitöö tulemusena testija arenemisvõimalused Syneralli tiimis avardunud. Kindlasti arendab testija end ka siis, kui automaattestimist ei tehta, aga manuaalse testimisega piirdudes need võimalused on oluliselt kesisemad. Selleks, et käia ajaga kaasas ja arendada ka oma erialaseid tehnilisi teadmisi, ei piisa kursustest, kui neid teadmisi ei rakendata reaalses elus.

Eesmärk 5: luua alus selleks, et Syneralli testijad ka edaspidi püüaks leida viise, kuidas oma tööd automatiseerides tõhusamaks ja huvitavamaks muuta.

Tihti on vaja teha ainult esimesed sammud selleks, et suunata inimest mõtlema väljapoole sellest kastist, milles ta varasemalt on tegutsenud: nagu Laozi on öelnud, et „tuhandemiiline („*li*“) teekond algab esimesest sammust“ [[46], lk 71]. Kuigi esimese sammu astumine võiks toimuda ka muul viisil, on see tihti keeruline olukorras, kus ajaline surve ja töökoormus on suur ning testijalt oodatakse, et asjad saaksid õigeks ajaks testitud ja dokumenteeritud. Nüüd on see esimene samm astutud. Automatiseerimise abil saavutatud ajaline võit loob samuti paremad eeldused selleks, et teise sammu astumine oleks lihtsam.

Eesmärk 6: luua alus Syneralli UI suitsu- ja regressioonitestide loomiseks.

Kuna lahendus on loodud modulaarsena POM disainimustrit kasutades, siis on võimalik taaskasutada juba olemasolevaid funktsioone kasutajaliidese suitsutestide loomisel. Suitsutestide jaoks tuleb neid lihtsalt täiendada, lisades koodi vajalikke tulemuste kontrolle (*assertions*). Edasiste, seni katmata funktsionaalsuste testimise lisamine on analoogne olemasolevale. Näiteks selleks, et hakata looma rakendusele suitsuteste, võib olemasoleva lahenduse kloonida ning olemasolevate spec.js failide asemele hakata kirjutama suitsuteste kasutades ja refaktoriseerides olemasolevaid funktsioone ning

lisades uusi POsid ja funktsioone. Sama kehtib regressioonitestide puhul. Seega nii suitsu- kui ka regressioonitestide loomist ei pea enam alustama tühjalt kohalt, vaid saab võtta aluseks olemasoleva lahenduse.

Eelneva alusel võib kokkuvõtvalt öelda, et diplomitöös seatud 6 eesmärki on saavutatud.

6 Edasised sammud

Kuna selle töö skoobis ei olnud versioonihalduse koostöö juurutamine, siis just see on järgmine loogiline samm, sest tiimis on teine testija veel.

Töö tulemusi on edaspidi võimalik kasutada nii Syneralli kasutajaliidese suitsu- kui ka regressioonitestide loomisel, kuna lahenduse funktsioonid on taaskasutatavad. Olemasoleva projekti saab võtta aluseks, seda refaktoriseerida ja täiendada ja sealt testide kirjutamisega edasi minna. Kõigepealt jätkatakse suitsutestide loomisega.

Ühe järgmise sammuna tasub kaaluda massarvelduse taustatööde – Arvete loomine ja Arvete väljastamine – käivitamise automatiseerimist, sest arveldamine on Synerallis üks kriitilisemaid funktsionaalsusi. Massarveldamise taustatöö käivitamise tulemuseks on taustatöö logi – kui käivitada taustatöö samade andmetega uue versiooni peal ja võrrelda selle logi eelneva versiooni taustatöö logi vastu ja logid on identsed, siis loetakse test õnnestunuks, vastasel korral saadetakse manuaalsesse kontrolli.

7 Kokkuvõte

Synerall on kommunaalteenuseid pakkuvate ettevõtete jaoks loodud kliendiinfo- ja arveldussüsteem. Diplomitöö autor on selle süsteemi arendustiimi testija.

Käesoleva diplomitöö põhiprobleemiks on Syneralli testimisprotsessi ebatõhusus: kogu testimine on seni läbiviidud manuaalselt. Syneralli testimises on aga tegevusi, mida tuleb sooritada korduvalt: näiteks klientide, objektide ja lepingute loomine. Manuaalselt kulub sellise tegevuse peale keskmiselt peaaegu pool tundi päevas. See pole kõige tõhusam viis testimisressurssi kasutada. Ka psühholoogilise poole pealt vaadatuna mõjutab rutiinsete tegevuste osakaal töös inimese tööga rahulolu. Rahulolev töötaja aga on suurema tõenäosusega tõhusam testija. Manuaalse testimise käigus inimene eksib aegajalt, misjärel tuleb osadel juhtudel sama tegevust veel korrata. Niiviisi objekte, kliente ja arvesteid luues kaob mõnikord testimisel fookus testloo sisuliselt poolelt.

Probleemist tulenevalt seati töös eesmärgid testimisprotsessi tõhustamiseks ja püüti leida sobiv lahendus eesmärkide saavutamiseks. Kõigepealt otsustati rutiinsed tegevused automatiseerida, valiti sobivad vahendid töö teostamiseks, milleks osutusid automatiseerimise vahend Cypress, koodiredaktor Visual Studio Code ja versioonihaldussüsteem Git koos majutusteenusega GitHub. Vajalikud andmed otsustati genereerida peamiselt npm paketihalduri kaudu installeeritavate JavaScript teekide abil. Seejärel sõnastati töös kasutatav metoodika. Metoodikast lähtudes identifitseeriti korduvtegevused ja tehti valik, mida neist automatiseerida. Järgnes tegevuste automatiseerimise teostamine. Lahenduses järgitakse *Page Object* mudelit, mis aitab vähendada koodi dubleerimist ning koodi paremini hallata. Syneralli eri instantside erinevad väärtused otsustati salvestada keskkonnamuutujatena vastava keskkonna konfiguratsioonifaili ja käivitada testimisel Cypress soovitud konfiguratsiooniga. HSY keskkonda sisselogimiseks AD-kontoga kasutatakse npm-i kaudu installeeritavat pluginat cypress-ntml-auth.

Töö tulemuste osas esitatakse kõigepealt mõõtmistulemused. Mõõdeti aega, mis kulub vaadeldud tegevuste sooritamiseks automatiseeritult ja võrreldi seda ajaga, mis kulub samade tegevuste sooritamiseks manuaalselt. Ajaline võit oli keskmiselt 23.08 minutit päevas. Lahendus on inimesest keskmiselt 5 korda kiirem. Seejärel vaadeldi töö tulemusi seatud eesmärkide suhtes. Automatiseeritult kulub rutiinsete tegevuste sooritamiseks keskmiselt 5 korda vähem aega. Inimlikest eksimustest tulenevate vigade määr on vähenenud selle arvelt, et kliente, objekte ja lepinguid luuakse nüüd automatiseeritult. Rutiinsete tegevuste automatiseerimine võimaldab testijal keskenduda testloo sisulisele poolele ja teha sel ajal, kui Cypress töötab, midagi kasulikku. Testija töö rutiinsuse määr on väiksem ja arenemisvõimalused ning tööga rahulolu suurem. Tehtud töö tulemusena on ka loodud alus Syneralli kasutajaliidese suitsutestide kirjutamiseks. Käesolevas diplomitöös seatud eesmärgid on saavutatud.

Edasiste sammudena tuleks esiteks juurutada versioonihalduse koostöö osa, et automatiseerimisele saaks panustada soovi korral ka teine testija. Teiseks võiks olemasolevat lahendust aluseks võttes luua põhifunktsionaalsust katvad kasutajaliidese suitsutestid. Kolmandaks võiks mõelda sellele, kuidas automatiseerida massarvelduse taustatööde käivitamine ja võrdlemine samade andmete vastu Syneralli uue ja eelmise versiooni vahel.

Kasutatud kirjandus

[1] Erinevus brauseriülese testimise ja tundliku testimise vahel Lambda test. [WWW] 2019, https://et.bccrwp.org/compare/difference-between-cross-browser-testing-responsive-testing-lambdatest-82810e (12.04.2020)

[2] Why use heuristics in Software testing? [WWW] https://conorfi.com/software-testing/software_testing_heuristics/ (12.04.2020)

[3] Intellisense. [WWW] https://code.visualstudio.com/docs/editor/intellisense (12.04.2020)

[4] Mit Licence. [WWW] https://tldrlegal.com/license/mit-license (12.04.2020)

[5] What is Planning Poker in Agile? [WWW] https://www.visual-paradigm.com/scrum/what-is-agile-planning-poker/ (12.04.2020)

[6] What is regression testing? [WWW] https://smartbear.com/learn/automated-testing/what-is-regression-testing/ (30.04.2020)

[7] Smoke testing. [WWW] http://softwaretestingfundamentals.com/smoke-testing/ (30.04.2020)

[8] 2019 Diffblue Developer Survey: What's wrong with software speed, quality, and cost? [WWW] https://www.diffblue.com/Education/research_papers/2019-diffblue-developer-survey/ (28.04.2020)

[9] What's automation? [WWW] https://www.redhat.com/en/topics/automation (8.3.2020)

[10] Maurice Siteur, "Implementing Test Automation", Leanpub, 2019, https://leanpub.com/ImplementationTestAutomation (7.3.2020)

[11] G. Tayar " Cypress vs Selenium WebDriver: Better, or just different?". [WWW] https://applitools.com/blog/cypress-vs-selenium-webdriver-better-or-just-different?utm_referrer=https%3A%2F%2Fdzone.com%2Farticles%2Fcypress-vs-selenium-webdriver-which-is-better-for (05.03.2020)

[12] Amir Rustamzadeh, "Introducing Firefox and Edge Support in Cypress 4.0". [WWW] https://cypress.io/blog/2020/02/06/introducing-firefox-and-edge-support-in-cypress-4-0 (7.3.2020)

[13] Cypress Pricing Plans. [WWW] https://www.cypress.io/pricing/ (12.04.2020)

[14] ChromeDriver – WebDriver for Chrome. [WWW] https://chromedriver.chromium.org/ (18.05.2020)

[15] Installing Cypress. [WWW] https://docs.cypress.io/guides/getting-started/installing-cypress.html (12.04.2020)

[16] RIP Tutorial, "Selenium Webdriver Installation or Setup". [WWW] https://riptutorial.com/selenium-webdriver/example/2963/installation-or-setup (12.04.2020)

[17] Introduction to Cypress. [WWW] https://docs.cypress.io/guides/core-concepts/introduction-to-cypress.html (12.04.2020)

[18] Shreya Bose, "How to use Wait Commands in Selenium WebDriver". [WWW] https://www.browserstack.com/guide/wait-commands-in-selenium-webdriver (12.04.2020)

[19] S. O'Grady, "The RedMonk Programming Language Rankings: January 2020". [WWW] https://redmonk.com/sogrady/2020/02/28/language-rankings-1-20/ (07.03.2020)

[20] Popic et al, "Data generators: a short survey of techniques and use cases with focus on testing", 2019, https://www.researchgate.net/publication/335397247_Data_generators_a_short_survey_of_techniques_and_use_cases_with_focus_on_testing (17.02.2020)

[21] id/check.artega.biz. [WWW] http://id-check.artega.biz/ (26.04.2020)

[22] Fake name generator. [WWW] https://www.fakenamegenerator.com/order.php (26.04.2020)

[23] finnish-ssn. [WWW] https://github.com/vkomulai/finnish-ssn (26.04.2020)

[24] isikukood. [WWW] https://www.npmjs.com/package/isikukood/v/1.2.3 (26.04.2020)

[25] finnish-business-ids. [WWW] https://github.com/vkomulai/finnish-business-ids (26.04.2020)

[26] finnish-bank-utils. [WWW] https://github.com/vkomulai/finnish-bank-utils (26.04.2020)

[27] Developer Survey Results 2019. [WWW] https://insights.stackoverflow.com/survey/2019/ (7.3.2020)

[28] 1.1 Getting started – About Version Control. [WWW] https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control (15.03.2020)

[29] Scott Chacon, Ben Straub "Pro Git", Apress, 2014. [WWW] https://git-scm.com/book/en/v2 (15.03.2020)

[30] Jonathan Mines, "The Ultimate GithubCollaboration Guide" [WWW] https://medium.com/@jonathanmines/the-ultimate-github-collaboration-guide-df816e98fb67 (15.03.2020)

[31] GitHub.com Help Documentation. [WWW] https://help.github.com/en/github (15.03.2020)

[32] GitLab Pricing. [WWW] https://about.gitlab.com/pricing/ (15.03.2020)

[33] Pricing. [WWW] https://github.com/pricing (15.03.2020)

[34] Alan Page "The "A" Word: Under the Covers of Test Automation", Leanpub, 2013. [Online] http://leanpub.com/TheAWord (17.02.2020)

[35] Get npm! [WWW] https://www.npmjs.com/get-npm (24.04.2020)

[36] Page Object Model (POM) | Design Pattern. [WWW] https://medium.com/tech-tajawal/page-object-model-pom-design-pattern-f9588630800b (24.04.2020)

[37] Page Object Model in Selenium. [WWW] https://www.gcreddy.com/2016/09/page-object-model-in-selenium-2.html (17.05.2020)

[38] Configuration. [WWW] https://docs.cypress.io/guides/references/configuration.html (24.04.2020)

[39] Configuration API. [WWW] https://docs.cypress.io/api/plugins/configuration-api.html (24.04.2020)

[40] Dealing with Environment Variables in Cypress.io. [WWW] https://www.bignited.be/2019/12/31/dealing-with-environment-variables-in-cypress-io/ (24.04.2020)

[41] Cypress-ntlm-auth. [WWW] https://www.npmjs.com/package/cypress-ntlm-auth (25.04.2020)

[42] Date-fns. [WWW] https://www.npmjs.com/package/date-fns (26.04.2020)

[43] Math.random(). [WWW] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math/random (24.04.2020)

[44] 1.3 Getting Started – What is Git? [WWW] https://git-scm.com/book/en/v2/Getting-Started-What-is-Git (26.04.2020)

[45] 2.5 Git Basics – Working with Remotes. [WWW] https://git-scm.com/book/en/v2/Git-Basics-Working-with-Remotes (26.04.2020)

[46] Laozi, "The Living Dao: The Art and Way of Living A Rich & Truthful Life". Tõlge annotatsioonidega: Lok Sang HO Lingnan University. [WWW] https://www.ln.edu.hk/econ/staff/daodejing%2822%20August%202002%29.pdf (17.05.2020)

[47] Ken Schwaber, Jeff Sutherland, "The Scrum Guide" , 2017. [WWW] https://scrumguides.org/scrum-guide.html (8.3.2020)

[48] Scrum Sprint. [WWW] https://t2informatik.de/en/smartpedia/scrum-sprint/ (8.3.2020)

[49] John X. Wang "Lean Manufacturing: Business Bottom-Line Based", CRC Press: Taylor & Francic Group, 2011

[50] Tilo Linz, "Testing in Scrum: A Guide to Software Quality Assurance in the Agile World", Rocky Nook Inc. , 2014

[51] "Practical Kanban Board Examples. [WWW] https://kanbanize.com/kanban-resources/kanban-software/kanban-board-examples (8.3.2020)

[52] Lead Time vs Cycle Time – Details explained. [WWW] https://kanbanize.com/kanban-resources/kanban-software/kanban-lead-cycle-time/ (8.3.2020)

Lisa 1 – Scrum

Scrum raamistiku arendasid välja Ken Schwaber ja Jeff Sutherland.

Scrum raamistik koosneb 3 rollist, 5 sündmusest ja 3 artefaktist [47].

Scrumi tiimi 3 rolli on järgnevad [47]:

- Tooteomanik (*Product Owner*) on vastutav tootekuhja haldamise ja toote väärtuse maksimeerimise eest.
- Arendustiim (*Development team*) on iseorganiseeruv, ristfunktsionaalne ja vastutab ühiselt. Optimaalne suurus on 3-9 liiget.
- Scrum-meister (*Scrum master*) on Scrum tiimi teener-liider, kes hõlbustab tiimi koostööd ja aitab tiimil luua kõrge väärtusega toodet.

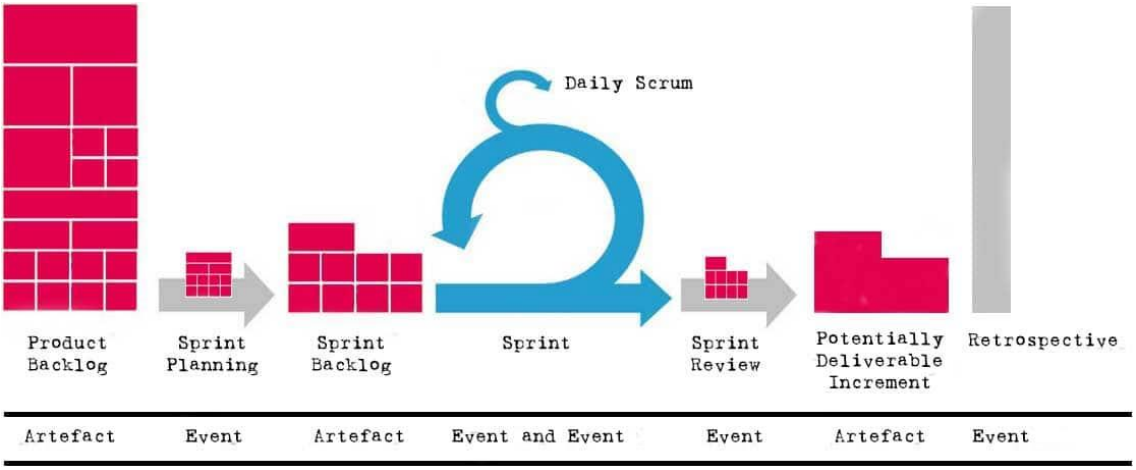
Scrumi 5 sündmust on järgnevad [47]:

- Sprint
- Sprindi planeerimine (*Sprint planning*)
- Igapäevane Scrum (*Daily Scrum*)
- Sprindi ülevaatus (*Sprint Review*)
- Sprindi tagasivaade (*Sprint Retrospective*)

Scrumi 3 artefakti on järgmised [47]:

- Toote kuhi (*Product backlog*)
- Sprindi kuhi (*Sprint backlog*)
- Inkrement (*Increment*)

Joonis 12 illustreerib, kuidas Scrumi artefaktid ja sündmused on omavahel seotud: tootekuhjast valitakse sprindi planeerimisel mingi hulk kasutuslugusid sprindi kuhja, mis teostatakse sprindi vältel. Sprindi jooksul toimuvad ka igapäevased Scrum püstijalakoosolekud. Sprindi lõpus tehakse sprindi ülevaatus. Sprindi tulemuseks on potentsiaalselt tarnitav inkrement. Enne järgmise sprindi algust toimub veel käesoleva sprindi tagasivaade.



Joonis 12. Scrumi artefaktid ja sündmused [48].

Lisa 2 – Kanban

Termin Kanban otsetõlkes: ”*kan*” tähendab visuaalne ja ”*ban*” tähendab kaarti või tahvlit [[49], lk 203].

Kanban on loodud pidevalt optimeerima tootmisprotsessi ja vähendama aega, mis kulub kogu toote tootmiseks.- teiste sõnadega, Kanbani idee on garanteerida ülesannete stabiilne voog [[50], lk 34].

Kanban baseerub 3 võtmekomponendil [[50], lk 33]:

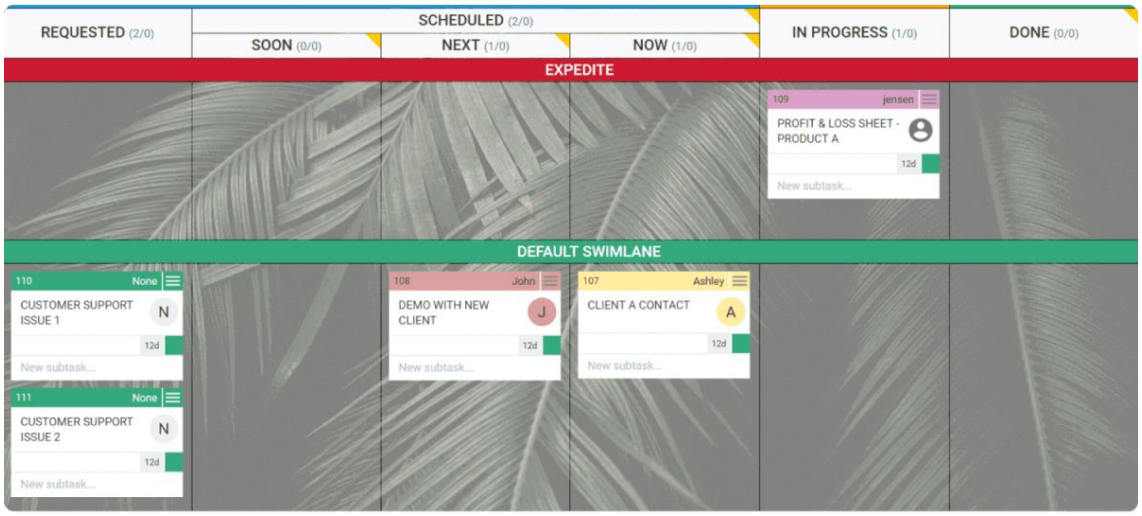
- Kanban tahvel (*Kanban board*) – protsess on visualiseeritud tahvlil kasutades kaarte. Sammud protsessis on representeeritud tulpadena ja ülesannet kaartidega, mida ülesande progresseerudes liigutatakse tahvlil vasakult paremale.

- WIP (*Work-In-Progress*) limiit – iga protsessi sammu jaoks on määratud WIP limiit, mis määrab, mitu kaarti võib seal samaaegselt olla. Niikaua kuni sammu limiit pole täis, tõmmatakse sinna kaart sellele sammule eelnevast sammust – seda lähenemist kutsutakse nn tõmbesüsteemiks *(pull system*).

- Teostusaeg (*Lead Time*) – Kanbanit kasutatakse pideva ülesannetevoo optimeerimiseks minimeerides kogu väärtusvoo keskmist teostusaega.

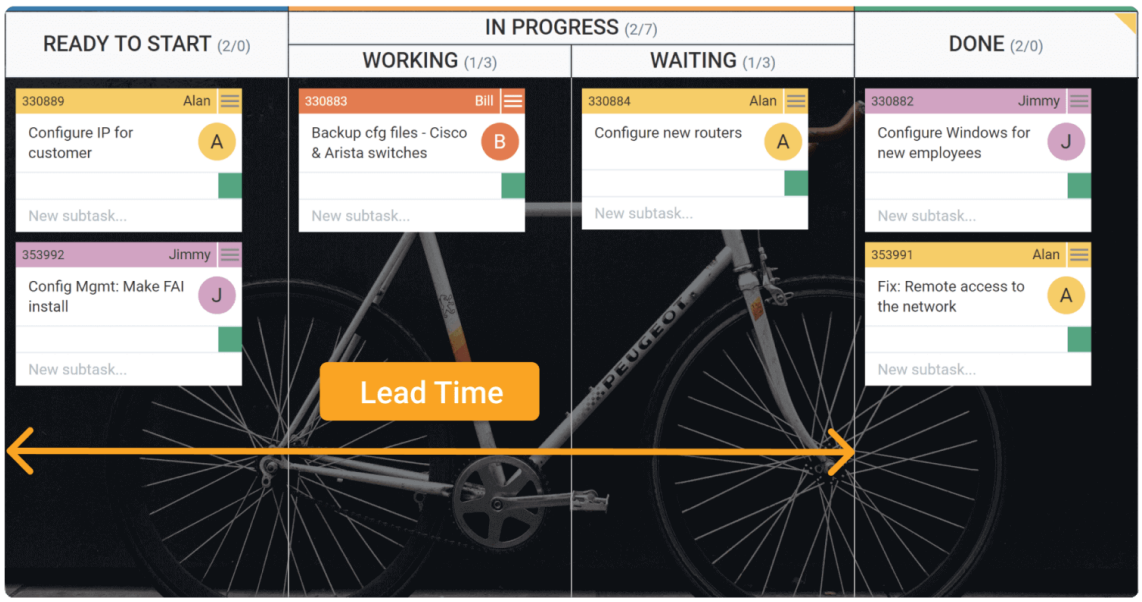
Erinevalt Scrumist ei kasuta Kanban iteratsioone või sprinte, vaid lubab tarnida tehtud tööd kasvõi ükshaaval. Ühtlasi ei eelda Kanban *timeboxingut* ehk teiste sõnadega - Kanbani mudel ei eelda seda, et toodet peaks tarnima eelnevalt määratletud intervallide kaupa [[50], lk34].

Järgnevalt on toodud näide Kanban tahvlist, milles on peale vertikaalsete sammude kasutatud ka horisontaalseid radasid eristamaks ajatundlikke ülesandeid – *Expedite* rajal on ülesanded, millega tuleb tegeleda kõigepealt ja siis *Default* rajal muud ülesanded, millega pole nii kiire (Joonis 13).



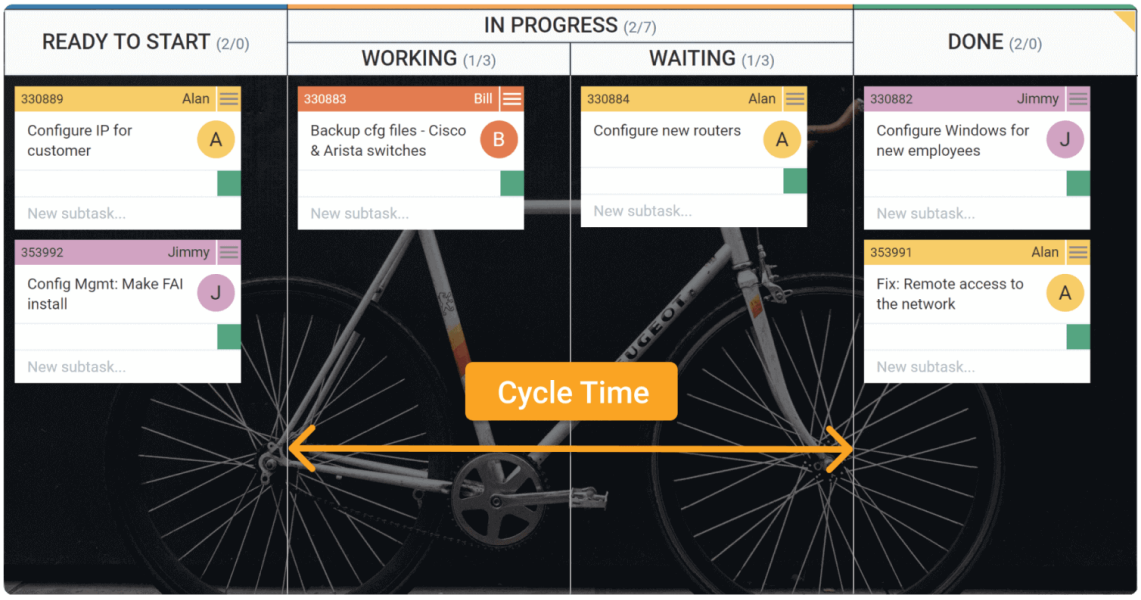
Joonis 13. Kanban tahvel horisontaalradadega [51].

Kanbani kontekstis eristatakse teostusaega (*Lead time*) ja tsükliaega (*cycle time*). Teostusaeg on periood sellest hetkest alates kui ülesanne ilmub töövoogu kuni selle lõpliku valmimiseni (Joonis 14).



Joonis 14. Teostusaeg [52].

Tsükliaeg on periood, mis algab sellest hetkest, kui keegi ülesandega tegelema hakkab ja lõpeb siis, kui ülesanne on lõplikult teostatud (Joonis 15).



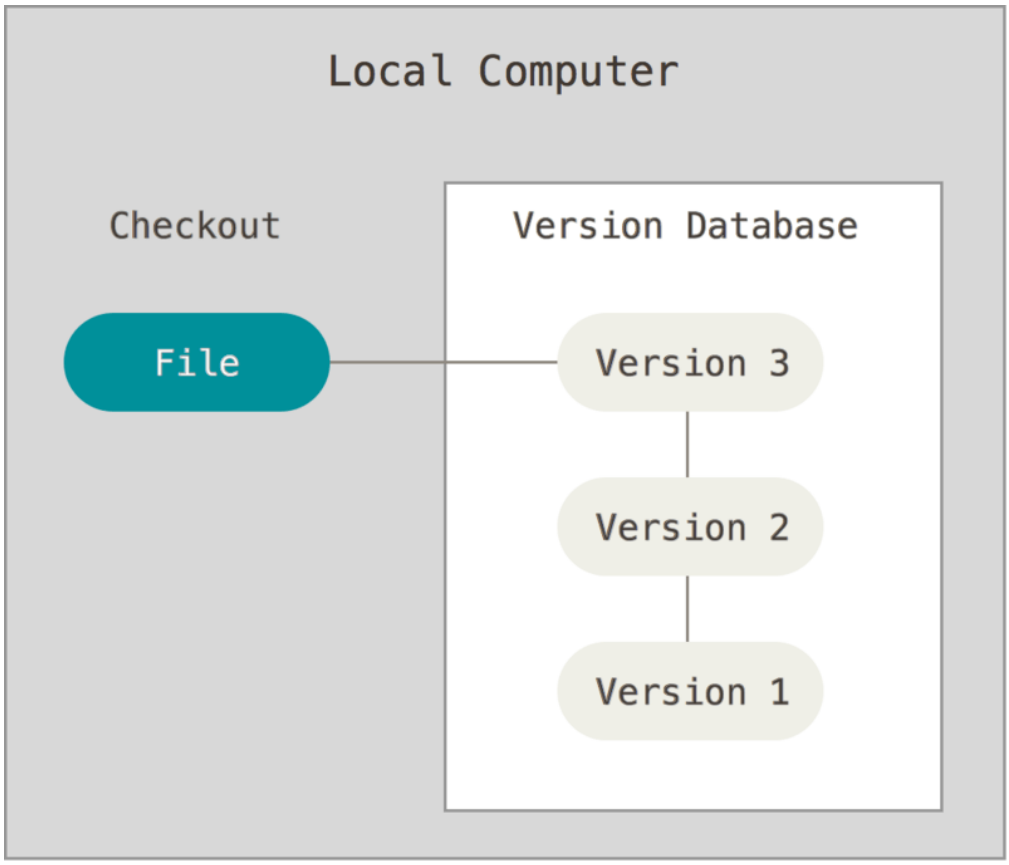
Joonis 15. Tsükliaeg [52].

Teiste sõnadega - tsükliaeg ei arvesta seda aega sisse, mis ülesanne võib oodata enne, kui see töösse võetakse [52].

Lisa 3 – Versioonihaldussüsteemide tüübid

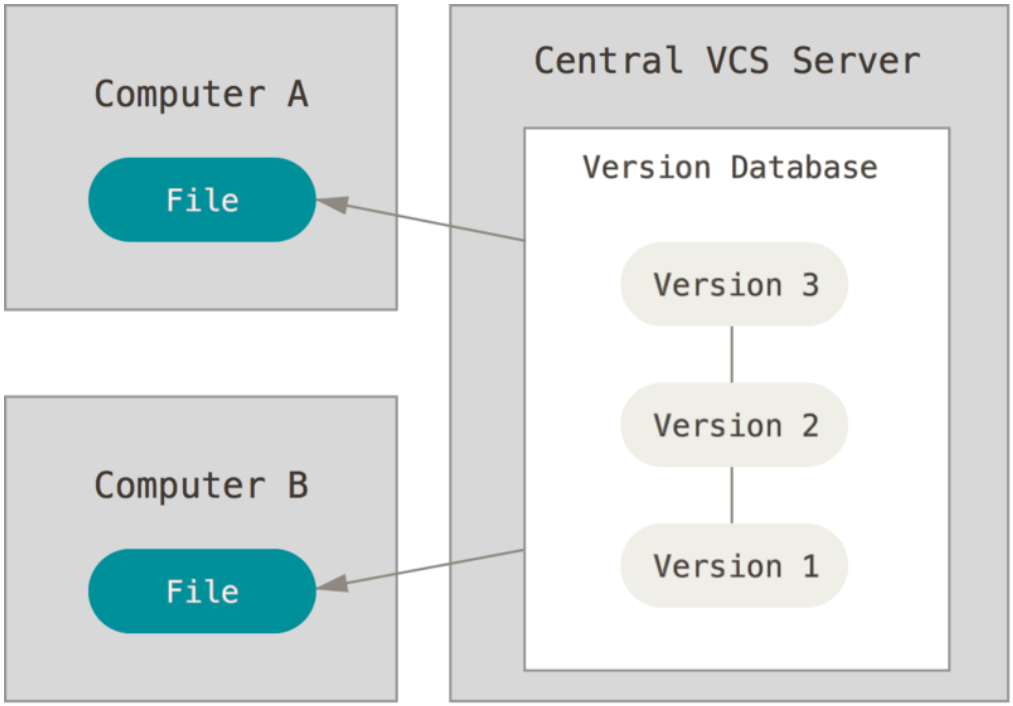
Versioonihaldussüsteemid võivad olla kohalikud, tsentraliseeritud või hajusad [28].

Kohalik versioonihaldussüsteem – üsna levinud on failide manuaalne kopeerimine teise kataloogi. See on väga lihtne meetod, aga ka veaohtlik, sest on pole raske unustada, millises kataloogis parasjagu ollakse ja kirjutada ekslikult valesse faili. Selle probleemi lahendamiseks on arendatud kohalik versioonihaldussüsteem (Joonis 16), mis kasutab lihtsat andmebaasi, mis failidesse tehtud muudatused versioonihalduskontrolli all hoiab. Üks tuntumaid neist oli RCS, mis säilitab failiversioonide erinevusi kohalikus süsteemis. [28]



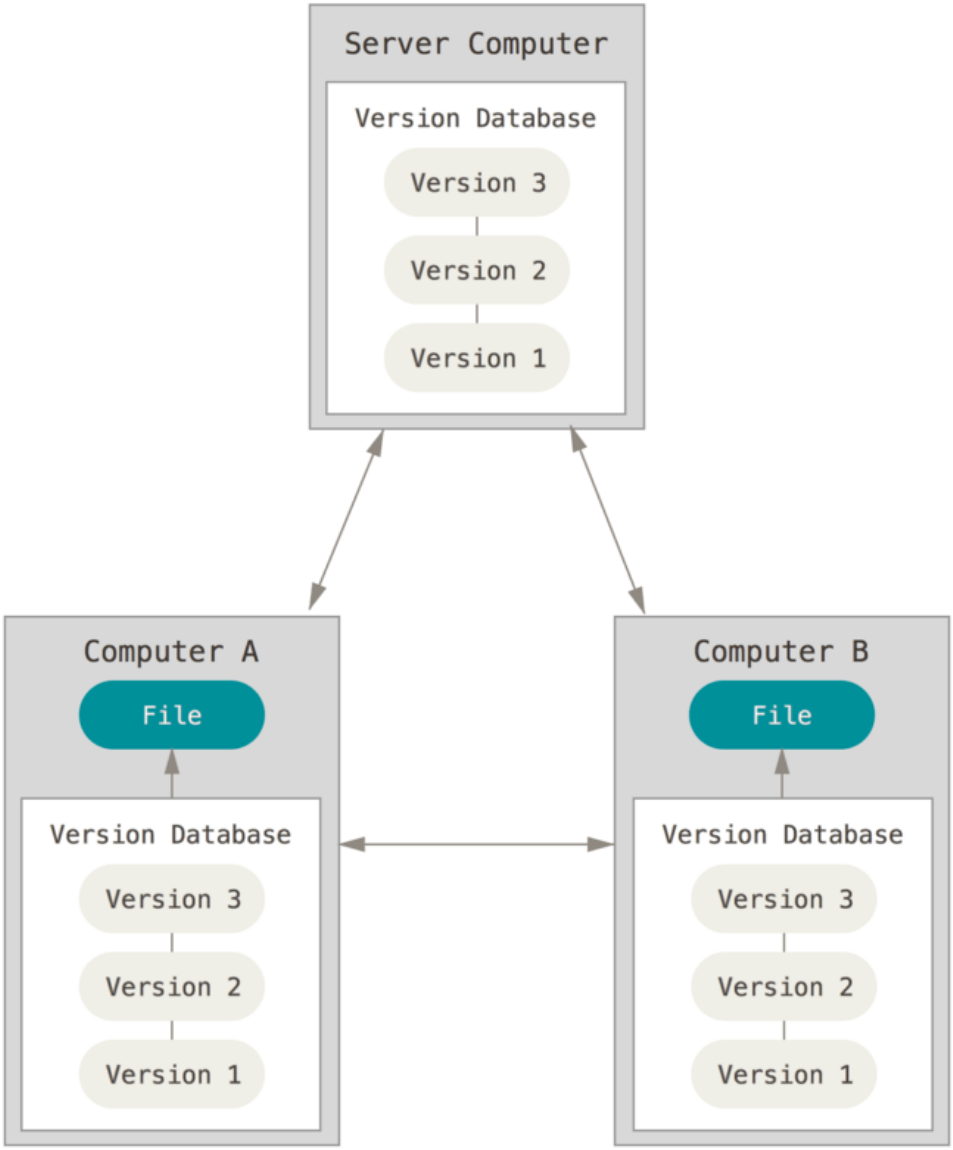
Joonis 16. Kohalik versioonihaldussüsteem [28].

Tsentraliseeritud versioonihaldussüsteemis (Joonis 17), mis oli pikka aega versioonihalduse standardiks, asuvad kõik versioneeritud failid ühes keskses serveris ja sellest kohast tõmbavad endale faile erinevad kliendid. Seda tüüpi on näiteks Subversion, CVS ja Perforce. Võrreldes kohaliku versioonihaldusega on selle suureks eeliseks see, et annab arendajale parema pildi sellest, millega teised projektiliikmed tegelevad. Selle tüübi nõrk koht on see, et keskne server on ainus koht, kus hoitakse projekti kogu ajalugu. Kui see lüli üles ütleb ja varundus pole olnud piisav, siis võivad tagajärjed olla tõsised. [28]



Joonis 17. Tsentraliseeritud versioonihaldussüsteem [28].

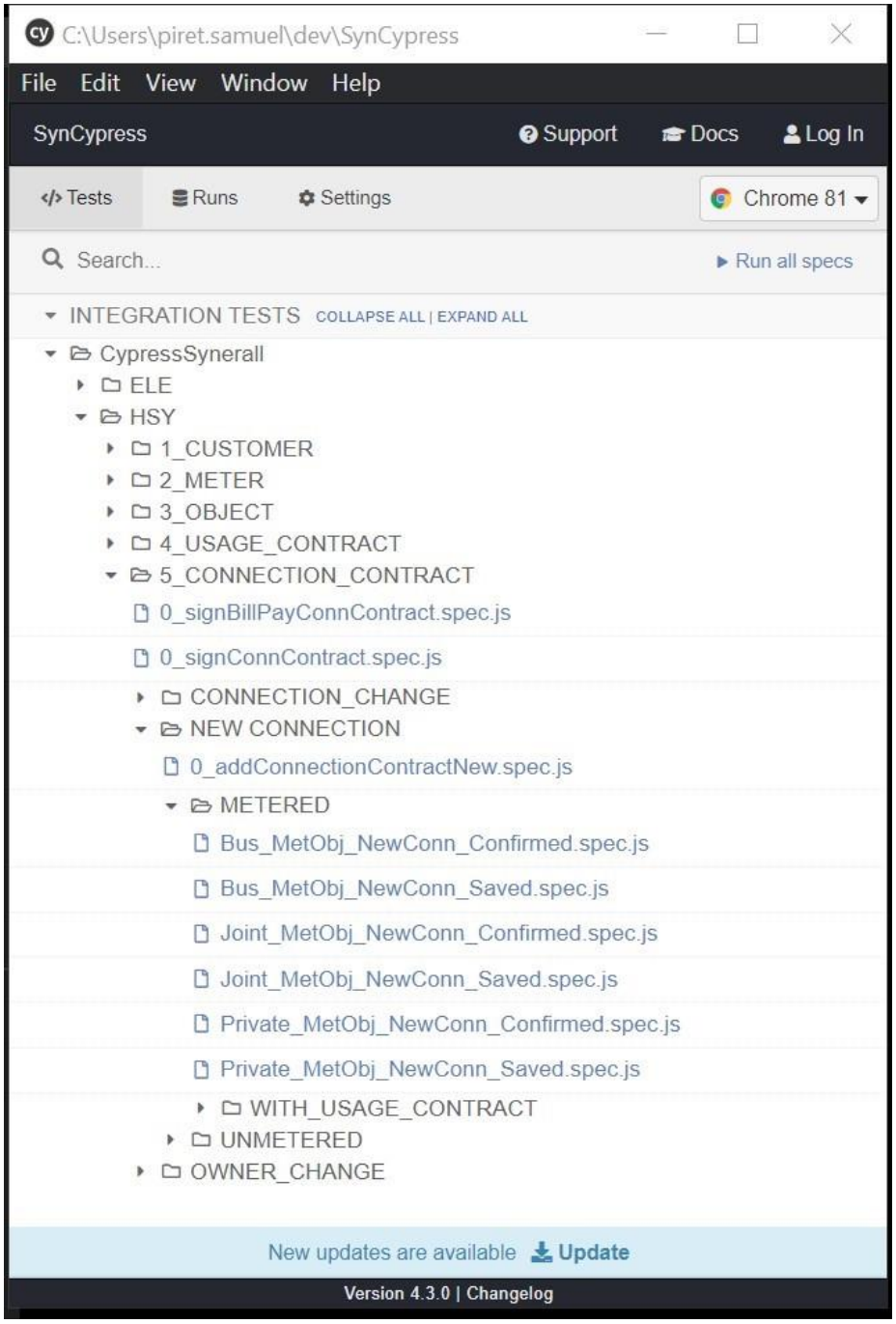
Hajusate versioonihaldussüsteemide (Joonis 18), nagu näiteks Git, Bazaar ja Mercurial, puhul ei lae kliendid endale kaughoidlast alla ainult uusimat failide ülesvõtet nagu keskse versioonihalduse mudeli korral, vaid iga kliendi kohalikus masinas on kogu projekti ajalugu. Selle tüübi üheks eeliseks keskse versioonihalduse ja ka kohaliku versioonihalduse ees on see, et iga kliendi kloon on ühtlasi andmete täielik varukoopia [28].



Joonis 18. Hajus versioonihaldussüsteem [28].

Lisa 4 – Cypressi graafiline kasutajaliides

Joonis 19 illustreerib Cypressi graafilist kasutajaliidest.



Joonis 19: Cypressi graafiline kasutajaliides.