TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Informaatikainstituut

IDK40LT

Tanel Annuste 134650IABB

AUTOMAATTESTIMISE RAAMISTIKE ANALÜÜS JA VÕRDLUS TELIA EESTI AS NÄITEL

Bakalaureusetöö

Juhendaja: Inna Švartsman
MSc
Lektor

Tallinn 2016

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: Tanel Annuste

20.05.2016

Annotatsioon

Antud töö põhilisteks eesmärkideks on analüüsida hetkel Telia Eesti AS ettevõttes olevaid automatiseeritud teste ning võrrelda nende kaudu Robot Framework ja Selenium IDE raamistike ning investeeringutasuvuse valemite abil välja selgitada, kas praegune olukord on jätkusuutlik ja automaattestimine on ennast ära tasunud.

Töö käigus kirjeldan testimise olulisust, toon välja automaattestimise olulised positiivsed ja negatiivsed küljed ning selgitan lühidalt nii Robot Framework kui ka Selenium IDE raamistike.

Töö tulemustest selgub, et erinevate parameetrite järgi analüüsides osutus paremaks raamistikuks Robot Framework. Telia Eesti AS Dealgate näitel on automaattestimine ennast ära tasunud – 2 aastase aja jooksul on ajaline võit tervelt 47%.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 42 leheküljel, 5 peatükki, 16 joonist, 1 tabeli.

Abstract

Analysis and comparison of automated testing frameworks of Telia Eesti AS

The main objectives of the given study is to analyze the currently available automated tests of Telia Eesti AS and comparing Robot Framework and Selenium IDE frameworks through them and to find out using return on investment formulas whether the current situation is sustainable and if automated testing has paid off.

During the study I describe the importance of testing, bring out the positive and negative aspects of automated testing and briefly explain Robot Framework as well as Selenium IDE environments.

The results of the study show that after analysing using different parameters, Robot Framework turned out to be the better tool. Based on Telia Eesti AS Dealgate, automated testing has paid off – 47% time gain during a 2 year period testing period.

The thesis is in Estonian and contains 42 pages of text, 5 chapters, 16 figures, 1 table.

Lühendite ja mõistete sõnastik

Regressioontestimine	Regression testing Testimine, mille peamiseks eesmärgiks on olla veendunud, et tarkvara kõik funktsionaalsused muudatuse tagajärjel endiselt töötavad.
GUI testimine	Graphical user interface test Kasutajaliidese testimine.
Unit-testimine	Unit-testing Väiksemate süsteemi osade testimine – eesmärk olla kindel, et programmi eraldiseisvad osad töötavad.
Integratsiooni testimine	Integration-testing Süsteemi väikeste osade omavahelise kooskõla testimine – eesmärk olla kindel, et süsteemi väiksemad komponendid suudavad koos töötada.
Automaattestimine	Automated testing Protsess, kus kasutatakse spetsiaalset tarkvara, et testida süsteemi korrektset toimimist automaatselt.
Investeeringu tasuvus	Return on investment (ROI) Näitab, kui tasuv on investeering konkreetsesse projekti.
Oracle	Oracle Andmebaasisüsteem.
Selenium	Selenium Testimise tarkvara.
Robot Framework	Robot Framework Üldine automaattestimise raamistik.
RIDE	RIDE Testimise tarkvara.
Dealgate	Dealgate Telia Eesti AS klienditeenindaja keskkond.

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Tarkvara arendamise juures üks olulisemaid etappe on testimine. Testimisega alustatakse juba paralleelselt koodi kirjutamisega ning testimise lõpetab klient. Kui tarkvara on üles ehitatud selliselt, mis vajab pidevat arendamist/uuendamist, siis ei tohiks testimine lõppeda. Seda liiki testimist nimetatakse regressioontestimiseks – teatud perioodi tagant läbib süsteem regressioontestid, mille abil on võimalik tuvastada uuenenud tarkvaras vigu. Regressioontestid võib läbida nii manuaalsel teel kui ka automatiseeritud kujul. Automatiseeritud testide jaoks on olemas mitmeid erinevaid vahendeid ning nendel on erinevaid positiivseid ja negatiivseid külgi.

Antud töös on praktiliseks osaks Telia Eesti AS klienditeenindaja töökeskkond (Dealgate). Hakkan seda analüüsima erinevate parameetrite järgi. Töö eesmärgiks on võrrelda ning analüüsida erinevate parameetrite järgi praegu kasutusel olevat Robot Framework'i Selenium IDE raamistikuga. Samuti on oluliseks eesmärgiks hinnata investeeringutasuvuse alusel Dealgate'i testide jätkusuutlikkust ning leida pidepunkte, kus saaks midagi paremini teha.

1.1 Taust ja probleem

Aastal 2012 oli Telias kõik Dealgate testid manuaalsed. See tähendas tööd inimesele, kes pidi väga tihti rutiinselt manuaalselt jooksutama teste, et kontrollida, kas kõik toimib nii nagu peab. Alates 2013. aastast hakati teste automatiseerima. Antud töö eesmärgiks on välja selgitada, kas praegune süsteem on jätkusuutlik ning võimalusel leida kohti, kus saaks midagi veelgi paremaks teha.

Töös kasutatud näited ning praktiline pool on tehtud Telia Eesti AS klienditeenindaja keskkonnas (Dealgate's). Töö tulemused on eelkõige vajalikud Telia Eesti AS ettevõttele, kes saab nende põhjal vajadusel muudatusi teha. Lisaks sellele on töö vajalik ka minule, et saada suurem pilt automatiseerimise teoreetilistest alustest. Samuti aitab antud töö ka tulevasi automaattestijaid, kes saavad seda tööd lugedes üldpildi automaattestimisest ning paarist erinevast vahendist – nende headest ja halbadest külgedest.

1.2 Ülesande püstitus

1. Kirjeldan tarkvara testimise olulisust lõpptarbija jaoks. Toon välja automaattestimise positiivsed ja negatiised küljed ning paari erineva raamistiku head ja puudused.

2. Analüüsin erinevate parameetrite järgi praegusi automaattestide lahendusi Dealgate näitel. Võrdlen olemasolevaid Robot Framework teste Selenium IDE testidega.

3. Leian, kas hetkel toimiv süsteem on jätkusuutlik ning millised osad võivad osutuda problemaatilisteks.

4. Analüüsin automatiseeritud teste – võrdlen kunagist manuaalset süsteemi praeguse automatiseeritud süsteemiga ROI – investeeringutasuvuse kaudu.

1.3 Metoodika

Selleks, et jõuda eesmärgini, kirjeldan testide olulisust, võrdlen erinevaid lahendusi automaattestimisest üldiselt ning regressioontestimisest. Selleks, et võrrelda ja analüüsida olemasolevaid Robot Framework teste, kirjutan sarnase testi valmis Selenium IDE-s ning kirjeldan mõlema häid ja halbu külgi. Erinevate parameetrite analüüsiks kasutan ROI metoodikat – investeeringutasuvust. Selle abil selgitan välja, kas testide automatiseerimisse investeerimine on olnud tulus.

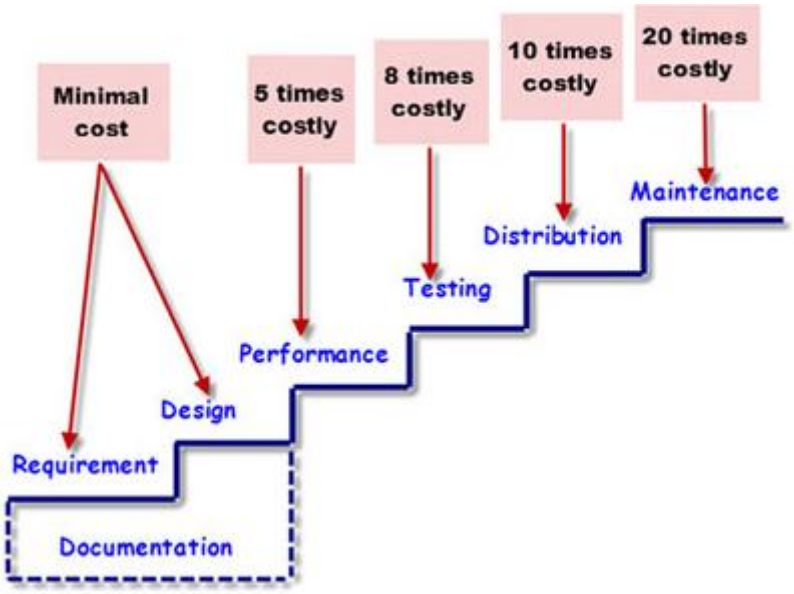
1.4 Ülevaade tööst

Töö on üles ehitatud selliselt, et alguses tuleb juttu üldiselt testimise olulisusest ning sellest, millised on regressioontestimise erinevad võimalused. Edasi tuleb juttu automaattestimisest – kirjeldan selle positiivseid ja negatiivseid omadusi. Samuti kirjeldan automaattestimise paarist erinevast vahendist (Robot Framework ja Selenium IDE). Seejärel analüüsin Telia Eesti AS näitel erinevate parameetrite järgi automaattestide raamistikke. Töö lõpus arvutan välja projekti investeeringutasuvuse spetsiaalse ROI valemi järgi, mille tulemusest selgub, kas automatiseerimine on olnud tulus.

2 Testimise olulisus

Lihtsamalt öeldes on testimise eesmärk leida programmis vigu. Tarkvara testimine hakkab pihta juba koodi kirjutamise hetkel. Esimesed testid viivad läbi arendajad, kuid paremaid tulemusi saavutavad tihti need, kes ei ole arendusega seotud (näiteks tulevased kasutajad). Tarkvaraga on seotud väga palju asju – enamike süsteemide üheks komponendiks on tarkvara. Eluliselt olulisemad süsteemid on kasutusel näiteks meditsiinis ning lennunduses. Seetõttu on eluliselt oluline, et tarkvara töötaks nii nagu peab [1] . Selleks, et olla kindel tarkvara kvaliteedis, nõuab see testimist.

Testimise juures on oluline, et mida varem viga leitakse, seda odavam on ka parandada. Seega tasub testimisega alustada juba võimalikult vara. Tavaliselt on testimine küll rahaliselt kulukas, kuid kui jätta testimise etapp vahele, võib kokkuvõttes osutuda projekt veelgi kulukamaks.



Joonis 1. Vea parandamise kulu sõltuvalt avastamisest [2]

Jooniselt 1 on näha, et mida kaugemas etapis viga avastatakse, seda kulukamaks osutub vea parandamine. See on illustratiivne joonis, mis näitab, kui oluline on testimine. Põhiline on veel ka see, et lisaks rahale kulutatakse ka aega vea parandamisele – mida hiljem viga avastada, seda kauem võtab aega parandamine. Selle arvelt võivad kannatada

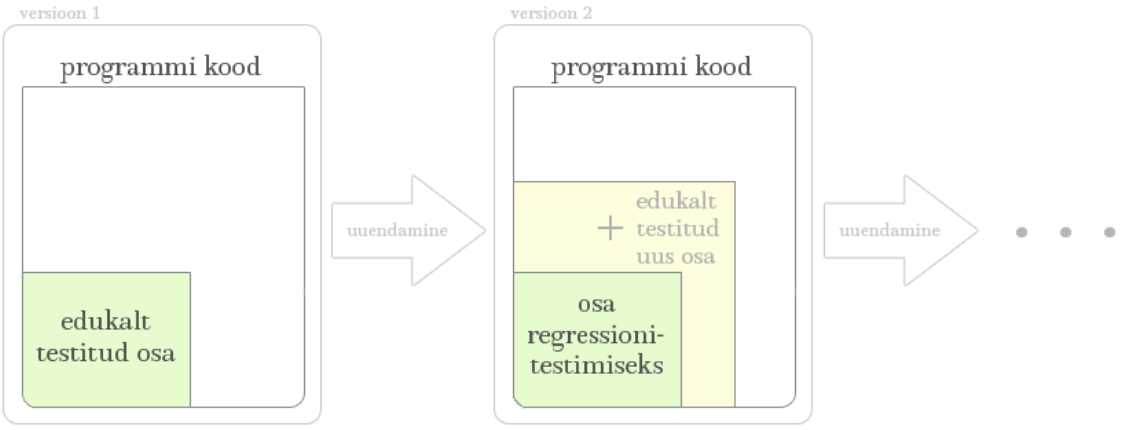
ka teised olulised projektid, kuna aeg tuleb tavaliselt võtta millegi arvelt. Tarkvara vähese testimise juures võib ohtu seada nii mõndagi [3] :

- Rahaline kahju – peamiselt klientide kaotus, kuid siia alla võivad kuuluda ka erinevad rahalised trahvid näiteks õigusaktide eest.
- Ajaline kahju – siia alla võib liigitada kaua aega võtvad protsessid tarkvaras või puudub töö tegemise võimalus süsteemi vea tõttu.
- Ettevõtte maine kahju – firmade ohuks võib olla see, kui ei ole võimalik kliendile soovitud teenust osutuda süsteemi vea tõttu. Selle tulemusena kliendid võivad antud ettevõtte vastu kaotada usalduse.
- Vigastus või surm – kui süsteem ei ole võimeline töötama nii nagu peab, siis võivad tagajärjed olla traagilised (näiteks meditsiin või lennundus).

Selleks, et olla kindel, kas süsteem toimib nii nagu peab, tuleb teda pidevalt testida. Sellist testimise viisi nimetatakse regressioontestimiseks.

2.1 Regressioontestimine

Regressioontestimist on kasulik kasutada nendes süsteemides, kus toimuvad koodi/süsteemi muudatused. Näiteks uue funktsionaalsuse lisamisel. Regressioontestimise eesmärgiks on olla kindel, et pärast muudatuste või veaparanduste tegemist ei ole funktsionaalsustesse tekkinud uusi vigu ning et kõik toimiks endiselt nõuete kohaselt [4] . Telias on Dealgate’s vaja regressioontestimist, sest pidevalt arendatakse välja uuemat süsteemi erinevate projektide raames. Sellega võivad kaasneda vead, millele tänu regressioonile saab ka kiirelt jälile. Tavaliselt algab regressioontestimine siis, kui esimene uuendus on tehtud ning lõppeb viimase versiooniga. Iga uuenduse tulemuseks on suurema hulga regressioontestide arv. Sellest tulenevalt suureneb ka teiste ressursside vajadus ning nende testide täitmise aeg [5] . Antud kirjeldust illustreerib järgmine joonis (vt Joonis 2).



Joonis 2. Regressioontestimise protsess[5] .

Regressioontestimist on kirjeldatud erinevate põhitüüpide järgi [6] :

- Vea regressioon – katse tõestada, et parandatud viga tegelikult ei ole parandatud.
- Vanade vigade regressioon – katse tõestada, et koodi või andmete viimane uuendus lõhkus vanade vigade parandust, nö vanad vead on uuesti tekitatavad.
- Kõrvalnähu regressioon – katse tõestada, et koodi või andmete viimane uuendus lõhkus teisi rakenduse osi.

Regressioontestimine on oma loomult üks parimaid võimalusi testimise automatiseerimiseks. See on ühetaoliste tegevuste kordamine mitmeid kordi ja just see omadus võimaldab protsessi automatiseerimist [7] . Kui tarkvara maht suureneb ning funktsionaalsused kasvavad, siis selle tõttu suureneb ka regressioontestimise testide hulk. Suurem testide hulk tähendab rohkem aega. Kui suur hulk teste tuleb läbi teha manuaalselt, siis suure ajakulu tõttu ei ole see enam otstarbekas. Seetõttu oleks kasulik regressioontestimine automatiseerida.

2.2 Võimalused

Regressioontestide tegemiseks on mitmeid erinevaid võimalusi:

- Täielik regressioontestimine (*The complete regression test*) – sellist võimalust on otstarbekas kasutada süsteemides, kus on riskid väga suured. Põhimõte on selles, et kui regressiooni käigus on välja tulnud viga, siis testid lähevad sellegipoolest lõpuni. Pärast vea parandamist ning uue versiooni tekkimist hakkab regressioon

uuesti peale ning kordab sama protseduuri uue versiooni peal [8] . Antud meetodi puhul on kasulik see, et süsteemis ilmnenud vead avastatakse võimalikult vara ning sellest tulenevalt on ka odav vigu parandada. Küll aga on miinuseks suur ajakulu. Sõltuvalt testide arvust võib regressioontestide lõpuni minek võtta mitmeid päevi.

- Jätkuv regressioontestimine (*The continuous regression test*) – kui programmi kohta väljastatakse uus versioon, siis antud meetodi puhul ei katkestata testimist. Näiteks kui vanal versioonil jäi regressioontestimine pooleli 10. testi juures ning väljastati uus versioon, siis testimist jätkatakse ka 10. testi juures uuel versioonil [8] . Antud võimaluse positiivseks küljeks oleks ajavõit, sest testimist ei alustata algusest. Meetodi negatiivseks küljeks on see, et süsteem ei saa täielikult läbi testitud, vaid testitakse etappide kaupa erinevaid versioone.

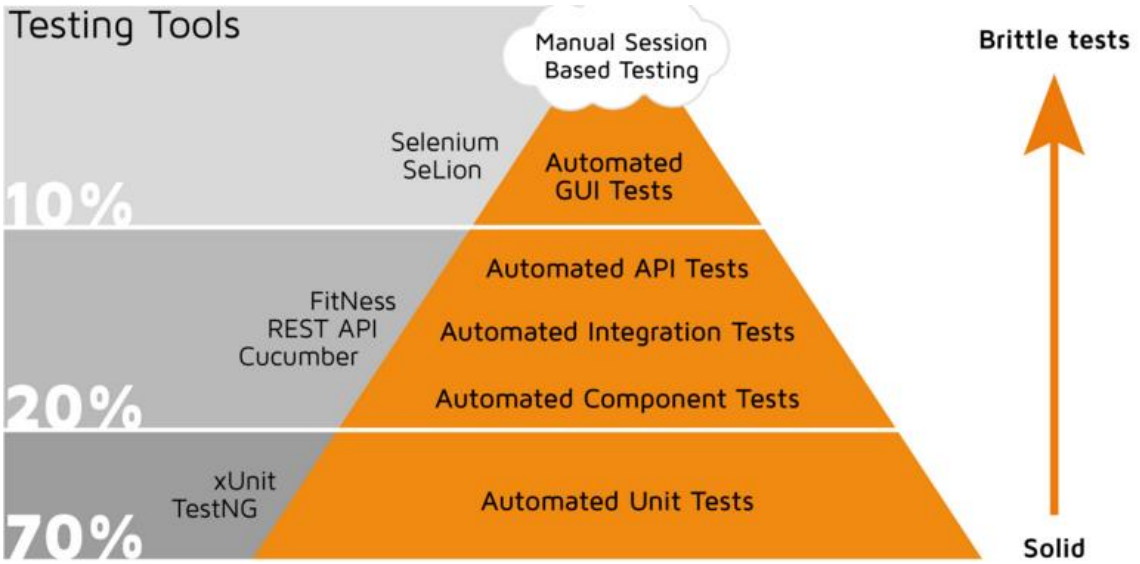
- Regressioon suitsutestimine (*The regression smoke test*) – eesmärgiks on katta versiooni kõik funktsionaalsused testidega, mis ei süvene sisusse. Kui on välja antud kõige värskem versioon, siis sellele tehakse täielik regressioontestimine, mis läbib kõik funktsionaalsused [8] . Selle võimaluse puhul on plussiks samuti aeg, kuid miinuseks võib pidada seda, et pinnapealse testimise käigus ei pruugi olulised vead välja tulla. Kui viga ilmneb alles viimases täielikus regressioontestimises, siis nende parandamiseks pole enam aega. Samuti ei ole kunagi teada, milline on viimane versioon, sest kui leitakse viga, siis vajab versioon uuesti täielikku regressioontestimist.

- Uuesti algav regressioontestimine (*The restartable regression test*) – kui täielik regressioontestimine läks igal juhul lõpuni, siis uuesti algav regressioontestimine alustab iga uue versiooni puhul uuesti. Seega kui ühes versioonis ilmneb viga, siis edasi testimist ei toimu seni, kuni on uus versioon ilmunud. Antud meetodi puhul on negatiivseks osaks see, et teatud osa teste võivad läbida alles viimaste versioonide juures ning vea ilmnemisel oleks parandamiseks vähe aega [8] .

Telia Eesti AS näitel on Dealgate’s hetkel kasutusel täielik regressioontestimine. Automaattestid töötavad öösel ning see annab eelise just sellele variandile.

3 Automaattestimine

Manuaalne süsteemi testimine on kindlasti rutiinne tegevus ning võimalus vigade otsimisel vigu teha on suur. Lisaks sellele kasvab iga uue funktsionaalsuse järel ka maht, mida tuleb testida. Seega oleks otstarbekas kasutusele võtta automaattestimine. Samas ei pruugi igas olukorras see aidata, sest päris kõike automatiseerida pole võimalik. Seega tuleks läbi mõelda, kui suures ulatuses seda teha. Alljärgnevalt on välja toodud joonis, kus on illustratiivselt testimise erinevad tasandid. Automaattestimise kolmnurgas on protsentuaalselt välja toodud, kui suures osas võiks süsteemi teatud tasandis testida (Joonis 3).



Joonis 3. Automaattestimise kolmnurk [9] .

Kõikidel tasanditel on oma eesmärk. Kolmnurgalt on näha, et kõige enam teste tuleb teha Unit-teste – nendega tegelevad enamasti arendajad ning eesmärgiks on saada teadmine, et rakenduse eraldiseisvad osad toimivad. Keskmise tasandi testimist nimetatakse integratsioonitestimiseks – eesmärgiks on saada teadmine, kas rakendus ka üldpildis toimib, kui eraldiseisvad osad on kokku võetud. Kolmnurga kõrgeima tasandi testimist nimetatakse GUI-testimiseks – eesmärk on kasutajaliidese testimine, mis näitab kas rakenduse funktsionaalsused toimivad.

3.1 Positiivsed küljed

Automaattestimise üheks oluliseks positiivseks küljeks on see, et ühte testi tuleb kirjutada vaid ühe korra. Samuti ka ühe testandmeid tuleb kirjutada ühe korra. Võrreldes automaattestimist näiteks manuaaltestimisega, siis manuaalsel testimisel on tähtis, et olulised testid saab üles kirjutada, sest siis saab neid vajadusel kiiresti korrata.

Samuti üheks positiivseks küljeks on see, et automaattestid on võimelised töötama ka siis, kui inimesi juures ei ole. Seega on võimalik automaattestimine käima panna ööseks ning hommikul saab inimtööjõud testide tulemused üle vaadata.

Automaattestimine on ajaliselt säästlikum. Arvuti ei vaja nii pikka mõtlemisaega kui inimene. Seega sel hetkel, kui testid automaatselt töötavad, siis töötajad saavad tegeleda muude oluliste asjadega ning kokkuvõtvalt hoiab see kõik aega kokku.

Lisaks eelmistele on kasulikuks ka see, et automaattestimist ei pea käivitama samas arvutis, kus tööd tehakse. Võimalik on teste käivitada ka teises serveris. See aitab kokku hoida käsitletava arvuti mahtu [10] .

On juhuseid, kus automatiseeritult on mõnda testi kordades lihtsam teha, kui manuaalselt. Samuti on manuaalselt üksikuid lühikesi teste lihtne ja kiire teha, kuid kui testide hulk kasvab ning versioonid täienevad, siis manuaalselt testimise maht kasvab samuti.

3.2 Negatiivsed küljed

Automaattestimise juures tuleb silmas pidada ka seda, et kõik ei ole siiski täielikult positiivne selle juures. Leidub nii mõnigi oluline negatiivne külg, mida peaks arvestama.

Enda kogemusest praktilise osa juures on automaattestimise üheks suurimaks miinuseks valehäired. See tähendab, et automaattestide tulemustest selgub, et programm on vigane, kuid tegelikult on vigased hoopis kas testandmed või rakenduse konfiguratsioon. Kogenumad testijad juba oskavad automatiseerida nii, et valehäireid tekiks võimalikult vähe. Lisaks testandmetele ja konfiguratsioonile võib valehäire põhjuseks olla ka vahend, milles automaattestid töötavad.

Üheks negatiivseks omaduseks võib veel tuua ka ajakulu automatiseerimise alguses. Kui on soov kasutusele võtta automaattestimine, siis see eeldab põhjalikku analüüsi ning

eeltööd. On olemas erinevaid raamistikke ning igal raamistikul on jällegi head ja halvad omadused. Seetõttu tuleks täpselt teada, millist projekti automatiseerida ning sellest tulenevalt valida vastavalt sobiv raamistik. See on vaid alguse aja investeering – hiljem, kui testid on automatiseeritud, on see jällegi positiivseks pooleks [10] .

Automaattestid on kasulikud just regressioontestide puhul. Uute vigade otsimise jaoks on parem siiski manuaalne testimisviis, sest inimene on suuteline loogiliselt mõtlema, arvuti paraku mitte. Arvuti teeb kõike nii, nagu on ette kirjutatud ning seetõttu on automaattestimise meetod kasulik valik just regressioontestide jaoks, kus on eesmärgiks saada teada, kas kõik toimib endiselt nii, nagu toimima peab.

Automaattestide miinuseks saab pidada ka seda, et ta vajab siiski ka inimest. Kui automaattest on leidnud vea, siis koostab ta selle kohta küll raporti, kuid see ei ole päris see, mille võiks arendajatele edasi saata. Inimene peaks vea siiski läbi vaatama ning tegema järelduse, kas tegu on siiski reaalse veaga, mitte valehäirega. Kui viga osutub reaalseks, tuleks täpsemalt viga raporteerida – milliste andmetega viga tekkis; mida tehes viga tekkis ning milline on vea tulemus. Sellise raporti puhul on arendajatel selgem arusaam ning parandamisele kulub tunduvalt vähem aega [10] .

3.3 Erinevad vahendid

3.3.1 Robot Framework

Robot Framework on raamistik automaattestimise jaoks. Samuti saab Robot Frameworki kasutada ka arenduse jaoks, mis põhineb vastuvõtu testimisel [11] . Raamistiku suureks plussiks on see, et tegu on vabavaralise tarkvaraga ning selle tõttu on see kasutusel suurel hulgal üle maailma. Lisaks on raamistikul veel mitmeid erinevaid kasulikke külgi:

1) Mugav kasutajaliides testide kirjutamiseks (RIDE).

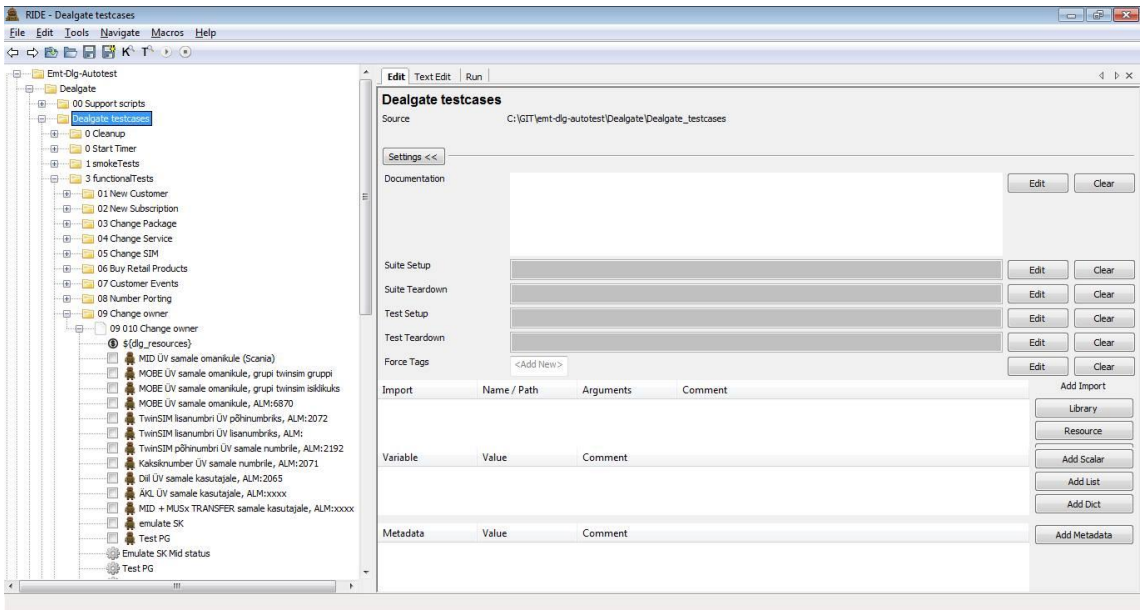
2) Detailsed raportid – lihtsalt loetavad raportid.

3) Java või Pythoni teekide kasutamise võimalus – võimalik laiendada testimise võimalusi.

4) Kergelt õpitav raamistik – koduleheküljel erinevaid selgitusi välja toodud.

RIDE kasutajaliidesel on samuti mitmeid olulisi positiivseid külgi võrreldes teiste raamistike kasutajaliidestega. Testide analüüsimiseks on RIDE keskkonnal üheks oluliseks plussiks see, et kui testid on lõpuni töötanud, siis koostatakse detailne ning kergesti loetav raport testide tulemuse kohta. Testide tulemused on formaadilt HTML kujul.

Automaattestide koostamisel on suureks positiivseks osaks see, et võtmesõnu on võimalik ise kirjeldada ning neid korduvalt kasutada. Võtmesõnad on teisiti öeldes funktsioonid, mida on võimalik erinevates testides välja kutsuda. See võimaldab ühe testiloo lühemalt kirja panna ning hoiab ka aega kokku. Seoses võtmesõnadega on heaks küljeks ka see, et testi käivitamise liideses on võimalik kontrollida, kas antud võtmesõna üldse eksisteerib. See aitab samuti aega kokku hoida, sest testi koostajal ei ole vaja spetsiaalselt kontrollima minna, kas võtmesõna on juba kirjeldatud.



Joonis 4. Dealgate RIDE vaade.

Joonisel 4 on pilt kasutajaliidesest, kus vasakul on hästi näha struktuur, kuidas RIDE-s on testid mugavalt ära jaotatud. Jaotada on neid võimalik (Joonisel 4 ülevalt alla) nii kataloogide, testide kogumite, parameetrite, testide ja võtmesõnade järgi.

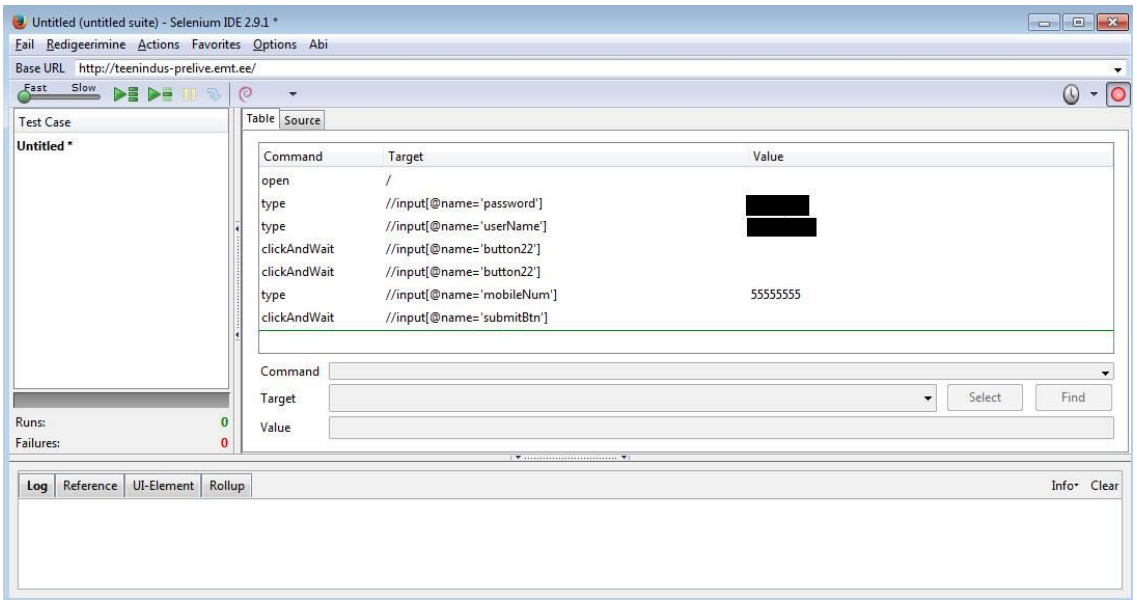
Testide kogumid võiks luua nii, et oleks loogiliselt üles ehitatud – näiteks sarnase funktsiooni või eesmärgiga testid. Enda kirjeldatud võtmesõnade kasutamine ei ole küll kohustuslik, kuid nende abil saab erinevaid testlugusid väiksema ajamahuga kirjutada ning testlugu on ka lühem. Küll aga tuleks silmas pidada, et muutuvat loogikat tuleks kasutada võimalikult palju just võtmesõnade defineerimisel, sest kui kuskil on vaja teha muudatusi, siis piisab vaid ühest muudatusest. Kui võtmesõnu ei ole kasutatud, tuleb muudatused teha igas testis, kus sama loogikat on kasutatud.

Jooniselt 4 on veel näha, et erinevate kataloogide ja testide kogumite juurde on võimalik lisada ka dokumentatsioon. See on hea selleks, et vajadusel kirjutada erinevate testkogumite juurde vajalikke märkmeid.

Suureks plussiks on ka veel, et erinevatele testidele on võimalik juurde lisada silt (*Tag*), mis võiks olla unikaalne. Selle alusel on hiljem hea erinevaid teste otsida ning samuti ka käivitada.

3.3.2 Selenium

Selenium IDE on kasutamiseks, et koostada ja käivitada lihtsamaid automaatteste. Selle kasutamise teeb mugavaks asjaolu, et teste on võimalik salvestada otse veebibrauserist. See tähendab, et inimene teeb erinevaid tegevusi otse veebibrauseris ning kõik tegevused salvestatakse. Hiljem on neid võimalik käivitada – võimalik on käivitada kogu test korraga või ainult teatud sammud. Suureks miinuseks on Seleniumile see, et kasutada saab seda ainult Firefox veebibrauseris [12] . Lisaks sellele on võimalik ka ise valida, kui kiiresti need läbitakse. Sellest on tingitud ka põhjus, miks Seleniumiga on võimalik koostada ja käivitada vaid lihtsamaid automaatteste. Küll aga on mugav seda vahendit kasutada näiteks eelmises punktis (peatükis 3.3.1) seletatud RIDE keskkonnas automaattestide kirjutamiseks. Põhjus on see, et Selenium salvestab veebibrauseris tehtud tegevused ning hiljem on neid võimalik ka vaadata ja selle kaudu saada vajalikku infot, mida kasutada RIDE keskkonnas.



Joonis 5. Selenium testi näide.

Joonisel 5 on toodud lihtne näide, kus on salvestatud veebribrauseris tegutsemised – logiti sisse Telia klienditeenindaja keskkonda ning sooritati otsing mobiilinumbriga 55555555. Jooniselt on ka näha, et soovi korral on võimalik antud testi muuta – selleks tuleb teatud väli, mida soovitakse muuta, teha aktiivseks ning seejärel „*Command*“ menüüst valida uusi käsklusi. See on testijale mugav, sest nii et ole vaja testijal kuskilt otsida käsklusi, mis vaja läheb. Lisaks on „*Find*“ nuppu kasutades näha, kus soovitud element veebibrauseris täpselt asub. Kui testija ei ole endas päris kindel, siis „*References*“ menüü all on näha ka valitud käsu kirjeldus.

Lisaks pakub Selenium IDE ka testide eksportimise võimalust. See on hea selleks, et on võimalik Seleniumi abiga lihtsam test ära teha ning see eksportida mõnes teises formaadis – C#, Java, Python, Ruby ja HTML. Peale nende formaatide on võimalik testijal ise endale sobiv formaat defineerida.

Enne sai mainitud, et Selenium IDE on loodud vaid lihtsamate automaattestide kirjutamiseks. Kui on vajadus Seleniumiga ka keerulisemaid teste koostada, siis selleks on olemas Selenium Webdriver. Webdriveriga on võimalik kirjutada veebibrauseris käivitatavaid automaatseid teste, mis on kasutajaliidese põhised [13] . Webdriveri plussiks on samuti see, et tegu on tasuta vahendiga. Erinevalt Selenium IDE-st, mis on mõeldud vaid Firefoxile, on Webdriver kasutatav erinevate veebibrauseritega – IE, Firefox, Opera, Safari ja Google Chrome [12] .

4 Erinevate parameetrite analüüs Telia Dealgate näitel

Antud peatükis on keskendutud Telia klienditeeninduse töökoha (Dealgate) testimise analüüsimisele erinevate parameetrite järgi. Erinevad parameetrid on järgmised:

- Testide koostamiseks kuluv aeg;
- Testide käimise aeg;
- Testide parandamiseks kuluv aeg;
- Tasuvus – ROI (return on investment) ehk investeeringutasuvus.

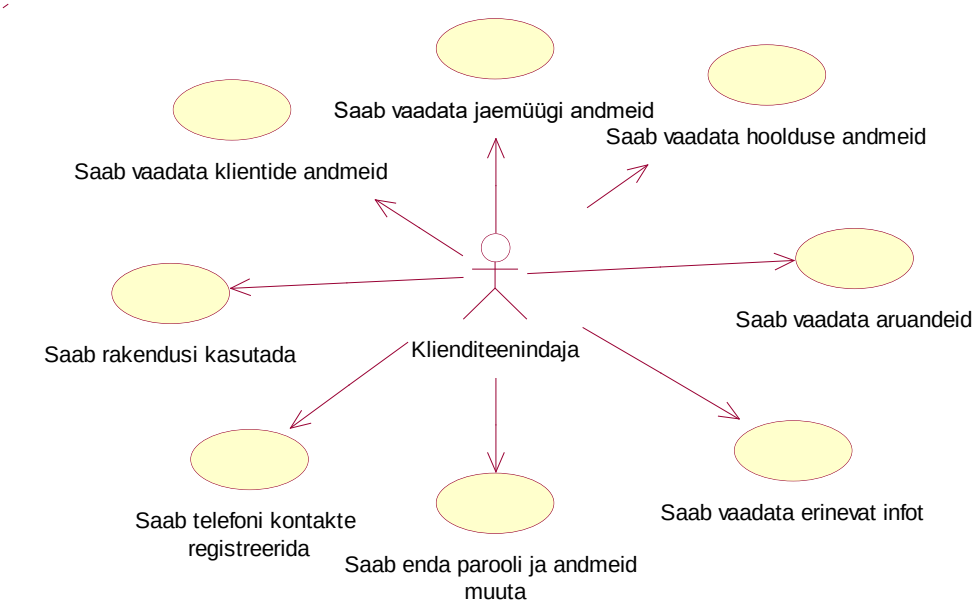
Lisaks nimetatud parameetritele võrdlen automaattestimist manuaaltestimisega.

4.1 Raamistike võrdlus ja analüüs

Telia Dealgate on mõeldud kasutamiseks klienditeenindajatele. Selle kaudu saavad klienditeenindajad oma tööd teha – vastavalt klientide soovile peab süsteem võimaldama uusi kliente registreerida, erinevaid teenuseid/pakette muuta, seadme remonti registreerima ja palju muud. Dealgate keskkonda arendatakse ja täiendatakse pidevalt uute versioonidega, seega on otstarbekas, et regressioontestide kaetavus oleks suur. Dealgate's on erinevate funktsionaalsuste arv niivõrd suur, et manuaalselt regressioontestimise peale kuluks umbes 2 nädalat. Praeguse seisuga on testide kaetavus vaid 45%, kuid eesmärgiks võiks olla pea 2 korda suurem kaetavus.

Joonis 6. Testitav Dealgate rakendus.

Jooniselt 6 on näha rakendus, mida on vaja testida. Antud juhul on punase kastiga välja toodud menüü, mille töökindlust on vaja kontrollida – iga menüü nupule vajutades peab ette tulema konkreetne pilt, mis peab võimaldama teha erinevaid funktsionaalsusi. Alljärgnevad testid on loodud selleks, et kontrollida, kas menüüle vajutades toimib süsteem nii, nagu on ette nähtud. Jooniselt 7 on näha kasutusjuhtude diagramm, kus on välja toodud, millised on funktsionaalsed nõuded süsteemile ning alljärgnevad testid kontrollivad seda.



Joonis 7. Kasutusjuhtude diagramm

Alljärgnevalt toon välja kasutusjuhtude kirjelduse:

Kasutusjuht: Klienditeenindaja saab vaadata klientide andmeid.

Tegutseja: Klienditeenindaja.

Eeltingimused: Klienditeenindaja on registreeritud ja on sisenenud süsteemi.

Järeltingimused: Avaneb leht, kus on näha klientide andmed.

Põhistsenaarium: Klienditeenindaja vajutab nupule „Kliendid“. Avaneb leht, kust on võimalik klientide andmeid vaadata.

Kasutusjuht: Klienditeenindaja saab vaadata jaemüügi andmeid.

Tegutseja: Klienditeenindaja.

Eeltingimused: Klienditeenindaja on registreeritud ja on sisenenud süsteemi.

Järeltingimused: Avaneb leht, kus on näha jaemüügi andmed.

Põhistsenaarium: Klienditeenindaja vajutab nupule „Jaemüük“. Avaneb leht, kust on võmalik jaemüügi andmeid vaadata.

Kasutusjuht: Klienditeenindaja saab vaadata hoolduse andmeid.

Tegutseja: Klienditeenindaja.

Eeltingimused: Klienditeenindaja on registreeritud ja on sisenenud süsteemi.

Järeltingimused: Avaneb leht, kus on näha hoolduse andmed.

Põhistsenaarium: Klienditeenindaja vajutab nupule „Hooldus“. Avaneb leht, kust on võmalik hoolduse andmeid vaadata.

Kasutusjuht: Klienditeenindaja saab vaadata aruandeid.

Tegutseja: Klienditeenindaja.

Eeltingimused: Klienditeenindaja on registreeritud ja on sisenenud süsteemi.

Järeltingimused: Avaneb leht, kus on näha erinevaid aruandeid.

Põhistsenaarium: Klienditeenindaja vajutab nupule „Aruanded“. Avaneb leht, kust on võmalik aruandeid vaadata.

Kasutusjuht: Klienditeenindaja saab vaadata erinevat infot.

Tegutseja: Klienditeenindaja.

Eeltingimused: Klienditeenindaja on registreeritud ja on sisenenud süsteemi.

Järeltingimused: Avaneb leht, kus on näha erinevaid kontakte ja andmed.

Põhistsenaarium: Klienditeenindaja vajutab nupule „Infoleht“. Avaneb leht, kust on võmalik vaadata erinevaid kontakte ja andmeid.

Kasutusjuht: Klienditeenindaja saab parooli ja andmeid muuta.

Tegutseja: Klienditeenindaja.

Eeltingimused: Klienditeenindaja on registreeritud ja on sisenenud süsteemi.

Järeltingimused: Avaneb leht, kus on võimalik muuta enda andmeid ja parooli.

Põhistsenaarium: Klienditeenindaja vajutab nupule „Seaded“. Avaneb leht, kus on võimalik muuta nii parooli kui ka enda andmeid.

Kasutusjuht: Klienditeenindaja saab telefoni kontakte registreerida.

Tegutseja: Klienditeenindaja.

Eeltingimused: Klienditeenindaja on registreeritud ja on sisenenud süsteemi.

Järeltingimused: Avaneb leht, kus on võimalik registreerida telefoni kontakte.

Põhistsenaarium: Klienditeenindaja vajutab nupule „Solidus“. Avaneb leht, kus on võimalik registreerida telefoni kontakte.

Kasutusjuht: Klienditeenindaja saab rakendusi kasutada.

Tegutseja: Klienditeenindaja.

Eeltingimused: Klienditeenindaja on registreeritud ja on sisenenud süsteemi.

Järeltingimused: Avaneb menüü, kus on võimalik valida erinevate rakenduste vahel.

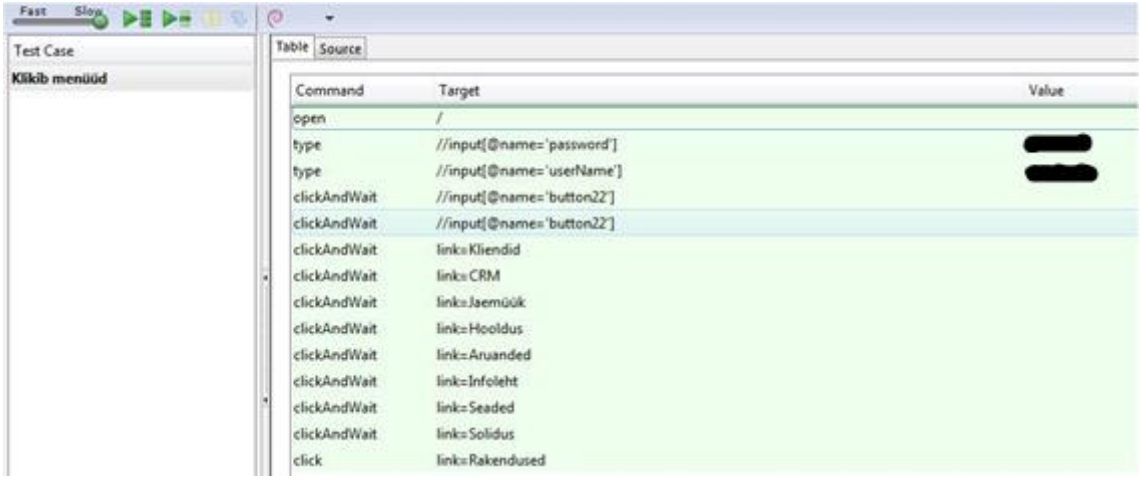
Põhistsenaarium: Klienditeenindaja vajutab nupule „Rakendused“. Avaneb menüü, kus on võimalik valida erinevate rakenduste vahel.

```
*** Settings ***
Documentation     Klikib läbi kõik Dealgate menüüd, kontrollib et ei tekiks tehnilist vms viga
...               TODO: Lisada tehnilise vea kontrol
...               TODO: Osadel lehtedel saab lisateste teha
Suite Setup       Set Selenium Timeout    25    # Longer timeout to wait menu items
Suite Teardown    Set Selenium Timeout    10    # Standard timeout
Test Setup        Test Case Setup
Test Teardown     Test Case Teardown
Test Timeout      10 minutes
Resource          ${dlg_resources}/dlg_common_resources.txt
Resource          ${dlg_resources}/dlg_login.txt
Resource          ${dlg_resources}/dlg_main_navigation.txt
Resource          ${dlg_resources}/environment_${ENVIRONMENT}_resources.txt
Resource          ${dlg_resources}/dlg_validation_sql.txt

*** Variables ***
${dlg_resources}    ../../Dealgate_resources

*** Test Cases ***
01. Põhimenüüde kontroll. ALM:2095
    [Tags]    Ready    Smoke    ALM:2095
    Log In    ${dealer}    ${password}    ${shop_id}
    Verify Landing Page
    Click Link    link=Kliendid
    Wait Until Page Contains    Kliendi otsing
    Click Link    link=CRM
    Wait Until Page Contains    Teisaldatav number:
    Click Link    link=Jaemüük
    Wait Until Page Contains    Kliendi otsing
    #
    Click Link    link=Hooldus
    # Peab olema tekst 'Töö nr' või 'Aktiivsed töökäsud'
    ${status}    ${value} =    Run Keyword And Ignore Error    Wait Until Page Contains    Töö nr
    Run Keyword If    '${status}' == 'FAIL'    Wait Until Page Contains    Aktiivsed töökäsud
    #
    Click Link    link=Äruanded
    Wait Until Page Contains    Statistika
    Click Link    link=Infoleht
    Wait Until Page Contains    Viimati lisatud informatsioon
    Click Link    link=Seaded
    Wait Until Page Contains    Minu salasõna muutmine
    Click Link    link=Solidus
```

Joonis 8. Dealgate lihtne RIDE automaattesti näide.

Jooniselt 8 on näha üks lihtne automaattesti näide Dealgate kohta, kus testi eesmärgiks on kontrollida, et midagi ei oleks juhtunud erinevate menüüdega. Test vajutab ette kirjutatud menüü lingile ning kontrollib, kas tekkinud leht sisaldab vajalikku infot. Antud test kasutab juba sisse defineeritud võtmesõnu. Ainukesed ise defineeritud võtmesõnad

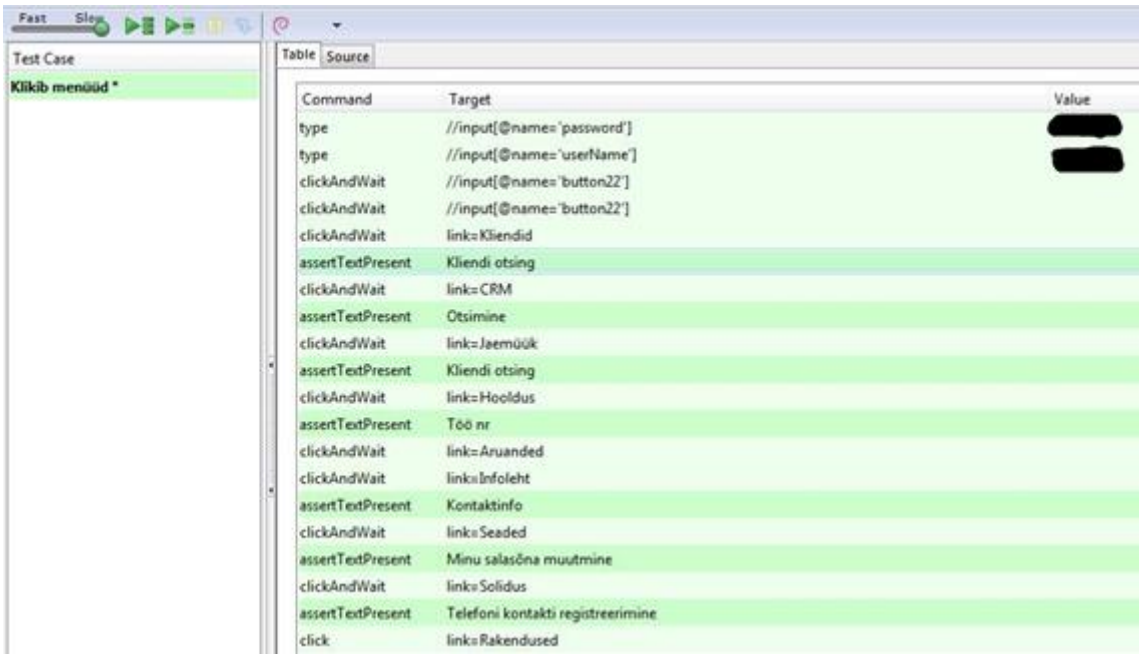
on *Log In* ja *Verify Landing Page*. Samuti ei vaja antud test ka andmebaasiga ühendust, seega läbitakse test kiirelt – 63 sekundiga, kuhu sisse on arvestatud ka Dealgate'i sisse logimine.



Joonis 9. Dealgate lihtne Selenium automaattesti näide.

Joonisel 9 on kujutatud testjuhtu, mis on realiseeritud Seleniumis. Eesmärgiks on sama, mis eelmise puhul – käib läbi erinevad menüüd ning kontrollib, et leht oleks endiselt olemas. Antud juhul on vahe selles, et Seleniumi näide ei sisalda kontrolle, nagu seda teeb RIDE näide – Seleniumi puhul vajutab lihtsalt antud lingile ning ei kontrolli, kas tekkinud leht sisaldab vajalikku informatsiooni. Kui analüüsida visuaalset pilti, siis Selenium paistab olevat kasutajasõbralikum kui RIDE. Antud juhul on piisavalt vähe teksti, et kõik vajalik oleks olemas ning on lihtsasti loetav.

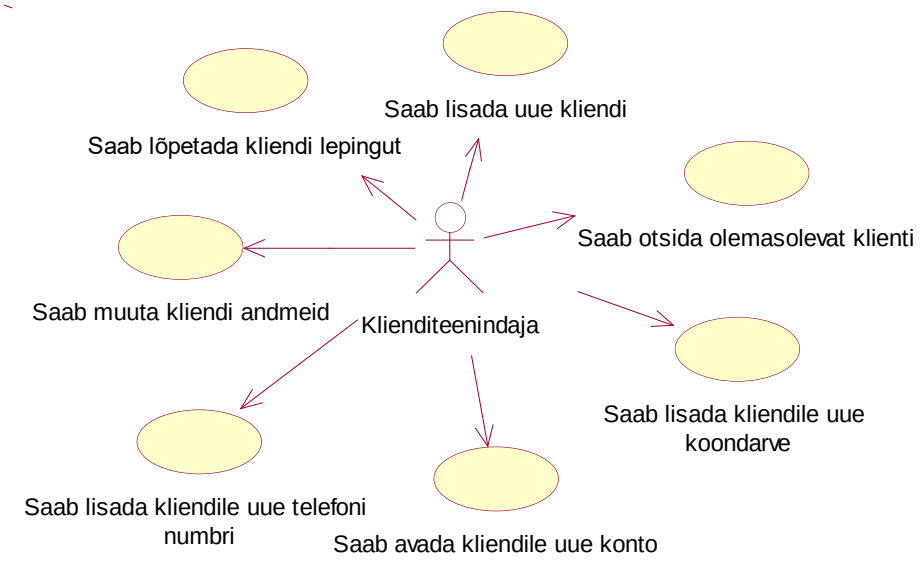
Analüüsides testide läbimise aega, siis Seleniumi puhul on seda jällegi raske teha, kuna kasutaja saab ise valida, kui kiirelt soovib, et test läbitakse. Joonisel 9 toodud näites on näha, et antud test on määratud läbima kõige aeglasema kiirusega – tulemuseks on 26 sekundit. Kui määrata läbimise ajaks kõige kiirem variant, siis jõuab test lõpuni 9 sekundiga. Ajaliselt on see küll kiire, kuid samas on selleks vaja piisavalt head internetiühendust. Seega võib probleemseks osutuda see, et liigse kiiruse tõttu võib Selenium hakata andma valehäireid. Antud hetkel on raske neid kahte ajaliselt analüüsida, sest Seleniumil on konkreetses testis vähem kontrolle ning selle tõttu läbib test kiiremini.



Joonis 10. Dealgate Selenium kontrolliga näide.

Joonisel 10 on toodud näide testjuhust, mis sisaldab ka kontrolle. Seega test on efektiivsem ning on väiksem tõenäosus, et soovitud lehekülg ei sisalda infot, mida ta peab sisaldama. Küll aga mida efektiivsem on test, seda rohkem nõuab ta läbimiseks aega. Antud juhul (Joonisel 10) läbitakse test 36 sekundiga, kui valida aeglasem läbimise kiirus. Eelmine test (vt. Joonis 9), mis kontrolle ei sisaldanud, vajas lõpetamiseks 26 sekundit. Antud testi on võimalik luua ka selliselt, mis sisaldab rohkem kontrolle, kuid sel juhul ei ole enam efektiivsuse ning ajalise võidu suhe paigas. Tuleks luua selliseid teste, mis annavad piisavalt kindla tõenäosuse, et süsteem teeb seda, mida on nõutud.

Jooniselt 11 on näha kasutusjuhtude diagramm funktsionaalsetele nõuetele, kus on vajalik ka andmebaasiga ühendus. Klienditeenindaja peab võimaldama süsteemis tegeleda kliendi andmetega – vajadusel neid muuta või lisada.



Joonis 11. Kasutusjuhtude diagramm andmebaasi kasutusega.

Järgnevalt kirjeldan kasutusjuhud täpsemalt lahti:

Kasutusjuht: Klienditeenindaja saab lisada uue kliendi.

Tegutseja: Klienditeenindaja.

Eeltingimused: Klienditeenindaja on registreeritud ja on sisenenud süsteemi. Kliendi andmed on teada.

Järeltingimused: Kliendi andmed on salvestatud andmebaasi.

Põhistsenaarium: Klienditeenindaja avab vormi, kus on võimalik sisestada uue kliendi andmed. Pärast vajalike vormide täitmist, lisatakse kliendi andmed andmebaasid ning uus klient on registreeritud.

Kasutusjuht: Klienditeenindaja saab otsida olemasolevat klienti.

Tegutseja: Klienditeenindaja.

Eeltingimused: Klienditeenindaja on registreeritud ja on sisenenud süsteemi. Kliendi andmed on teada ning andmebaasis olemas.

Järeltingimused: Olemasolev klient on leitud.

Põhistsenaarium: Klienditeenindaja avab vormi, kus on võimalik otsida andmebaasi registreeritud kliente. Pärast vajalike lahtrite täitmist otsib süsteem andmebaasist soovitud kliendi.

Kasutusjuht: Klienditeenindaja saab lisada kliendile uue koondarve.

Tegutseja: Klienditeenindaja.

Eeltingimused: Klienditeenindaja on registreeritud ja on sisenenud süsteemi. Kliendi andmed on teada ning andmebaasi registreeritud.

Järeltingimused: Kliendi uus koondarve on lisatud andmebaasi.

Põhistsenaarium: Klienditeenindaja vajutab nupule „Uue konto avamine“. Avaneb leht, kus on võimalik täita vajalikud kliendi andmed. Kui andmed on täidetud, lisatakse uus kliendi koondarve andmebaasi ning uus koondarve on lisatud.

Kasutusjuht: Klienditeenindaja saab lisada kliendile uue konto.

Tegutseja: Klienditeenindaja.

Eeltingimused: Klienditeenindaja on registreeritud ja on sisenenud süsteemi. Kliendi andmed on teada ning andmebaasi registreeritud.

Järeltingimused: Kliendi uus konto on lisatud andmebaasi.

Põhistsenaarium: Klienditeenindaja vajutab nupule „Kliendile uue konto avamine“. Avaneb leht, kus on võimalik registreerida kliendile uus konto. Pärast andmete lisamist ning salvestamist on kliendile uus konto registreeritud.

Kasutusjuht: Klienditeenindaja saab lisada kliendile uue telefoni numbri.

Tegutseja: Klienditeenindaja.

Eeltingimused: Klienditeenindaja on registreeritud ja on sisenenud süsteemi. Kliendi andmed on teada ning registreeritud andmebaasi.

Järeltingimused: Kliendi uus telefoni number on lisatud andmebaasi.

Põhistsenaarium: Klienditeenindaja vajutab nupule „Lisa mobiili number“. Avaneb leht, kus on võimalik registreerida uusi telefoni numbreid. Pärast vajalike andmete lisamist, registreeritakse uus telefoni number andmebaasi ning kliendi uus telefoni number on lisatud.

Kasutusjuht: Klienditeenindaja saab muuta kliendi andmeid.

Tegutseja: Klienditeenindaja.

Eeltingimused: Klienditeenindaja on registreeritud ja on sisenenud süsteemi. Kliendi andmed on teada ning registreeritud andmebaasi.

Järeltingimused: Muudetud kliendi andmed on andmebaasi registreeritud.

Põhistsenaarium: Klienditeenindaja vajutab nupule „Muuda kliendi andmeid“. Avaneb leht, kus on võimalik klientide andmeid muuta. Pärast vajalike muudatuste tegemist registreeritakse andmed andmebaasi ning kliendi andmed on muudetud.

Kasutusjuht: Klienditeenindaja saab lõpetada kliendi lepingut.

Tegutseja: Klienditeenindaja.

Eeltingimused: Klienditeenindaja on registreeritud ja on sisenenud süsteemi. Kliendi andmed on teada ning andmebaasi registreeritud.

Järeltingimused: Andmebaasi on lisatud märge, et leping on lõpetatud.

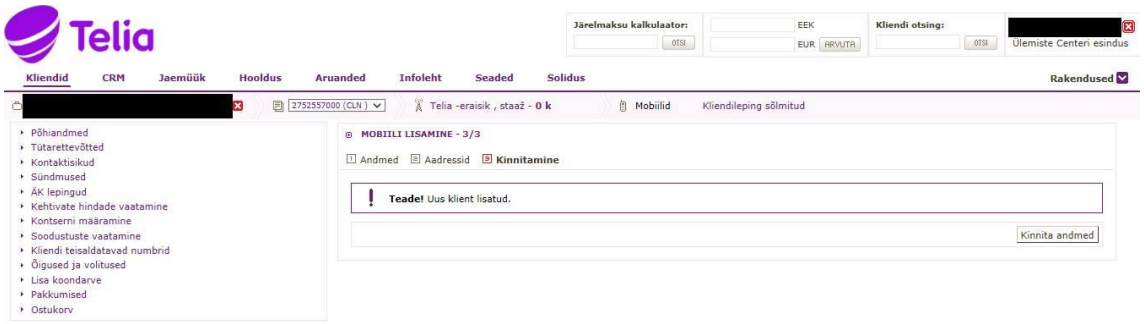
Põhistsenaarium: Klienditeenindaja vajutab nupule „Muuda kliendi andmeid“. Avaneb leht, kus on võimalik lõpetada kliendi lepingut. Pärast „Lõpeta kliendi leping“ kinnitamist lisatakse andmebaasi märge ning kliendi leping on lõpetatud.

Joonisel 12 kujutatud testid kontrollivad eelnevaid kasutusjuhte ning realiseeritud on need järgnevalt. Joonisel 12 on näide testi kohta, mis suhtleb andmebaasiga. Selle testi eesmärgiks on kontrollida, kas uue koondarve loomine õnnestub probleemideta.

```
*** Settings ***
Documentation     New Agreement
Test Setup        Test Case Setup
Test Teardown     Test Case Teardown
Force Tags        NEWMAAC
Resource          ${dlg_resources}/dlg_common_resources.txt
Resource          ${dlg_resources}/dlg_login.txt
Resource          ${dlg_resources}/dlg_main_navigation.txt
Resource          ${dlg_resources}/01_010_resources.txt
Resource          ${dlg_resources}/environment_${ENVIRONMENT}_resources.txt
Resource          ${dlg_resources}/dlg_validation_sql.txt
Resource          ${dlg_resources}/02_010_resources.txt    #for new subs
Resource          ${dlg_resources}/03_020_resources.txt    # MeieEMT slaiderid
Library           OperatingSystem

*** Variables ***
${dlg_resources}    ../../../Dealgate_resources

*** Test Cases ***
New MAAC for closed account, ALM:1981
    [Documentation]    Loob koondarve kliendile kelle kõik koondarved on suletud
    [Tags]    Ready    ALM:1981
    Log    Suletud koondarve otsimine..
    Connect To Oracle Database    ${DB_USERNAME}    ${DB_PASSWORD}    ${DB_DSN}
    @{query_results}    Query From String    select acs.acco_ref_num from account_statuses acs, accounts ac where acs.acst_
    Disconnect From Oracle Database
    ${maac_ref_num}    Convert Sim Number    ${query_results[0][0]}
    Log In    ${dealer}    ${password}    ${shop_id}
    Verify Landing Page
    Search By MAAC Number    ${maac_ref_num}
    #Validate MAAC Search Landing Page
    #Click Link Add New Phone Number
    Wait Until Page Contains Element    link=Lisa mobiili number    60
    Click Link    link=Lisa mobiili number
    Wait Until Page Contains    Kliendi leping on lõpetatud    60
    Wait Until Page Contains Element    link=Kliendile uue konto avamine    60
    Click Link    link=Kliendile uue konto avamine
    Choose Virtual Operator And Continue    # EMT
    Wait Until Page Contains    Kliendi andmeid ei kuvata    60
    Click Button    css=td.tbl-actions > input[name="submitBtn"]
    Wait Until Page Contains    Lisa aadress    60
    Click Element    //input[@name='specialRefNum']
```

Joonis 12. Dealgate RIDE andmebaasiga näide.

Joonisel 13 on kujutatud antud testi tulemus. Klienditeenindajale kuvatakse ette teade, mis kinnitab, et uue kliendi lisamine õnnestus. See tähendab samal ajal ka seda, et uus klient lisati ka andmebaasi.



Joonis 13. Uue kliendi lisamise tulemus.

Antud test sisaldab juba ka testija enda poolt defineeritud võtmesõnu. Võrreldes eelmise näitega, on testide ridade arv peaaegu sama, kuid viimane näide vajab aega umbes 123 sekundit, mille hulgas on samuti ka Dealgate'i sisse logimine. Pea kahekordne ajaline vahe tuleneb andmebaasiga suhtlemisest – viimane test kasutab ka andmebaasi, mis vajab aega. Seega oleks otstarbekas teha selliseid päringuid, mille täitmise aeg on võimalikult lühike.

Kui võrrelda nende kahe testi jõudlust, siis esimese näite puhul nõuab see mälu mahtu 17 000K, teise puhul 53 000K. Mõlema testi käivitamise hetkel võtab see enda alla 21% kõvakettast. Analüüsi tulemustest võib järeldada, et andmebaasidega koostöö nõuab suuri mahte. Seleniumi puhul nõuab see test mälu mahtu 33 000K. Kuna tegu on testimisvahendiga, mis nõuab töötamiseks Firefox brauserit, siis selle tõttu on ka mälu maht suurem.

Võrdluseks saab veel tuua ka automaattestimisvahendi kasutamismugavuse. Selleks, et Robot Frameworki kasutada, ei ole vaja programmeerimisoskust. Erinevad teadmised IT valdkonnast tulevad kasuks, kuid erinevaid õpetusi on palju ning dokumentatsioon on avalik. Seega on Robot Framework kergesti õpitav vahend. Lisaks on üks-üheselt aru saada, mida mingi test tegema peab, sest võtmesõnade abil on võimalik selgelt väljendada, mida konkreetne võtmesõna teeb ning seega on ka kogu testjuhu loetavus parem.

Lisaks nendele parameetritele annab Robotile veel eelise ka suhtlemise võimalus andmebaasiga. Andmebaasiga suhtlemine on oluline, et testimine oleks töökindlam ning

vajaks vähem inimese abi. Näitena saab tuua testjuhu, mis vajab konkreetset telefoni numbrit. Kui testile sisse kirjutada konkreetne telefoni number, siis võib tulevikus tekkida olukord, et antud number on kustutatud. Selleks, et sellist olukorda vältida, tuleks automaattestid andmebaasiga suhtlema panna – iga testi korral küsitakse andmebaasist uus number, mis vastab soovitud kriteeriumitele ning selle tulemusena ei teki tulevikus probleeme, et number on kustutatud. Joonisel 14 on näide, kuidas toimub Roboti suhtlus Oracle andmebaasiga. Sellist suhtlust Selenium IDE puhul luua ei saa. Siit tuleneb ka järjekordne Robot Framework eelis Selenium IDE ees.

```
New MAAC for closed account, ALM:1981
    [Documentation]    Loob koondarve kliendile kelle kõik koondarved on suletud
    [Tags]    Ready    ALM:1981
    Log    Suletud koondarve otsimine...
    Connect To Oracle Database    ${DB_USERNAME}    ${DB_PASSWORD}    ${DB_DSN}
    @{query_results}    Query From String    select acs.acco_ref_num from account_statuses acs, accounts ac
    where acs.acst_code = 'CLN' and ac.ref_num = acs.acco_ref_num and ac.bicy_cycle_code in ('MN2')
    and (select count(ref_num) from accounts where part_ref_num = ac.part_ref_num) = 1
    order by acs.acco_ref_num desc
    Disconnect From Oracle Database
```

Joonis 14. Roboti suhtlus andmebaasiga.

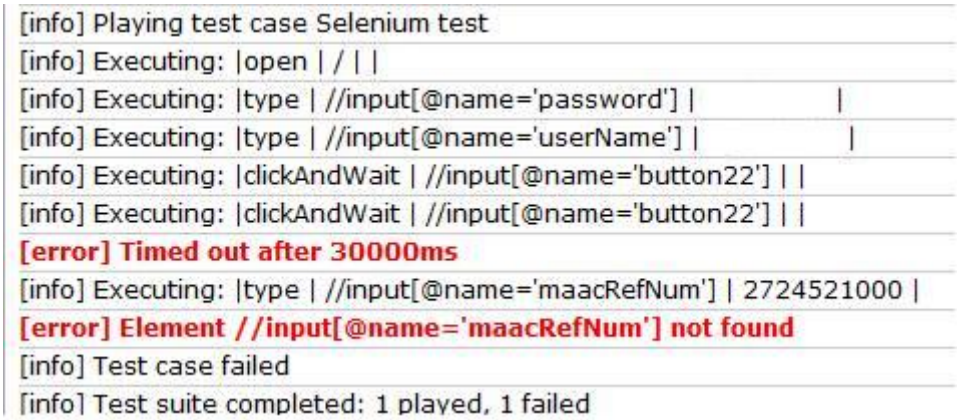
Roboti puhul üheks oluliseks funktsionaalsuseks on põhjalikud logid iga läbi käinud testi kohta. See annab eelise testijatele, kes saavad logide olemasolul kiiresti ja kergelt aru, miks test ebaõnnestus. Lisaks tavalisele logile salvestab Robot ka pildi sel hetkel, kui test ebaõnnestus. Ka selle abil on hea saada ülevaade, kas tegu on reaalse veaga või valehäirega.

```
KEYWORD: ${group_id} = dlg_common_resources.Get MultiSIM group nr for MAAC ${MAAC_nr}
KEYWORD: Selenium2Library.Execute Javascript redirect('','SharedDataGroupsComponent','edit','${group_id}');
KEYWORD: Selenium2Library.Wait Until Page Contains Element name=newContract, 60
KEYWORD: dlg_common_resources.Vajuta Nuppu Uus liitumine
KEYWORD: Selenium2Library.Wait Until Page Contains Element name=nextNewmobStep, 60
KEYWORD: dlg_common_resources.Vajuta Nuppu Komplektiga liitumine
KEYWORD: Selenium2Library.Wait Until Page Contains Kasutaja andmed, 60
KEYWORD: Selenium2Library.Select Checkbox name=isClientPersonally
KEYWORD: dlg_common_resources.Vajuta Nuppu Salvesta
KEYWORD: BuiltIn.Wait Until Keyword Succeeds 60 sec, 10 sec, JMLIIDES Get Phone
  Documentation:  Waits until the specified keyword succeeds or the given timeout expires.
  Start / End / Elapsed:  20160430 12:44:05.480 / 20160430 12:44:05.696 / 00:00:00.216
  KEYWORD: query_JMLIIDES_resources.JMLIIDES Get Phone
    Start / End / Elapsed:  20160430 12:44:05.481 / 20160430 12:44:05.696 / 00:00:00.215
    KEYWORD: OracleDB_Library.Connect To Oracle Database ${DB_CCM_JM_USERNAME}, ${DB_CCM_JM_PASSWORD}, ${DB_CCM_JM_DSN}
    KEYWORD: @{query_results} = OracleDB_Library.Query From File With Argument ${resource_path}/Jaemyyk_queries/jmliides_get_phone.txt, ${loca_code}
    KEYWORD: OracleDB_Library.Disconnect From Oracle Database
    KEYWORD: BuiltIn.Set Suite Variable ${kronos_results}, @{query_results}[0]
      Documentation:  Makes a variable available everywhere within the scope of the current suite.
      Start / End / Elapsed:  20160430 12:44:05.689 / 20160430 12:44:05.696 / 00:00:00.007
      12:44:05.696  FAIL  Variable '@{query_results}[0]' not found. Did you mean:
          @{query_results}
TEARDOWN: dlg_common_resources.Test Case Teardown
```

Joonis 15. Põhjaliku logi näide.

Jooniselt 15 on näha, millisel kujul Robot logi salvestab. Logi kuvatakse ühe testjuhu kohta võtmesõnade kaupa – vajadusel on iga tegevuse sisse võimalik ka täpsemalt näha. Rohelisega kuvatud *Keyword* tähendab ühe testjuhu puhul õnnestunud käsku, punasega kuvatud tähendab ebaõnnestunud käsku. Selenium IDE sellisel kujul logi ei kuva. Seal

on näha vaid seda, kas ühe testjuhu kohta teatud etapp õnnestus või ebaõnnestus. Kuvatakse välja kõik tegevused , kuid seda struktureerimata kujul (vt Joonis 16).



Joonis 16. Selenium IDE logi näide.

Jooniselt 16 on näha, et infot ühe testi kohta kuvatakse vähe ning seega on testijal vaja rohkem aega, et välja selgitada, milles täpsemalt viga seisneb.

Tabelis 1 on välja toodud koondtabel raamistike võrdluse kohta, kus on näha lühidalt ja konkreetselt analüüsi tulemused.

Tabel 1. Raamistike võrdluse koondtabel.

	Robot Framework	**Selenium IDE**
Töökindlus	+	
Kasutajasõbralik	+	
Salvestamise võimalus		+
Kõvaketta vähene koormus	+	
Andmebaasiga sidumise võimalus	+	
Põhjaliku logi kuvamine	+	
Testi läbimise kiirus		+

4.2 ROI – investeeringutasuvus

Investeeringutasuvuse arvutamisel on mitmeid erinevaid võimalusi. Kui jagada kasu projekti investeeringutega, siis tulemuseks on investeeringutasuvus. Ehk siis valemina kirja pannes oleks üks ROI arvutusviis järgmine [14] :

$$ROI = \frac{t-b}{b}, \quad (1)$$

kus ROI – investeeringutasuvus;

t – tulu;

b – müüdud kaupade maksumus.

Lisaks sellele, et investeeringutasuvuse kaudu saab teada, kui suur tulu on millegi kaudu saadud, on ROI abil ka hea selgeks teha, kuidas oma eelarvet kulutada. Eelolevat valemit saab kasutada ka võrdlemaks automaattestimist manuaaltestidega. Selleks tuleb lahutada investeering automatiseerimisse manuaalsest testimise ajalisest kulust ning tulemus jagada investeeringuga automatiseerimisse. Ehk valemina oleks see järgmine [14] :

$$ROI = \frac{man\_test\_aeg - aut\_inv}{aut\_inv}, \quad (2)$$

kus man_test_aeg – manuaalse testimise aeg;

aut_inv – automatiseerimise investeering.

Antud töös kasutan sellist varianti, kus ei arvestata sisse raha (töötasud, projekti kulud jne). Seega tulevad selle puhul välja rohkem ajalised kasud. Valem, millele tuginema hakkan, on järgmine:

$$\boldsymbol{ROI = \frac{k-inv}{inv}}, \quad (3)$$

kus k – kasu;

inv – investeering.

Kasu on antud valemis lahti kirjutatud järgmiselt:

$$k = \frac{kogu\_test\_arv * inv\_per * man\_test\_aeg}{tp}, \quad (4)$$

kus k – kasu;

kogu_test_arv – kõikide manuaalsete ja automaatsete testide arv kokku;

inv_per – investeeringu periood;

man_test_aeg – manuaalselt testimiseks vajaminev aeg;

tp – testija tööpäeva pikkus.

Investeeringu leidmiseks tuleb arvutada vahetulemused ning need omavahel kokku liita:

$$aut\_aeg = \frac{kogu\_aeg * kogu\_test\_arv}{tp}, \quad (5)$$

kus aut_aeg – automaatsete testide kirjutamiseks vaja läinud aeg;

kogu_aeg – keskmine automaatse testi kirjutamiseks vaja läinud aeg;

kogu_test_arv – kõikide manuaalsete ja automaatsete testide arv kokku;

tp – testija tööpäeva pikkus.

$$anal\_aeg = \frac{anal\_aeg * inv\_per}{tp}, \quad (6)$$

kus anal_aeg – analüüsimisele kulunud aeg;

inv_per – investeerimise periood;

tp – testija tööpäeva pikkus.

$$aut\_töö\_aeg = \frac{ühe\_aeg * kogu\_test\_arv * inv\_per}{olet\_aeg}, \quad (7)$$

kus aut_töö_aeg – automaatsete testide töötamiseks vajaminev aeg;

ühe_aeg – ühe keskmise testi töötamiseks vajaminev aeg;

kogu_test_arv – kõikide manuaalsete ja automaatsete testide arv kokku;

inv_per – investeerimise periood;

olet_aeg – automatiseeritud testide oletatav tööaeg tundides.

$$kogu\_hool\_aeg = \frac{aut\_hool\_aeg * inv\_per}{tp}, \quad (8)$$

kus kogu_hool_aeg – testide hoolduseks vajaminev aeg;

aut_hool_aeg – automaatsete testide hooldamiseks vajaminev aeg;

inv_per – investeerimise periood;

tp – testija tööpäeva pikkus tundides.

$$kogu\_man\_aeg = \frac{ühe\_man\_aeg * man\_test\_arv * inv\_per}{tp}, \quad (9)$$

kus kogu_man_aeg – manuaalsete testide tegemiseks vajaminev aeg;

ühe_man_aeg – ühe testi manuaalselt testimiseks vajaminev aeg;

man_test_arv – manuaalsete testide arv kokku;

inv_per – investeerimise periood;

tp – testija tööpäeva pikkus.

Liites kokku valemite (5) – (9) tulemused, saame teada valemi (3) jaoks vajamineva muutuja inv (investeeringu):

inv = aut_aeg + anal_aeg + aut_töö_aeg + kogu_hool_aeg + kogu_man_aeg *(10)*

4.3 Arvutused Dealgate näitel

Aastal 2012 oli Telia Dealgate veel täielikult manuaalselt testitav. Kuna nüüd on juba mõned aastad kasutusel olnud automatiseeritud testimine, siis on hea analüüsida, kas

automatiseerimine on ennast ära tasunud. Selleks analüüsiks on otstarbekas kasutada eelmises peatükis välja toodud investeeringutasuvuse meetodit. Antud juhul ei paku huvi rahaline pool vaid just ajaline – kas automatiseerimine oli kasulik ning kas praegune olukord on jätkusuutlik.

Analüüsi tegemiseks on teada andmed:

- Dealgate eeldatav kasutusperiood (tinglik) – 2 aastat = 104 nädalat
- Testija tööpäeva pikkus – 8 tundi
- Automaatsete testide töötamise aeg - ~10 tundi
- Keskmine automaatse testi kirjutamiseks vaja läinud aeg – 5 tundi
- Kõikide testide arv kokku – 90 testi
- Testide analüüsimusele kulunud aeg – 3 tundi nädalas
- Ühe keskmise automaatse testi töötamiseks vajaminev aeg – 4,29 min = 0,07 tundi
- Keskmiselt ühe testi manuaalseks testimiseks vajaminev ajaline kulu – 0,3 töötundi
- Manuaalsete testide arv - 20

Kasutades eelmises peatükis välja toodud valemeid (5) – (9), saame vahetulemusteks järgmised tulemused:

1) Automaatsete testide kirjutamiseks vajaminev aeg:

$$\text{aut\_aeg} = \frac{5 * 90}{8} = 56{,}25 \text{ (päeva)}$$

2) Analüüsimisele kulunud aeg:

$$\text{anal\_aeg} = \frac{3 * 104}{8} = 39 \ (päeva)$$

3) Automaatsete testide töötamiseks vajaminev aeg:

$$\text{aut\_töö\_aeg} = \frac{0{,}07 * 90 * 104}{10} = 65{,}52 \text{ (päeva)}$$

4) Olemasolevate testide hooldamise sidusin kokku analüüsimisega, sest kui test leiab vea, tuleb esmalt analüüsida, kas tegu on reaalse veaga või on töövahend tekitanud valehäire. Kui osutub valehäireks, parandame testi juba analüüsi käigus paremaks, et valehäiret enam ei annaks.

5) Manuaalsete testide tegemiseks vajaminev aeg:

$$\text{kogu\_man\_aeg} = \frac{0{,}30 * 20 * 104}{8} = 78(päeva)$$

Vastavalt valemile (4), saame välja arvutada kasu:

$$k = \frac{0{,}30 * 90 * 104}{8} = 351\ (päeva)$$

Vastavalt valemile (10), saame välja arvutada investeeringu:

inv = 56,25 + 39 + 65,52 + 78 = 238,77 (päeva)

Vastavalt valemile (3) leian investeeringutasuvuse:

$$\mathbf{ROI} = \frac{\mathbf{351 - 238,77}}{\mathbf{238,77}} = \mathbf{0,47}$$

Kui see arvestada protsentidesse, saame: 0,47 * 100% = **47%**

Antud tulemusest selgub, et kui Dealgate eeldatav eluiga oleks 2 aastat, siis oleks meie võit 47%. Seega ajaline kasu on pea poole suurem, kui manuaalselt testides. Tõenäoliselt on Dealgate eluiga veelgi pikem, seega tulemusest järeldub, et praegune süsteem on jätkusuutlik.

5 Kokkuvõte

Antud töö eesmärgiks oli erinevate parameetrite järgi võrrelda ja analüüsida Telia Eesti AS Dealgate praeguseid Robot Framework baasil kirjutatud automaatteste. Lisaks sellele oli eesmärgiks investeeringutasuvuse põhjal välja selgitada, kas praegune olukord on jätkusuutlik.

Analüüsi tulemusena selgus, et hetkel kasutusel oleval Robot Framework'l on mitmeid eeliseid Selenium IDE raamistiku ees:

- Töökindlam
- Kasutajasõbralikum
- Koormab vähem kõvaketast
- Andmebaasiga sidumise võimalus
- Põhjalik ja sisukas logi

Investeeringutasuvust rakendasin Telia Eesti AS Dealgate automatiseeritud regressioontestide kohta ning tulemusest järeldub, et automatiseerimine on olnud tulus. Antud analüüsil valisin ajavahemikuks 2 aastat ning automatiseerimise tulemusena kulutame 238 päeva testimiseks. Kui automatiseerimist ei oleks kasutanud, siis läheks 351 päeva manuaalseks testimiseks, et sama hulk testjuhte läbi testida. Seega protsentuaalselt on ettevõtte ajaliseks võiduks tänu automatiseerimisele 47%.

Antud töö kohta võib teha järeldusi, et võrreldes Robot Framework Selenium IDE raamistikuga, siis Robotil on rohkem kasulikke omadusi. Samuti võib järeldada, et Telia Eesti AS ettevõttes on Dealgate keskkonna testide automatiseerimine olnud efektiivne. Kuigi sõltuvalt projektist ja ajalisest arvestusest võib tulemus olla praegusega erinev, siis pikemas perspektiivis tasub testide automatiseerimine ennast ära. Kui oleksin töös kasutanud pikemat ajavahemikku kui 2 aastat, oleks tulemus olnud veelgi efektiivsema protsendiga.

Antud projekti analüüsi oleks võimalik veelgi täpsemini teha. Hetkel on kasutusel mitmed testid, mis on vananenud või mida on võimalik kiiremini läbima ehitada. Üheks võimaluseks oleks kirjutada efektiisemad andmebaasipäringud, mis võimaldavad kiiremini tulemusi tagastada. Teiseks võimaluseks oleks testid üles ehitada selliselt, et nende läbimine võtaks vähem aega ning samas oleksid ka töökindlamad. Pärast taolisi muudatusi tuleks teha sarnane analüüs (võrrelda uut süsteemi vanaga) ning teha ka uus investeeringutasuvus, et selgitada välja, kui palju muutus süsteem efektiivsemaks.

Summary

The aim of this study was to analyze and compare the different parameters of the current Telia Eesti AS Dealgate Robot Framework based automated tests. Moreover, the objective was to identify the return on investment on the basis of whether the current situation is sustainable.

The analysis showed that the Robot Framework has a number of advantages over the Selenium IDE framework:

- More reliable
- More user-friendly
- Strains the hard drive less
- Ability to connect to a database
- A thorough and comprehensive log

Return on investment was applied to the Telia Eesti AS Dealgate automated regression tests and the results show that the automation has been profitable. In the given analysis, I chose a period of 2 years and as a result of automation 238 days will be spent for testing. If automation wasn't used, it would take 351 days to manually test the same number of test cases. Thus, making the percentual time gain thanks to automation 47%.

This study can concludes, that when comparing Robot Framework to Selenium IDE, than Robot has more useful features. Also, it can be concluded that in Telia Eesti AS Dealgate test automation environment has been effective. Although depending on the project and the testing period, results could differ from the current, but in the long run automation pays off. If I would have used a longer testing period than 2 years, the result would have been even more efficient.

The analysis of the given project could have been done even more accurately. There are several tests used, which are obsolete or which can be built to pass faster. One option

would be to write more effective database queries, enabling results to be returned faster. Another option would be to build up the tests so that their completion would take less time and thus be more reliable. A new analysis should be done after such changes to compare the new system to the old one and also identify the return of investment to determine how much more efficient the system has become.

Kasutatud kirjandus

[1] Tepandi, Jaak. (2005). Tarkvara kvaliteet ja standardid. [WWW] http://www.lap.ttu.ee/erki/failid/konspekt/tarkvara_kvaliteet_ja_standardid_idx5721/idx5721_konspekt.pdf (05.04.2016).

[2] Start Software Testing On Early Stages. How Does This Affect Cost? [WWW] http://qatestlab.com/knowledge-center/QA-Testing-Materials/start-software-testing-on-early-stages-how-does-this-affect-cost/ (05.04.2016).

[3] Why is software testing important to a business? [WWW] http://www.softwaretesting.com.au/Why_is_Software_Testing_important.php (05.04.2016).

[4] Testimise tüübid. [WWW] http://www.e-uni.ee/e-kursused/eucip/arendus/143_testimise_tbid.html (06.04.2016).

[5] Regressioonitestimine. [WWW] https://et.wikipedia.org/wiki/Regressioonitestimine (06.04.2015).

[6] Anderson, Stuart. Regression Testing. [WWW] http://www.inf.ed.ac.uk/teaching/courses/st/2011-12/Resource-folder/11_regression.pdf (12.04.2016).

[7] Rouse, Margaret. Regression testing. [WWW] http://searchsoftwarequality.techtarget.com/definition/regression-testing (09.04.2016).

[8] One Hour Regression Testing. [WWW] https://qatarun.wordpress.com/category/regression-testing/ (12.04.2016).

[9] Humans Create, Computers Can Do The Rest. [WWW] http://www.symbio.com/solutions/quality-assurance/test-automation/ (19.04.2016).

[10] Elfriede Dustin, Thom Garrett, Bernie Gauf. Implementing Automated Software Testing: How to Save Time and Lower Costs While Raising Quality, Addison-Wesley Professional. 2009.

[11] Robot Framework. [WWW] http://robotframework.org/ (21.04.2016).

[12] Platforms Supported by Selenium. [WWW] http://www.seleniumhq.org/about/platforms.jsp (22.04.2016).

[13] What is Selenium? [WWW] http://docs.seleniumhq.org/ (22.04.2016).

[14] How to Calculate ROI for Test Automation. [WWW] http://www.testing-whiz.com/blog/how-to-calculate-roi-for-test (25.04.2016).

[15] SeleniumHQ. [WWW] http://docs.seleniumhq.org/projects/ide/ (26.04.2016).

[16] Solntsev, Andrei. Selenide – Concise UI Tests in Java. [WWW] http://www.methodsandtools.com/tools/selenide.php (27.04.2016).